

PROCEEDINGS OF
THE 2008 CONFERENCE ON DESIGN &
ARCHITECTURES
FOR SIGNAL AND IMAGE PROCESSING

DASIP'08



Editor
Dragomir Milojevic, Adam Morawiec,

Associate Editors
Bertrand Granado, Tughrul Arslan, Marco Mattavelli

November 24-26
Bruxelles, Belgium

DASIP'08

CONFERENCE ON DESIGN & ARCHITECTURES FOR SIGNAL AND IMAGE PROCESSING

The development of complex applications, involving signal, image and control processing, is classically divided into three consecutive steps: a theoretical study of the algorithms, a study of the target architecture, and finally the implementation. Such a linear design flow is reaching its limits due to the intense pressure on design cycle and strict performance constraints. The approach, called Algorithm-Architecture-Matching, aims to leverage the design flow by a simultaneous study of both algorithmic and architectural issues, taking into account multiple design constraints, as well as algorithm and architecture optimizations.

Introducing new design methodologies is necessary when facing the new emerging applications as for example advanced mobile communication systems or smart sensors based systems. This forms a driving force for the future evolutions of embedded system designs methodologies.

The research community in Europe addressing these issues is very active both in academy and industry. The goals of this conference are to present the latest results in the domain of design and architecture for signal and image processing and to initiate a regular meeting of European researchers addressing this topic.

The DASIP conference will give the opportunity for researchers to exchange the ideas and to build the collaboration on emerging topics and technologies. It also aims to strength the links between the European Information Society Technologies (IST) priorities and the researchers in the domain of design and architecture for signal and image processing.

Dragomir Milojevic
Adam Morawiec
DASIP General Co-Chairs

Bruxelles
July 2008

DASIP'08 WORKSHOP ORGANIZATION

General Co-chairs

MILOJEVIC Dragomir, Université Libre de Bruxelles, Belgium
MORAWIEC Adam, ECSI, France

Vice General Chair

MATTAVELLI Marco, EPFL, Switzerland

Program Co-chairs

ARSLAN Tughrul, University of Edinburgh, Scotland
GRANADO Bertrand, ETIS-ENSEA, France

Program Committee

ABID Mohamed, Enis, Tunisia
AMIRA Abbes, Brunel University, England
ARIDHI Slaheddine, Texas Instruments, France
BECKER Juergen, Karlsruhe University, Germany
BEREKOVIC Mladen, Technical University of Braunschweig, Germany
BIELECKI Wlodzimierz, Politechnika Szczecinska, Poland
BOBDA Christophe, Kaiserslautern University, Germany
BOURIDANE Ahmed, Queen's University Belfast, Ireland
CRAWFORD David, Epson Scotland Design Centre, Scotland
DIGUET Jean-Philippe, Lab-STICC, France
DRUTAROVSKY Milos, T. U. of Kosice, Slovakia
ERDOGAN Ahmet T., University of Edinburgh, Scotland
FERRER Carles, Universitat Autònoma de Barcelona, Spain
GALAJDA Pavol, T. U. of Kosice, Slovakia
GARDA Patrick, University Pierre Et Marie Curie, France
GOGNIAT Guy, Lab-STICC, France
GOOSSENS Joël, Université Libre de Bruxelles, Belgium
HANNIG Frank, University of Erlangen-Nuremberg, Germany
HOWELLS W. Gareth J., University of Kent, England
INDRUSIAK Leandro Soares, T. U. of Darmstadt, Germany
KOCH Peter, CSDR, Denmark
LE MOULLEC Yannick, Aalborg University, Denmark
LEGAT Jean-Didier, Université catholique de Louvain, Belgium
MAMALET Franck, France Télécom, France
MANCINI Stéphane, GIPSA-lab/INPG-ENSIMAG, France
MANET Philippe, Université Catholique de Louvain, Belgium
MCDONALD-Maier Klaus D., University of Essex, England
MEDULLA Giuseppe, Atmel, Italy
PAINDAVOINE Michel, Le2i, France
PIGUET Christian, CSEM, Switzerland
PLAKS Toomas, London, England
PONSARD Christophe, CETIC, Belgium
ROBERT Frédéric, Université Libre de Bruxelles, Belgium
ROUX Sébastien, France Télécom, France

RUPP Markus, Vienna U. of Technology, Austria
SHAWKY Mohamed M., Heudiasyc, France
SOREL Yves, INRIA, France
TAPANI Ahonen, Tampere Univ. of Technology (TUT), Finland
VLADIMIROVA Tanya, University of Surrey, England
WEBER Serge, Lien, France

Organization Committee

ERDOGAN Ahmet T., University of Edinburgh, Scotland
GOGNIAT Guy, Lab-STICC, France
MANCINI Stéphane, GIPSA-lab/INPG-ENSIMAG, France

DASIP'08 CONTENT

INVITED TALK 1

Kristof Denolf, IMEC
AAA philosophy applied to Multimedia System on Chip design -----13

INVITED TALK 2

Johan Eker, Ericsson
Rethinking Embedded Software for Mobile Devices -----15

INVITED TALK 3

Tapani Ahonen, Tampere University of Technology
Middleware Centric Design for Massively Parallel Stream Processing Platforms-----17

OBJECT RECOGNITION AND TRACKING

Accuracy Amelioration of an Integrated Realtime 3D Image Sensor-----19-26
J. Ayoub, O. Romain, B. Granado, Y. Mhanna

Multiple Target Tracking System Design for Driver Assistance Application -----27-34
J. Khan, S. Niar, A. Menhaj, Y. Elhillali

Palmprint Verification Using DCT Coefficients and Linear Discriminant
Analysis -----35-41
M. Laadjel, A. Bouridane, F. Kurugollu, O. Nibouche, A. Meraoumia, M. Siagaa

Phase Only Based Shoeprint Recognition Techniques-----42-46
O. Nibouche, A. Boudane, D. Crookes, M. Gueham, M. Laadjel

SECURITY SYSTEMS AND CRYPTOGRAPHY

Hardware Implementation of Elliptic Processor Over Standard Binary Curves-----47-51
D. Nassar, M.W. El-Kharashi, A. El Halim Mahmoud Shousha

Design and Implementation of a High Quality and High throughput TRNG
in FPGA ----- 52-56
C. Klein, O. Cret, A. Suciu

ANALOG CIRCUITS AND MIXED-SIGNAL CIRCUITS

A Standard 3.5T CMOS Imager including a Light Adaptive System for Integration Time Optimisation-----57-60

E. Labonne, R. Rolland, G. Sicard

Self-Timed Implementation of an Impulse Radio Synchronisation Acquisition Algorithm -----61-68

J. Hamon, B. Miscopein, J. Schwoerer, L. Fesquet and M. Renaudin

PERFORMANCE ANALYSIS AND ESTIMATIONS

Optimization of automatically generated multi-core code for the LTE RACH-PD algorithm -----69-76

M. Pelcat, S. Aridhi, J.F. Nezan

Memory Management for Octree-like Structure Traversal -----77-84

T. Toczek and S. Mancini

RTOS FOR EMBEDDED SYSTEMS

OLLAF: a Fine Grained Dynamically Reconfigurable Architecture for OS Support -----85-91

S. Garcia, B. Granado

Reconfigurable Artificial Neural Network Model for Task Scheduling on Reconfigurable SoC -----92-99

D. Chillet, S. Pillement and O. Sentieys

RESOURCE MANAGEMENT TECHNIQUES FOR RTOS IN A CODESIGN FRAMEWORK

Partitioned scheduling of sporadic task systems: an ILP-based approach-----100-105

S. K. Baruah, E. Bini

A New Task Model and Utilization Bound for Uniform Multiprocessors -----106-111

S. Funk

SystemC multiprocessor RTOS model for services distribution on MPSoC platforms -----112-116

E. Huck, B. Miramond, F. Verdier, T. Lefebvre

EMBEDDED PLATFORMS FOR MULTIMEDIA AND TELECOM

A Novel Video Packet Loss Concealment Algorithm & Real Time Implementation	117-123
<i>A. Suissa, J. Mellor, F. Lohier, P. Garda</i>	
MOREA: A Memory-Oriented Reconfigurable Embedded Architecture	124-131
<i>E. Grace, D. Chillet, R. David, O. Sentieys</i>	
Energy reduction in wireless systems by dynamic adaptation of the fixed-point Specification	132-139
<i>H.-N. Nguyen, D. Menard, R. Rocher, O. Sentieys</i>	
Motion Aware Slicing for H.264 Selective Visual Encryption	140-146
<i>M. Leny, F. Prêteux, C. Le Barz</i>	

FORMAL MODELS AND TRANSFORMATIONS

Power Consumption Model at Functional Level for VLIW Digital Signal Processors	147-152
<i>M. E. A. Ibrahim, M. Rupp, S. E.-D. Habib</i>	
Data Cache-Energy and Throughput Models: A Design Exploration for Overhead Analysis	153-159
<i>M. Yasir Qadri, K. D. McDonald-Maier</i>	
Using Traceability for Reverse Instance Transformations with SiTra	160-167
<i>S. Shah, K. Anastasakis, B. Bordbar</i>	
Dataflow/Actor-Oriented language for the design of complex signal processing systems	168-175
<i>C. Lucarz, M. Mattavelli, M. Wipliez, G. Roquier, M. Raulet, J. W. Janneck, I.D. Miller and D. B. Parlour</i>	

ARCHITECTURAL AND LOGIC SYNTHESIS

Automatic Allocation of Redundant Operators in Arithmetic Data path Optimization	176-183
<i>S. Belloeil, R. Chotin-Avot, H. Mehrez, A. Munier-Kordon</i>	
High-Precision VLSI Architecture of Lifting-Based Forward and Inverse Wavelet Transforms	184-191
<i>A. Guntoro, M. Glesner</i>	
An Image Compression System for Earth Observation Satellites	192-199
<i>T. Vladimirova and G. Yu</i>	

A Design Space Exploration Flow for FPGA Implementation of Intensive Signal Processing Applications	200-207
<i>S. Le Beux, P. Marquet and J.-L. Dekeyser</i>	

SYSTEMS AND ARCHITECTURES FOR REAL-TIME IMAGE PROCESSING

Lossless Multi-mode Interband Image Compression and its Hardware Architecture	208-215
<i>X. Chen, C. N. Canagarajah, J. L. Nunez-Yanez</i>	
A Clustered Architecture for Scaling Video Processing Across Multiple Machines and Multiple Cores	216-221
<i>T. Harris, T. Arslan, B. Devlin, J. Diggins, I. Lindsay, K. May</i>	
Automatic Dynamic Structural-level Pipelining in Reconfigurable Processors	222-228
<i>M. Muir, N. Aslam, I. Nousias, A. Major, T. Arslan, I. Lindsay</i>	
Design and Implementation of an Efficient Shape Analysis Engine on FPGAs	229-236
<i>K. Benkrid</i>	
Multiresolution Analysis on Reconfigurable Hardware for Imaging	237-241
<i>A. Amira</i>	

POSTERS

A design and an implementation of a parallel based SIMD architecture for SoC on FPGA	242-248
<i>M. Baklouti, M. Abid, P. Marquet, J. L. Dekeyser</i>	
FPGA Design of ICA for real time Blind Signal Separation	249-254
<i>M. Ounas, S. Chitroub, R. Touhami, S. Gaoua, M. C. E. Yagoub</i>	
High speed binary image processor for compact real time vision systems.....	255-261
<i>A. Loos, M. Schmidt, D. Fey, J. Gröbel</i>	
Emerging Description Language Capabilities Matching Arithmetic Hardware Trends	262-269
<i>A. Zamfirescu</i>	
FPGA Implementation of Harmonic Currents Identification Algorithms using Neural Networks	270-276
<i>S. R. Dzonde, H. Berviller, J-P Blondé, F. Braun, C. H. Kom</i>	
A modelling of the throughput of various architectures and its confrontation in the case of the AES algorithm implementation	277-282

Y. Berviller, H. Rabah and S. Weber

OverSoC Graphical Design Environment -----283-283

M. Aichouch, E. Huck, B. Miramond

MDE benefits for Real Time Operating Systems modeling -----284-290

Y. Hadj Kacem, A. Mahfoudhi, M. Abid

Efficient Component-Based Face Recognition System for High Speed Devices -----291-298

N. Zaeri, J. Kittler, A. Cherri

Dataflow design of a co-processor architecture for image processing -----299-306

R. Thavot, R. Mosqueron, M. Alisafae, C. Lucarz, M. Mattavelli, J. Dubois

Insights to Variable Block Size Motion Estimation by Design Space Exploration ----307-313

J. Boutellier, P. Brisk and P. lenne

A Hardware Oriented Integer Pel Fast Motion Estimation Algorithm in
H.264/AVC -----314-320

O. Ndili , T. Ogunfunmi

INVITED TALK 1

Kristof Denolf, IMEC

AAA philosophy applied to Multimedia System on Chip design

Achieving a cost-efficient implementation of state-of-the-art multimedia requires designing to conform with the triple A philosophy: balancing the Application, Algorithm and Architecture. The main current hurdles are breaking the memory, power and instruction level parallelism wall. Properly introducing and exploiting parallelization is an important key of the solution to overcome these. This presentation will first give an overview of our current experience with triple A design on a variety of platforms (both on commercially available multiprocessor platforms as on an in-house designed MPSoC).

To meet the ever demanding requirements (cost, throughput, processing capabilities and often power) of future applications, like for instance hyper-spectral imaging, future developments will need to further exploit this Triple A philosophy in all its dimensions: heterogeneous design and 3D integration enabling new opportunities, variability/reliability that arise when moving to the latest technology node, etc.

INVITED TALK 2

Johan Eker, Ericsson
Rethinking Embedded Software for Mobile Devices

With the introduction of increasingly parallel hardware in the mobile space, established paradigms for embedded software are being challenged. Evolutionary changes of existing methodologies and tools are doomed to fail and the situation calls for new ideas and concepts. The problem of resource allocation, which is complicated for the uniprocessor case, becomes even more complex with multiple, possibly heterogeneous, cores. Traditionally, systems are carefully tuned by hand for a number of uses. The increasing complexity of mobile handset software, paired with the likewise more complex hardware, makes manual resource allocation infeasible. A control theoretically sound approach for dynamic resource management is proposed together with a logical view on resource utilization. A dataflow programming model, which allows for explicit parallelism is also discussed.

INVITED TALK 3

Tapani Ahonen, Tampere University of Technology
Middleware Centric Design for Massively Parallel Stream Processing Platforms

Design space for Digital Signal Processing (DSP) applications is limited by the degree to which the target architecture allows tradeoffs between: (a) correctness (fault tolerance, dependability) (b) responsiveness (task deadlines, realtime requirements) (c) energy consumption (power density, heat sinking, battery lifetime).

The key challenge in designing domain specific or general-purpose application development platforms is to provide trade-off flexibility with low overhead. Most importantly, the required application engineering effort should be kept moderate. Middleware and its application programming interface (API) are in a central role in hiding the management effort required by the added flexibility. The major roles of middleware are: to provide service access mechanisms hiding the details of the architecture from the application layer, and to autonomously tune the system according to its current state and the function(s) required. The emergence of distributed, massively parallel stream processing architectures calls for novel approaches. Besides efficient communication and resource allocation the platform architectures need to provide middleware friendly interfaces for monitoring and modifying their state.

This presentation highlights the techniques adopted in the EU cofunded projects: the Cutting edge Reconfigurable ICs for Stream Processing (CRISP) project and the “Silicon Cafe” platform that includes the Cofee CPU.

Accuracy Amelioration of an Integrated Real-time 3D Image Sensor

J. Ayoub¹, O. Romain¹, B. Granado², and Y. Mhanna³

Abstract— *In this paper we investigate an active vision technique implemented in an embedded system for 3D shapes reconstruction. The main objective of the work is to have a balance in the accuracy of all components in the system where the size and autonomy of such an embedded sensor are hard constraints. This is achieved through the improvement of the pre-processing algorithms by reducing the time needed to compute the spots centers. In addition, lens distortion of the camera is included in the model to increase accuracy when reconstructing objects. Experimental evaluation shows that the size and the time are reduced, precision increased, when the resources spent on processing are relatively acceptable in comparison to the benefits.*

Index Terms— 3D reconstruction, Active stereovision, Camera calibration, Lens distortion.

I. INTRODUCTION

RECONSTRUCTING 3D shapes is needed in several applications in computer vision and computer graphics, namely object recognition for robotic vision.

Numerous techniques have been developed to give solutions to the 3D reconstruction problem. The most common are those based on vision systems basing on either passive or active stereovision methods, where image sensors are used to provide the necessary information to retrieve the depth, since is not the case in traditional photography. The most commonly employed passive method consists of taking two images of a scene at two different shooting angles using either two cameras or only one camera for which an acquisition in two different positions is done. Then the 3-D coordinates of any point can be deduced from the 2-D coordinates by triangulation. Using this method, only characteristic points, with high gradient or high texture can be detected [1].

The active stereovision methods offer an alternative approach to the use of two cameras. They consist in replacing one of the two cameras by a projection system

which projects a set of structured rays. In this case, only one image is necessary. Many implementations of active stereovision methods have been realized. Some of them [2,9] provided significant results using traditional computer for application of such methods. In our research work, we have focused on an integrated 3D active vision sensor: "Cyclope" [1]. This sensor allows making real time 3D reconstruction while respecting the size and power consumption constraints of embedded systems [12] to be used in special applications like wireless capsule endoscopes, robotic heart surgery, or even asteroids exploration [11].

To realize this sensor, many techniques should be involved, starting with image capturing, spatial filtering, morphological operations, conversion gray-scale image to binary, segmentation, region labeling, correction of lens distortion, computation of spot centers, matching centers to epipolar lines, and finally 3D shape reconstruction. All these techniques will be realized in real-time. Beside, other necessary techniques should be executed off-line, like calibration of camera lens distortion, determination of epipolar lines and depth model.

Among the steps listed above, we will focus in this paper on two influencing parts to improve the accuracy of our sensor: the computation method of the spots centers, and the correction of lens distortion that highly affects the resulting measures.

The second section describes briefly Cyclope. Section 3 deals with the principles of the active stereovision system and 3D reconstruction method, explaining problem statement. In section 4 we discuss lens distortion and its influence on hardware implementation. In section 5, we present methods used to extract the centers of laser spots taking into account time and size demands. In section 6 we summarize the experimental results. And finally we conclude in section 7.

II. CYCLOPE

Cyclope is our integrated wireless 3D vision system based on active stereovision technique, it uses many different algorithms to increase accuracy and reduce processing time and sensor size. Such properties let this sensor more compatible for emergent and demanding applications where size and autonomy are hard constraints. The block diagram of "Cyclope" (figure 1) is composed of three essential parts [1]:

¹ J. Ayoub, O. Romain, Université Pierre et Marie Curie (UPMC), groupe SYEL, 4 place Jussieu, Paris, France (jad.ayoub@etu.upmc.fr; romain@upmc.fr)

² B. Granado, ETIS, CNRS, ENSEA, Université Cergy Pontoise 95014 CERGY cedex, France, (bertrand.granado@ensea.fr)

³ Y. Mhanna, Department of Physics and Electronics, Faculty of Sciences I, Lebanese University, Beirut, Lebanon (yamoha@ul.edu.lb)

- Instrumentation block: containing CMOS camera and a structure light projection system.
- Processing block: integrates a microprocessor core and a reconfigurable array. The microprocessor is used for sequential processing and the reconfigurable array is used to implement time consuming algorithms.
- Wireless communication block: This part is dedicated to the OTA (Over the Air) communication to have a wireless sensor.

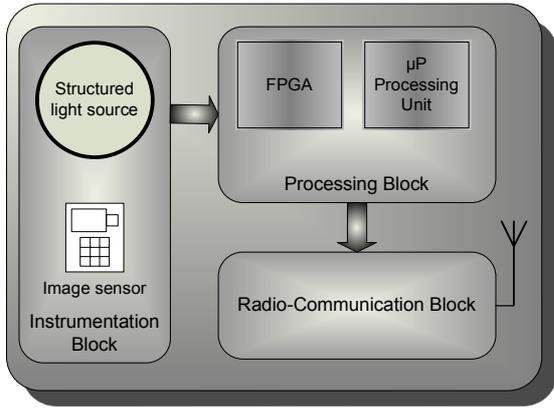


Fig. 1. Block diagram of CYCLOPE

To test and validate our system, a large scale demonstrator have been realized using an original CMOS imager, a generator of structured light constituted by an array of 361 (19x19 laser each being separated from its neighbors by a fix angle equal to 0.77°), XUP Virtex-II Pro Development System Board, and a Zigbee module.

III. ACTIVE STEREOVISION SYSTEM AND 3D RECONSTRUCTION

A. Principle and mathematical model

Active stereovision system consists of a single camera and a generator of laser-structured light that replaces the second camera of the passive stereovision system [1].

The laser projector is combined with a diffraction network in order to illuminate the studied scene with an array of laser beams. Each ray is separated from its neighbors by a fix and equal angle. The setup of active stereovision system is

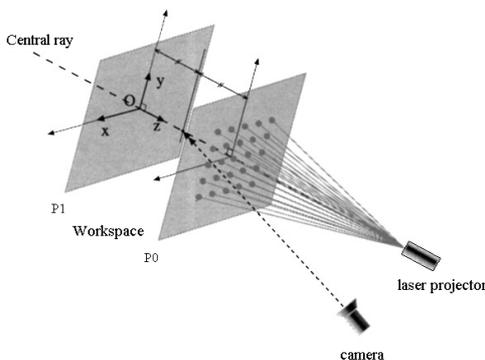


Fig. 2. Active stereovision system

represented in figure 2.

The calibration is processed according to a workspace delimited by two planes P_0 and P_1 perpendicular to the central ray of the beam, within which the object to be analyzed is placed [3].

The 3D reconstruction is achieved through triangulation between laser and camera. Each point of the projected pattern on the scene represents the intersection of two lines:

- The line of sight, passing through the pattern point on the scene and its projection in the image plane.
- The laser ray, starting from the projection center and passing through the chosen pattern point.

The calibration process provides two sets of parameters for each of the 361 points of the mesh [3]:

- A set of segments (eq.1), each representing the projection, in the image plane, of the part of a ray which crosses the workspace. This projection is the epipolar line in a passive stereovision system with two cameras where the laser ray is identified to the second camera line of sight (see Fig. 4(b)).
- The relationship between the center position of a given point on the corresponding segments of the ray and depth of the physical point corresponding to the point. This relation is modeled through an hyperbolic curve (eq.2).

$$v = a.u + b \quad \text{with } (a, b) \in \mathbb{R}^2 \quad (\text{eq.1})$$

$$z = \frac{1}{\alpha u + \beta} \quad \text{with } (\alpha, \beta) \in \mathbb{R}^2 \quad (\text{eq.2})$$

(u, v) are the image coordinate of a laser point.

B. Calibration

The values of parameters of equations (1) and (2) are obtained off-line using a least-square fit on the experimental data obtained through a calibration process. Note that the data are calculated considering lens distortion correction of the camera. Figure 4(a) represents the sequence of 12 planes used for calibration.

The calibration process yields a set of segments that represents the projections of the laser rays on the image plane (see Figure 4(b)). Our goal is to determine, for each spot appearing on the image, the segment on which it lies and, consequently, to associate the spot with the ray from which it stems [3].

C. Workflow of 3D reconstruction process

As shown in Figure 3, the scene illuminated by structured light source is captured by a CMOS camera and then stored into memory. The captured image submits a pre-processing phase to extract the useful data concerning the light spots.

The first stage of the pre-processing phase is to apply a median filter to blur the boundaries between bright spots and dark background and to remove outlier pixel intensities. The next stage is to apply a thresholding operation in order to

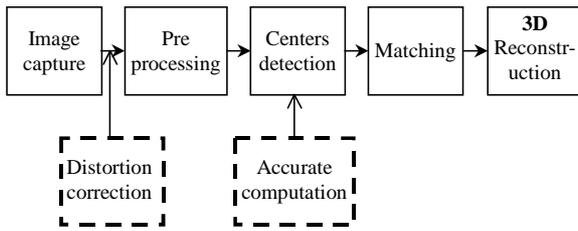


Fig. 3. A flowchart showing different phases of 3D image reconstruction

classify the bright areas and the dark area. The threshold process often produces an image that is less than perfect and is not sufficient to separate objects from their background, common problems are noise produced by incoherent lighting. It is often desirable to process a binary image before analysis to remove these abnormalities. This accomplished by applying morphological operations (erosion) on the binary picture to remove pepper noise remaining from the median filtration process that was applied in the first stage. Then a segmentation process is applied to extract and label the foreground subjects from the scene, to compute later their centers positions.

After that, the spots centers have to be matched to corresponding epipolar lines that were obtained off-line

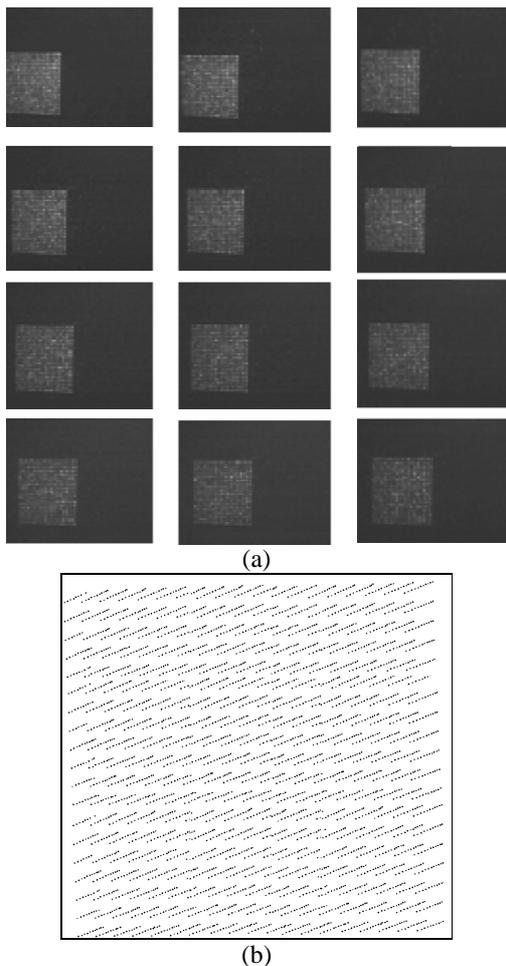


Fig. 4. (a) Calibration process: workspace delimited between 42cm and 64cm. (b) Epipolar lines

through a calibration process [2].

At the end, the distance between each spot and the stereoscopic system is computed from the depth model that is also obtained off-line through calibration process [2].

This workflow is represented in Figure 3 by blocks with solid lines. Since, the blocks with dashed lines represent the improvement implemented in our work to ameliorate the accuracy of such a system.

D. Problem statement

As equations (1) and (2) show, the 3D reconstruction of an object depends on the centers coordinates (u,v) of light spots in the image plane. Thus, any inaccurate representation of these points will highly affect the accuracy of our results in the 3D estimation stage. Indeed, many problems were encountered when performing this procedure, in this paper we will focus on two important aspects:

- Lens distortion prevents accurate perception of range [5], just because the true coordinates of laser spots are deviated due to lens distortion. That makes measurement and distance judgment difficult. So, distortion correction process will be necessary to accurately reconstruct the 3D coordinates of studied object. In our work, we will focus on correcting distortion taking into account the size, time, and power consumption constraints of embedded system.
- How to compute the coordinate of spot centers without consuming a considerable amount of resources, and considering high accuracy of our embedded system?

IV. DISTORTION CORRECTION

A. Pinhole camera model:

Physical camera parameters are commonly divided into extrinsic and intrinsic parameters [14]. Extrinsic parameters define the location and orientation of the camera reference frame with respect to the world coordinate system. Whereas the intrinsic parameters are used to link the pixel coordinates of an object with corresponding world coordinates in the

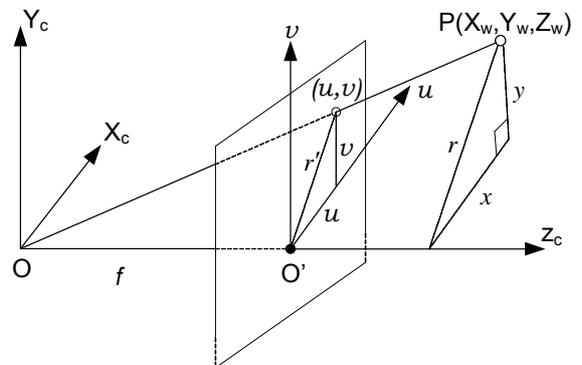


Fig. 5. Pinhole camera model (X_w, Y_w, Z_w) : World coordinates, (O, X_c, Y_c, Z_c) : camera coordinates, (O', u, v) : image plane coordinates.

camera reference frame, they characterize the optical, geometric, and digital characteristics of the camera.

As a first approximation, we will consider our model as a pinhole camera (see fig. 5) that neglects all optical distortion.

The relationship between world coordinate (X_w, Y_w, Z_w) and camera coordinates (X_c, Y_c, Z_c) defines the image extrinsic parameters [13], and is expressed in expression (3):

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = R \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + T \quad (3)$$

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \text{ and } T = [t_1 \quad t_2 \quad t_3]^t$$

Where R is a 3x3 rotation matrix defining camera orientation, and T is a 3x1 translation vector representing the distance from the origin of the camera coordinate system to the origin of the world coordinate system.

Since, the intrinsic parameters are obtained by using simple triangulation for the pinhole model; the projection of the point P to the image plane is expressed in (4), where f is the focal length:

$$\begin{bmatrix} \tilde{u}_i \\ \tilde{v}_i \end{bmatrix} = \frac{f}{Z_c} \begin{bmatrix} X_c \\ Y_c \end{bmatrix} \quad (4)$$

Where $(\tilde{u}_i, \tilde{v}_i)$ represents true coordinates (undistorted).

the relationship between the image pixel coordinates and the image coordinates is given by the expressions:

$$\begin{bmatrix} u_i' \\ v_i' \end{bmatrix} = \begin{bmatrix} D_u s_u \tilde{u}_i \\ D_v \tilde{v}_i \end{bmatrix} + \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} \quad (5)$$

Where (u_0, v_0) are the coordinates of the principal point O', (D_u, D_v) are conversion factors that change metric units to pixels, and s_u is the scale factor [13].

B. Lens Distortion

Pinhole camera model is based on the principle of collinearity where each point in the object space is projected by a straight line through the projection center into the image plane. This model can be used only as an approximation of the real camera that is actually not perfect and sustains a variety of aberration [13]. So, pinhole model is not valid when high accuracy is required like in our expected applications (Endoscopes, robotic surgery..). In this case, a more comprehensive camera model must be used, taking into account the corrections for the systematically distorted image coordinates.

As a result of several types of imperfections in the design and assembly of lenses composing the camera optical system, the real projection of the point P in the image plane expressed above in expressions (5), will be replaced by expressions that take into account the error between the real image observed coordinates and the corresponding ideal (non observable) image coordinates.

$$\begin{cases} u' = u + \delta_u(u, v) \\ v' = v + \delta_v(u, v) \end{cases} \quad (6)$$

Where (u, v) are the ideal non-observable, distortion-free image coordinates, and (u', v') are the corresponding real coordinates, δ_u and δ_v are respectively the distortion along the u and v axes.

Usually, the lens distortion consists of radial symmetric distortion, decentering distortion, affinity distortion, and non-orthogonality deformations.

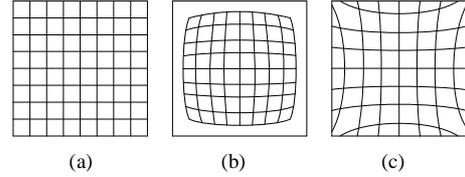


Fig. 6. : (a) The ideal undistorted grid. (b) Barrel distortion. (c) Pincushion distortion

i) Radial distortion: is caused by flawed radial curvature of a lens and causes the actual image point to be displaced radially in the image plane. A negative radial displacement of the image points is referred to as barrel distortion, and due to the fact that many wide angle lenses have higher magnification in the image center than at periphery. This causes the image edges to shrink around the center and form a shape of a barrel. The pincushion distortion is the inverse effect when the edges are magnified stronger.

As illustrated in Figure 6, this type of distortion is strictly symmetric about the optical axis, and can be approximated [15] using the expression (7) in terms of the Cartesian coordinates (u, v) :

$$\begin{aligned} \delta_{ur} &= k_1 u(u^2 + v^2) + O[(u, v)^5] \\ \delta_{vr} &= k_1 v(u^2 + v^2) + O[(u, v)^5] \end{aligned} \quad (7)$$

k_1 is the coefficient of radial distortion.

ii) Decentering distortion: This type of distortion caused from the fact that the optical centers of lens surfaces are not strictly collinear. This distortion has both radial and tangential components [16], which can be written as resulting along the u and v axes in the following form [14]:

$$\begin{aligned} \delta_{ud} &= p_1(3u^2 + v^2) + 2p_2 uv + O[(u, v)^4] \\ \delta_{vd} &= 2p_1 uv + p_2(u^2 + 3v^2) + O[(u, v)^4] \end{aligned} \quad (8)$$

p_1 and p_2 are coefficients for decentering distortion.

iii) Thin prism distortion: It arises from imperfection in lens design and manufacturing, as well as camera assembly. This type of distortion can be modeled by the adjunction of a thin prism to the optical system, causing additional amounts of radial and tangential distortions [17]. Such distortions can be expressed [14] with distortion coefficients s_1 and s_2 along the u and v axes as:

$$\begin{aligned}\delta_{up} &= s_1(u^2+v^2)+O[(u,v)^4] \\ \delta_{vp} &= s_2(u^2+v^2)+O[(u,v)^4]\end{aligned}\quad (9)$$

Other distortion types have also been proposed in the literature [13], in most cases the error is small and the distortion component is insignificant. However, it is impossible and unnecessary to consider into account all types of distortion. Only the major three types listed above need to be considered in practice.

iv) Total distortion:

The effective distortion can be modeled by addition of the corresponding expressions. Combining (7, 8, and 9), gives the total amount of distortion along the u and v axes:

$$\begin{aligned}\delta_u(u,v) &= \delta_{ur} + \delta_{ud} + \delta_{up} \\ \delta_v(u,v) &= \delta_{vr} + \delta_{vd} + \delta_{vp}\end{aligned}\quad (10)$$

Assuming that only the first and second order terms are enough to compensate for the distortion, and the terms of order higher than three are negligible, we obtain [18] a camera model to become fifth order polynomials (expression 11), where (u_i, v_i) are the distorted image coordinates in pixels, and $(\tilde{u}_i, \tilde{v}_i)$ are true coordinates (undistorted).

$$\begin{aligned}u_i &= D_u s_u (k_2 \tilde{u}_i^5 + 2k_2 \tilde{u}_i^3 \tilde{v}_i^2 + k_2 \tilde{u}_i \tilde{v}_i^4 + k_1 \tilde{u}_i^4 + k_1 \tilde{u}_i \tilde{v}_i^2 \\ &\quad + 3p_2 \tilde{u}_i^2 + 2p_1 \tilde{u}_i \tilde{v}_i + p_2 \tilde{v}_i^2 + \tilde{u}_i) + u_0 \\ v_i &= D_v (k_2 \tilde{u}_i^4 \tilde{v}_i + 2k_2 \tilde{u}_i^2 \tilde{v}_i^3 + k_2 \tilde{v}_i^5 + k_1 \tilde{u}_i^2 \tilde{v}_i + k_1 \tilde{v}_i^3 \\ &\quad + p_1 \tilde{u}_i^2 + 2p_2 \tilde{u}_i \tilde{v}_i + 3p_1 \tilde{v}_i^2 + \tilde{v}_i) + v_0\end{aligned}\quad (11)$$

The inverse model is deduced in expression (12):

$$\begin{aligned}\tilde{u}_i &= \frac{\tilde{u}_i^1 + \tilde{u}_i^1 (a_1 r_i^2 + a_2 r_i^4) + 2a_3 \tilde{u}_i^1 \tilde{v}_i^1 + a_4 (r_i^2 + 2\tilde{u}_i^1{}^2)}{(a_5 r_i^2 + a_6 \tilde{u}_i^1 + a_7 \tilde{v}_i^1 + a_8) r_i^2 + 1} \\ \tilde{v}_i &= \frac{\tilde{v}_i^1 + \tilde{v}_i^1 (a_1 r_i^2 + a_2 r_i^4) + 2a_4 \tilde{u}_i^1 \tilde{v}_i^1 + a_3 (r_i^2 + 2\tilde{v}_i^1{}^2)}{(a_5 r_i^2 + a_6 \tilde{u}_i^1 + a_7 \tilde{v}_i^1 + a_8) r_i^2 + 1}\end{aligned}\quad (12)$$

Where:

$$\tilde{u}_i^1 = (u_i - u_0) / (D_u s_u), \tilde{v}_i^1 = (v_i - v_0) / (D_v), \text{ and } r_i^2 = \sqrt{\tilde{u}_i^1{}^2 + \tilde{v}_i^1{}^2}$$

The unknown parameters a_1, \dots, a_8 are solved using direct least mean squares fitting [18] in the off-line calibration process.

C. Geometric camera calibration

The objective of the geometric camera calibration procedure is to determine the intrinsic and extrinsic parameters of the camera model [13]. There are many

proposed methods that can be used to estimate these parameters, and there are also methods that produce only a subset of the parameter estimates. We chose a traditional calibration method based on observing a planar checkerboard in front of our system at different poses and positions (see Figure 7) to solve the equations of unknown parameters (12). Some Matlab toolbox are available to perform this calibration procedure [20]. The results of the calibration procedure are presented in Table 1.

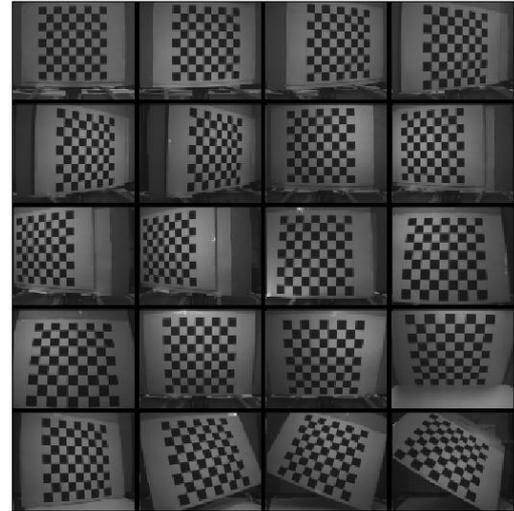


Fig. 7. Different checkerboard positions used for calibration procedure

TABLE. 1 CALIBRATION RESULTS

Parameter	Value	error	
u_0	[pixels]	178.04	1.28
v_0	[pixels]	144.25	1.34
$f \cdot D_u \cdot s_u$	[pixels]	444.99	1.21
$f \cdot D_v$	[pixels]	486.39	1.37
a_1		-0.3091	0.0098
a_2		-0.0033	0.0031
a_3		0.0004	0.0001
a_4		0.0014	0.0004
a_5		0.0021	0.0002
a_6		0.0002	0.0001
a_7		0.0024	0.0005
a_8		0.0011	0.0002

It is clear from the results that the two first parameters a_1 and a_2 are the dominant parameters, this fact is expected because they belong to the radial distortion, known in the literature [13, 14, 15] as the dominant distortion model. And for this reason some proposed correction methods simplify their models by restricting the lens distortion effect to radial distortion [5, 16].

In this section we have introduced the distortion correction model that we applied to both off-line calibration process needed to estimate the eppipolar lines, and to section dealing

with spots centers computation. To reduce the computation time, we have implemented the correction model only to the active light spots (361 points), where the remaining part of the picture belong to background. This improvement will increase the accuracy of our system, specially in the last stage of our work, where the essential feature of our sensor, ‘‘Cyclope’’, is to compute the 3D coordinates of the illuminated spots, characterized by their centers. The computation methods of these centers will be the object of the next section.

V. COMPUTATION OF SPOTS CENTERS

The Threshold and Labeling processes applied to the captured image allow us to determine the area of each spot (number of pixels). So, the centers coordinates of these points could be calculated as follow:

$$u_{gl} = \frac{\sum_{i \in I} u_i}{N_I} \quad (\text{eq. 13}) \quad \text{and} \quad v_{gl} = \frac{\sum_{i \in I} v_i}{N_I} \quad (\text{eq. 14})$$

u_{gl}, v_{gl} : abscissa and ordinate of I^{th} spot center.
 u_i and v_i : coordinates of pixels constructing the spot.
 N_I : Number of pixels of I^{th} spot (area in pixels).

The goal of our work in this section is to compute the spots centers taking into account precision demand of our implementation, where the sub-pixel precision is important because its influence to distance prediction stage (see equation 2). The hardest step in center computation part is the division operations A/B in equations (13) and (14). Several methods are established to solve this problem.

A. Implementation of a hardware divider

Hardware dividers are computationally expensive and consume a considerable amount of resources, and not completely acceptable for high accuracy embedded systems. Even the completely parameterized designs like Xilinx pipelined divider [8] are not less expensive in slices (see table 2). Since, some other techniques are used to compute the center of laser spots avoiding the use of hardware dividers. A comparison between different approaches is present in section VI.

TABLE 2 : 32 BIT, FIXED POINT, XILINX PIPELINED DIVIDER IP CORE SPECIFICATIONS

Property	Divisor and dividend width	
	8	32
Number of slices	129	1666
F_{max} , MHz	385	203

B. Approximation method

Some studies suggest approximation methods to avoid implementation of hardware dividers. Such methods like that implemented in [7] replace the active pixels by the smallest rectangle containing this region, and then replace the usual division by simple shifting (division by 2).

$$u_{gl}^* = \frac{\text{Max}(u_i) + \text{min}(u_i)}{2} \quad (\text{eq. 15})$$

$$v_{gl}^* = \frac{\text{Max}(v_i) + \text{min}(v_i)}{2} \quad (\text{eq. 16})$$

This approach is approximated in equations (15) and (16), where (u_i, v_i) are the active pixel coordinates, (u_{gl}^*, v_{gl}^*) are the approximated coordinates of the spot center.

The determination of rectangle limits needs two times scanning of the image, detecting in every scanning step, respectively, the minimum and maximum of pixels coordinates.

For each spot, we should compare the coordinates of every pixel by last registered minimum and maximum to assign new values to $u_m, u_M, v_m,$ and v_M . (m: Minimum; M: maximum). While N_p is the average area of spots (number of pixels), we can estimate the number of operations needed to calculate the center of each spot by $4N_p + 6$. And in global, $N_{op} \approx 25 \times N \times (4N_p + 6)$ operations are needed to calculate the centers of N spots (video-cadence 25 fps).

Such approximation is simple and easy to use but still needs considerable time to be calculated. Beside, the error is not negligible.

The error in such method is nearly 0.22 pixel, and the maximum error is more than 0.5 pixel [7]. Taking the spot of Figure 8 as an example of inaccuracy of such a method, the real center position of these pixels is (4.47 ; 6.51). But when applying this approximation method, the center position will be (5;6). This inaccuracy will result mismatching problem that affects the measurement result when reconstructing object.

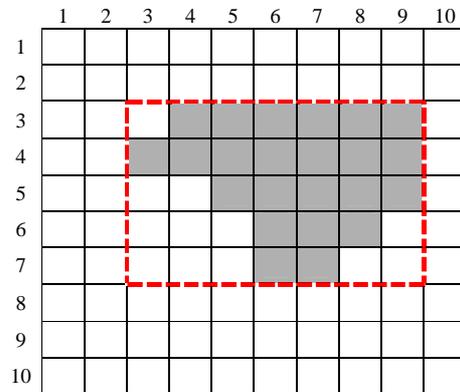


Fig. 8. Smallest rectangle containing active pixels

C. Our method

The area of each spot (number of pixels) is always a positive integer, while its value is limited in a pre-determinate interval $[N_{\text{min}}, N_{\text{max}}]$. Where N_{min} and N_{max} are respectively the minimum and maximum areas of laser spots in the image.

The spot areas depend on object illumination, distance between object and camera, and the angle of view of the scene.

Our method consists on realizing a FIR filter to replace the division by multiplication, to calculate the centers of each spot.

$$u_{gI} = a_1.u_1 + a_2.u_2 + \dots + a_{N_I}.u_{N_I}$$

$$v_{gI} = a_1.v_1 + a_2.v_2 + \dots + a_{N_I}.v_{N_I}$$

In our case, the filter coefficients a_i are constants and equals. For the spot I, with area equal N_I , the filter coefficients are: $a_1 = a_2 = \dots = a_{N_I} = 1/N_I$.

In other words, it is sufficient to perform a simple convolution between inputs that are the pixels coordinates of each spot, and a short pre-determinate sequence formed of constant filter coefficients registered in a Look Up Table (see Table 3) with inverse proportionality relationship with the area of each spot, the contents of LUT are indexed by spot size.

TABLE 3: FILTER COEFFICIENTS INDEXED BY SPOT SIZE

1	2	3	4	...	N_{max-1}	N_{max}
1	0.5	0.25	0.125	...	$1/(N_{max-1})$	$1/N_{max}$

The implementation of such a filter is very easy, regarding that the most of DSP functions are provided for earlier FPGAs. For example Virtex-II architecture [10] provides an 18x18 bits Multiplier with a latency of about 4.87ns at 205MHz, and optimized for high-speed operations. Additionally, the power consumption is lower compared to a slice implementation of an 18-bit by 18-bit multiplier [10].

For N luminous spots source, the Number of operations needed to compute the centers coordinates is $N_{op} \approx 25 \times N \times N_p$, N_p is the average area of spots. When implementing our approach to Virtex II Pro FPGA (XC2VP30), it was clear that we gain in execution time and size. Comparison of different implementation approaches is described in the next section.

VI. EXPERIMENTAL RESULTS

The implementation results of distortion correction method to Xilinx Virtex II Pro FPGA (xc2vp30) are summarized in Table 4. Figure 7(a,b) presents image example before and after correction of lens distortion. Regarding size and Latency, it is clear that their results are suitable for our application, and the cost-benefits assessment is acceptable regarding high precision demands of our applications.

Comparing our method used to compute the spots centers with two other methods (see table 5), it is clear that our approach has higher accuracy, smaller size than approximation method. Since it has nearly the same accuracy of method using hardware divider, but still have a considerably small size and uses less resources. Regarding latency, the results of all three approaches respect real time constraint of video cadence (25 fps).

Comparing many measures on the depth estimation before and after the implementation of our improvements, the results indicate that the precision of the system increased, so that the residual error is reduced about 33% (see Figure 7(c)).

TABLE 4: DISTORTION CORRECTION PERFORMANCE CHARACTERISTICS

<i>Slices</i>	1795 (13 %)
<i>Latency</i>	11.43 μ s
<i>Error</i>	< 0.01 pixels

TABLE 5: CENTERS COMPUTATION PERFORMANCE CHARACTERISTICS

Method	Slices	Latency (μ s)	Error (pixels)
<i>Approximation</i>	287	4.7	0.21
<i>Hardware divider</i>	1804	1.77	0.0078
<i>Our approach</i>	272	2.34	0.015

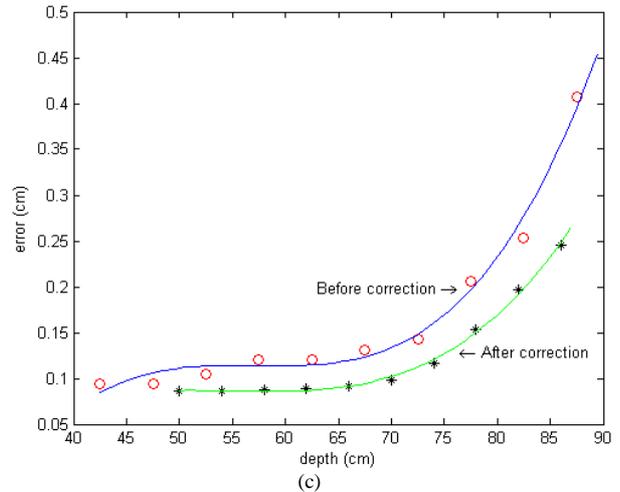
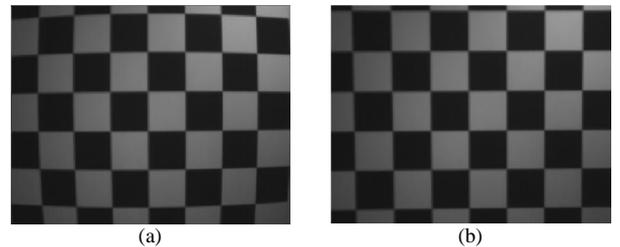


Fig. 7. a)Checkerboard image before distortion correction b) Checkerboard image after correction. c) error comparison before and after applying distortion correction and centers re-computing

VII. CONCLUSION AND FURTHER WORK

In this paper, we have described an easy and simple method to compute spots centers in an active stereovision system, to guarantee accurate results respecting the constraints of an embedded system. Whereas the accuracy increased, because the error is approximately negligible. On the other hand, the correction of lens distortion will increase the response time because the model becomes fifth order polynomials. But this cost appears acceptable regarding the benefits in precision needed to reconstruct the 3D shape of

the object, where experimental results show that we reduced the residual error.

Since, we have a reliable model which allows us to determine, with high accuracy, the 3D coordinates of the studied object from its 2D image. Our future researches will focus on recognition of this object basing on features extraction and statistical analysis of the shape, with implementation of such theories on FPGA.

REFERENCES

- [1]. T. Graba, B. Granado, O. Romain, T. Ea, A. Pinna, and P. Garda. "Cyclope: an integrated real-time 3d image sensor" In XIX Conference on design of circuits and integrated systems, 2004.
- [2]. F. S. Marzani, Y. Voisin, L. Y. Voon and A. Diou, "Calibration of a three-dimensional reconstruction system using a structured light source" Optical Engineering, Vol. 41 No. 2, February 2002.
- [3]. S. Woo, A. Dipanda, F. Marzani, and Y. Voisin, "Determination of an optimal configuration for a direct correspondence in an active stereovision system" Proc. On 2nd IASTED INT. Conf. VIIP, Malaga, Espagne, September 2002, p. 596-601.
- [4]. T. Pavlidis, "Structural Pattern Recognition", Springer-Verlag, Berlin (1977).
- [5]. J. P. Helferty, C. Zhang, G. McLennan, and W. E. Higgins, "Videoendoscopic Distortion Correction and Its Application to Virtual Guidance of Endoscopy", IEEE Transaction on medical imaging, vol. 20, NO. 7, July 2001.
- [6]. Stuart F. Oberman, and Michael J. Flynn, "Division Algorithms and Implementations". IEEE Transactions on computers, Vol. 46, No. 8, August 1997.
- [7]. T. Graba, "Etude d'une architecture de traitement pour un capteur intégré de vision 3D", Thèse de doctorat, Université Pierre et Marie Curie, 2006.
- [8]. "Xilinx pipelined divider V1.0", DS530 January 18, 2006.
- [9]. W. Li, F. Boochs, F. Marzani, and Y. Voisin, "Iterative 3D surface reconstruction with adaptive pattern projection", in Proc. of the Sixth IASTED International Conference on Visualization, Imaging and Image Processing (VIIP), Palma De Mallorca, Espagne, pp.336-341, August 2006.
- [10]. M. Adhiwiyogo, "Optimal Pipelining of I/O Ports of the Virtex-II Multiplier", XAPP636 (v1.4) June 24, 2004.
- [11]. J. Stooke, "Cartography of asteroids and comet nuclei from low resolution data" Asteroids, Comets, Meteors, pp. 583-586, 1991.
- [12]. A. Kolar, T. Graba, A. Pinna, O. Romain, and B. Granado, "Smart Bi-Spectral Image Sensor for 3D Vision", IEEE sensors 2007, pp.295.
- [13]. J. Heikkilä and O. Silven, "A four step camera calibration procedure with implicit image correction", Proc. IEEE Conference on Computer Vision and Pattern Recognition, June 17-19, San Juan, Puerto Rico, 1:1106-1112.
- [14]. J. Weng, P. Cohen, and M. Herniou, "Camera calibration with distortion models and accuracy evaluation", IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 14, Issue 10 (October 1992).
- [15]. J. Wang, F. Shi, J. Zhang and Y. Liu, "A new calibration model of camera lens distortion", Pattern recognition (2007).
- [16]. C. Slama, "Manual of photogrammetry", American society of photogrammetry, Falls church VA, USA, 4th edition 1980.
- [17]. W. Faig, "Calibration of close-range photogrammetric systems: Mathematical formulation", Photogrammetric Engineering Remote Sensing, vol. 41, no. 12, pp. 1479-1486, Dec 1975.
- [18]. Heikkilä, J. (1997) "Accurate camera calibration and feature based 3-D reconstruction from monocular image sequences" Ph.D. dissertation, University of Oulu, Finland.
- [19]. "Optimal Pipelining of I/O Ports of the Virtex-II Multiplier" XAPP636(v 1.4), June 24, 2004.
- [20]. Jean-Yves Bouguet, "Camera Calibration Toolbox for Matlab", http://www.vision.caltech.edu/bouguetj/calib_doc/

Multiple Target Tracking System Design for Driver Assistance Application

Jehangir Khan Smail Niar Atika Menhaj Yassin Elhillali

Université de Valenciennes et du Hainaut Cambrésis, France

[/jehangir.khan, smail.niar, atika.menhaj, yassin.elhillali}@univ-valenciennes.fr](mailto:{jehangir.khan, smail.niar, atika.menhaj, yassin.elhillali}@univ-valenciennes.fr)

Abstract --- This paper discusses the design of an entire Multiple Target Tracking (MTT) system. Use of MTT in driver assistant systems makes them very efficient and effective in collision avoidance and early warning. We describe the procedure that we chose for implementing an MTT system on a reconfigurable platform. We also examine in detail each of the task composing the MTT chain. In our implementation, several independent parallel tasks have been identified and mapped onto a multiprocessor architecture to achieve the deadlines imposed by the application. An Automotive-radar is used as the front end sensor in our application. We also take into account the constraints imposed by our embedded system. This study demonstrates that the joint utility of reconfigurable circuits (namely FPGA) and MPSoC, facilitates the development of a flexible and efficient MTT system.

I. INTRODUCTION

Unfavorable weather, low visibility conditions, misjudgment of delicate situations and physical or mental stress are among the reasons that put the lives of the driver and his/her passengers in danger. Relieving the driver of the stressful driving conditions guarantees a drop in road accidents. Driver Assistant Systems (DAS) help drivers take correct and quick decisions in delicate situations. These systems provide the driver a realistic assessment of the dynamic behavior of potential obstacles well before it is too late to react.

Use of Multiple-Target Tracking (MTT) enhances the affectivity of driver assistance systems to aid drivers in taking correct decisions in critical situations. The purpose of target tracking is to collect data from the sensor field of view (FOV) containing one or more potential targets of interest and to partition the sensor data into sets of observations, or tracks [1]. In context of driver assistance, a target tracking system detects and monitors the dynamic behavior of one or more obstacles in the way of the host vehicle.

Our implementation of the MTT system has two main features among others. First, being radar-based, it has the advantages of longer range as compared to camera based systems. It performs better in bad visibility conditions and has lower computational requirements [3]. Radar based multi-target tracking is achievable with relatively more ease and less complexity as compared to camera based tracking.

Moreover, radar helps detect obstacles at longer distances and hence ensures longer reaction time for vehicle drivers.

Secondly, we implement our system on FPGA using MPSoC architecture which is inherently flexible and adaptable. This feature capitalizes on advances in FPGA fabrication technology allowing a cost effective implementation of complex embedded applications. Charting of processor properties over the last three decades shows that the performance of a single processor has leveled off in the last decade[13]. Dedicated hardware implementation may be useful for high speed processing but it does not offer the flexibility and programmability desired for system evolution. Applications with tight resource-consumption and runtime constraints are increasingly resorting to MPSoC architectures. The move to MPSoC design elegantly addresses the power issues faced on the hardware side. Creating multiple processors that execute at lower frequency, results in comparable overall performance in terms of instructions per second while allowing designers to slow down the clock speed, which is a major constraint for low power designs [13].

Many studies have been done on the isolated parts of MTT system [2, 3 7,8] but implementation of the complete MTT system on a reconfigurable platform and its application to automotive safety is rather rare.

II. THE APPLICATION

A. Terminology

In context of target tracking applications, a *target* represents an obstacle in the way of the host vehicle. With every obstacle is associated a *state* which is represented as a vector that contains parameters defining target's position and its dynamics in space e.g. its distance, speed, azimuth or elevation etc. A state vector with n elements is called n -state vector. A concatenation of target states defining the target trajectory or movement history at discrete moments in time is called a *track*. As detailed below in MTT, tracking deals with 3 quantities: the *Observation*, which corresponds to the measurement of a target's state by a sensor (radar) at discrete moments in time. It is one of the two representations of the true state of the target. The other representation is a calculated "guess" or *prediction* of the

target's true state before the observation arrives. Taking into account the *observation* and the *prediction*, an *estimate* about the true target state is made. Estimate is the corrected state of the target that depends upon the variances of both the observation and the prediction. In this paper, the term *scan* is used to name the periodic sweep of radar FOV giving observations of all the detected targets.

B. MTT building blocks

A simplified view of Multiple Target Tracking (MTT) system is given in figure 1. The system can broadly be divided into two main functions namely *Data Association* and *Filtering & Prediction*. The two functions work in a close loop. The data association function is further divided into three sub-functions; "*Track maintenance*", "*Observation-to-Track Assignment*" and "*Gate Computation*". Detailed description of these functions and sub-functions is given in the next section.

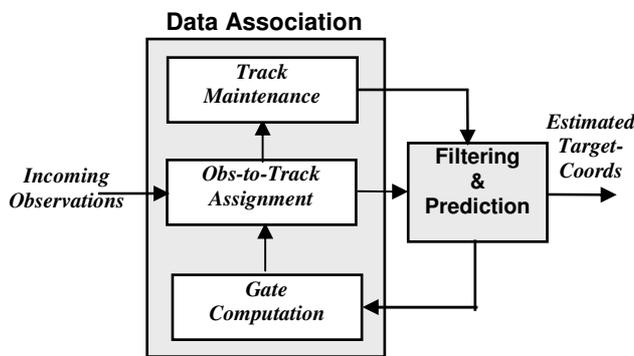


Figure 1: A simplified view of MTT

III. APPLICATION DEVELOPMENT

For the purpose of parallelized implementation we organized the application into sub-modules as shown in figure 2. The functioning of the system is explained as follows.

Assuming recursive processing as shown by the loop in figure 1, tracks would have been formed on the previous radar *scan*. When new observations are received from the sensor the processing loop is to be executed. Incoming observations are first considered by the "*Gate checker*" for updating of the existing tracks. Gating tests determine which possible "*observation-to-track*" pairings are reasonable, by attributing a cost to each pairing. The costs are calculated as the statistical distance between the *predictions* of the target states given by the filters and the *observed state* coordinates received from the radar. These

costs are put together in a *cost matrix* which is then passed on to the *assignment solver* to determine the finalized pairings. The pairings are made in a way to ensure minimum total cost for all the pairings. The finalized observation-track pairings are passed on to the tracking filters which use them for estimating the current states of targets and predicting the next states as well as the *error covariance* associated with these predictions.

The predicted states and predicted error covariance are used by the "*Gate compute*" function to define probability *gates* or windows around the predicted states. The dimensions of the gates being dictated by the prediction error covariance, these gates demarcate the probability boundaries for the next state coordinate measurements. The "*Gate Compute*" sub-function can be viewed as a first level of "*screening out*" the unlikely target-track associations in case of multiple observations falling close to a single prediction or vice versa. In the second level of "*screening*", namely observation-to-track assignment, a strictly one-to-one coupling is established between observations and tracks.

The "*Track Maintenance*" sub-function consists of three blocks. The "*obs-less Gate Identifier*" identifies the gate where no observation falls. This indicates a probable disappearance of an already known target and hence the deletion of its track after confirmation. The "*New Target Identifier*" detects observations that fall outside all the gates. These observations are potential candidates for initiating new tracks after confirmation. The "*Track Init/Del*" block initiates new tracks or deletes existing ones when needed. In context of this work, 3 observations out of 5 scans for the same target initiate a new track while 3 consecutive misses out of 5 scans for an existing target prompts the deletion of its track. The "*Tracking filters*" block in figure 2, is particularly important. We use Kalman filters for this block. The number of filters employed is equal to the maximum number of targets to be tracked. In our current work we have fixed this number at 10. In the final system we will increase it up to 20 as the radar we are using can measure the coordinates of a maximum of 20 targets. Hence this block will use 20 similar filters in final system.

At start up, at most 10 of the "*incoming observations*" would simply pass through the "*Gate Checker*", "*Cost Matrix Generator*" and "*Assignment Solver*" on to the filters' inputs. The filter takes an observation as an "*inaccurate*" representation of the "*true state*" of the target and the amount of inaccuracy of the observation depends on the measurement variance of the sensor. The filter then estimates the current state of the target and predicts its next state before the next observation is available. To estimate the true state we need a *process model*, a *measurement model* and an *estimator*. These are all detailed below.

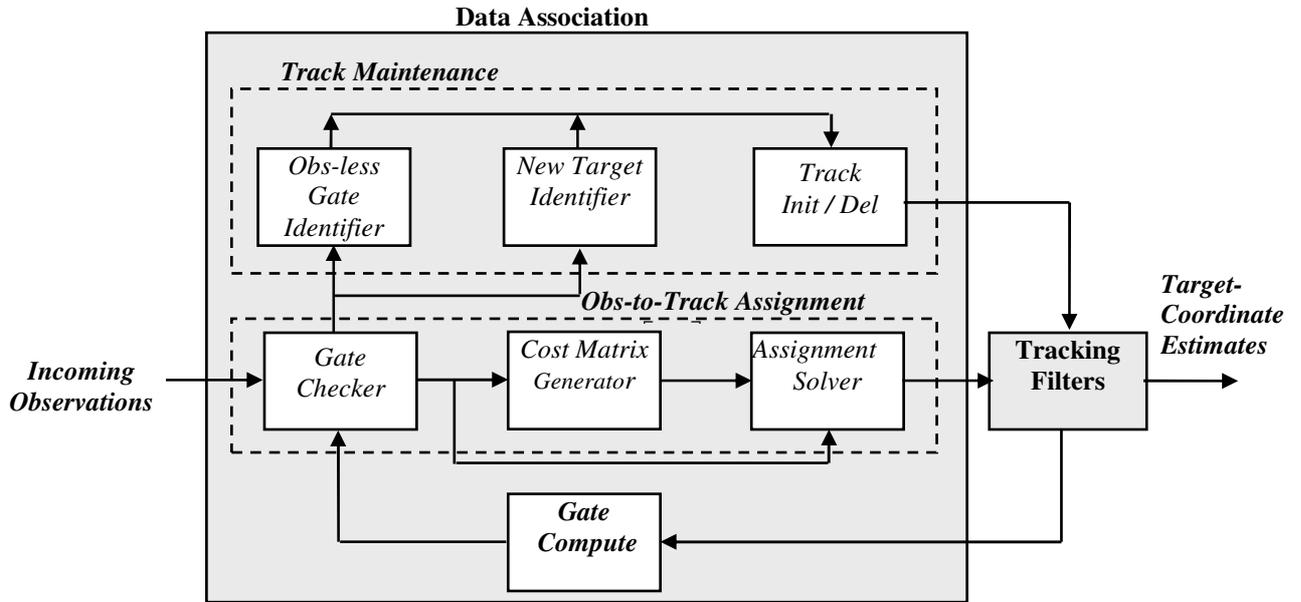


Figure 2: The Proposed MTT implementation

A. Process Model

The *process model* mathematically projects the process current state to the future. This can be presented in a linear stochastic difference equation as

$$Y_k = A Y_{k-1} + B U_k + W_{k-1} \quad (3.1)$$

In equation (3.1) Y_{k-1} and Y_k are n -dimensional state vectors that includes the quantities to be estimated. Vector Y_{k-1} represents the state at instant $k-1$ while Y_k represents the state at instant k . The $n \times n$ matrix A in the difference equation (3.1) relates the state at time step $k-1$ to the state at time step k , in the absence of either a driving function or process noise. Matrix A is the assumed known *state transition matrix* which may be viewed as the coefficient of state transformation from instant $k-1$ to instant k , in absence of any driving signal and process noise. The $n \times l$ matrix B relates the optional control input $U_k \in \mathcal{R}^l$ to the state Y_k whereas W_{k-1} is zero-mean additive white Gaussian process noise (AWGN) with assumed known covariance Q . Matrix B is the assumed known *control matrix* and U_k is the deterministic input, such as the relative position change associated with the host-vehicle (own ship) motion.

B. Measurement Model

To describe the relationship between the true state and the measurements (observations) a *measurement model* is required. It can be described as a linear expression

$$Z_k = H Y_k + V_k \quad (3.2)$$

Here Z_k is the measurement or observation vector containing two elements distance d and angle θ as shown below. The $m \times n$ *observation matrix* H in the measurement

equation (3.2) relates the current state to the *measurement (observation) vector* Z_k . The terms V_k in equation (3.2) is a random variable representing the measurement noise.

For implementation we chose the example case given in [2]. In this example the matrices and vectors in equations (3.1) and (3.2) have the forms shown below. In the rest of the paper the numerical values of all the matrix and vector elements are borrowed from this example. Quantity T is equal to 0.02 seconds and it is the radar Pulse Repetition Time (PRT) specific to the radar unit we are using in our project. Y_k , A and Z_k have the following forms:

$$Y_k = \begin{bmatrix} y_{11} \\ y_{21} \\ y_{31} \\ y_{41} \end{bmatrix} \quad A = \begin{bmatrix} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Z_k = \begin{bmatrix} d \\ \theta \end{bmatrix}$$

Here y_{11} is the target range or distance, y_{21} is range rate or speed, y_{31} is angle (azimuth), y_{41} is angle rate or angular speed. In vector Z_k the element d is the distance measurement and θ is the azimuth angle measurement. Matrix B and control input U_k are ignored here because they are not necessary in our application.

Having devised the process and measurement models, we need an *estimator* which would use these models to estimate the true state. Our solution is based on the Kalman filter which is a recursive Least Square Estimator (LSE) and is considered to be the optimal estimator for linear systems [4,5].

C. Kalman Filter

Many good derivations of the Kalman filter and discussions of its applications are presented in the literature [4,5,6], so only the resulting equations are given here.

Given the process and the measurement models from (3.1)

and (3.2), the Kalman filter equations are

$$\hat{Y}_k^- = A \hat{Y}_{k-1} + B U_k \quad (3.3a)$$

$$P_k^- = A P_{k-1} A^T + Q \quad (3.3b)$$

$$K = P_k^- H^T (H P_k^- H^T + R)^{-1} \quad (3.3c)$$

$$\hat{Y}_k = \hat{Y}_k^- + K (Z_k - H \hat{Y}_k^-) \quad (3.3d)$$

$$P_k = (I - KH) P_k^- \quad (3.3e)$$

Here \hat{Y}_k is state estimation vector, \hat{Y}_k^- is state prediction vector, K is Kalman gain matrix, P_k^- is prediction error covariance matrix, P_k is estimation covariance matrix and I is an identity matrix of the same dimensions as P_k .

The newly introduced vectors and matrices in equations (3.3) have the following forms.

$$\hat{Y}_k = \begin{bmatrix} \hat{y}_{11} \\ \hat{y}_{21} \\ \hat{y}_{31} \\ \hat{y}_{41} \end{bmatrix} \quad \hat{Y}_k^- = \begin{bmatrix} \hat{y}_{11}^- \\ \hat{y}_{21}^- \\ \hat{y}_{31}^- \\ \hat{y}_{41}^- \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad R = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} = \begin{bmatrix} 10^6 & 0 \\ 0 & 2.9 * 10^{-4} \end{bmatrix}$$

$$Q = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{31} & q_{32} & q_{33} & q_{34} \\ q_{41} & q_{42} & q_{43} & q_{44} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 330 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.3 * 10^{-8} \end{bmatrix}$$

Here \hat{y}_{11}^- is range Prediction \hat{y}_{21}^- is speed prediction \hat{y}_{31}^- is azimuth angle prediction, \hat{y}_{41}^- is angular speed prediction \hat{y}_{11} is range estimate, \hat{y}_{21} speed estimate, \hat{y}_{31} is angle estimate and \hat{y}_{41} is angular speed estimate, all for instant k .

Matrices K and P_k^- have the following forms.

$$K = \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \\ k_{31} & k_{32} \\ k_{41} & k_{42} \end{bmatrix} \quad P_k^- = \begin{bmatrix} P_{11}^- & P_{12}^- & P_{13}^- & P_{14}^- \\ P_{21}^- & P_{22}^- & P_{23}^- & P_{24}^- \\ P_{31}^- & P_{32}^- & P_{33}^- & P_{34}^- \\ P_{41}^- & P_{42}^- & P_{43}^- & P_{44}^- \end{bmatrix}$$

Matrix P_k is similar in form to P_k^- except for the superscript '-'. The scan index k has been ignored in the elements of these matrices and vectors for the sake of notational simplicity.

The Kalman filter cycles through the “prediction–correction” loop shown pictorially in figure 3. In the prediction step (also called *time update*), the filter predicts the next state and error covariance associated with the state prediction using equations (3.3a) and (3.3b) respectively. In

the correction step (also called *measurement update*), the filter calculates the filter gain and estimates the current state and the error covariance of this estimation using equations (3.3c) through (3.3e).

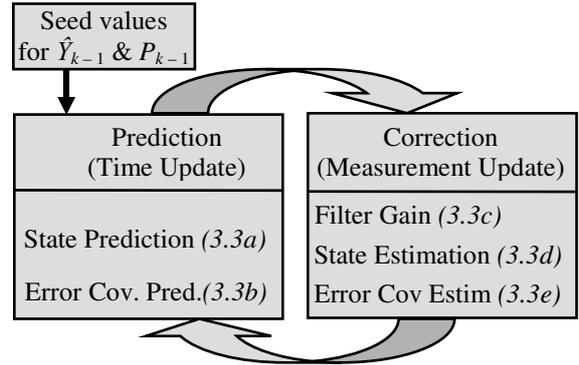


Figure 3: The Kalman filter

Figure 4 shows the state (position) of a target estimated by the Kalman filter against the true position and that measured by the radar. Notice how closely the estimated position follows the real position as compared with the measured position after the 20 transitional iterations.

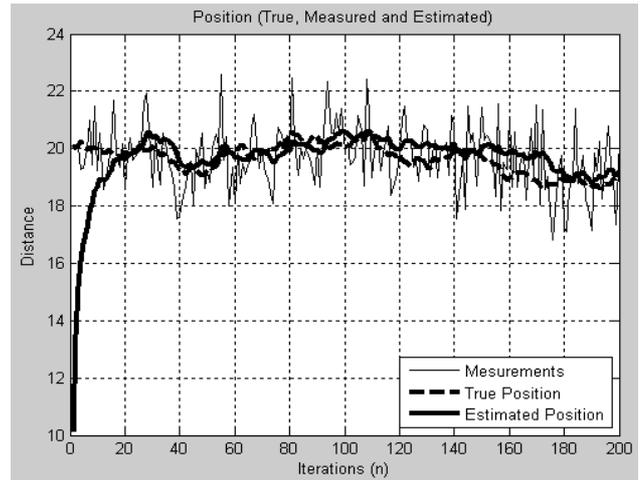


Figure 4: Estimated target position

As we are dealing with multiple targets at the same time, we have to identify which of the incoming observed states to associate with which of the predicted states for making the estimations. This is the job of *data association* function. The data association sub-modules are explained in the following paragraphs.

D. Gate Computation

The first step in data association is gate computation. The “Gate Compute” block receives \hat{Y}_k^- and P_k^- from the Kalman Filters for all the tracks. Using these two quantities the “Gate Compute” block defines the probability gates or windows which are used to verify whether an incoming

observation can be associated with an existing track. The gate computation process is summarized below.

Define \tilde{Y} to be the *residual* or *innovation* vector which is the difference between the actual measurement Z_k and the expected (predicted) measurement vector $[H\hat{Y}_k^-]$. In general at scan k ,

$$\tilde{Y}_i = Z_i - H \hat{Y}_i^- \quad (3.4)$$

Time index k is ignored for simplicity. Now define a rectangular region such that an observation vector Z_k (with elements $z_{k\perp}$) is said to satisfy the gate of a given track if all elements \tilde{Y}_{il} of residual vector \tilde{Y}_i satisfy the relationship

$$|Z_{k\perp} - H \hat{Y}_{il}^-| = |\tilde{Y}_{il}| \leq K_{Gl} \sigma_r \quad (3.5)$$

Here i is an index for track i , G signifies for gate and l is replaced either by d or by θ , whichever is appropriate (see equations 3.10 and 3.11). The term σ_r is the *residual standard deviation* and is defined in terms of the *measurement variance* σ_z^2 and *prediction variance* $\sigma_{\hat{y}_k}^2$. Typical choice for K_{Gl} is $[K_{Gl} \geq 3.0]$. This large choice of gating coefficient is typically made in order to compensate for the approximations involved in modeling the target dynamics through the Kalman filter covariance matrix [1]. This concept comes from the famous “3 sigma rule” in statistics.

In its matrix form equation (3.4) can be simplified down to

$$\tilde{Y}_i = \begin{bmatrix} \tilde{y}_{i11} \\ \tilde{y}_{i21} \end{bmatrix} = \begin{bmatrix} d_i - \hat{y}_{11}^- \\ \theta_i - \hat{y}_{31}^- \end{bmatrix} \quad (3.6)$$

Consequently equation (3.5) gives

$$\begin{bmatrix} \tilde{y}_{i11} \\ \tilde{y}_{i21} \end{bmatrix} \leq K_{Gl} \sigma_r \quad (3.7)$$

The residual standard deviations for the two state vector elements are defined as follows

$$\sigma_{rd} = \sqrt{r_{11} + p_{22}^-} \quad (3.8)$$

$$\sigma_{r\theta} = \sqrt{r_{22} + p_{44}^-} \quad (3.9)$$

From (3.7), (3.8) and (3.9) we get

$$|\tilde{y}_{i11}| = |\tilde{y}_{id}| \leq 3.0 \sqrt{r_{11} + p_{22}^-} \quad (3.10)$$

$$|\tilde{y}_{i21}| = |\tilde{y}_{i\theta}| \leq 3.0 \sqrt{r_{22} + p_{44}^-} \quad (3.11)$$

Equations (3.10) and (3.11) together put the limits on the residuals \tilde{y}_{id} and $\tilde{y}_{i\theta}$. In other words, the difference between an incoming observation and prediction for track i must obey equations (3.10) and (3.11) for the observation to be assigned to track i .

E. Gate Checker

The “Gate checker” tests whether an incoming observation fulfills the conditions set in equations (3.10) and (3.11). In effect, this block sets or resets the binary elements of an $N \times N$ matrix termed as the “Gate Mask” matrix M .

Here N is the maximum number of targets and tracks being processed. If an observation j fulfills both these conditions for a track i , the corresponding element m_{ij} of matrix M is set to 1 otherwise it is reset to 0.

Matrix M would typically have more than one 1’s in a column or a row. The ultimate goal is to have only one ‘1’ in a row or a column for a one-to-one coupling of observations and predictions.

$$M = \begin{matrix} & \underbrace{\hspace{10em}}_{\text{Tracks}} & \\ \left. \begin{matrix} m_{11} & m_{12} & \dots & \dots & m_{1N} \\ m_{21} & m_{22} & \dots & \dots & m_{2N} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ m_{N1} & m_{N2} & \dots & \dots & m_{NN} \end{matrix} \right\} & \text{Obs} & \\ m_{ij} = \begin{cases} 1 & \text{if obs } j \text{ obeys (3.10) \& (3.11) for track } i \\ 0 & \text{otherwise} \end{cases} \end{matrix}$$

To achieve this goal, the first step is to attach a cost to every possible coupling. This is done by the “Cost Generator” block explained next.

F. Cost Matrix Generator

The “Cost Matrix Generator” associates a cost with every possible observation-prediction pairing. The cost c_{ij} for associating an observation j with a prediction i is the statistical distance d_{ij}^2 between the observation and the prediction when m_{ij} is 1. The cost is an arbitrarily large number when m_{ij} is 0.

The distance d_{ij}^2 is calculated as follows.

$$\text{Define } S_{ij} = H P_k^- H^T + R \quad (3.12)$$

Here j is an index for observation j and i is the index for track i in a scan, S_{ij} is the residual covariance matrix. The statistical distance d_{ij}^2 is the norm of the residual vector \tilde{Y}_{ij}

$$d_{ij}^2 = \tilde{Y}_{ij}^T S_{ij}^{-1} \tilde{Y}_{ij} \quad (3.13)$$

$$C = \begin{matrix} \text{Tracks} \\ \left[\begin{array}{cccccc} c_{11} & c_{12} & \bullet & \bullet & \bullet & c_{1N} \\ c_{21} & c_{22} & \bullet & \bullet & \bullet & c_{2N} \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ c_{N1} & c_{N2} & \bullet & \bullet & \bullet & c_{NN} \end{array} \right] \end{matrix} \left. \vphantom{\begin{matrix} \text{Tracks} \\ \left[\begin{array}{cccccc} \end{array} \right] \end{matrix}} \right\} \text{Obs}$$

$$c_{ij} = \begin{cases} \text{Arbitrary large number if } m_{ij} \text{ is } 0 \\ d_{ij}^2 \text{ if } m_{ij} \text{ is } 1 \end{cases}$$

Equation (3.12) can be written in its matrix form and simplified down to

$$S_{ij} = \begin{bmatrix} p_{11}^- + r_{11} & p_{13}^- \\ p_{31}^- & p_{33}^- + r_{22} \end{bmatrix} \quad (3.14)$$

Using equations (3.6), (3.13) and (3.14), d_{ij}^2 is calculated as follows.

$$d_{ij}^2 = \frac{\begin{bmatrix} \tilde{y}_{i11} & \tilde{y}_{i21} \end{bmatrix} \begin{bmatrix} p_{33}^- + r_{22} & -p_{13}^- \\ -p_{31}^- & p_{11}^- + r_{11} \end{bmatrix} \begin{bmatrix} \tilde{y}_{i11} \\ \tilde{y}_{i21} \end{bmatrix}}{\left((p_{11}^- + r_{11}) * (p_{33}^- + r_{22}) - p_{13}^- * p_{31}^- \right)}$$

Recall here that $\tilde{y}_{i11} = \tilde{y}_{id}$ and $\tilde{y}_{i21} = \tilde{y}_{i\theta}$.

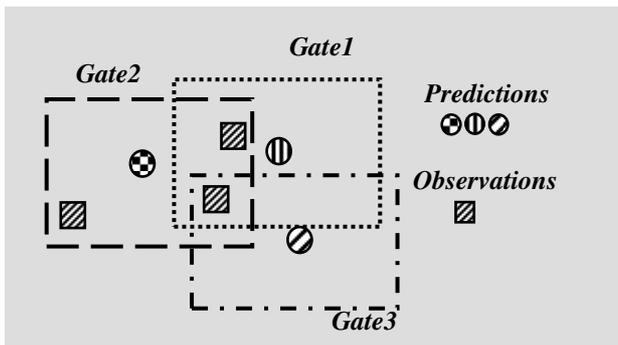


Figure 5: Conflict situation in data association

The cost matrix demonstrates a conflict situation where several observations are potential candidates to be associated with a particular prediction and vice versa. A conflict situation is illustrated in figure 5.

To resolve the conflicts, the cost matrix is passed on to the “Assignment Solver” block which treats it as the well

known assignment problem.

G. Assignment Solver

The assignment problem is stated as follows.

Given a cost matrix of elements C_{ij} , find a matrix $X = \{x_{ij}\}$, such that $C = \sum_{i=1}^n \sum_{j=1}^m c_{ij} x_{ij}$ is minimized

$$\text{subject to } \begin{cases} \sum_i x_{ij} = 1 \forall j \\ \sum_j x_{ij} = 1 \forall i \end{cases}$$

Here x_{ij} is a binary variable used for ensuring that an observation is associated with one and only one track and a track is associated with one and only one observation. This requires x_{ij} to be either 0 or 1 i.e. $x_{ij} \in \{0,1\}$.

There are several algorithms for finding matrix X. The most commonly used among them are Munkres algorithm [9] and Auction algorithm. We use the former in our application. Matrix X below shows a result of the “Assignment Solver” for a 4x4 cost matrix. It shows that observation 1 is to be paired with prediction 3, observation 2 with prediction 1 and so forth.

$$X = \begin{matrix} \text{Predictions (Tracks)} \\ \left[\begin{array}{cccc} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{array} \right] \end{matrix} \left. \vphantom{\left[\begin{array}{cccc} \end{array} \right]} \right\} \text{Observations}$$

These pairs are then finally passed on to the relevant Kalman filters to estimate the states of the concerned targets.

All the steps “A” through “G” are repeated for ever in the loop in figure 1. However, there are certain cases where some additional steps have to be taken too. These steps and the circumstances where they become relevant are explained in the next section. Together these three steps are called “Track Maintenance”.

H. Track Maintenance

In real conditions there would be one or more targets that are detected in the current scan but did not exist in the previous scans. On the other hand there would be situations where one or more of the targets being tracked would no more be in the radar FOV. The first case is the “New target Identification” whereas the latter one is the “Observation-less Gate Identification” case. A new target is identified when its observation fails all the already established gates i.e. when all the elements of a row in the “Gate Mask” matrix M are zero. The “new target identifier” starts a counter for the newly identified target. If the counter reaches 3 in five scans, the target is confirmed and a new track is initiated for it. The

counter is reset every five scans. The case of “*Observation-less Gate*” indicates the disappearance of a target from radar FOV. This is manifested when all the elements of a column in the “*Gate Mask*” matrix M are zero. The “*Obs-less Gate Identifier*” looks for 3 consecutive “*misses*” in 5 scans to confirm the disappearance of a target. The “*Track Init/Del*” initiates or deletes a track when needed.

IV. APPLICATION MAPPING TO MPSOC

We coded the application in ANSI C and distributed the application over different processors as distinct functions. Communication among the function is such that we have to implement a producer-consumer architecture for the system as shown in figure 6. Details of this architecture can be found in [15]. Similar considerations have been proposed in [10, 11, 12].

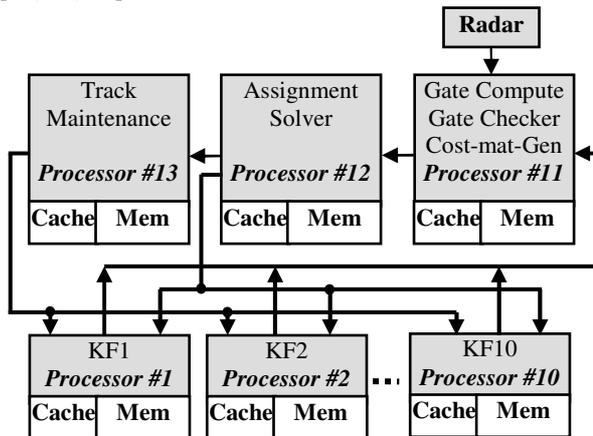


Figure 6 : MPSoC on FPGA

Kalman filter, as mentioned earlier above, is recursive algorithm looping around prediction and correction steps. Both these steps involve matrix operations on floating point quantities. This makes the filter a strong candidate to be mapped onto a separate processor. Thus for tracking 10 targets at a time, we need 10 identical processors executing 10 Kalman filters.

The assignment-solver is an algorithm consisting of six distinct iterative steps [9]. Looping through these steps demands a long execution time hence this function cannot be combined with any of the other functions. So the assignment solver is another strong candidate to be mapped onto a separate processor

The “*Gate Compute*” block regularly passes information to “*Gate Checker*” which in turn, is in constant communication with “*Cost Matrix Generator*”. So we group these three blocks together and map them onto a single processor to minimize inter-processor communication which would have required extra buffers and would have added to the complexity of the system. Avoiding unnecessary inter-processor communication is also desirable for saving power.

The three blocks of the “*Track Maintenance*” sub-function individually don’t demand heavy computational

resources, so we grouped them together for mapping onto a processor.

Using Altera’s Quartus II and SOPC design tools, we implemented the system with the NIOS II processors. The choice of using NIOS II (e) or NIOS II (s) is driven by the results of several tests done on the codes. Inserting time stamps into the code at strategic points, we came to know that the filter spent more than 90% of the time in multiply/divide operations. So the optional “hardware multiply” and “hardware divide” are included with the standard processors for the filters to augment their ALU’s. This augmentation is termed as “*custom instruction*” in NIOS II literature [14]. It is helpful in speeding up the filtering functions because the filters involve numerous multiply/divide operations on floating point numbers.

V. DISCUSSION AND RESULTS

In the very early stages of the work, we implemented a 2-state Kalman filter in hardware using VHDL. It not only took enormous design efforts but also consumed 23,327 LUTs on the FPGA. In fact one 2-state Kalman filter needs about 48% of the Stratix II 2S60 FPGA resources.

In our current design we are using ten 4-state filters apart from the other system components. Implementing a fully hardware system with ten 4-state filters would have been simply infeasible. The NIOS II (e) consumes 600 to 700 logic elements (LEs) whereas NIOS II (s) consumes 1200 to 1400 LEs. The whole MPSoC that we propose consumes almost 30,000 logic elements which are 50% of the reconfigurable resources on Stratix II 2S60 FPGA.

The radar sensor we are using has a PRT of 20 ms so we have to go round the “data association-filtering” loop for all the 10 targets in less than 20 ms. Using 100 MHz clock, the slowest function in the application i.e. the filter takes 15 ms to complete the prediction-correction loop discussed earlier. This is well below the 20 ms threshold set by the radar PRT.

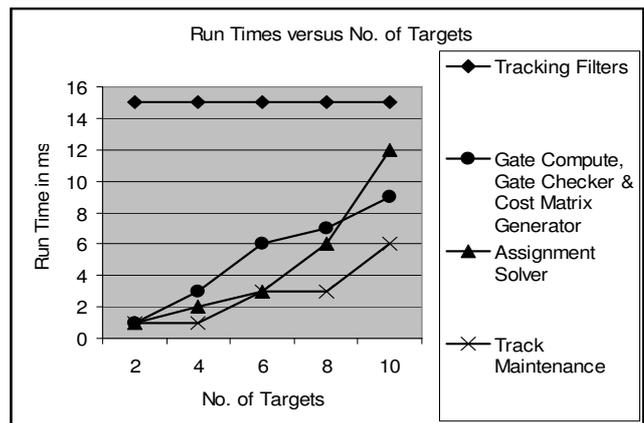


Figure 7: Run times Vs No. of targets

Figure 7 shows how the run times of different functions vary with respect to the maximum number of targets being tracked. Because there are as many filters as the maximum

number of targets and all the filters run concurrently on separate processors, their run time remains unchanged whatever the number of targets and filters. The run time for the combined functions of Gate-Compute, Gate-Checker and Cost-Matrix-Generator varies almost linearly with the number of targets. Here it would be expected that the run time vary exponentially because the number of elements in Gate Mask matrix M and cost matrix C is the square of the number of targets. But a higher number of targets also increase the number of observations failing one or several gates. As discussed above, the computations reduce considerably when an observation does not pass one or several gates. This accounts for the almost linear change in the run time for these functions. The Assignment Solver shows an exponential rise in run time because the algorithm is $O(N^2)$. The Track Maintenance function behaves randomly since the numbers of disappearing and new targets vary randomly.

VI. RELATED WORK

To our understanding, comprehensive literature about implementation of complete MTT system in FPGA does not exist. Some work has been done on isolated parts of the MTT system. For example in [2] the authors propose an FPGA implementation of the estimation part of MTT system. Apart from being limited only to estimation, this is a fully hardware implementation. As mentioned earlier in the introduction, fully hardware designs lack the flexibility and programmability needed for the ever evolving modern day applications. A fully hardware implementation of the Kalman filter only, is also proposed in [3]. This implementation also has all the above mentioned limitations for the same reasons. An attempt to implement an MTT system in hardware for maritime application is documented in [7]. Besides being a completely hardware implementation and specific to maritime applications, the work presented here is inconclusive.

In contrast to the works mentioned above, we consider a complete MTT system implementation. Our MPSoC architecture of the system is inherently flexible, programmable and scalable. Thus it can evolve very easily with advances in technology, improvement in application algorithms and market demands. The use of several concurrently running processors achieves the overall run time constraints. Several low frequency processors running concurrently consume less power compared to a single processor with a high frequency [13]. The reconfigurability of the components enables their customization according to application and further saving FPGA resources.

VII. CONCLUSIONS AND FUTURE WORK

We presented an application-specific MPSoC architecture for MTT based driver assistant system. The system is implemented in FPGA with Altera's SOPC builder and

NIOS II processors. The system uses 50% of the reconfigurable resources on a Stratix II 2s60 FPGA which contains 60,000 logic elements. We currently cater for a maximum of 10 targets. The system is easily evolvable. It is not only a complete embedded MTT solution but is also economical and power efficient as compared to fully hardware implementations. We plan to take the system further up the evolution hierarchy in the near future. On the application side, we intend to take dynamically varying number of targets into account. This would require us to dynamically vary the number of filter configurations on the FPGA. Tracking precision can be improved through better / newer algorithms e.g. extended Kalman Filter (EKF), auction algorithm for assignment, radar-signature aided data association etc.

On the architecture side, new processor configurations and interconnections shall be evaluated for run time and power consumption improvement. To integrate the system with other electronic safety systems onboard a vehicle, we shall add a CAN (Controller Area Network) interface to the system.

REFERENCES

- [1] Samuel Blackman and Robert Popoli, "Design and analysis of modern tracking systems", Artech House Publishers 1999, ISBN 1-58053-006-0.
- [2] Zoran Salcic and Chung-Ruey Lee, "FPGA-Based Adaptive Tracking Estimation Computer" IEEE transactions on aerospace and electronic systems, 2001.
- [3] Salcic, Z. et al. "Scalar-based direct algorithm mapping FPLD implementation of Kalman filter." IEEE Transactions on Aerospace and Electronic Systems, 2000.
- [4] Kalman, R. E. "A New Approach to Linear Filtering and Prediction Problems", Transaction of the ASME--Journal of Basic Engineering, pp.35-45(Mar 1960).
- [5] Welch, G and Bishop, G. 2001. "An introduction to the Kalman Filter", <http://www.cs.unc.edu/~welch/kalman/>
- [6] Brookner, E. "Tracking and Kalman filtering made easy" John Wiley & Sons Inc. 1998.
- [7] Boismenu, Y. "Etude d'une carte de tracking radar---" Thèse de doctorat année 2000, Université de Bourgogne, France.
- [8] P. Konstantinova et al. "A study of Target Tracking Algorithm Using Global Nearest Neighbor Approach" CompSysTech' 2003.
- [9] Munkres' Assignment Algorithm, Modified for Rectangular Matrices <http://csclab.murraystate.edu/bob.pilgrim/445/munkres.html>
- [10] Joost, R.; Salomon, R. "Advantages of FPGA-based multiprocessor systems in industrial applications" Industrial Electronics Society, 2005. IECON 2005. 31st Annual Conference of IEEE Volume, Issue, 6-10 Nov. 2005.
- [11] Epson's Breakthrough REALOID Printer SOC Powered by Multiple Xtensa Processors http://www.tensilica.com/products/lits_success-stories_epson.htm
- [12] Hannu Penttinen, Tapio Koskinen, Marko Hännikäinen."Leon3 MP on Altera FPGA" Altera Innovate Nordic 2007, Final Project Report 8/28/2007
- [13] Frank Schirrmeister Imperas, Inc." Multi-core Processors: Fundamentals, Trends, and Challenges" Embedded Systems Conference 2007 ESC351.
- [14] Altera Corporation. "NIOS II processor reference handbook". www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf
- [15] J. Khan et al. "An MPSoC Architecture for the Multiple Target Tracking Application in Driver Assistant System" 19th IEEE International Conference ASAP08, 2-4 July 2008, Leuven Belgium.

Palmprint Verification Using DCT Coefficients and Linear Discriminant Analysis

M. Laadjel, A. Bouridane, O. Nibouche
 Institute of Electronics, Communications
 and Information Technology (ECIT)
 Queens University Belfast, Northern Ireland
 Belfast BT3 9DT, United Kingdom

Email: {mlaadjel01, a.bouridane, o.nibouche}@qub.ac.uk

¹A. Meraoumia, ²M. Siagaa
 School of Electronics and Control
¹University of Ouargla, Ouargla, 30000
²University of Tebessa, Tebessa 12000
 Algeria
 Email: {meraou2, siagaa12}@yahoo.fr

Abstract—Biometric verification methods have proven to be very efficient, more acceptable and easy for users than traditional token based methods. This paper presents an efficient yet simple Palmprint verification system based on the discrete cosine transforms (DCT) and linear discriminant analysis (LDA). First, the dimensionality of the original palmprint images is reduced by applying DCT on the non-overlapping blocks. Next, the truncated DCT coefficients for all the blocks are concatenated to form an independent 1-D data vector. This process makes the subsequent LDA more efficient for data discrimination since implementing an LDA on the DCT space helps to maintain the most discriminating and variant palmprint features. Finally, a cosine measure is used to calculate the similarity between registered and claimed person based on palmprint data. The proposed method was assessed on PolyU palmprint database of 2000 images giving a verification rate of 99.17%.

I. INTRODUCTION

Automatic verification systems that make use of biometric data, such as, distinctive physiological and behavioral characteristics of the human, are becoming ever more widely used for access control, surveillance, computer security, and in law enforcement applications. Such biometrics are iris, retina, fingerprint, face, palmprint and voice. In the past few years palmprint based authentication has been regarded as a new attempt and a necessary complement to the existing biometric measures, due to its high acceptance by users in many access control applications. For example, users accept palmprint biometrics as is being less stressful than fingerprint. Moreover, users admit that they feel less comfortable when iris is used as a biometric system since they have to present their eyeball to the sensing device. Furthermore, when compared to other biometrics traits, palmprint offers a large palm area for feature extraction with plenty of discriminating features such as principal lines, wrinkles, ridges, minutiae points, singular points, texture, ... etc.

Recently a number of research papers on palmprint based biometrics have been proposed. The proposed methods can be divided into two approaches: line-based and statical-based. In a line-based approach, recognition is based on extracting lines and creases features from the palmprint using different edge detection methods [1][2][3]. The success of this approach relies highly on the accuracy of the feature detection

schemes. However, this method is quite complex due to the difficulties in extracting line structures. In addition, creases and ridges of the palm very often overlap, which complicate the feature extraction task. Moreover, line-based approaches are not invariant against rotation, translation and illumination changes. The second statical-based approach is more reliable and is based on an attempt to capture and define the palmprint as a single data set. The palmprint image is treated as a two-dimensional pattern of intensity variations. Under this approach, palmprint matching is carried out by identifying its basic statistical regularities. Well known statical methods such as principal component analysis (PCA) and Karhunen-Loeve transform (KLT) are then employed to describe, analyze and classify palmprint data [4][5]. Other more reliable statistical approaches such as Fisherpalms technique in which a linear discriminant analysis (LDA) is performed on the reduced dimensionality feature vector resulting after a PCA is applied of raw feature data [6]. Compared with an eigenpalms approach (i.e., through using a PCA approach), the Fisherpalms approach is more insensitive to illumination variations and seeks projections that are efficient for data discrimination and dimension reduction. Another statistical approach such as independent component analysis (ICA) which computes the basis components that are statistically independent or as independent as possible based on de-correlating higher order statistics from the training images has also been used in palmprint recognition problems[7][8]. However the discriminating power and computational requirements of these approaches are greatly related to the dimensionality of the original data and the number of training samples. When a palmprint database becomes large, the training time, memory requirements will significantly increase and the discrimination performance will be affected (i.e., large covariance matrix). Consequently, dimensionality reduction is a very important step which will greatly improve the feature extraction accuracy, verification speed and storage capacity especially for palmprint biometric problems where data is (i) high dimensional, (ii) very often contains information that is less discriminative or that is not useful for recognition (irrelevant information) and (iii) data dimensionality reduction will reduce the system's memory and computational complexity.

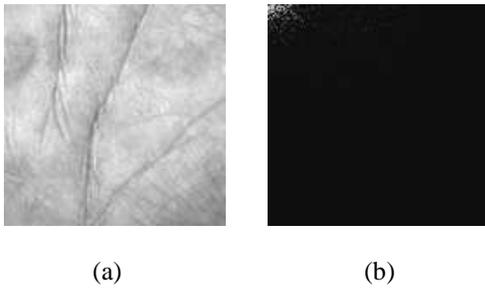


Fig. 1. Palmprint image and its DCT image

In this paper, a DCT is applied on non-overlapping blocks in the image for dominant feature compactness and then the significant DCT components are selected to form 1-D independent data vector. Once this done, an LDA is performed on the sorted data vectors to extract the most discriminating features. By combining the DCT and LDA those useful/highly discriminating features can be obtained efficiently. Finally, the similarity between a registered vectors and the tested ones is carried out using a cosine measure in order to classify/recognise palmprint data.

The rest of the paper is organized as follows. A review of discrete cosine transform (DCT) is described in Section II. The proposed approach is described in detail Section III while Section IV gives some experimental results. Finally, conclusions are drawn in Section V.

II. DISCRETE COSINE TRANSFORM

The discrete cosine transform (DCT), which has been widely used in numerous problems such as signal processing, computer vision and image processing applications, can be seen to be asymptotically equivalent to the original Karhunen-love transform (KLT) for signal de-correlation [9]. For an $M \times N$ image, the 2-D DCT can be calculated as follows:

$$F(u, v) = a(u)a(v) \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \times \cos\left[\frac{(2x+1)u\pi}{2M}\right] \cos\left[\frac{(2y+1)v\pi}{2N}\right] \quad (1)$$

with:

$$0 \leq u \leq M-1, \quad 0 \leq v \leq N-1 \quad (2)$$

and:

$$a(u) = \begin{cases} 1/\sqrt{M} & \text{for } u = 0 \\ \sqrt{2/M} & \text{for } 1 \leq u \leq M-1 \end{cases} \quad (3)$$

$$a(v) = \begin{cases} 1/\sqrt{N} & \text{for } v = 0 \\ \sqrt{2/N} & \text{for } 1 \leq v \leq N-1 \end{cases} \quad (4)$$

It can be noticed that the $M \times N$ DCT coefficient matrix covers all the spatial frequency components of the image. Fig.1(a) and (b) represent a palmprint image and its DCT coefficients, respectively. From Fig.1(b) it can be clearly seen that the coefficients with large magnitude (energy) are mainly located in the upper-left corner of the DCT matrix.

III. DCT COEFFICIENTS SELECTION

In our proposed method, a DCT is used as a pre-processing step followed by a more sophisticated method for extracting the significant features from a palmprint image. The advantage of using DCT can be attributed to its de-correlation power and feature compactness properties. As shown in Fig.2, the first step in our proposed scheme is to subdivide the palmprint image into non-overlapping pixel blocks of size 8×8 . They are subsequently processed from left to right then top to bottom and its DCT is computed. Once each block's DCT coefficients are carried out, they are re-ordered in a zig-zag fashion as illustrated in Fig.3. The resulting 1-D re-ordered vector is qualitatively arranged according to increasing spatial frequency from the low frequency (large magnitude) to the high frequency (small magnitude). In each block, the coefficients with less information can be empirically discarded since there is none rule that can be used to exactly distinguish between the spectrum pass and high frequencies. Determining the number of the discarded coefficients is a crucial task and needs an careful care. In our method, a range of a DCT coefficients are kept the same for each block after the zig-zag scan in order to use them in the construction of the eigen space. For example for an image of 128×128 palmprint image divided into 256 non-overlapping blocks with 35 coefficients each, a 1-D vector of 8960 coefficients are obtained. The 1-D vector x representing a palmprint image is composed by concatenating the truncated coefficient of the blocks $Block_1, Block_2, \dots, Block_{256}$ (see Fig.3).

Finally, in order to obtain the most discriminant features of the palmprint, an LDA is applied onto the truncated DCT coefficients obtained. This is useful since an LDA will not only reduce the high dimensionality of the truncated 1-D feature vector but also will identify those features that are highly efficient for feature discrimination. In other words, it will determine a low-dimensional space which clusters the images of the same class and separates images of different classes.

Given a set of an N training images $[x_1, x_2, \dots, x_N]$ where each image belongs to one of c classes images $[X_1, X_2, \dots, X_c]$, an LDA selects a linear transformation matrix W in such a way that the ratio of the between-class scatter and the within-class scatter is maximized. Mathematically, the between-class scatter matrix and the within-class scatter matrix are defined by:

$$S_B = \sum_{i=1}^c N_i (u_i - u)(u_i - u)^T \quad (5)$$

and

$$S_W = \sum_{i=1}^c \sum_{x_k \in X_i} (x_k - u_i)(x_k - u_i)^T \quad (6)$$

respectively, where u_i denotes the mean image of class X_i , u denotes the mean image of entire training set and N_i denotes the number of images in class X_i . However, all the scatter matrix S_W can be singular since the dimension of image vector exceeds, in general, the number of data points (usually

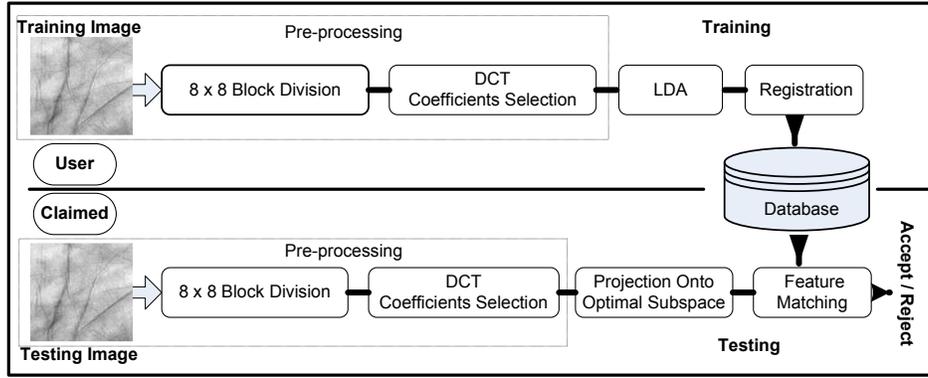


Fig. 2. Verification system description

known as singularity problem). To circumvent this problem a PCA technique is used to reduce the vector dimension before applying LDA. Then, LDA computes an orthonormal matrix W_{opt} to maximize the ratio of the determinant of the between-class scatter matrix to the determinant of the within-class scatter matrix.

$$\frac{|W^T S_B W|}{|W^T S_W W|} \quad (7)$$

W_{opt} is known to be the solution of the following eigenvalue problem:

$$S_B w_i = \lambda_i S_W w_i, i = 1, 2, \dots, m \quad (8)$$

where $[w_1, w_2, \dots, w_m]$ is the set of generalized eigenvectors corresponding to the $m(m \leq c - 1)$ largest generalized eigenvalues $\lambda_i, i = 1, 2, \dots, m$. The dimensionality reduction also allows S_W and S_B to be efficiently calculated. The optimal linear feature extractor W_{opt} is then defined as:

$$W_{opt} = W_{lda}^T * W_{pca}^T \quad (9)$$

where W_{pca} is the PCA projection matrix and W_{lda} is the optimal projection obtained by maximizing:

$$W_{lda} = \operatorname{argmax} \frac{|W^T W_{pca}^T S_B W_{pca} W|}{|W^T W_{pca}^T S_W W_{pca} W|} \quad (10)$$

The discriminating feature vectors y projected from the truncated DCT domain to the optimal subspace can be calculated as follows:

$$P = W_{opt}^T x \quad (11)$$

where x are the truncated DCT coefficient vectors.

The similarity between a training and testing images is carried out using cosine distance given by:

$$\operatorname{sim}(P, Q) = 1 - \frac{\sum_i^n p_i q_i}{\sqrt{\sum_i^n p_i^2 * \sum_i^n q_i^2}} \quad (12)$$

where $P = [p_0, p_1, \dots, p_n]$ is the database feature vector, $Q = [q_0, q_1, \dots, q_n]$ is the claimed feature vector and n is the length of the feature vector.

IV. EXPERIMENTAL RESULTS

To evaluate the effectiveness of the proposed method, the PolyU palmprint Database [7] consisting of 2000 palmprint images of 100 subjects was used. The resolution of all of the original palmprint images is 384x284 pixels taken with 75 dpi (see Fig.8). 20 samples from each subject were collected in two sessions: 10 samples were captured in the first session and the other 10 in the second session. The average interval between the first and the second collection was around two months. The palmprint images are orientated and the central part of the image where an area of 128x128 pixels is cropped to represent a palmprint. The lighting conditions in the two sessions clearly result to variation of visual texture of the images (see Fig.8).

The 5 images of the first session are chosen randomly as training samples while the remaining 15 images are chosen for testing. P is stored as the template for each palmprint class. In the verification stage, the input palmprint image is transformed to the DCT domain, then projected onto the optimal eigen subspace to compute its feature vector Q . Finally, the resulting feature vectors are compared with the registered templates to obtain the verification result.

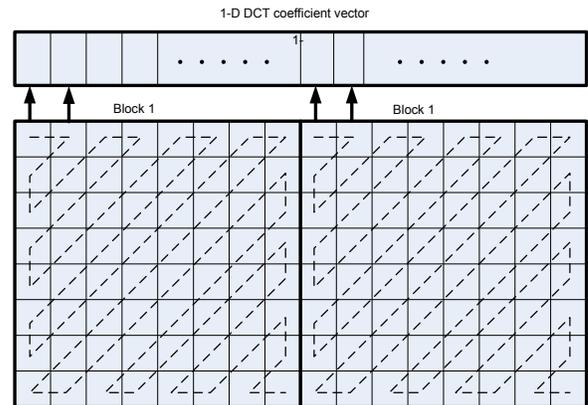


Fig. 3. DCT feature vector formulation using blocks coefficients

TABLE I
PERFORMANCE VERSUS NUMBER OF THE DCT COEFFICIENTS

Nbr of Coeffs	Feature vector length	Threshold	EER (%)
9	2304	0.394	1.325
14	3584	0.390	1.135
20	5120	0.387	1.054
27	6912	0.390	0.905
35	8960	0.390	0.835
63	16128	0.381	0.821

A. Number of DCT Coefficients

In order to determine the effect of the number of DCT coefficients on the verification performance, the verification rate with a varying numbers of DCT coefficients from each block has been evaluated using the equal error rate (EER) which is the rate when both false acceptance rate (FAR) and false acceptance rate (FRR) are equal. Table I depicts the corresponding EER values on the optimal operating point. It is clearly shown that varying the number of coefficients from 19, 14, 20, 27, 35 to 63 in each block does not yield to much improvement which mean that the discriminating features are located on a few AC coefficients. Fig.4 depicts the corresponding receiver operating characteristic (ROC) curves showing clearly the system performance at different FAR values. It can be seen that, if 35 and 63 DCT coefficients are selected from each block, the difference in the verification accuracy is marginal. Therefore, 35 coefficients provide a good compromise between vector feature length and verification accuracy and this has been used thereafter in this work since it also gives a good memory saving and verification accuracy compromise, in order to distinguish between genuine subject and imposters

In the investigation, each of the testing palmprint images was matched with all of the registered palmprint templates. Since a registered palm has 5 templates in the templates databases, a palmprint image of the testing database is matched with all the templates to produce 5 correct cosine distances. The minimum of these 5 distances has been taken as the correct verification distance. Similarly, a palmprint image in the testing database is compared with all templates to produce 495 incorrect cosine distances. A matching is noted as a correct match if two palmprint images are from the same palm. The total number of matchings is 750,000. The number of comparisons that have a correct matching are 7500 (1500 x 5) and the rest are incorrect matchings. The higher the distance is, the higher is the dissimilarity between the registered and claimed palmprint images. Fig. 5 shows that the distance distributions of the genuine and imposter matching scores have two distinct and distant peaks. The corresponding ROC curve depicting the values of FAR and FRR is illustrated in Fig. 6. This corresponds to an EER of 0.83% for a threshold of 0.390 and the genuine acceptance rate (GAR) of 99.17% at a value of FAR of 0.83%. From this investigation, one can conclude that a truncated number of DCT coefficients contains the

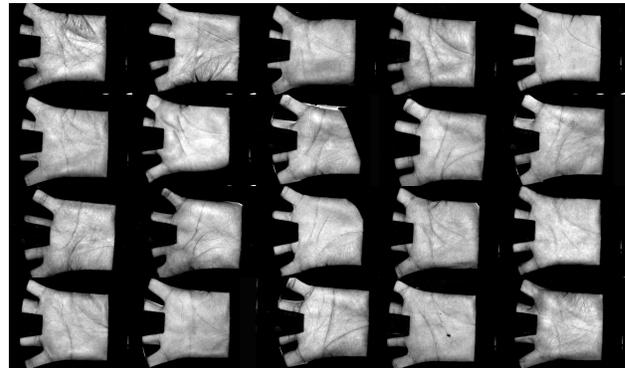


Fig. 8. Palmprint images samples from the polyU database

dominant discriminating feature (energy) since the verification rate does not improve by much and seems to stabilize beyond the number of the DCT coefficients used. This comes from the fact that the higher frequency coefficients do not carry much energy and might be related to noise. It is also interesting to mention that the use of truncated DCT coefficients guarantees an optimal projection direction for the LDA. In addition, the use of just 35 coefficients per each block yields to significant saving in memory requirement and computation complexity.

The proposed approach has also been compared with the following approaches: (i) raw (spatial domain) linear discrimination analysis (raw LDA), which is a widely used discrimination criterion in biometrics, (ii) raw principal component analysis (raw PCA) and (iii) a similar approach in which the DCT coefficients are replaced by Fourier spectrum (FS + LDA and FS + PCA)[10]. All the above techniques have been implemented on the same database using 5 images for training and 15 images for testing. The length of the eigen vectors that have been used to feed the matcher (cosine distance) are 99 and 100 using LDA and PCA techniques, respectively. The performances are depicted by the ROC curves (Fig.7) suggesting that the eigen vectors calculated from the frequency domain (DCT or Fourier space) efficiently model the palmprint images and yield to a significant improvement compared to the raw domain (spatial domain). For example, the DCT + LDA yields an EER of 0.83%, a verification rate of 99.17% and reduces the training (or testing) feature vector length (FVL) from 16384 components to about half (8960). It is interesting to observe that DCT and FS spaces provide more or less the same performance if combined with PCA and LDA.

V. CONCLUSION

This paper proposes an efficient approach for palmprint verification. The proposed technique is based on a combination of DCT and LDA techniques. It can clearly be concluded from the results that using DCT for dominant feature extraction is a very useful pre-processing step since it helps to remove the irrelevant/redundant information from the original palmprint image. In addition, this process of constructing an optimal

eigen subspace helps to keep the more useful discriminating information that can efficiently characterize the palmprint. The determination of the exact number of the DCT coefficients that can give the best compromise between the verification accuracy and feature vector length is also crucial task and will be investigated carefully in our future work.

REFERENCES

- [1] D. Zhang and W. Shu, "Two novel characteristics in palmprint verification: Datum point invariance and line feature matching," *Proceedings of the IEEE Int conference. Pattern Recognition*, vol. 24, pp. 1463–1467, 1999.
- [2] J. You, W. Li, and D. Zhang, "Hierarchical palmprint identification via multiple feature extraction," *Proceedings of the IEEE Conf. Pattern Recognition*, vol. 35, pp. 847–859, 2002.
- [3] C. C. Han, H. L. Cheng, C. L. Lin, and K. C. Fan, "Personal authentication using palmprint features," *Pattern Recognition Letter*, vol. 36, pp. 281371, 2003.
- [4] G. Lu, D. Zhang, and K. Wang, "Palmprint recognition using eigenpalms features," vol. 24, pp. 14731477, 2003.
- [5] S. Ribaric and I. Fratric, "A biometric identification system based on eigenpalm and eigenfinger features," vol. 24, pp. 1698–1709, 2005.
- [6] X. Wu, D. Zhang, and K. Wang, "Fisherpalms based palmprint recognition," in *Pattern Recognition Letters*, Aug. 2003, vol. 24, pp. 2829–2938.
- [7] T. Connie, A. T. B. Jin, M. G. K. Ong, and D. N. C. Ling, "An automated palmprint recognition system," in *Pattern Recognition Letters*, May. 2005, vol. 23, pp. 501–515.
- [8] G. M. Lu, K. Q. Wang, and D. Zhang, "Wavelet based independent component analysis for palmprint identification," in *Proceedings of the IEEE Int Conf. Machine Learning and Cybernetics*, 2004, vol. 6, pp. 3547– 3550.
- [9] X. Y. Jing and D. Zhang, "A face and palmprint recognition approach based on discriminant dct feature extraction," in *IEEE Trans. Systems, Man and Cybernetics*, Dec. 2004, vol. 34, pp. 2405– 2415.
- [10] M. Laadjel, A. Bouridane, and F. Kurugollu, "Eigenspectra palmprint recognition," *Proceedings of the IEEE Int conference.delta.*, vol. 0, pp. 382–385, 2008.

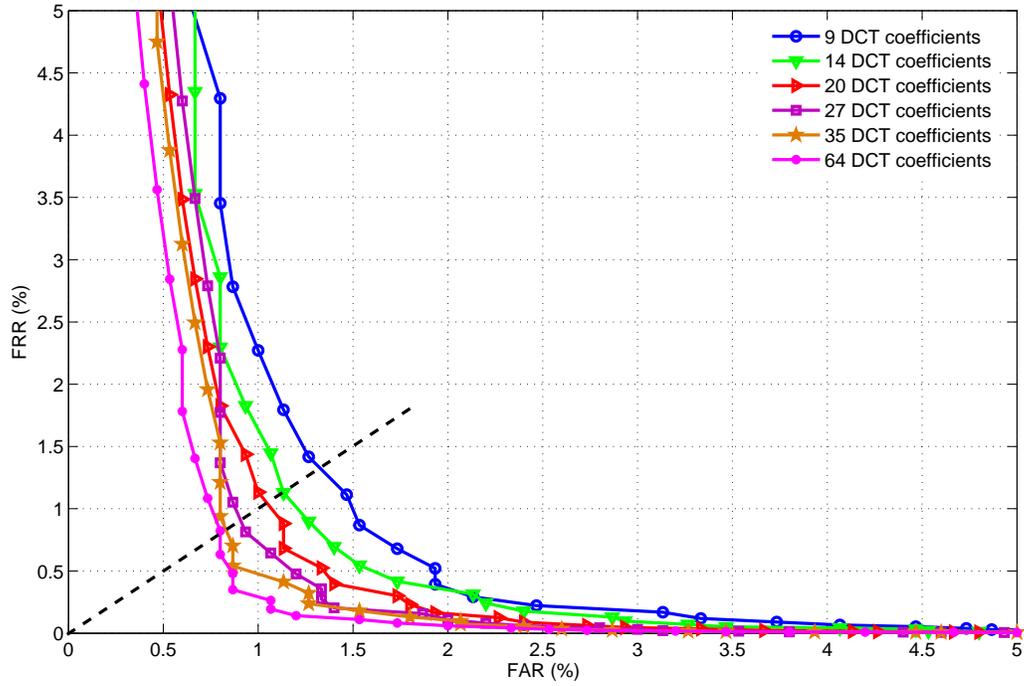


Fig. 4. ROC curves evaluation of the effect of the number of the DCT coefficients on the system performance

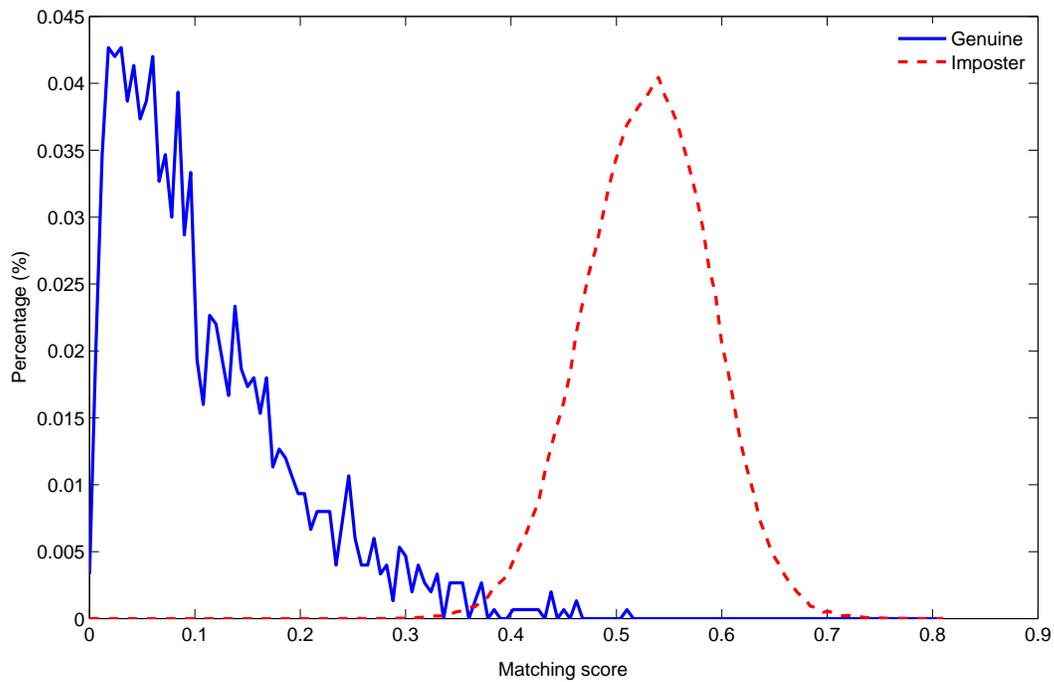


Fig. 5. Genuine and imposters distributions using 35 DCT coefficients

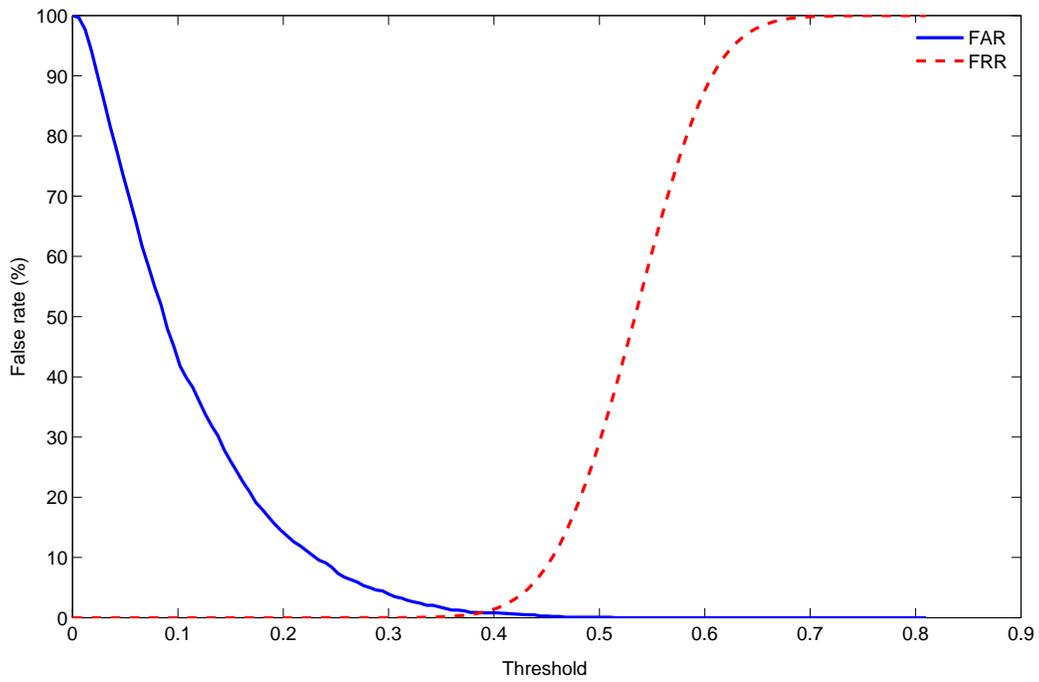


Fig. 6. ROC curve corresponding to 35 DCT coefficients

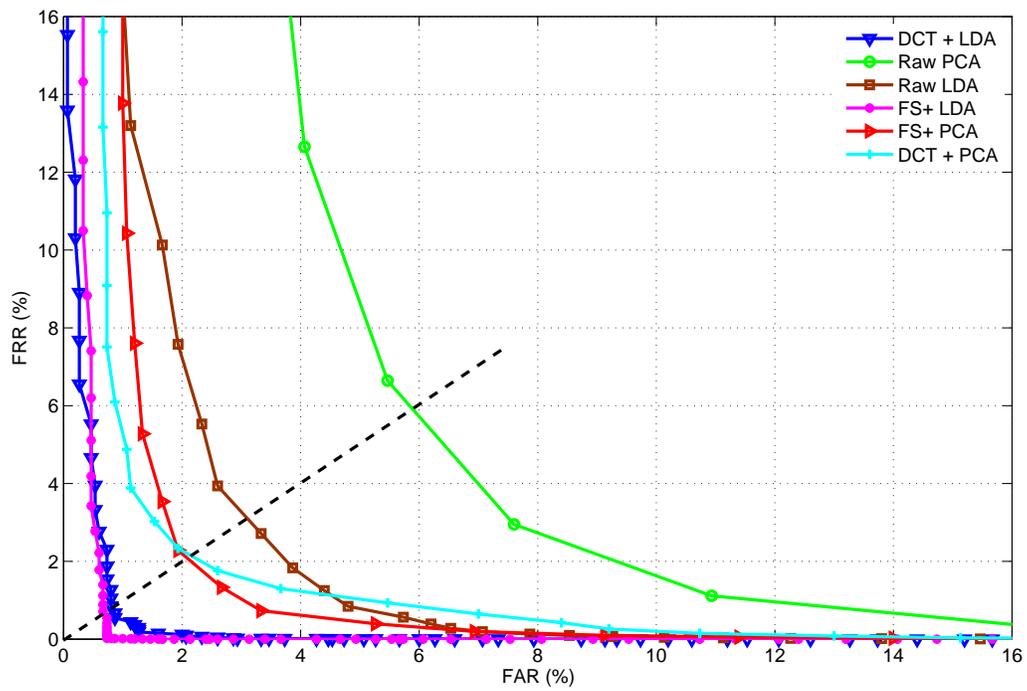


Fig. 7. ROC curves compared similar approaches

Phase Only Based Shoeprint Recognition Techniques

O Nibouche

ECIT Center, Queen's
University of Belfast,
Queen's Rd, Belfast,
BT9 3DT, UK
o.nibouche@qub.ac.uk

A Boudane

ECIT Center, Queen's
University of Belfast,
Queen's Rd, Belfast,
BT9 3DT, UK
a.bouridane@qub.ac.uk

D Crookes

ECIT Center, Queen's
University of Belfast,
Queen's Rd, Belfast,
BT9 3DT, UK
d.crookes@qub.ac.uk

M Gueham

ECIT Center, Queen's
University of Belfast,
Queen's Rd, Belfast,
BT9 3DT, UK
mgueham02@qub.ac.uk

M Laadjel

ECIT Center,
Queen's University of
Belfast, Queen's Rd,
Belfast, BT9 3DT, UK
mlaadjel01@qub.ac.uk

Abstract— In this paper, techniques based on the use of the image phase only are suggested for the problem of shoe print recognition. With the aim of automatically classifying rotated and shifted partial prints, in frequency domain, only the phase component of the transformed images is taken into account while the magnitude component is completely discarded. Two phase only correlation filters are suggested for such task. The designed filters are rendered invariant to rotation within the range -10 to 10 degrees by using rotated versions of the original images in the training phase of the filter design, with a rotation step of 1 degree. For classification, only the phase of the query image is used for correlation. In the resulting correlation plane, classification is based on a Peak to Side lobe Ratio (PSR) metric. Our experiments were carried out on a database of 100 images. Both filters attain high recognition rates that near a perfect 100% rate for the first hit and better the performances of available work in the literature.

I. INTRODUCTION

As a form of physical evidence, a shoemark, which is a mark made when the sole of a shoe comes into contact with a surface, can provide an important link between the criminals and the place where the crime occurred. It has been reported that there should be equal and perhaps even greater chance that footwear impressions could be present at a crime scene, compared with the presence of latent fingerprints [1]. So far, the later has been widely accepted as a powerful tool in forensic applications, however, footwear impressions possess a great potential in playing an assistant role in forensic investigations. As a matter of fact, a study in [2] suggests that footwear impressions could be located and retrieved at approximately 35 per cent of all crime scenes. A shoeprint lifted from a Scene of Crime (SoC) can be checked against a database that includes the shoeprints of shoes in the market to determine its model. It can also be matched against other SoC prints and shoeprints taken from the crime suspects so that a given shoeprint can be identified as being made by a specific shoe. An example of such a database is the commercial database 'SoleMate' maintained by Foster and Freeman Lt [3], which stores more 20,000 shoeprint images from popular footwear maker in UK. The company provides matching services of a SoC print against prints in the company database [3].

Several techniques and algorithms have been reported in the literature for automatic classification, recognition, indexing and retrieval of shoe prints. The authors in [4] used a technique based on Fourier features, invariant moments and neural

networks. Such technique was reported to work well for simple shapes such as circles and triangles. However, it is not well adapted for the more complex shapes in shoe soles [5]. Alexander et al [2,6], developed an approach for the detection and classification of shoeprints based on fractal geometry. Chazal et al. [7] proposed a system for automatically sorting a database of shoeprints based on the outsole patterns in the Fourier domain in response to a reference shoeprint. As shown in [7], the Power Spectral Density (PSD) coefficients of the image are calculated using the Fourier Transform and used as features. A correlation function of the PSD coefficient from a reference database and a query images is used as a similarity metric [7]. Multi resolution based techniques have been used in [8], where images features are extracted from wavelet maxima points. Other techniques employed for shoe print image retrieval and classification are based on extracting local features, which are either points or regions of interests, and using a SIFT like descriptor to describe them [9]. A major issue in automatic classification of shoe prints is that a real scene of crime print can be a noisy, clustered image of poor quality with a textured background. This can lower the performances of the above mentioned techniques and leads to poor extraction of the basic shapes and local features [10]. One solution that can be very robust to such deteriorations of image quality was proposed in [5]. It is based on phase only correlation which captures more discriminative information when compared to amplitude and/or PSD methods. Such technique has been shown to be robust to high level of noise and to textured background [5].

In this paper we show that phase only correlation can be made invariant to rotation within a training range and to partial occlusion, in addition to the well know invariance to translation. We suggest two correlation filters, which in our experiments have achieved a very high recognition rate. The remainder of the paper is organised as follows Section 2 discusses the phase only techniques and introduces the proposed phase only correlation filters trained to be rotation invariant in the range [-10 10] degrees. Results and analysis are presented in section 3. Conclusions are drawn in section 4.

II. PHASE ONLY TECHNIQUES

Work on image reconstruction from its Fourier transform has shown that an image can be fully specified by its phase only under some rather general conditions [11-12]. Even for a 1-D signal, when the signal has minimum or maximum phase, magnitude and phase are related through the Hilbert transform [11-12]. An illustration of the importance of phase in retaining

an image discriminating features is shown in Fig 1. The figure shows an image reconstructed purely from the phase of the original image in Fourier domain. Therefore, image processing techniques that are based on phase only have received a lot of attention [13-18]. In pattern recognition and biometrics, phase based correlation has successfully been adopted for efficient matching of fingerprints that are either very poor in quality or represent partial fingerprints [13-14]. For palm print recognition, two successive Fourier transforms can be used, where the magnitude of the first transform is converted into a log polar representation and the phase of the second Fourier transform is used for registration and correlation [15]. Phase only correlation is coupled with a PCA representation in [16], where a robust illumination-tolerant face recognition is presented. Such technique has led to very high recognition rates [16]. The correlation deals with the fact that PCA on phase on its own is not invariant to translation. However, in this case, PCA itself consists of a representation of images and when combined with phase only correlation, it is the inverse Fourier transform of angle difference which determines the classification result.

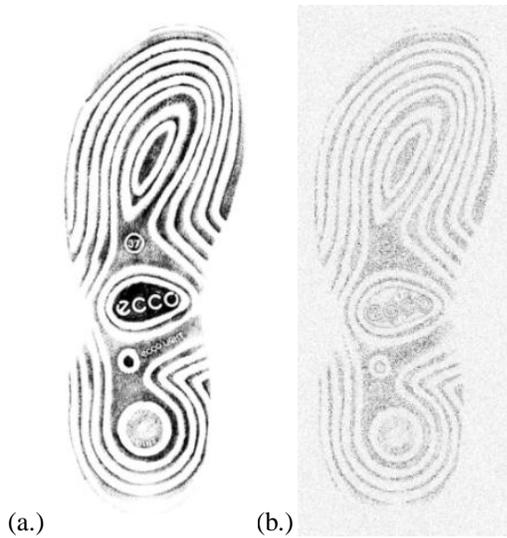


Fig 1. a original image. b. Its reconstruction from only its phase.

Let the Fourier transform of an image $g(x,y)$ be:

$$G(u, v) = \frac{1}{N^2} \sum_0^{N-1} \sum_0^{N-1} g(x, y) e^{-j2\pi \left(\frac{ux+vy}{N} \right)} \quad (1)$$

This can be expressed in terms of phase and magnitude components as:

$$G(u, v) = |M_G(u, v)| e^{j\Phi(u, v)} \quad (2)$$

Let us now consider two images $g_1(x,y)$ and $g_2(x,y)$ with Fourier transforms $G_1(u, v) = |M_{G_1}(u, v)| e^{j\Phi_1(u, v)}$ and $G_2(u, v) = |M_{G_2}(u, v)| e^{j\Phi_2(u, v)}$, respectively.

The phase-only correlation function is given as:

$$\text{cor}(x, y)_{g_1, g_2} = F^{-1} \left\{ \frac{G_1(u, v) G_2(u, v)^*}{|M_{G_1}(u, v)| |M_{G_2}(u, v)|} \right\} = F^{-1} \left\{ e^{j(\Phi_1(u, v) - \Phi_2(u, v))} \right\} \quad (3)$$

It is defined as the inverse Fourier transform of the difference of the two phases. Under the constraint that the maximum result is 1, it is clear (which can be shown by a Lagrange multiplier for instance) that the maximum can only be attained if there is a perfect match between the two phases in equation (3); that is:

$$\max | \text{cor}(x, y)_{g_1, g_2} | = 1 \text{ iff } \Phi_1(u, v) = \Phi_2(u, v) \quad (4)$$

The metric from equation (4), is the peak of the correlation plane. The bottom row of Fig 2 shows the correlation plane of the images in the top row with the original image of Fig 1. Clearly, the better the match, the higher the maximum peak in the correlation plane and the lower are the side lobes. For the wrong match in Fig 2.d, the peak value in the correlation plane is low. Its neighbours' values which form the side lobes are comparable to the peak. However, in the case of the shoeprint in Fig 2.c, which is only a 1/16 fraction of the original image, the meaningful peak is that in the middle of the correlation plane (indicated by an arrow), which is smaller in value than other peaks in the plane. However, its neighbourhood is completely flat. Thus, beyond looking for the peak as a matching metric, one can adopt, the Peak to Side lobe Ratio (PSR) given in equation (5) which is invariant to illumination changes [14].

$$\text{PSR} = \frac{\text{Peak} - \text{Mean}}{\sigma} \quad (5)$$

One can also examine many local correlation peaks and their neighbourhood to look for the best PSR. The mean and standard deviation are computed in the neighbourhood of the peak, as depicted in the outer square of Fig 3. The sizes of the inner and outer squares are determined empirically. In the context of this work the outer square size is 15 pixels.

In the case of multiple images per class, the average of image phases can be used to represent the class. Such approach was adopted in [16] for face recognition, where the average phase was used to replace and simplify the computation of the minimum average correlation energy (MACE) advanced correlation filter. However, in the context of our work, one aims to make the filter invariant to rotation within a certain range, namely [-10 10] degrees and to partial occlusion as it is well known that correlation filters are intrinsically invariant to translation. We only retain the average phase of a given class of images. Such class contains rotated version of the original one, with the rotation angle increased by a constant step within the desired range. In the proposed design, the rotation angle step is of one degree, thus, there are 19 images per class. From the various definition of average phase, we adopt the phase of the average of transformed images, which is weighted by their magnitude. In the remainder of the paper, this correlation filter is referred to as *Filter1*.

Let the non-rotated original image be denoted by $g_0(x,y)$. The index i in equation (6) refers to the rotated image $g_i(x,y)$ obtained by rotating $g_0(x,y)$ by i degrees. *Filter1* is given by:

$$\text{Filter1}(x, y) = F^{-1} \left\{ \frac{\sum_{-9}^9 |M_{G_i}(u, v)| e^{j\Phi_i(u, v)}}{\left| \sum_{-9}^9 |M_{G_i}(u, v)| e^{j\Phi_i(u, v)} \right|} \right\} \quad (6)$$

In addition we defined a modified phase only correlation filter, termed *Filter2* as follows. Let S be a similarity matrix defined as a similarity between individual phases $\Phi_i(u, v)$ of the

transformed images $g_i(x,y)$ and the average phase $\Theta(u,v)$ denoted by:

$$e^{j\Theta(u,v)} = \frac{\sum_{i=-9}^9 |M_{G_i}(u,v)| e^{j\Phi_i(u,v)}}{\left| \sum_{i=-9}^9 M_{G_i}(u,v) e^{j\Phi_i(u,v)} \right|} \quad (7)$$

The matrix S is diagonal of dimension $N^2 \times N^2$ given that the dimension of the training images is $N \times N$. The entry in position $(Nu+v, Nu+v)$ in the matrix S is computed using the elements $\Theta(u,v)$ and $\Phi_i(u,v)$:

$$S(Nu+v, Nu+v) = \frac{\sum_{i=-9}^9 (e^{j\Theta(u,v)} - e^{j\Phi_i(u,v)}) (e^{j\Theta(u,v)} - e^{j\Phi_i(u,v)})^*}{19} \quad (8)$$

One attempts to maximise the peak of the correlation plane, which usually occurs at the origin (however, it shifts with translation) and minimises the similarity matrix S . Let f be the frequency response of *Filter2* arranged in a single column and let g be Fourier transform of *Filter1* also arranged in a single column. Finding the correlation filter which maximises the peak in the origin of *Filter1* and minimising the similarity matrix is equivalent to maximising the quantity:

$$\frac{f^T g g^T f}{f^T S f} \quad (9)$$

By setting the gradient of the above equation with respect to f to zero, the filter f which maximises equation (9) should satisfy:

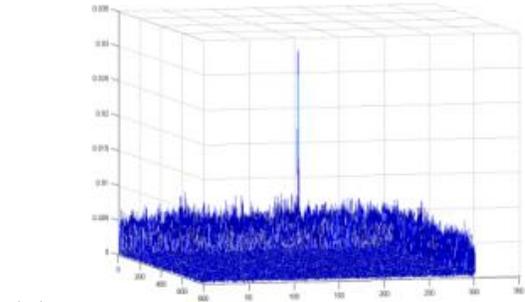
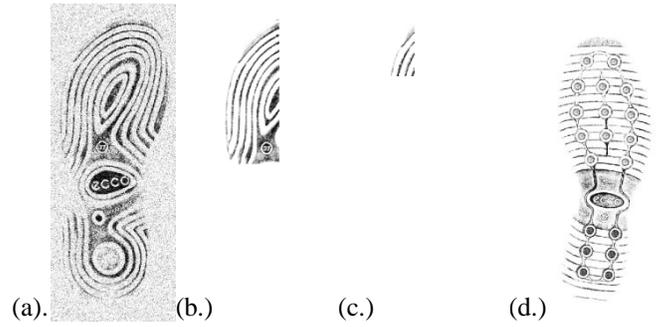
$$g g^T f = \lambda S f \quad (10)$$

Which is a generalised eigenvalue problem [23] as the value of S at the origin $S(0,0)$ is always equal to zero. Nevertheless, in our experiments we set $S(0,0)$ to a constant which can be equal to the minimum of S , to its average or to its maximum with very little impact on the results. Taking account of this modification, the generalised eigenvalue problem of equation (10) is simplified to an eigenvalue problem [23] and its solution f is given by:

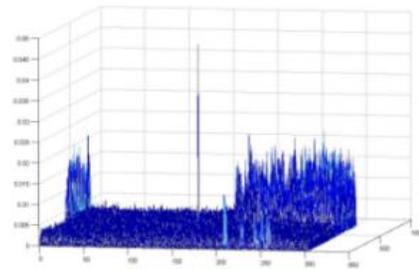
$$f = S^{-1} g \quad (11)$$

III. EXPERIMENTS AND RESULTS

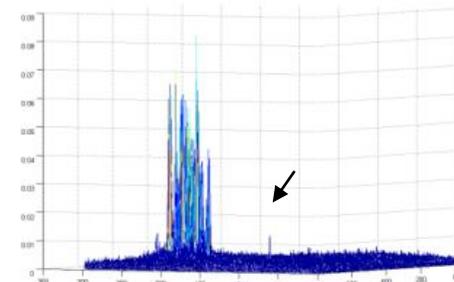
The proposed recognition system is illustrated in Fig 5. The correlation filters are generated from a database of 100 reference images. In addition to storing the generated 100 filters for each of the two proposed correlation filters, we also store their average class filter. As a matter of fact, to reduce the number of selected peaks to compute equation (5), in the recognition phase, the average filter is subtracted from the reference filter before point wise multiplication in Fourier domain can take place. It is then followed by the inverse transform. For our experiments, the query images are generated by randomly selecting a quarter of a reference image and then randomly rotating it by an angle equal to $x + 0.5$ where x is randomly selected integer within the range $[-10, 9]$. For classification, only a single peak, which is the maximum of the correlation plane is taken into account when computing equation (5).



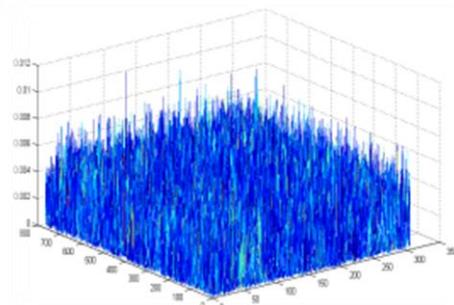
(a.)



(b.)



(c.)



(d.)

Fig 2. Top: Query images to be matched against the original image of Fig 1. Bottom: Result of phase only correlation.

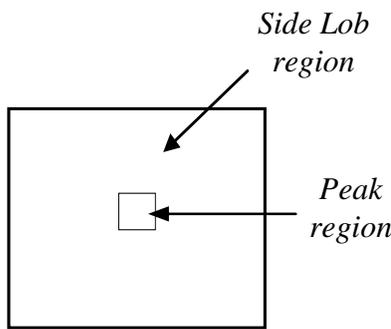


Fig 3. Computation of the PSR

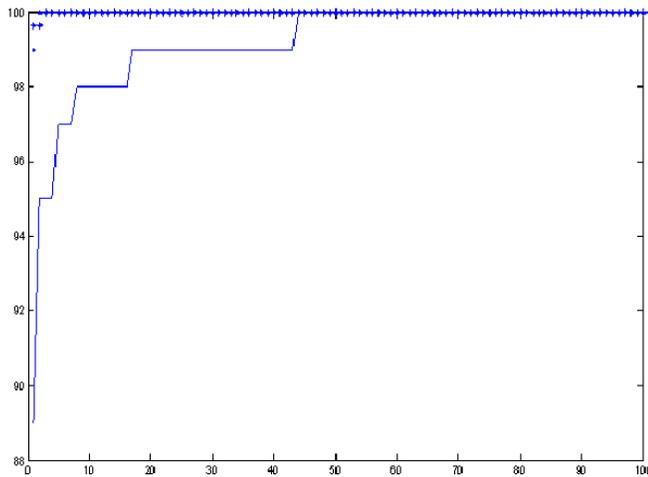


Fig 4. Cumulative match characteristic of the proposed filters and work in [7]. ‘+’ Filter2, ‘^’ Filter1, ‘-’ PSD in [7].

The experiments were run twice, that is for each filter, we attempted to classify 200 query images. The results, as shown in Table I and Fig 4, demonstrate that both filters achieve a high recognition rate, and that *Filter2* attains a 100% classification rate for the top hit. Both filters are superior to available work in the literature [7]. The PSD technique attains less than 90% of top rank classification success.

Table I. Performances comparison

Technique	Top rank	≤ 5	≤ 10	≤ 25	≤ 50
PSD [7]	89.5%	96%	97%	99.5%	100%
<i>Filter1</i>	99.5%	100%	100%	100%	100%
<i>Filter2</i>	100%	100%	100%	100%	100%

IV. CONCLUSIONS

In this paper, two correlation filters are suggested for the classification of partial, shifted and rotated shoe prints. The training step of the two filters using rotated versions of the original image has made the proposed correlation filters invariant to rotation within the training range of [-10 10] degrees. The proposed work achieve better performances than comparable solutions in the literature.

ACKNOWLEDGMENT

The authors gratefully acknowledge the financial support of the EPSRC through grant number EP/C008057/1.

REFERENCES

[1] W.J. Bodziak, “Footwear Impression Evidence – Detection, Recovery and Examination”, CRC Press LLC, 2000.

[2] A. Alexander, A. Bouridane, and D. Crookes, “Automatic Classification and Recognition of Shoeprints” Proc. Seventh Int’l Conf. Image Processing and Its Applications, vol. 2, pp. 638-641, 1999.

[3] Foster and Freeman Ltd, 2006, <http://www.fosterfreeman.co.uk>

[4] Z. Geradts and J. Keijzer, “The image data REBEZO for shoeprint with developments for automatic classification of shoe outsole designs,” Forensic Science Int’l, vol. 82, pp. 21-31, 1996.

[5] M. Gueham, A Bouridane and D Crookes, “Automatic Recognition of Partial Shoeprints Based on Phase-Only Correlation”, in proc. of the IEEE International Conference on Image Processing ICIP 2007, Vol. 4, pp 441-444.

[6] A. Bouridane, A. Alexander, M. Nibouche, and D. Crookes, “Application of Fractals to the Detection and Classification of Shoeprints,” Proc. 2000 Int’l Conf. Image Processing, vol. 1, pp. 474-477, 2000.

[7] P. D. Chazal, J. Flynn, and R.B. Reilly, “Automated processing of shoeprint images based on the Fourier transform for use in forensic science,” IEEE Trans. Pattern Analy. Machine Intell., vol. 27, no. 3, pp. 341-350, Mar. 2005.

[8] L. Ghouti, A. Bouridane, and D. Crookes, “Edge-Directed Invariant Shoeprint Image Retrieval”, proc. of the IET International Conference on Visual Information Engineering, VIE 2006, 26-28 Sept. 2006 pp: 58-61.

[9] H. Su, D. Crookes, A. Bouridane and M. Gueham, “Local Image Features for Shoeprint Image Retrieval”, proc. of the British Machine Vision Conference, BMVC 2007, Warwick, September 10-13, 2007.

[10] H. Su, A. Bouridane, and D. Crookes, “Thresholding of Noisy Shoeprint Images Based on Pixel Context”, Pattern Recognition Letter, Vol. 28 (2), 2007, pp. 301-307.

[11] A.V. Oppenheim and J.S. Lim, “The importance of phase in signals,” IEEE Proc., vol. 69, no. 5, pp. 529-541, 1981.

[12] S.R. Curtis, A.V. Oppenheim, J.S. Lim, Signal reconstruction from Fourier transform sign information, IEEE Trans. Acoust. Speech Signal Process. ASSP-33 (1985) 643–657.

[13] Javed and M.S Hamid, “Improved phase-only correlation method for low quality fingerprint, matching”, Proc. of the 2nd IASTED international conference on Advances in computer science and technology, pp 135-140, Puerto Vallarta, Mexico, 2006.

[14] K Ito and H Nakajima “A fingerprint matching algorithm using phase-only correlation,” IEICE Trans, Fundamentals, Vol. E87-A, No.3, March 2004.

[15] K Ito, T Aoki, H Nakajima, K Kobayashi and T Higuchi, "A palmprint recognition algorithm using phase-only correlation", IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E91-A, No. 4, pp. 1023--1030, April 2008.

[16] M. Savvides, B.V.K. Vijaya Kumar and P.K. Khosla, “Corefaces: robust shift invariant PCA based correlation filter for illumination tolerant face recognition, Proceedings of the 2004 IEEE Computer Vision and Pattern Recognition (CVPR), Vol.2, pp.834-841, July 2004.

[17] A. Moya, David Mendlovic, J. Garcia and C. Ferreira , “Projection-invariant pattern recognition with a phase-only logarithmic-harmonic-derived filter”, Applied Optics, Vol. 35, Issue 20, pp. 3862-3867.

[18] K. Chalasinska-Macukow, F. Turon, M. J. Yzuel, and J.Campos, "Performance of the pure phase-only correlation method for pattern recognition," in Optical Information Processing Systems and Architectures II, pp 262-273 (1990).

[19] P. Belhumeur, J. Hespanha, and D. Kriegman, "Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection", IEEE Transactions on pattern analysis and machine intelligence, 19, pp 711–720, July 1997.

[20] M. Laadjel, A. Bouridane and F. Kurugollu, “Eigenspectra Palmprint Recognition”, DELTA 2008, pp 382-385.

- [21] M. Savvides, B.V.K. Vijaya Kumar and P.K. Khosla, "Eigenphases vs. Eigenfaces", IEEE International Conference Pattern Recognition (ICPR), Vol.3,23-26,pp.810-813, August 2004.
- [22] J. McClellan and T. Parks, "Eigenvalue and eigenvector decomposition of the discrete Fourier Transform", IEEE Trans. On Audio and Electroacoustics, 20:66-74, 1972.
- [23] G.W. Stewart, "Matrix Algorithms, Volume II: Eigensystems", Society for Industrial and Applied Mathematics, 2001.

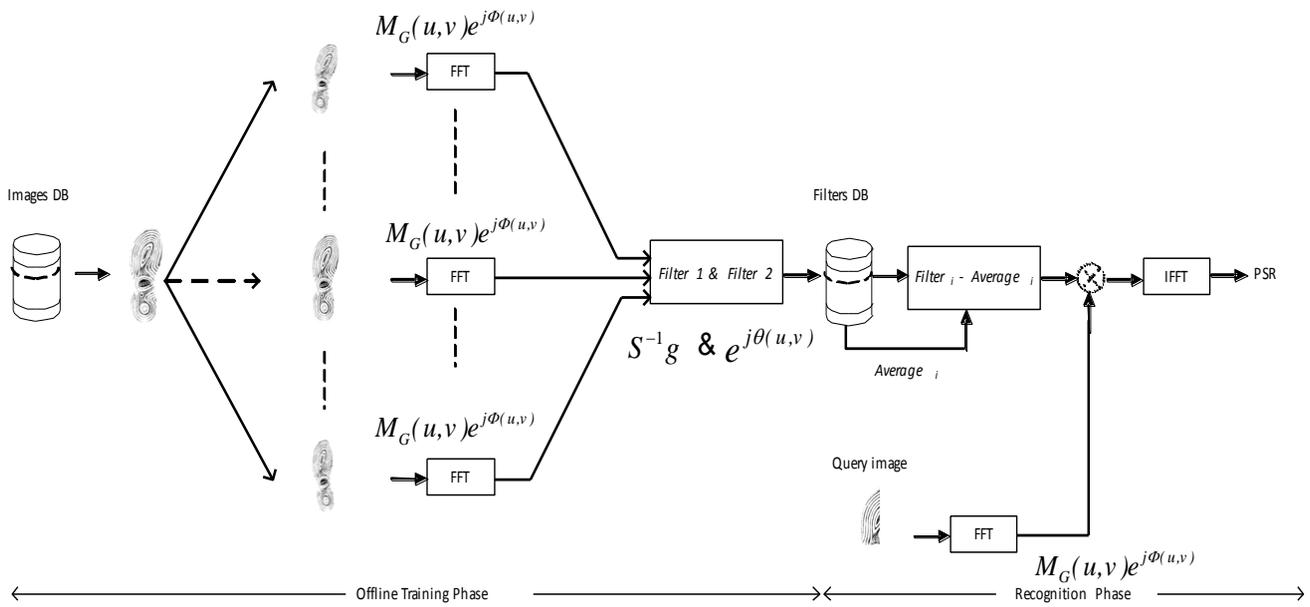


Fig 5. The proposed recognition system

Hardware Implementation of Elliptic Processor Over Standard Binary Curves

Doaa Nassar
Mentor Graphics
Cairo 1134, Egypt
doaa_nassar@mentor.com

M. Watheq El-Kharashi
Computer Dept.
Ain Shams University, Egypt

Abd El Halim Mahmoud Shousha
Electronic & Communication Dept.
Cairo University, Egypt

Abstract— This paper describes a hardware implementation of an elliptic curve cryptography (ECC) over $GF(2^m)$ using FPGA technology. Elliptic curve encryption is becoming increasingly popular as they provide the highest strength per bit of any cryptosystem commonly used today and can be used in wide range of electronic devices, smart cards and hardware security modules. The processor consists of special handling of the point multiplication that is the main operation of any elliptic curve cryptosystem and, hence, it must be implemented efficiently. The new implementation will take the advantage of Xilinx mapping within Precision to produce good area results. The ECC architecture described can also perform the encryption for different binary curves, making it suitable for use in many different ECC applications and environments.

Index Terms—Elliptic Curve Encryption (ECC), FPGA implementation, digital signatures, public-key cryptography, Federal Information Processing Standards

I. INTRODUCTION

Elliptic Curve Cryptography [ECC] is a relatively new cryptosystem, suggested independently in 1986 by Miller [1] and Koblitz [2]. At present, ECC has been commercially accepted, and has also been adopted by many standardizing bodies such as ANSI [3], IEEE [4], ISO [5], SEC [6] and NIST [7]. Currently FPGA are used in the prototyping phase and production versions of electronic systems. FPGA has strong commercial drivers in which it has the ability to update the design and reduce time to market.

A number of hardware implementations for standardized elliptic curve cryptography have been suggested in literature, but very few of them are aimed for low-end devices. A survey of different ECC implementations can be found in [8]. The different ECC processor implementations that have been suggested for such low-end applications [9, 10, 11] normally use non-standardized curves and hence are not acceptable for commercial applications. Standards compliant implementations are however very important for mass acceptance of a reliable public key infrastructure.

For efficient hardware implementation of elliptic curve processor, it is necessary to have an efficient method of multiplying and adding in the underlying field.

Among the most significant hardware architectures for elliptic curves defined over fields $GF(2^m)$ are [12], [13]-[17]. The fastest prototyped $GF(2^m)$ processor was the one designed by Orlando [17] and could compute an arbitrary point multiplication for curves defined over fields $GF(2^{167})$ in 0.21 milliseconds. In this paper, we designed a new $GF(2^m)$ processor using FPGA whose functionality is programmable unlike traditional very large scale integration (VLSI) hardware possessing fixed functionality after fabrication. And it exploits the abilities of reconfigurable hardware to deliver optimized circuitry for different elliptic curves and finite fields.

This paper is organized as follows. Section II briefly described the ECC followed by a description of the point multiplication processor in Section III. The results for the FPGA implementation together with other FPGA implementations comparison are given in Section IV. The paper ends with some conclusions in Section V.

II. ELLIPTIC CURVE CRYPTOSYSTEM

The Federal Information Processing Standards [7] recommends some collection of elliptic curves, which can be used to generate a digital signature. Digital signatures are used to detect unauthorized modifications to data and to authenticate the identity of the signatory. In addition, the recipient of signed data can use a digital signature in proving to a third party that the signatory in fact generated the signature. Choice of private key length and underlying fields are very important for the application being used. These are the different choices used to select the proper curve.

A. Key Lengths

The principal parameters for elliptic curve cryptography are the elliptic curve E and a designated point G on E called the base point. The base point has order r , a large prime. The number of points on the curve is $n = fr$ for some integer f (the cofactor) not divisible by r . For efficiency reasons, it is desirable to take the cofactor to be as small as possible.

The curves used have cofactors 1, 2, or 4. As a result, both the private and public keys are approximately the same length. Each length is chosen to correspond to the cryptovisible length of a common symmetric cryptologic. In each case, the private key length is, at least, approximately twice the symmetric cryptovisible length.

B. Underlying Fields

There are given two kinds of fields for each cryptovariable length. A prime field is the field GF(p) which contains a prime number p of elements. The elements of this field are the integers modulo p, and the field arithmetic is implemented in terms of the arithmetic of integers modulo p. Other is the binary field GF(2^m) which contains 2^m elements for some m (called the degree of the field). The elements of this field are the bit strings of length m, and the field arithmetic is implemented in terms of operations on the bits.

C. Basis

This is to describe how a bit string is to be interpreted. This is referred to as choosing a *basis* for the field. There are two common types of bases: a *polynomial basis* and a *normal basis*. A polynomial basis is specified by an irreducible polynomial modulo 2, called the *field polynomial*. The bit string (a_{m-1} ... a₂ a₁ a₀) is taken to represent the polynomial

$$a_{m-1}t^{m-1} + \dots + a_2t^2 + a_1t + a_0 \tag{1}$$

The field arithmetic is implemented as polynomial arithmetic modulo p(t), where p(t) is the field polynomial. A normal basis is specified by an element θ of a particular kind. The bit string (a₀ a₁ a₂ ... a_{m-1}) is taken to represent the element

$$a_0\theta + a_1\theta^2 + a_2(\theta^2)^2 + a_{m-1}(\theta^2)^{m-1} \tag{2}$$

Normal basis field arithmetic is not easy to describe or efficient to implement in general, but is for a special class called *Type T lowcomplexity* normal bases. For a given field degree m, the choice of T specifies the basis and the field arithmetic.

D. Curves

There are two kinds of curves; one is a *Pseudo-random curve* whose coefficients are generated from the output of a seeded cryptographic hash. If the seed value is given along with the coefficients, it can be verified easily that the coefficients were indeed generated by that method. Other is the *Special curves* whose coefficients and underlying field have been selected to optimize the efficiency of the elliptic curve operations. For each size, the following curves are given:

- 1) A pseudo-random curve over GF(p)
- 2) A pseudo-random curve over GF(2^m)
- 3) A special curve over GF(2^m) called a *Koblitz curve*

The pseudo-random curves are generated via the SHA-1 based method given in the ANSI X9.62 [3] and IEEE P1363 standards [4].

E. Curves over Binary Fields

Elliptic curve domain parameters over GF(2^m) are a septuple: T = (m,f(x),a,b,G,n,h) consisting of an integer m specifying the finite field 2^m, an irreducible binary polynomial f(x) of degree m specifying the polynomial basis representation of F2^m, two elements a,b ∈ GF(2^m) specifying an elliptic curve E defined by the equation:

$$E: y^2 + xy = x^3 + ax^2 + b \tag{3}$$

a base point G=(x_G, y_G), a prime n which is the order of G, and an integer h which is the cofactor h =#E/n. Again following SEC 1 [18], elliptic curve domain parameters over F2^m must have:

$$m \in \{113,131,163,193,233,239,283,409,571\}$$

Elliptic curve domain parameters over GF(2^m) must use the reduction polynomials listed in Table 1 below. This restriction is designed to encourage interoperability while allowing implementers to supply efficient implementations at commonly required security levels.

Table 1. Elliptic curve adopted in the ECC processor

Field	Reduction Polynomial(s)
GF(2 ¹¹³)	$F(x) = x^{113} + x^9 + 1$
GF(2 ¹³¹)	$F(x) = x^{131} + x^8 + x^3 + x^2 + 1$
GF(2 ¹⁶³)	$F(x) = x^{163} + x^7 + x^6 + x^3 + 1$
GF(2 ¹⁹³)	$F(x) = x^{193} + x^{15} + 1$
GF(2 ²³³)	$F(x) = x^{233} + x^{74} + 1$
GF(2 ²³⁹)	$F(x) = x^{239} + x^{36} + 1$
GF(2 ²⁸³)	$F(x) = x^{283} + x^{12} + x^7 + x^5 + 1$
GF(2 ⁴⁰⁹)	$F(x) = x^{409} + x^{87} + 1$
GF(2 ⁵⁷¹)	$F(x) = x^{571} + x^{10} + x^5 + x^2 + 1$

III. ELLIPTIC CURVE PROCESSOR

The GF(2^m) elliptic curve processor is designed to meet the specification of ANSI X9.62 [3] optimized for the use of efficient elliptic curve algorithms, which is also well suited for implementations in reconfigurable hardware. Table1 represents the elliptic curve and curve parameters used in the ECC processor.

The processor includes the Core Unit and Control Unit as shown in Fig 1. Core Unit can perform point multiplication (kP), point addition (p + Q), point doubling (2P) finite-field division operations. Sel signal determines m-bit key size. To implement the above operations, the point multiplication core is developed which consists of GF adder, squarer, multiplier and inverter shown in Fig 2.

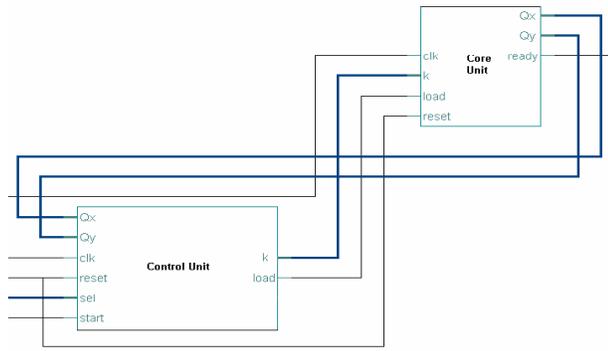


Fig. 1. Architecture of GF(2^m) elliptic curve processor

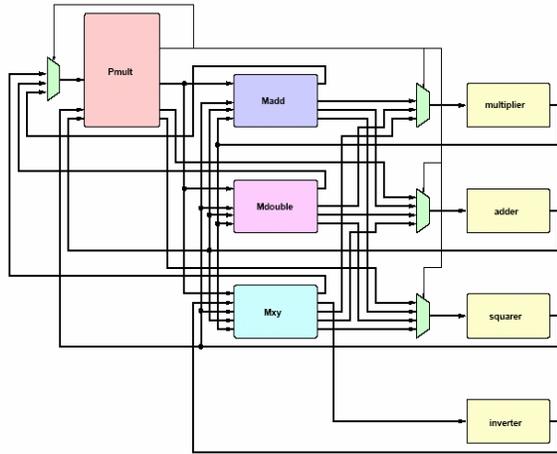


Fig. 2. Architecture of Point multiplication core.

A. Point addition

Let $P(x, y) \in E(F_2^m)$ be a point defined on the curve E , then the computation of point addition can be obtained from the execution of the sequence indicated in (4)

$$\begin{aligned}
 T &= x \\
 p &= X1 \chi Z2 \\
 q &= Z1 \chi X2 \\
 r &= p + q \\
 Z3 &= r^2 \\
 m &= p \chi q \\
 n &= T \chi Z3 \\
 X3 &= m + n
 \end{aligned}
 \tag{4}$$

Thus, the point addition computation consists on 4 multiplications, 2 additions and only one squaring. The multiplier used is based on the binary strategy [11] and the addition is a simple XOR operation executed for final outputs. Field squaring is easy to perform if irreducible polynomials are fixed.

B. Point doubling

The computational complexity of Point doubling (Mdouble) is simpler than the one of point addition. The following

equation (5) is the sequence of instructions needed to compute a single point doubling operation.

$$\begin{aligned}
 T &= c \\
 p &= X^2 \\
 q &= Z^2 \\
 Z2 &= p \chi q \\
 l &= T \chi q \\
 m &= l^2 \\
 n &= p^2 \\
 X2 &= m + n
 \end{aligned}
 \tag{5}$$

C. Multiplication

The multiplication is the critical operation of an ECC implementation thus the multiplication must be implemented carefully. Large Galois fields $GF(2^m)$, are used in ECC implementations which leads to large multipliers. The multiplication $c(x) = a(x)b(x)$ in $GF(2^m)$ consists of two separate operations: an algebraic multiplication of polynomials and a reduction modulo the irreducible polynomial. Coefficients d_i of the result of the algebraic multiplication $d(x) = a(x) b(x)$ are calculated through convolution formula of equation (6)

$$d_i = \sum_{k=0}^i a_k b_{i-k}
 \tag{6}$$

This convolution calculated in blocks called LUT-trees. A LUT-tree consists of three different kinds of blocks: and2xor2-LUTs, xor4-LUTs and 1-bit registers. An and2xor2-LUT is a block that calculates $a_0b_0 + a_1b_1$ in $GF(2)$, i.e. the addition is performed with a 2-bit xor-operation and the multiplication is a 2-bit and-operation. Thus, an and2xor2-LUT can be implemented with one 4-input LUT. A xor4-LUT is a 4-bit xor-operation and, therefore, it calculates $a_0 + b_0 + a_1 + b_1$ in $GF(2)$. Also a xor4-LUT can be implemented with one 4-input LUT. The first level of a LUT-tree consists of and2xor2-LUTs and all the other levels are composed of xor4-LUTs. Part of the LUT tree is illustrated in Fig 3. A complete LUT-tree with k levels calculates the following formula:

$$c = \sum_{j=0}^{T-1} a_j b_j
 \tag{7}$$

where a_j, b_j and c are elements in $GF(2)$, i.e. bits, and T is the length of the inputs of the LUT-tree in bits. The length of the inputs is calculated as follows:

$$T = 2 * 4^{k-1}
 \tag{8}$$

D. Point Multiplication

The elliptic curve point multiplication $Q = kP$, where Q and P are points on the curve and k is an integer, is implemented using the Montgomery point multiplication shown in Fig 4.

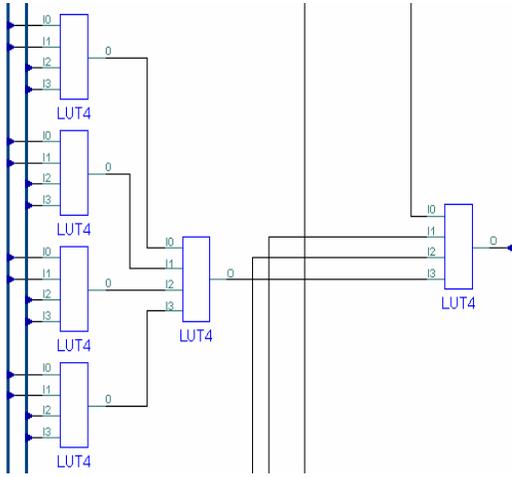


Fig. 3. LUT-Tree

Consecutive point addition and point doubling operations are the heart of the point multiplication algorithm. These operations are implemented as presented in Section 4.2.1 and 4.2.2. The point multiplication is performed in projective coordinates and, therefore, the point $P = (x, y)$ must be mapped from affine coordinates to projective coordinates by setting $X = x$, $Y = y$ and $Z = 1$. This mapping is done for the point P and for a doubled point $2P$ in step 1 of Fig. 4. Because the point multiplication is performed using the Montgomery method, information of the y -coordinate is not needed in the point multiplication and, thus, the mapping has to be done only for the x -coordinates of the points P ($X1, Z1$) and $2P$ ($X2, Z2$). After the Montgomery point multiplication in projective coordinates, affine coordinates of the result point $Q = (x, y)$ are calculated with the Mxy algorithm implemented in [20].

Input: $k = (k_{n-1}, k_{n-2}, \dots, k_1, k_0)_2$ with $k_{n-1} = 1$, $P(x, y) \in E(\mathbb{F}_2^m)$

Output: $Q = kP$

1. Set $X1 \leftarrow x, Z1 \leftarrow 1, X2 \leftarrow x^4 + b, Z2 \leftarrow x^2$
2. For i from $n - 2$ downto 0 do
3. if $(k_i = 1)$ then
4. Madd($X1, Z1, X2, Z2$), Mdouble($X2, Z2$)
5. else
6. Madd($X2, Z2, X1, Z1$), Mdouble($X1, Z1$)
7. Return($Q = M_{xy}(X1, Z1, X2, Z2)$)

Fig. 4. Montgomery point multiplication Algorithm

IV. RESULTS

The proposed elliptic curve point multiplication processor is implemented using VHDL and synthesized using Precision® RTL which is the Mentor Graphics synthesis tool [20] and implemented on Xilinx Virtex-E device using 4-input LUTs. [21]. A set of recommended elliptic curves for cryptography given by Standards for Efficient Cryptography Group (SECG)

have been implemented using VHDL. The synthesis was performed with Precision and the place & route done using Xilinx ISE 9.2. Virtex-E XCV2600E-8 device is chosen from the Virtex-E family. Virtex-E was chosen because it is the most commonly used device family in other published ECC implementations and, thus, the comparison presented is easier to perform. The implementation results on Virtex-E are presented in Table 2. Clock is the achieved clock frequency after the implementation process. As can be seen the point multiplication can be performed very efficiently on elliptic curves over relatively small Galois fields ($m < 200$) but, when the field sizes grow, the results become poorer. The main reason for the poorer results is the area requirements, which grow near to the limits of the target device. Thus, the place & route process cannot be performed as efficiently as in the case of smaller implementations. This leads to smaller clock frequencies, which further causes longer point multiplication times. When small implementations are considered, it can be said that it is worthwhile to use two multipliers with small latency in order to achieve short point multiplication times. However, if the area requirements of the design are close to the limits of the target device, the choice of the number and latency of the multipliers is not as straightforward. Curves using the largest fields, for which $m = \{409, 571\}$, do not fit into Virtex-E XCV2600E at all. These curves must be implemented on a larger target device.

Table 2. Implementation results of the elliptic processor

Elliptic Curve	m	Slices	Clock Mhz
GF(2^{113})	113	8500	98.2
GF(2^{131})	131	10900	106.1
GF(2^{163})	163	14305	99.6
GF(2^{193})	193	18970	80.5
GF(2^{233})	233	25234	55.3
GF(2^{239})	239	19433	65.9
GF(2^{283})	283	23851	51.2
GF(2^{409})	409	28974	N/A
GF(2^{571})	571	45583	N/A

Another experiment had been done targeting Xilinx Virtex5 synthesizing the VHDL using Precision capabilities to support Virtex-5 family technology that is based on 6-input LUT architectures. The Virtex-5 family is the first FPGA platform to offer a real 6-input LUT with fully independent (not shared) inputs. This leads to some very compelling advantages. The 6-input LUT leads to several benefits, as it implements wider functions directly in the LUT. Hence, the number of logic levels between registers is reduced, leading to higher performance. It also implements significantly more logic than a LUT with four inputs, which reduces the amount of, required interconnect and routing resources. That makes Virtex5 is more suitable for application such as ECC especially that Precision Synthesis [20] offers some advanced synthesis

capabilities during the mapping and optimization targeting Virtex5, which lead to these good results.

FPGA-based designs implemented on Xilinx devices are collected into Table 3. Results show that for Virtex5, our implementation gives the smallest area (43% improvement in number of slices) for $m=193$ together with approximately the same frequency compared to SIG-ECPM [19] which we share the same architecture.

Table 3. FPGA Based Implementations

Design	Device Family	m	Slices	Clock (Mhz)
SIG-ECPM [19]	VirtexII	113	10686	108.3
		163	18079	90.2
		193	19250	90.2
		233	23020	73.6
Our-impl	Virtex5	193	10977	91.9
Bednara [22]	Virtex	191	N.A	50
Gura [23][24]	VirtexE	163	N.A	66.4
		193	N.A	66.4
		233	N.A	66.4
Eberle [23]	VirtexII	163	N.A	66.4
		193	N.A	66.4
		233	N.A	66.4
Nguyen [25]	VirtexII	233	13180	N.A
		Orlando [17]	VirtexE	167

V. CONCLUSION

We have presented the high-performance reconfigurable elliptic curve processor for $GF(2^m)$. The architecture of the proposed processor is based on the Galois Field of $GF(2^m)$ and is configurable for the binary elliptic curves standards. The design was successfully tested on a Xilinx VirtexE and Virtex5. The processor implement the Galois field operations, i.e. multiplication, squaring, addition and possibly inversion in $GF(2^m)$. Future work to be addressed includes further improvements in the performance of the algorithms, especially by reducing the critical path trying to balance between required area and speed.

REFERENCES

- [1] V. S. Miller. Use of elliptic curves in cryptography. In H. C. Williams, editor, *Advances in cryptology CRYPTO '85*, volume 218 of LNCS, pages 417-426, Berlin, Germany, August 1986. Springer-Verlag Inc.
- [2] N. Koblitz. *Elliptic Curve Cryptosystems*. Mathematics of Computation, 48(177), 203-209, January 1987.
- [3] American National Standards Institute, New York, USA. ANSI X9.62: Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), 1999.
- [4] IEEE Computer Society Press, Silver Spring, MD, USA. IEEE P1363-2000: IEEE Standard Specifications for Public-Key Cryptography.
- [5] International Organization for Standardization, Geneva, Switzerland. ISO/IEC15946: Information Technology | Security Techniques: Cryptographic Techniques based on Elliptic Curves, 2002.
- [6] SEC 2. Standards for Efficient Cryptography Group: Recommended Elliptic Curve Domain Parameters. Version 1.0, 2000.
- [7] National Institute for Standards and Technology, Gaithersburg, MD, USA. FIPS 186-2: Digital Signature Standard (DSS). 186-2, February 2000. Available for download at <http://csrc.nist.gov/encryption>.
- [8] L. Batina, S. B. Ors, B. Prenee, and J. Vandewalle. Hardware architectures for public key cryptography. *Integration, the VLSI journal*, 2003.
- [9] R. Schroepfel, C. L. Beaver, R. Gonzales, R. Miller, and T. Draelos. A low-power design for an elliptic curve digital signature chip. In B. S. Kaliski Jr., C. K. Koc, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems | CHES 2002*, volume 2523 of LNCS, pages 366-380, Berlin, Germany, August 2002. Springer-Verlag Inc.
- [10] E. A. Öztürk, B. Sunar, and E. Savas. Low-power elliptic curve cryptography using scaled modular arithmetic. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems*, pages 92-106, Berlin, Germany, August 2004. Springer-Verlag Inc.
- [11] J. W. Chung, S. G. Sim, and P. J. Lee. Fast Implementation of Elliptic Curve Defined over $GF(p)$ on CalmRISC with MAC2424 Coprocessor. *Cryptographic Hardware and Embedded Systems*, pages 57-70, Berlin, Germany, August 2000. Springer-Verlag Inc.
- [12] G. B. Agnew, R.C. Mullin, and S. A. Vanstone. "An implementation of elliptic curve cryptosystems over $F_{2^{155}}$ ", *IEEE Trans. Select. Areas Commun.*, vol. 11, pp. 804-813, 1993.
- [13] M. Rosner. *Elliptic curve cryptosystems on reconfigurable hardware*. Master's thesis, ECE Dept., Worcester Polytechnic Institute, Worcester, USA, May 1998.
- [14] L. Gao, S. Shrivastava, and G. Sobelman. "Elliptic curve scalar multiplier design using FPGAs". In *Cryptographic Hardware and Embedded Systems -CHES '99 (LNCS 1717)*, pages 257-268. Springer-Verlag, 1999.
- [15] S. Sutikno, R. Effendi, and A. Surya. "Design and implementation of arithmetic processor $F_{2^{155}}$ for elliptic curve cryptosystems". In *The 1998 IEEE Asia-Pacific Conference on Circuits and Systems*, pages 647-650, 1998.
- [16] K.H. Leung, K.W. Ma, W.K. Wong, and P.H.W. Leong. "FPGA implementation of a microcoded elliptic curve cryptographic processor". In *Eight Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM '00*, Napa Valley, California, USA, 2000.
- [17] G. Orlando and C. Paar. "A high performance elliptic curve processor for $GF(2^m)$ ". In *Cryptographic Hardware and Embedded Systems - CHES '00 (LNCS 1965)*, pages 41-56. Springer-Verlag, 2000.
- [18] Certicom Research. SEC 1: Elliptic Curve Cryptography. *Standards for Efficient Cryptography*, September 20, 2000. Version 1.0, internet: www.secg.org/collateral/sec1_final.pdf, (visited November 14, 2003).
- [19] K. Järvinen, M. Tommiska and J. Skyttä. "A Scalable Architecture for Elliptic Curve Point Multiplication", proceedings of the 2004 IEEE International Conference on Field-Programmable Technology, FPT, pages 303-306. 2004
- [20] Precision® RTL Synthesis User's Manual, Release 2006a1, Feb. 2007.
- [21] Xilinx, Inc. Virtex-II, Virtex5 Platform FPGAs: Data Sheet. Oct.2007.
- [22] M. Bednara, M. Daldrup, J. von zur Gathen, J. Shokrollahi, and J. Teich. "Reconfigurable Implementation of Elliptic Curve Crypto Algorithms," IPDPS'02, Fort Lauderdale, FL, USA, pp 157-164. Apr. 15-19.2002
- [23] H. Eberle, N. Gura. and S. Chang-Shantz. "A Cryptographic Processor for Arbitrary Elliptic Curves over $GF(2^m)$," *Int. Conf on Application Specific Systems, Architectures, and Processors, ASAP'03*, The Hague, The Netherlands, pp W 5 4, Jun.24-26,2003.
- [24] N. Gura. H. Eberle. and S. C. Shantz. "Generic Implementations of Elliptic Curve Cryptography using Partial Reduction," *CCS'02*, Washington, DC, USA, ACM Press, pp 108-116, Nov. 18-22. 2002.
- [25] N. Nguyen. K. Gai. D. Caliga, and T. El-Ghazawi. "Implementation of Elliptic Curve Cryptosystems on a Reconfigurable Computer," *FPT compelling advantages*. 2003, Japan, pp 6 M 7 , Dec. 15-17, 2003.

Design and Implementation of a High Quality and High Throughput TRNG in FPGA

Cristian KLEIN¹

Technical University of Cluj-Napoca
E-mail: cristi@net.utcluj.ro

Octavian CRET²

Technical University of Cluj-Napoca
E-mail: Octavian.Cret@cs.utcluj.ro

Alin SUCIU²

Technical University of Cluj-Napoca
E-mail: Alin.Suciu@cs.utcluj.ro

Abstract—This paper focuses on the design and implementation of a high-quality and high-throughput true-random number generator (TRNG) in FPGA. Various practical issues which we encountered are highlighted and the influence of the various parameters on the functioning of the TRNG are discussed. We also propose a few values for the parameters which use the minimum amount of the resources but still pass common random number generator test batteries such as DieHard and TestU01.

I. INTRODUCTION

Random numbers are at the very core of cryptographic algorithms. They are used to generate either the public / private key pair in asymmetric algorithms, or the shared secret / initialisation vector in symmetric cyphers. The ability of an adversary to predict the random numbers used, voids the security of the cypher. In fact, the only cypher whose security is proven to be perfect (one time pad) relies on the fact that the random number source is perfectly uniform and unpredictable.

Random number generators are of two types. The first one, *pseudo-random number generators* (PRNG), are the ones in which a person who designed the system, or has access to its internal state can predict the next random number. The system is a deterministic Finite State Machine, whose evolution can usually be described based on an arithmetic formula which determines its transition from a given internal state to another state, while outputting a random number based on a portion of the state. They have the advantage of having high speeds and some of them are cryptographically secure. However, all of them require an initial state (also called *seed*), which determines the sequence of numbers which will be generated. The importance of well seeding a pseudo-random number generator has recently been highlighted in a Debian security vulnerability[1].

The second type of random numbers generators are *true-random number generators* (TRNG), whose output cannot be predicted, not even by the person who designed them. They are usually based on sampling some kind of physical phenomenon (such as noise) which has a lot of randomness. Although one would be tempted to use only TRNGs in cryptography, their smaller throughput prohibits this, so they are commonly used to seed PRNGs.

FPGAs are becoming a popular choice for implementing cryptographic devices, due to the fact they represent the middle ground between the flexibility of the microprocessor and the speed of an ASIC. They allow creating high-throughput cryptographic devices while at the same time making it possible to change or improve the underlying algorithms, should a security flaw be discovered.

Many papers [3][6][7][8][9] have explored the possibility of implementing TRNGs in FPGAs, motivated by the avoidance of additional hardware, and the impossibility to intercept the data stream between the TRNG and the actual cryptographic implementation. While all of them claim to obtain good-quality TRNG, few mention explicitly the methods involved in transforming a hardware which is supposed to work predictably into a source of entropy.

This paper elaborates on the design and implementation of the TRNG principle presented by Martin and Stinson in [6] and highlights a few practical issues encountered while implementing a high-quality TRNG based on it. We identified a few generic parameters, whose influence on the TRNG will also be presented in this paper. The ultimate purpose is to enable the reader to easily implement this TRNG on a low-cost FPGA development board, such as one featuring a Xilinx Spartan 3E.

II. PRINCIPLE

Like many TRNG implemented in FPGA, this design is based on sampling jitter. In essence, due to various noise sources such as that induced by the power supply but also by nearby components, the behaviour of „demanding a 0 or 1” from the transition slope of an output is unpredictable. This is caused by the fact that each technology defines a low (L) threshold, which is the upper limit for voltages which represent a logic 0, and a high (H) threshold which is the lower limit of logic 1. Output behaviour between these two values is not well defined. This can be modelled as if the output of the component would have a perfectly vertical slope, but the time of the transition is unknown and can range from the beginning until the end of the real slope (figure 1).

In order to produce jitter, TRNGs employ one or more ring oscillators (RO) (figure 2). These are composed of a ring of odd number of inverting elements and an arbitrary number of delaying elements. The simplest RO is composed of a single inverter and a buffer. The output of a RO is never

¹Financiar support from INRIA is hereby greatly acknowledged.

²This work was supported by the CNMP funded CryptoRand project, nr. 11-020/2007.

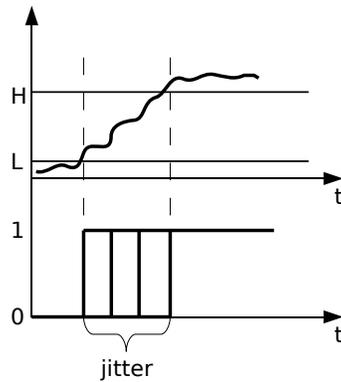


Figure 1. The Jitter Model

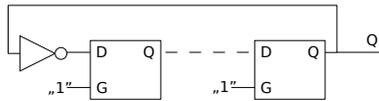


Figure 2. Ring Oscillator (RO)

stable and does transitions from 0 to 1 and back to 0 with a frequency given by the propagation delay of the constituting elements. Due to the above described phenomena, the period of an oscillation will not be constant, because it will vary by a small amount each time. This is the manifestation of jitter and the source of entropy which our TRNG will collect.

Our first attempt was to create a TRNG based on [3], which uses one RO to sample the output of the other RO. We appreciated this approach due to the fact that the whole stream before the post-processing phase is random (although it might be biased a little bit). We also favoured this design, because, if there is some kind of predictable jitter (such as coming from the power source), both ROs are influenced the same way which should cancel out at the sampler. However, we found that putting this design into practice is very challenging. The two ROs have to be nearly identical, which requires manual placing and routing. Even after achieving that, the design proved to be very sensitive to other components in the FPGA. At the time of this writing we have been able to create a TRNG which outputs very good numbers on a serial interface, but have been unable to obtain good quality random numbers at the TRNG's highest speed.

Therefore, we chose to implement [6] which uses multiple ROs whose outputs are XOR-ed. A flip-flop whose clock is driven by a fixed frequency will sample the combined output of the ROs. The obtained stream will hit both jitter zones (our source of entropy) and flat zones (which are highly predictable). A post-processing phase is required which consists in a resilience function[2]. In essence, the function takes an m -bit input, out of which n -bits are known to be random (but we can't determine which ones) and outputs n -bits which are known to be random. For $n = 1$, the simplest resilience function is to xor all the input bits. Suppose all but one bits are deterministic, but the probability of a 0 or 1 value of one bit are equal, the output of the xor will also have equal

probability of being 0 or 1.

III. IMPLEMENTATION ISSUES

A. Creating ROs in VHDL

Our first goal was to create a VHDL component which would implement a ring oscillator with a parametrised length.

We first studied what resources are available in the FPGA to create ring oscillators. The main building block of the FPGA, the CLB are the only ones that actually contain logic, and are interconnected by a network of routing wires. The CLB contains a LUT, an inverter and a memory element which can be either used as a latch or as a flip-flop. The output of latch / FF goes directly out of the CLB into the interconnection network. Two CLBs are grouped together in a slice, however in order to connect the output of one latch / FF to the other CLB in the same slice, the wire has to exit the slice, go through the interconnection network and reenter the CLB. From Xilinx's reports we noticed that the main delays in FPGA come from latches and routing. The inverter induces a negligible delay. Another interesting thing we noticed is that during the mapping phase, a GLOBAL_LOGIC1 signal is created which provides logic „1” for all the CLBs that require it.

Having the knowledge above, we chose to have a single inverter at the beginning of the chain and a variable number of latches as delay components (like in figure 2). A single inverter allows us to create ROs which both even and odd number of latches. By default Xilinx's synthesis tool optimises out all but one latch, due to the fact that they seem redundant from its viewpoint. In order to prevent this, we must set the „keep” attribute[10] of the `d` bus which interconnects the latches:

```
attribute keep : string;
attribute keep of d : signal is "true";
```

This tells the synthesis and mapping tool that we want the individual `d` signals not to be absorbed into a CLB. Each of them must pass through the interconnection network, which forces the tools to map the redundant latches to CLBs.

To make sure that the inverter does not add more delay, we added the `not` keyword directly into the port map of the first latch, without assigning it a signal. This has the effect that the inverter and the first latch are mapped to the same CLB.

B. Sampler

We chose to give the whole TRNG circuit the same interface as the one used by [3], to which we added an input clock signal (figure 3). The `BitReady` output signal is high when the TRNG has a new random bit, which will appear at the `RandomBit` pin. When the external circuit has stored the random bit, it will acknowledge the TRNG by rising the `ReadAck` pin.

Although our particular TRNG is synchronous, all three signals are assumed to be asynchronous, both inside the TRNG and the external circuitry that connects to it. We took this decision for two purposes: first, we wanted to be able to use a RO's output as the sampling clock, which would make the

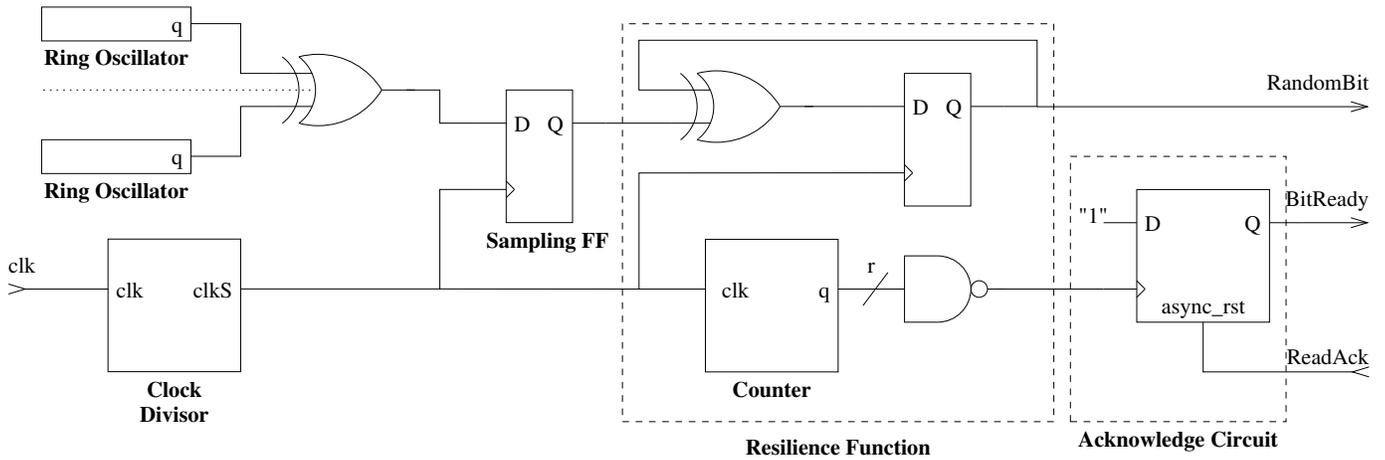


Figure 3. TRNG Scheme

TRNG truly asynchronous, and secondly, we wanted to use the very same design to test future TRNG, which might be asynchronous in nature.

C. Resilience Function

Contrary to the design employed by others, we chose as the resilience function a simple XOR of 2^r -bits (where r is a generic parameter). We did this because we feared that using a more complex resilience function may hide possible defects in our TRNG, which we obviously want to avoid. Moreover, some resilience functions (such as cyclic codes) are implemented using shift registers and XORs which might act as PRNG. We specifically want to test how well the TRNG works with minimal post-processing. Using the TRNG to seed a PRNG (although a possibly weak one) is against the purpose of our paper.

D. High-throughput Measurements

It was very important for us to validate the TRNG at its maximum speed. We feared that the output interface from the FPGA to the computer (where the random bits are collected and analysed), whether RS232 or USB, would do additional sampling of the (possible partially) random stream. This would return more optimistic results compared to the TRNG being used only inside the FPGA.

In order to achieve this, we created a design which would first fill a 16 Kbit BlockRAM with TRNG output, then transfer this to the output interface (figure 4). We think that this is very close to how a TRNG would be used in a FPGA cryptographic application: the cypher gets values from the entropy buffer and while the algorithm proceeds, the TRNG fills back the entropy buffer.

The design is able to handle burst transfers from the TRNG. The data-in port of the RAM is directly connected to the TRNGs output. The control signals of the address counter and the write-enable port of the RAM are directly connected to the `BitReady` port of the TRNG, provided the FSM is in the `FillRAM` state. A separate circuit is used to drive the `ReadAck` port of the TRNG which sets it to 1 at the very

next clock rising edge, exactly when the RAM has stored the random bit.

The FSM which controls this circuit has 8 states (figure 5). The first, `Idle` is the state in which the FSM is set immediately after reset. Transition is made immediately to the `PrepareFillRAM` state, which resets the address counter. Next, the `FillRAM` state allows the counter to increase and the RAM to store values when a new random bit is ready. The FSM stays in this state until the RAM is filled (i.e. the RAM address counter wraps around). The next three states (`ReadRAM`, `ShiftIn`, `CheckSR`) serialise the bits stored in the RAM into a byte for being transmitted to the UART module. The same counter is used to control the address of the RAM, but it is only incremented in the `ShiftIn` state. Finally, when a byte is complete (i.e. the `Serialiser` sets the `ready` port to 1) the FSM will wait for the UART to complete the previous transmission (`WaitUART`), then dispatch the data (`UARTSend`). If there is more data to transmit (i.e. RAM address counter is non-zero) then the FSM will transition to the `ReadRAM` state, serialising the next byte. If the whole contents of the RAM has been transmitted, it will be freshly filled with random numbers, by jumping to the `PrepareFillRAM` state.

IV. TUNING THE TRNG

A very important practical aspect of the TRNG is to know the influence of its generic parameters on the quality of its output. We also wanted to test practically what is the smallest number of FPGA resources which are required for this TRNG. We used the `DieHard`[5] and the `TestU01`[4] (NIST, Rabbit and Alphabit battery) suites to test the quality of the TRNG output. We only considered parameters for which the output of the TRNG passed all tests, i.e. all `DieHard` p-values are different from 0 or 1, and `TestU01` prints „All tests were passed“. All files which we downloaded had at least 10 MB, due to limitations in `DieHard`. Interestingly, the `TestU01` library proved to be a lot more sensitive than `DieHard`.

The proposed TRNG has the following generic parameters:

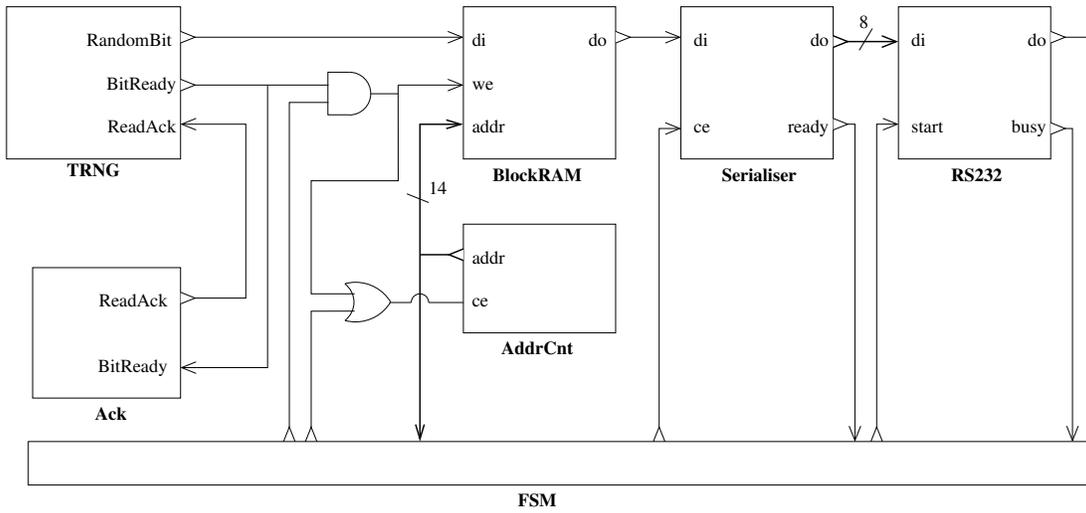


Figure 4. High-throughput Measurement Scheme

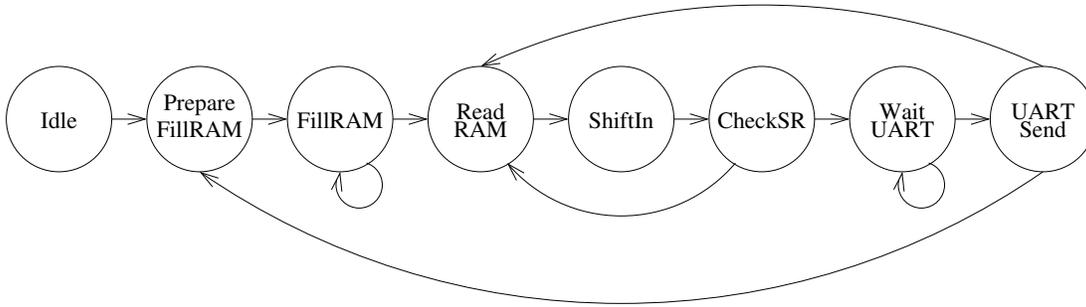


Figure 5. FSM State Diagram

number of ring oscillators (n), length of ring oscillators (l), sampling frequency divisor (2^d) and resilience function input width (2^r).

The first two aspects in which we were interested is the throughput and the amount of resources this design uses. The throughput can be easily computed as the output rate of the TRNG is the input clock frequency, divided first by the clock divider, then the resilience function. The formula is:

$$b = \frac{f}{2^r * 2^d} \quad (1)$$

where f is the input clock frequency of the TRNG and b is the throughput in bps.

The amount of resources can also be easily estimated. Each RO uses l CLBs. The xor stage is synthesised as a tree of LUTs. Due to the fact that the slice of a Spartan 3E device contains 4-bit input LUTs the number of CLBs is $\lceil \frac{n-1}{3} \rceil$. The clock divisor uses approximately $\frac{d}{4}$ CLBs. The counter uses about $\frac{r}{4}$ CLBs, while the and stage at its output uses $\lceil \frac{r-1}{3} \rceil$. The other components (sampler FF, resilience XOR and FF, acknowledge circuit) use 3 CLBs. Therefore the total number of used CLBs (C) is:

$$C = l + \left\lceil \frac{n-1}{3} \right\rceil + \frac{d}{4} + \frac{r}{4} + \left\lceil \frac{r-1}{3} \right\rceil + 3 \quad (2)$$

During our experiments we concluded that the quality of the output random bit stream increases with the increase of d , r and n . As the number of ring oscillators (n) increases and because the ring oscillators don't have the exactly same frequency, the signal after xoring them will be composed of much more jitter than flat zones. This means that the sampler will return much more non-deterministic bits compared to the amount of deterministic bits. The more input bits the resilience function has the more non-deterministic bits will be xored with the deterministic bits, which in effect will increase the chance of the TRNG to output a truly random bit.

Regarding the clock divider, if d is too small (even comparable to the frequency of the ring oscillators), the sampler tends to hit the same flat zone or return the same non-deterministic bit several times. The resulting correlated bits can of course be eliminated in the resilience stage, provided that r is large enough. We can clearly see that the well-known throughput vs. resources conflict also holds in case of this TRNG.

We haven't found any significant influence of l on the quality of the random numbers. This might be due to the fact that while each delay element increases the output period of the ring oscillators, it also increases the amount of jitter, so the percentage of the jitter after the sampling stage remains roughly the same. Although one is tempted to use ring oscil-

Table I
PARAMETERS FOR HIGH QUALITY TRNG

d	r	n	l	throughput (Kbps)
0	2	20	3	12500
0	3	10	3	6250
2	2	10	3	3125
5	3	5	3	195

lators with the minimum length, we recommend to use $l \geq 3$ to make sure that the system does not remain without jitter in extreme conditions such as sudden temperature variations.

In our experiments the parameters values presented in table I created a TRNG which passed all tests, while minimizing the number of ring oscillators. Please note that in case one wants to be absolutely sure that the TRNG will output high-quality random numbers, higher values should be used for r or, if bandwidth is an issue, n .

V. SPEEDING UP THE TRNG

FPGAs are becoming large enough to allow massive pipelining of arithmetic operands and compute one result per clock. In some applications it might be desirable to generate random numbers at the maximum frequency of the FPGA. In the above design, both the resilience function (characterised by r) and the sample clock divider (d) lower the frequency of the TRNG. While we could set d to zero, so that the sampling clock is set to maximum, we can never set r to zero, while at the same time obtain good quality random numbers.

First solution which would come to one's mind is to use multiple parallel TRNGs and multiplex their outputs. Suppose the sample clock divider is equal to zero, each TRNG would output one bit each 2^r cycles. This means that we would need 2^r TRNGs for generating one random bit on each FPGA clock. While this solution would surely work (due to the fact that by interleaving truly random streams one obtains another truly random stream), we wanted to find a design that would minimise the resource utilisation.

Our idea is that we require the resilience function because not all our bits are sampled from jitter. The same would apply if we would XOR bits coming from different samplers. This way, we would save 2^r counters, FFs and AND gate and replace them with one big XOR.

Indeed, we have practically validated the fact that good quality random numbers are generated using the above concept, for 8 samplers and 20 ROs / sampler. Interestingly, the number of samplers required is equal to the number of bits which enters the resilience function in the design presented in figure 3.

Note however, what for the mentioned values, we used 160 ROs, eight times more. An interesting question is whether this number of ROs could be used to generate a random bit stream, without using a resilience function ($d = 0$ and $r = 0$ in figure 3). We have practically shown that this is not possible, as explained in [7]. In essence, the probability of sampling a random bit increases with the number of ROs, but never reaches 1. The small percent of the resulting correlated bits is enough to make the TRNG fail quality tests.

VI. CONCLUSION

In this paper we have shown how a simple yet of high-quality and high-throughput TRNG can be implemented on a low-end Xilinx Spartan 3E FPGA and presented the main implementation issues one might encounter. We have also discussed the various parameters of the TRNG and the influence they have on the design. We believe that this paper has paved the way to implementing secure cryptographic applications in low-end FPGAs, without requiring any external component.

REFERENCES

- [1] Debian Project. DSA-1571-1 openssl – predictable random number generator. Available Online: <http://www.debian.org/security/2008/dsa-1571>.
- [2] Gopalakrishnan and Stinson. Applications of designs to cryptography. In *Charles J. Colbourn and Jeffrey H. Dinitz (Eds.), The CRC Handbook of Combinatorial Designs*, CRC Press, 1996. Available Online: <http://citeseer.ist.psu.edu/126555.html>.
- [3] P. Kohlbrenner and K. Gaj. An embedded true random number generator for FPGAs. In *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 71–78, New York, NY, USA, 2004. ACM.
- [4] P. L'Ecuyer and R. Simard. TestU01: A C library for empirical testing of random number generators. *ACM Trans. Math. Softw.*, 33(4):22, 2007.
- [5] G. Marsaglia. Diehard: Battery of tests of randomness. Available Online: <http://www.stat.fsu.edu/pub/diehard/>.
- [6] W. J. Martin and D. R. Stinson. A provably secure true random number generator with built-in tolerance to active attacks. *IEEE Trans. Comput.*, 56(1):109–119, 2007. Member-Berk Sunar.
- [7] D. Schellekens, B. Preneel, and I. Verbauwhede. FPGA vendor agnostic true random number generator. In *FPL*, pages 1–6. IEEE, 2006.
- [8] K. H. Tsoi, K. H. Leung, and P. H. W. Leong. Compact fpga-based true and pseudo random number generators. In *FCCM '03: Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, page 51, Washington, DC, USA, 2003. IEEE Computer Society.
- [9] M. Šimka, M. Drutarovský, and V. Fischer. Embedded True Random Number Generator in Actel FPGAs. In *Workshop on Cryptographic Advances in Secure Hardware – CRASH 2005*, Leuven, Belgium, Sept. 6–7, 2005.
- [10] Xilinx Corporation. „KEEP” Attribute. Available Online: http://toolbox.xilinx.com/docsan/xilinx7/books/data/docs/cgd/cgd0109_70.html.

A Standard 3.5T CMOS Imager including a Light Adaptive System for Integration Time Optimisation

Estelle LABONNE
 TIMA Laboratory
 Grenoble, France
 Estelle.Labonne@imag.fr

Robin ROLLAND
 CIME Nanotech
 Grenoble, France
 Robin.Rolland@cime.inpg.fr

Gilles SICARD
 TIMA Laboratory
 Grenoble, France
 Gilles.Sicard@imag.fr

Abstract— This paper presents a light adaptive system which allows an automatic management of the integration time value of a standard 3 transistors (3T) CMOS imager. A low resolution network of high dynamic range pixels is included in this standard CMOS sensor. This low resolution network is regularly distributed on the entire photosensitive array, and computes the average incident light information. This value allows the control system to choose the best integration time value which provides the optimal image quality. This imager has been designed in a 0.35 μm , 3.3V CMOS technology. The basic photosensitive block layout contains four 3T standard pixels and one non linear 2T pixel. Due to this distribution, we obtain a 3.5T per pixel. This sensor has been tested and TV video sequences show the efficiency of this very simple control system.

Index Terms—CMOS image sensor, light adaptive system, optimized integration time value, low-cost camera

I. INTRODUCTION

The CMOS image sensors currently present on the market have average performances such as: a dynamic range (DR) and a SNR about 60-70dB, a correct sensitivity (limited by the integration time and the small size of the photodiode) and a correction of the fixed pattern noise (FPN) carried out in the column amplifier [1].

In comparison with CCD sensors, CMOS Active Pixel Sensors (APS) propose lower performances in term of dynamic range, sensitivity and noise (including dark current, temporal noise and fixed pattern noise). But CMOS technology offers advantages in term of production cost, power consumption and integration capabilities.

Researches are undertaken to improve CMOS imagers and to reduce their major drawbacks. Basically, the sensitivity improvement and the dark current minimization could be resolved with optimized CMOS technology. But, dynamic range, temporal noise and FPN problems concern the electronic design. To minimize the noise, several structures exist [1]. To increase the input dynamic range over 100dB (thus better than CCD sensors), several works propose a lot of methods or pixel structures: a long integration time [2], a variable integration time [3], multiple exposures [4],

multigain [5] and continuous operating pixels using a pixel with a logarithmic response [6].

Majors disadvantages of these high dynamic range (HDR) integration pixels are a higher pixel area compared to a standard 3T pixel (Figure 1), a very long readout phase (cumulative integration time) or complex external computation in order to obtain the final HDR image. In another way, continuous operating pixels have the advantage of being very simple (pixel with 3 transistors, Figure 2), providing an instantaneous high dynamic range, about 120dB. But this very simple architecture presents a lower sensitivity, a huge Fixed Pattern Noise (FPN) and a non linear response.

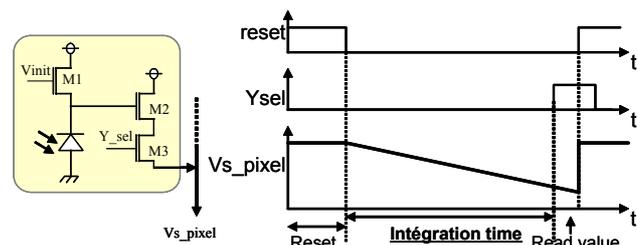


Figure 1. Schematic and timing diagram of a standard 3T pixel

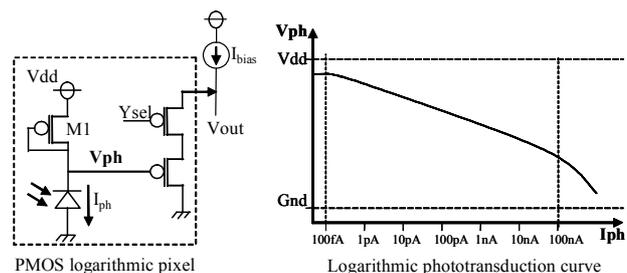


Figure 2. Logarithmic pixel architecture and phototransduction curve

For consumers market, like webcams or mobile phones, all these improved propositions are not really profitable due to the extra costs.

In this work, an intermediate solution is proposed: the

70dB dynamic range of a standard CMOS imager is automatically adapted to the light conditions. It means that the sensor changes instantaneously its integration time value in order to obtain the best image quality. To obtain this light adaptive system, an in-pixel system detects the variation of the average incident light power and modifies automatically the integration time value. The major constraints are to preserve the linear response of a standard pixel and to minimize the silicon area overhead. Another constraint is to implement the simplest possible solution, in order to minimize the cost, the power consumption and to preserve the main electrical and electro-optical characteristics of a standard CMOS imager.

In the state of the art, [7], [8] and [10] propose very interesting solutions: they obtain a specific phototransduction curve (based on logarithmic pixel) and they propose to shift this curve according the illumination condition. These works are bio-inspired systems (Silicon Retina). The major disadvantages of these methods are a large silicon area pixel and a non linear response. Another solution, proposed by [9], explains how authors control the image variation (with the same scene) based on histogram information. But their pixels contain a high number of transistors.

The following section presents the principle of our low-cost light adaptive system. In section III, the sensor architecture is described. In section IV, experimental results are reported and an overview of the sensor is dressed. Finally, conclusions and perspectives are presented.

II. AUTOMATIC CONTROL OF THE INTEGRATION TIME VALUE

In order to detect the variation of the average incident light, a specific array has been designed and inserted inside the photosensitive array with a lower resolution. The goal of this matrix is to provide an output voltage ($V_{ph_average}$) in relation to the average incident light (Figure 3).

Through a feedback loop, this output voltage controls the integration time value. As shown in Figure 4, the analogue voltage $V_{ph_average}$ is amplified and converted in a 3bits binary word. Once digitalized, this information drives the pixel integration time through the reset control signal, managed by the row decoder.

To provide the average incident light value, we have chosen to implement an independent photosensitive array with a high dynamic range. The first feature, the independency, has been decided in order to keep a completely standard functional array (3T pixels), without any interaction with this dedicated array. The second feature, a high dynamic range, has been decided in order to always obtain a valid output, whatever the light condition, without saturation effect.

The chosen pixel architecture is the logarithmic one, originally presented in [6], as it is simple, robust and allowing a high dynamic range. The logarithmic pixel we have designed includes only two NMOS transistors and a photodiode (Figure 5).

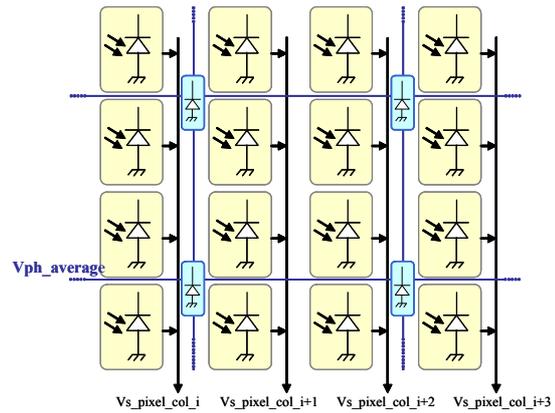


Figure 3. Block diagram of our CMOS imager

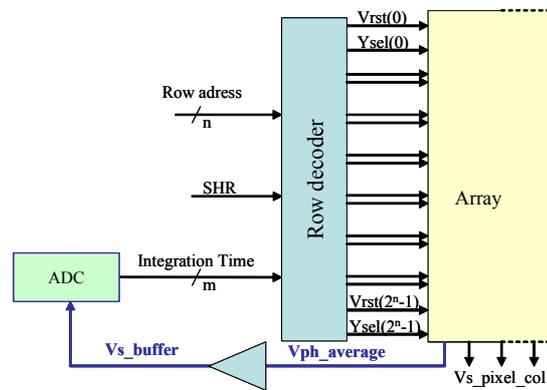


Figure 4. Integration time control system

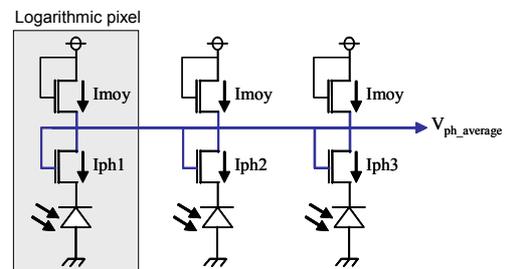


Figure 5. Logarithmic pixels network

The structure implemented is derived from the one proposed in [8]. All logarithmic pixels have a common node and this node voltage is logarithmically dependent of the average photocurrent value. The simulated transfer function curve of this pixel is presented in Figure 6. The output voltage $V_{ph_average}$ is logarithmically dependent of more than five decades of photocurrent, from 1pA to 100nA. The output buffer curve V_{s_buffer} gives a voltage variation of about few hundred millivolts (in the logarithmic part), which allows to provide several integration time values.

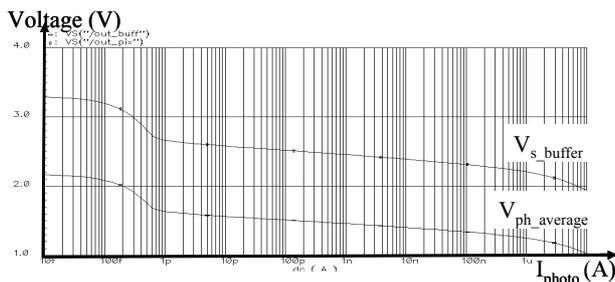


Figure 6. Simulation of the logarithmic pixel response

III. ARCHITECTURE OF THE SENSOR

The proposed image sensor, called IMAGYNE2, is composed of two arrays: a 128x128 standard integration 3T pixel array and a 64x64 logarithmic pixel array, which is regularly distributed with the standard array, and 128 column amplifiers. Two address decoders drive respectively the array rows and the column amplifiers.

The basic layout is shown in Figure 7. This block contains four standard integration pixels and one logarithmic pixel. By abutment of this block, we obtain 128 x 128 standard pixels including a 64 x 64 sub-matrix which provides the average value of luminosity. The area of this basic block layout is 24 x 24 μm^2 . The standard pixel includes three NMOS transistors and a 36 μm^2 N+-P-well photodiode. In this layout, the logarithmic pixel photodiode area is 17 μm^2 . The fill factor is about 25%.

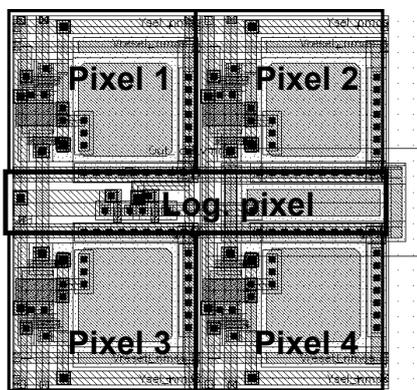


Figure 7. Layout of our light adaptive system basic block: four standard integration pixels including a logarithmic pixel

The integration pixel outputs are connected to the column amplifiers. These readout circuits are located at the bottom of each column. These amplifiers allow to sample and hold the two pixel levels corresponding to the photocurrent output level and the reset output level (according to the classical readout of the standard integration imagers, Figure 1). A special care has been carried out in their design because column amplifiers are a Fixed Pattern Noise (FPN) source. To reduce this offset variation, our column amplifiers present a traditional structure described initially by [1]. This structure allows Correlated Data Sampling (CDS) and Double Delta Sampling (DDS) techniques in order to minimize the pixel to pixel and column to column FPN. The logarithmic sub-matrix provides only one output corresponding to the analogue

voltage $V_{ph_average}$. This voltage is amplified by a buffer and converted by an external ADC into a 3bits code. This code is used by the row decoder, driving the reset control signal of each line and providing the optimized integration time value.

IV. OVERVIEW AND MEASURES OF OUR CIRCUIT

This 128x128 pixel image sensor IMAGYNE2 has been designed in a standard, 0.35 μm , four-metal layers, 3.3V CMOS technology. This sensor has been designed in a multi-projects IMAGYNE test chip, integrating four different imagers. One is a standard 3T imager called REFERENCE. The Figure 8 shows an overview of this chip. Sensors IMAGYNE1 has been presented in [11].

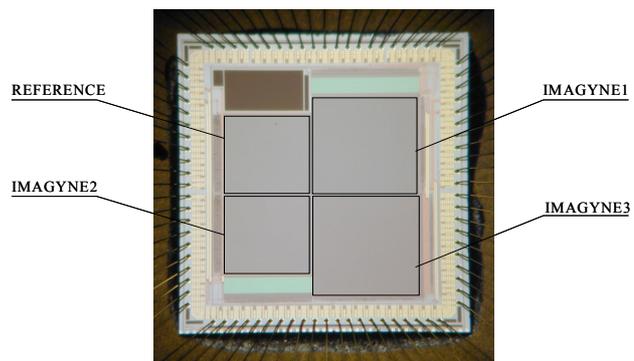


Figure 8. Chip photograph

Figure 9 illustrates the light adaptive capability of our sensor. This figure shows 2 films (TV video format) of the same scene with the same evolution of the light condition. On the left, a film with the REFERENCE array is shown. This standard REFERENCE imager consists in a 128x128 3T pixel array without any feedback loop control. The integration time is controlled with an external command. The images obtained with the light adaptive system (IMAGYNE 2 array) are shown on the right.

Under ambient light (Figure 9a), the light adaptive system allows obtaining an image with a good trade-off of grey levels. An appropriate integration time value is chosen in order to obtain the same trade-off with the reference imager. When a high power light is switched on, the light adaptive system adapts instantaneously the integration time, allowing a good image, while the image obtained by the standard imager presents a majority of saturated pixels (Figure 9b). The integration time is too long and a shorter value is chosen to obtain a better image (Figure 9c). When the high power light is switched off, again, the light adaptive system adapts instantaneously the integration time, allowing a good image, while the standard imager provides a darker response due to the shorter integration time (Figure 9d). A longer value is needed to obtain a good image (Figure 9e). Whatever is the luminosity, the light adaptive system allows to adapt instantaneously the integration time and to obtain good images, while the same imager without this system presents darker or saturated images.

In both films, images obtained with a high light power

(Figure 9c) show two parts in the image with two different integration times. This problem is due to our VHDL code which controls the decoders and the integration phase: This first version doesn't take into account the duration of the row blanking. First rows have a longer integration time due to the addition of the integration time with the blanking row time duration.

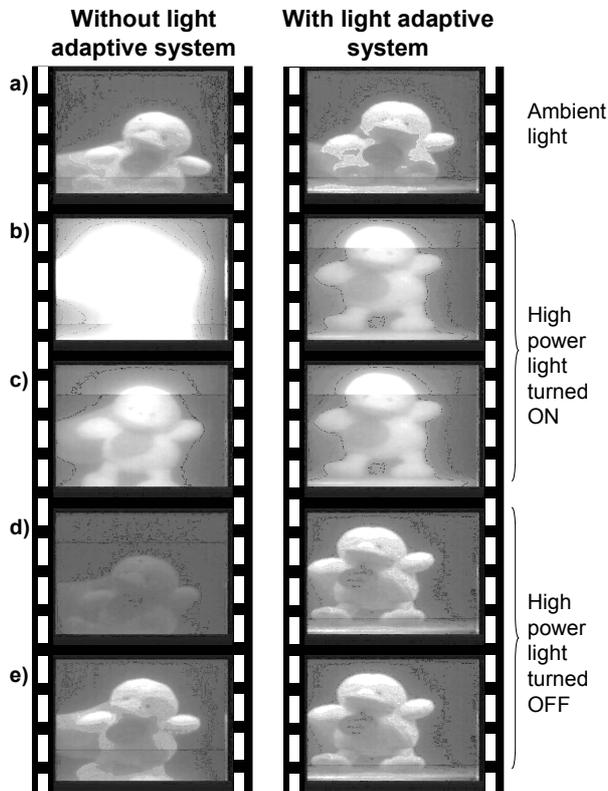


Figure 9. Video sequences with our sensors

V. CONCLUSIONS AND PERSPECTIVES

A light adaptive system has been implemented in a standard CMOS image sensor in order to control its integration time value. The average value of the global incident light power is measured and allows choosing the optimal integration time in a continuous way.

This light adaptive system is implemented through a feedback loop: a network of logarithmic pixels provides information on the sensor average illumination and this data allows computing the optimal integration time. The logarithmic pixels, all connected to a common node, are regularly distributed in the standard pixel array. One logarithmic pixel is inserted in the middle of four standard pixels. The output voltage of this network is logarithmically dependent of the average photocurrent value. The table 1 resumes the main characteristics of our CMOS imager.

At the pixel level, there is an area overhead (about 50%) due to the logarithmic network. But it could be drastically reduced with a more aggressive layout (about 10 to 20%). Moreover, due to the output dynamic voltage obtained and the very low resolution of the ADC used (3bits), the

photodiode area of the logarithmic network can be reduced and we are investigating on the reduction of this sub matrix resolution.

This light adaptive system allows obtaining a very good and simple control of the integration time value. With this system, no anti-blooming system and no mechanical aperture control are needed, contributing to the entire camera cost reduction.

TABLE I. MAIN CHARACTERISTICS OF THE PROPOSED SENSOR

Prototype Chip summary	
Technology	0,35µm CMOS
Standard array resolution	128 x 128 pixels
Log. network resolution	64 x 64 pixels
Transistors per pixel	3,5 NMOS
Pixel pitch	12µm
Photodetectors	N+ P-well photodiode
Fill factor	25%
Acquisition mode	Rolling shutter
Power supply	3,3V
ADC resolution	8 bits
Integration time	According average illumination
Dynamic range	as a standard 3T imager
FPN	as a standard 3T imager
Temporal noise	as a standard 3T imager

REFERENCES

- [1] S.K. Mendis, S.E Kemeny, R.C. Gee, B. Pain C.O. Staller, Q. Kim, E.R. Fossum , "CMOS active pixel image sensors for highly integrated imaging systems", IEEE JSSC, vol.32, February 1997.
- [2] D. Stoppa, A. Simoni, L. Gonzo, M. Gottardi, G.F. Dalla Betta, "Novel CMOS image sensor with a 132dB dynamic range", IEEE JSSC, vol.37, December 2002.
- [3] E. Curluciello, R. Etienne-Cummings, K.A. Boahen, "A biomorphic digital image sensor", IEEE JSSC, vol.36, February 2003.
- [4] D. Yang, A. El Gamal, B. Fowler, H. Tian, "A 640x512 CMOS image sensor with ultrawide dynamic range floating-point pixel-level ADC" IEEE JSSC, vol.34, December 1999.
- [5] M. Schanz, C. Nitta, A. Bußmann, B. J. Hosticka, R. K. Wertheimer, "A high Dynamic range CMOS image sensor for automotive applications", IEEE JSSC, vol.35, July 2000.
- [6] C. Mead & M. Ismael, "Analog VLSI implementation of neural systems", Eds. Boston, MA: Kluwer, 1989.
- [7] T. Delbrück and C.A.Mead "Analog VLSI phototransduction by continuous-time, adaptive, logarithmic photoreceptor circuits", in Vision Chips: Implementing vision algorithms with analog VLSI circuits, C. Koch and H. Li editors, IEEE Computer Society Press, 1995, pp. 139-161
- [8] G. Sicard, G. Bouvier, A. Lelah, V. Fristot, "A light adaptive 4000 pixels analog silicon retina for edge extraction and motion detection", Workshop on Machine Vision and Applications (MVA'98), Chiba, Japon, november 1998.
- [9] Y Ni, F Devos, M Boujrad, J.H Guan, "Histogram-Equalization-based adaptive image sensor for real-time vision", IEEE journal of solid state circuits, Vol. 32, N°7, pp 1027-1036, July 1997.
- [10] T. Delbruck, D. Oberhoff, "Self-biasing low power adaptive photoreceptor", IEEE International Symposium on Circuits and Systems ISCAS 2004, pp. 844-847, may 2004.
- [11] E. Labonne, G. Sicard, M. Renaudin, "An on-pixel FPN reduction method for a high dynamic range CMOS imager", 33rd European Solid-State Circuits Conference, ESSCIRC 2007, Munich, Germany, September 11-13, 2007, pp 332-335.

Self-Timed Implementation of an Impulse Radio Synchronisation Acquisition Algorithm

J eremie Hamon^{†*}, Benoit Miscopein[†], Jean Schwoerer[†], Laurent Fesquet^{*} and Marc Renaudin[‡]

[†] Orange Labs - {jeremie.hamon, benoit.miscopein, jean.schwoerer}@orange-ftgroup.com

^{*} TIMA Laboratory - {jeremie.hamon, laurent.fesquet}@imag.fr

[‡] TIEMPO SAS - marc.renaudin@tiempo-ic.com

Abstract—This paper describes a self-timed implementation of an ultra wideband impulse radio (UWB-IR) synchronisation acquisition algorithm for a non-coherent receiver. The performance of the proposed algorithm has been evaluated by numerical simulations using various statistical channel propagation models. A qualitative comparison of the electrical activity between the proposed asynchronous solution and the standard synchronous one demonstrates the energy efficiency and the relevance of our asynchronous implementation choice.

I. INTRODUCTION

In recent years, academic and industrial communities have been studying the advantages and the opportunities provided by the Ultra Wide Band radio (UWB), compared to short range narrow band solutions, for extremely low-power applications. Among the different ways to generate UWB signals that have been studied [1] [2], impulse radio (UWB-IR) is definitely the most original approach as it compels to reverse the usual time-frequency paradigm of radio communications. Indeed, this technique is based on the emission of very short baseband impulses, whose bandwidth extends up to several GHz of the spectrum. The baseband electromagnetic radiation prevents from employing any sinusoidal carrier modulation. Fig. 1 shows an example of a possible impulse shape both in time and frequency domains.

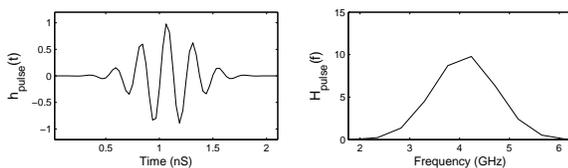


Fig. 1. Example of an elementary impulse in time and frequency domains

Thanks to its pulsed nature, amenable to low complexity implementation, UWB-IR presents several interesting properties. One of them is very low power consumption. Compared to usual narrow band radio systems, for which an RF carrier is always radiated independently of the information to transmit, an UWB-IR system only radiates a number of impulses, depending on this information; thus its energy consumption is limited to the minimum required. A similar minimum property also characterises self-timed circuits. Indeed, in asynchronous systems, the dynamic consumption

is only due to the logical blocks actually implicated in a processing at a particular time. This property comes from the substitution of global synchronisation signal (i.e. the clock) by a local synchronisation implemented by a bidirectional signalling between the different logical blocks.

This paper describes a self-timed implementation of an impulse radio detection and synchronisation acquisition algorithm. Section II exposes the principles of UWB-IR communication, and more particularly the synchronisation phase. This phase represents the key step to establish an impulse radio communication. The proposed synchronisation acquisition algorithm and the associated self-timed architecture are described in section III. Finally, section IV presents the performance measurements obtained by means of numerical simulation performed with *Matlab*. These results validate the proposed method on different statistical channel propagation models provided by the IEEE 802.15.4a Standardisation Group [3]. Furthermore, a qualitative evaluation of the receiver electrical activity induced by the algorithm demonstrates the interest of the asynchronous approach for the low power consumption issue.

II. UWB-IR COMMUNICATION DESCRIPTION

A. System model

The regulation imposes the use of impulses with a power spectral density limited to -41.3 dBm/MHz. As this corresponds to only several tens of μ W for a 1 GHz band, it is necessary to insert redundancy at the impulse emission to improve the signal-to-noise ratio (SNR) at the decision stage. A data symbol is then coded by N_{chip} impulses, modulated in On-Off Keying (OOK) as shown in Table I. The last line, entitled *Synchro* corresponds to the specific mapping used for synchronisation, as explained later on.

TABLE I
ON-OFF KEYING (OOK) OF A DATA SYMBOL ($N_{chip} = 4$)

Symbol	Modulation
0	0,1,0,1
1	1,0,1,0
Synchro	1,1,1,1

In order to avoid forbidden spectral lines due to a periodic impulse emission, a *Time Hopping* (TH) scheme is employed to break the *pulse repetition period* and whiten the resulting spectrum. Each of the chips composing the duration of a symbol is sub-divided into N_{slot} slots as shown on Fig. 2. The pseudo-random code, C_{TH} , that governs the TH process defines in which slot of each chip an impulse should be emitted.

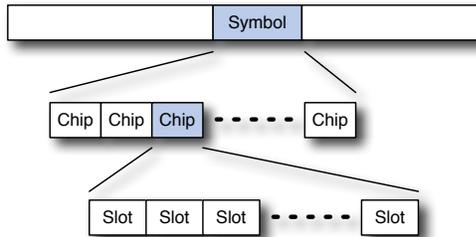


Fig. 2. Temporal subdivision of a symbol

Fig. 3 shows an example of an emitted symbol combining On-Off Keying and Time Hopping. The TH sequence is periodic with period equal to the symbol duration, i.e. the TH sequence is repeated at each symbol. In addition, we note that this spread spectrum technique allows for multiple access to the channel by assigning a different pseudo-random code, C_{TH} , to each emitter/receiver pair [4]. Indeed, if these different codes respect some criteria [5], asynchronous concurrent communications between several emitter/receiver pairs are possible. The impact of multi-user interferences is out of the scope of this study.

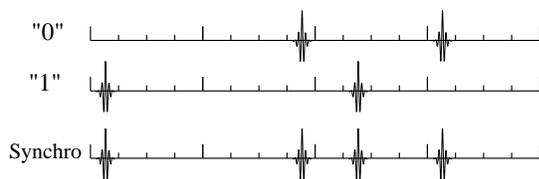


Fig. 3. Data and synchronisation symbols ($N_{chip} = N_{slot} = 4$; $C_{TH} = \{1, 4, 2, 1\}$)

B. UWB-IR reception overview

Once synchronisation is achieved, the receiver is able to locate the transmitted impulses and to demodulate the data by processing the received signal during observation windows, positioned according to the TH sequence. The nature of the required signal processing is discussed in Section III.

In order to retrieve the emitter synchronisation and properly open the observation windows, the receiver has to identify the TH sequence in the incoming signal. To do so, it exploits a synchronisation preamble, emitted before every data packet. As shown on Table I and Fig. 3, this synchronisation preamble composed of 32 unmodulated symbols, only contains the TH

sequence information and corresponds to a kind of temporal signature. During the reception of the synchronisation preamble the receiver seeks for a succession of detected impulses that match the TH sequence.

C. Reception architecture

Two main classes of UWB-IR receivers can be found in the literature: the coherent and non-coherent receivers. Coherent receivers are inspired from the work presented in [1]. These receivers are based on performing a correlation of the incoming signal and a locally generated correlation template. The template shape is tailored in such a way that the correlation ratio obtained in an observation window is good enough to decide upon the received symbol. This technique exploits the signal phase to trigger the correlation at the instant that produces the best correlation ratio and therefore the best signal to noise ratio. This point exhibits a key constraint of the coherent receiver: the required timing resolution for the correlation triggering is of the order of several tens of picoseconds. This explains why this kind of receivers is not well suited for low power consumption devices. Such devices are mainly based on non-coherent architectures. This reception technique is based on the incoming signal energy and is therefore subject to less stringent timing resolution constraints. As a matter of fact, a non-coherent receiver resolution is of the order of the impulse duration (~ 1 ns). Furthermore, non-coherent receivers have more simple architecture and consume less energy per bit than coherent receivers [6].

D. Energy detection

The functional architecture of a non-coherent receiver is depicted on Fig. 4. One can recognise the classic envelope detector, based on a square-law device followed by a low-pass filter. The latter is equivalent to a signal integration during T_{int} , which corresponds to the inverse of the filter cutoff frequency.

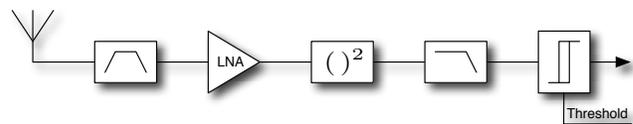


Fig. 4. RF Front-End schematic

In the literature, UWB-IR non-coherent circuits employ a 30-50 ns integrator ([6], [7]). This duration is tailored to gather a significant part of the radiated energy after its propagation through a time dispersive channel. Such a choice allows to get, at the integrator output, the aggregated energy of the main multipath components, thus dispensing us from implementing a multipath receiver. As an illustration, Fig. 5 shows a realisation of the NLOS (Non Light Of Sight) office

environment channel model, defined by the IEEE 802.15.4a task group[8]). On this figure, the main propagation path delays are spread over more than 50 ns.

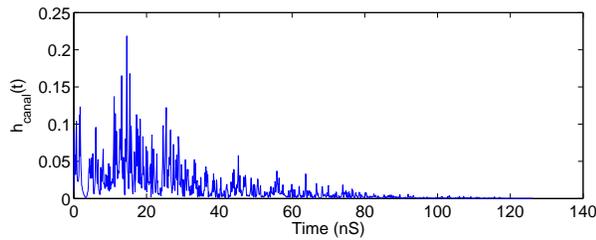


Fig. 5. Channel impulse response of the IEEE 802.15.4a NLOS office environment channel model

However, the signal temporal resolution is severely decreased when using such integration durations. This resolution degradation can be harmful when looking for high accuracy ranging applications. Another major drawback is that a long integration duration provokes a sensitivity to noise and multi-user interference. [9] shows that a non-coherent receiver is particularly affected by concurrent UWB communications and suffers severe performance degradation.

This explains the choice in this paper for non-coherent reception based on a very short (2 ns) integration duration. The performance analysis of this receiver is out of the scope of this study but one can intuitively see that a very time selective receiver might be more robust to noise and co-channel interference. The counterpart of this choice is that we need to resort to multipath processing to get a sufficient SNR at the decision stage. The general architecture of the considered receiver is presented on Fig. 6 and described in [10].

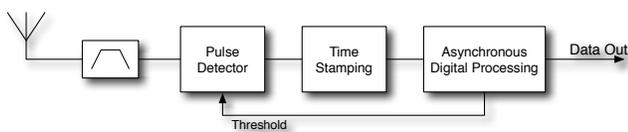


Fig. 6. General reception architecture

As it is depicted in Fig. 6, the incoming signal envelope passes through a threshold device (the *Pulse Detector* block) and each threshold crossing is interpreted as an impulse detection. The threshold setting is a key point of the receiver performance and it is governed by a trade-off between minimum detectable signal level and asymptotic bit error rate. In this paper, we assume that the threshold value is set according to a radar oriented method called CFAR (Constant False Alarm Rate) [11] described in Section IV. During a calibration process, the threshold is adjusted so as to observe a fixed number of noise peaks. The threshold crossing events trigger a time base which is used to *time-stamp* them. These events then activate the asynchronous baseband logic and

initially the synchronisation acquisition algorithm block.

III. SYNCHRONISATION ALGORITHM AND ARCHITECTURE

The synchronisation acquisition algorithm presented in this paper is an asynchronous finite state machine (*A-FSM*) implementation of the algorithm described in [10]. Once again, the synchronisation is acquired by identifying the temporal pattern of the TH sequence in the detected impulses. In this approach, the retrieval of the time-hopping sequence is based on comparing the measured time gaps between the different detected impulses, to the expected distances composing the TH sequence.

A point to retain, since it will be used later on, is that the TH sequence is built on the basis of the slot repetition rate. It means that no matter what the UWB signal alterations are (e.g. multipath propagation or noise peaks or even co-channel interference), the TH sequence distances can be effectively detected at the receiver side by means of only temporal distances which are multiples of the time slot. That is why we propose to sub-divide the slot in N_{unit} independent time units. Furthermore, we set the time unit equal to the temporal resolution of the system, here 2 nanoseconds.

The measurement of the temporal distances between the detected impulses and the synchronisation search are made concurrently and independently on each time unit. Fig. 7 shows this parallel architecture of the time stamping and synchronisation blocks, tailored to a slot duration of 20 ns, leading to $N_{unit} = 10$.

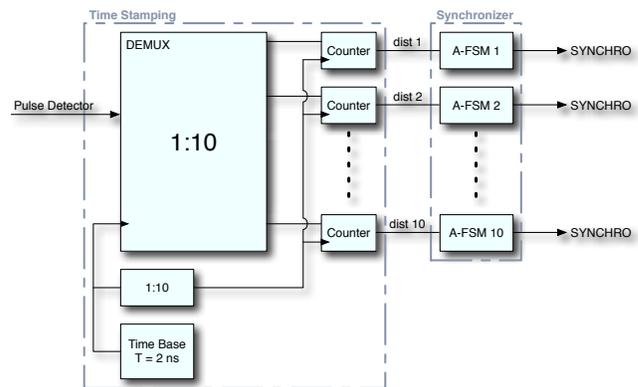


Fig. 7. Time stamping and synchronisation architecture - 10 time units per slot

A. Time Stamping Block

This block is in charge of two functions. Firstly, it performs a kind of *parallelisation* of the signal at the time unit rate: if an impulse is detected in time unit i , then the signal is connected to the corresponding counter i and the detection windows will be re-evaluated at the same time unit of the next time slot. Secondly, the time stamping block carries out

the temporal distance measurement between the impulses detected in the same time unit. It is implemented by a counter clocked at the slot rate. By this way, the temporal distances delivered concurrently by the time stamping block are measured in time slot.

B. A-FSM Description

Each of the N_{unit} asynchronous FSMs is activated by every reception of a new temporal distance. This distance is then compared to the TH code distances, to determine what was the last detected impulse. By repeating this operation several times, it is possible to identify the complete TH sequence, and then synchronise with the emitter.

However, due to the signal alterations, some of the emitted impulses will not be detected. Thus the synchronisation algorithm has to be robust enough to cope with potentially missed impulses. We propose to address this point by enriching the temporal distance alphabet with the addition of temporal distances of the 2nd and 3rd order as defined here.

$$\forall i \in [1, N_{chip}] :$$

$$\begin{cases} D_{i,i+1} = C_{TH}(i+1) - C_{TH}(i) + N_{slot} \\ D_{i,i+2} = C_{TH}(i+2) - C_{TH}(i) + 2 \times N_{slot} \\ D_{i,i+3} = C_{TH}(i+3) - C_{TH}(i) + 3 \times N_{slot} \end{cases}$$

Along the same lines, the synchronisation algorithm must manage spurious detections (false alarms) either due to either noise peaks or due to co-channel interference. The synchronisation algorithm must be able to identify valid temporal distances equal to the sum of several received distances triggered by spurious detections. As an example, if a spurious impulse s has been detected between two valid ones i and $i+1$, the algorithm can detect the temporal distance $D_{i,i+1}$ on the sum of the distances $D_{i,s}$ and $D_{s,i+1}$.

In this algorithm, the synchronisation is declared as acquired if all the impulses composing a symbol have been detected at least one time. Fig. 8 outlines the structure of one synchronisation A-FSM and its states are detailed next.

Init: Reset of the different variables of the A-FSM.

State 0: A received distance D is compared to the full distance alphabet. If the comparison succeeds, the next state is *State i* with i corresponding to the last detected impulse ($D = D_{i-1,i}$, $D = D_{i-2,i}$ or $D = D_{i-3,i}$); else, the next state is *Noise 0*.

State i: The impulse detection flag i is activated: $flag_i = 1$. The spurious detection counter is reset to 0: $noise = 0$ and the variable that represents the last detected impulse is

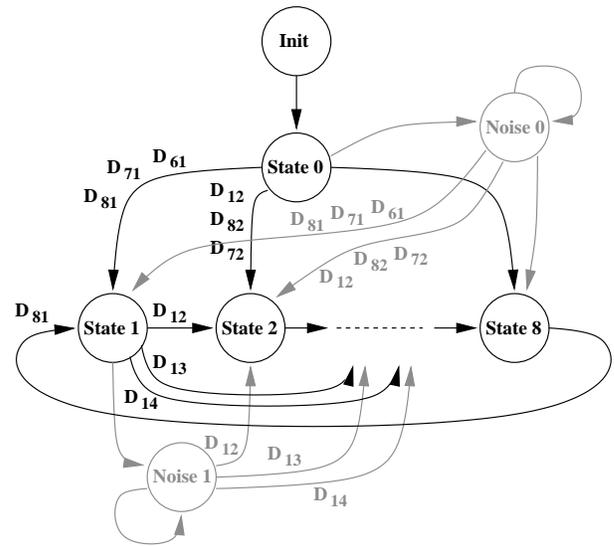


Fig. 8. Outline of the synchronisation A-FSM - $N_{chip} = 8$

updated: $last = i$. When a new received distance D is provided by the time stamping block, D is compared to the TH sequence alphabet based on chip i ($D_{i,i+1}$, $D_{i,i+2}$ or $D_{i,i+3}$). If the comparison succeeds, the next state is *State i+1*, *State i+2* or *State i+3* with respect to the identified distance; otherwise, the next state is *Noise i*.

Noise 0: The spurious detection counter is incremented: $noise = noise + 1$. A new received distance D is compared to the full distance alphabet, and if this comparison fails, the sum of the new distance D and of the previous one D^{-1} is compared to the full alphabet. This operation is repeated until either a valid distance is identified or the maximum number of tolerated spurious impulses is exceeded. In the former case, the A-FSM reaches state i , i being the last detected impulse. In the latter case, the A-FSM is reset in its state *Init*.

Noise i: The spurious detection counter is incremented: $noise = noise + 1$. The sum of the new received distance D and the previous ones is compared to the distances of the alphabet which exist from the impulse i : $D_{i,i+1}$, $D_{i,i+2}$ or $D_{i,i+3}$. This operation is repeated until a valid distance is identified or the maximum number of spurious detections limit is exceeded. Both cases in this state are similar to those of state *Noise 0*.

Synchronisation is achieved when all the impulse detection flags are set to 1 ($flag_1 = flag_2 \dots flag_{N_{chip}} = 1$) and the position in the symbol time is provided with the $last$ variable. Then, applying the time hopping code, it is possible to predict the arrival of the next impulse.

IV. DIGITAL SIMULATIONS

A. Simulation environment

Both architecture and algorithm presented in this paper have been implemented in *Matlab*. In order to validate the synchronisation algorithm and evaluate its performances with realistic stimuli, a signal generator which integrates statistical propagation channel models for different kind of usual propagation environments has been designed (the channel models are provided by the standardisation committee IEEE 802.15.4.a [8]). Fig. 9 represents the simulator synopsis.

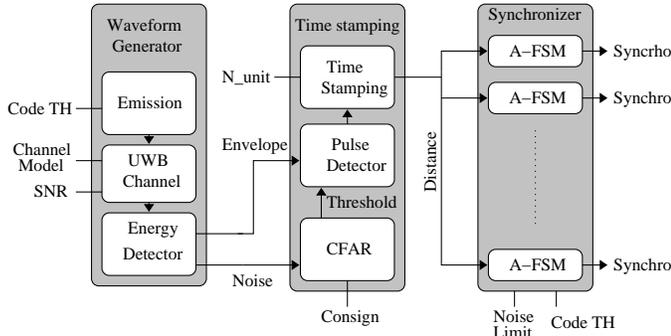


Fig. 9. Matlab simulator synopsis

B. Simulation parameters

For these simulations, the UWB-IR physical layer is defined as follows:

- Synchronisation preamble duration : 32 symbols,
- Symbol duration, $T_s = 1280 \text{ ns}$,
- Number of impulses per symbol (i.e. number of chips per symbol), $N_{chip} = 8$,
- Number of slots per chip, $N_{chip} = 8$,
- Number of time units per slot, $N_{unit} = 10$, leading to $T_{unit} = 2 \text{ ns}$
- Time hopping code: $C_{TH} = \{6, 4, 0, 5, 6, 1, 1, 0\}$.

As mentioned previously, the detection threshold is calibrated on the noise level. The CFAR method consists in determining the maximum number of spurious detections due to the noise to maintain a suitable bit error rate level (BER). By computing the false alarm probability that corresponds to this BER level, it is possible to determine the average number of spurious detections during a chip duration. In our case, the BER performance floor is set to 10^{-4} what corresponds to set the CFAR consign to an average of 9.51 spurious detections during a chip duration.

As well, we propose to evaluate the performances and the robustness of the algorithm for a SNR range from 0 dB to -17 dB . In this study, the SNR is defined as the signal to noise ratio measured after the first pass band filter of the RF front end (Fig. 4).

Finally, the algorithm is validated on these 4 different statistically defined propagation channel models:

- CM1 and CM2 which correspond to a residential propagation environment respectively in light of sight (LOS) and non light of sight (NLOS) conditions.
- CM3 and CM4 which correspond respectively to LOS and NLOS office propagation environments.

For each channel model and SNR value, the simulation is run 100 times so as to obtain a statistical performance evaluation. A lot of indicators could be used to measure the performances of a UWB-IR synchronisation algorithm, in this paper, we propose to focus on :

- the synchronisation success rate,
- the number of detected paths,
- the synchronisation duration.

C. Simulation Results

In this performance study, the synchronisation is considered as acquired if, on the one hand, the TH sequence has been detected on one time unit at least before the end of the synchronisation preamble, and on the other hand, if this detected time unit is robust enough to bear the demodulation process (i.e. if the chip error rate from the instant of synchronisation to the end of the preamble is less than 50 %).

1) *Synchronisation success rate*: This indicator represents the ability of the algorithm to acquire the synchronisation in a given propagation condition. Actually, it allows to define the minimum SNR level for which the synchronisation process succeeds with a suitable rate, assumed to be 90%. Fig. 10 represents the synchronisation success rate for the 4 propagation channels. Obviously, the minimum SNR level depends on the propagation channel characteristics. In case of LOS channels (CM1 and CM3), for which it exists some high energy propagation paths, this level is about -14.5 dB . In case of NLOS channels, for which the impulse energy is spread on more propagation paths, this minimum SNR level rises to -13.5 dB for CM3 and to -12 dB for CM4.

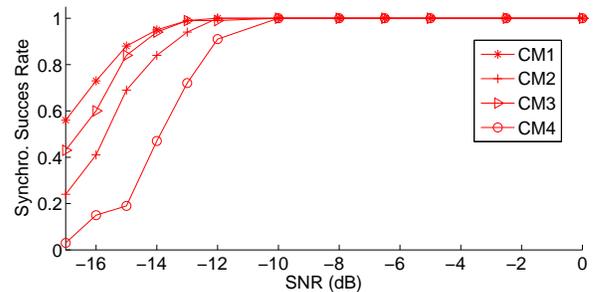


Fig. 10. Synchronisation success rate

2) *Number of detected paths*: This indicator corresponds to the sensitivity of the synchronisation algorithm. It represents the number of time units where the TH sequence has been detected. As previously mentioned, due to the specific RF front end, a multi-path processing is needed to gather a significant part of the emitted energy. A multi-path processing is intrinsically available in the proposed synchronisation architecture, without any complexity addition, thanks to the parallelisation of the UWB signal reception at the time unit rate. However, the number of detectable multi-path components is limited to the number of parallel time units. Moreover, it is important to remark that two propagation paths separated with a multiple of the time slot duration can not be discriminated. Indeed, in this particular case, the impulse detections of the second path are stamped in the same time unit of the first one and then, are interpreted as spurious detections in the synchronisation process of the first one. Fig. 11 shows the number of detected paths for the 4 tested channel models. Naturally, the number of detected paths decreases with respect to the SNR. However, we can remark that the number of detected paths is larger for NLOS channels than for LOS ones for relatively high SNR (from 0 dB to -10 dB), and inversely for relatively low SNR (from -10 dB to -17 dB). This behaviour can be easily explained by the characteristics of the different propagation channels: the emitted energy is spread over a lot of multi-path components in NLOS channels as it is spread on fewer high energy paths in LOS ones. Therefore, it is easier to detected several paths at high SNR in NLOS channels and respectively easier to detect the few high energy paths of the LOS channels in case of lower SNR.

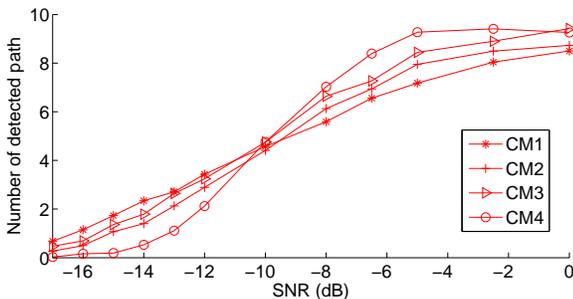


Fig. 11. Number of detected paths

3) *Synchronisation acquisition duration*: This indicator represents the required duration to establish the synchronisation. It corresponds to the required number of symbols to identify the TH sequence on the first time unit. Fig. 12 represents this indicator for the 4 evaluated propagation channels. At high SNR, the synchronisation algorithm needs only one or two symbols to get the synchronisation: in this case, almost all emitted impulses of TH sequence can be detected, and then the synchronisation process succeeds quickly. At lower SNR, the synchronisation process needs more symbols to identify the whole TH

sequence. This comes from the fact that some emitted impulses are not detectable (due to the noise). In this case, the synchronisation algorithm uses the temporal distances of the 2nd and 3rd order. That causes a longer synchronisation time.

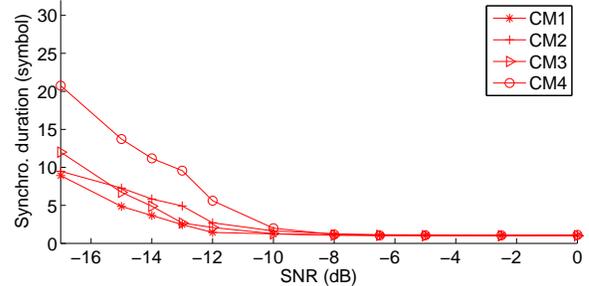


Fig. 12. Synchronisation acquisition duration

D. *Qualitative evaluation of the electrical activity*

It is well known that the consumption comparison between a self-timed implementation and a synchronous one of the same system is tricky. First of all, these two different methods of design lead to very different architectures usually hard to compare. And afterwards, the energy consumption of an asynchronous system depends on the circuit class (QDI, Micropipeline,...), on the data coding scheme and even on the chosen communication protocol. In this paper, we deal with the feasibility and the interests of the asynchronous approach for UWB-IR signal processing without taking care of the circuit implementation issues. Therefore, we propose to evaluate the energy consumption of the synchronisation algorithm by the number of computations executed by the FSMs during the whole synchronisation preamble.

In the case of a synchronous implementation, the FSMs should be activated at slot rate. During the N_{symp} symbols of the synchronisation preamble, the number of computations that represents the electrical activity of the N_{unit} FSMs can be expressed as:

$$N = N_{unit} \times N_{symp} \times N_{chip} \times N_{slot} = 20480$$

In the case of a self-timed implementation as described in this paper, the number of computations executed by the A-FSMs corresponds to the number of detected impulses. This number directly depends on the UWB-IR signal, the propagation channel (i.e. the number of detectable paths), the noise level and especially the threshold value. Figure 13 represents the number of events (i.e. the number of detected impulses) processed by the A-FSMs during the whole synchronisation preamble with respect to the SNR.

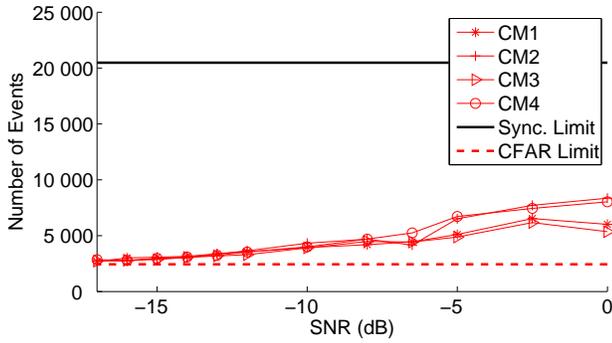


Fig. 13. Number of processed events

Firstly, we remark that whatever the SNR level or the propagation channel, the number of events of the asynchronous approach is widely inferior to the number of computations of the synchronous one (continuous black line). Moreover, the number of events depends on the propagation channels and decreases with respect to the SNR to tend toward the limit fixed by the CFAR consign. Contrary to a synchronous implementation for which the number of computations remains constant, the self-timed implementation directly takes advantage of these event number variations.

However, the actual number of events processed by the FSMs in both implementations also depends on the synchronisation time duration and on the number of detected paths. Indeed, since a FSM detects the synchronisation, it does not consume any more during the tail of the synchronisation preamble. Fig. 14 represents this actual number of events processed by the N_{unit} A-FSMs for the different propagation channel models. At high SNR, it is the high number of impulse detections that mainly contributes to the electrical activity. Despite of the global reduction of the number of impulse detections at low SNR, the length of the synchronisation process and the restricted number of detected paths causes an increase of the actual number of processed events. Finally, it exists an optimal SNR range for which the algorithm performances remain quasi-optimal (synchronisation success rate, synchronisation duration and number of detected paths) and the electrical activity really limited.

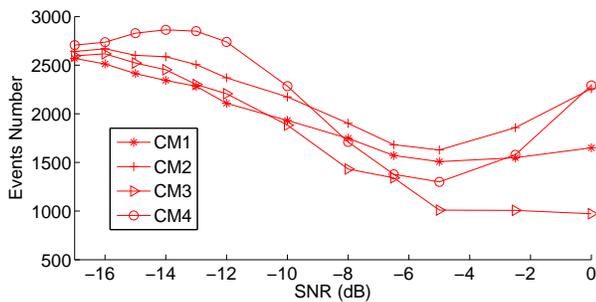


Fig. 14. Actual number of processed events

E. Influence of the detection threshold

In addition, as previously mentioned, the detection threshold value is a key point of the receiver performances. As example, if the number of spurious detections of the CFAR consign is tailed off, the sensitivity of the receiver is damaged but the BER performance floor is enhanced. We propose to study the influence of the CFAR consign on the performances of the synchronisation algorithm and also on its electrical activity. A new set of numerical simulations is performed on the CM4 channel model with 3 different BER performance floors: 10^{-3} , 10^{-4} and 10^{-5} . They correspond respectively to an average number of spurious detections of: 16.92, 9.51 and 5.35 per chip.

Fig. 15 represents the synchronisation success rate for the 3 CFAR consigns. Naturally, when the sensitivity is increased, the algorithm is able to acquire the synchronisation in harder propagation conditions.

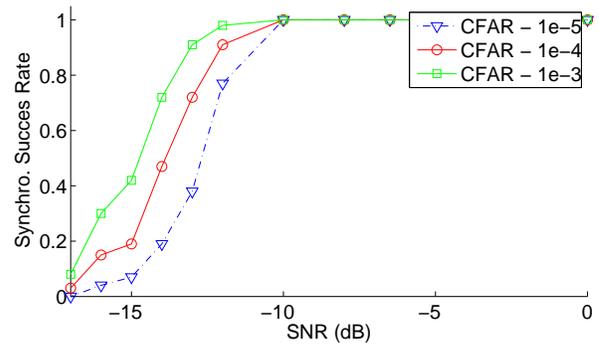


Fig. 15. Synchronisation success rate

Fig. 16 represents the number of detected paths. As well as previously, at low SNR, if the sensitivity is increased then the number of detected paths is increased. However, for high SNR, we observe that the number of detected paths is bigger when the sensitivity is decreased. In fact, the CFAR method is not suited to high SNR values: the number of detections due to the multi-path components are too important and it is not easy to identify the TH sequence in all these detections.

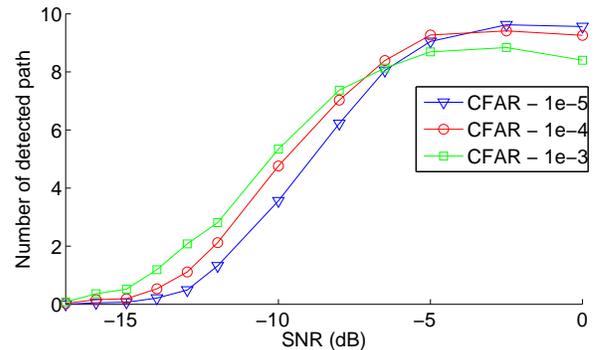


Fig. 16. Number of detected paths

Fig. 17 represents the synchronisation duration. Naturally, a better sensitivity allows a faster synchronisation process.

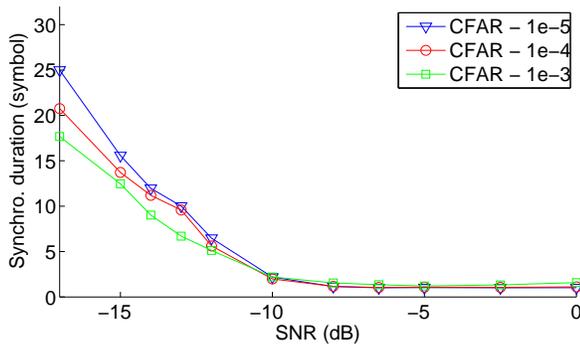


Fig. 17. Synchronisation duration

Fig.18 represents the actual number of events for the different CFAR consigs. It clearly appears that the electrical activity of the synchronisation algorithm can be tuned by the detection threshold setting.

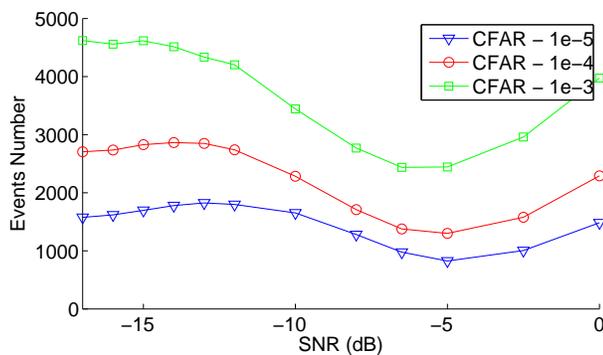


Fig. 18. Actual number of processed events

This set of simulations demonstrates that the performances of the algorithm and its electrical activity can be mitigated by tuning the detection threshold setting. Thanks to the event nature of the asynchronous logic, the synchronisation algorithm proposed in this paper directly benefits from the variation of the threshold to reduce its electrical activity.

V. CONCLUSION

This paper presents a self-timed implementation of an UWB-IR synchronisation acquisition algorithm for a non-coherent receiver. The performances of the algorithm are evaluated by numerical simulations performed with realistic stimuli. Furthermore, a qualitative evaluation of the energy consumption demonstrates the interest of the asynchronous approach compared to an usual synchronous implementation.

In addition, the numerical simulation results underline the impact of the detection threshold on the electrical activity of

the receiver. Our future works will focus on a *ad hoc* method to adjust the detection threshold in order to take benefit of the great flexibility offered by the asynchronous design style.

VI. ACKNOWLEDGMENT

This work has been conducted as part of the MEDEA SWANS project and partially funded by the French MINEFI ministry.

REFERENCES

- [1] R. Scholtz, "Multiple access with time-hopping impulse modulation," in *Military Communications Conference, 1993. MILCOM '93. Conference record. 'Communications on the Move'.*, IEEE, vol. 2, Boston, MA, USA, Oct. 1993, pp. 447–450.
- [2] S. Hara and R. Prasad, "Overview of multicarrier CDMA," *IEEE Communications Magazine*, vol. 35, no. 12, pp. 126–133, Dec. 1997.
- [3] "IEEE Standard for Information Technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - specific requirement Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)," *IEEE Std 802.15.4a-2007 (Amendment to IEEE Std 802.15.4-2006)*, pp. 1–203, 2007.
- [4] M. Z. Win and R. A. Scholtz, "Ultra-wide bandwidth time-hopping spread-spectrum impulse radio for wireless multiple-access communications," *IEEE Transactions on Communications*, vol. 48, no. 4, pp. 679–689, Apr. 2000.
- [5] C. Le Martret, A.-L. Deleuze, and P. Ciblat, "Optimal time-hopping codes for multi-user interference mitigation in ultra-wide bandwidth impulse radio," *Wireless Communications, IEEE Transactions on*, vol. 5, no. 6, pp. 1516–1525, June 2006.
- [6] D. D. Wentzloff, F. S. Lee, D. C. Daly, M. Bhardwaj, P. P. Mercier, and A. P. Chandrakasan, "Energy Efficient Pulsed-UWB CMOS Circuits and Systems," in *Ultra-Wideband, 2007. ICUWB 2007. IEEE International Conference on*, Sep. 2007, pp. 282–287.
- [7] S. Dubouloz, B. Denis, S. de Rivaz, and L. Ouvry, "Performance analysis of LDR UWB non-coherent receivers in multipath environments," *IEEE international Conference on Ultra-Wideband*, 2005.
- [8] A. F. Molisch, K. Balakrishnan, D. Cassioli, C.-C. Chong, S. Emami, A. Fort, J. Karedal, J. Kunisch, H. Schantz, U. Schuster, and K. Siwiak, "IEEE 802.15.4a channel model - final report," IEEE 802.15.4a Channel Sub-Committee, Tech. Rep. Document IEEE 802.15-04-0662-02-004a, 2005.
- [9] M. Flury, R. Merz, J.-Y. Le Boudec, and J. Zory, "Performance evaluation of an ieee 802.15.4a physical layer with energy detection and multi-user interference," *Ultra-Wideband, 2007. ICUWB 2007. IEEE International Conference on*, pp. 663–668, 24–26 Sept. 2007.
- [10] B. Miscopein and J. Schwoerer, "Low complexity synchronization algorithm for non-coherent UWB-IR receivers," in *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*, Apr. 2007, pp. 2344–2348.
- [11] J. Dugundji and E. Ackerlind, "Automatic bias control for a threshold detector," *Information Theory, IEEE Transactions on*, vol. 3, no. 1, pp. 65–70, Mar 1957.

Optimization of automatically generated multi-core code for the LTE RACH-PD algorithm

Maxime Pelcat

IETR/INSA, UMR CNRS 6164,
Rennes, France
mpelcat@insa-rennes.fr

Slaheddine Aridhi

Texas Instruments, CIV Division,
Villeneuve Loubet, France
saridhi@ti.com

Jean-François Nezan

IETR/INSA, UMR CNRS 6164,
Rennes, France
jnezan@insa-rennes.fr

Abstract— Embedded real-time applications in communication systems require high processing power. Manual scheduling developed for single-processor applications is not suited to multi-core architectures. The Algorithm Architecture Matching (AAM) methodology optimizes static application implementation on multi-core architectures.

The Random Access Channel Preamble Detection (RACH-PD) is an algorithm for non-synchronized access of Long Term Evolution (LTE) wireless networks. LTE aims to improve the spectral efficiency of the next generation cellular system. This paper describes a complete methodology for implementing the RACH-PD. AAM prototyping is applied to the RACH-PD which is modelled as a Synchronous DataFlow graph (SDF). An efficient implementation of the algorithm onto a multi-core DSP, the TI C6487, is then explained. Benchmarks for the solution are given.

I. INTRODUCTION

The recent evolution of digital communication systems (voice, data and video) has been dramatic. Over the last two decades, low data-rate systems have been replaced or augmented by systems capable of data rates of several Mbit/s, supporting multimedia applications (such as DSL, cable modems, 802.11b/a/g/n wireless local area networks, 3G and WiMAX). The 3GPP Long Term Evolution (LTE) represents a recent part of this evolution, enabling data rates beyond hundreds of Mbit/s in potentially very wide cells.

As communication systems have evolved, the resulting increase in data rates has necessitated higher system algorithmic complexity. A more complex system requires greater flexibility in order to function with different protocols in diverse environments. Additionally, there is an increased need for the system to support multiple interfaces and multi-component devices. Consequently, this requires the optimization of device parameters over varying constraints, such as performance, area and power. Achieving this device optimization requires a good understanding of the application complexity and the choice of an appropriate architecture to support this application.

System on a Chip (SoC) with several cores such as multi-core DSPs is becoming the standard basic element used to build complex telecommunication systems. The task of distributing pieces of an algorithm over a multi-component architecture is not straightforward. When performed manually, the

result is inevitably a sub-optimal solution. There is a need for new methodologies that allow the exploration of several solutions thus producing a more optimal result. For the current work, the methodology of Algorithm-Architecture Matching (AAM, previously called AAA [6]) is employed using the Parallel Real-time Embedded Executives Scheduling Method (PREESM) tool. The PREESM tool is an open framework which provides a flexible method for exploring architectures suited for deterministic applications. More than just a simulation tool, PREESM can generate code. Associated with well-optimized code, communication and synchronization, the automatic generation leads to an efficient algorithm implementation.

This article presents an overview of the LTE Random Access Channel (RACH) preamble detection algorithm and the PREESM tool. Subsequently, the preamble detection application is described using a Synchronous DataFlow graph (SDF). The virtual prototyping of this application over multi-processor architectures using PREESM tool features is then detailed. The target architecture is a multi-core DSP from Texas Instruments, the C6487. An implementation onto this DSP is performed with optimized inter-core communication and synchronizations using Direct Memory Access (DMA). Finally future work is discussed and conclusions are given.

II. PREAMBLE DETECTION PROCESS

The RACH is a contention-based uplink channel used mainly for initial transmission requests from the User Equipment (UE) to the evolved base station (eNodeB) for connection to the network. The UE seeking connection with a base station sends its signature in a RACH preamble dedicated time and frequency window in accordance with a predefined preamble format. Signatures have special auto-correlation and inter-correlation properties that maximize the ability of the eNodeB to distinguish one UE from another. The RACH preamble procedure is implemented in the LTE eNodeB to detect and identify each user's signature and is dependent on the cell size and the system bandwidth. We assume that the eNodeB has the capacity to handle the processing of this RACH preamble detection every millisecond in a worst case scenario.

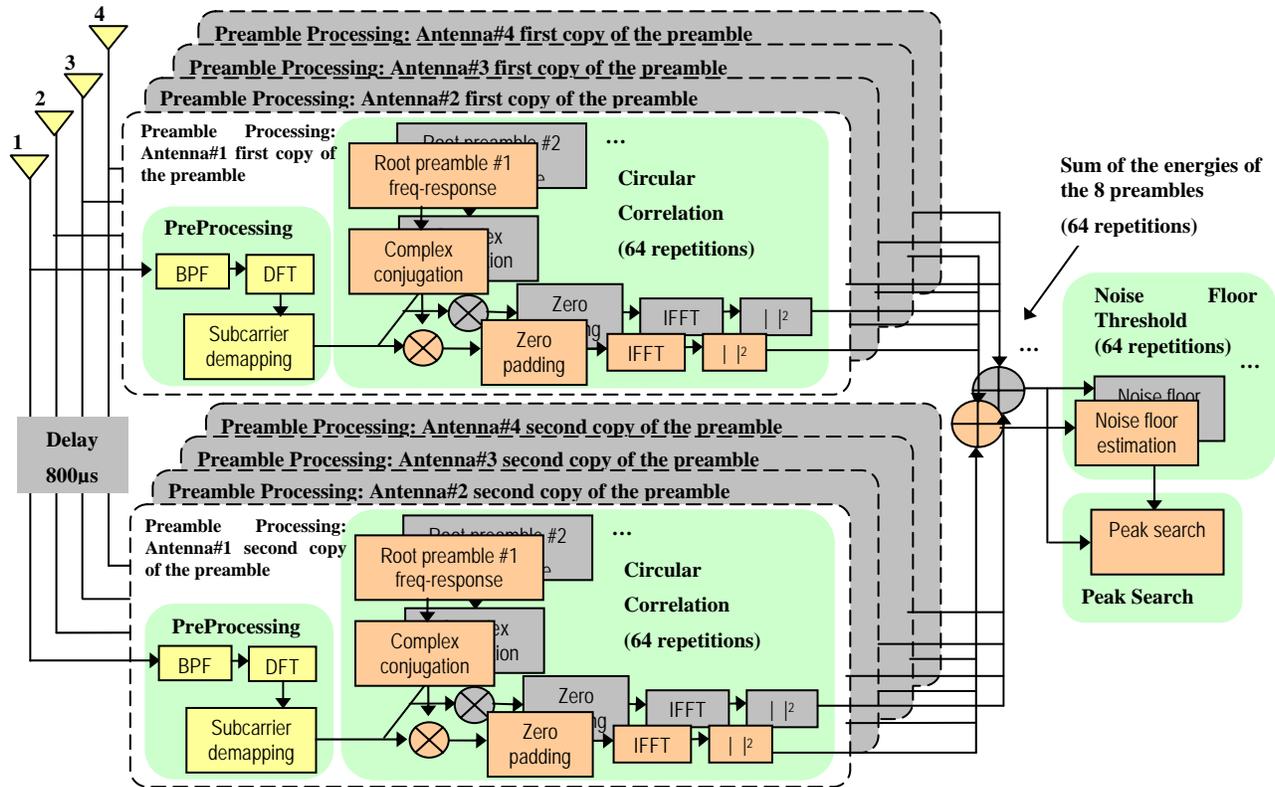


Fig. 1 Random Access Channel Preamble Detection (RACH-PD) Algorithm

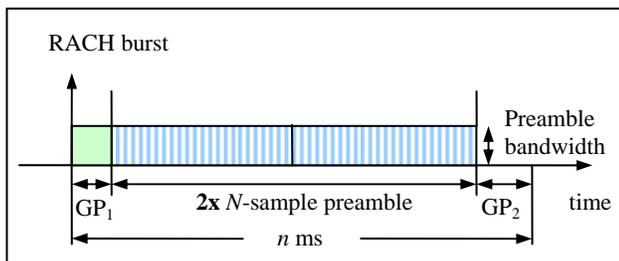


Fig. 2 A Random Access Slot Structure

The preamble is sent over a specified time-frequency resource, denoted as a *slot*, available with a certain cycle period and a fixed bandwidth. Within each slot, a guard period (GP) is reserved at each end to maintain time orthogonality between adjacent slots [1]. This preamble-based random access slot structure is shown in Figure 2.

The case study in this article assumes a RACH-PD for a cell size of 115 km. This is the largest cell size supported by LTE and also the case requiring the most processing power. According to [2], preamble format#3 is used with 21,012 complex samples as a cyclic prefix for GP1, followed by a preamble of 24,576 samples followed by the same 24,576 samples repeated. In this case the slot duration is 3 ms which gives a GP2 of 21,996 samples.

As per Figure 1, the algorithm for the RACH preamble detection can be summarized in the following steps [1]:

- After the cyclic prefix removal, the preprocessing (Preproc) function isolates the RACH bandwidth, by filtering with downsampling and then transforms the data into the frequency domain.
- Next, the circular correlation (CirCorr) function correlates data with several pre-stored preamble root sequences (or signatures) in order to discriminate between simultaneous messages from several users. It also applies an IFFT to return to the temporal domain and calculates the energy of each root sequence correlation.
- Then, the noisefloor threshold (NoiseFloorThr) function collects these energies and estimates the noise level for each root sequence.
- Finally, the peak search (PeakSearch) function detects all signatures sent by the users in the current time window. It additionally evaluates the transmission timing advance corresponding to the approximate user distance.

In general, depending on the cell size, three parameters of RACH may be varied: the number of receive antennas, the number of root sequences and the number of times the same preamble is repeated. The 115 km cell case displayed in Figure 1 implies 4 antennas, 64 root sequences, and 2 repetitions.

III. THE ALGORITHM ARCHITECTURE MATCHING (AAM)

Currently, development tools for processors are primarily based on the C-language and an associated compilation tool. The major issue with a monolithic syntax is the inability to express parallelism. One solution is to use a Real-Time Operating System (RTOS) and to describe threads and their communication links (Mailboxes and pipes). Unfortunately, the application model used in an RTOS is too complex to handle multi-processor architectures when the number of threads increases [3]. For this reason, there is a need to explore methodologies better adapted at expressing the inherent parallelism within the application. Algorithm Architecture Matching (AAM [4]) is an example of one of these methodologies.

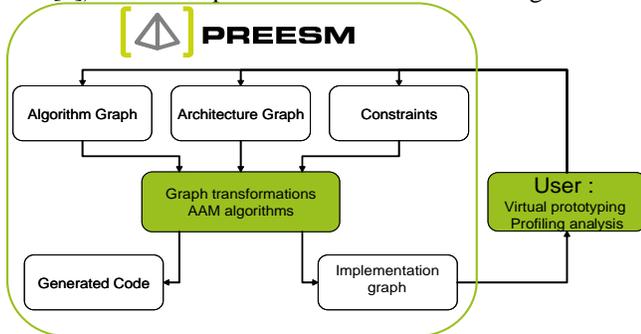


Fig. 3 PREESM Description

Algorithm Architecture Matching (AAM) maps an algorithm to a physical architecture given a set of constraints. The algorithm is described within PREESM using the algorithm graph (Figure 3). It relies on a description model which matches the application behavior. In the case of deterministic systems (including signal, image and communication applications), dataflow graphs have proven to be an efficient representation [5][6] for transformation-oriented systems and heterogeneous multi-component architectures. The algorithm graph (Figure 3) in PREESM is a Synchronous DataFlow graph (SDF) suitable for multi-processor architecture implementations [7]. Each vertex of the SDF represents an operation at coarse grain (equivalent of C function) and each edge represents a data dependency between the two operations at the end vertex. The vertices can be hierarchical, so allowing the description of the application at different resolutions. Thus, the SDF specifies the potential parallelism used in the matching step. The finest resolution vertex is called atomic operation; this type of operation may be described in a programming language such as C, VHDL, C++.

Within the PREESM tool, the architecture is described as the architecture graph (Figure 3) in which vertices represent operators and edges represent communication over a certain medium. An operator in this methodology is usually a processor connected to a local memory and has several communication resources. In this paper, operators are DSP cores and the media is an Enhanced Direct Memory Access (EDMA). The architecture graph specifies the available parallelism.

The matching consists of manually or automatically (AAM algorithms, Figure 3) exploring the implementation solutions with optimization heuristics. These heuristics aim to minimize the total execution time of the algorithm running on the multi-component architecture, by taking into account the execution time of operations and of data transfers between operations. The result of the matching allows automatic code generation [8] for multi-processor architectures handling synchronizations and data transfers between processors. Thus PREESM provides off-line static scheduling for multi-processor architectures. An implementation of AAM using the PREESM tool consists of:

- Performing a distribution (allocating parts of the algorithm to architecture components)
- Scheduling (determining the order for the operations distributed over a component) the algorithm on the architecture.
- Providing an implementation graph including simulation results of the distributed application functions.
- Generating C-code to verify the partitioning on target hardware and to provide a flexible implementation.

These functions enable PREESM to be used as an efficient virtual prototyping tool for our architecture exploration.

IV. ARCHITECTURE EXPLORATION

A. Algorithm Model

The goal of this exploration is to determine through simulation the architecture best suited to the 115km cell RACH-PD algorithm. The RACH-PD algorithm behavior is described as a SDF [3][9] in PREESM. An SDF description brings two major benefits to our implementation. The first is the proven possibility to schedule the algorithm statically. A static implementation enables static memory allocation, so removing the need for runtime memory administration. The second advantage is the high flexibility of communication parameter tuning, as achieved by modifying the SDF.

The RACH-PD algorithm model is shown in Figure 4. Initialization operations on the left-hand side are executed once as the system starts. Next, the three operations PreambleProcess, NoiseFloorThreshold and PeakSearch are executed sequentially in a loop while AntennaGen delivers samples to decode. The PreambleProcess operation is executed four times in each loop iteration; once per antenna. At the beginning of PreambleProcess, the atomic operation Preprocessing executes sequentially the bandpass filter, DFT and subcarrier demapping. It is repeated once for each of the two preamble repetitions. Then the circular correlation with 64 preamble root sequences is performed. Each circular correlation contains the correlation of the two preamble repetitions (SingleZCProc) with power accumulation similar to antenna power accumulation.

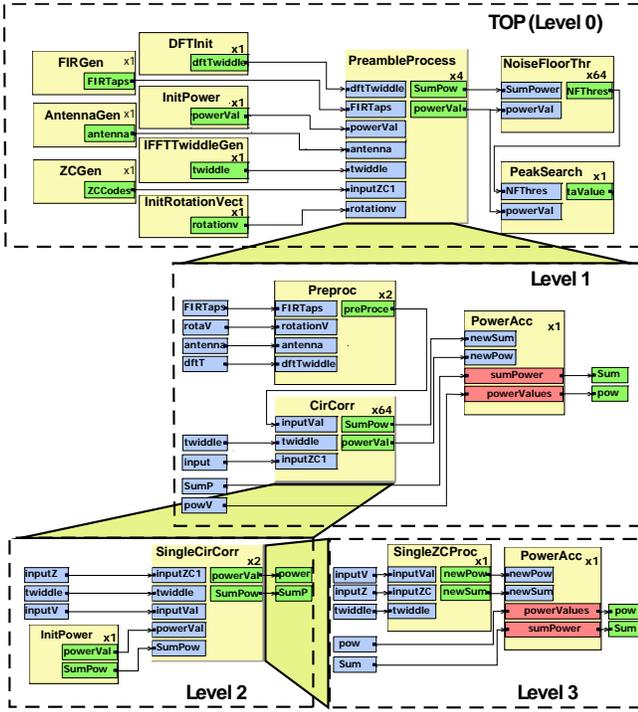


Fig. 4 Preamble Detection SDF Description

Using the same approach as in [10], valid scheduling derived from the representation in Figure 4 can be described by the compact expression:

$$(8Preproc)(4(64(InitPower(2((SingleZCProc)(PowAcc))))PowAcc))(64NoiseFloorThreshold)PeakSearch$$

We can separate the preamble detection algorithm in 4 steps:

- Preprocessing step: $8Preproc$
- Circular correlation step: $4(64(InitPower(2((SingleZCProc)(PowAcc))))PowAcc)$
- Noise floor threshold step: $(64NoiseFloorThreshold)$
- Peak search step: $PeakSearch$

Each of these steps is mapped on the available cores and will appear in the exploration results detailed in Section IV-D. The given description generates 1,357 operations; this does not include the communication operations necessary in the case of multi-core architectures. Placing these operations by hand on the different cores would be greatly time-consuming. The architecture exploration PREESM tool offers an automatic scheduling, avoiding the problem of manual placement.

B. Architecture Exploration

The four architectures explored are shown in Figure 5. The cores are all Texas Instrument TMS320C64x+ DSPs running at 1 GHz [11]. The connections are made via Direct Memory Access (DMA) links. The first architecture is a single-core DSP such as the TMS320TCI6482. The second architecture is dual-core, with each core similar to that of the TMS320TCI6482. The third is a tri-core and is equivalent to

the new TMS320TCI6487 [12]. Finally, the fourth architecture is a theoretical architecture for exploration only, as it is a quad-core. The exploration goal is to determine the number of cores required to run the random RACH-PD algorithm in a 115 km cell and how to best distribute the operations on the given cores.

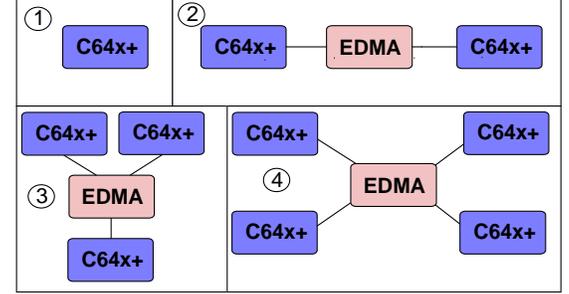


Fig. 5 Four architectures explored

C. Architecture Model

To solve the implementation problem, each operation is assigned an experimental timing (in terms of CPU cycles). These timings are measured with implementations of the atomic functions on a single C64x+. The EDMA is modelled as a non-blocking medium transferring data at a constant rate. Assuming the EDMA has the same performance from the L2 internal memory to the L2 internal memory as the EDMA3 of the TMS320TCI6482, then the transfer of N bytes via EDMA should take approximately (see [13]):

$$transfer(N) = 135 + \frac{N}{3.375} \text{ cycles}$$

The average size of the transmitted buffers in the 115 km preamble detection procedure is 4,800 bytes. Consequently, the average transfer speed used for simulation is 3.08 GBytes/s.

D. Architecture Choice

The PREESM automatic scheduling process (i.e. the application of the AAM methodology to the RACH-PD algorithm) is applied for each architecture. The simulation results obtained are shown in Figure 6. Due to the 115 km cell constraints, preamble detection must be processed in less than 4 ms. Two kinds of experimental timings feed the simulation. The first set of timings is measured in loops, each calling a single function with L1 cache activated and appears as striped bars in Figure 6. It represents the application behaviour when data access is ideal. The second set of benchmarks is measured with L1 cache deactivated and leads to the higher cycles displayed in light grey. It represents the worst case of internal data accesses. For more details about C64x+ cache, see [11]. The RACH application is well suited for a parallel architecture, as the addition of one core reduces the latency dramatically. With L1 cache activated, two cores can process the algorithm within a time frame close to the real-time deadline. Simulation on the dual core with deactivated cache produces significantly higher cycles and misses the real-time deadline, so disqualifying the 2-core solution.

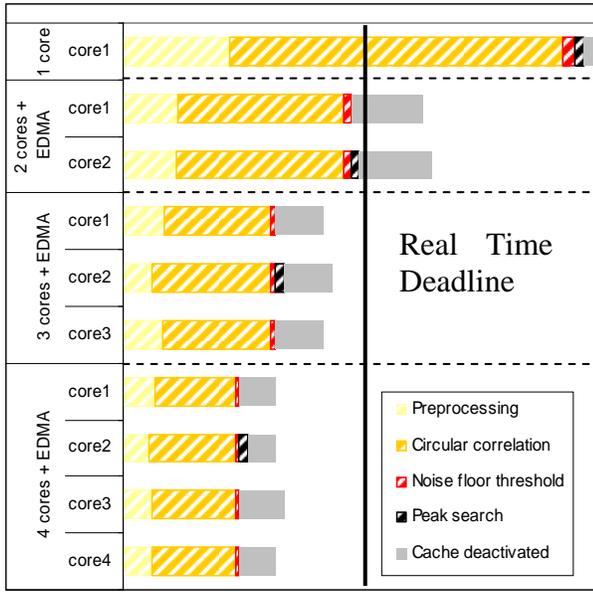


Fig. 6 Timings of the RACH-PD algorithm schedule on target architectures

The 3-core solution is clearly the best one: its CPU loads (68% with realistic cache misses and 88% without cache) are satisfactory and do not justify the use of a fourth core, as can be seen in Figure 6.

V. IMPLEMENTATION ON THE CHOSEN ARCHITECTURE

With the architecture chosen, we can now start the static implementation process. Our goal is to automatically generate a highly optimized and flexible code with the necessary transfers and synchronization.

A. Description of the chosen Architecture

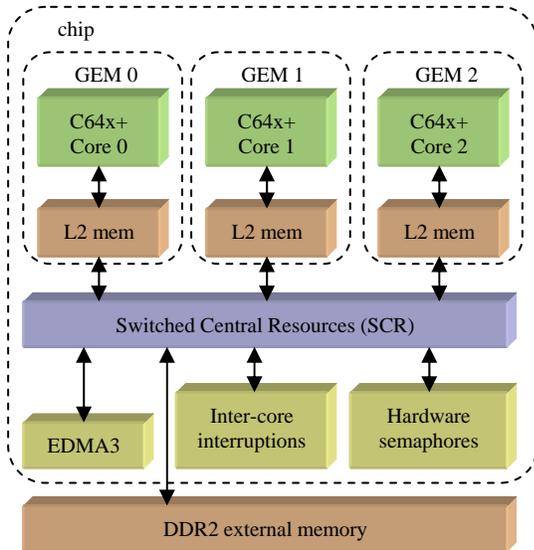


Fig. 7 Architecture of the TMS320TCI6487

The TMS320TCI6487 [12] is a three-core DSP specifically created for communication signal processing. Two modes are available for memory sharing: in symmetric mode, each CPU

has 1MByte of L2 memory while in asymmetric mode, core 0 has 1.5Mbyte, core 1 has 1MByte and core 0.5MByte. Each CPU can access the L2 memory of the two other cores via the EDMA. Each CPU has also access to an external DDR2 memory. The EDMA can transfer a value from one core on-chip L2 memory to another core L2 memory in parallel with CPU calculation. This capability brings a higher flexibility than an architecture with cores interconnected via communication media.

Shared accesses between cores can be synchronized with hardware semaphores and inter-core interruptions. 32 semaphores may interrupt any core when a resource is accessed or released. Inter-core interruptions may be launched from any core by writing in specific registers. Interruptions can carry a 7-bit value to distinguish one from another. Local to each CPU, the RTOS, DSP/BIOS, provides threads and local synchronization between threads with software semaphores. These features will be exploited to implement the RACH-PD algorithm and automatically generate function calls and synchronization. The use of software semaphores is consistent with a high performance implementation as passive wait is generated. Waiting for a DSP/BIOS semaphore puts the CPU in idle state.

B. Using the EDMA as a message passing system

In order to prepare for code generation, we need to develop a communication library which provides synchronization. The communicator interface should be simple and may be called by generated code. The target architecture offers two communication possibilities: queues which are a message passing system built in DSP/BIOS operating system or the EDMA. As the queues are expected to be slower, the choice was made to use the EDMA.

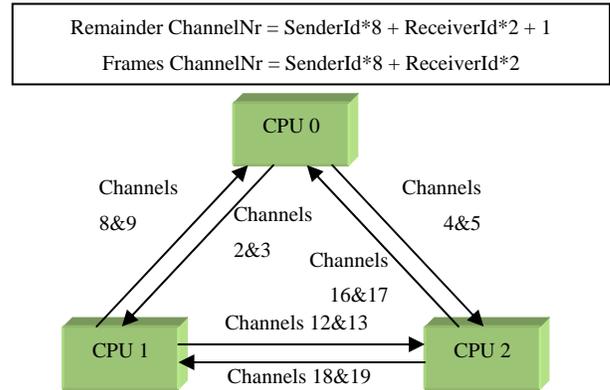


Fig. 8 EDMA channels used for communication between cores

The EDMA module offers 64 channels. Messages are split into N small frames and a remainder. The frames and the remainder are sent on two different chained channels. In order to avoid conflicts, 12 channels are used as shown in Figure 8. Since the channel number contains the sender and receiver identifiers, the receiver always knows, even in interruption routine, the sender of each communication received.

C. Designing the Communication Process

Using the same method as in [6], the PREESM tool generates two threads per core: one for processing function calls and one for sending communication orders and waiting for transfer completion. As shown in Figure 9, when two successive functions are distributed on different CPUs, two semaphores Sem1 and Sem2 are generated on each core to synchronize the processing and communication threads. While communication threads are waiting for the completion of a transfer, processing threads can process data that does not impact this transfer. These local semaphores are implemented with DSP/BIOS operating system.

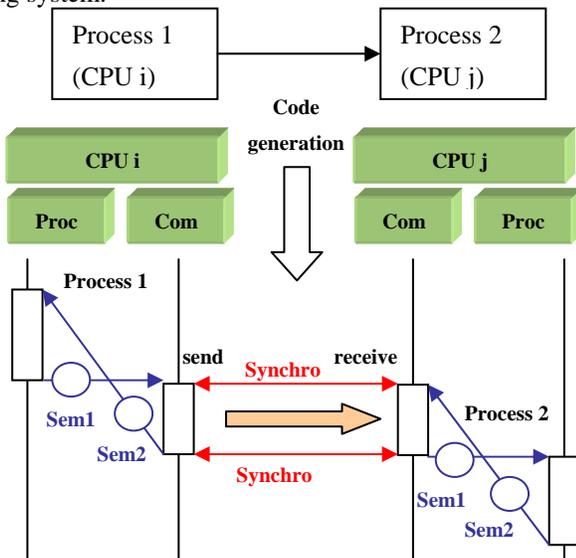


Fig. 9 Threads and synchronization within the cores

The next design problem to solve is the communication between cores. At the beginning of the communication process, the sender alone knows the source buffer address and the receiver alone knows the destination buffer address. There are two solutions (Figure 10) to complete a transfer in this situation: use an intermediate address or transfer the destination address.

When an intermediate address is used, the sender does not need to be aware of the destination address but each transfer must occur twice: from local memory to intermediate and from intermediate to destination. The dimension of the intermediate buffer is also a problem. In Figure 10, only the communication threads of the CPUs are represented.

The second solution is called memory pull because the receiver requests the data by sending its address. This solution imposes a bidirectional communication but is lighter than the preceding solution, as the address transfer of 4 bytes may be achieved through scratch buffers and synchronization through hardware semaphores or inter-core interruptions. During these transfers, we use only 12 of the 256 parameter set registers of

the EDMA. The unused registers can be utilized as scratch buffers to transmit the transfer destination addresses.

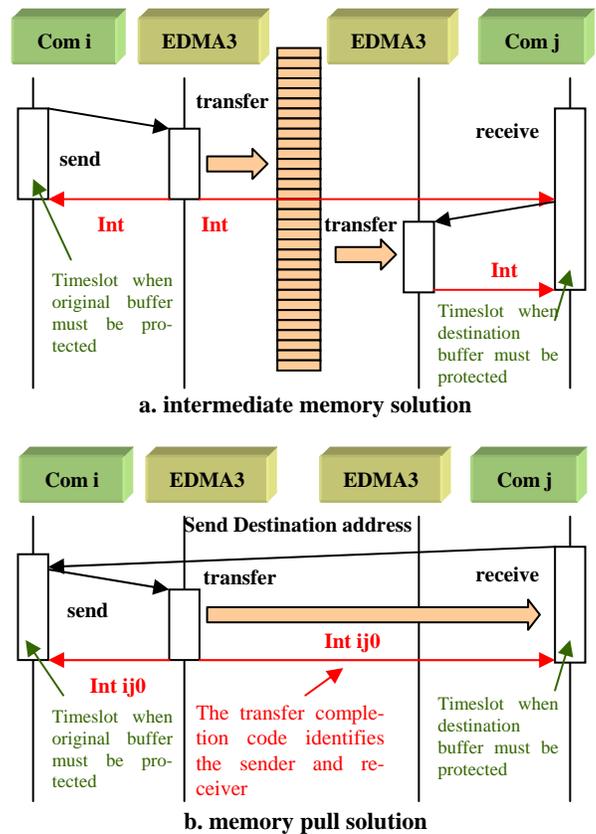


Fig. 10 Two solutions for the inter-core communication

The memory pull solution is chosen for two reasons. Firstly, the intermediate memory solution imposes the static allocation of an additional buffer of size at least as large as the largest one in the algorithm. In the RACH-PD case, this buffer should be of a minimum size of 100 kBytes with strong memory optimization. Secondly, the double transfer results in a division by 2 of the communication speed. Even with rates of several GBytes/s, this data rate reduction is not negligible. Simulation shows a communication cost of almost 5% for the RACH-PD algorithm.

As stated previously, two different modules may be used to generate inter-core synchronization: hardware semaphores or inter-core interruptions. In order to simplify the synchronization, a small library is created, with an interface close to the one of the local semaphores:

- HardSEMPend(int id): Waits for a semaphore with the given semaphore identifier
- HardSEMPPost(int id): Launches a semaphore with the given semaphore identifier.

The semaphore identifier is chosen to be the Sender CPU number. Hardware semaphores are designed to protect critical sections from multiple accesses. They are typically used in resource access requests; access is granted when a semaphore is acquired. Programming the synchronization library with such a system leads to a clumsy implementation where the acquired resource is purely virtual. It is for this reason that the choice was made to base the library on inter-core interruptions. The principle of a inter-core synchronization library based on interruptions is quite simple. Any core can send an inter-core interruption to another core by providing the right identifier. An interruption is launched by the sender `HardSEMPPost` function and is caught in an interrupt service routine of the receiver. The receiver then releases a local semaphore that was pending in the `HardSEMPend` function.

When the sender receives an interruption indicating the availability of the address, it launches the EDMA copy and waits. At the end of the transfer, the EDMA launches an interruption giving a transfer completion code equal to the EDMA channel number. The sender and receiver identifiers are deduced and communication threads of both CPUs are released.

When the cores are started, each core waits for the two other cores to run before sending any interruption. This system ensures that no inter-core interruption is ignored.

D. RACH-PD memory consideration

Once the communication model is complete, our attention turns to memory. The data buffers are statically allocated, some in the fast L2 memory of their CPU and others in the huge DDR2 external memory. We thus need to decide which buffers should be allocated in L2 and which configuration (symmetric or asymmetric) should be chosen. For the buffers in DDR2, sections of L2 memory must be used as cache for DDR2 so that performance does not decrease dramatically (see [14] for more details). Thus, we activate L2 cache with its maximal size of 256kBytes to improve DDR2 access time.

When the multi-core program is run with data in external memory, the EDMA reads and writes DDR2 data cached in L2. Cache coherency must then be taken into account. Indeed, the EDMA module runs the risk to read “dirty” data or to write in a cached value. Before sending data, a cache “write-back” is called to retrieve the data from cache. Before receiving data, a cache “invalidate” is called to mark the cache value as obsolete. With these precautions, cache coherency is maintained.

E. Implementation tests

When the communication programming is complete, we generate a 3-core code for C64x+ with the PREESM tool. We initially test a simple PREESM project by copying a buffer from one core L2 to another. The benchmarks of these copies are shown in Figure 11. The fixed overhead of an inter-core copy is approximately 2,700 cycles. This overhead is due to the synchronization process, the interruption routines and the EDMA configuration. When transferring big buffers, the

EDMA data rate reaches 1.6 GBytes/s, half the speed of the TMS320TCI6482 EDMA employed in the simulations and benchmarked in [13].

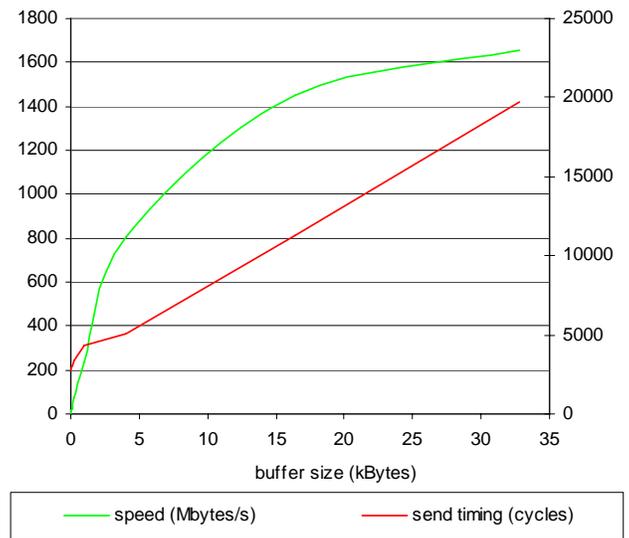


Fig. 11 Benchmarks of TCI6487 synchronized EDMA3 buffer copy

The RACH-PD algorithm is then tested and optimized on the TMS320TCI6487 platform in several steps:

1) *Single-core test*: A first version is tested on one core with code in L2 and data in DDR2. This implementation allows us to debug, dimension the stack and ensure that the code is working. The time of one preamble detection with this configuration is 240 ms.

2) *Cache activation*: L2 cache is then activated with the maximal size of 256 kBytes. The preamble detection time is then reduced to 69 ms.

3) *Three-core implementation*: The application is distributed over three cores. When we let the PREESM tool choose a strongly parallel implementation without constraints, it generates a very complex solution with 10,000 semaphores. We thus tune this solution, reducing inter-core communication and still allowing good pipelining (see Figure 12). The number of semaphores is then reduced to approximately 100. The complex solution which results from the non-constrained operation shows a limitation in the present PREESM mapping: a fixed cost for transfers needs to be added to the tool to avoid the explosion of communication. The non-constrained PREESM automatic code generation allocates buffers of approximately 1.65 Mbytes for one core, 1.25 Mbytes for a second core and 200 kBytes for a third core, to which the heap and the code size must be added. This asymmetry justifies the use of asymmetric memory. With this configuration, one preamble detection takes 50 ms.

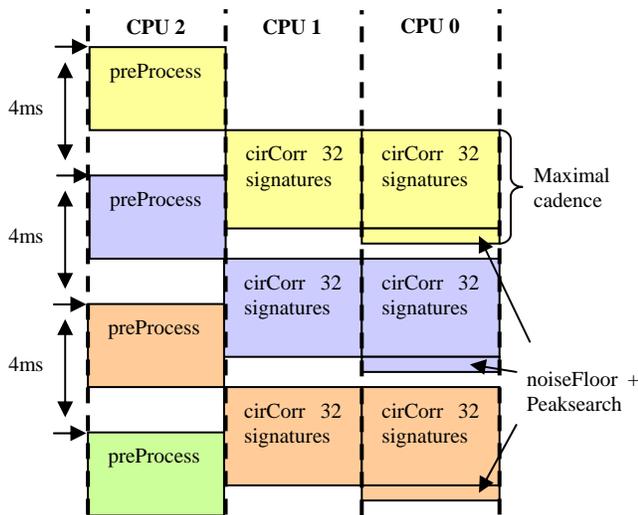


Fig. 12 Simple pipelining of the RACH-PD decoding

VII. CONCLUSIONS

The intent of this paper was to demonstrate a methodology using rapid prototyping and automatic code generation to develop an optimized multi-core implementation of a communication algorithm. After exploring 4 solutions, the best target architecture for the 115km cell RACH-PD algorithm was chosen, and an implementation is described and benchmarked. Memory allocation, function calls and EDMA calls are generated in C-code by the PREESM tool. Inter-core communication and memory partitioning are considered in the prototyping methodology. The result is an efficient and highly reconfigurable implementation, proving that the generation of static implementations from SDF descriptions is a viable solution for deterministic signal processing applications.

In the near future, an increasing number of CPUs will be available in complex System on Chips. Developing methodologies to efficiently partition code on these architectures is thus an increasingly important objective.

REFERENCES

- [1] J. Jiang, T. Muharemovic, P. Bertrand, and S. Aridhi, *Random Access Preamble Detection for Long Term Evolution Wireless Networks*, submitted to SIPS 2008.
- [2] *3GPP technical specification group radio access network; Evolved universal terrestrial radio access (E-UTRA); Physical channels and modulation (Release 8)*, 3GPP, TS36.211 (V 8.1.0)
- [3] E. A. Lee, *The Problem with Threads*, EECS in IEEE Computer 39(5):33-42, May 2006.
- [4] S. Moreau, S. Aridhi, M. Raullet, J.-F. Nezan, *On Modeling the RapidIO communication link using the AAA Methodology*, DASIP 2007.
- [5] Shuvra S. Bhattacharyya, *Ptolemy*.
- [6] T. Grandpierre and Y. Sorel, *From algorithm and architecture specifications to automatic generation of distributed real-time executives: a seamless flow of graphs transformations*, In First ACM and IEEE International Conference on Formal Methods and Models for Co-Design, Mont Saint-Michel, France, June 2003.
- [7] J. L. Pino and E. A. Lee, *Hierarchical static scheduling of dataflow graphs onto multiple processors*, In IEEE International Conference on Acoustics, Speech, and Signal Processing, Detroit, Michigan, USA, 1995.
- [8] M. Raullet, F. Urban, JF Nezan, O. Deforges, C. Moy, *SynDEX Executive Kernels For Fast Developments of Applications over Heterogeneous Architectures*, EUSIPCO 2005.
- [9] E.A. Lee and D.G. Messerschmitt, *Synchronous Data Flow*, IEEE Proceedings of the IEEE volume 75, numéro 9, 1987.
- [10] S. S. Bhattacharyya, E. A. Lee, *Memory Management for Dataflow Programming of Multirate Signal Processing Algorithms*, IEEE Trans. on Signal Processing, 42, No. 5, May 1994, pp. 1190-1201., 1994.
- [11] *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide*, Texas Instrument technical document (SPRU732G), February 2008.
- [12] *TMS320TCI6487 DSP Platform*, Texas Instrument Product Bulletin (SPRT405).
- [13] B. Feng, R. Salman, *TMS320TCI6482 EDMA3 Performance*, Texas Instrument technical document (SPRAAG8), November 2006.
- [14] *TMS320TCI6487/8Communications Infrastructure Digital Signal Processor*, Texas Instrument technical document (SPRS358E), 2008.

VI. FUTURE WORK

In the near future, a new communication model for the TMS320TCI6487 will be built based on message queues from DSP/BIOS operating system. This model will then be compared with that based on the EDMA. Its advantage will be the portability on devices using RapidIO (see [4]), as the operating system DSP/BIOS can use this communication system to pass messages.

Furthermore, other algorithms of LTE will be studied and implemented. It is expected that this will help to improve the architecture models of the PREESM tool. Specifically, internal/external allocation and advanced timings taking into account data caching may be automated, thus eliminating the manual step of memory allocation. Additionally, a more accurate communication model in the PREESM tool will remove the need for the manual reduction of semaphores.

Memory Management for Octree-like Structure Traversal

Tomasz TOCZEK and Stéphane MANCINI
GIPSA-lab, INPG-CNRS,

961 rue de la Houille Blanche Domaine universitaire – B.P. 46, 38402, Saint Martin d’Hères, France

Email: toczek@lis.inpg.fr, stephane.mancini@grenoble-inp.fr

Abstract—This paper presents an architecture for ray casting through recursive grids, a generalization of the concept of octree. The main feature of our architecture is to reduce the needs in bandwidth to external memory by exploiting the spacial and temporal locality of coherent sets of rays. To this end, we use a recursive grid structure aware adaptive and predictive cache, associated with a specifically crafted “phase locked” ray propagation algorithm. Our findings are that the use of recursive grids, which allow significant reductions in terms of required storage space and ray traversal steps in locally sparse scenes, are fully compatible with on-the-fly memory prefetching mechanisms as implemented by our cache.

I. INTRODUCTION

Ray shooting is a well known problem consisting in computing ray paths through diversely structured scenes ranging from sets of primitives such as triangles to density fields [1]. Algorithms involving ray shooting are called ray tracing or ray casting algorithms, and have applications such as realistic rendering or tomographic reconstruction. Despite their great results, most of those algorithms are considered as very costly computationally.

For the past two decades, ray shooting acceleration heuristics have been a hot research topic. The ray traversal of so-called acceleration structures (AS) accounts for a lot of the research done in this field. It consists in finding a way to represent the scene – the AS –, as well as an associated traversal algorithm, which allow the best average ray shooting performances. This problem can be solved in a variety of ways, depending on the kind of scene one is willing to represent, and on the kind of computational resources one is trying to spare.

Acceleration structures tend to work by subdividing in some way or another the space the scene is located in. There are at least two ways to build and use an acceleration structure. The first is to consider the AS as a simple space indexing mean for the scene primitives. Examples of this approach include SAH-based *kd-trees*, bounding box hierarchies, and so on [2]. Alternatively, some AS can be thought of as space partitions. When that is the case, it is possible to consider the cells of the AS as an approximation of the scene geometry. Generally, such approaches advocate the use of cubic cells, called “voxels” (volume pixels). Uniform 3d grids, as well as octrees [3], can be used in this way.

To our mind, from a hardware designer point of view, acceleration structures ought to be assessed through a too-seldom-thought-of criterion, which is the quantity of im-

PLICIT information it holds regarding the scene geometry. For instance, a uniform 3d grid holds a lot of such implicit information, since the memory address of a cell of the grid can be computed directly knowing the cell position, whereas structures such as *kd-trees* hold little such information, since finding a cell requires a tree search. Of course, even highly irregular structures tend to preserve spatial locality. While this allows those structures to be used with generic caches at the cost of (more or less acceptable) performance penalties, it makes the design of an efficient application-specific prefetching based cache virtually impossible for them.

Quite clearly, the use of uniform grids seems to be an obvious choice if one’s trying to design such a cache, but sadly, those have little practical interest. Indeed, they grow very large with the need for small cells, which makes them not only impractical to store in memory, but also costly to traverse. Real world scenes, however, often exhibit both large homogeneous areas and small very detailed parts. The use of hierarchical space subdivision structures allows the efficient representation and traversal of such scenes.

In this article, we introduce $2^n \times 2^n \times 2^n$ recursive grids, a generalisation of octrees very similar to [4], yet a bit more constrained. Recursive grids allow hierarchical non-homogeneous space partitioning and traversal by beams of rays. We propose a recursive grid cache allowing pre-fetching of scene data as needed by our traversal pipeline. Our cache offers good performance while being suited to large scene representations, an aspect often neglected in the current research effort.

In the section II, we briefly describe some of the hardware designed so far for memory management for ray shooting. In the section III, we present our architecture, which includes the above mentioned cache and traversal pipeline. Finally, we discuss its performances in the section IV and conclude in the section V.

II. STATE OF THE ART

A. Generalities on parametric grid traversal

Among the methods used for acceleration structure traversal, the parametric ones are very popular. They are conceptually easy and their implementation is often efficient. Most of them are variants of the DDA algorithm adapted to ray tracing [5], which parametrizes the ray and performs all the computations in the parameter space thereafter. The Fig. 1 illustrates the variables used by the DDA on a 2D example.

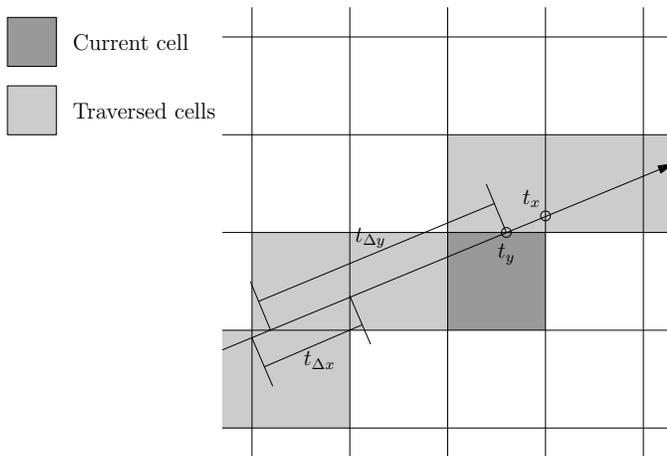


Fig. 1: Geometrical meaning of the variables used by the DDA algorithm

The algorithm iterates on the cells of an uniform grid, storing the parameters of the intersection between the ray and the yet-uncrossed cell faces' planes along each axis (called t_x and t_y on the Fig. 1). It comes the next cell is the neighbour of the current one, sharing the face corresponding to the smallest of the t_x and t_y parameters (that is, t_y in our example). This smallest parameter t_l (where $l \in \{x, y\}$) is updated by being added $t_{\Delta l}$, the parameter difference between the intersection points with the ray of the two faces of a cell orthogonal to a given axis l . In our example, it comes the next cell is the one just above the dark gray one, and the new face intersection parameters are $(t'_x, t'_y) \leftarrow (t_x, t_y + t_{\Delta y})$. The $t_{\Delta x}$ and $t_{\Delta y}$ parameters are computed once for all for a given ray at the initialisation on the DDA algorithm. The resulting traversed nodes are shown in light gray on the Fig. 1.

When performing ray casting (that is, a particular case of ray tracing where one ray matches exactly one pixel), the contribution¹ of each traversed cell is taken into account for the resulting pixel value computation. This is called compositing.

B. Memory management

Most of the ray shooting dedicated hardware design with an emphasis on efficient memory management was done in the field of volume rendering. This can be explained by the fact that volume rendering naturally involves very large data sets².

The *Cube* series is an example of regular-grid-sampling-based volume rendering hardware designed to spare this very bandwidth. While *Cube-3* [6] is a regular ray parallel ray tracing architecture, *Cube-4* [7] is based on object-order ray-tracing. The *Cube-4* hardware is designed so as each voxel is fetched exactly once per frame from the central memory. Therefore, it is optimally efficient in terms of memory bandwidth usage, if we admit that every voxel accounts for each frame. However, *Cube-4* has a number of severe limitations,

¹which may be emitted or re-emitted light (rendering), density (PET reconstruction), attenuation (X-ray based reconstruction), ...

²for instance, up to 1024^3 grids with recent medical appliances

one of them being a scene size limited to 256^3 equally sized voxels. Moreover, the very principle of object-order raytracing upon which *Cube-4* was built makes perspective rendering implementation impractical; that is why VolumePro, a commercial implementation of *Cube-4*, only supports isometric rendering. Several other problems were underlined by [8].

VoxelCache [9] is a cache specifically crafted for sampling based ray casting, as well. It is small enough to be implemented on reconfigurable hardware. It uses only a single external memory bus, but has an internal 8 memory bank organisation. This makes possible to fetch a tri-linearly interpolated sample every cycle. VoxelCache also has a prefetching mechanism, requiring that beams of 4×4 coherent rays are being shot. Despite the fact it was designed for uniform grids, VoxelCache was successfully used for full-octree-based volume rendering as well [10]. This however required the use of off-chip SRAM to keep the performances high, and the caching strategy was shown inefficient for large scenes due to cache trashing. On the contrary, we tried in our approach to use as much as possible inexpensive components found on most prototyping boards, while focusing on arbitrary large sparse octree-like AS, much more flexible than full octrees.

C. Coherent ray shooting

Coherent ray shooting has been an intensive research topic as well, and is most often presented as a generic mean to increase memory access temporal locality without changing the underlying memory organisation. Most often, it involves "beams" of coherent rays which are traced through the scene all at once. Several approaches have been proposed, focussing on exploitation of SIMD parallelism in processors [11], constructing a simplified version of the AS by discarding parts not intersecting with the beam [12], or merely suggesting the brute force shooting of all the rays of a beam in a parallel fashion. Most of those studies suggest strong performance improvements due to the increased spatial locality of memory accesses [13].

III. ARCHITECTURE

The architecture we propose is composed of two main parts: an adaptive and predictive cache for recursive grids and a traversal unit, capable of determining the sequence of recursive grid cells traversed by each ray of a coherent beam. The generated sequences are meant to be communicated to a compositing unit. Once a ray of the beam ends, its accumulated value may be written to a frame buffer. The Fig. 2 presents an overview of a whole rendering architecture as could be implemented on a prototyping board. The traversal unit and the cache were designed and synthesized with the Xilinx Virtex 4 technology as a target.

The compositing unit can be implemented in a variety of ways; for our tests, we used voxel-based volume rendering, considering each voxel as having uniform density, and integrating this density along the ray as a compositing rule. We could have used virtually any front-to-back compositing method instead. Also, the compositing unit can be turned into a

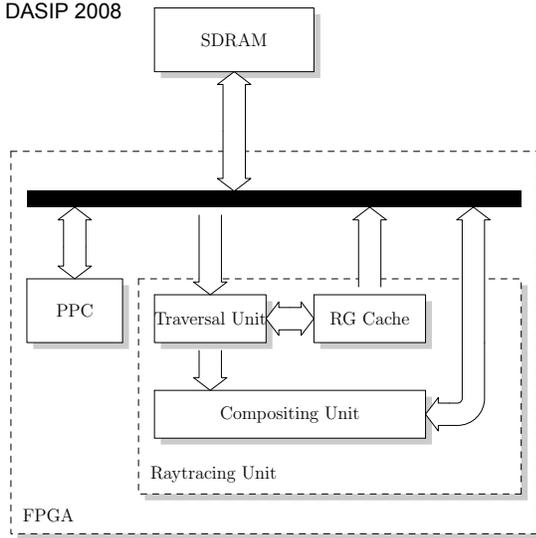
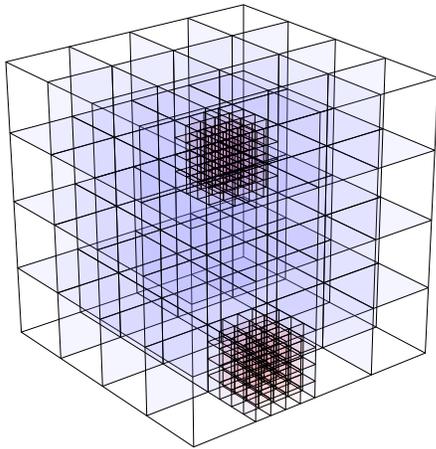


Fig. 2: An overview of a complete rendering system

Fig. 3: An example $4 \times 4 \times 4$ recursive grid

primitive intersection test unit when our acceleration structure is used as a space indexing mean.

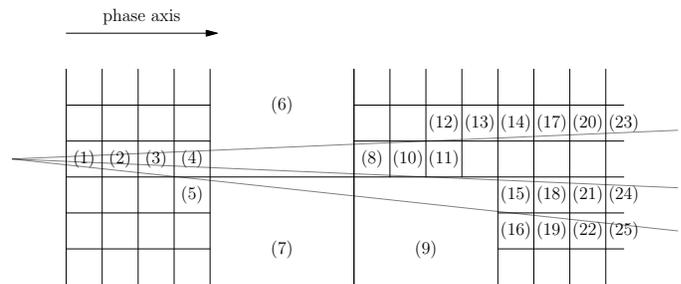
A. Recursive Grids

The Acceleration Structure we chose to use for our ray casting hardware is a simple generalisation of octrees [3] we call recursive grids (for the lack of an established name). A $2^n \times 2^n \times 2^n$ recursive grid is a tree with the following characteristics:

- each node is a cube
- each internal node has 2^{3n} equally-sized children

As a consequence, every node of the recursive grid has the same size as any other node on the same depth. We call the leaf nodes voxels. The Fig. 3 shows a $4 \times 4 \times 4$ recursive grid, with two non-voxel children at the root node. It is clear that, under our formulation, octrees are $2 \times 2 \times 2$ recursive grids.

$2^n \times 2^n \times 2^n$ recursive grids, especially for “moderately large” values of n such as 2 or 3, hold several interesting properties few other hierarchical acceleration structures do. While still having the benefits of sparse hierarchical structures, they also offer more regularity than most of them, allowing

Fig. 4: Expected voxel access order for a phase-locked tracing of a beam of 3 rays in a 4×4 recursive grid (2D example)

more efficient caching. It can be noticed that recursive grids are a subset of adaptive grids [14], and hence inherit of most of their advantages when it comes to their traversal.

For our tests, we chose to encode a $4 \times 4 \times 4$ recursive grid by an array of at most 32768 nodes, the node 0 being the root. The exact coding of a node is itself that of an array of 64 words of 16 bits, each coding for a child. Each of these 16 bit words is divided into a 15 bit datum and a 1 bit flag. The flag determines if the child is divided or not. When set, the remaining 15 bits are the index of the child node. Otherwise, they code for the child density. This 15 bit range is adapted to the representation of, for instance, reconstructed MRI volumes. With this encoding, an internal node size is exactly 128 bytes. While it could be argued that the node count limit is very restrictive, it is also clear that such a limitation can be easily removed without significantly increasing the size of each node by adopting a more complex encoding, with “implicit pointers”, relative addressing, alignment constraints on child nodes located far enough from their parent, variable node size, and so on. All of those techniques make the building of the tree a bit more complex, but are generally simple enough to result in a very little combinatorial overhead on the hardware.

B. Hierarchical phase-locked propagation principle

We use a phase-locked propagation principle similar to the one exposed in [13], in order to keep the accesses coherent and the cached zone minimal.

More specifically, for each beam of rays, we pick a *phase axis*, a major axis along which the rays propagate quickly³. Once that axis chosen, we propagate in a way that ensures the cell accesses during traversal are ordered by their coordinate on the phase axis. An example of such an access sequence is given on Fig. 4. The section III-D.2 gives more detail as how this can be implemented.

Since it is known that the *phase* (ie., access coordinate along the phase axis) moves in only one direction and that a lot of consecutive accesses will request cells sharing the same phase, we can afford to cache only a very narrow slice of the scene along the phase axis. Along the two other axes, the cached

³for example, by picking the axis with the greatest average of absolute values of dot products between unitary direction vectors of the rays of the beam and an unitary direction vector of that axis

zone needs to be just broad enough to contain all the rays of the beam.

C. Recursive Grid Cache

The RG Cache, described in Fig. 5, aims at caching a part of the recursive grid by exploiting the spatial coherency of references. Furthermore, it provides a virtual interface to the processing unit. This means that the processing unit issues a 3D coordinate (x, y, z) together with a resolution level and the cache provides the corresponding datum, preventing the processing unit to manage the tree structure. The RG cache uses the nD-AP Cache [13] as a basic block.

The proposed strategy is to cache each level of resolution with a nD-AP Cache and to perform prefetching in the tree. Indeed, when a reference occurs, it is likely that the next reference is at a close coordinate, either at the same, upper or lower resolution. Each nD-AP cache is in charge of tracking references at a resolution and the neighboring ones (see [13] for details about the nD-AP Cache behaviour).

The tree manager (TM) unit returns parts of the scene requested by the nD-AP caches and maintain a coherent state of the caches at different resolutions. Each time a cache requests a part of the scene, the TM reads the corresponding data at the upper level to determine if the requested block is a leaf or a child node. In the later case, the TM fetches the data at the obtained address to fill the nD-AP Cache. As a consequence, the nD-AP Cache is slightly modified to allow the TM to read a nD-AP Cache concurrently with reads at the processing unit interface. Also, the cached zone at level n has to be inside the cached zone a level $n - 1$ to maintain cache coherency. Without this constraint, when the cache n would request a part out of the zone in the $n - 1$ cache, it would be too slow to get the data by traversing the tree from the root node.

Each of the cache is optimized to manage data at its level of resolution. The size of embedded cache memories fits the level of resolution to save space. For example, because the level 1 cache contains only $4 \times 4 \times 4 = 64$ data, it doesn't need trackers and has a simplified control management.

D. Traversal Unit

We implemented a hardwired traversal unit for efficient ray shooting through recursive grids. It works on beams of up to 256 rays, this limitation being easy to overcome if need is. The Fig. 6 shows its structure. Its three main parts are:

- the phase-locked beam propagation unit, which determines the order in which grid nodes are fetched so as to minimize the probability of cache misses
- the cache interface, which performs cache requests while pipelining ray parameters linked to the request
- the neighbour finding unit, which determines the next node a ray should access and its updated parameters, based on the response from the cache and the former ray parameters

The cache interface is trivial to implement. It contains two ray state holding FIFOs, which minimize the impact of small

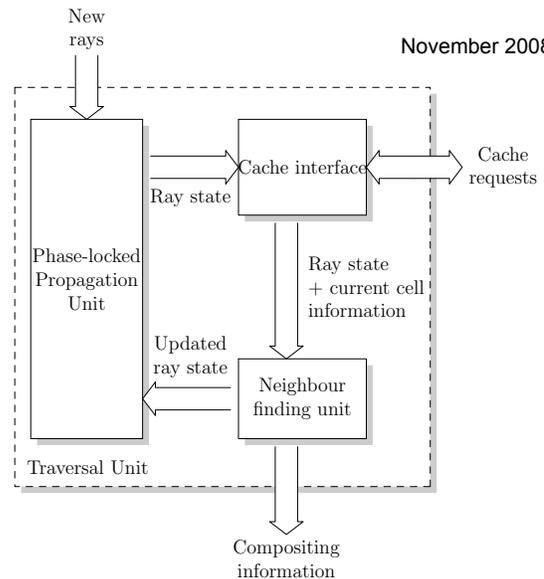


Fig. 6: Architecture of the traversal unit

```

ray_state.r ← unitary direction vector of our ray
ray_state.t ← current cell entry point parameter
ray_state.(t_x, t_y, t_z) ← border intersection parameters
ray_state.(t_Δx, t_Δy, t_Δz) ← parameter increments
ray_state.(p_x, p_y, p_z) ← current cell absolute position
ray_state.depth ← depth of the current cell
max_depth ← maximum depth of the tree

```

Listing 1: Variables characterizing a ray state

pipeline bubbles. Beyond 3 or 4 elements, the size of those FIFOs has little consequence on performance. Therefore, in the rest of this section, we will focus on the two other components.

1) *Neighbour finding unit*: We use a slightly modified version of the DDA algorithm for neighbour finding, in a fashion very similar to that presented in [5]. As we need to be able to vertically traverse the tree as well, we adapted the principles exposed for octrees in [15] to $2^n \times 2^n \times 2^n$ recursive grids, by simply considering each of our nodes as a n -level perfect octree. On a side note, this approach works for adapting pretty much any algorithm working on octrees to recursive grids, and without increasing its asymptotic cost.

To sum everything up, let us consider that a ray propagation state is fully characterized by the variables of the Listing 1. The “parameters increments” are the differences between the parameters of the intersection points between the ray and two opposite faces of our cell, for each of the three possible such pairs of faces. The absolute position of the current cell is given in units corresponding to the size of cells located at a given maximum depth (the constant `max_depth`). If this depths is 6, for instance, the maximum detail level of a $4 \times 4 \times 4$ recursive grid will be the same as that of a 4096^3 uniform grid.

Let's suppose without loss of generality that our ray propagates in the positive direction along each axis⁴. Our neighbour

⁴Indeed, if it is not the case along one or more axes, we can bring ourselves back to the case where it is by taking as absolute cell position the one's complement of the actual cell position along those axes. Of course, the “correct” position must still be used for the memory accesses. This strategy is suggested in [15], where the reader may find extensive detail of such an approach.

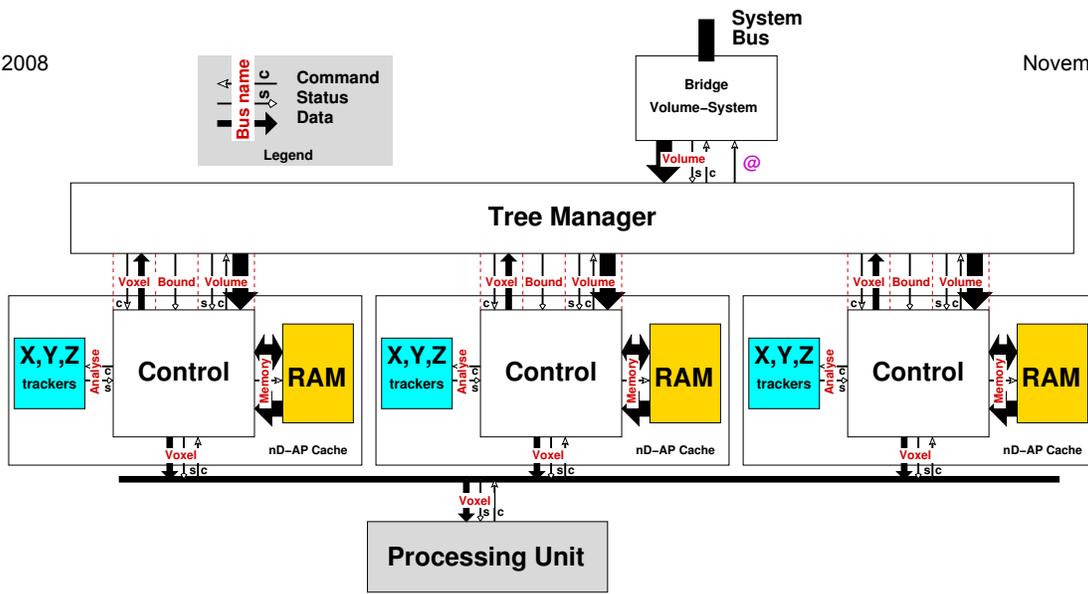


Fig. 5: Recursive Grid (RG) Cache

```

if (current cell is a voxel) then
    // We need to find the next cell (on the same
    // depth or above); determining which direction
    // the next cell is:
    k | tk = min(tx, ty, tz)

    send current segment to the compositing unit

    // Then, undiving as far as necessary:
    while pk[(h-1)..(h-n)] = '1..1' with h=n*(max_depth-depth)
        if depth > 0
            depth ← depth - 1
        else
            traversal is over for current ray
            for m in 0 to (n - 1)
                h ← n * (max_depth - depth) - (m + 1)
                foreach l ∈ {x, y, z}
                    if pl[h] = '1' then
                        (tl)max ← (tl)max - tΔl
                        tΔl ← 2 * tΔl

    // Finally, advancing to the next cell:
    pk ← pk + 2 ** (n * (max_depth - depth - 1))
    (tk)max ← (tk)max + tΔk
else
    // We need to dive further
    for m in 0 to (n - 1)
        h ← 2 * (max_depth - depth) - (m + 1)
        foreach l ∈ {x, y, z}
            tΔl ← tΔl / 2
            (tl)max ← (tl)max - tΔl
            if t > (tl)max then
                (tl)max ← (tl)max + tΔl
                pl[h] ← '1' // hth bit of pl to 1
            else
                pl[h] ← '0' // hth bit of pl to 0
        depth ← depth + 1;
    
```

Listing 2: Neighbour finding algorithm

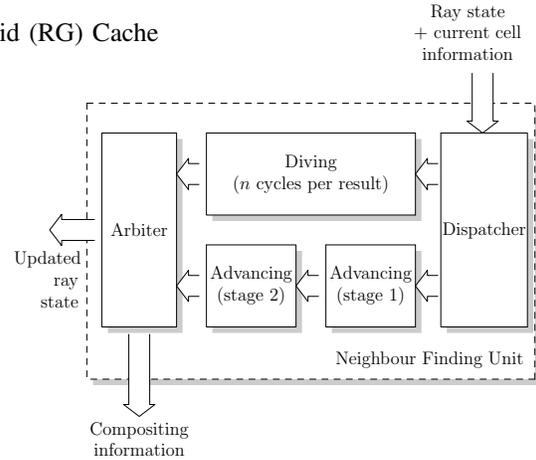


Fig. 7: Neighbour finding unit

finding algorithm for $2^n \times 2^n \times 2^n$ recursive grids is then summarized by the pseudo code of the Listing 2. This code corresponds to the processing needed for each memory access. The top level “if” conditional shows our neighbour finding unit has two different modes: advancing and diving. We have designed our hardware in such a way as the advancing part (then clause) has a throughput of one ray per cycle, while the diving part (else clause) generates one result every n cycles.

This implementation choice is justified by the fact that the probability of depth change during a neighbour finding step is about only $\frac{1}{n}$ for $2^n \times 2^n \times 2^n$ recursive grids, making the else clause evaluation less frequent. The Fig. 7 presents the organization of our unit. As the diving and advancing modes are mutually exclusive, there is hardware reuse between them which does not appear on this figure for the sake of clarity.

One should pay attention that the input and output data of the neighbour finding unit may grow moderately large depending on its exact coding. What call *ray propagation state* a structure composed of the variables seen on Listing 1, at the exception of the direction vector of the ray (which does not matter for the traversal assuming all the other variables are known). As max_depth is a constant, it requires no storage space either. Hence, storage of a propagation state for one ray requires:

- seven times the quantity of data needed per parameter (for $t, t_x, t_y, t_z, t_{\Delta x}, t_{\Delta y}, t_{\Delta z}$). Depending of the desired representation and precision, this quantity of data may vary. Since we use projective geometry [13], we have 2×16 bits per parameter. One could also use integer arithmetic or a floating point representation.
- three times the quantity of data needed par absolute voxel

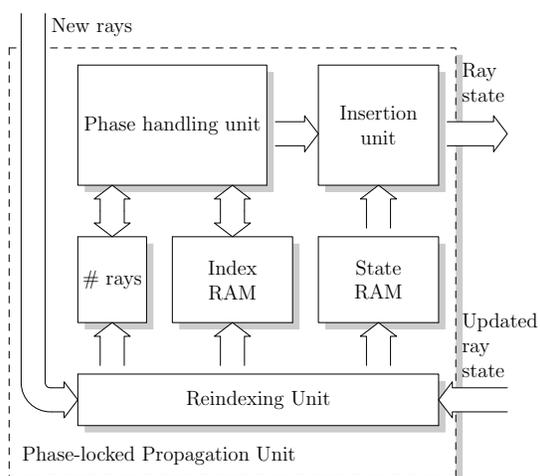


Fig. 8: Architecture of the phase-locked propagation unit

- coordinate at maximum depth (ie. $3 \times n \times \text{max_depth}$ bits)
- $\lceil \log_2(\text{max_depth}) \rceil$ bits for depth
- 3 bits to store the direction of propagation along the three major axes

Assuming $\text{max_depth} = 5$, $n = 2$, and 32 bits per parameter, we need 260 bits per ray state. For each ray, those parameters need to be initially computed from the ray geometry before they are fed to the propagation unit. It involves obtaining the intersection points parameters between the ray and each of the faces of the root node cube. This can be done by an on board microprocessor while the ray shooting unit is busy.

2) *Phase-locked ray beam propagation*: The phase-locked propagation unit, as shown on Fig. 8, holds the states of all the rays of the current beam in such a way it is able to ensure they all propagate with the same speed along a given major axis, which we call the phase axis. This phase axis is the one which is the closest to the direction of the ray, and is pre-computed at the same time the initial states of the rays of the beams are generated (typically, by a hard or soft processor).

The phase-locked propagation unit (PPU) manages the indexes of rays to enter the propagation pipeline. The indexes of active rays are stored in the “Index RAM” in a way to manage efficiently the synchronisation of propagation along the beam phase. Hereinafter a “ray” stands for its index in the “State RAM”. The “Index RAM” is divided in several ranges to manage the phase synchronisation over the different resolutions. The diving of rays being one of the most tricky behavior to deal with.

The beam phase is memorized in the PPU. It is updated when all the ray phases⁵ are further than the beam phase. To that end the memory is divided into in-phase rays and out-phase rays.

A in-phase ray is sent to the “Insertion Unit” to update its next state. On its way back, the ray is either still in-phase, if the propagation occurred on an other axis, or out-phase in other cases. When all the rays are out-phase, the PPU increments the

⁵A ray phase is the ray coordinate along the phase axis

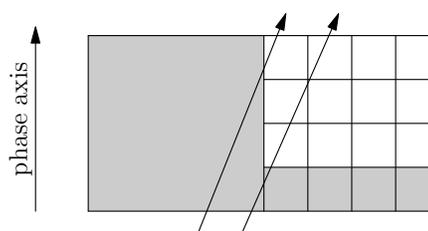


Fig. 9: In some circumstances, diving may out-phase some of the rays of a beam, while keeping others in-phase (gray voxels are on the beam phase)



Fig. 10: Render of the scene used in our tests (128 × 128 version)

beam phase and swaps the out-phase rays into the in-phase. This is actually done when all the rays sent to the propagation pipeline are back. That for, the number of processed rays is counted : #rays is incremented at each new ray inserted in the pipeline or decremented at their return. Rays exiting the scene are counted back but not inserted in the “Index Memory”.

Also, the phase synchronisation have to be performed on all the levels of resolution. So, the “Index RAM” in-phase and out-phase parts are again divided in sub-parts for each level of resolution. Higher depth rays are first sent to the propagation pipeline and the lowest one are then sent, until there are no more in-phase rays at any resolution.

At last, rays diving into a higher resolution have to be sorted according to the beam phase. Indeed, the diving may cause a ray phase to be higher than the current one because the entry point in the higher resolution node may be anywhere on the child border (see Fig. 9). To speed-up the phase sorting, rays are stored in the “Index RAM” according to their relative phase in a node. This relative phase is simply the n bits starting at the $n(\text{max_depth} - d)$ bit of the ray phase, where d stands for the current depth of the ray.

Hereof, we conclude that the “Index RAM” is a memory of $(\text{max_depth} + 1) \cdot 2^n \cdot \text{max_rays}$ words of $\log_2(\text{max_rays})$ bits. A more tolerant phase synchronisation allowing a 2^{dn} phase deviation would need a $2 \cdot \text{max_rays}$ “Index RAM” but it would increase the cache’s memory.

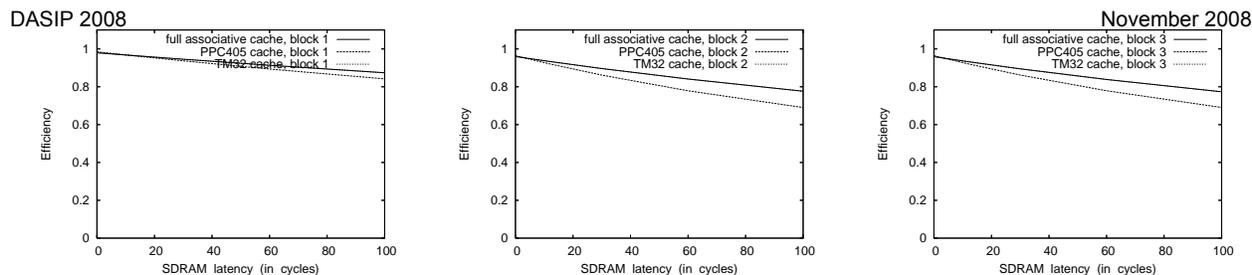


Fig. 13: Pipeline efficiency depending on SDRAM access latency (general purpose caches, for each of the blocks)

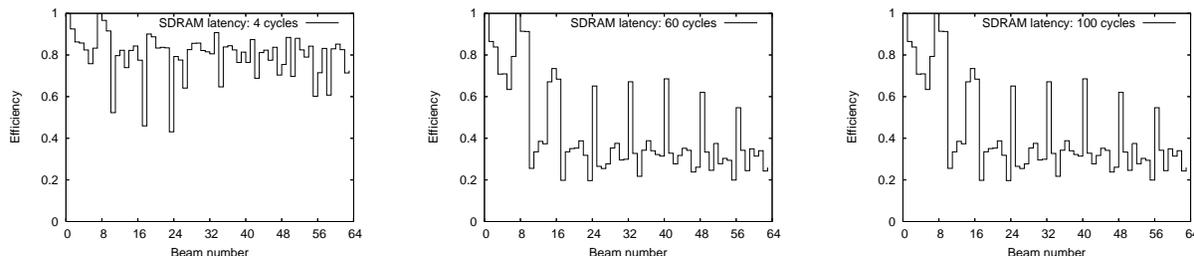


Fig. 14: Pipeline efficiency for the different beams (RG cache, the 128×128 image is swept from the top to bottom, each line being swept from the left to the right (ie. if N is a beam number, the corresponding “macroblock” coordinates are $(N \bmod 8, N \text{ div } 8)$); data are given for three latencies)

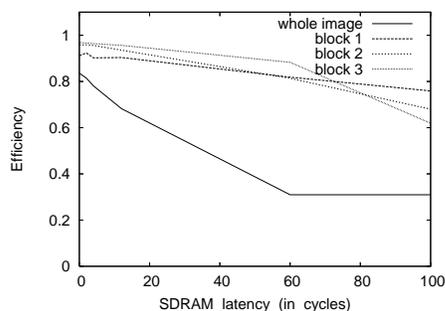


Fig. 11: Pipeline efficiency depending on SDRAM access latency (RG cache)

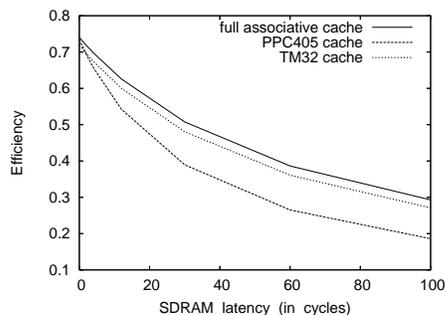


Fig. 12: Pipeline efficiency depending on SDRAM access latency (generic caches, whole image)

IV. RESULTS

This section presents estimations of the RG Cache performance, measured as the pipeline utilisation rate depending on the main memory access latency. The measures are performed on a CABA⁶ VHDL model of the whole system. The initiali-

sation step is performed with a C co-simulation engine. These accurate measures are performed after a first validation step on traces from a C model of the recursive grid traversal unit. The simulations take into account the deviations introduced by the RG Cache and the traversal pipeline. As this paper focuses on memory management, and due to the lack of room, complexity/area results are not given. The point is that the complexity of individual units have already been studied [13] and the new ones were designed carefully.

We have tested our unit with reconstructed MRI data from the PET-SORTEO database⁷. Since the data is provided as an uniform grid, we have build a $4 \times 4 \times 4$ recursive grid from it by merging adjacent voxels which are about the same density (ie., within 37% of the dynamic range). Since each node has 64 children, this criterion does not lead to severe losses in quality.

The Fig. 10 shows a render of the test recursive grid performed by our unit. We used that very same 128×128 render to perform our measures of performance on a whole image. Also, we chose 3 relevant 16×16 pixel areas (hereafter, blocks) corresponding to 3 beams of rays from a 1024×1024 render of the same scene, from the same viewpoint, and benchmarked them. Indeed, the simulation time required for a whole 1024×1024 render with Modelsim is prohibitively long.

The Fig. 12 shows the efficiency of the RG Cache for the reference 128×128 render, and figure Fig. 11 as well as for the 3 blocks of the 1024×1024 render. These performances are compared with those of general purpose caches:

- Full Associative, 16K, 256 lines

⁶Cycle Accurate Bit Accurate

⁷<http://sorteo.cermep.fr/>

- TM32 cache, 16K, 8 way set-associative, 256 lines
- PPC 405 cache (Virtex 4), 16K, 2 way set-associative, 512 lines

The performance is measured for different latencies to estimate the robustness of the system to slow external memories.

Details on the RG Cache performance for the rendering of beams of the reference image are given Fig. 14 for some of the latencies.

These measures show that the RG cache works a lot better on the individual beams than on the reference image. The main reason of such a variation seems to be due to the difference of sampling factors: a beam issued from a 16×16 tile of the 1024×1024 render is thinner than a beam of the 128×128 render. Hence, the size of the cached area is smaller in the case of the 1024×1024 render, which allow a faster load of the cached zone comparatively to the time to process the beam. Indeed, the cached area is 4 time smaller. Also, the setting of cache parameters adapted to the whole scene is more difficult due to the long running time to simulate a rendering. However, these results prove the effectiveness of the RG caching strategy and the cache tuning has still to be improved. For example, the cache efficiency drops down at high latencies because all the measures are performed with the same cache parameters. This could be solved by fine tuning the cache trackers (or designing better trackers).

Also, there is little difference between the performances of the 3 beams we consider, with high hit rates for each of them. This is interesting, since those 3 beams are very different (one does not hit the head, one is somewhat tangent, and one crosses the central region of the brain). The cache seems to be able to prefetch data efficiently both at high and low depths in the recursive grid.

Eventually, performance is not the only gain our cache offers: its interface is done in such a way that it only needs cell coordinates as a request. When using a generic cache, obtaining the cell involves knowing its address, either by performing a research in the recursive grid at each step, or more realistically, for each ray state, storing the whole node hierarchy leading to current cell (which may be a high overhead depending on the exact implementation).

To our knowledge, the presented memory management strategy and recursive grid traversal unit has no equivalent. Hardware architectures presented in the literature address regular grid traversal [7] and full octrees [10]. The later are different from sparse octrees in the way that nodes are always data and never pointers to a higher resolution. They are much like multi-resolution volumes and their traversal is done at a given resolution.

V. CONCLUSION AND PERSPECTIVES

In this paper, we have seen a ray casting algorithm and an associated cache for the efficient traversal of recursive grids, which are acceleration structures similar to octrees. The phase locked traversal algorithm preserves memory access locality, permitting efficient caching.

Measures of the RG Cache performance on a CABA VHDL model have shown the effectiveness of the implemented principle. The cache interface is designed in such a way it completely abstracts the underlying memory structure, allowing similar performances to those well-established generic caches. The RG cache fully exploits the spatial and temporal locality induced by the proposed dynamic re-ordering of computations thanks to the phase-locked propagation.

Some improvements of the system can be made, still. The RG Cache trackers have to be better parameterized and some new tracking mechanisms could be designed. A wider exploration of the system efficiency would be done on a hardware prototype to reduce the simulation time.

Eventually, more generally speaking, this study tends to show the effectiveness of the design of structured-data aware caches. The principles that guided this work could be extended to other applications. For example, hierarchical structures may be used in H.264 Movement Estimation, for multi-resolution textures in 3D triangle rasterization, etc. . .

REFERENCES

- [1] A. Knoll, "A short survey of octree volume rendering techniques," in *Proceedings of 1st IRTG Workshop*, Dagstuhl, Germany, Jun. 2006.
- [2] V. Havran, "Heuristic ray shooting algorithms," Ph.D. Thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, November 2000. [Online]. Available: <http://www.cgg.cvut.cz/havran/phdthesis.html>
- [3] A. S. Glassner, "Space subdivision for fast ray tracing," *IEEE Computer Graphics & Applications*, vol. 4, no. 10, pp. 15–22, Oct. 1984.
- [4] D. Jevans and B. Wyvil, "Adaptive voxel subdivision for ray tracing," in *Proceedings of Graphics Interface '89*, Jun. 1989.
- [5] J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing," in *Eurographics '87*. Amsterdam, North-Holland: Elsevier Science Publishers, 1987, pp. 3–10. [Online]. Available: citeseer.ist.psu.edu/amanatides87fast.html
- [6] H. Pfister, A. Kaufman, and T. Chiueh, "Cube-3: A real-time architecture for high-resolution volume visualization," in *1994 Symposium on Volume Visualization*, A. Kaufman and W. Krueger, Eds., 1994, pp. 75–82. [Online]. Available: citeseer.ist.psu.edu/pfister94cube.html
- [7] H. Pfister and A. E. Kaufman, "Cube-4 - a scalable architecture for real-time volume rendering," in *VVS, 1996*, pp. 47–. [Online]. Available: citeseer.ist.psu.edu/pfister96cube.html
- [8] R. Osborne, H. Pfister, H. Lauer, N. McKenzie, S. Gibson, W. Hiatt, and T. Ohkami, "EM-Cube: An architecture for low-cost real-time volume rendering," in *1997 SIGGRAPH / Eurographics Workshop on Graphics Hardware*. ACM Press, 1997. [Online]. Available: citeseer.ist.psu.edu/osborne97emcube.html
- [9] U. Kanus, G. Wetekam, and J. Hirche, "Voxelcache: a cache-based memory architecture for volume graphics," in *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003.
- [10] G. Wetekam, D. Staneker, U. Kanus, and M. Wand, "A hardware architecture for multi-resolution volume rendering," in *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. New York, NY, USA: ACM Press, 2005.
- [11] A. Reshetov, "Omnidirectional ray tracing traversal algorithm for kd-trees," *Interactive Ray Tracing 2006, IEEE Symposium on*, Sep. 2006.
- [12] A. Reshetov, A. Soupikov, and J. Hurley, "Multi-level ray tracing algorithm," *ACM Trans. Graph.*, vol. 24, no. 3, 2005.
- [13] S. Mancini and M. Desvignes, "Efficient memory management for ray casting," in *Workshop on Design and Architectures for Signal and Image Processing*, Grenoble, France, Nov. 2007.
- [14] K. S. Klimaszewski and T. W. Sederberg, "Faster ray tracing using adaptive grids," *IEEE Comput. Graph. Appl.*, vol. 17, no. 1, pp. 42–51, Jan.-Feb. 1997.
- [15] J. Revelles, C. Ureña, and M. Lastra, "An efficient parametric algorithm for octree traversal," 2000. [Online]. Available: citeseer.ist.psu.edu/476719.html

OLLAF : a Fine Grained Dynamically Reconfigurable Architecture for OS Support

Samuel GARCIA and Bertrand GRANADO

ENSEA - ETIS

95014 CERGY, France

email: samuel.garcia@ensea.fr,

bertrand.granado@ensea.fr

Abstract—In the context of large versatile platform for embedded real time system on chip, a fine grained dynamically reconfigurable architecture could be used as one possible computational resource. In order to manage efficiently this resource we need a specific OS kernel able to manage such a hardware adaptable architecture. Both the history of micro-processor based system and our previous work based on currently available FPGA devices led us to think that not only an OS kernel must be defined to handle an FGDR but a FGDR must also be designed to handle this OS kernel. This article relate our original work in this direction. OLLAF¹, an original FGDR core that we have designed will be presented. A comparison with other methods used today using commercially available FPGA is also presented concerning the particular preemption service.

I. INTRODUCTION

This work takes place in the SMILE project. This project aims at provide a distributed middle layer to efficiently handle the complexity of a tomorrow's RSoC². This system may contains several computing units of different types. It will embed at least one or more General Purpose Processor (GPP), but also dynamically reconfigurable architectures (DRA) at different granularities and especially FGDR³. Tomorrow's computing systems has to comply with lots of constraints. Those constraints may be time related, to meet real time requirements, but also power consumption constraints, as it is, and will be more and more, one of the primary concern of electronical devices.

By fine grained, we here means an architecture which is reconfigurable at the bit level. A dynamically reconfigurable architecture, using single bit LUT and flipflop, and providing a bit level reconfigurable interconnection matrix, as the one presented here, or basic logic fabric of most commercial FPGA, are examples of FGDR. Those kind of architecture can be adapted to any application more optimally than a coarser grain DRA. This feature make them today the platform of choice when it comes to handle computational tasks in a highly constrained context.

In more general terms FGDR can achieves much better efficiency than GPP does, while offering the same versatility and, potentially, a very close flexibility. The counterpart is that

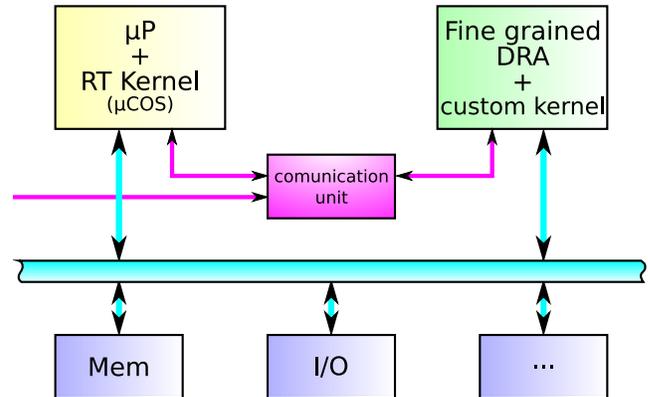


Fig. 1. Basic view of a RSoC in SMILE

it introduces a much greater complexity for application designers. This complexity could be lowered to an acceptable level in two ways. First by providing powerful CAD tool. Lots of research are thus led in the field of high level synthesis [1]. The second way is to abstract the system complexity by providing a middle layer, e.g an operating system, that abstracts the lower level of the system [2]. Moreover, an OS could manage new tasks at run time. This property is a feature of importance for DRA. For all those reasons, a specialized operating system is required for FGDR.

In our work we make a difference between a FGDR, which is a general term, and a FPGA which, for us, relate to an actual silicon device sold under this designation and which can be used as a FGDR but is actually not designed especially thor that purpose.

The SMILE project follows a distributed approach of the system. Each computing unit of a RSoC (GPP, DSP, DRA, ...) has its own real time kernel. This topology allows to use a specific custom made real time kernel for each computing unit. It then allows to take into account every specificities of each computing unit. A message passing communication scheme, based on MPI⁴, ensure a consistent operation of the whole system. In this frame of mind, we developed a dedicated real time kernel for a FGDR.

This kernel is an adaptation to FGDR of an abstract OS

¹Operating system enabled Low Latency Fgdra

²Reconfigurable System on Chip

³Fine Grained Dynamically Reconfigurable Architecture

⁴Message Passing Interface

model which could be described as follow :

- it manages the execution of a set of task on a given versatile computational resource. More concretely it will run periodically a special algorithm to evaluate where and when to run each tasks. This period is called Tick and is a tradeoff between efficiency and flexibility, a typical value in classical OS is tens of milliseconds.
- it offers an abstracted view of the platform to the task designer. In other terms, each task can be designed without worrying about other tasks and sometimes even about the platform. It then offers a standardized set of services such as communications or synchronizations between tasks.

This model slightly differs from most OS implementation proposed for FPGA management even if the overall idea remain the same.

Both the history of micro-processor based system and our previous work based on currently available FPGA devices led us to think that not only an OS kernel must be conceived to handle a FGDR, but a FGDR must also be designed to support efficiently this OS kernel. This article relate our original works in this direction. The FGDR core that we have designed will be presented as well as a more general view of our approach of a FGDR and its related OS kernel.

This paper is organized as follows. Section 2 discuss of related works in the field of OS for FGDR. Section 3 explains our original FGDR platform proposition named OLLAF. Section 4 discuss more precisely of the context management scheme and its extension to configuration management. Section 5 explains how our architecture can affect different OS services. Section 6 exposes an analytic comparison between OLLAF and other methods used today in terms of preemption overhead and efficiency. Finally, conclusions are drawn in section 7.

II. RELATED WORK

A. OS for FGDR

Several research have been led in the field of OS for FGDR [3], [4], [5], [6]. All those studies present an OS more or less customized to enable specific FGDR related services. Example of such services are : partial reconfiguration management, hardware task preemption or hardware task migration. They are all designed on top of a platform composed of a commercial FPGA and a micro-processor. This microprocessor may be a softcore processor, an embedded hardwired core or even an external processor.

In the 90's, some works have also been published about the design of a specific architecture for dynamical reconfiguration. In [7] authors discuss about the first multi-context reconfigurable device. This concept as been implemented by NEC on the Dynamically Reconfigurable Logic Engine (DRLE) [8]. At the same period, the concept of DPGA was introduced, it was also proposed in [9] to implement a DPGA in the same die as a classic microprocessor to form one of the first SoC including dynamically reconfigurable logic. In 1995, Xilinx

even applied a patent on multi-context programmable device proposed as an XC4000E FPGA with multiple configuration planes [10].

More recently, in [11], authors propose to add special material to a DRA to support OS services, they worked on top of a classic FPGA.

The work presented in this paper try to take advantage of those previous work both about hardware reconfigurable platform and OS for FGDR.

B. previous work

Our first work on OS for FGDR was related to preemption of hardware task on FPGA [12]. For that purpose we explored the use of a scanpath at the task level. In order to accelerate the context transfer we explore the possibility of using multiple parallel scanpaths. We also provided the Context Management Unit or CMU, which is a small IP capable to manage the whole process of saving and restoring tasks contexts.

In that study both the CMU and the scanpath were build to be implemented on top of any available commercial FPGA. This approach showed number of limitations. They could be summarized in this way: implementing this kind of OS related material on top of the existing DRA introduce unacceptable overhead on both the task and the OS service. Differently said, most of OS related material should be as much as possible hardwired into the platform's architecture.

III. OLLAF : GENERAL OVERVIEW

A. Specifications of a FGDR with OS support

We have designed a FGDR with OS support following those specifications.

It should first address the problem of the configuration speed of a task. This is one of the primary concerns because if the system spend more time configuring itself than actually running tasks, then its efficiency will be poor. The configuration speed will thus have a big impact on the scheduling strategy.

In order to enable more choice on scheduling scheme, and to match some real time requirement, our FGDR platform must also include preemption facilities. For the same reasons than configuration, the speed of context saving and restoring process will be one of our primary concerns. On this particular point, previous work we have discussed in section 2 will be adapted and reused.

Scheduling on a single GPP system is just a matter of time. The problem is to distribute the computation time between different tasks. In the case of a DRA the system must distribute both computation time and computation resources. Scheduling in such a system is then no more a one dimensional problem, but a three dimensional one. One dimension is the time and the two others are the surface of reconfigurable resources. Performing such a scheduling at run time with real time constraints is at this stage not conceivable. But the FGDR should help getting close to that goal. The primary concern on this subject is to ensure an easy task relocation. For that, the reconfigurable logic core should be splited into several equivalent blocks. This will allow to move a task from a block

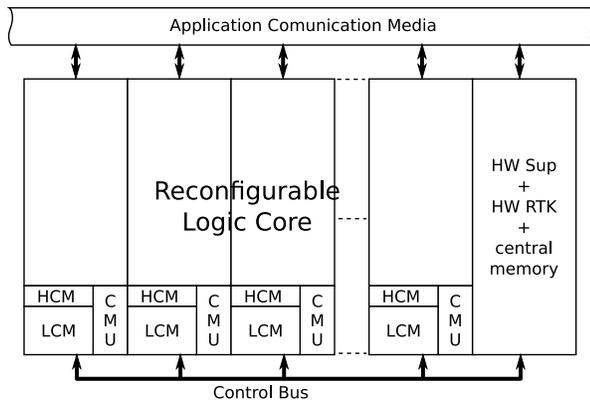


Fig. 2. Global view of the FGDR A

to any another block or from a group of blocks to another group of blocks of the same size and the same form factor without any change on the configuration data. The size of those blocks would be a tradeoff between flexibility and scheduling efficiency.

Another aspect of an operating system is to provide inter task communication services. In our case we will distinguish two cases. First the case of a task running on top of our FGDR A and communicating with another task running on a different computing unit, for example a GPP. This case will not be covered here as this problem concern the whole heterogeneous platform, not only the particular FGDR A computing unit. The second case is when two, or more, tasks run on top of the same FGDR A communicate together. This communication channel should remain the same wherever the task is placed on the FGDR A reconfigurable core and whatever state those tasks are (running, pending, waiting, ...). That mean that the FGDR A platform must provide a rationalized communication medium including some sort of exchange memories.

The same arguments could also be applied to inputs/outputs. Here again two cases exist. First the case of I/O being a global resource of the whole platform. Secondly the case of special I/O directly bound to the FGDR A.

B. Proposed solutions

Figure 2 show a global view of OLLAF, our original FGDR A designed to support OS services as they have just been specified.

In the center stand the reconfigurable logic core of the FGDR A. This core is organized in columns, each column can be reconfigured separately and offer the same set of services. That means that a task uses an integer number of columns. This topology as been chosen for two reasons. First using a partial reconfiguration by column transforms the scheduling problem into a two dimensional problem (time + 1D space) which will be easier to handle in real time situations. Secondly as every columns is the same and offers the same set of services, tasks can be moved from one column to another without any change on the configuration data.

In the figure, at the bottom of each column you can notice

two hardware blocks called CMU and HCM. The CMU as said earlier is an IP able to manage automatically task's context saving and restoring. The HCM standing for Hardware Configuration Manager is pretty much the same but to handle configuration data also called bitstream. On each column a local configuration/context memory is added. This memory can be seen as a first level of cache memory to store contexts and configurations close to the column where it might most probably be required. The internal architecture of the core provides adequate materials to work with CMU and HCM. More about this will be discussed in the next section.

On the right of the figure stands a big block called "HW Sup + HW RTK + central memory". This block contain a classic microprocessor which serves as a hardware supervisor. It runs a custom real time kernel specially adapted to handle FGDR A related OS services and platform level communication services. Along with this hardware supervisor a central memory is provided for OS use only. Basically this memory will store configuration and eventual context of every task that may run on the FGDR A. This supervisor communicates with all columns using a dedicated control bus.

Finally, on top of the figure 2 you can see the application communication medium. This communication medium provides a communication port to each column. Those communications ports will be directly bound to the reconfigurable interconnection matrix of the core. If I/O had to be bound to the FGDR A they would be connected with this communication medium in the same way reconfigurable columns are.

C. Logic core overview

In order to make the description of the FGDR A core more understandable, we will here split its functionalities between two points of view. The first one is the functional point of view, it consists on the information that a task designer may have to know in order to design the architecture. The second point of view is the configuration point of view, it consists on information about reconfiguration plane. As one of the main goals of the OS is to abstract configuration management, this point of view could be seen as the OS point of view.

Internal architecture of a LE in the functional point of view can be seen on figure 3. This architecture integrates elements that compose a classic Logic Element of FGDR A. If we want to improve functional architecture, it should not change our conclusion on configuration point of view.

A multiplexor based interconnect as been chosen instead of the passing MOS transistor used in most commercial FPGA. In this way we can lower the number of configuration bit required to allow the same connection flexibility. In this last interconnection scheme, the number of configuration bit grow linearly with interconnection possibility while using multiplexor makes it grow as a log2 function.

At first, configuration memory points are modellized as a D flip-flop. This allow us to rapidly apply our works on context management to configuration management. However, configuration and context management remains two separate path, a context swap can be performed without any change

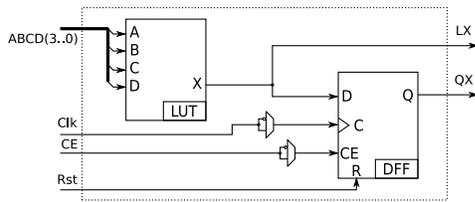


Fig. 3. Functional, task designer point of view of LE

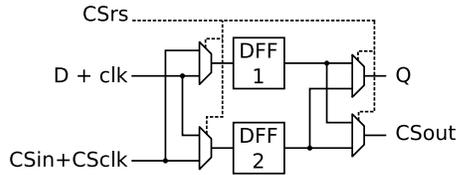


Fig. 4. Dual plane memory point

in configuration. This can be interesting for checkpointing or when more than one instance of the same task running.

IV. CONTEXT MANAGEMENT SCHEME

In [12] we proposed a context management scheme based on a scanpath, a local context memory and the CMU which is a small IP capable of managing automatically context transfer between the scanpath and the local memory. The context management scheme in OLLAF is slightly different in two ways. First, every context management related material is hard wired into the platform. Secondly, we added two more stage in order to even lower preemption overhead and to ensure the consistency of the system.

As context management materials are added at platform level and no more at task level, it needed to be split differently. As the Programmable Logic Core is column based, it was then natural to implement context management at columns level. A CMU and a local memory have then been added to each column, and one scanpath is provided for each column's set of flipflops.

In order to lower preemption overhead, our reconfigurable logic core use a double memory plane. Flipflops used in LE are thus replaced with two FF with switching material. Architecture of this double plane FF can be seen on figure 4. *Run* and *scan* are then no more two working modes but two parallel planes which can be swapped as will. With this topology, the context of a task can be shifted in while the previous task is still running and shifted out while the next one is already running. The effective task switching overhead is then taken down to one clock cycle as illustrated in figure 6.

Contexts are transfered by the CMU into Local Context Memories using this hidden scanpath. Because the context of every column can be transfered in parallel, Local Context Memories are placed at column level. It is particularly useful when task use more than one column. Those memories can contain at this stage 10 contexts. They can be seen as local cache memories to optimize access to a bigger memory called the Central Context Repository.

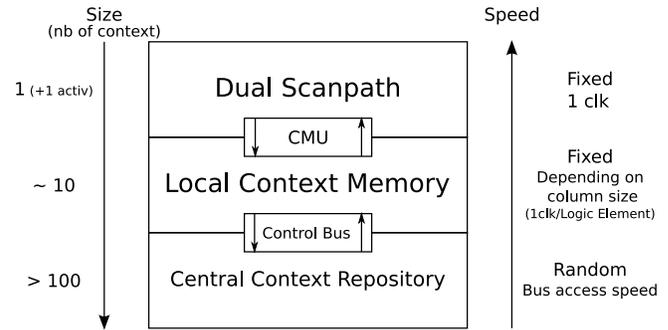


Fig. 5. Context memories hierarchy

The Central Context Repository is a large memory space storing the context of each task instance run by the system. Local Context Memories should then store contexts of tasks who are most likely to be the next to be ran on the corresponding column.

After a preemption of the corresponding task, a context can be stored in more than one LCM in addition to the copy stored in the Central Context Repository. In such situation, care must be taken to ensure the consistency of the task execution. For that purpose, contexts are tagged by the CMU each time a context saving is performed with a version number. The operating system keep tracks of this version number and also increment it each time a context saving is performed. In this way the system can then check for the validity of a context before a context restoration. The system must also try to update the context copy in the CCR as short as possible after a context saving is performed.

Dual Plan Scanpath, Local Context Memory and Central Context Repository form a complex memory hierarchy specially designed to optimize preemption overhead. The same memory scheme is also used for configuration management except configuration do not change during execution so it does not need to be saved and then no versioning control is required here. The programmable logic core use a dual configuration plane equivalent to the Dual Plane Scanpath used for context. Each column has a Hardware Configuration Manager which is a simplified version of the CMU (without saving mechanism). A Local Configuration Memory is provided beside Local Context Memory, the name LCM is used as in figure 3 to relate to both those memories. In the same way, the CCR can refer to Central Context/Configuration Repository.

In best case, preemption overhead can then be bound to one clock cycle.

A scenario of a typical preemption is presented here. In this scenario we consider the case where context and configuration of both task are already stored into the right LCM. Let's consider that a task T1 is preempted to run another task T2, scenario of task preemption is then as follow :

- T1 is running and the scheduler decide to preempt it to run T2 instead
- T2's configuration and eventually context is shifted on the second configuration plane

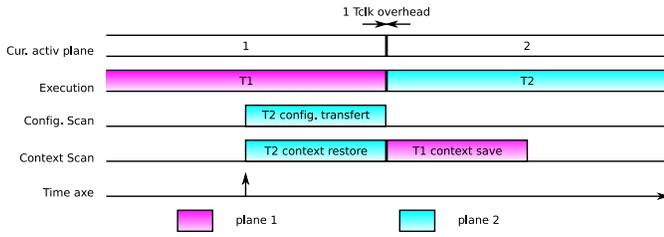


Fig. 6. Typical preemption scenario

- once the transfer is completed the two configurations planes are switched
- now T2 is running and T1's context can be shifted out to be saved
- T1's context is updated as soon as possible in the CCR

This scenario is illustrated in figure 6.

This is the case when both context and configuration of T2 are already stored into LCM. That means that, in order to have this favorable case, we need an anticipated scheduling to manage our Context/Configuration Memories Hierarchy as a smart cache.

V. CONFIGURATION, PREEMPTION AND OS INTERACTION

In previous sections an architectural view of our FGDR has been exposed. In this section, we discuss about the impact of this architecture on OS services. We will here consider the three services most specifically related to the FGDR.

First, the configuration management service. On the hardware side, each column provides a hardware configuration manager and an associated local memory. As stated earlier that mean that configurations have to be placed in advance in the local configuration memory. The associated service running on the hardware supervisor micro-processor will thus need to take that into account. That imply that this service must manage an intelligent cache to prefetch task configuration on the columns where it might most probably be placed. In order to do so, an anticipated scheduling must be performed.

Secondly, the preemption service. The same principle must be applicable here as those applied for configuration management. Except that contexts also have to be saved. The context management service must ensure that it never exist more than one valid context for each task in the entire FGDR. Context must thus be transferred as soon as possible from local context memory to the centralized global memory of the hardware supervisor. This service will also have a big impact on the scheduling service as the ability to perform preemption with a very low overhead allow the use of more flexible scheduling algorithms.

And last the scheduling service and in particular the space management part of the scheduling. It takes advantage of the column topology and of the centralized communication scheme. As stated, fewer computing power will be required to manage a one dimensional space at run time. The problem is here similar to memory management in classical GPP based system. The reconfigurable resource could then be managed as

a virtual infinite space containing an undetermined number of columns. The job is then to dynamically map the required set of columns (task) into the real space (the actual reconfigurable logic core of the FGDR).

VI. PREEMPTION COST COMPARISON

This section present an analytic comparison of preemption efficiency in OLLAF and other solution from past works or literature. We will here consider six methods :

- XIL
a solution based on the xilinx XAPP290 [13] using ICAP to transfer both context and configuration and using the readback bitstream for context extraction.
- Scan
a solution using a simple scanpath for context transfer as described in both [14] and [12], and using ICAP interface for configuration.
- PCS8
is similar to Scan solution but using 8 parallel scanpath as described in [12].
- DPScan
use a dual plane scanpath similar to the one used in OLLAF for context and ICAP for configuration. This method is also studied in [14], referred as a shadow Scan Chain.
- MM
use once again ICAP for configuration and the memory mapped solution proposed in [14].
- OLLAF
this last solution being the use of separate, column distributed, dual plane scanpath for configuration and context as proposed in this article.

In this study we consider two parameter. The preemption overhead H is the cost of a preemption for the system in terms of time. The efficiency of preemption process λ is then $\lambda = 1 - \frac{H}{P}$ with P is the minimum period at which preemption occurs so in our case P is the clock tick of the operating system. In this study we use a typical clock tick of 10ms. In order to focus on the architectural view only all times will be expressed and estimated in number of clock cycle. Assuming a typical clock frequency of 100MHz the OS tick is 10^6 tlck. Task sizes will be expressed as n , the number of flipflop used. The time cost of a preemption take into account two context transfers and one configuration transfer.

Analytic expression of H for each case are estimated as follow :

- XIL
In [14] authors estimate that bitstream contain 20 times more data than context related data so the bitstream of a task of size n is approximately $21n$. Assuming that it use a 32bits width access bus, the ICAP interface can transfers 32bits per clock cycle. In the same article, authors estimate that it takes 20 clock cycles to extract each context bit from the readback bitstream.

$$H = \frac{21n}{32} + \frac{21n}{32} + 20n \simeq 21.3n \quad (1)$$

	XIL	Scan	PCS8	DPScan	MM	OLLAF
H (tclk)	15188	1897	642	472	492	1
λ	98.4%	99.8%	99.94%	99.95%	99.95%	$1 - 10^{-6} \simeq 100\%$

TABLE I

COMPARISON OF TASK PREEMPTION OVERHEAD AND EFFICIENCY FOR 713FF TASK

	XIL	Scan	PCS8	DPScan	MM	OLLAF
H (tclk)	21.3×10^6	2.66×10^6	900×10^3	660×10^3	690×10^3	1
λ	-2030%	-166%	10%	34%	31%	$\simeq 100\%$

TABLE II

COMPARISON OF TASK PREEMPTION OVERHEAD AND EFFICIENCY FOR A WHOLE 1MFF FGDR

- Scan

Using a simple scanpath for context transfer requires 1 clock cycle per flipflop for each transfer.

$$H = \frac{21n}{32} + 2n \simeq 2.66n \quad (2)$$

- PCS8

Using 8 parallel scanpath it requires 1 clock cycle for 8 flipflops.

$$H = \frac{21n}{32} + \frac{2n}{8} \simeq 0.9n \quad (3)$$

- DPScan

Using a double plane scanpath, the context transfers can be hidden, the cost of those transfer is then always 1 clock cycle.

$$H = \frac{21n}{32} + 1 \simeq 0.66n + 1 \quad (4)$$

- MM

Using 32 bits memory access, this case is similar to the PCS8 but using 32 parallel paths instead of 8.

$$H = \frac{21n}{32} + \frac{2n}{32} \simeq 0.69n \quad (5)$$

- OLLAF

In OLLAF, both context and configuration transfer are hidden so the total cost of the preemption is always 1 clock cycle whatever the size of the task.

$$H = 1 \quad (6)$$

In order to make a concrete case comparison, we will consider two task T1 and T2. We consider a DES56 cryptographic IP that requires 862 flipflops, and a 16tap FIR filter that requires 563 flipflop. Both of those IPs can be found in www.opencores.org. To ease the computation we will consider two task using the average number of flipflop of the two considered IP. So for T1 and T2 we got $n = \frac{862+563}{2} \simeq 713$. Table I show the overhead H and the efficiency λ for each method presented.

Those results show that in this case, using our method leads to a preemption overhead around 500 times smaller than the bests others cases.

If we now consider that not only one task is preempted but the whole FGDR, assuming a 1 Million LE's logic core, estimation of overhead and efficiency for each method are shown in table II. Those results show clearly the benefit of OLLAF platform over actual FPGA concerning preemption. Using actual methods, preemption overhead is linearly dependant on the size of the task. In OLLAF, this overhead do not depends on the size of the task and is always of only one clock cycle.

In OLLAF, both context and configuration transfers are hidden due to the use of a dual configuration plane. The latency L between the moment a preemption is asked and the moment the new task effectively begin to run can also being studied. This latency only depends on the size of the columns. That means that for a given platform, it will be a constant. In the worst case this latency will be far shorter than the OS tick period. OS tick period being in any case the shortest time in which the system can respond to an event, we can consider that this latency will not affect the system at all.

VII. CONCLUSION AND PERSPECTIVES

A global view of OLLAF, a FGDR that enhance OS service support has been presented, and in more details its reconfigurable logic core. We claim that OS and platform must be closely linked to each others in order to perform as optimally as possible.

In this paper we presented in more details our context management scheme and its extention to configuration management. It has been shown that this scheme permit a far better preemption efficiency than other methods in use today.

Today, the reconfigurable logic core have been designed and is being tested by several simulations. The rest of the FGDR is also in progress. The dedicated custom OS services are written as an extension of $\mu C/OS-II$, a well proven real time OS. We are also working on the distributed management of the whole heterogeneous system including, at least, one of our FGDR and its dedicated real time kernel, and one GPP.

REFERENCES

- [1] P. Coussy, G. Corre, P. Bomel, E. Senn, and E. Martin, "High-level synthesis under i/o timing and memory constraints," in *Proceeding of IEEE International Symposium on Circuits and Systems (ISCAS)*, 2005.

- [2] Q. Deng, S. Wei, H. Xu, Y. Han, and G. Yu, "A Reconfigurable RTOS with HW/SW Co-scheduling for SOPC," in International Conference on Embedded Software and Systems (ICCESS), 2005, pp. 116–121.
- [3] H. Simmler, L. Levinson, and R. Männer, "Multitasking on FPGA Coprocessors." in Field Programmable Logic and its Applications (FPL), ser. Lecture Notes in Computer Science, no. 1896, 2000, pp. 121–130.
- [4] G. Chen, M. Kandemir, and U. Sezer, "Configuration-Sensitive Process Scheduling for FPGA-Based Computing Platforms." in Design Automation and Test in Europe (DATE), 2004, pp. 486–493.
- [5] H. Walder and M. Platzner, "Reconfigurable Hardware Operating Systems: From Design Concepts to Realizations," in Engineering of Reconfigurable Systems and Algorithms (ERSA), 2003, pp. 284–287.
- [6] G. Wigley, D. Kearney, and D. Warren, "Introducing reconfirme: An operating system for reconfigurable computing," in Conference on Field Programmable Logic and Application, September 2-4 2002.
- [7] X. ping Ling and H. Amano, "Wasmii : a data driven computer on virtuel hardware," in IEEE workshop on FPGAs for custom computing machines, 1993.
- [8] Y. Shibata and al., "A virtual hardware system on a dynamically reconfigurable logic device," in IEEE symposium on FPGAs for custom cmputing machines, 2000.
- [9] A. DeHon, "Dpga-coupled microprocessors : Commodity ics for the early 21st century," in IEEE Workshop on FPGAs for custom computing machines, 1994.
- [10] Xilinx, "Time multiplexed programmable logic device," Patent no.5646545, 1997.
- [11] V. Nollet, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins, "Designing an Operating System for a Heterogeneous Reconfigurable SoC," in International Parallel and Distributed Processing Symposium (IPDPS), 2003, p. 174a.
- [12] S. Garcia, J. Prevotet, and B. Granado, "Hardware task context management for fine grained dynamically reconfigurable architecture," in Workshop on Design and Architectures for Signal and Image Processing (DASIP), 2007.
- [13] Xilinx, "Two flows for partial reconfiguration: Module based or difference based," Xilinx, Application Note, 2004, application Note: Virtex, Virtex-E, Virtex-II, Virtex-II Pro Families XAPP290 (v1.2) September 9, 2004.
- [14] D. Koch, C. Haubelt, and J. Teich, "Efficient hardware checkpointing - concepts, overhead analysis, and implementation," in Field Programmable Logic and its Applications (FPL), 2007.

Reconfigurable Artificial Neural Network Model for Task Scheduling on Reconfigurable SoC

Daniel Chillet, Sebastien Pillement and Olivier Sentieys
 CAIRN Team, ENSSAT, University of Rennes 1, INRIA
 6 rue de kerampont
 BP 80518, 22305 Lannion
 Email: *firstname.name@irisa.fr*

Abstract—Through the introduction of reconfigurable unit, the recent System-on-Chips are now able to support dynamic applications and to ensure the execution of several tasks within the same resource. Nevertheless, an efficiency management of the reconfigurable unit needs some specific services. One of the most important is the scheduling service which ensures the correct execution of all the application tasks. In classical systems, the number of tasks which can be scheduled at each cycle depends on the number of execution targets. For the reconfigurable unit, the number of tasks in execution depends on the available area within this unit and a variable number of tasks can be executed simultaneously while the area is not full. In this context, the scheduling problem is then different and more complex than in classical systems. To solve this problem, we propose a model of Reconfigurable Artificial Neural Networks used for on-line task scheduling on a reconfigurable system-on-chip architecture. Our proposal is based on the Hopfield model adaptation with a regular reconfiguration of the neural network. The major advantage of our proposal concerns the small number of neurons to model the problem which ensures a rapid convergence of the neural network. Our proposal is then able to manage, i.e. to schedule, the large number of tasks of the future reconfigurable SoC.

I. INTRODUCTION

Embedded applications are usually implemented on complex System-on-Chip (SoC) which are built around heterogeneous processing units. New approach consists in implementing processor cores, hardware block (i.e. Intellectual Property IP blocks) and dynamically reconfigurable accelerators (Figure 1) in the same circuit designing a Reconfigurable SoC (RSoC). Classically, the system is organized around a general-purpose processor which runs an Operating System (OS). The main objective of this OS consists in controlling all the available resources.

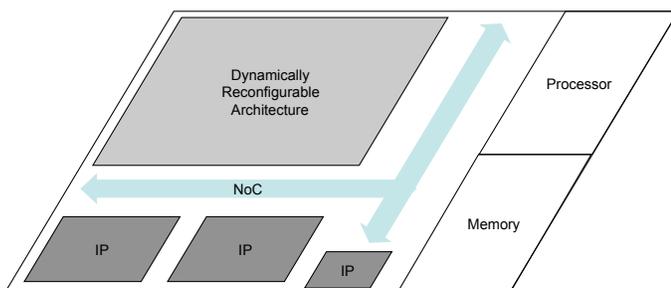


Fig. 1. Example of Reconfigurable SoC architecture

More precisely, task instantiation on execution resources is realized by using the scheduling OS service. As each task can

be defined for several targets (in figure 2, D arrows indicate task definitions for software and/or hardware targets), this service must decide at run time, on which resource the task should be instantiated. We define two levels of abstraction: The applicative tasks represents the computation of the algorithm independent of its final implementation. The executive tasks are the instantiation of each applicative tasks on a particular execution target. Figure 2 shows an example of tasks and the different possibilities to map it on a minimal RSoC. In this example, we can show that applicative task 2 is defined by 3 different executive tasks, which are tasks HW task 2, HW task 2' and SW task 2. On the other hand, tasks 1 and 3 are defined with just one executive task. To be execute, some tasks need to be configured and then launched (for example HW task 2 is configured C onto the reconfigurable resource and then launched/executed L). In other hand, for hardware tasks executed in IP blocs, only launching step L is necessary. This figure also illustrates the configuration of a processor core in the DRA (Dynamically Reconfigurable Area) which can ensure the execution of software tasks if necessary. In this case, a configuration step C is necessary before launching the execution E .

In this context, we can define the list of characteristics that needs to be controlled by the OS. First, as we can see on the figure, the system is composed of multiple resources which are heterogeneous in their characteristics, and particularly in their execution times, the system corresponds to a heterogeneous multiprocessor. Second, to ensure an efficient execution of the application, the OS must manage the task dynamicity to adapt the execution to the dynamic support provide by the DRA. This property ensures the adaptation of the execution contexts by partial reconfiguration without disturbing running tasks. Nevertheless, the overhead cost of reconfiguration (in time and in energy consumption) must be limited as much as possible. For this reason, the task model for the DRA resource needs to be limited to an unpreemptive model. Third, unlike the classical processor core, the DRA resource is limited by its area usage, not by a number of tasks. The task scheduling must be able to manage the execution of several simultaneous tasks. Finally, because the number of execution targets increases rapidly, we assume that an efficient management of tasks cannot be done globally. So we assume that a part of OS services needs to be implemented in the different

execution targets and in particular in the reconfigurable unit. Two solutions can be used to implement this service in the reconfigurable unit. The first solution consists in embed a processor core within this unit and to develop a specific software scheduling service. The second solution consists in defining a specific hardware structure for this service. We have chosen to explore the second solution in order to limit as much as possible the area impact of this service implementation within the reconfigurable unit. Indeed, the area cost of a core processor is not negligible, and we hope that a hardware solution can limit the occupied area. Furthermore, due to time aspects, we think that specific hardware implementation will be more efficient than a software solution, in particular because instruction fetch/decode/execute does not exist in hardware solution. For these reasons, our work focus on defining a on-line scheduling services able to manage an unfixed number of tasks and that can be implemented with the low overhead, within the reconfigurable unit.

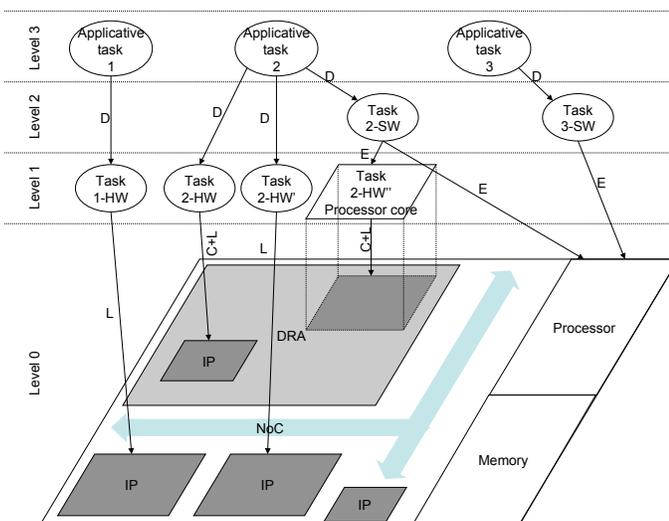


Fig. 2. Mapping example of an application on RSoC

In this paper, we propose an on-line scheduling based on a Artificial Neural Network (ANN) for RSoC architectures. The main objectives of our proposition is to support a variable number of tasks to schedule according to their area usage rate in the reconfigurable hardware. For this purpose we define a Reconfigurable Artificial Neural Network (RANN) which is updated at specific scheduling cycles. Our model takes into account the task dependencies and supports a non preemptive model of tasks. This last point ensure a minimal number of reconfigurations and then limits the energy consumption of this unit.

The remainder of this paper is as follows. Section II, presents the state-of-art on the scheduling solutions for multiprocessor systems. Scheduling through ANN models are presented in this section. In section III, we present our RANN. Section IV presents simulation of the proposed RANN. Finally, we conclude and discuss our future work.

II. RELATED WORKS

The task scheduling is one of the most important problem in the context of system-on-chip. Managing software and hardware targets to ensure an efficient execution of tasks is difficult in this context due to the heterogeneous execution targets. Furthermore, nowadays the most SoC embed reconfigurable unit to ensure application flexibility and adaptability. The reconfigurable unit can support the execution of high number of tasks, but in general only few tasks are executed at the same time. The dynamic reconfiguration of this unit allows to sequentially map tasks onto the same area, and a static scheduling is often insufficient. We can also note that the complexity increasing leads designers to propose some hardware OS service implementations to ensure an efficient execution of tasks. The scheduling OS service, known as spatiotemporal scheduling, has been studied, and two majors techniques are usually proposed: the one dimensional or two dimensional schemes to map tasks onto the reconfigurable unit [3], [4], [16]. Static and dynamic schedulers are proposed but on-line solutions are often preferred due to the flexibility.

In the context of SoC architecture, specific task scheduling services has been proposed to take into account heterogeneity characteristics of execution resources [11], [15]. These service implementations are often complex, and are not always appropriate to real-time systems [9]. They are generally time consuming and do not consider the dynamic behavior of applications. The PFair algorithm [2], focus on an optimal solution for periodic tasks on homogeneous multiprocessors. In [14], the authors propose an approximate solution to reduce global complexity and to design hardware implementations of the PFAIR approach. Nevertheless, this type of solution is not adapted to the management of dynamicity of the reconfigurable execution unit.

Due to the increasing number of constraints to solve the scheduling problem, approximate methods was developed. Among them, we can cite genetic algorithms [13], simulated annealing [6] and Artificial Neural Networks [1]. Neural networks have demonstrated their efficiency for optimization problems that take into account several constraints. They converge in a reasonable time (i.e. in a few cycles) if the number of neurons and connections between neurons can be limited as much as possible. Until now, ANNs have been used for scheduling tasks on classical SoC architectures, i.e. SoC without reconfigurable hardware. Most of the proposals address essentially mono-processor architecture or homogeneous multi-processors.

In [5], the authors have proposed the ANNs for on-line real-time scheduling. Their solution extends the results obtained in [17] and the theoretical basis for ANN design for optimization problems is defined in [8], [10]. By using a Hopfield model [12], they ensure the existence of a Lyapunov function, called *energy function*. This model ensures that the network evolution converges towards a stable state for which the optimization constraints are respected. This function is defined as:

$$E = -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N T_{ij} \cdot x_i \cdot x_j - \sum_{i=1}^N I_i \cdot x_i \quad (1)$$

where N is the number of neurons, T_{ij} is the connection weight between neurons i and j , x_i is the state of neuron i and I_i is the external input of neuron i .

Based on this model, a design rule that facilitates the neural network construction can be defined using equality or inequality constraints. The k -out-of- N rule [17] allows the construction of a network of N neurons for which the evolution leads to a stable state with "exactly k active neurons among N ". This rule was a major result in ANN for optimization. The corresponding energy function is defined as:

$$E = (k - \sum_{i=1}^N x_i)^2 \quad (2)$$

This function is minimal when the active neuron sum is equal to k , and is positive in the other cases. Equation 2 can be written in the form of equation 1 as follow:

$$E = -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N T_{ij} \cdot x_i \cdot x_j - \sum_{i=1}^N I_i \cdot x_i \quad (3)$$

With $\begin{cases} T_{ij} = -2 \overline{\delta_{i,j}} & \forall i, j \\ I_i = 2k - 1 & \forall i \end{cases}$

$\overline{\delta_{i,j}}$ is the inverse Kronecker function and is equal to zero if $i = j$, or one in the other cases.

Cardeira and Mammeri [5] demonstrate the additive character of the Hopfield model and apply it to a mono-processor architecture. In this case, the scheduling problem is modelled through ANN by the following representation:

- Neurons $N_{i,j}$ are arranged in a $T \times C$ matrix form, where line i corresponds to the task i and the column j corresponds to the schedule time unit j . The number of time units C depends on the hyperperiod of tasks (i.e. the least common multiple of all the task periods) and T is the number of tasks.
- An active neuron $N_{i,j}$ indicates that during the corresponding schedule time unit j , the task i is being executed.
- One line of neurons is added to model the possible inactivity of the processor during schedule times. These neurons are called *slack neurons* since they are not used to represent the solution. The total number of neurons is in this case $N = (T + 1) \times C$.

In the case of an homogeneous multiprocessor architecture, several matrices are organized in layers to model the different execution resources. New slack neurons are then necessary to manage the exclusive execution of each task on resources. Figure 3 presents an example of network with p resource layers. In this case, the total number of neurons increases by a factor p .

By addition of new specific rules on lines or columns of neurons, this model can manage task dependencies. Likewise,

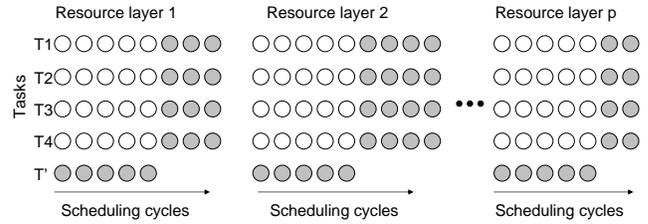


Fig. 3. Classical structure used to model the scheduling problem with ANN. Grey circles represent slack neurons

the preemption can be supported or not by adding new rules on lines. Finally, this model has been extended to take the heterogeneity of SoC architecture into account [7]. In this work, the main drawback concerns the high number of neurons to model the scheduling problem which depends on tasks and on scheduling cycles numbers.

In conclusion, we can note that even if some works are able to manage reconfigurable unit, none of the previous proposals are able to efficiently manage variable multi-tasks scheduling. The main problem concerns the number of tasks that can be supported simultaneously on this resource. This number is not fixed and can evolved according to the area usage rate of each task. In the section III, we show how our proposal efficiently manages this problem.

III. SCHEDULING FOR RECONFIGURABLE HARDWARE USING NEURAL NETWORK

The above mentioned problems lead us to the definition of RANN which focuses on the problem of scheduling an unfixed number of tasks. Our proposal allows to manage instantiation of tasks within the reconfigurable unit. The task model considered here is non-preemptive, this choice is motivated by the low power constraint which imposes the limitation of the number of reconfigurable phases of the reconfigurable unit. Indeed, an important contribution of energy consumption is due to the reconfiguration of this specific unit, thus we propose to disable the preemption of tasks during their executions. Note that the reconfiguration time of each task is included in its global execution time, this assumption simplifies the works but eliminates potential optimizations on scheduling (such as reconfiguration prefetch).

In the following sections, we present firstly how managing an unfixed number of tasks and secondly, how task dependencies are taken into account in our model.

A. Management of a variable number of tasks within the reconfigurable unit

First, let us consider only the scheduling of several tasks according to the area available on the reconfigurable unit. The problem can be formulated as follow:

For a dynamically reconfigurable unit with a total area equal to TA *au* (with *au* the area unit) and a set of tasks $\{T_i\}$ define as:

$$T_i = \{A_i, E_i\} \quad \forall i = 1, \dots, T \quad (4)$$

with A_i the area usage rate of the task i , E_i the execution time of the task i and T the number of tasks. Note that the reconfiguration time of each task, which is proportional to the task's area, is included in the execution time E_i .

The problem is to find all the possible instantiated task combinations which ensure the maximum configuration.

We define the Maximum Configuration (MC) as a configuration which cannot accept a supplementary task, due to the area usage of the reconfigurable unit. In other words, for all non-maximum configurations, the available area is sufficient to accept another task instantiation. So, the MC can be defined as a set of tasks $\{T_i\}$ as follow:

$$MC = \left\{ \begin{array}{l} T_i \mid \sum_{i \in \{1 \dots T\}} A_i \leq TA \\ \wedge \nexists T_j \mid \sum_{i \in \{1 \dots T\}} A_i + A_j \leq TA \end{array} \right\} \quad (5)$$

Let us consider a simple example with a reconfigurable unit having a total area equals to 100 *au* and an application composed of three tasks. Usage rate A_i and execution time E_i of each task on the reconfigurable resource are listed below:

- task T_1 : $A_1 = 20$ *au*; $E_1 = 40$ *tu* (with *tu* the time unit);
- task T_2 : $A_2 = 40$ *au*; $E_2 = 30$ *tu*;
- task T_3 : $A_3 = 80$ *au*; $E_3 = 20$ *tu*;

Table I shows all the possible configurations according to the available reconfigurable area. This table shows that

	Configurations $Conf_i$							
	0	1	2	3	4	5	6	7
Task 1		X		X		X		X
Task 2			X	X			X	X
Task 3					X	X	X	X
Area	0	20	40	60	80	100	120	140
Conf type	iC	iC	iC	MC	MC	MC	IC	IC
Nb of tasks	0	1	1	2	1	2	2	3

TABLE I

LIST OF ALL POSSIBLE COMBINATIONS OF THREE TASKS. MC IS A MAXIMUM CONFIGURATION, IC IS AN IMPOSSIBLE CONFIGURATION DUE TO THE AREA USAGE RATE GREATER THAN THE TOTAL AVAILABLE AREA WITHIN THE RECONFIGURABLE UNIT, AND FINALLY IC IS AN INCOMPLETE CONFIGURATION DUE TO THE EQUATION 5.

configurations 3, 4 and 5 are maximum and defined as:

$$MC = \{\{T_1, T_2\}; \{T_1, T_3\}; \{T_3\}\} \quad (6)$$

All other are either incomplete ($Conf_0, Conf_1, Conf_2$) or impossible ($Conf_6, Conf_7$). For example, in configuration 1, it is possible to instantiate task T_2 or T_3 in the remaining area. For the three maximum configurations, we can notice that the numbers of tasks are equal to 1 or 2. For a mono-thread scheduling problem, a fixed number of tasks must be scheduled at each cycle and classical neural networks can be used. For reconfigurable unit, or multi-threaded processor, the classical solutions cannot be used due to this variable number of tasks.

To schedule an unknown number of tasks, we propose a specific neural network structure. This structure is presented in figure 4. In this figure, each neuron models a task to schedule. An active neuron at cycle t indicates that the corresponding task is running at this cycle. The main idea of the neural

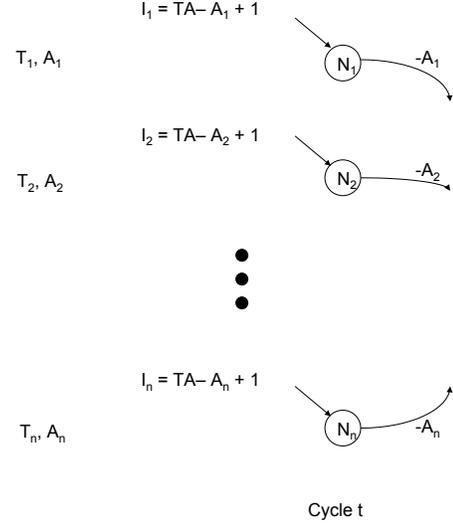


Fig. 4. Neural network structure to schedule a variable number of tasks at each cycle. This figure shows the input and weight values of the neural network to manage the available reconfigurable unit area.

network structure presented in the figure 4 is to allow a neuron activation if the available area is sufficient. Thus the neuron input of task T_i is defined as $I_i = TA - A_i + 1$ which corresponds to the remaining area when the task T_i is instantiated on the reconfigurable unit. It also corresponds to the maximum area that can be used by other tasks to ensure the T_i instantiation. Furthermore, each neuron receives the areas occupy by all other scheduled tasks, this is ensured by the connection weight with a value $-A_i$. The complete connection matrix is define as follow:

$$\mathcal{T} = \begin{pmatrix} 0 & -A_1 & -A_1 & \dots & \dots & -A_1 \\ -A_2 & 0 & -A_2 & \dots & \dots & -A_2 \\ -A_3 & \underline{-A_3} & 0 & \dots & \dots & -A_3 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ -A_n & -A_n & -A_n & \dots & \dots & 0 \\ T_1 & T_2 & T_3 & \dots & \dots & T_n \end{pmatrix} \begin{matrix} T_1 \\ T_2 \\ T_3 \\ \dots \\ \dots \\ T_n \end{matrix} \quad (7)$$

For example, the underline value $-A_3$ is the weight connection value $T_{3,2}$ from task T_3 to task T_2 which indicates that task T_3 needs A_3 *au* within the reconfigurable unit.

B. Management of task dependencies

The proposition above is insufficient to model a real task graph, because no dependency are considered. To support the task dependencies, we propose to complete the previous structure by adding a controller and logical function at each neural input. The goal of the controller is to manage the neuron inputs to ensure that neuron can become active or not.

The task dependencies are modeled as follow:

- Let \mathcal{D} the task dependencies matrix ($T \times T$ matrix size), with $d_{i,j}$ a binary variable equals to 1 if task T_i precede task T_j , and equals 0 elsewhere. Note that $d_{i,i}$ is equal to 0.

- Let \mathcal{F}_t a binary vector (T vector size) which depends on the schedule cycles t and composed of $f_{t,i}$ binary elements equal to 1 if task T_i has finished its executions before schedule time t , or equal 0 elsewhere. For example, if task T_1 (with $E_1 = 40$) starts its execution at schedule time 0, then the binary values $f_{t,1}$ of this task are $f_{t,1} = 0 \forall t < 40$ and $f_{t,1} = 1 \forall t \geq 40$.

- Let \mathcal{X}_t the task execution vector (T vector size), with $x_{t,i}$ an integer variable which represents the number of schedule cycles obtained by the task T_i until the schedule cycle t .

The goal of the controller is to manage the $f_{t,i}$ variable at each cycle. So after each cycle t , the values $x_{t,i}$ are incremented for all active neurons of this cycle t :

$$x_{t,i} = x_{t-1,i} + 1 \quad \forall i \mid N_i \text{ is an active neuron at cycle } t \quad (8)$$

Then at cycle t , the binary variables $f_{t,i}$ is evaluated as:

$$f_{t,i} = \left(x_{t,i} \stackrel{?}{=} E_i \right) \quad (9)$$

with $(a \stackrel{?}{=} b)$ a comparison operation which returns logic value 1 if the test is *true*, 0 elsewhere.

The control input vector \mathcal{C}_t is then computed at each cycle, and defined as follow:

$$\mathcal{C}_t = \mathcal{F}_t \odot \mathcal{D} \quad (10)$$

with \odot the logical *minterme* operator for the matrix and with $c_{t,i}$ defined as:

$$c_{t,i} = \bigodot_{j=1}^T f_{t,j} \vee \overline{d_{i,j}} \quad (11)$$

with \bigodot the symbol the *and-sum* of binary values and \vee the *logical-or* function.

The control, defined by the $c_{t,i}$ variables, is done through an *and-logic* function placed on the neuron input, as shown in figure 5. Finally, due to the expression of the $c_{t,i}$ it is possible to extend the number of inputs of the *logical-and* function of figure 5 and to define the set of neural inputs $\mathcal{R}_{t,i}$ as:

$$\mathcal{R}_{t,i} = \{f_{t,j} \mid d_{i,j} = 1 \forall j = 1, \dots, T\} \quad (12)$$

The figure 6 shows the final structure of our proposal.

C. Example of RANN

To illustrate our proposal, we present our RANN structure for the three tasks defined above. If we consider two task dependencies from tasks T_1 and T_3 to task T_2 , the dependencies matrix is defined as follow:

$$\mathcal{D} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{matrix} T_1 \\ T_2 \\ T_3 \end{matrix} \quad (13)$$

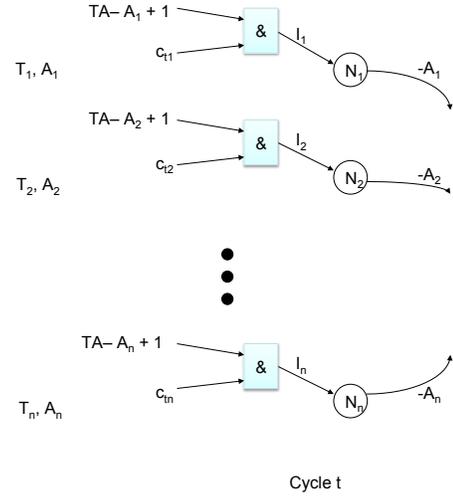


Fig. 5. Adding *and-logic* function in the neural network structure to ensure the management of the task dependencies.

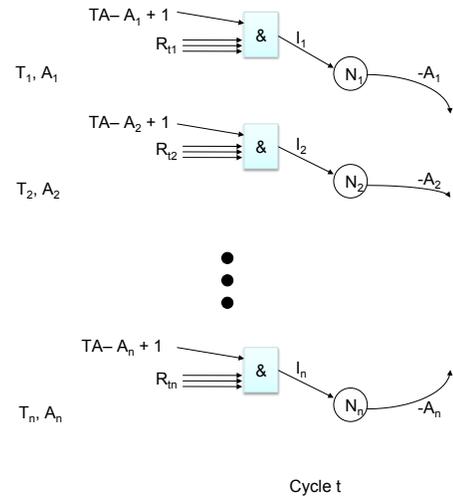


Fig. 6. Final structure of our Reconfigurable Artificial Neural Network proposal. Our proposal is named Reconfigurable ANN (RANN) because its behavior needs neural inputs adaptation at each cycle.

In this case, the control input variables are:

$$\begin{aligned} \mathcal{C}^T &= \begin{pmatrix} c_{t,1} \\ c_{t,2} \\ c_{t,3} \end{pmatrix} = \begin{pmatrix} (f_{t,1} \vee 1) \wedge (f_{t,2} \vee 1) \wedge (f_{t,3} \vee 1) \\ (f_{t,1} \vee 0) \wedge (f_{t,2} \vee 1) \wedge (f_{t,3} \vee 0) \\ (f_{t,1} \vee 1) \wedge (f_{t,2} \vee 1) \wedge (f_{t,3} \vee 1) \end{pmatrix} \\ &= \begin{pmatrix} 1 \\ f_{t,1} \wedge f_{t,3} \\ 1 \end{pmatrix} \end{aligned} \quad (14)$$

Note that for $c_{t,i}$ expression equals to 1, the input function for task T_i is no necessary and the neuron input is simply equal to $TA - A_i + 1$. Furthermore, a simple optimization can reduce the expression of some $c_{t,i}$. In our example, the input control $c_{t,2}$ can be simplified and reduced to expression $c_{t,2} = f_{t,3}$. Indeed, according to the dependencies, the minimum start time of task 2 is equal to the maximum finish time of the previous tasks T_1 and T_3 . In this case, the start time of task

2 is computed as $Max(EndTime(T_1), EndTime(T_3)) = EndTime(T_3)$, so only one dependency is sufficient to ensure a correct scheduling. This optimization can be done if the end times of tasks are known before the execution. In the opposite case. i.e. end times of tasks unknown, this optimization cannot be done and complete expressions of $c_{t,i}$ must be implemented.

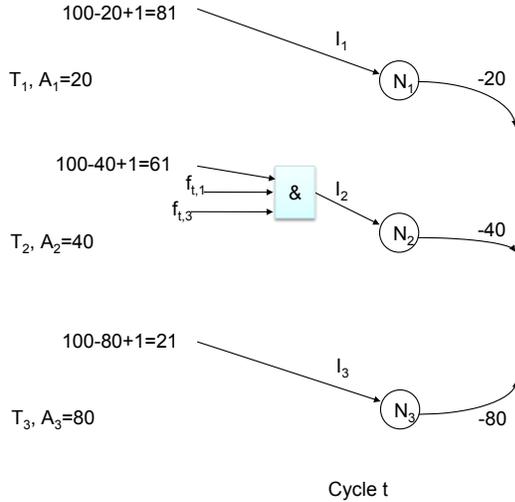


Fig. 7. Example of RANN structure in the case of dependencies between tasks (dependencies from T_1 to T_2 and from T_3 to T_2).

IV. SIMULATION RESULTS

To illustrate the convergence of our proposal, let us consider a simple example with a dynamically reconfigurable unit having a total area equals $50 au$ and an application composed of four tasks defined as follow:

- task T_1 : $A_1 = 10 au$; $E_1 = 40 tu$;
- task T_2 : $A_2 = 20 au$; $E_2 = 20 tu$;
- task T_3 : $A_3 = 10 au$; $E_3 = 30 tu$;
- task T_4 : $A_4 = 40 au$; $E_4 = 20 tu$;

Due to the different execution charges of tasks onto the reconfigurable unit, we propose to define a reconfiguration step equals to the greatest common divisor of each E_i values. In our case, the greatest common divisor, called Reconfigurable Schedule Time (RST), is equal to $10 tu$. In this case, the schedule is compute each 10 cycles, and the RANN is reconfigured each 10 cycles.

For this example, we suppose that there is a dependency between Task 4 and Task 3 ($T_4 \rightarrow T_3$) as defined in the previous section. In this case, the RANN is defined with four neurons and the connection and input values are defined as presented in matrix definition in eq 7. One possible network (neurons are fired randomly in the hopfield model) evolution is illustrated in Figure 8.

RST 1: Due to the dependency, input of neuron $n3$ is forced to value 0. The $n3$ neuron can not be switched to an active state. Then we suppose that neuron $n1$, corresponding

to tasks 1 is evaluated. For this neuron, energy is sufficient to switch it to an active state ($Energy_1 = I_1 = 41$). Then the neuron $n4$ is fired, its energy is equal to $Energy_4 = I_4 - A_1 = 11 - 10 = 1$, so the neuron is switched to an active state. As this configuration is maximum no other neuron can be activated.

RST 2: The evaluation of the new schedule cycle is done between RST 1 and RST 2. The previous neuron states is conserved before starting this new convergence step. So, since tasks 1 and 4 are not finished no input and nor connection weight modifications are applied on the network and the previous neurons states are conserved. The network stays in its previous state, even if the neurons 2 and 3 are evaluated.

RST 3: At this tick, Task 4 resume its execution. To prevent to re-schedule this task, the input value of its neuron $n4$ is forced to '0' and neuron $n4$ switches to an inactive state. At the same time, the register bit r_4 is set to value 1 indicating that Task 4 is completed. The input of neuron 3 is modified and not forced to value 0 anymore. In this case, the convergence of the network switches neurons $n2$ and $n3$ to an active state as the two tasks can be supported by the reconfigurable architecture.

RST 4: For the same reason than RST 2, the network stays stable at this tick: the previous neuron states is conserved because tasks 1 and 2 are not finished. So neurons 1 and 2 stay in an active state.

RST 5: Finally, Tasks 1 and 2 complete their execution, so the inputs of the corresponding neurons are forced to value 0. The network convergence leave the Task 3 in an active state.

Table II shows the number of neuron evaluations for each RST to obtain the convergence. For each RST, the list of

Evaluated neurons	RST: Reconfiguration Schedule Tick				
	1	2	3	4	5
1st neuron	4	-	2	-	3
Neuron evolution	0 → 1	-	0 → 1	-	1 → 1
2th neuron	3	-	1	-	1
Neuron evolution	0 → 0	-	1 → 1	-	1 → 0
3th neuron	1	-	4	-	2
Neuron evolution	0 → 1	-	1 → 0	-	1 → 0
4th neuron	-	-	3	-	-
Neuron evolution	-	-	0 → 1	-	-
Nb of neuron evaluations	3	0	4	0	3

TABLE II
EXAMPLE OF NEURONS EVALUATIONS

evaluated neurons and the state evolution of these neurons are given. The "-" character indicates that the network has converged and stays stable for the next simulation steps. For example, at RST 1, the first evaluated neuron is the neuron number 4. This neuron is switched to an active state. Next, neuron 3 is evaluated and maintained in an inactive state due to its dependence on task 4. Next, neuron 1 is evaluated and switched to an active state. Finally, no other neurons can

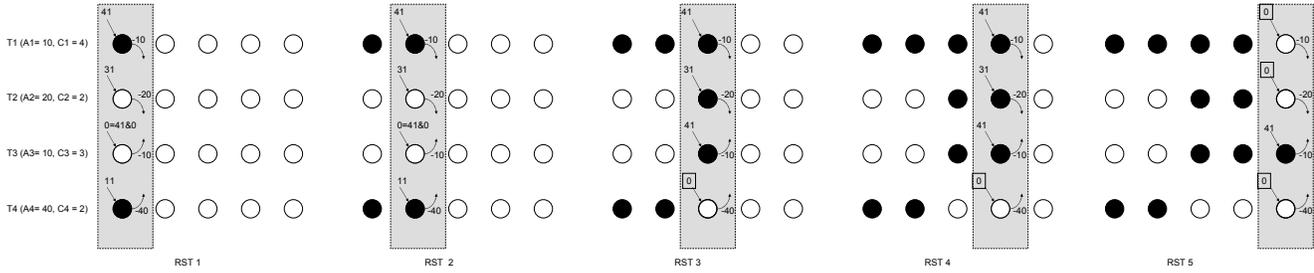


Fig. 8. Example of several RST for a four task scheduling on reconfigurable hardware

change and the network is stable. The last line of the table shows the number of evaluated neurons at each RST for one specific scheduling. These numbers of evaluation is always less than the number of tasks. So, no more than 4 evaluations for this simulation is necessary. For this example, the number of neuron evaluations for the complete scheduling is equal to 11. An equivalent scheduling problem, with the same number of tasks, the same number of RST and modelled by classical neural networks needs more than 100 neuron evaluations. In [5] a very similar example is presented: the scheduling problem with four tasks and height cycles converges after approximatively 200 fired neurons. These results show that our proposal needs approximatively 10 times less evaluations to converge. Furthermore, the number of neurons to model the problem with RANN is equal to 4 while this number is superior to $N_T \times N_C = 20$ for a classical model, this represents a reduction by a factor 5.

Figure 9 shows the evolution of neurons which need to be evaluated to ensure convergence, according to the number of tasks to schedule in the reconfigurable resource. If the tasks are completely independent and if there is no area constraint for the scheduling, the number of neurons to evaluate is done by the line called *minimum constraint*. Generally, some dependencies exist between tasks, and the reconfigurable resource is never dimensioned to ensure the execution of all the tasks simultaneously. So in this case, several constraints can be exploited to limit the number of neurons which need to be evaluated, for example, in an extreme case, the the number of neurons to evaluated is done by the line called *maximum constraint*. This line corresponds to the maximum of constraints (incompatible areas and/or tasks dependencies). in the figure 9, the line called *Intermediate number of constraints* corresponds to a more realistic cases with some dependencies between tasks and some incompatible area placements for tasks. In theses cases, the number os neurons to evaluated is approximatively divided by 2. We can evaluated the dependency constraint by the following metric:

$$DependencyConstraint = \frac{NbTotalTaskDependencies}{NbTasks} \quad (15)$$

The area constraint can be evaluated by the minimal number of tasks which can instantiated due to the total available area,

this can be formulated by :

$$AreaConstraint = \frac{\min(Card(MC_i))}{NbTasks} \quad \forall MC_i \in MC \quad (16)$$

In the example presented in this section, the system is more constrained by area resource than by task dependencies.

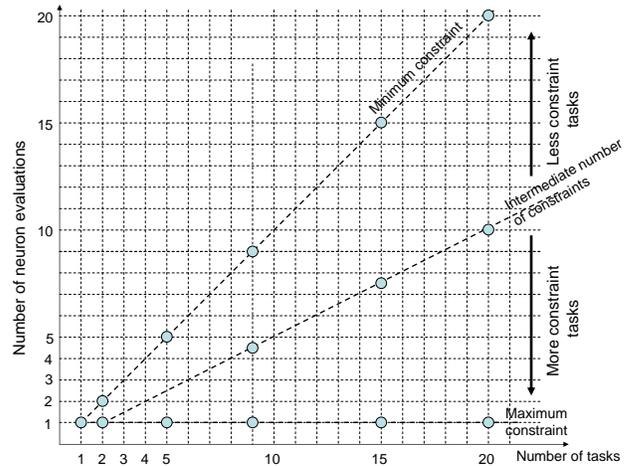


Fig. 9. Evolution of number of evaluated neurons according to the number of tasks to schedule.

In the table III, we compare results obtained with classical neural solutions and our proposal.

Different number of tasks have been evaluated. For each configuration, the area and dependencies constraints are fixed to value 0,5. We estimated that these constraints are representative than classical applications. From this table, we can show that our proposal ensure the neural network convergence in few cycles, while classical solutions need a great number of cycles which mainly due to the number of neurons to model the problem. These results are important in the context of implementation of the scheduling service within the reconfigurable unit. Indeed, the drastically reduction of neurons for the modelization ensures that the implementation of the neural network can be limited. Furthermore, the rapid convergence ensures that it can be possible to manage tasks with in very short time, which ensures the reactivity of the system. This is also an important point, notably because the reconfigurable unit can be the support for new tasks that

Number of tasks	Reconf area size	$\sum A_i$	Area constraint	Dep constraint	Classical modelization		Our proposal	
					Number of neurons NT * NbCycles	Number of cycles for convergence	Number of neurons	Number of cycles for convergence
10	100	200	0,5	0,5	10*100	> 1000	10	\approx 10
20	200	400			20*100	> 2000	20	\approx 20
40	400	800			40*100	> 4000	40	\approx 40
80	800	1600			80*100	> 8000	80	\approx 80
100	1000	2000			100*100	> 10000	100	\approx 100

TABLE III
RESULTS OF LARGE TASK GRAPHS IMPLEMENTATION AND CONVERGENCE

need to be schedule in short time. The adaptability of our proposal (by a simple reconfiguration of neural network) is an interesting characteristic for this type of tasks.

V. CONCLUSION

In this paper, a scheduling service for reconfigurable hardware by neural network is presented. This work take into account a variable number of tasks that could be scheduled in this type of resource. This specificity comes from the different area usage rate of tasks, and needs a scheduling services which can be adapt to the execution context. The classical approaches based on ANNs allow to manage the scheduling with only one active task at each time. This proposition is not satisfying for reconfigurable hardware which is limited by area usage rate of each task and not by the number of tasks. To solve this problem, we propose a Reconfigurable Artificial Neural Network structure that ensures management of tasks according to their area constraints. The main advantage of our RANN is its capacity to converge very quickly, this is due to the very limited number of neurons to model the scheduling problem and the decomposition of the complete and complex scheduling into a sequence of small and simple scheduling steps. This decomposition is done through the definition of the Reconfigurable Schedule Time (RST) which define the tick interval between two reconfiguration steps. To manage precisely the reconfigurable resource, we propose a RST-by-RST adaptation of the network. We show that it is possible to manage tasks at each tick by simple modifications of neuron input values. We also show that our proposition supports task dependencies and limits the number of tasks switching, through the non preemptive model of tasks and a specific control of the neural network.

These contributions are important advances for our current works which consist in defining an efficient hardware implementation of the scheduling service in the context of Reconfigurable SoC. By limitation of the task reconfiguration number, our proposition limits the time overhead and the energy consumption, these two parameters need to be controlled in the context of RSoC.

We are currently working on the FPGA implementation of our proposal. According to the actual density and the new capability of dynamic reconfiguration, FPGA circuits are now good candidate for efficient implementation of the RANN structure. Our first implementation results show that

our proposal is very interesting for the future SoC including reconfigurable unit.

REFERENCES

- [1] D. Abramson, K. Smith, P. Logothetis, and D. Duke. Fpga based implementation of a hopfield neural network for solving constraint satisfaction problems. In *IEEE Design and Test -Special Issue on Reconfigurable Computing*, 17(1):68–83, Jan.-Mar. 2000.
- [2] J. Anderson and A. Srinivasan. Pfair scheduling: Beyond periodic task systems. In *Proc. of the 7th International Conference on Real-Time Computing Systems and Applications*, pages 297–306, Cheju Island, South Korea, december 2000.
- [3] K. Bazargan, R. Kastner, and M. Sarrafzadeh. Fast template placement for reconfigurable computing systems. In *IEEE Design and Test -Special Issue on Reconfigurable Computing*, 17(1):68–83, Jan.-Mar. 2000.
- [4] G. Brebner and O. Diessel. Chip-based reconfigurable task management. *Field-Programmable Logic and Applications*, pages 182–191, 2001.
- [5] C. Cardeira, M. Silva, and Z. Mammeri. Handling precedence constraints with neural network based real-time scheduling algorithms. In *Proc. of the 9th Euromicro Workshop on Real Time Systems*, pages 207–214, Toldeo, Spain, june 1997.
- [6] O. Catoni. Solving scheduling problems by simulated annealing. *SIAM Journal on Control and Optimization*, 36(5):1539–1575, 1998.
- [7] D. Chillet, I.Benkermi, S. Pillement, and O. Sentieys. Hardware task scheduling for heterogeneous soc architectures. In *EUSIPCO 2007, European Signal Processing Conference*, Poznan, Poland, September, 3-7 2007.
- [8] M. Cohen and S. Grossberg. Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE transactions on systems, man, and cybernetics*, 13, no 5:815–826, 1983.
- [9] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri. *Scheduling in Real-Time Systems*. John Wiley & Sons, England, 2002.
- [10] S. Grossberg. *Studies of mind and brain : neural principles of learning, perception, development, cognition and motor control*. D. Reidel publishing Company, 1988.
- [11] B. Hamidzadeh, D. Lilja, and Y. Atif. Dynamic scheduling techniques for heterogeneous computing systems. *Journal of Concurrency: Practice and Experience*, 7:633–652, October 1995.
- [12] J. J. Hopfield and D. W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–52, 1985.
- [13] Y.-H. Lee and C. Chen. A modified genetic algorithm for task scheduling in multiprocessor systems. *Proc. of the Ninth Workshop on Compiler Techniques for High-Performance Computing (CTHPC'2003)*, Taipei, Taiwan, ROC, March 2003.
- [14] D. Liu and Y.-H. Lee. Pfair scheduling of periodic tasks with allocation constraints on multiple processors. In *Proc. of the 18th International Parallel and Distributed Processing Symposium*, volume 03, page 119, Los Alamitos, CA, USA, 2004. IEEE Computer Society.
- [15] J. Noguera and R. M. Badia. Multitasking on reconfigurable architectures: microarchitecture support and dynamic scheduling. *ACM Trans. on Embedded Computing Systems*, 3(2):385–406, may 2004.
- [16] C. Steiger, H. Walder, and M. Platzner. Operating systems for reconfigurable embedded platforms: Online scheduling of real-time tasks. *IEEE Trans. Computers*, 53(11):1393–1407, November 2004.
- [17] G. Tagliarini, J. F. Christ, and W. E. Page. Optimization using neural networks. *IEEE Trans. Comput.*, 40(12):1347–58, December 1991.

Partitioned scheduling of sporadic task systems: an ILP-based approach

Sanjoy K. Baruah
The University of North Carolina
Chapel Hill, NC, USA

Enrico Bini
Scuola Superiore Santa Anna
Pisa, Italy.

Abstract—The multiprocessor partitioned scheduling of sporadic task systems is considered. The problem of obtaining feasible partitionings under both Earliest Deadline First (EDF) and Fixed Priority (FP) scheduling is represented as integer linear programs comprised of binary (zero / one) integer variables only.

I. INTRODUCTION

Real-time systems are often modeled as collections of recurrent tasks, each of which generates a potentially infinite sequence of jobs according to well-defined rules. One commonly-used formal model for representing such recurrent real-time tasks is the *sporadic* task model. Each sporadic task τ_i in this model is characterized by three parameters — a worst-case execution time C_i , a relative deadline D_i , and a period T_i . Such a task generates a potentially infinite sequence of jobs, with successive jobs of τ_i arriving at least T_i time units apart, each job having an execution requirement $\leq C_i$ and a deadline D_i time units after its arrival time.

In this paper, we consider real-time systems that can be modeled as collections of sporadic tasks and are implemented upon a platform comprised of several identical processors. We assume that the processors are fully *preemptive* — an executing job may be interrupted at any instant in time and have its execution resumed later with no cost or penalty.

Partitioned and global scheduling. In *partitioned* scheduling of collections of recurrent tasks, each task is assigned to a processor and all the jobs generated by the task are required to execute upon that processor. In *global* scheduling, on the other hand, different jobs of the same task may execute on different processors and a preempted job may resume execution on a processor different from the one it had been executing on prior to preemption.

Global scheduling is more general than partitioned scheduling (in the sense that every partitioned schedule is also a global schedule while the converse is not true); however, the current state of the art seems to favor partitioned scheduling. There are two major reasons for this. First, the run-time cost of inter-processor migration is unacceptably high on many platforms. Second, it has been shown [3], [4] that current schedulability tests for partitioned scheduling are superior to current schedulability tests for global scheduling. Although research and technology promises to ameliorate both these factors and render them less critical in the future, it is likely

that partitioned scheduling will have a role to play in system design and implementation at present.

This research. We consider the partitioned scheduling of sporadic task systems upon multiprocessor platforms. We study two very widely-used scheduling algorithms — the *Earliest Deadline First* scheduling algorithm (EDF), [20], [11] and *Fixed Priority* scheduling (FP) [20], [2] when scheduling systems of sporadic tasks upon such preemptive platforms.

It is already known (see, e.g., [21]) that EDF- and FP-partitioning are intractable –NP-hard in the strong sense—even for task models simpler than the sporadic task model, since they are a generalization of the well-known bin-packing problem [17], [16]; hence, we do not expect to be able to obtain polynomial-time algorithms for partitioning sporadic task systems. Instead, we describe how the problem of partitioning sporadic task systems upon multiprocessors for EDF or FP scheduling may be formulated as an *integer linear programming* (ILP) problem. Although this may at first seem inconsequential — converting one intractable problem to another (since ILP is also known to be NP-hard [18]) — we believe that there is a real benefit to such conversion. This is because the optimization community has devoted immense effort to coming up with extremely efficient (although still exponential-time) algorithms for solving ILP’s, and highly-optimized libraries implementing these efficient algorithms are widely available. (This is particularly true for ILP’s in which each integer variable is further constrained to take in only the values zero or one — these are called *binary or zero / one integer variables*, and the corresponding ILP’s *zero-one* or *binary* ILP’s (BILP’s); for instance, the optimization library that comes with MATLAB¹ has a very efficient solver for BILP’s, called `bintprog()`). We will derive BILP representations of EDF and FP partitioning. Both BILP formulations use polynomially many integer variables. The BILP formulation of general EDF-partitioning requires exponentially many linear constraints; we will describe how a sufficient (rather than exact) EDF-partitioning algorithm can be formulated as a BILP with only pseudo-polynomially or polynomially many constraints. For FP-partitioning, the exact BILP formulation requires pseudo-polynomially many constraints, while sufficient FP-partitioning can be represented using polynomially many constraints.

¹<http://www.mathworks.com/products/matlab/>

Organization. The remainder of this paper is organized as follows. In Section II, we briefly describe the task and machine model used in this work. In Section III, we discuss how partitioning problems may in general be represented as integer linear programs. In Section IV, we delve deeper into the ILP representation of EDF-partitioning. We describe how to represent EDF-partitioning exactly and approximately using zero-one ILP's, and discuss the tradeoffs that go into choosing the parameters for the approximation algorithms. In Section V, we describe how FP-partitioning can also be represented as an ILP. We conclude in Section VI by placing this work within the larger context of multiprocessor scheduling.

II. MODEL AND DEFINITIONS

As stated in Section I above, a *sporadic task* $\tau_i = (C_i, D_i, T_i)$ is characterized by a *worst-case execution requirement* C_i , a *(relative) deadline* D_i , and a *minimum inter-arrival separation* parameter T_i , also referred to as the *period* of the task. Such a sporadic task generates a potentially infinite sequence of jobs, with successive job-arrivals separated by at least T_i time units. Each job has a worst-case execution requirement equal to C_i and a deadline that occurs D_i time units after its arrival time. We assume a fully *preemptive* execution model: any executing job may be interrupted at any instant in time, and its execution resumed later with no cost or penalty. A *sporadic task system* is comprised of a finite number of such sporadic tasks. It is evident from the definition of sporadic tasks, and of sporadic task systems, that any given system may legally generate infinitely many different collections of jobs.

In the remainder of this paper, we will let τ denote a system of n sporadic tasks: $\tau = \{\tau_1, \tau_2, \dots, \tau_n\}$, with $\tau_i = (C_i, D_i, T_i)$ for all i , $1 \leq i \leq n$. Task system τ is said to be a *constrained* sporadic task system if it is guaranteed that each task $\tau_i \in \tau$ has its relative deadline parameter no larger than its period: $D_i \leq T_i$ for all $\tau_i \in \tau$. We restrict our attention here to *constrained task systems*².

Demand bound function (DBF). For any interval length t , the demand bound function $\text{DBF}(\tau_i, t)$ of a sporadic task τ_i bounds the maximum cumulative execution requirement by jobs of τ_i that both arrive in, and have deadlines within, any interval of length t . It has been shown [9] that

$$\text{DBF}(\tau_i, t) = \max \left(0, \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i \right) \quad (1)$$

Approximation schemes have been defined for computing the value of DBF to any desired degree of accuracy (see, e.g. [1], [12]). Equation 2 below gives such an approximation scheme for DBF; for any fixed value of k , $\text{DBF}^{(k)}(\tau_i, t)$ defines an approximation of $\text{DBF}(\tau_i, t)$ that is exact for the first k steps

²This is primarily for pedantic reasons — although our techniques extend to sporadic task systems that are not constrained in much the same manner that uniprocessor EDF- and FP- scheduling results for constrained systems extend to task systems that are not constrained, the presentation of these extensions involves many grungy details and is likely to detract attention from the main ideas.

of $\text{DBF}(\tau_i, t)$, and an upper bound for larger values of t :

$$\text{DBF}^{(k)}(\tau_i, t) = \begin{cases} \text{DBF}(\tau_i, t) & \text{if } t \leq (k-1)T_i + D_i \\ C_i + (t - D_i)U_i & \text{otherwise} \end{cases} \quad (2)$$

The following lemma provides a quantitative bound on the degree by which $\text{DBF}^{(k)}$ may deviate from DBF:

Lemma 1:

$$\forall t \geq 0$$

$$\text{DBF}(\tau_i, t) \leq \text{DBF}^{(k)}(\tau_i, t) < \left(1 + \frac{1}{k}\right) \text{DBF}(\tau_i, t) .$$

Proof Sketch: This lemma is easily validated informally by sketching $\text{DBF}(\tau_i, t)$ and $\text{DBF}^{(k)}(\tau_i, t)$ as functions of t for given k (see Figure 1). $\text{DBF}(\tau_i, t)$ is a step function comprised of steps of height C_i , with the first step at $t = D_i$ and successive steps exactly T_i time units apart. The graph of $\text{DBF}^{(k)}(\tau_i, t)$ tracks the graph for $\text{DBF}(\tau_i, t)$ for the first k steps, and is a straight line with slope C_i/T_i after that. It is evident from the figure that $\text{DBF}^{(k)}(\tau_i, t)$ is always $\geq \text{DBF}(\tau_i, t)$, and that the ratio $\text{DBF}^{(k)}(\tau_i, t)/\text{DBF}(\tau_i, t)$ is maximized at t just a bit smaller than $kT_i + D_i$, where it is $< (k+1)C_i/(kC_i) = (1 + \frac{1}{k})$ as claimed. ■

III. PARTITIONING AND ILP'S

In an integer linear program (ILP), one is given a set of N variables, some or all of which are restricted to take on integer values only, and a collection of “constraints” that are expressed as linear inequalities over these variables. The set of all points in N -dimensional space over which all the constraints hold is called the *feasible region* for the integer linear program³.

In the following sections we consider a constrained-deadline task system τ comprised of n tasks that is to be partitioned among m unit-speed processors. We will let $\tau(j)$ denote the subset of τ that is assigned to the j 'th processor by the partitioning algorithm, $1 \leq j \leq m$. Thus, $\tau(1), \tau(2), \dots, \tau(m)$ represents a partitioning of τ into m mutually disjoint subsets and comprises the desired output of the partitioning algorithm.

To convert partitioning to a BILP we define $(n \times m)$ binary integer variables X_{ij} , with the following intended interpretation⁴

$$X_{ij} \leftarrow \begin{cases} 1 & \text{if } \tau_i \in \tau(j) \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

That is, $X_{ij} = 1$ if and only if the i 'th task is assigned the j 'th processor. This intended interpretation is enforced by n constraints of the following form, one for each $i \in \{1, \dots, n\}$, encapsulating the requirement that each task must be assigned to some processor:

$$\sum_{j=1}^m X_{ij} = 1 \quad (4)$$

³One is also typically given an “objective function,” also expressed as a linear inequality of the variables, and the goal is to find the extremal (maximum/ minimum) value of the objective function over the feasible region. However, for our purposes here it suffices to construct the constraints and determine whether the feasible region is empty or not.

⁴Recall that by definition, binary integer variables may take on values zero or one only.

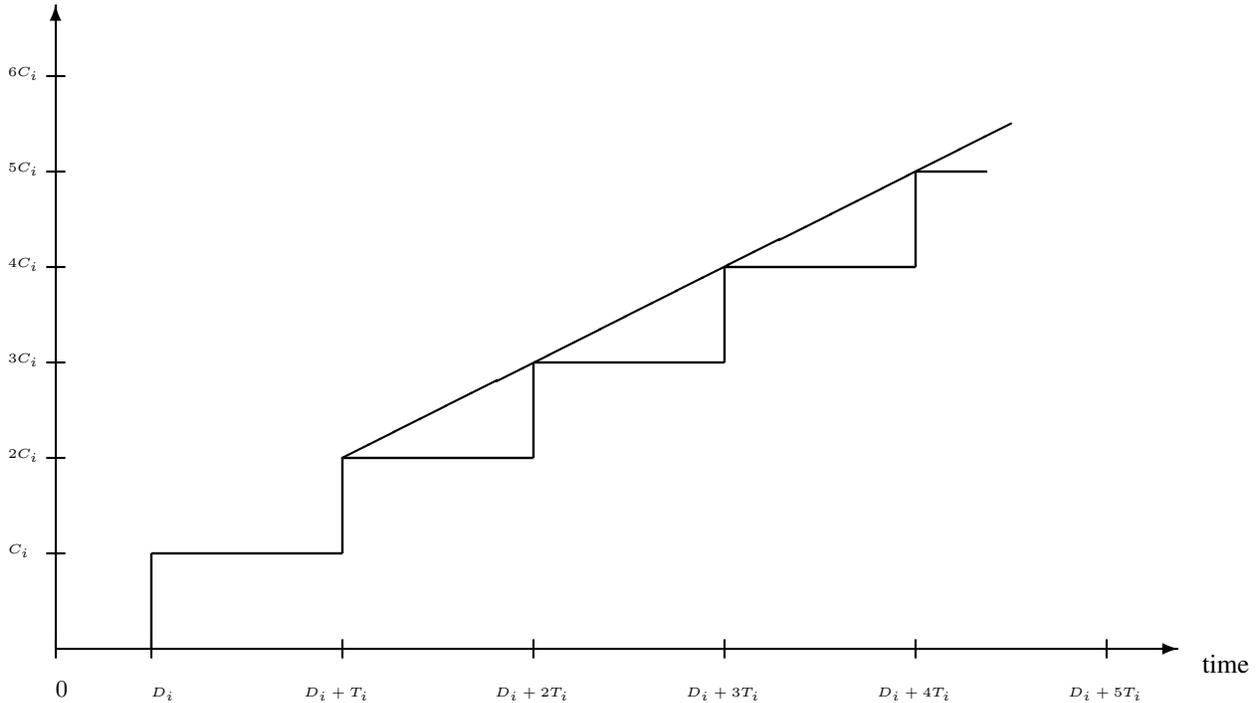


Fig. 1. Illustrating the proof of Lemma 1. The step plot represents $DBF(\tau_i, t)$. The plot for $DBF^{(k)}(\tau_i, t)$, for $k = 2$, is identical to the step plot for $t \leq d_i + T_i$, and is denoted by the straight line for larger t .

The constraints expressed in Equation 4 must be true for partitioning under any scheduling algorithm; in the following two sections, we describe the additional constraints that must be satisfied for the specific scheduling algorithms EDF (Section IV) and FP (Section V).

IV. PARTITIONED-EDF SCHEDULABILITY

In EDF scheduling [20], jobs are assigned priorities according to their absolute deadlines – the earlier the deadline of a job, the greater its priority. Under partitioned-EDF scheduling, all the tasks assigned to the j 'th processor (i.e., all tasks in $\tau(j)$) must be uniprocessor EDF-schedulable, for each j , $1 \leq j \leq m$. Exact schedulability tests for uniprocessor EDF-scheduling are known [9]; these tests essentially require that the following linear constraint be satisfied

$$\sum_{\tau_i \in \tau(j)} DBF(\tau_i, t_o) \leq t_o \quad (5)$$

for each $t_o \in \mathcal{TS}(j)$, where $\mathcal{TS}(j)$ (sometimes called the “testing set”) is a set of numbers defined as follows:

$$\mathcal{TS}(j) = \bigcup_{\tau_i \in \tau(j)} \{t | t \equiv D_i + (k-1)T_i \wedge t \leq P(j)\} . \quad (6)$$

Here, $P(j)$ denotes the least common multiple of the period parameters of all the tasks in $\tau(j)$: $P \stackrel{\text{def}}{=} \text{lcm}_{\tau_i \in \tau(j)} \{T_i\}$.

Observe that Equation 5 is indeed a linear constraint, since for given t_o , $DBF(\tau_i, t_o)$ evaluates to a constant. Hence Equation 5 is of the form $\sum_{i=1}^n A_i X_{ij} \leq t_o$ where each A_i is a constant, and the X_{ij} 's binary integer variables as stated above.

Note that $\tau(j)$ is not known when the ILP is being formulated — indeed, the goal of the ILP is to determine the $\tau(j)$'s. Let P denote the least common multiple of the period parameters of all the tasks in τ — $P \stackrel{\text{def}}{=} \text{lcm}_{i=1}^n \{T_i\}$ — and observe that $\mathcal{TS}(j) \subseteq \mathcal{TS}$, where

$$\mathcal{TS} \stackrel{\text{def}}{=} \bigcup_{i=1}^n \{t | t \equiv D_i + (k-1)T_i \wedge t \leq P\} . \quad (7)$$

Equation 5 is rewritten in a form that makes no reference to the unknown set $\tau(j)$ of tasks, as follows

$$\sum_{i=1}^n DBF(\tau_i, t_o) X_{ij} \leq t_o \quad (8)$$

Observe that Equations 5 and 8 are identical under the intended interpretation of the X_{ij} variables, since X_{ij} equals one for all $\tau_i \in \tau(j)$ and zero for all $\tau_i \notin \tau(j)$.

Since one such constraint must be written for each $t_o \in \mathcal{TS}$ for each processor j , there are $(m \times |\mathcal{TS}|)$ such linear constraints.

Reducing the number of constraints - I

In general, $|\mathcal{TS}|$ may be **exponential** in the representation of the task system; however, it has been shown in [9] that if the total utilization $U(\tau(j)) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau(j)} (C_i/T_i)$ of all the tasks assigned to the j 'th processor is bounded from above by a constant c strictly less than one, then a \mathcal{TS} can efficiently be identified with cardinality pseudo-polynomial in the representation of the task system. By choosing an appropriate value of c (the factors that influence this choice is discussed below) and then adding m linear constraints — one for each j , $1 \leq j \leq m$ — of the form

$$\sum_{i=1}^n X_{ij} \frac{C_i}{T_i} \leq c, \quad (9)$$

we need verify Equation 8 at at most pseudo-polynomially many points t_o for each j , and hence need write at most **pseudo-polynomially** many constraints of the form given in Equation 8.

What determines the choice of c ? As has been shown in [9], mandating such a restriction — that $U(\tau(j)) \leq c$ — is essentially equivalent to decreasing the computing capacity of the processors from 1 to the chosen constant c . Hence while any value of $c < 1$ will provide a pseudo-polynomial time bound, the tradeoff involved in choosing a value for c is as follows: the smaller the value of c , the smaller the value of $|\mathcal{TS}|$, but the larger the fraction of the computing capacity of the processors that remains unused by the partitioning algorithm and hence the more likely that our algorithm will fail to partition a task system that is partitionable.

Reducing the number of constraints - II

Equation 8 being true for all j , $1 \leq j \leq m$, and all $t_o \in \mathcal{TS}$ corresponds to an exact partitioning algorithm. If we were to use $\text{DBF}^{(k)}$ instead of DBF in Equation 8:

$$t_o \geq \sum_{\tau_i \in \tau} \text{DBF}^{(k)}(\tau_i, t_o) X_{ij} \quad (10)$$

the partitioning algorithm is no longer exact: there may be partitionable task systems corresponding to which these constraints are no longer satisfiable (this is because, as Lemma 1 states, $\text{DBF}^{(k)}$ over-estimates the computational demand of each task). However, constructing a partitioning algorithm based on Equation 10 instead of on Equation 8 offers the advantage that the size of the testing set for which Equation 10 must be evaluated is known to be $|\tau(j)| \times k = \mathcal{O}(nk)$, which means that, for a given choice of k there are $\mathcal{O}(nm)$ such linear constraints. Furthermore, we may use Lemma 1 to quantify the degree of inaccuracy of the partitioning algorithm. In this manner, we can write an ILP representation of EDF-partitioning with the number of linear constraints **polynomial** in the representation of the task system.

To summarize the discussion above, we can transform EDF-partitioning to an binary integer linear program on the $n \times m$ binary integer variables X_{ij} with n linear constraints of the form Equation 3 (denoting that each task gets assigned to some processor), and some additional constraints generated according to one of the following three options:

- 1) Exponentially many linear constraints of the form Equation 8. The resulting ILP corresponds to an exact partitioning algorithm — the ILP has a feasible solution if and only if the task system is partitionable on m processors.
- 2) Choose a real-valued constant $c < 1$, and generate
 - a) m constraints of the form Equation 9, and
 - b) pseudo-polynomially many linear constraints of the form Equation 8. The resulting ILP corresponds to a sufficient partitioning algorithm — if the task system is partitionable on m processors by using no more than a fraction c of the computing capacity of each processor, then the ILP has a solution.
- 3) Choose an integer-valued constant of $k > 1$, and generate polynomially many linear constraints of the form Equation 10. The resulting ILP corresponds to a sufficient partitioning algorithm — if the task system is partitionable on m processors by using no more than a fraction $k/(k+1)$ of the computing capacity of each processor, then the ILP has a solution.

We can thus represent the partitioning problem exactly, or approximately to any desired degree of accuracy by generating the appropriate set of linear constraints. In this manner, we get a BILP that can be solved very efficiently (albeit still in worst-case exponential time) using highly optimized solvers (such as the `bintprog()` solver that is a part of the MATLAB optimization toolbox).

V. PARTITIONED-FP SCHEDULABILITY

In FP scheduling, each task is assigned a distinct priority and all the jobs generated by a task inherit the priority of the task that generates it. During run-time, each processor is allocated to the highest-priority job (if any) that needs to use the processor. *We will assume without loss of generality that the tasks are indexed in decreasing priority order* — task τ_1 is assigned the greatest priority, and τ_i has greater priority than τ_{i+1} for all i , $1 \leq i < n$.

Recall that we are restricting our attention in this paper to constrained-deadline sporadic task systems only. For such systems, it is known [19] that the response time of a job $\tau_i \in \tau(j)$ — i.e., assigned to the j 'th processor — is maximized when the job arrives concurrently with a job of each greater-priority task in $\tau(j)$, and each such greater-priority task generates subsequent jobs as soon as allowed (i.e., with successive job arrivals separated by exactly the period parameter of the task). A sequence of job arrivals of a task systems in which each task has a job arrive at the same instant and subsequent jobs as soon as allowed is sometimes called the *synchronous arrival sequence (SAS)* of the task system.

Let us now focus on a particular task $\tau_i \in \tau(j)$. Let $\mathcal{AS}(i, j)$ denote all the time-instants in $[0, D_i)$ at which tasks that have been assigned to processor j have jobs arrive during the SAS: $\mathcal{AS}(i, j)$ contains D_i , as well as all $t < D_i$ of the form $t \equiv (\ell \times T_k)$, for $\ell = 0, 1, 2, \dots$ and $\tau_k \in \tau(j)$. It is known [19] that τ_i meets its deadline if and only if there is a $t_o \in \mathcal{AS}(i, j)$ such that

$$t_o \leq C_i + \sum_{\tau_k \mid k < i \wedge \tau_k \in \tau(j)} \left\lceil \frac{t_o}{T_k} \right\rceil C_k. \quad (11)$$

As in the EDF case (Section IV) we do not know $\tau(j)$ when we are setting up the ILP; hence, we must write m equations as follows, one for each j , $1 \leq j \leq m$, to represent the information conveyed by Equation 11:

$$t_o \leq X_{ij} C_i + \sum_{k=1}^{i-1} \left\lceil \frac{t_o}{T_k} \right\rceil X_{kj} C_k \quad (12)$$

Since t_o , T_k , and C_k are all known, Equation 12 is a linear constraint in the X_{ij} variables. Also, it is known that the cardinality of $\mathcal{AS}(i, j)$ is pseudo-polynomial in the representation of the task system⁵. We thus see that τ_i meets its deadlines under FP-scheduling for a given priority assignment if and only if at least one of pseudo-polynomially many inequalities is satisfied. There is a standard technique in binary integer programming (see Section in the appendix) for expressing the requirement that at least one of a collection of inequalities be satisfied; by using this technique, we can thus express the FP-schedulability of τ_i as a binary integer program. By repeating this procedure for all the n tasks, we express FP-schedulability as a binary integer program on a polynomial number of variables and pseudo-polynomially many constraints.

Reducing the number of constraints

Since $\mathcal{AS}(i, j)$ may contain pseudo-polynomially many points, our BILP has pseudo-polynomially many constraints. Techniques are known [13], [14] for trading off some accuracy to reduce the number of points in $\mathcal{AS}(i, j)$ to a polynomial number; by applying these techniques, we can obtain a BILP representation of a sufficient FP-partitioning algorithm that has only polynomially many constraints.

VI. CONTEXT AND CONCLUSIONS

As real-time embedded systems increasingly come to be implemented upon multiprocessor platforms, it is important that algorithms for efficiently scheduling such systems be obtained. In this work, we have considered the partitioned scheduling of sporadic task systems. Recent research [5], [6], [4], [15], [7], [8] has addressed this topic; however, all this research has been directed at obtaining approximate solutions to the partitioning problem. This work is instead directed at obtaining exact algorithms for partitioning, regardless of computational complexity. Unlike in the case of simpler task models (such as the *implicit-deadline* or “Liu and Layland”

⁵Bini and Buttazzo [10] have devised a technique to identify at most 2^{i-1} points in $\mathcal{AS}(i, j)$ at which Condition 11 must be validated.

task model), in which the design of such an exact algorithm is a straightforward extension of bin-packing, obtaining exact representations of partitioning for constrained-deadline sporadic task systems turned out to not be quite as straightforward.

We have obtained a zero-one ILP representations for both EDF and FP partitioning. Such zero-one ILP’s can be solved very efficiently in practice using widely-available libraries (such as MATLAB’s `binprog()` function). We have shown how the size (the number of constraints) of the ILP can be tuned depending on the degree of accuracy desired, i.e., the fraction of the computing capacity of the processing platform one is willing to “waste.”

ACKNOWLEDGEMENTS

This research has been Supported in part by NSF Grant Nos. CNS-0834270, CNS-0834132, CCF-0541056, and CCR-0615197, ARO Grant No. W911NF-06-1-0425, and funding from IBM and the Intel Corporation.

REFERENCES

- [1] ALBERS, K., AND SLOMKA, F. An event stream driven approximation for the analysis of real-time systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems* (Catania, Sicily, July 2004), IEEE Computer Society Press, pp. 187–195.
- [2] AUDSLEY, N. C., BURNS, A., DAVIS, R. I., TINDELL, K. W., AND WELLINGS, A. J. Fixed priority preemptive scheduling: An historical perspective. *Real-Time Systems* 8 (1995), 173–198.
- [3] BAKER, T. P. Comparison of empirical success rates of global vs. partitioned fixed-priority and EDF scheduling for hard real time. Tech. Rep. TR-050601, Department of Computer Science, Florida State University, 2005.
- [4] BAKER, T. P. A comparison of global and partitioned EDF schedulability tests for multiprocessors. In *Proceeding of the International Conference on Real-Time and Network Systems* (Poitiers, France, 2006).
- [5] BARUAH, S., AND FISHER, N. The partitioned multiprocessor scheduling of sporadic task systems. In *Proceedings of the IEEE Real-Time Systems Symposium* (Miami, Florida, December 2005), IEEE Computer Society Press.
- [6] BARUAH, S., AND FISHER, N. The partitioned scheduling of sporadic real-time tasks on multiprocessor platforms. In *Proceedings of the Workshop on Compile/Runtime Techniques for Parallel Computing* (Oslo, Norway, June 2005).
- [7] BARUAH, S., AND FISHER, N. The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems. *IEEE Transactions on Computers* 55, 7 (July 2006), 918–923.
- [8] BARUAH, S., AND FISHER, N. The partitioned dynamic-priority scheduling of sporadic task systems. *Real-Time Systems: The International Journal of Time-Critical Computing* 36, 3 (2007), 199–226.
- [9] BARUAH, S., MOK, A., AND ROSIER, L. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium* (Orlando, Florida, 1990), IEEE Computer Society Press, pp. 182–190.
- [10] BINI, E., AND BUTTAZZO, G. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers* 53, 11 (2004), 1462–1473.
- [11] DERTOUZOS, M. Control robotics : the procedural control of physical processors. In *Proceedings of the IFIP Congress* (1974), pp. 807–813.
- [12] FISHER, N., BAKER, T., AND BARUAH, S. Algorithms for determining the demand-based load of a sporadic task system. In *Proceedings of the International Conference on Real-time Computing Systems and Applications* (Sydney, Australia, August 2006), IEEE Computer Society Press.
- [13] FISHER, N., AND BARUAH, S. A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems* (Palma de Mallorca, Balearic Islands, Spain, July 2005), IEEE Computer Society Press, pp. 117–126.

- [14] FISHER, N., AND BARUAH, S. An approximation scheme for feasibility analysis in static-priority systems with bounded relative deadlines. *Journal of Embedded Computing* (2007). Accepted for publication.
- [15] FISHER, N., BARUAH, S., AND BAKER, T. The partitioned scheduling of sporadic tasks according to static priorities. In *Proceedings of the EuroMicro Conference on Real-Time Systems* (Dresden, Germany, July 2006), IEEE Computer Society Press.
- [16] JOHNSON, D. Fast algorithms for bin packing. *Journal of Computer and Systems Science* 8, 3 (1974), 272–314.
- [17] JOHNSON, D. S. *Near-optimal Bin Packing Algorithms*. PhD thesis, Department of Mathematics, Massachusetts Institute of Technology, 1973.
- [18] KARP, R. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, R. Miller and J. Thatcher, Eds. Plenum Press, New York, 1972, pp. 85–103.
- [19] LEHOCZKY, J., SHA, L., AND DING, Y. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the Real-Time Systems Symposium - 1989* (Santa Monica, California, USA, Dec. 1989), IEEE Computer Society Press, pp. 166–171.
- [20] LIU, C., AND LAYLAND, J. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM* 20, 1 (1973), 46–61.
- [21] SARKAR, V. *Partitioning and scheduling parallel programs for execution on multiprocessors*. MIT Press, 1989.

APPENDIX

EXPRESSING DISJUNCTS AS BINARY INTEGER PROGRAMS

In this section, we illustrate by an example the technique for expressing the requirement that at least one of a given set of inequalities be true, within the syntactic limitations of a binary integer program.

Suppose that we require that at least one of the four inequalities

$$\begin{aligned}
 A_1X &\leq b_1 \\
 A_2X &\leq b_2 \\
 A_3X &\leq b_3 \\
 A_4X &\leq b_4
 \end{aligned}$$

be satisfied. Let Z denote a very large positive constant. Introduce the two binary integer variables Y_1 and Y_2 (in general, the number of such variables that must be introduced is logarithmic in the number of inequalities). Replace the above 4 inequalities by the following:

$$\begin{aligned}
 A_1X &\leq b_1 + ((1 - Y_1) + (1 - Y_2))Z \\
 A_2X &\leq b_2 + (Y_1 + (1 - Y_2))Z \\
 A_3X &\leq b_3 + ((1 - Y_1) + Y_2)Z \\
 A_4X &\leq b_4 + (Y_1 + Y_2)Z
 \end{aligned}$$

For any assignment of values to the newly-introduced binary variables Y_1 and Y_2 , it is evident that all but one of these equalities is trivially satisfied (since the RHS is $\geq Z$), while the remaining inequality is exactly equivalent to the corresponding inequality in the original problem (i.e., prior to this transformation). Hence, all four of the latter inequalities is satisfiable if and only if at least one of the original four inequalities is satisfiable; we have thus converted the problem of satisfying at least one of the given set of four inequalities to a “conventional” problem of satisfying a set of all four inequalities.

A New Task Model and Utilization Bound for Uniform Multiprocessors

Shelby Funk

Department of Computer Science, The University of Georgia

Abstract

This paper introduces a new model for describing jobs and tasks for real-time systems. Using this model can improve utilization bounds on uniform multiprocessors, in which each processor has an associated speed. Traditionally, a job executing on a processor of speed s for t units of time will perform $s \times t$ units of work. However, this uniform scaling only occurs if tasks are completely CPU bound. In practice, tasks will have some portion of execution that does not scale with the increased CPU speed. Dividing the execution into CPU execution and fixed execution allows the scheduler to place CPU bound tasks on faster processors, as they can take full advantage of the extra speed. This model is used in a new utilization test for EDF scheduling with restricted migration, r-EDF, in which tasks are allowed to migrate, but only at job boundaries. The test is proven to be better than the existing r-EDF test for uniform multiprocessors.

Keywords: hard real-time systems, periodic tasks, earliest deadline first, uniform heterogeneous multiprocessors, migration, memory bound tasks

1 Introduction

In real-time systems, all jobs have deadlines and missing a deadline is considered a system failure. Before these systems can be run, tests must be performed that ensure that no jobs will miss their deadlines. These tests must account for the worst-case behavior of the system, which is a function of both the worst-case arrival configuration of jobs, and their worst-case execution times (WCET). In general, this WCET is estimated and any estimate must provide an *upper bound* of the actual execution time. In order to allow systems to utilize as much of the processor as possible, the estimate of the WCET should be as accurate as possible.

This paper introduces a new task model for scheduling analysis on uniform multiprocessors and uses this model to improve a known schedulability test. This model applies specifically to systems in which processor speeds

may vary. It has been explored in the context of dynamic voltage scaling (DVS) processors [7]. To our knowledge, this model has not been used in the context of uniform multiprocessors.

In uniform multiprocessors, each processor has an associated speed s . In traditional analysis, we assume that if a job executes on a processor of speed s for t time units, then $s \times t$ units of work are performed. This assumption is conditional on all jobs scaling perfectly to CPU speed. In reality, jobs have portions of time that are CPU bound and other portions, such as memory accesses, that do not scale with the CPU speed. This paper explores the power of dividing the WCET into two portions and performing schedulability analysis assuming the CPU portion executes more quickly on faster processors, but the “fixed” portion does not. Because we must account for worst case behavior, if we do not distinguish between the CPU portion and fixed portion but assume perfect scaling with the CPU speed, the given WCET must be increased to account for the fact that only part of the execution actually scales with the CPU speed.

Example 1 Assume J 's CPU execution time is 4 and fixed execution time is 2 and J can execute either on a processor of speed 1 or on a processor of speed 2. On the speed-1 processor, J takes 6 time units to complete. On the speed-2 processor, J takes 4 time units to complete because the CPU execution can be completed in half the time. If we were assign J a single execution requirement, we would have to say J 's WCET is 8. If we say J 's WCET is 6, then we would calculate that J requires only 3 time units when executing on a speed-2 processor, which could result in a deadline miss.

This paper provides an r-EDF-schedulability test using CPU and fixed execution times. r-EDF uses a uniprocessor EDF scheduling algorithm on each processor. This algorithm allows tasks to migrate, but only at job boundaries, when the overhead due to migration is smallest. This model allows for better load balancing among the processors [1], while still putting a bound on the level of migration.

This paper is organized as follows. Section 2 introduces our model and definitions. Section 3 presents a new test for r-EDF scheduling on uniform multiprocessors. Section 4 proves that finding the exact bound presented in Section 3 is NP-complete and provides an ILP formulation to calculate the value as well as providing approximation methods that efficiently find an upper bound by relaxing the problem assumption. Finally, Section 5 concludes and discusses future work.

2 Model and Definition

The analysis that follows assumes we are executing a set of n periodic [5] or sporadic [2, 6] tasks $\tau = \{T_1, T_2, \dots, T_n\}$ on an m -processor uniform multiprocessor $\pi = [s_1, s_2, \dots, s_m]$, where $s_1 \geq s_2 \geq \dots \geq s_m$. In a mild abuse of notation, we allow s_i to denote both the i^{th} processor and the speed of the i^{th} processor. $S(\pi)$ denotes the total speed of π – i.e., $\sum_{i=1}^m s_i$.

Tasks are described by the 3-tuple $T_i = (p_i, e_{cpu,i}, e_{fixed,i})$, where $p_i, e_{cpu,i}$ and $e_{fixed,i}$ are T_i 's period, CPU execution time, and fixed execution time, respectively. We assume $e_{cpu,i}$ scales with processor speed, but $e_{fixed,i}$ does not – if T_i executes on processor s_j , it will require $e_{cpu,i}/s_j + e_{fixed,i}$ time units to complete. Periodic and sporadic tasks generate jobs, and we let $T_{i,j}$ denote the j^{th} job of task T_i . Each job will have CPU and fixed execution requirements $e_{cpu,i}$ and $e_{fixed,i}$, respectively. If T_i is a periodic task, then it will release a new job at times $r_{i,j} = j \cdot p_i$, where $j = 0, 1, 2, \dots$. If T_i is a sporadic task, then consecutive jobs will be released at least p_i time units apart. For both types of tasks, $T_{i,j}$'s deadline is $r_{i,j} + p_i$.

The utilization of T_i is also split into a CPU portion, $u_{C,i} = e_{cpu,i}/p_i$, and a fixed portion, $u_{F,i} = e_{fixed,i}/p_i$. These values measure the proportion of processor time the CPU execution and fixed executed will require, respectively, when executing on a speed-1 processor. Because e_{cpu} scales and e_{fixed} does not, the required proportion of CPU execution decreases as speed increases, but the proportion of fixed execution does not. Thus, by the optimality of EDF on uniprocessors [5], a task set $\tau = \{T_1, T_2, \dots, T_n\}$ is EDF-schedulable on a speed- s uniprocessor if $\sum_{i=1}^n (u_{C,i}/s + u_{F,i}) \leq 1$ – or, equivalently, if $\sum_{i=1}^n (u_{C,i} + s \cdot u_{F,i}) \leq s$. Hence, we define T_i 's total utilization on s_j as follows $u_{i,j} = u_{C,i} + s_j \cdot u_{F,i}$. The total CPU utilization of τ , denoted $U_{cpu}(\tau)$, is $U_{cpu}(\tau) = \sum_{i=1}^n u_{C,i}$.

If s_k is executing tasks $T_{i_1}, T_{i_2}, \dots, T_{i_r}$, then the s_k 's slack, denoted $slack_k$, is equal to $s_k - \sum_{j=1}^r u_{i_j,k}$. This is the maximum total utilization that can be added to s_k without incurring a deadline miss.

A task T_i is said to be *present* on processor s_k at time

t if that task's utilization contributes to the calculation of $slack_k$ – i.e., if some job $T_{i,j}$ was assigned to processor s_k and $r_{i,j} \leq t < r_{i,j} + p_i$. An assignment, $a(\cdot)$ is a mapping of tasks to processors – if $a(i) = k$ at time t then T_i is active on processor s_k at time t . At times, we need to consider the set $\tau \setminus \{T_i\}$ – i.e., τ without some task T_i . We denote this set τ_{-i} .

We let $Pack_{\pi,\tau}$ denote the set of all possible ways of assigning the tasks of τ to the processors of π without allowing slack to be negative on any processor. Finally, we let $MP_{\pi,\tau}$ denote the maximum sum of $s_{a(j)} \cdot u_{F,j}$ that can be achieved for any assignment $a(\cdot)$.

$$MP_{\pi,\tau} = \max_{a \in Pack_{\pi,\tau}} \left\{ \sum_{j \in \tau} s_{a(j)} \cdot u_{F,j} \right\}$$

Intuitively, $MP_{\pi,\tau}$ is the maximum processing capacity that might be wasted due to letting tasks with a large proportion of fixed execution execute on the fastest processors.

3 A New Test for r-EDF on Uniform Multiprocessors

Using the model described above, we can improve the schedulability test for r-EDF on uniform multiprocessors. Theorem 1 below gives the new schedulability test. Corollary 1 proves that theorem provides a tighter bound than the previous schedulability test.

Theorem 1 *Let π be any m -processor uniform multiprocessor and let $\tau = \{T_1, T_2, \dots, T_n\}$ be any periodic or sporadic task set. Define $M_{\pi,\tau}$ as follows.*

$$M_{\pi,\tau} = \max_{1 \leq i \leq n} \left\{ (m-1)u_{C,i} + S(\pi) \cdot u_{F,i} + MP_{\pi,\tau_{-i}} \right\} \quad (1)$$

If the following inequality is satisfied

$$U_{cpu}(\tau) \leq S(\pi) - M_{\pi,\tau} \quad (2)$$

then τ is r-EDF-schedulable on π .

Proof: (By contradiction.) Assume a job T_i generates a job at a time t when all of the processors of π do not have enough slack to admit T_i safely. Let t_o be the earliest time at which this occurs. By the optimality of EDF on uniprocessors all jobs with deadlines at or before t_o met their deadlines.

Let $P(t_o)$ be the set of tasks that are present on some processor at time t_o and let $a_o(\cdot)$ denote the mapping of tasks to processors at time t_o – if $T_j \in P(t_o)$ then T_j is executing on processor $s_{a_o(j)}$. Because T_i has a job that has just arrived and all earlier jobs of T_i met their deadlines, T_i can not be in $P(t_o)$. On the other hand, every

task of τ other than T_i may be in $P(t_o)$ and these tasks may be assigned to processors in a way that causes the maximum total utilization on π . Therefore, the following is an upper bound for $\sum_{T_j \in P(t_o)} u_{j,a_o(j)}$.

$$\begin{aligned} & \max_{a \in \text{Pack}_{\pi, \tau-i}} \left\{ \sum_{T_j \in \tau-i} (u_{C,j} + s_{a(j)} \cdot u_{F,j}) \right\} \\ &= \sum_{T_j \in \tau-i} u_{C,j} + MP_{\pi, \tau-i} \\ &\leq \sum_{T_j \in \tau-i} u_{C,j} + M_{\pi, \tau} - (m-1)u_{C,i} - S(\pi)u_{F,i}. \end{aligned}$$

The last step follows from the definition for $M_{\pi, \tau}$.

Because $\text{slack}_k < u_{i,k}$ for all $k, 1 \leq k \leq m$,

$$\begin{aligned} \sum_{k=1}^m \text{slack}_k &< \sum_{k=1}^m u_{i,k} \\ &= m \cdot u_{C,i} + S(\pi) \cdot u_{F,i} \end{aligned}$$

Note that the total slack is the difference between $S(\pi)$ and $\sum_{T_j \in P(t_o)} u_{j,a_o(j)}$. Hence, the following is a lower bound for $\sum_{T_j \in P(t_o)} u_{j,a_o(j)}$.

$$S(\pi) - m \cdot u_{C,i} - S(\pi) \cdot u_{F,i}$$

Combining these two bounds gives

$$\begin{aligned} \sum_{T_j \in \tau-i} u_{C,j} &> S(\pi) - u_{C,i} - M_{\pi, \tau} \\ \Rightarrow U_{cpu}(\tau) &> S(\pi) - M_{\pi, \tau}, \end{aligned}$$

which contradicts the condition of the lemma. ■

The previous utilization bound for r-EDF-scheduling on uniform multiprocessors [3, 4] is an immediate corollary to Theorem 1 above.

Corollary 1 Any periodic task set τ satisfying

$$U_{sum}(\tau) \leq S(\pi) - (m-1)u_{max}(\tau) \quad (3)$$

will meet all its deadlines when scheduled on uniform multiprocessor π using r-EDF.

Proof: Assume the above condition holds. Because execution is not divided into e_{cpu} and e_{fixed} , we have to set $u_i = e_i/p_i$ to its maximum value in order to ensure all tasks will meet their deadlines. For all tasks T_i , the maximum value of the total utilization on any processor is $u_{i,1} = u_{C,i} + s_1 u_{F,i}$. Thus, $U_{sum}(\tau) = U_{cpu}(\tau) + s_1 \sum_{j=1}^n u_{F,j}$ and $u_{max}(\tau) = \max_{1 \leq j \leq n} \{u_{C,j} + s_1 \cdot u_{F,j}\}$. Substituting these identities into Condition 3 gives the following upper bound for $U_{cpu}(\tau)$.

$$\begin{aligned} & S(\pi) - \max_{1 \leq i \leq n} \{(m-1)u_{C,i} + (m-1)s_1 u_{F,i} + s_1 \sum_{j=1}^n u_{F,j}\} \\ &= S(\pi) - \max_{1 \leq i \leq n} \{(m-1)u_{C,i} + m s_1 u_{F,i} + s_1 \sum_{T_j \in \tau-i} u_{F,j}\} \\ &\leq S(\pi) - \max_{1 \leq i \leq n} \{(m-1)u_{C,i} + S(\pi)u_{F,i} + MP_{\pi, \tau-i}\}. \end{aligned}$$

The final inequality is the condition of Theorem 1. ■

In the last step, the bound increases by at least

$$(m \cdot s_1 - S(\pi))u_{F,i} + s_1 \sum_{T_j \in \tau-i} u_{F,j} - MP_{\pi, \tau-i},$$

and perhaps more if the index i associated with the maximum values changes. If s_1 is significantly faster than the other processors of π , this could be a significant savings. For example, if $\pi = [2, 1, 1, 1]$, then the coefficient of $u_{F,i}$ is reduced from 8 to 5. It is hard to determine how much the entire bound changes because $MP_{\pi, \tau-i}$ is difficult to compute. The next section explores methods of calculating this value.

4 Calculating the Utilization Bound

Unfortunately, in Theorem 2 below, we prove that finding the exact bound presented in Theorem 1 is an NP-hard problem. The NP-hardness stems from finding the packing that results in the largest value for $\sum_{T_j \in \tau-i} s_{a(j)} \cdot u_{F,j}$ among all possible packings. Therefore, we need to employ approximation strategies to find an upper bound for this expression.

First, we prove that finding $MP_{\pi, \tau-i}$ is NP-hard by proving the following decision problem is NP-complete.

THE MIXED TASK PENALTY PROBLEM: Given a uniform multiprocessor $\pi = [s_1, s_2, \dots, s_m]$, a periodic or sporadic task set $\tau = \{T_1, T_2, \dots, T_n\}$, and a number K , does there exist an assignment $a : \tau \rightarrow \pi$ of tasks to processors such that $\sum_{i=1}^n s_{a(i)} \cdot u_{F,i} \geq K$?

Theorem 2 THE MIXED TASK PENALTY PROBLEM is NP-complete.

Proof: This problem is clearly in NP. We will show the bin packing problem is reducible to the mixed task penalty problem. The bin packing problem is stated as follows.

Given a positive integer M , a finite set I of ℓ items, where each $x_i \in I$ has a size $w(x_i)$ such that $0 < w(x_i) \leq 1$, can all the items in I be placed into M unit-sized bins so that to total sized of the items in each bin is at most 1?

Given any bin packing instance we let π be comprised of M speed-1 processors and let $|\tau| = \ell$ and for each $x_i \in I$, the associated task $T_i \in \tau$ has fixed utilization $u_{F,i} = w(x_i)$ and CPU utilization $u_{C,i} = 0$. Finally, we let $K = \sum_{i=1}^{\ell} w(x_i)$. Then the maximum sum in the mixed task penalty problem is K if and only if all the items in I can fit into M bins. ■

Even though the mixed task penalty problem is NP-complete, we can find $M_{\pi, \tau}$ (which requires finding $MP_{\pi, \tau}$) using an integer linear program (ILP).

4.1 Finding $M_{\pi,\tau}$ Using an ILP

The ILP has $2 \times m \times n$ integer variables, $x_{i,j}$ and $y_{i,j}$, where $1 \leq i \leq n$ and $1 \leq j \leq m$. The variable $x_{i,j}$ indicates T_i is assigned to processor s_j . The one exception is if $y_{i,j}$ also equals 1. In this case, T_i is the task that maximizes $M_{\pi,\tau}$. Below, we present the constraints for the ILP.

The first constraint ensures none of the processors are over filled. For each processor s_j , we have the following constraint.

$$\sum_{i=1}^n (x_{i,j} - y_{i,j})u_{i,j} \leq s_j$$

The remaining constraints ensure the values of $x_{i,j}$ and $y_{i,j}$ are valid. The first constraint below ensures each task is assigned to only one processor. The second constraint ensures only one task is *not* assigned to any processor. The third constraint ensures $y_{i,j}$ removes a task from the processor it was assigned to. Note that even though the first two constraints are not set equal to 1, their values will both be 1, as making these values as large as possible increases the objective function.

$$\begin{aligned} \sum_{j=1}^m x_{i,j} &\leq 1 \\ \sum_{i=1}^i \sum_{j=1}^m y_{i,j} &\leq 1 \\ \sum_{j=1}^m (x_{i,j} - y_{i,j}) &\geq 0 \end{aligned}$$

Finally, we maximize the objective function.

$$\sum_{i=1}^n \sum_{j=1}^m [s_j u_{F,i} (x_{i,j} - y_{i,j}) + ((m-1)u_{C,i} + S(\pi)u_{F,i})y_{i,j}]$$

If the ILP is able to find a solution, it will be the maximum possible value of $(m-1)u_{C,i} + S(\pi)u_{F,i} + MP_{\pi,\tau-i}$ – i.e., the ILP objective function will equal $M_{\pi,\tau}$.

Unfortunately, ILPs can be unstable and a solution might not be found even if one exists. If this occurs, the ILP can be relaxed into an LP. The LP constraints and objective function would be the same. However, the variables x and y might take on fractional values. If this happens, it can be viewed as allowing jobs generated by the given task to migrate among processors. Below, we discuss another method for bounding $M_{\pi,\tau}$ by allowing jobs to migrate.

4.2 Bounding $M_{\pi,\tau}$ in Polynomial Time

The mixed task penalty problem is similar to the 0-1 knapsack problem, in which items are given values and weights and we want to select items to place into a knapsack that so that a weight limit is not exceeded and the value is maximized. The difference is that we have multiple knapsacks and the values and weights of the items vary depending on which knapsack we use – if task T_i is assigned to processor s_j , we give T_i a value of $s_j \cdot u_{F,i}$ and a weight of $u_{i,j}$.

Even so, we can use knowledge about the knapsack problem to find an upper bound for $MP_{\pi,\tau}$. Specifically, we know that the fractional knapsack problem, which allows fractions of items to be selected, can be solved in polynomial time using a greedy algorithm. The greedy algorithm sorts the items by the ratio of value to weight and selects the item with the largest ratio to put in the knapsack repeatedly until that item won't fit, at which point the fraction of the item that makes the total weight equal to the limit is selected.

The greedy solution to the fractional knapsack problem cannot be directly applied to the mixed task penalty problem because, once again, the value and the weight both vary depending on which processor the job is assigned to. However, we can use this idea to create the algorithm $fracM_{\pi,\tau}$, which is shown in Figure 1. Given an assignment of tasks to processors, $a(\cdot)$, let $value(a)$ be the sum of $s_{a(i)}u_{F,j}$ over all tasks. Let $MaxSum_{\pi,\tau}$ be the maximum value of any assignment when tasks can be assigned to multiple processors. The algorithm $fracM_{\pi,\tau}$ finds $MaxSum_{\pi,\tau}$ by using the fractional knapsack technique on one processor at a time, always working with the fastest processor that has remaining slack.

We first note that, even though the value and weight of tasks varies between processors, we can sort tasks once by $u_{F,i}/u_{C,i}$ to get the proper ratio of value to weight.

Lemma 1 *Let T_i and T_j be any two tasks. Then, for any processor s_k , we have*

$$\frac{s_k u_{F,i}}{u_{i,k}} \leq \frac{s_k u_{F,j}}{u_{j,k}} \text{ if and only if } \frac{u_{F,i}}{u_{C,i}} \leq \frac{u_{F,j}}{u_{C,j}}.$$

Proof:

$$\begin{aligned} \frac{u_{F,i}}{u_{C,i}} &\leq \frac{u_{F,j}}{u_{C,j}} \\ \Leftrightarrow \frac{u_{C,i}}{u_{F,i}} &\geq \frac{u_{C,j}}{u_{F,j}} \\ \Leftrightarrow \frac{u_{C,i} + s_k u_{F,i}}{u_{F,i}} &\geq \frac{u_{C,j} + s_k u_{F,j}}{u_{F,j}} \\ \Leftrightarrow \frac{u_{F,i}}{u_{i,k}} &\leq \frac{u_{F,j}}{u_{j,k}} \end{aligned}$$

■

$fracM_{\pi,\tau}(\pi, \tau)$

1. $MaxSum \leftarrow 0$
2. for $i \leftarrow 1$ to n do
3. $r(i) \leftarrow u_{F,i}/u_{C,i}$
4. sort τ in decreasing order by $r(i)$
5. for $j \leftarrow 1$ to m
6. $slack \leftarrow s_j$
7. while $slack > 0$ and $\tau \neq \emptyset$
8. $T_i \leftarrow$ task with maximum value of $r(i)$
9. Remove T_i from τ
10. If $u_{i,j} > slack$
11. $\alpha \leftarrow slack/u_{i,j}$
12. $T'_i \leftarrow (u'_{C,i}, u'_{F,i}) \leftarrow (1 - \alpha)(u_{C,i}, u_{F,i})$
13. Add T'_i to τ
14. else
15. $alpha \leftarrow 1$
16. $slack \leftarrow slack - \alpha u_{i,j}$
17. $MaxSum \leftarrow MaxSum + \alpha s_j u_{F,i}$
18. return $MaxSum$

Figure 1: Algorithm $fracM_{\pi,\tau}$

Algorithm $fracM_{\pi,\tau}$ sorts tasks by the ratio of fixed utilization to CPU utilization (lines 2 through 4). It then uses the fractional knapsack solution to assign tasks to processors in decreasing order of processor speed (lines 5 through 17). If a task is selected that does not fit on the current processor (i.e., $u_{i,j} > slack_j$), it divides the task into two pieces and assigns one piece to the current processor and the remaining piece is added to τ (lines 10 through 14). Let αT_i denote a task with CPU and fixed utilization equal to $\alpha u_{C,i}$ and $\alpha u_{F,i}$, respectively. Specifically, if $u_{i,j} > slack_j$ we let $\alpha = slack_j/u_{i,j}$ and assign αT_i to s_j and add $(1 - \alpha)T_i$ to τ to be assigned to the next processor. This reduces $slack_j$ to 0 so $fracM_{\pi,\tau}$ begins assigning tasks to s_{j+1} . Note that splitting tasks this way will have no effect on the value of $r(i)$ used to determine the order in which tasks are added to processors. Finally, the algorithm $fracM_{\pi,\tau}$ returns the sum of $s_{a(i)}u_{F,i}$ that results from this assignment. Below, we prove that $fracM_{\pi,\tau}$ returns $MaxSum_{\pi,\tau}$.

Theorem 3 Let $\tau = \{T_1, T_2, \dots, T_n\}$ be any periodic task set and $\pi = [s_1, s_2, \dots, s_m]$ be a uniform multiprocessor. Assume $a(\cdot)$ is an assignment of tasks τ to processors of π in which migration of tasks is permitted. If $s_{a(i)} \geq s_{a(j)}$ whenever $u_{F,i}/u_{C,i} > u_{F,j}/u_{C,j}$ then $value(a) = MaxSum_{\pi,\tau}$.

Proof: (By contradiction.) Assume $u_{F,i}/u_{C,i} > u_{F,j}/u_{C,j}$ and $s_{a(i)} = \ell$ and $s_{a(j)} = k$, where $s_k > s_\ell$.

Define α_i and α_j as follows.

$$\alpha_i = \min \left\{ 1, \frac{u_{i,k}}{u_{j,k}} \right\}$$

$$\alpha_j = \frac{\alpha_i u_{i,k}}{u_{j,k}}$$

By the above equation, we know that

$$\alpha_i u_{i,k} = \alpha_j u_{j,k}. \quad (4)$$

Hence, $\alpha_i T_i$ will take the same amount of slack from s_k as $\alpha_j T_j$. Let $\hat{a}(\cdot)$ be the assignment that results from moving $\alpha_i T_i$ to s_k and $\alpha_j T_j$ to s_ℓ . We wish to prove $value(a) < value(\hat{a})$. We begin by proving the following properties.

Claim 1: $\alpha_i u_{F,i} > \alpha_j u_{F,j}$

This follows directly from Lemma 1 and Equation 4. Multiplying these together proves the claim.

Claim 2: $\alpha_i u_{i,\ell} < \alpha_j u_{j,\ell}$. In particular,

$$\alpha_j u_{j,\ell} - \alpha_i u_{i,\ell} = (\alpha_i u_{F,i} - \alpha_j u_{F,j})(s_k - s_\ell). \quad (5)$$

Recall $u_{i,k} = u_{C,i} + s_k u_{F,i}$. Therefore,

$$\begin{aligned} \alpha_j u_{j,\ell} - \alpha_i u_{i,\ell} &= \\ \alpha_j (u_{C,j} + s_\ell u_{F,j}) - \alpha_i (u_{C,i} + s_\ell u_{F,i}) &= \\ + (\alpha_i u_{F,i} - \alpha_j u_{F,j})(s_k - s_\ell) &= \\ - (\alpha_i u_{F,i} - \alpha_j u_{F,j})(s_k - s_\ell) &= \\ \alpha_i (u_{C,i} + s_k u_{F,i}) - \alpha_j (u_{C,j} + s_k u_{F,j}) &= \\ + (\alpha_i u_{F,i} - \alpha_j u_{F,j})(s_k - s_\ell) &= \\ (\alpha_i u_{F,i} - \alpha_j u_{F,j})(s_k - s_\ell). & \end{aligned}$$

The last step follows from Equation 4. By assumption $s_k > s_\ell$, and, by Claim 1, $\alpha_i u_{F,i} > \alpha_k u_{F,k}$. Therefore, the final result is positive. This proves the claim.

Claim 2 tells us that moving T_j to s_ℓ will increase the total utilization of tasks assigned to s_ℓ . In the worst case $slack_\ell = 0$, and swapping T_i and T_j forces some task(s) to be removed from s_ℓ . Assume some portion, α_q of T_q is forced off of s_ℓ , where $u_{F,q}/u_{C,q} \leq u_{F,j}/u_{C,j}$. Specifically,

$$\alpha_q = \frac{\alpha_k u_{k,\ell} - \alpha_i u_{i,\ell}}{u_{q,\ell}}. \quad (6)$$

Then moving $\alpha_q T_q$ to a slower processor ensures s_ℓ will not be overfilled as a result of swapping T_i and T_j . (Note that T_q may be T_k . Also, if T_k forces multiple tasks to be removed from s_ℓ , the proof below will have to be modified slightly. We assume a single task is removed from s_ℓ for

readability.) Hence,

$$\begin{aligned}
& \sum_{h=1}^n s_{\hat{a}(h)} u_{F,h} - \sum_{h=1}^n s_{a(h)} u_{F,h} \\
& \geq \alpha_i u_{F,i} (s_k - s_\ell) + \alpha_j u_{F,j} (s_\ell - s_k) \\
& \quad - \alpha_q u_{F,q} s_\ell \\
& = (\alpha_i u_{F,i} - \alpha_j u_{F,j}) (s_k - s_\ell) \\
& \quad - \frac{\alpha_j u_{j,\ell} - \alpha_i u_{i,\ell}}{u_{q,\ell}} u_{F,q} s_\ell.
\end{aligned}$$

By Claim 2, this is equal to

$$\begin{aligned}
& (\alpha_i u_{F,i} - \alpha_j u_{F,j}) (s_k - s_\ell) \\
& \quad - \frac{(\alpha_i u_{F,i} - \alpha_j u_{F,j}) (s_k - s_\ell)}{u_{q,\ell}} u_{F,q} s_\ell \\
& = (\alpha_i u_{F,i} - \alpha_j u_{F,j}) (s_k - s_\ell) \left(1 - \frac{s_\ell u_{F,q}}{u_{q,\ell}}\right).
\end{aligned}$$

Each of these terms is positive. Therefore, $value(\hat{a}) > value(a)$. Thus, if any other order is used, $value(a) \neq MaxSum_{\pi,\tau}$. ■

Because packing without migration is a special case of packing with migration, algorithm $fracM_{\pi,\tau}$ finds an upper bound for $MP_{\pi,\tau}$. In order to find $M_{\pi,\tau}$, we loop through all tasks T_i in τ – each time calling $fracM_{\pi,\tau}(\pi, \tau - T_i)$ and calculating $(m-1)u_{C,i} + S(\pi) \cdot u_{F,i} + MaxSum_{\pi,\tau}$. After looping through all tasks, we return the maximum value.

Even replacing this upper bound for $M_{\pi,\tau}$ into Theorem 1 will improve the analysis compared to the previous test, stated in Corollary 1. In the proof of the corollary, we see that $u_{F,i}$ must be multiplied by s_1 for all $i = 1, 2, \dots, n$ if the previous test is used. By contrast, algorithm $fracM_{\pi,\tau}$ is able to multiply some of the fixed priorities by smaller values.

4.2.1 Running Time

Lines 2 through 3 of algorithm $fracM_{\pi,\tau}$ take $\Theta(n)$ time. Line 4 takes $O(n \lg n)$ time, assuming a reasonable sorting algorithm is used. Lines 5 and 6 are executed m times. Lines 7 through 17 are executed once for each task plus once for each fraction of a task that is added to τ . At most $(m-1)$ tasks are added back to τ . Therefore lines 7 through 17 are executed $\Theta(m+n)$ times, giving a total running time for $fracM_{\pi,\tau}$ of $O(n \log n + m)$. In general, we assume $n > m$, which gives the running time of $O(n \log n)$. Algorithm $fracM_{\pi,\tau}$ is executed n times, giving a total running time of $O(n^2 \log n)$.

5 Conclusion and Future Work

This paper has presented a variation of the task model that differentiates between execution that scales with CPU

speed and execution that does not. We have shown that using this model can improve a known utilization bound for scheduling on uniform multiprocessors — namely the bound for EDF scheduling with restricted migration. Unfortunately, the bound involves using an NP-complete problem. Hence, we have shown methods of finding this bound using integer linear programming. We have also shown a method to approximate this bound in $O(n^2 \log n)$ time by calculating the expression when jobs are allowed to migrate.

In the future, we hope to apply this task model to EDF using other migration strategies and also to RM scheduling. We also plan to determine the competitive ratio of $MaxSum_{\pi,\tau}$, the output of algorithm $fracM_{\pi,\tau}$, and improve the estimate if the ratio is too large.

References

- [1] Sanjoy K. Baruah and John Carpenter. Multiprocessor fixed-priority scheduling with restricted interprocessor migrations. *Journal of Embedded Computing*, 1(2), 2004.
- [2] Michael Dertouzos and Aloysius K. Mok. Multiprocessor scheduling in a hard real-time environment. *IEEE Transactions on Software Engineering*, 15(12):1497–1506, 1989.
- [3] Shelby Funk. *EDF Scheduling on Heterogeneous Multiprocessors*. PhD thesis, CS Department, UNC Chapel Hill, 2004.
- [4] Shelby Funk and Sanjoy K. Baruah. Restricting EDF migration on uniform multiprocessors. Technical Report TR03-035, CS Department, UNC Chapel Hill, 2003.
- [5] Chung Laung Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.
- [6] Aloysius K. Mok. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983. Available as Technical Report No. MIT/LCS/TR-297.
- [7] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg. Fast: Frequency-aware static timing analysis. In *RTSS '03: Proceedings of the 24th IEEE International Real-Time Systems Symposium*, pages 40–51. IEEE Computer Society, 2003.

SystemC multiprocessor RTOS model for services distribution on MPSoC platforms

DASIP 2008

November 2008

Emmanuel Huck*

Benoît Miramond*

François Verdier*

Thomas Lefebvre*

* *ETIS, CNRS, ENSEA, Univ Cergy-Pontoise*
F-95000 Cergy-Pontoise

E-mail: [firstname.name]@ensea.fr

Abstract—This paper addresses design issues of embedded software on multiprocessor platforms and for real-time applications. We focus on applications for which specific Real-Time Operating Systems (RTOS) are required. In this context the exploration of application tasks distribution onto multiprocessor architectures must follow a high-level approach for early evaluating solutions performance. In this paper we present a high-level multiprocessor RTOS model. The main contribution of this model is to provide an efficient method to abstract the dynamic and real-time mechanisms of embedded software, to customize RTOS services on each processor and to adjust the multiprocessor scheduling policy early in the design flow. In addition, this distributed model provides the application a unique Application Programming Interface. This simplifies the exploration and the mapping of software tasks onto the system. Experiments conducted with a vision application show that very precise scheduling behaviors and synchronizations between multiple processors can be modeled at a high abstraction level with little simulation overhead.

I. INTRODUCTION

With the increasing complexity of embedded applications platform architectures include more and more parallel execution units. At the same time technology is able to integrate 1 billion transistors into a single chip. Systems on Chip (SoC) now count heterogeneous processors with memory hierarchy, DMA, I/O and hardware accelerators sometimes reconfigurable communicating around a shared bus or a Network-on-Chip (NoC). In a real-time context the control of these complex systems is often devoted to one or more RTOS. These can either be deployed in software or hardware, partially or completely depending on the non-functional constraints of the global system. As a result, new design decisions must be taken earlier regarding the implementation of specific distributed RTOS.

The exploration of hardware/software systems was a research topic for many years in the last decades, yielding a unified modeling and simulation environment commonly adopted, namely the SystemC language [12]. However, efficient and accurate modeling solutions for real-time mechanisms in a multiprocessor context are still lacking.

In this paper we propose a method that tackles this design challenge by introducing a high-level RTOS model for custom system design. Working at a high level of abstraction allows the designer to jointly explore the distribution of the RTOS in terms of custom services adapted to the application and the

parallel SoC architecture. Both dynamic behavior control and embedded constraints satisfaction problems can thus be solved by a single approach early in the design flow. Contributions of this work thus consist in proposing a distributed and modular RTOS SystemC model. The model follows a Transaction Level Modeling (TLM) approach and introduces a Service Accurate level in the sense that it allows both functional and timed verifications without the need of modeling processing resources. By working at this level of abstraction, an early exploration of the architecture dimensioning is also feasible.

This work falls under the project [1] OveRSoC, which consists in developing a methodology for the design and the evaluation of specific OS for reconfigurable system-on-chip (RSoC).

The remaining of the paper is organized as follows: Section II presents the related work; Section III and IV introduces the abstract modular RTOS model; Section V describes the distribution of several instances of the model in a multiprocessor context. Experimental results are provided in section VI. Finally, we conclude and discuss future works.

II. RELATED WORK

Several approaches have been developed to deal with exploration and validation of embedded software at high-level. He et al. in [8] classifies research on RTOS modeling and simulation into three categories: System Call translation, Native OS and Virtual OS. The latter corresponds to abstract models that ease exploration. In that context, SystemC [12] was mainly selected as the underlying modeling language for the developed executable models. A first step was then to extend SystemC to embedded software modeling features which still not supports these facilities in its actual version. The works presented in [4], [7], [13] and [8] are examples of simulation environments dealing with this challenge. In these works similar solutions are provided to model the mechanisms of target RTOS such as scheduling of multithreaded applications, preemption and synchronization. Indeed, in SystemC a way to implement several scheduling policies is to assign static or dynamic priorities to the simulation processes and control which are declared ready in the SystemC simulation kernel. Ordering task executions on a sequential processing unit thus consists in maintaining only one process unblocked. The API provided by these models must be as general as possible in order to match with most of existing (commercial) operating systems

and so allow exploration. For example the SPACE project [2] encapsulates existing RTOS into a common API. As a consequence, the entire code of the operating system must be available since the RTOS and the application software are cross-compiled and the resulting binary code is executed by an ARM ISS. In [2] the method has only been applied to μ C/OS-II. In order to reach generality our approach is based on a modular API and the corresponding embedded software implementation. The API is generated according to the needed services inside each RTOS instance in the multiprocessor platform. In that sense, it is similar to [6] where authors propose to automatically extract the services required by system calls in the application code.

Most of the proposed abstract RTOS models target mono-processor architectures. Yet increasing embedded systems complexity demands parallel and heterogeneous platforms. Madsen et al. in [11] have proposed a framework that supports modeling of multiple RTOS. Application software is modeled as a set of task graph executing on multiprocessor platform. The approach is not interested in the exact functionality of tasks. As said by Hwang et al. in [10] a tradeoff is needed to determine the best abstraction level of simulation models. Exactly, the dynamic behavior of distributed real-time applications can only be reached with functional models either using ISS approaches [3] or annotated TLM models [10] as it is the case in this paper.

III. LEVEL OF ABSTRACTION

As the general methodology that consists in designing new system from top to bottom, and considering that many projects reuse already proved to work building blocks, we decide to model a system at a high level of abstraction, where the hardware is partially hidden. This will allow to simulate the behavior of the application at high speed. As we want to focus on the behavioral part of the system, generally managed by one or several RTOS and sometimes some dedicated hardware blocks like interrupts or timing, we focus our modelisation on the services provided by the platform. The model needing to be functional (running real calculus for modeled tasks or IP) and also timed, if we want take into account all unpredictable behaviors like interruptions. For those reasons, we call our level of abstraction SAT for *Service Accurate plus Time*.

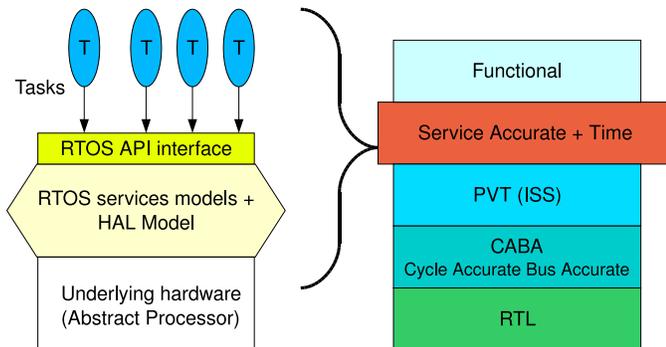


Fig. 1. SAT level of modelisation

This level of modelisation, as shown on Fig. 1, implies that the architecture is not modeled explicitly, all the application tasks are functional, annotated with approximated or measured execution timing, and all the RTOS services are explicit and timed. We could also compare our level of abstraction to the one defined by Donlin [5], introducing five level of abstraction, shown on Table I.

TABLE I
LEVELS OF ABSTRACTION, IN HARDWARE DESIGN

ALG (Algorithmic)
Parallel processes behavioral simulation, without time or architecture details. Used to functionally validate application
Communicating Processes with Time (CP+T)
Explicit processes parallelism and communications. Architecture highly abstract, with estimations on tasks and communications timings.
PV and PVT (Programmer's View + time)
Architectural components are explicit with registers details and communications explicit arbitration, contentions and their time costs. The communication timing model may account the multi-cycle cost of a subset of the transactions possible over the interconnect structure.
CA (Cycle Accurate)
It captures micro-architectural details: it typically have bit-level interfaces. The model is clocked and all timing annotations are accurate to the level of individual clock cycles and arbitration of the communication infrastructure is modeled.
RTL (Register Transfer Level)
As used in traditional design, models are implementation and architecture accurate. Intra-clock cycle timing behavior is explicitly simulated.

We could consider that our level of abstraction is equivalent to CPT, but this terminology of abstraction level is too much focus on the Hardware point of view. Here we want to focus on the RTOS service point of view, and all the dynamics it include, whatever the implementation decided for each services.

IV. DISTRIBUTED RTOS MODEL

A. RTOS SAT model

The core element of our distributed architecture model is a high-level functional model of a real time operating system written in SystemC [9]. This RTOS model acts as a *Service Accurate + Time* model of a processor in the sense that all the necessary services of an embedded RTOS are modeled as independent *service modules* with their own behavior. Moreover, each *service module* has its proper interface that provides the corresponding service to an embedded application. Figure 2 illustrates the hierarchical structure of the SystemC RTOS model. This model includes mechanisms for modeling dynamic creation of tasks, tasks preemption and interrupt handling. In addition, each service function may be annotated with timing information allowing a timed simulation of a realistic system. The RTOS model is built as a collection of *service modules* implemented in the form of hierarchical `sc_channel` to foster high level exploration of custom architectures. The main RTOS model instantiates all its modules and uses `sc_export` to provide a global API to the application code. Moreover, modules may have internal APIs and ports to establish inter-module communications.

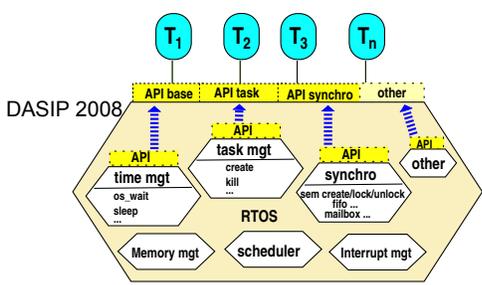


Fig. 2. Modular RTOS model system

B. Distant services sharing

Based on this model, we propose a model for distributed multiprocessor architectures exploration with the following features: the whole application is decomposed into multiple threads sharing the same addressable memory, the application is statically partitioned onto multiple processing nodes (no load balancing), each processor has its own scheduling strategy (policy, priorities etc). In addition, multiprocessor communications are modeled as shared synchronization services (with their own arbitration policy).

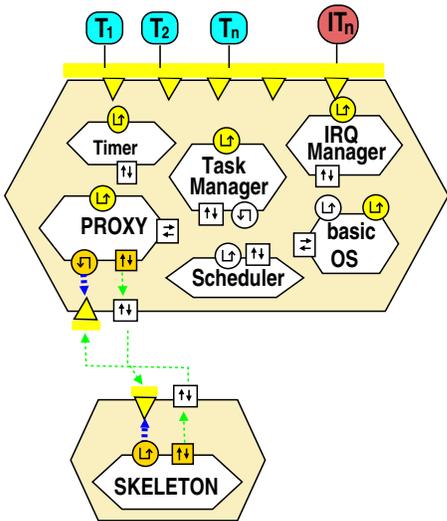


Fig. 3. Connecting proxy to skeleton

Our approach for modeling distributed OS services is inspired from the middleware CORBA philosophy by using one of its principle: The proxy/skeleton service scheme (see figure 3). A proxy service provides a local entry point to a distant service accessible through some interconnection infrastructure. This imply to have a bidirectional connection for communications between the proxy and the skeleton. And also this could change the semantic of the service call, becoming potentially blocking, because communications will potentially delay randomly the result, depending on the architecture (bus congestion etc.). This adds dedicated ports and interfaces to the RTOS modules responsible of this behavior.

C. Distant communications and invocations

All inter-processor communications are modeled using transactions with respect to TLM 1.0 methodology. A unique transport method is used for both requests and replies. All communications are currently done instantaneously but this allows a communication refinement process after. Thus the use of a communication infrastructure model supports the communication delay consumption allowing time accurate simulation, by introducing bus or network-related timings into transactional ports.

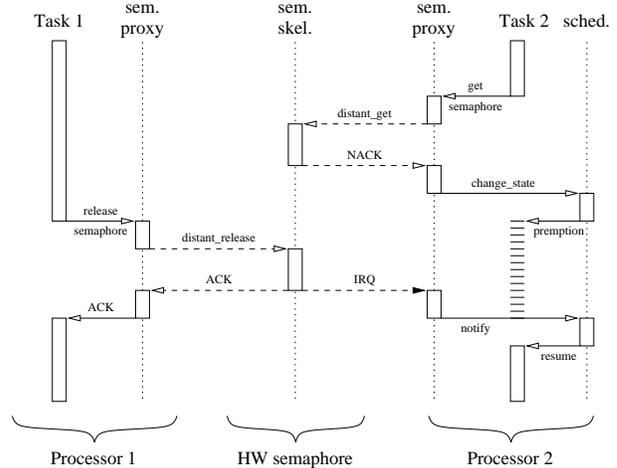


Fig. 4. Activity diagram of local/distant calls to a shared semaphore proxy/skeleton between two OS models

Figure 4 illustrates transactions between two local semaphore services (proxies) and a shared distant semaphore implementation (skeleton). Get and release semaphore invocations are done locally to the proxy which forwards transactions to the distant service. By using a simple transport method, all distant calls put the caller tasks into an active waiting state. In case of access conflicts, the shared service has its own arbitration policy. Replies are sent back to the caller at the end of service execution, thus give flow control back to the caller's scheduler.

Communications from a distant service to local proxies are done by using signals similar to interrupt requests that are managed by local proxies. Suspended tasks may then be resumed by their own schedulers depending on local policies.

In order to work correctly, a multi-RTOS system require at least a synchronization service, allowing to exchange data and perform more complex communications. For that, we develop a model of a shared hardware semaphore service (Figure 5). A task on a local RTOS ask for a distant semaphore as for a local semaphore, and the local proxy semaphore service then call the distant skeleton through its port. So the current task is set to sleep until the skeleton send the answer back, after its processing delay, and the communications delay, integrated here because our high level model does not explicit the communication channel.

When a semaphore release liberate a waiting task, the skeleton inform the right distant proxy of this action, trough

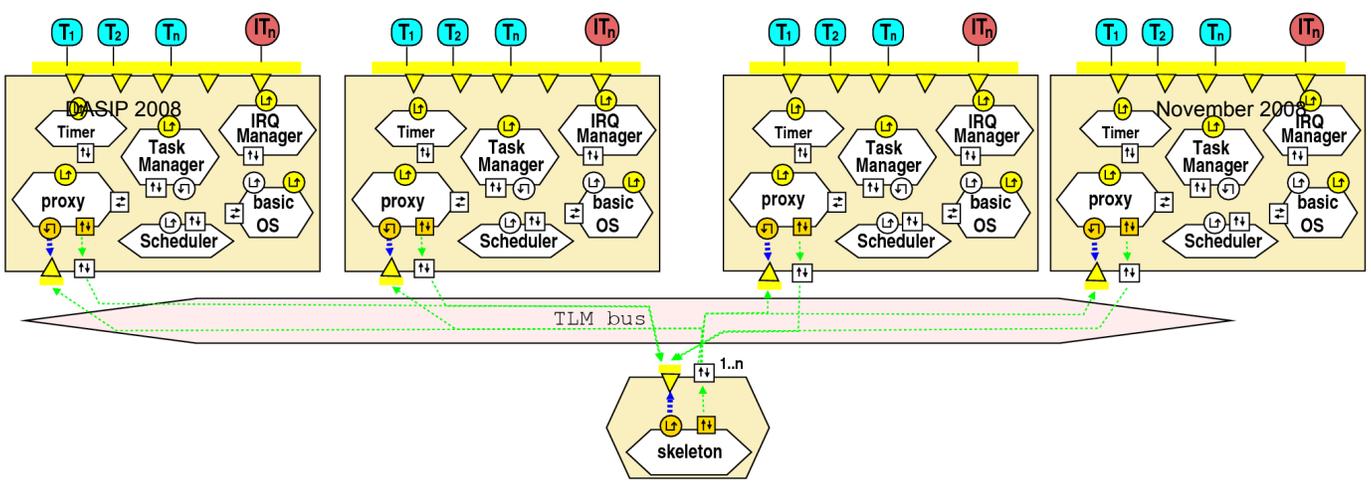


Fig. 5. Multi RTOS model with a shared hardware semaphore service

a specific distant call, triggering the specific interrupt handler own by the proxy. This one will immediately stop the current running process, set the ready state of the corresponding task waiting for the semaphore, and then relaunch the scheduler.

By this way, at a high level, we are able to model simple communication between nodes, and also as model synchronization between multiple RTOS.

V. EXPERIMENTS

We applied our framework to a realistic application in the field of image processing for robotic vision. The application is used to learn object views or landscapes and extracts local visual features from the neighborhood. We specified the application as a set of 30 communicating tasks. The full description of our application is out of the scope of this paper. However, following a biologically inspired approach, this vision architecture belongs to a larger sensorimotor loop that brings interesting dynamical properties: the degree of parallelism and the execution time varies according to input data, namely the number of interest points.

In this context we made the profiling of the entire application

TABLE II
AVERAGE APPLICATION EXECUTION TIME IN MS

On board	in simulation	error in %
29268570400 ns	28369598240 ns	3-4

on a hardware platform. We also built the profile of the μ C/OS-II services (deterministic). For the purpose of the exploration we targeted a multiprocessor architecture (MPSoC) prototyped onto an Altera Cyclone-II FPGA circuit. The timing data were measured and back-annotated into the high-level model in order to explore and evaluate the architecture dimensioning and the implementation strategies: tasks distribution, services distribution, scheduling algorithms...

To evaluate the efficiency of our modeling approach, we

performed two set of experiments. First we evaluated the model accuracy and compared the simulated execution time relatively to actual board measurements for multiple implementations. The average application times are depicted in table II. The high level simulation accuracy stands within 3-4% of board measurements.

TABLE III
SIMULATION COSTS FOLLOWING NUMBER OF RTOS

nb RTOS	0	1	2	3	4	5	6
simulation time(second)	5.5	6	7.4	8.6	9.8	11.1	12.8
overhead(%)	-8.9	0	22.5	43.2	63.2	84.2	112

Then we evaluated the simulation time of the application on top of our RTOS model in comparison to a purely functional description. With this aim we explored and simulated the deployment of the application tasks. We vary the number of processor within the architecture from 1 to 6 Nios-II. Tasks execute and communicate in the same way on board and in simulation trough a single shared memory space protected with hardware semaphores. Table III shows the scalability of our model till 6 processors, and indicates the average simulation overhead for different platform sizes. Simulations were realized on a workstation equipped with Intel DualCore at 1.66 GHz with 2GB of RAM under Linux. For monoprocessor platforms the RTOS model does not impact the simulation time since the overhead is only 9.8% more than the purely functional application description. Results indicate that the overhead is around 22% more per simulated RTOS. However simulations times are kept in seconds even for a 6 processor architecture where the number of system calls and preemptions becomes very important.

VI. CONCLUSION

We have presented in this paper the basic concepts of an abstract SystemC RTOS model for embedded software design

onto multiprocessor architectures. Results with design of a robotic vision system demonstrate the accuracy of the simulations and the scalability to complex multiprocessor platforms. As a result the framework allows designers to early evaluate the distribution of both application tasks and RTOS services. The main contribution is to provide, thanks to a modular structure, a high-level distributed RTOS model in SystemC augmented with a multiprocessor synchronization mechanism. Future works concern hardware/software co-simulation issues. The distributed model is actually used in an industrial project as the synchronization and control layer of a high-performance embedded computer.

REFERENCES

- [1] I. Benkermi *et al.*, "System-Level Modelling for Reconfigurable SoCs," in *Design of Circuits and Integrated Systems (DCIS'05)*, Nov. 2005.
- [2] J. Chevalier *et al.*, "SPACE: a hardware/software SystemC modeling platform including an RTOS," in *Forum on Design Languages*, 2003.
- [3] M.-K. Chung *et al.*, "System-level hw/sw co-simulation framework for multiprocessor and multithread SoC," in *Int. symposium on VLSI Technology Systems and Applications*, 2005, pp. 177–180.
- [4] D. Desmet, D. Verkest, and H. D. Man, "Operating System based software generation for Systems-on-Chip," in *Conference on Design Automation*, 2000, pp. 396–401.
- [5] A. Donlin, "Transaction level modeling: flows and use models," in *IEEE/ACM/FIP international conference on Hardware/software code-sign and system synthesis (CODES+ISSS'04)*, 2004, pp. 75–80.
- [6] L. Gauthier, S. Yoo, and A. Jerraya, "Automatic Targeting of Application Specific Operating Systems and Embedded Systems Software," 2001, pp. 679–685.
- [7] P. Hastono, S. Klaus, and S. Huss, "Real-Time Operating System Services for Realistic SystemC Simulation Models of Embedded Systems," in *Forum on specification and Design Languages*, Sep. 2004.
- [8] Z. He, A. Mok, and C. Peng, "Timed RTOS Modeling for Embedded System Design," in *IEEE Real Time on Embedded Technology and Applications Symposium*, 2005, pp. 448–457.
- [9] E. Huck, B. Miramond, and F. Verdier, "A Modular SystemC RTOS Model for Embedded Services Explorations," in *Design and Architectures for Signal and Image Processing, Grenoble*, 2007.
- [10] Y. Hwang, S. Abdi, and D. Gajski, "Cycle-approximate retargetable performance estimation at the transaction level," march 2008.
- [11] J. Madsen, K. Virk, and M. Gonzalez, "Abstract RTOS Modeling for Multiprocessor System-on-Chip," in *Symposium on System-on-Chip*, Nov. 2003, pp. 147–150.
- [12] OSCI, "IEEE 1666TM Standard SystemC Language," available at : <http://www.systemc.org>.
- [13] H. Posadas *et al.*, "RTOS modeling in SystemC for real-time embedded SW simulation: A POSIX model," *Design Automation for Embedded Systems*, vol. 10, no. 4, pp. 209–227, Dec. 2005.

A Novel Video Packet Loss Concealment Algorithm & Real Time Implementation

Abraham Suissa

Pierre and Marie Curie University – Paris 6, EA 2385 – BC 252, 4 Place Jussieu, 75252 PARIS CEDEX 05, France

Abraham.Suissa@gmail.com

Jennifer Mellor

New Platforms and Technologies, ICBU, Logitech Inc., Fremont, California, USA

Jennifer_Mellor@Logitech.com

Frantz Lohier

New Platforms and Technologies, ICBU, Logitech Inc., Fremont, California, USA

Frantz_Lohier@Logitech.com

Patrick Garda

Pierre and Marie Curie University – Paris 6, EA 2385 – BC 252, 4 Place Jussieu, 75252 PARIS CEDEX 05, France

Patrick.Garda@upmc.fr

Abstract— Streaming video data over a digital Radio Frequency (RF) link operating in the ISM band (e.g., 2.4GHz/5GHz) is known to be a challenging task that has recently captured considerable industry attention. Various sources of air interference cause packets to be dropped despite retries and forward error correction (FEC) schemes found either at the transport or video encoding layer. However, there is an industrial interest for a concealment technique suitable for wireless cameras, featuring very low processing power. This article presents therefore an alternative to the conventional wisdom: a novel video packet loss concealment scheme that operates after the decoding/ decompression of a truncated video bitstream. The approach is therefore codec-independent and does not incur the overhead associated with typical FEC techniques. We demonstrate recovery up to 20dB using a real-time implementation of our proposed approach.

I. INTRODUCTION

In order to ensure proper Quality of Service (QoS) when streaming audio/video data over an error prone transmission link, such as an RF channel challenged by interference sources including the hopping nature of certain technologies (e.g., Bluetooth), multi-path fading challenges, and competing air-time access of devices, a combination of techniques is typically necessary both at the transport layer and data encoding/compression layer. At the transport layer, we can cite co-existence schemes that have been promoted around the Bluetooth technology or convolutional data coding techniques at the modulation layer (e.g., viterbi). For the case of video encoders such as H.264 or JPEG2000 (e.g. of Chapter 11), standards typically discuss the insertion of resynchronization markers in the compression loop as well as a structural bit-stream arrangement to allow the decoder to continue the decoding process despite erasures encountered in the compressed image, which essentially prevents frame skipping.

However, there is an industrial interest for a concealment technique suitable for wireless cameras [1], sharing the electromagnetic spectrum with some wireless networks. The very low complexity encoder processing power of these devices limits the video coding to very low complexity encoder. The Wyner-Ziv video coding algorithm [2] is attractive for its low

complexity, but it is not yet standardized. Thus we relied on the Motion JPEG (MJPEG), which also features low complexity encoder. Then, the concealment task is performed by the decoding receiver. This receiver is actually a PC or a server (Figure 1); it has thus far more computing power than the wireless camera.

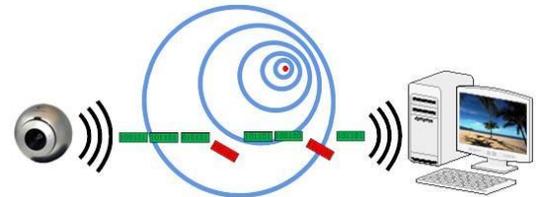


Figure 1 : Video PLC difficulty overview

Therefore, in this paper, we pursue an approach where a video packet loss concealment (VPLC) technique is executed after the truncated stream is decompressed. The technique is independent of the underlying transport or video encoding technology, the main assumption being that the video decoder is capable of identifying when an erasure occurs and can continue the video decoding process of an image as opposed to dropping the frame. Once the partially decoded image is reconstructed, our algorithm, by using spatial and temporal information of the stream, is able to recover the non-truncated frame with minimal distortion.

This article is structured as follows. In section 2, we describe our unique approach for VPLC, which leverages several known techniques operating in the spatial and temporal domain. At the algorithm's heart are the following techniques: motion estimation of deleted blocks, block copy and spatial interpolation of missing data, a deblocking filter and a sophisticated decision tree which selects the execution of each of the above building blocks. In section 3, we discuss the key aspects of a sample implementation based on the motion JPEG (MJPEG) industry standard. We review the provision that such a standard offers for a decoder to resynchronize on a truncated stream and propose extensions to the baseline mechanism to further optimize the performance of our VPLC algorithm. Finally, in section 4, we present the performance gain of our

approach based on our experimental MJPEG-over-wireless streaming demonstrator. The benefits of our approach are discussed in terms of video quality improvement, processor execution time impact and recovered frame rate.

II. VIDEO PACKET LOSS CONCEALMENT SCHEME

In this section, we present our new approach to video packet loss concealment. The VPLC scheme uses motion vector estimation after decoding and a decision tree employing several methods of error concealment, including block copy and spatial interpolation, to achieve recovery of lost video data with minimal visual artifacts.

In order to characterize the size and frequency of truncations to be concealed, we consider a straightforward compression and packetization scheme for streaming wireless video data. For example, with a compressed JPEG image transmitted over an 802.11 connection, a single lost transport packet could realistically result in up to 4 consecutive 8x8 block rows of missing uncompressed video data. Depending on the frequency of errors in transmission, each decoded video frame may incur several lost packets of several rows each. This nature of video data loss is inherently challenging for recovery mechanisms and serves to demonstrate the applicability of our VPLC algorithm for a variety of codecs and/or packetization schemes for low to moderate probabilities of transport packet loss.

A. Error Concealment Using Motion Vector Estimation

1) *Definition:* For our proposed VPLC algorithm, there are several core concepts. First, we maintain a reference image for motion detection based on a valid decoded video frame (e.g., the previous frame). Then, we calculate motion vectors for the regions surrounding lost video data in the current frame based on the reference image. A block matching technique [7] is used for this purpose. We interpolate the values to estimate motion vectors for the missing data itself, and finally we use that motion vector information in our decision tree to apply the most appropriate techniques for replacing the missing blocks.

One key point is that motion vector estimation is not only used to gauge whether there is motion in the region but to determine where in the reference image the now-missing data would have been. At 15-30 frames per second (fps), there is generally little enough motion between frames that the likelihood of locating the missing video data in the reference image is high.

To obtain motion vectors of the missing blocks we perform block matching on the closest neighbor blocks, calculating motion vectors for the nearest valid blocks on the top and bottom of the missing blocks. To estimate the motion vectors for the missing blocks, we use an interpolation between these two motion vectors as shown in Figure 2.

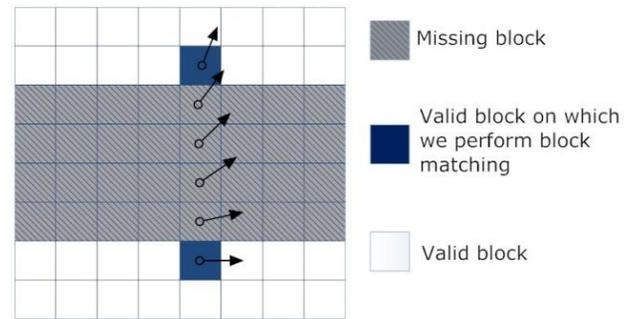


Figure 2 : Motion vector interpolation

2) *Block Matching:* Block matching techniques [7] match blocks from the current frame with blocks from a reference frame. In the proposed mechanism the block matching process is done on the luminance plane; processing in the chrominance plane is not required here because the luminance plane stores in average 80% of the signal energy.

The displacement in block location from the current frame to the location in the reference frame is the motion vector. Block matching techniques can be divided into three main components:

- Block determination
- Search method
- Matching criteria

The **first** component, block determination, specifies the position and size of blocks in the current frame, the start location of the search in the reference frame, and the scale of the blocks. We focus on fixed size, disjoint blocks spanning the frame, with initial start location at the corresponding location of the block in the reference frame.

The search method is the **second** component, specifying where to look for candidate blocks in the reference frame. A fully exhaustive search consists of searching every possible candidate block in the reference frame. This search is computationally expensive and another search method has been proposed to reduce the error in the matching (section II.B.2. *Search Method*).

The **third** component is the matching criteria. The matching criteria are a similarity metric to determine the best match among the candidate blocks.

SAD The sum of the absolute values of the differences in the two blocks:

$$M(p, q) = \sum_{i=1}^n \sum_{j=1}^n |I_c(i, j) - I_r(i + p, j + q)|$$

MAD The mean of the absolute values of the differences in the two blocks:

$$M(p, q) = \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n |I_c(i, j) - I_r(i + p, j + q)|$$

MSD The mean of the square of the differences in the two blocks:

$$M(p, q) = \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n (I_c(i, j) - I_r(i + p, j + q))^2$$

MPC The sum of the non-matching pixels in the two blocks; a match is determined by the absolute value of the difference being less than a threshold, t_{MPC} .

$$M(p, q) = \sum_{i=1}^n \sum_{j=1}^n D(I_c(i, j), I_r(i + p, j + q))$$

$$D(a, b) \begin{cases} = 0 & \text{if } |a - b| \leq t_{MPC} \\ = 1 & \text{otherwise} \end{cases}$$

SSD The square of the differences in the two blocks:

$$M(p, q) = \sum_{i=1}^n \sum_{j=1}^n (I_c(i, j) - I_r(i + p, j + q))^2$$

SAD and MAD only differ by a constant in the case of fixed size blocks and can be used interchangeably in our comparison. In practice, SAD is faster due to the removal of the divide operation. While MAD incorporates large differences, MSD penalizes blocks with one or more large differences. MPC on the other hand equally weights any difference above a threshold. In our implementation the MSD criteria is optimized, making it faster than the other criterions and the best choice for us to use.

3) *Error Concealment Procedure*: The concealment procedure is as follows. If there is no motion in the missing block region, we conceal the missing block using *Block Copy Concealment*. This common method exploits the statistical assumption that a lost block in the previous frame is unlikely to be lost in the next frame. Replacing a lost block with the similarly positioned block in the previous frame is the easiest and fastest concealment method [4] [5] and works well in still regions of the image, such as backgrounds and inanimate objects.

If there is motion in the region, however, we estimate the location of the missing block in the reference image and use that for copying to conceal the missing block. This works well with animated objects or persons or in situations where the camera itself is in motion.

The final path is when there is motion but the results of motion vector estimation cannot be used for concealment purposes. There are three cases in which this could occur: when the block on which we perform block matching does not exist in the reference frame, when the displacement of the missing block is larger than the search area, or when the interpolated motion vector indicates a block outside the boundary of the reference image. In any of these cases, we use *Spatial Concealment* [4] [5] [6], a method which conceals each pixel of the missing block with a weighted interpolation of the surrounding four pixels of the adjacent undamaged blocks.

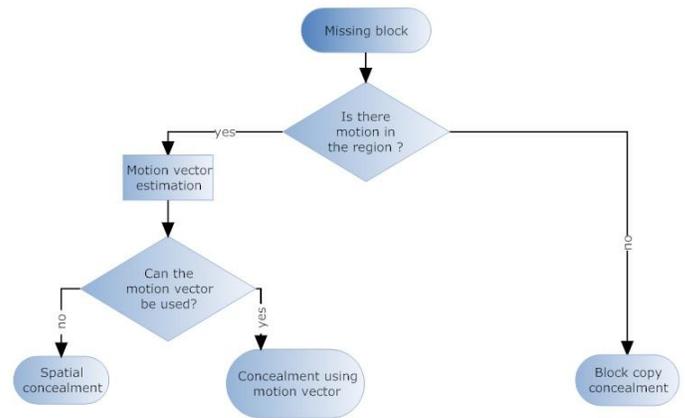


Figure 3 : Block diagram of the algorithm decision tree

Typically, we use the previous frame as the reference image for block matching; we refer to this as *Simple Concealment*. For better precision when performance constraints allow, we use both the previous and next frames, which we call *Double Concealment*, as shown in Figure 4.

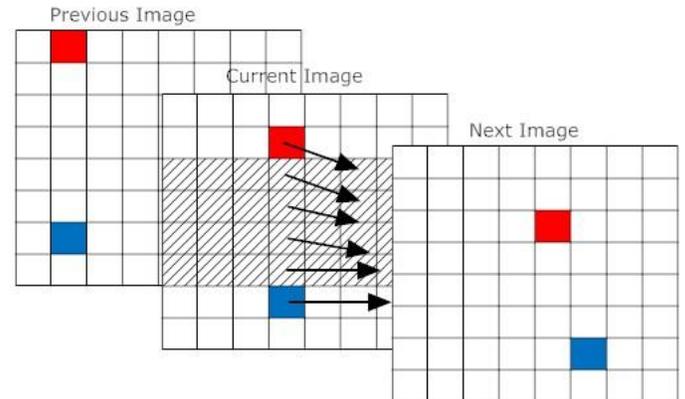


Figure 4: Motion vector estimation with two reference images

4) *Post Processing*: In the process of concealing missing blocks with those from other images we often introduce an undesired blocking artifact, not unlike that seen with high JPEG compression ratios. By applying a deblocking filter [9] to the concealed section, we are able to significantly reduce this artifact and improve the image quality of concealed regions.



Figure 5: Blocking Artifact



Figure 6: After Deblocking Filter

B. Multi Resolution and Varying Block Size Concealment

1) *Uniform Regions:* In uniform color regions, the process of estimating and compensating blocks' motion is challenged and can yield incorrect results. Issues can appear in situations where objects of interest have a uniform color. Blocks do not appear to be moving because all the blocks around them have the same color. We propose two methods and an improvement of the search method used for block matching in order to deal with this issue.

2) *Search Method:* We propose an additional search algorithm that reduces the Uniform Region Problem. In a classic search method like the Full Search method (Figure 7), the reference block is indicated in blue and the search window is carried out around this block. In full search, we start the scan with the top left corner and finish with the bottom right corner following the arrow in Figure 7. The color of the reference block is light blue, and block matching is going to return all the light blue blocks as matched. As the figure suggests, there are multiple source candidates.

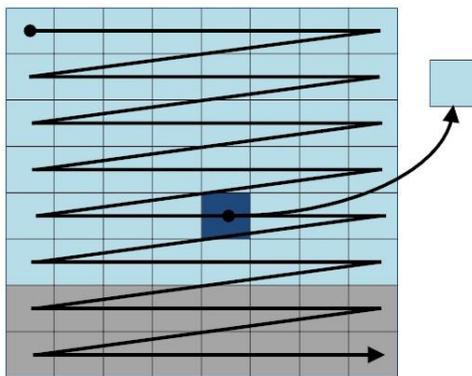


Figure 7: Full search method

In the Full search method the corresponding block is the first or last minimum, but as we can see in Figure 6 these are the farthest blocks from the reference block. We can assume that the movement of a block between two images is not large. We are more likely to find a block near its original position. This suggests it would make sense to start the search in the middle of the search window, in the nearest position to the reference block.

The Spiral search method used in some MPEG-2 implementations is exactly what we need in this case. In the Spiral search method (Figure 8) we start with the nearest block and we finish with the farthest block. With the spiral search method in a uniform color region the returned corresponding block is always the closest block to the reference block. In this work we favored the image quality of the block searching algorithm over its complexity.

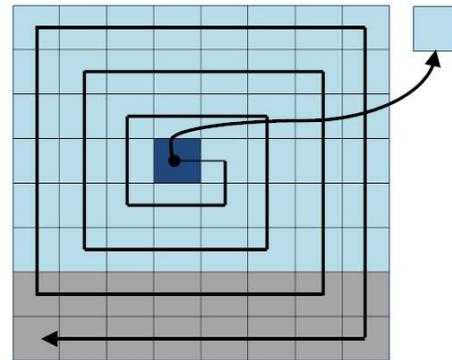


Figure 8: spiral search method

3) *Multi Resolution Error Concealment:* In order to improve the visual concealment and further mitigate the issues with uniform regions, it helps to perform block matching with high amplitude gradient blocks. Our suggested approach is to use the same image at a lower resolution. In this case the same block size as before contains more information, as shown in Figure 9. The lower the resolution is, the larger the search area in an 8 by 8 block and the more likely we are to get blocks with high amplitude gradient pixels.



Figure 9: Multi Resolution Error Concealment

4) *Varying Block Size Error Concealment:* Another method to deal with motion vector estimation inaccuracies in uniform regions is to use bigger blocks, i.e., 16x16 or 32x32, to perform block matching. This method gives the highest results in terms of image quality, but it increases execution time considerably and violates the real-time constraint of our implementation.

III. REAL TIME IMPLEMENTATION USING M-JPEG

In order to validate the VPLC algorithm described in the previous section, we propose a real-time PC implementation of our approach using the low complexity, mature and widely

accepted Motion-JPEG (M-JPEG) industry protocol. Leveraging the JPEG still image standard, this format offers a basic error detection scheme for corrupted bit-streams based on the concept of restart markers. We propose an extension to the restart marker concept to better support our VPLC approach while introducing very limited overhead in terms of transport size. Finally, we discuss how the decoding process integrates the video packet loss concealment algorithm and the experimental results we saw.

A. Error Detection and Recovery With JPEG

1) *JPEG Restart Markers and a Proposed Extension*: The JPEG standard defines a mechanism to partition a compressed stream into independently decodable segments called *restart intervals* [3]. A restart interval is composed of a fixed number of MCUs (Minimum Coded Units) identified in JPEG headers by a DRI (Define Restart Interval) marker. The restart marker allows a decoder to identify a bit-stream segment that can be processed independently of what was decoded up to that point. The error recovery procedure is not explicitly defined by the JPEG standard, but the restart marker combined with the DRI provides the necessary information for a decoder to handle errors in the stream. Below is an example of 16-bit restart markers segmenting compressed data:

```
FFD0 data FFD1 data FFD2 ... FFD7 data
FFD0
```

As shown in the example, the restart marker uses a modulo-8 count of the restart interval. The counter resets to 0 for each new scan, and it can wrap around after or within a scan. Thus, depending on the truncation error rate, it is quite possible to get a situation where the decoder is unable to uniquely identify exactly where, within an image extracted from a video stream, the (x, y) coordinate of an erasure is located. Alternatively, to avoid the issues caused by the counter wrapping around, we could impose the rule that only 8 legacy restart markers are used in each image. For a color YUV 422 VGA image (640×480 resolution), which corresponds to 4800 8×8 blocks or 2400 MCUs, each restart marker would cover 7.5×8 lines of the image. This translates to a very large data loss, which would challenge any error concealment algorithm.

Instead, we suggest an extension of the JPEG Restart Marker syntax by placing an additional byte next to FFDx, which we call the *resync marker*. This *resync marker* wraps around only after reaching 0xFE. We do not use 0xFF for the extra byte because of JPEG syntax restrictions. The sequence becomes:

```
FFD000 data FFD100 data FFD200 data ...
FFD700 data FFD001 data FFD101 ...
```

With this improved scheme, the maximum number of uniquely identifiable restart markers becomes 2039. For a QVGA image we can insert a resync marker every MCU, and in VGA every 2 MCUs, which greatly improves the decoder's ability to detect loss and resynchronize on a truncated bit-stream.

2) *Error Detection and Recovery with Resync Markers*: While parsing a JPEG bit-stream featuring the proposed resync markers, we first determine if there are any missing resync markers within the image and then calculate the size and location of the erasure. Once all the erasures of a frame have been identified, we reconstruct the truncated bit-stream by adding "blank" compressed data at the location of erasures and converting our resync syntax to that of the legacy restart markers. After this operation, the bit-stream is ready to be decoded by any legacy JPEG decoder, with truncated areas appearing as black regions in the decoded image. The frame is then ready to be processed by the video packet loss concealment algorithm presented in section 2.

B. Experimental Results

To validate our VPLC approach, we first focused on a pure PC software implementation. Various test sequences were collected and, for initial validation, erasures were artificially inserted before invoking our optimized software VPLC implementation. Then, we expanded the validation and benchmarking of our approach by using an integrated video over wireless streaming demonstrator challenged by interference in the ISM band. The test clips and test environment were selected to cover a number of usage scenarios and data content ranging from high global motion to more static scenes, with various levels of detail in the captured videos.

The main metrics collected for all these experiments were the CPU load of the PLC algorithm on a given PC configuration, the recovered frame rate, and the image quality improvement. To measure image quality, in addition to using the typical PSNR (Peak Signal-to-Noise Ratio), we measured the VQM (Video Quality Metric) scores [8]. This metric collects and aggregates various statistical attributes of images (percentage of blurriness/blockiness, percentage of unnatural motion, etc.) that are important to the human perception of image quality to report an overall Mean Opinion Score (MOS). Considering the video PLC problem space, we find this metric significantly more relevant than PSNR alone. The error rate for simulating random erasures was set at 14%, with resync markers introduced every MCU for QVGA images. The PC configuration used was a desktop PC running Windows XP Pro on an Intel Pentium 4 3.2GHz with 1GB of SDRAM.

Table 1 shows a comparison between common VPLC methods (*Block Copy Concealment* [4] [5], *Spatial Concealment* [4] [5] [6]) and our methods (*Simple Concealment*, *Double Concealment*, *Multi Resolution*, *Varying Block Size*) for VPLC. The image quality is improved as our concealment algorithm gets refined and features more CPU-demanding algorithmic blocks. It is intuitive that the higher the image quality, the more CPU time is needed to conceal a frame. As the table suggests, varying the block size during VPLC does not allow our implementation to run in real-time as the execution time exceeds 33ms and was thus excluded from the implementation.

Method	VQM	PSNR (dB)	Time (ms)
Truncated clip	74,8191	16,1353	0
Block copy Concealment	36,2835	29,2143	0,0487013
Spatial Concealment	48,7362	27,7853	0,0974026
Simple concealment	28,4018	31,8446	10,1461039
Double Concealment	24,8869	32,4958	19,6428571
Multi resolution Double	25,4259	35,7346	24,4480519
Varying block size 16x16	16,2172	34,6087	144,074675
Varying block size 32x32	14,0224	35,2812	582,282468

Table 1: Image Quality vs. Execution Time (the smaller the VQM score, the higher the image quality)

As the table 2 suggests, in an office environment with significant interference in the ISM band, we have observed that using MJPEG compression, we can easily face close to half the received video frames being corrupted, even when the distance between the sender and receiver is limited (less than a meter). For example, table 2 shows that without our VPLC algorithm, 14fps would be achieved at QVGA resolution. With our VPLC solution, even by dropping frames exceeding 30% erasures (1.4fps on average in this test) we are able to conceal up to ~12 frames while maintaining a high PSNR (35.7dB using the multi resolution scheme) for most scenes, including those featuring a high degree of motion.

Perturbations	Reconstructed FPS	FPS with Errors	FPS with more than 30% Errors	FPS displayed after PLC	FPS displayed before PLC
Office environment	28.76	1.47	0.4	28.36	27.29
Heavy interference + Office environment	25.85	11.72	1.4	24.45	14.12

Table 2: Frame rate improvement using our PLC algorithm and a hardware prototype

IV. CONCLUSION

In this paper, we presented a new video packet loss concealment (VPLC) technique which is suitable for wireless transmission but independent of the encoding and transport mechanisms used. Video PLC is traditionally handled at the encoder and/or transport level. Our approach is unique in that video PLC is addressed after the decoding process. More specifically, we performed the motion estimation after the image decoding. We used Motion-JPEG and a proposed improvement to the JPEG syntax for error detection. We validated our VPLC algorithm in terms of real-time performance on a PC platform as well as from the standpoint of image quality and frame rate improvement. Frame rate was improved by 70% to 80% on a wireless hardware streaming

demonstrator while video quality showed a 100% improvement over a wide range of reference sequences.

This VPLC algorithm is therefore well-suited to the wireless cameras, as it was performed in real-time by the decoding PC, and requires minimal modifications of the MJPEG codec. Observe that this algorithm could also fit the constraints of forthcoming wireless cameras operated on batteries or with green energy sources [1] and forthcoming low complexity encoders [2].

Our work could be further extended to characterize how our approach compares or complements video PLC tools offered by recent encoding techniques such as H.264 (data partitioning, reversible Huffman encoding, flexible macro block ordering) or the various tools offered by JPEG2000 and the recent Chapter 11 focusing on error resiliency.

REFERENCES

- [1] IF Akyildiz, T Melodia, KR Chowdhury, "A survey on wireless multimedia sensor networks", *Computer Networks, Elsevier*, 2007
- [2] C. Brites, J. Ascenso, F. Pereira, "Improving Transform Domain Wyner-Ziv Video Coding Performance", *IEEE ICASSP*, 2006
- [3] WB Pennebaker, JL Mitchell, *JPEG Still Image Data Compression Standard*, ISBN 0442012721, Van Nostrand Reinhold, New York, 1993.
- [4] C Dvrolis, D Tull, P Ramanathan, "Hybrid spatial/temporal loss concealment for packet video", *Proc. 9th International Packet Video Workshop*.
- [5] A Raman, M Babu, "A low complexity error concealment scheme for MPEG-4 coded video sequences", *Tenth Annual Symposium on Multimedia Communications*, IEEE Bangalore Section, 2001.
- [6] Trista Pei-chun Chen and Tsuhan Chen, "Second-generation error concealment for video transport over error-prone channels", *ETRI Journal*, Carnegie Mellon University, Pittsburgh, 2005.
- [7] Nicole S. Love and Chandrika Kamath, "An Empirical Study of Block Matching Techniques for the Detection of Moving Objects", Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore.
- [8] ITS, Video Quality Research, ANSI T1.801.03-2003 <http://www.its.bldrdoc.gov/n3/video/>
- [9] International Standard ISO/IEC 14496-2. Information Technology. Coding of Audio-Visual Objects - Part 2: Visual.



Figure 10: Concealment examples – Erroneous image



Figure 11: Concealment examples – Block copy concealment



Figure 14: Concealment examples – Double concealment



Figure 12: Concealment examples – Spatial concealment



Figure 15: Concealment examples – Multi resolution concealment



Figure 13: Concealment examples – Simple concealment



Figure 16: Concealment examples – Varying block size concealment (32x32)

MOREA : A Memory-Oriented Reconfigurable Embedded Architecture

Erwan Gr ace^{†‡}, Daniel Chillet[†], Rapha el David[‡], and Olivier Sentieys[†]

[†]University of Rennes 1, IRISA/INRIA

6, rue de Krampont BP 80518

F-22305 Lannion

E-mail : *firstname.name@irisa.fr*

[‡]CEA, List Embedded Computer Laboratory

Mailbox 94, F-91191 Gif-Sur-Yvette

E-mail: *firstname.name@cea.fr*

Abstract—Memory unit design becomes the main challenge in embedded integrated circuit. Indeed, it has to minimize power consumption while meeting high bandwidth requirements as well as supporting versatile data access patterns. To deal with such constraints, the MOREA reconfigurable architecture is proposed. In addition to a flexible computing unit, MOREA supports dynamic reconfiguration of storage resources. In this paper, we focus on the address generation subsystem which controls data streams between memories and computing unit. A dedicated programmable unit has been designed to improve regular address patterns generation while supporting complex address generation schemes during control dominated threads. The address generation unit implementation performances are discussed and its silicon efficiency is evaluated on representative computation kernels.

I. INTRODUCTION

Today’s embedded integrated circuit (IC) designers must deal with low power consumption, high performances, low cost and short time-to-market (TTM) constraints. In addition, embedded applications have dynamic behavior, i.e they exhibit versatile computation and memory access patterns.

For example, Fig. 1 shows a block diagram of a 3G telecommunication system. It handles 16-bit audio samples and 8-bit video pixels both. These data flow through several stages before being physically transmitted. Firstly, source coding reduces information quantity by processing a wide set of word-level data. In such processings, memory accesses have strong temporal locality. Secondly, channel coding increases signal robustness against noise. Contrary to source coding, it computes bitwise data and exhibits good spatial locality. Hence, bit-level operators and datapaths are needed to meet performance constraints. Thirdly, access technique multiplexes the users and modulation adapts the signal to the transmission support. These stages handle respectively wordwise and bitwise data in a streaming fashion. Finally, implemented algorithms evolve over the years, e.g video codecs upgrading to MPEG-4 standard and later. Thus, embedded applications exhibit versatile computation and memory access patterns.

As a potential hardware support, digital signal processor (DSP) offers good flexibility because of Harvard/Von Neumann execution model. However, its high clock frequency

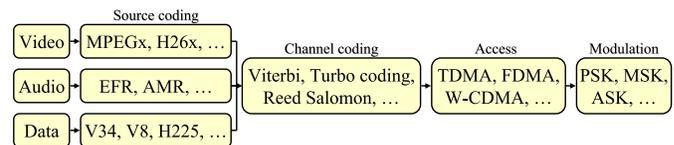


Fig. 1. Block diagram of a 3G transmission system.

TABLE I
WCDMA RECEIVER IMPLEMENTATION RESULTS FOR DART
RECONFIGURABLE PROCESSOR [2].

Architecture	Energy efficiency
DART	39MOPS/mW
DSP TMS320C64x	2MOPS/mW
FPGA Xilinx Virtex 200E	5MOPS/mW

as well as its load/store execution model incur a significant energy consumption overhead. On the contrary, fixed-functional hardwired ASIC design provides a near-optimal consumption/performance trade-off for a single algorithm. Therefore, it is a costly solution in a complete application context. Finally, a promising concept has emerged as the coarse-grain reconfigurable architecture (CGRA) [1]. CGRA allows to alter dynamically, i.e at runtime, datapaths according to calculation patterns but, unlike FPGA, it targets at a narrow application domain. As a result, CGRA is more power-efficient than typical hardware solutions. For example, in [2], the energy efficiency of the DART architecture has been discussed and compared to programmable and fine-grain reconfigurable architectures (Table I).

DART is a coarse-grain reconfigurable architecture optimized for mobile telecommunication applications. It is organized in a hierarchical way to ease the distribution of control, storage and processing resources (Fig. 2). The processing primitives in DART are Reconfigurable DataPath (RDP) integrating functional units and memories interconnected with a multi-bus network. Each memory is associated to a dedicated address generator unit (AGU) built on a RISC architecture. The memory hierarchy in DART is fixed: first, data are stored in a shared cluster memory and then, they are transferred to

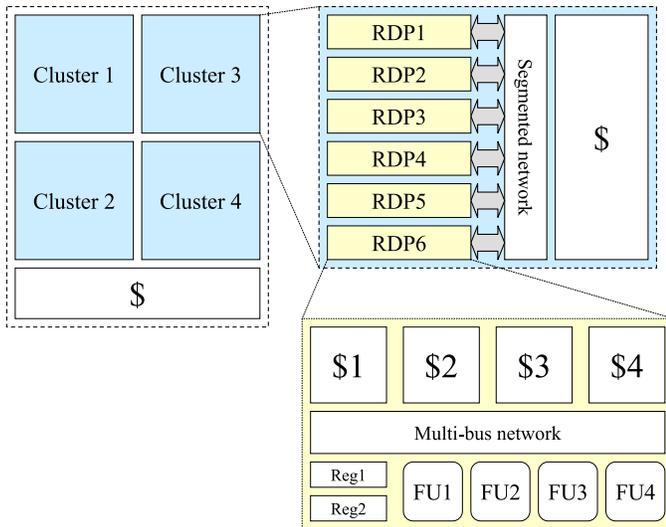


Fig. 2. System-level architecture of DART processor [2].

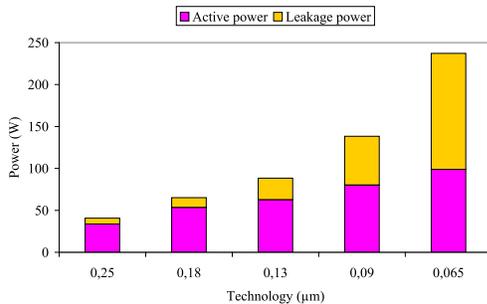


Fig. 3. Power consumption in deep submicron CMOS chips (*IC Insights Inc., Technology Trends, 2003*).

RDP local memories so as to be handled by functional units.

So far, as in DART, reconfigurability has been especially applied in the literature [1] to the processing unit. However, storage resources are projected to cover about 90% of the silicon area in 2010 [3]. Moreover, leakage current increases as transistor geometry scale shrinks (Fig. 3). In addition, static power is approximately proportional to integration density. As a consequence, IC power consumption is going to be dominated by on-chip memory energy dissipation.

Furthermore, the memory hierarchy is usually sized considering the most demanding task. Consequently, important amount of time and energy are wasted within the memory subsystem during low activity periods and memory accesses cannot be optimized with respect to application constraints. So, in this context, we propose a memory-oriented CGRA. It allows to dynamically reconfigure the memory unit according to the processing needs.

In this paper, we are focusing on the address generation subsystem which manages data flows between memories and computation resources. The remainder of this paper is organized as follows. The next section briefly depicts existing reconfigurable solutions for memory subsystems. In the section III, MOREA architecture is described. In section IV, the address generation unit is detailed and section V evaluates its

performance. Section VI discusses the AGU implementation results. Section VII concludes this paper and discusses future work.

II. RECONFIGURABLE MEMORY HIERARCHY

So far, except for memory blocks embedded in FPGA [4], hardware reconfiguration has been mainly applied to cache memories. For example, multi-banking is exploited in set-associative caches so as to share ways (or banks) with computing unit [5], i.e ways are configured as LUT, or with upper level(s) of the hierarchy [6]. In [7]–[11], authors propose to turn off banks or cache lines so as to reduce power for little performance degradation. Besides, reconfiguration control mechanism implements an online management algorithm, i.e specific hardware detects changes in application's cache usability and reacts with selecting a new configuration [6], [8], [10]. However, in n -way set-associative caches, retrieving one word requires to check n banks. Therefore, these self-adaptive memories provide low bandwidth and consume a lot of energy per access.

On the contrary, a compiler-controlled memory is described in [12]. Smart Memories chip is a modular architecture which contains 64 tiles. A tile integrates a multi-banked memory unit which communicates with a processing unit through a crossbar. The crossbar is dynamically routed and requests contain a tag indicating which bank to access. As a result, dynamic power can be saved when accessing a subset of the banks. The memory unit is made of 16 *mats* and each of them includes a SRAM with peripheral logic. The logic implements a reconfigurable AGU which allows to use efficiently a *mat* as a FIFO. However, for application with versatile address patterns, address calculation is supported by the PU. Therefore, a Smart Memories tile provides low performance when running embedded applications.

III. MEMORY SYSTEM OVERVIEW

The proposed Memory-Oriented Reconfigurable Embedded Architecture (MOREA) aims at minimizing power and delay by taking advantage of different parallelism granularities (process, task, operation, word-level). In computing systems, applications are composed of processes. Because of implementation performance considerations, a process computes data stored in a private address space. Then, result data are sent to one or more processes thanks to inter-process communications which are managed by synchronization mechanisms, e.g message queues or pipes. So, at the highest level, storage resources in MOREA are distributed over an array of tiles (Fig. 4). A tile contains a single-port global memory which stores data handled by a process. In addition, a NoC-based interconnect implements inter-process communications.

figure

Then, a process is structured as a graph of threads. Threads access to a common address space which results in massively parallel communications. To support them, a tile is made of a crossbar driving data among multiple clusters and the global memory (Fig. 4), allowing threads to share and concurrently

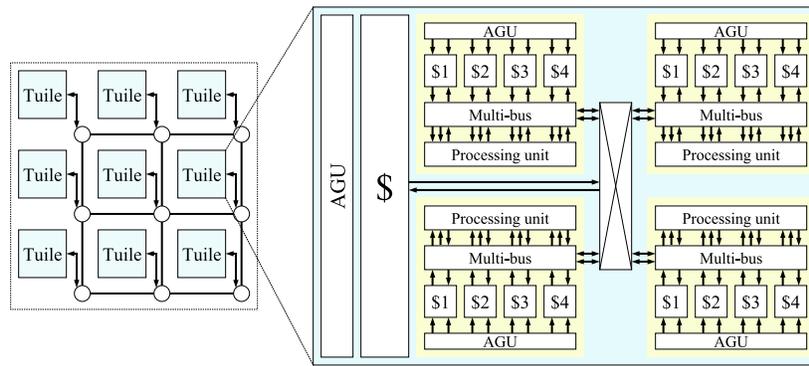


Fig. 4. System-level architecture of MOREA.

TABLE II
IMPLEMENTATION AREA COST OF MULTI-BUS AND CROSSBAR
INTERCONNECTS (TECHNOLOGY CMOS *STMicroelectronics* 0.13 μm).

Nb of I/O per cluster	Multi-bus area cost	Crossbar area cost
4	50948 μm^2	104860 μm^2
6	58093 μm^2	197050 μm^2
8	64682 μm^2	312414 μm^2

access to an unique memory space. Thanks to a number of clusters equal to 4, the crossbar layout can be done in a space-efficient manner. More precisely, as shown in Fig. 4, the clusters are floorplanned assuming a symmetrical point which minimizes the crossbar area and so, propagation delays as well as power consumption.

Crossbar between clusters allows threads implemented on computing resources of a given cluster to access data stored in other clusters. This avoids useless data transfers between threads which share the same memory space. Moreover, the sizing of threads, according to instruction-level parallelism (ILP), is thus simplified since a thread can use the right amount of resources spread across several clusters. However, since energy, timing and area crossbar overheads are evolving very quickly with I/O amount and wire length (see table II), full connectivity within the tile cannot be allowed and special attention must be paid on its sizing. Practically, length of wire can be limited in MOREA thanks to clusters floorplanning and only 4 extra transfers are allowed for each cluster.

At the lowest level of the hierarchy, clusters are built of 4 local memories storing threads data. According to processing needs and bandwidth requirements, these storage primitives, depicted in Fig. 5, can be used as single-port memories or simple dual-port memories allowing concurrent read and write operations. As shown in Fig. 4, local memories are connected to processing unit with multi-bus networks allowing complete connectivity between computing and storage resources within a cluster. An address generation unit is available for each port of the local memories. These AGUs will be discussed in the next sections and are used to automatically and efficiently generate addresses of the data handled in the clusters. It can be noticed here that we are not focusing in this study on com-

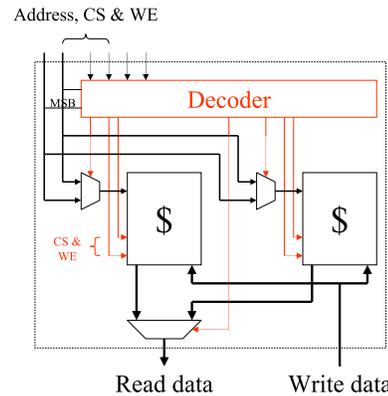


Fig. 5. Local memory architecture in MOREA.

puting resources organization and management. We consider, at first, standard organization as in the DART architecture.

The configuration of the links between AGUs and memories as well as memories and functional units are managed by an hierarchical configuration unit. First, a global controller manages the tile reconfiguration at each thread activation/ending. This global configuration includes AGU program loading as well as crossbar configuration between clusters. It may take several cycles. Next, a lower level configuration controller, located in each cluster, handles fast minor changes during the execution of a thread, e.g a modification of the multi-bus network configuration. In the next section, we are focusing on the address generation unit which manages data stream between memories and computation resources.

IV. ADDRESS GENERATION UNIT

A. Patterns classification

In embedded applications, processed data are typically organized in 1D/2D arrays. From a programming view point, these arrays are accessed through nested loop kernels and two types of address patterns are generally produced. The first type concerns the regular address sequences. In this case, the sequence $\{a\}$ can be defined as an affine function which depends on nested loop indexes. So, regular address patterns can be formulated as follow:

$$\{a(i_1, \dots, i_N) = (I^T \cdot B) \cdot C + d_0\} \quad (1)$$

```

for(i=0; i<X; i++){
  for(j=0; j<Y; j++){
    read(pixel[i][j]);
  }
}

```

Fig. 6. Example of a C code with a regular access to an $X \times Y$ image.

```

/* initialization */
begin = 0;
end = N-1;

index = (end-begin)/2;
tmp = read(A[index]);

/* dichotomy search */
while(tmp != searched_value){

  if(tmp > searched_value){
    end = index;
  }
  else{
    begin = index;
  }

  index = (end-begin)/2 + begin;
  tmp = read(A[index]);
}

```

Fig. 7. Example of a C code with an irregular access to an 1D array.

with i_j the index of loop j and i_1 the innermost loop index, N the number of nested loops and I the loop index vector, B an $N \times M$ matrix which determines the index(es) of the M -dimension array element accessed at iteration I , C a translation vector which defines the address memory of each array element and d_0 the address offset value. For example, in the case of an image processing which extracts pixel value through two nested loops, as presented in Fig. 6, the sequence $\{a\}$ is defined as¹:

$$\left\{ a(i, j) = [i \ j] \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot [Y \ 1] + *pixel[0][0] \right\} \quad (2)$$

On the contrary, irregular address patterns do not exhibit any linearity. For example, Fig. 7 presents the dichotomy search algorithm. As we can see, the index of the read array element depends on a tested condition in the *if* statement. Therefore, this sequence can not be modelled with an affine function.

The next section describes the AGU architecture and explains how it can be programmed to generate regular and irregular address sequences.

B. Address generator architecture

Generally, AGUs are built as RISC cores [13]. Because of Von Neumann execution model, such AGUs can implement all types of address pattern. Nonetheless, for regular address sequences, instruction fetching and decoding waste significant amount of time and energy. In MOREA, we propose an hybrid architecture built on the basis of a RISC core coupled with a reconfigurable unit (RU). A RU generates regular patterns with

¹*pixel[0][0] is the memory address of the first pixel

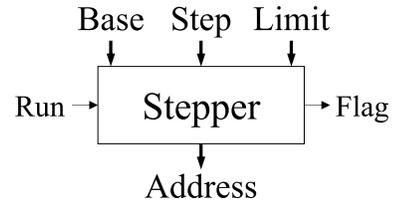


Fig. 8. System-level view of a stepper.

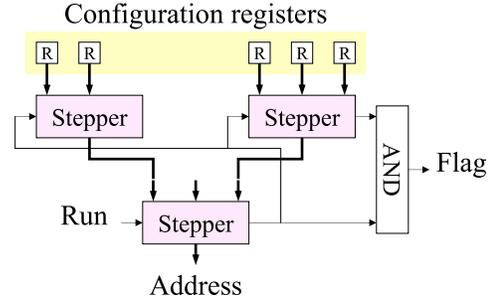


Fig. 9. Reconfigurable Unit (RU) architecture.

few configuration instructions. Thus, it allows to save time and energy for regular address sequences generation.

The RU architecture is based on generic address generator (GAG) developed for the Xputer machine [14]. This address generation architecture supports efficiently complex regular address patterns. GAGs are made of *steppers* and the AGU uses this unit as its building block.

A stepper produces a linear sequence of M addresses a_k at clock frequency and according to (3). Pattern generation starts when *Run* goes up (see Fig. 8). As soon as *Output* is equal to *Limit*, *Flag* is asserted which signals the end of the sequence.

$$\begin{cases} a_0 = \text{Base} \\ a_{k+1} = a_k + \text{Step}, \text{Step} \in \mathbb{Z} \\ a_{M-1} = \text{Limit} \end{cases} \quad (3)$$

Basically, *Base*, *Step* and *Limit* parameters describe one loop index evolution. Since index value update and comparison are automatically done within a clock cycle, a stepper generates more efficiently stride-based address patterns than typical RISC core. Nevertheless, embedded applications manage 2D data arrays which are accessed through two nested loops. In this case, *Base* and *Limit* inputs are no longer fix and depend on the outermost loop index. Therefore, in order to sustain high address generation throughput, two supplementary steppers are added to compute *Base* and *Limit* values according to the outermost loop index evolution (Fig. 9). To update the outermost loop index when the innermost loop exits, the *Flag* output of the first stepper is connected to the *Run* input of the two other steppers.

Fig. 10 shows the AGU architecture. It is made of a control unit which fetches and decodes address generation instructions. These instructions ensure the configuration of all stepper entries. Each stepper entry is directly connected to a specific register. The AGU includes an ALU which allows computation necessary to configure stepper entries. Furthermore, the ALU

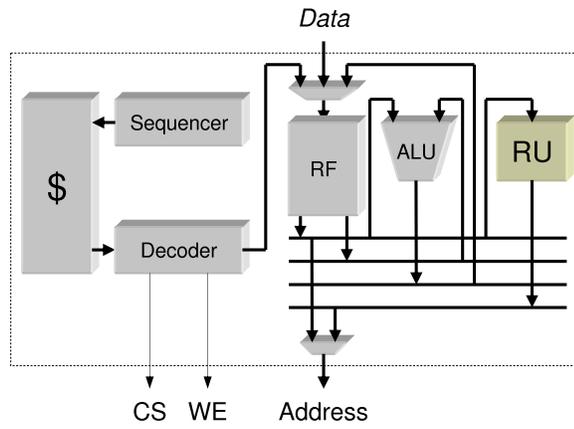


Fig. 10. Address generator architecture.

TABLE III
INSTRUCTION SET OF CORE PROCESSOR.

Instruction mnemonics	Operands	Description
LOAD	$RA_i, \langle value \rangle$	$RA_i \leftarrow \langle value \rangle$
STORE	RA_i	$RA_i \leftarrow data$
MOVE	RX_i, RA_j	$RX_i \leftarrow RA_j$
OUT {W, R}	RA_i	$@ \leftarrow RA_i$
ADD	RA_i, RA_j, RA_k	$RA_i \leftarrow RA_j + RA_k$
SUB	RA_i, RA_j, RA_k	$RA_i \leftarrow RA_j - RA_k$
AND	RA_i, RA_j, RA_k	$RA_i \leftarrow RA_j \cdot RA_k$
LSH	$RA_i, \langle direction \rangle, \langle n \rangle$	logical shift
ROP {W, R}	$\langle N \rangle$	$@ \leftarrow RU (N \text{ cycles})$
BNZ	$\langle label \rangle$	branch if not zero
BS	$\langle label \rangle$	branch if signed
BF	$\langle label \rangle$	branch if PU flag set
WAIT	$\langle N \rangle$	wait for N cycles
END		end of execution
NOP		No Operation

can be used for every address calculation. This unit is the main block to produce irregular address sequences. The ALU block supports several simple operations, e.g add/sub, which are completed by control and transfert instructions at core processor level. Table III shows the instruction set of the core processor. Plus, a status register indicates sub operation results in a short manner. ALU computes operands from register file and data network as well as immediate values. Consequently, a lot of addressing modes are available.

V. EXAMPLES OF ADDRESS GENERATION

In this section, we discuss AGU performance and consumption and present three examples of address generation with the corresponding AGU program. The two first examples are simple but show that it is possible to generate efficiently regular and irregular address sequences with our architecture. The third example presents a more complex application, the motion estimation. Finally, we present some results about the implementation of our address generator structure. With the first estimation, we give some information about the area cost and power consumption.

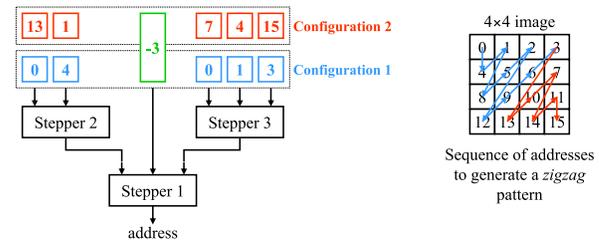


Fig. 11. Configuration of stepper entries to execute a zigzag scan pattern for a 4×4 image.

```

LOAD RA0, 0
MOVE RX0, RA0
MOVE RX2, RA0
LOAD RA0, -3
MOVE RX1, RA0
LOAD RA0, 4
MOVE RX3, RA0,
MOVE RX3, RA0,
LOAD RA0, 1
MOVE RX4, RA0
MOVE RX4, RA0
LOAD RA0, 1
MOVE RX5, RA0
ROP R

LOAD RA0, 13
MOVE RX0, RA0
LOAD RA0, 7
MOVE RX2, RA0
LOAD RA0, 1
MOVE RX3, RA0,
MOVE RX6, RA0
LOAD RA0, 4
MOVE RX4, RA0
LOAD RA0, 15
MOVE RX5, RA0
ROP R
    
```

a) Program for configuration 1. b) Program for configuration 2.

Fig. 12. AG program to execute a zigzag scan pattern to a 4×4 image.

A. Example of regular address pattern generation

The first algorithm considered here is the zigzag scan pattern as it can be used in JPEG compression for example. Fig. 11 presents the two consecutive RU configurations needed to produce the address sequence for a 4×4 image.

The first part of the address sequence (sequence 0, 4, 1, 8, 5, 2, 12, 9, 6, 3) is produced by the first configuration. To produce the second part (sequence 13, 10, 7, 14, 11, 15), a second configuration is necessary.

For the first configuration, the steppers 2 and 3 allow to produce the consecutive *base* (stepper 2) and *limit* (stepper 3) addresses for the address sequence. The generator produces addresses in 4 steps : first step, *base* = 0, *limit* = 0 which produces address 0; second step, *base* = 4, *limit* = 1, *step* = -3 which produces addresses 4, 1; third step, *base* = 8, *limit* = 2, *step* = -3 which produces addresses 8, 5, 2; etc.

The same analysis can be done for the second configuration.

For this kind of address sequence, our solution can ensure the generation of addresses with two consecutive configurations. They are done through the AGU program presented in Fig. 12.

B. Example of irregular address generation

Another example of address generation concerns a data-dependent address pattern. The computation of an image histogram uses the value of each pixel to address a memory and increments the pixel value occurency. Fig. 13 presents the two address generators used to implement this application. The first AGU is configured to read the value of image pixels. So, if the image is stored at address memory 1000, it generates addresses 1000, 1001, 1002, ... Then, each pixel value is used as an input data of the second AGU. This value is simply used as an offset added to the histogram base address. For

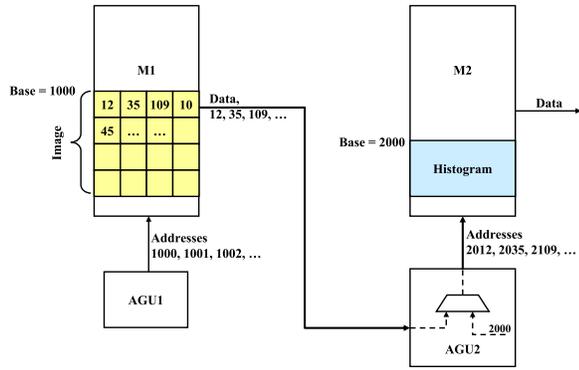


Fig. 13. Hardware implementation in MOREA of the image histogram algorithm.

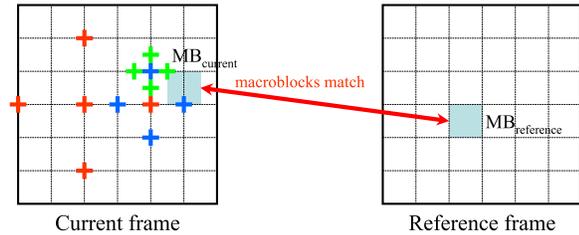


Fig. 14. Description of the motion estimation, three-step search (TSS) algorithm.

example, if the histogram is stored at address value 2000, the first address produced is $2000 + 12 = 2012$ which corresponds to the occurency of the pixel value 12.

C. Example of motion estimation algorithm

In this section, we present the implementation of the motion estimation application as it is used in video compression. Motion estimation exploits temporal redundancy within a video sequence to reduce information quantity. For each 16×16 macroblock of a *reference* frame, the most similar one, according to a particular metric, e.g Sum of Absolute Difference (SAD, see (4)), is searched within a *current* frame (Fig. 14). Then, the address memory of the first pixel of the matching macroblock is stored which defines the translation vector of the reference macroblock within the current frame.

$$SAD = \sum_{i=0}^{15} \sum_{j=0}^{15} |MB_{reference}[i][j] - MB_{current}[i][j]| \quad (4)$$

The current frame is scanned according to the *three-step search* (TSS) algorithm. In the first step, four macroblocks, at a distance $d = 8$ of the reference one, plus the centre macroblock (red marks in Fig. 14) are compared. Then, in the second step, the centre of the search path is moved to the macroblock with the minimum distortion and the distance d is halved, etc. Consequently, contrary to the preceding examples, SAD comparisons require branching mechanism in AGU. A current macroblock is accessed through a pseudo-code C which is presented in Fig. 15.

To produce this sequence of addresses, our model of address generator allows two execution models. The first consists in

```
do{
  tmp = DATA; /* External data */
  tmp += OFFSET; /* MB first pixel address */

  /* Read of current MB */
  for(i=0; i<16; i++){
    for(j=0; j<16; j++){
      pixel_read_address(tmp+j);
    }
    tmp += WIDTH;
  }
} while(condition)
```

Fig. 15. Part of the motion estimation TSS algorithm which focuses on the $MB_{current}$ pixels address generation.

<pre>STORE RA0 LOAD RA1, 25344 label13 : ADD RA0, RA0, RA1 LOAD RA1, 0 label12 : LOAD RA2, 0 label11 : LOAD RA3, 1 ADD RA2, RA2, RA3 LOAD RA3, 16 SUB RA3, RA3, RA2 BNZ label11 ADD RA3, RA0, RA2 OUT R, RA3 LOAD RA3, 1 ADD RA1, RA1, RA3 LOAD RA3, 16 SUB RA3, RA3, RA1 BNZ label12 LOAD RA3, 144 ADD RA0, RA0, RA3 WAIT N BXF label13 STORE RA0 LOAD RA1, 25344</pre>	<pre>STORE RA0 LOAD RA1, 25344 label : ADD RA0, RA0, RA1 MOVE RX0, RA0 LOAD RA1, 1 MOVE RX1, RA1 MOVE RX6, RA1 LOAD RA1, 15 ADD RA0, RA0, RA1 MOVE RX2, RA0 LOAD RA0, 144 MOVE RX3, RA0 MOVE RX4, RA0 ROP R, 255 WAIT N BF label STORE RA0 LOAD RA1, 25344</pre>
a)	b)

Fig. 16. AGU programs to produce motion estimation memory accesses; a) this version only uses the classical processor instructions, i.e. without using the reconfigurable unit instruction set; b) this version uses the reconfigurable unit instruction set to produce the addresses of the $MB_{current}$ pixels.

executing a program which produces, cycle by cycle, the value onto the address bus of the image memory (see Fig. 16.a). The second consists in configuring the reconfigurable unit of the AGU and keeps this unit producing the correct sequence of address (see Fig. 16.b)

These two programs produce exactly the same sequence of addresses but, if we consider their execution, we can see that the second needs much less cycles than the first. Table IV shows that the presence of a reconfigurable unit in the address generator allows to drastically reduce the number of decoded instructions. So, the energy consumption will be also strongly reduced. Furthermore, for the version which uses the reconfigurable unit, the AGU produces one address per cycle, while the version without can produce a new address each 9 cycles. This last point is very important and ensures that the address generator can support the processing unit frequency.

table

TABLE IV
COMPARISON BETWEEN THE TWO POSSIBLE ADDRESS GENERATOR PROGRAMS, I.E WITH AND WITHOUT USING RECONFIGURABLE UNIT.

	w/o RU	w RU	Gain
Code size (number of instructions)	23	17	26%
Execution time (number of cycles)	35011	6827	81%
Number of accesses to instruction memory	32696	197	99%

TABLE V
AGU ARCHITECTURE SYNTHESIZED AREA RESULTS.

	Area cost (μm^2)			RU area %	
	RU	AG datapath	AG	vs. AG DP	vs. AG
Global memory	7962	26792	125414	30%	6%
Local memory	4529	15857	47998	29%	4%

VI. IMPLEMENTATION RESULTS OF OUR ADDRESS GENERATOR ARCHITECTURE

To validate our proposal, our address generator architecture has been synthesized with Synopsys Design Compiler tool for CMOS *STMicroelectronics* $0.13\mu\text{m}$ transistor technology. The timing constraint is set to 5ms to produce a 200MHz clock frequency. Actually, two address generation units were implemented. First, an architecture for a $64K \times 32$ bits global data memory. It handles 16-bit address data and its instruction memory can store up to 256 23-bit words. Second, an AGU for a 256×32 bits local memory which computes 9-bit operands. The capacity of its instruction memory is 64×16 bits. Table V presents the extracted area results. The reconfigurable unit covers 30% of AGU datapath area. Overall, it represents less than 10% of the combined datapath and instruction memory area.

However the integration of a reconfigurable unit in the AGU improves significantly address generator performances (Table VI). Thus address generation throughput, expressed as Million of Addresses Per Second (MAPS), is increased by a factor $\times 5$ as compared with a RISC architecture. Moreover, the area efficiency is also improved by a factor $\times 4.8$. So, this hybrid structure offers a much better area/performance tradeoff for versatile pattern generation than typical solutions.

Besides, thanks to RU capabilities, up to 99% of instruction memory accesses can be eliminated (see table IV), and so the dynamic energy consumption.

Finally, we must consider the area penalty due to AGU within the memory subsystem. As shown in Fig. 17, the relative occupied silicium area by AGU within memory subsystem decreases as memory capacity increases. For example, a 30% area penalty is induced for a 2048-word data memory. Consequently, our address generation solution incurs acceptable area overhead for large memory block.

VII. CONCLUSION

In this paper, we have presented a distributed memory architecture and, in particular, the structure of the address

TABLE VI
PERFORMANCES OF THE TWO POSSIBLE AGU ARCHITECTURES, I.E WITH AND WITHOUT USING A RECONFIGURABLE UNIT, FOR THE MOTION ESTIMATION ALGORITHM IMPLEMENTATION.

	Performance (MAPS)		Area efficiency (MAPS/ mm^2)	
	w/o RU	w RU	w/o RU	w RU
Global memory	19	97	162	773
Local memory	19	97	189	921

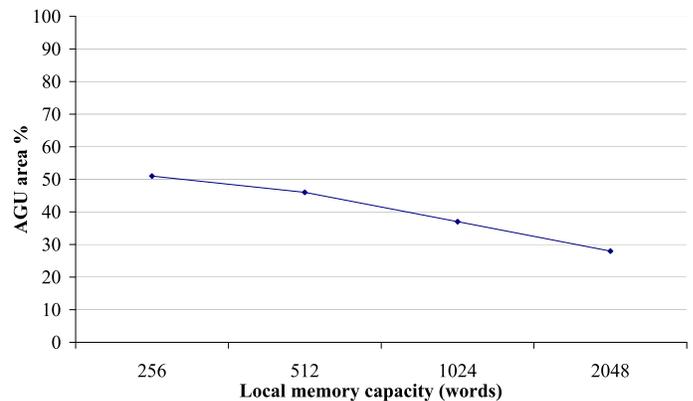


Fig. 17. AGU area penalty within MOREA cluster memory subsystem.

generator. This specific block is built from a core processor which executes an address generation program. To be efficient, this block includes a reconfigurable unit which is able to support the generation of regular address sequences. The core processor supports the configuration of the RU. The regular address sequence is then produced through sequential logic configured to generate one address per cycle. For all irregular address patterns, the core processor can execute classical instructions through a small execution pipeline.

We have showed how our generator architecture can be configured to produce regular and irregular sequences of addresses, via the implementation of representative data access patterns in our AG. Furthermore, we have presented implementation results of our address generator. These results show that substantial time and energy savings can be obtained thanks to the reconfigurable unit. Further works include the prototyping of a complete MOREA tile and the demonstration of a complete embedded vision application.

REFERENCES

- [1] R. Hartenstein, "A Decade of Reconfigurable Computing : A Visionary Retrospective," in *DATE'01 : Proceedings of the conference on Design, Automation and Test in Europe*. IEEE, 2001, pp. 642–649.

- [2] R. David, S. Pillement, and O. Sentieys, *Low Power Electronics Design*. CRC Press, 2005, ch. Low Power Reconfigurable Processor, pp. 20–1–20–15.
- [3] E. J. Marinissen, B. Prince, D. Keitel-Schulz, and Y. Zorian, “Challenges in Embedded Memory Design and Test,” in *DATE’05 : Proceedings of the conference on Design, Automation and Test in Europe*. IEEE Computer Society, 2005, pp. 722–727.
- [4] T. Ngai, J. Rose, and S. J. E. Wilton, “An SRAM-Programmable Field-Configurable Memory,” in *CICC’95 : Proceedings of the IEEE 1995 Custom Integrated Circuits Conference*. IEEE, 1995, pp. 499–502.
- [5] P. Ranganathan, S. Adve, and N. P. Jouppi, “Reconfigurable Caches and their Application to Media Processing,” in *ISCA’00 : Proceedings of the 27th International Symposium on Computer Architecture*. ACM, 2000, pp. 214–224.
- [6] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas, “Memory Hierarchy Reconfiguration for Energy and Performance in General-Purpose Processor Architectures,” in *MICRO 33 : Proceedings of the 33rd annual ACM/IEEE International Symposium on Microarchitecture*. ACM, 2000, pp. 245–257.
- [7] D. H. Albonesi, “Selective Cache Ways : On-Demand Cache Resource Allocation,” in *MICRO 32 : Proceedings of the 32nd Annual International Symposium on Microarchitecture*, 1999, pp. 248–259.
- [8] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, “Gated- V_{dd} : A Circuit Technique To Reduce Leakage in Deep-Submicron Cache Memories,” in *ISLPED’00 : Proceedings of the 2000 International Symposium on Low Power Electronics and Design*. ACM, 2000, pp. 90–95.
- [9] S. Kaxiras, Z. Hu, and M. Martonosi, “Cache Decay : Exploiting Generational Behavior to Reduce Cache Leakage Power,” in *ISCA’01 : Proceedings of the 28th Annual International Symposium on Computer Architecture*. IEEE Computer Society, 2001, pp. 240–251.
- [10] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, “Drowsy caches : Simple techniques for reducing leakage power,” in *ISCA’02 : Proceedings of the 29th Annual International Symposium on Computer Architecture*. IEEE Computer Society, 2002, pp. 148–157.
- [11] C. Zhang, F. Vahid, and W. Najjar, “A Highly Configurable Cache Architecture for Embedded Systems,” in *ISCA’03 : Proceedings of the 30th Annual International Symposium on Computer Architecture*, 1995, pp. 136–146.
- [12] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz, “Smart Memories : A Modular Reconfigurable Architecture,” in *ISCA’00 : Proceedings of the 27th International Symposium on Computer Architecture*. ACM, 2000, pp. 161–171.
- [13] M. Herz, R. Hartenstein, M. Miranda, E. Brockmeyer, and F. Catthoor, “Memory Addressing Organization for Stream-Based Reconfigurable Computing,” in *ICECS’02 : Proceedings of the 9th International Conference on Electronics, Circuits and Systems*. IEEE, 2002, pp. 813–817.
- [14] R. W. Hartenstein, A. G. Hirschbiel, and M. Weber, “XPUTERS : Very High Throughput by Innovative Computing Principles,” in *JCIT’90 : Proceedings of the 5th Jerusalem Conference on Information Technology*. IEEE Computer Society Press, 1990, pp. 43–50.

Energy reduction in wireless systems by dynamic adaptation of the fixed-point specification

H.-N. Nguyen, D. Menard, R. Rocher, O. Sentieys
 IRISA/INRIA, University of Rennes,
 6 rue de Kerampont
 F-22300 Lannion
 hai-nam.nguyen@irisa.fr

Abstract—One of the most important applications of Digital Signal Processing is wireless communication. This kind of application requires low power implementation of DSP, which generally uses fixed-point arithmetic. The fixed-point architectures should be developed to maintain the energy consumption power at a reasonable level. In this paper, an approach which adapts the fixed-point specification according to the input receiver SNR (Signal-to-Noise Ratio) is proposed. To underline our approach interest, two applications are examined, one for QPSK receiver and other for WCDMA receiver. An architecture used for this dynamic precision scaling concept is also detailed.

I. INTRODUCTION

Wireless communication is one of the most important sector for Digital Signal Processing (DSP) applications. In 2007, 74 % of Digital Signal Processors sold was used for wireless applications [1]. Most of wireless terminals are nomadic and are supplied by battery. The design of low power terminals is one of the key challenges in this domain. New services are provided (image, video, Internet access) and require high data rate. Consequently, the complexity of the baseband digital part is growing. However, the energy consumption can not be increased due to the limited battery lifetime. Thus, new strategies to reduce or maintain the energy consumption power at a reasonable level must be proposed.

Efficient implementation of DSP applications in embedded systems requires the use of fixed-point arithmetic. Thus, the vast majority of embedded DSP applications are implemented in fixed-point architectures [2], [3], [4], [5]. Indeed, fixed-point architectures are cheaper and more energy efficient than floating-point architectures because in fixed-point architecture, the word-lengths of the data are lower.

The energy consumption of an application depends on the word-length of the manipulated data. The energy consumption can be reduced by decreasing the word-length of the data. Nevertheless, this also reduces the computation accuracy. The unavoidable error due to finite word-length computation increases when the data word-length is reduced. In [6], an LMS (least mean square) adaptive filter has been studied. The energy consumption is divided by a factor of two between two fixed-point

specifications having a signal to Quantization Noise Ratio of 90 dB and 30 dB.

The traditional approach to design a fixed-point system is based on the worst-case principle. For a digital communication receiver, the maximal performances and the maximal input dynamic are retained and the more constraint transmission channel is considered. Nevertheless, the noise and the signal levels evolve during time. Now, the data rate depends on the service (video, image, speech) used with the terminal and the required performances (bit error rate) are linked to the service. These different elements underline that the fixed-point specification depends on external elements (noise level, input signal dynamic range, quality of service) and can be adapted during time to reduce the average power consumption.

In [7], different trade-offs between accuracy and energy are explored in the context of Software Defined Radio. New standards like the further extension of WLAN (802.11g) offer multiple configurations according to link noise robustness and data rate. Different modes (modulation scheme and coding rate) are proposed. For each mode an optimized fixed-point specification is determined and leads to a specific implementation. The selection of a fixed-point specification for each modulation scheme and coding rate makes possible the decrease of the average energy consumption by a factor three. In this approach, the adaptation of the fixed-point specification is only linked to the modulation scheme and coding rate.

In [8], word-length tunable VLSI architecture for an OFDM (Orthogonal Frequency Division Multiplexing) demodulator has been proposed. The data word-length are determined at run time according to the observed error at the system output. For this word-length determination process, word-length search symbols are inserted in the frame. This approach saves 32% and 24% of the power consumption for different transmission channels. This technique requires a specific hardware and energy are wasted for the fixed-point optimization process which is carried-out at run time. Moreover, this technique must modify the transmission packet format and can not be used in standard systems.

In this paper, an approach in which the fixed-point specification is adapted according to the input receiver

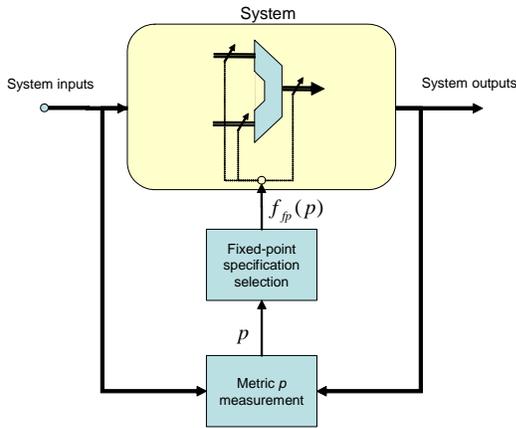


Fig. 1. Synoptic of the Dynamic Precision Scaling approach

SNR (Signal-to-Noise Ratio) for one modulation scheme and one data rate is proposed. This concept is called Dynamic Precision Scaling (DPS). Our approach interest is underlined through a WCDMA (Wide-band Code Division Multiple Access) receiver example.

The paper is organized as follows. In Section 2, the principles of Dynamic Precision Scaling is detailed and the target architectures used for implementing this concept are presented. A simple example corresponding to a QPSK (Quadrature Phase Shift Keying) receiver is analyzed in Section 3. Then, a WCDMA receiver is studied in Section 4. The path search module and the rake receiver are considered. For these different examples, the number of bits required for the integer and the fractional part are determined according to the signal-to-noise ratio.

II. DYNAMIC PRECISION SCALING (DPS)

A. Principle

In the Dynamic Precision Scaling (DPS) approach, to reduce the energy consumption, the fixed-point specification is adapted according to the external environment parameters. During time, the system switches between different fixed-point specifications when the external environment parameters are modified. In our approach, different fixed-point specifications are available. They are determined at the system design level. Let S_{fp} , be the set of all the fixed-point specifications which can be used.

To adapt the fixed-point specification to the external environment parameters, a metric p describing the external conditions is used. This metric is determined inside the digital system from the measurement of the input signal and/or the output signal. The fixed-point specification is selected according to this metric value as illustrated in Figure 1. Let f_{fp} , be the function defining the fixed-point specification to used according to the p metric value:

$$f_{fp} : \mathbb{R} \longrightarrow S_{fp} \\ p \longmapsto f_{fp}(p) \quad (1)$$

In the examples presented in this paper, the metric p is the signal to noise ratio (SNR) at the input of the receiver. Different techniques can be used to estimate this SNR [9]. For the WCDMA receiver, data-aided techniques can be used. Indeed, a pilot sequence is available in the control symbol frame (DPCCH) of the WCDMA norm in the context of UMTS/3G wireless communications. Otherwise for the QPSK receiver, SNR estimators can be used with an estimate of the transmitted symbols from the receiver decisions. The selection of the estimator is a trade-off between the estimation quality and the estimator complexity. Indeed, the supplementary energy consumption due to adaptation part must be minimized to not wreck the energy gain due to fixed-point specification adaptation.

B. Architecture for DPS

To adapt the fixed-point specification during time, the architecture must be programmable or reconfigurable. For processors, a specific code (function) is associated to each fixed-point specification. The processor switches between the part of code when the metric p is modified. For reconfigurable architectures, a configuration is associated to each fixed-point specification. The architecture is reconfigured when the metric p is modified.

To adapt the fixed-point specification during time the processing unit must be flexible in terms of supported word-length. The aim is to reduce the energy when the word-length is lower than the supported maximal value. Two kinds of approaches are available to minimize the energy consumption through word-length flexibility. One way is to have operators supporting Sub-Word Parallelism (SWP) operations. The operator processes several operations in parallel on operands of smaller word-length. An operator (multiplier, adder, shifter) with a word-length of N is split to execute k operations in parallel on sub-words of N/k word-length as illustrated in Figure 2. This technique can accelerate the code execution time up to a factor k . Thus, the energy consumes at each cycle is constant and independent of the operand word-length. But, the execution time of the processing depends of the operand word-length. Thus the global energy consumption is reduced by diminishing the processing execution time. The other way is to use operator executing only one operation per cycle but able to manipulate data with different word-lengths. In [10], a multiplier able to perform operations on 9, 11, 14 and 16 bits is proposed.

III. SYMBOL DETECTION

A. Introduction

In this section, to analyse deeply the quantization noise effects, a simple example is considered. It consists of a transmitter and a receiver using QPSK modulation. The transmission channel is additive white gaussian noise (AWGN) with E_b/N_0 varying between 0 dB and 10 dB. No channel coding is used. The input signal of the receiver and the bit error rate are observed. Throughout this paper, the

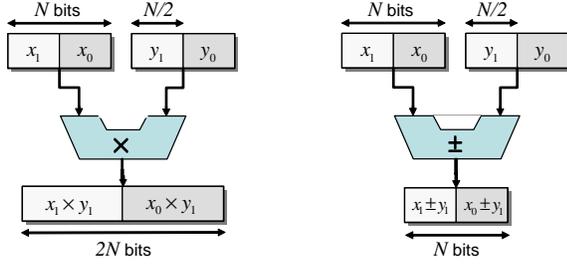


Fig. 2. Sub-Word Parallelism operator for multiplication and addition in the case of $k = 2$

term E_b/N_0 is used in the different simulations. In general, E_b/N_0 is equal to SNR divided by spectral efficiency. In the case of QPSK modulation, E_b/N_0 is equal to $\text{SNR}/\log_2 4$. In case of spread spectrum transmission with a spreading factor SF, E_b/N_0 is equal to $\text{SNR} \times \text{SF}$. For convenience, the term SNR is used in the text in the rest of the paper.

The aim of the following analyses is to determine the minimal number of bits required for the integer and fractional parts. The fixed-point specification must guarantee no overflow and maintains the bit error rate performances.

B. Range estimation

The aim of this part is to determine the minimal value of the integer word-length which guarantees no overflow. In case of an AWGN channel, the input signal $y(k)$ of the receiver is the sum of emitted symbols and white gaussian noise:

$$y(k) = s(k) + c(k) \quad (2)$$

where $s(k)$ is the set of modulated symbols:

$$s(k) \in \{\pm 1 \pm i\} \quad (3)$$

Equations (2) and (3) show that the higher the noise level is, the larger the dynamic range of received signal. For example, the dynamic range at 0 dB should be as twice as the one for very low noise level.

The dynamic range obtained for different SNR levels is presented in Figure 3. The range of input signal decreases when the signal/noise ratio increases. It decreases from 3.2 (at 0 dB) to 1.7 (at 10 dB). Between these two SNR values, one bit at the receiver input can be saved.

C. Precision analysis

The aim of this part is to determine the minimal value of the fractional word-length which leads to a Bit Error Rate (BER) close to the BER obtained with infinite precision (floating point simulation in our case) computation.

The channel is assumed to be a white gaussian $c(k)$ with noise variance $\sigma_c^2 = \frac{N_0}{2E_b}$. The expression of the probability density function (pdf) $f_c(x)$ of $c(k)$ is thus as follows:

$$f_c(x) = \frac{1}{\sigma_c \sqrt{2\pi}} \exp \frac{-x^2}{2\sigma_c^2} \quad (4)$$

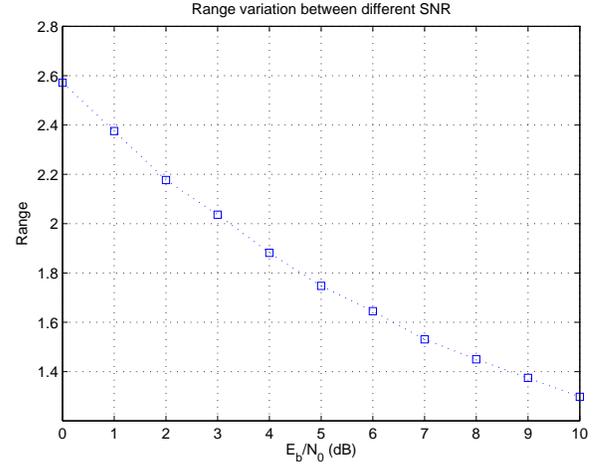


Fig. 3. Range analysis for a symbol detection in the case of AWGN channel and QPSK modulation.

The signal $y(k)$ is then quantized with quantization step size $q = 2^{-m}$. The expression of the probability density function $f_q(x)$ of the quantization noise is then as follows:

$$f_q(x) = \frac{1}{q} \text{Id}_{\left[-\frac{q}{2}, \frac{q}{2}\right]} \quad (5)$$

Thus the noise after quantization corresponding to the sum of the channel noise and the quantization noise has the following probability density function:

$$f_n(x) = f_c * f_q(x) = \int_{-\infty}^{\infty} f_c(t) f_q(x-t) dt \quad (6)$$

$$= \frac{1}{2q} \left(\text{erf} \frac{x + \frac{q}{2}}{\sqrt{N_0}} - \text{erf} \frac{x - \frac{q}{2}}{\sqrt{N_0}} \right) \quad (7)$$

and the following probability distribution:

$$F_n(x) = \int_{-\infty}^x \frac{1}{2q} \left(\text{erf} \frac{t + \frac{q}{2}}{\sqrt{N_0}} - \text{erf} \frac{t - \frac{q}{2}}{\sqrt{N_0}} \right) dt \quad (8)$$

In case of BPSK and QPSK, the probability of bit error for single quantization is:

$$\text{BER}(m, \text{SNR}) = 1 - F_n(1) \quad (9)$$

Figure 4 shows the results of the analytical expression (9) for $m \in \{0, 1, 2, 3\}$ and for infinite precision. The precision criteria is defined with a parameter a ($a \ll 1$) so that $\text{BER}(m, \text{SNR}) < (1+a) \text{BER}(\infty, \text{SNR})$. From this analysis, the theoretical data fractional word-length of a QPSK modulation can be deduced and is presented in Figure 5 for different SNR values. The fractional word-length increases with the SNR and up to three bits at the receiver input can be saved depending on the channel conditions.

In order to verify this theoretical analysis, simulations of the BER for a QPSK modulation in different quantization and channel conditions have been performed and are presented in Figure 6. As the emitted signal is binary (for real part and imaginary part), a fractional word-length

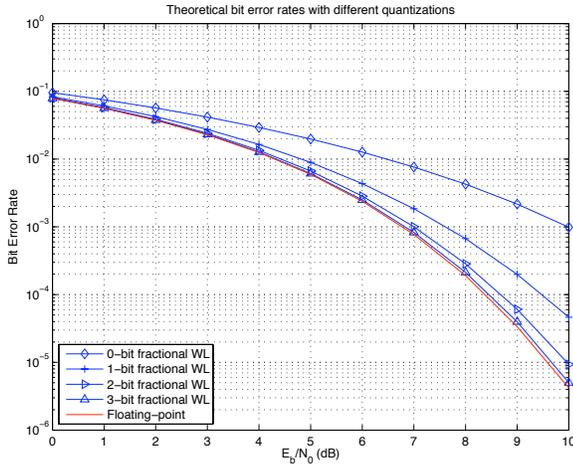


Fig. 4. Theoretical values of the BER for a QPSK modulation with an AWGN channel in different quantization and channel conditions.

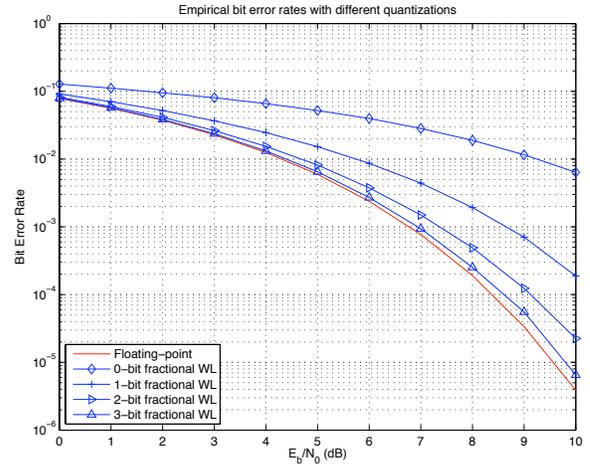


Fig. 6. Simulation values of the BER for a QPSK modulation with an AWGN channel in different quantization and channel conditions.

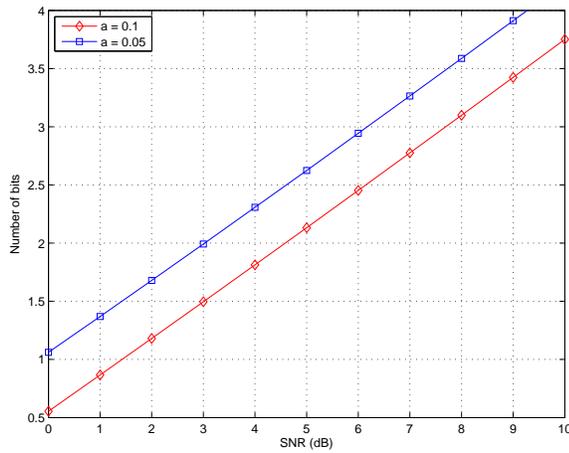


Fig. 5. Theoretical fractional word-length of a QPSK modulation with an AWGN channel for different SNR values.

of zero, one, two and three bits will be used. Again, the more the SNR increases, the longer the fractional word-length required to meet the precision criteria. Figure 6 confirms that the demodulation process needs one bit at 0 dB, two bits at 4 dB and more than three bits at 10 dB. After the approximation to finite word-length precision, the quantization error must be negligible compared to channel noise SNR. Thus, a higher number of bits is needed for fractional part when the noise level is lower.

With a dynamic adaptation of the fixed-point specification according to the SNR, a reduction of the word-length can be achieved compared to a classical approach. For instance, in this case, a classical design needs a word length of $2 + 5 = 7$ bits. With the proposed dynamic precision scaling, $2 + 1 = 3$ bits at 0 dB and $1 + 5 = 6$ bits at 10 dB. Between these two SNR values, three bits at the receiver input can be saved.

D. Precision analysis for multiple quantization errors at the receiver

In the general case, the receiver includes some processing and therefore generates multiple rounding quantization noises q_1, q_2, \dots, q_K . Due to the central limit theorem, the sum of these noises is then considered gaussian and has the following pdf:

$$f_Q(x) = \frac{1}{\sigma_Q \sqrt{2\pi}} \exp \frac{-x^2}{2\sigma_Q^2} \quad (10)$$

where

$$\sigma_Q^2 = \sum_{i=1}^K \sigma_{q_i}^2 = \sum_{i=1}^K \frac{q_i^2}{12} \quad (11)$$

Thus, the total noise including the processing and quantization errors at the receiver has the following probability density function $f_N(x)$:

$$f_N(x) = \frac{1}{\sqrt{\sigma_c^2 + \sigma_Q^2} \sqrt{2\pi}} \exp \frac{-x^2}{2(\sigma_c^2 + \sigma_Q^2)} \quad (12)$$

and the following probability distribution $F_N(x)$:

$$F_N(x) = \frac{1}{2} \left(1 + \operatorname{erf} \frac{x}{\sqrt{\sigma_c^2 + \sigma_Q^2} \sqrt{2}} \right) \quad (13)$$

In case of BPSK and QPSK, the probability of bit error for multiple quantizations is:

$$\begin{aligned} \operatorname{BER}(\sigma_Q, \operatorname{SNR}) &= 1 - F_N(1) = \frac{1}{2} \operatorname{erfc} \frac{1}{\sqrt{\sigma_c^2 + \sigma_Q^2} \sqrt{2}} \\ &= Q \left(\frac{1}{\sqrt{\sigma_c^2 + \sigma_Q^2}} \right) \end{aligned} \quad (14)$$

The case of the WCDMA receiver presented in the following section corresponds to this theoretical BER.

IV. WCDMA RECEIVER

A. Presentation

WCDMA is a standard for the third-generation of cellular network which is based on DS-CDMA (Direct Spread CDMA) technology. In these systems, a rake receiver is used to counter with the effects of multi-path fading. A finger is allocated to each path to decode the symbol associated with the path. One important component associated with the rake receiver is the path searcher. A path searcher finds the delay of different paths, which is then used to synchronize the input signal with the code generated in the receiver and thus to obtain an optimal combination of received energy.

In WCDMA, there are two layers of spreading codes [11]: channelization code and scrambling code. The channelization code C_{ch} is used to achieve orthogonality between channels when time-shift is equal to 0. The scrambling codes used in uplink are Gold codes S_G . The input data d_t is multiplied with the spreading codes, and the transmitted signal Tx_t is:

$$Tx_t = d_t C_{ch} S_G \quad (15)$$

In a multi-path (time dispersive) Rayleigh channel, the global received signal Rx_t is the sum of elementary signals $RX_{k,t}$ for different channel paths:

$$Rx_{k,\tau_k} = a_k Tx_{-\tau_k} + n_k + i_k \quad (16)$$

where τ_k is the delay of k^{th} path in the channel, a_k is the attenuation, n_k and i_k are additive white gaussian noise and channel interference respectively.

At the receiver, the delay τ_k of each path is estimated by the *path searcher* module. Then, the symbol are decoded with a rake receiver made-up of different fingers, one for each estimated path. Each path is processed by a finger and is despreading synchronously by multiplying with conjugated spreading codes:

$$\begin{aligned} d_{r,k} &= Rx_{k,\tau_k} C_{ch} S_G^* \\ &= (a_{k,\tau_k} Tx_t + n_{k,\tau_k} + i_{k,\tau_k}) C_{ch} S_G^* \\ &= (a_{k,\tau_k} d_t C_{ch} S_G + n_{k,\tau_k} + i_{k,\tau_k}) C_{ch} S_G^* \\ &= a_{k,\tau_k} d_t + (n_{k,\tau_k} + i_{k,\tau_k}) C_{ch} S_G^* \end{aligned} \quad (17)$$

Then, $d_{r,k}$ is summed up on a symbol duration. Due to the code properties, the second part of the right handside of (17) is negligible regarding to the first part which is $a_{k,\tau_k} d_t \cdot SF$, with SF the spreading factor. SF is then the processing gain of spreading spectrum. In our case, a spreading factor of $SF = 16$ is used for DPDCH, which is the data symbol frame of the WCDMA norm in the context of UMTS/3G wireless communications at the speed of 240 Kbps without channel coding. The processing gain is thus equal to 12 dB.

A Rayleigh channel model respecting the 3GPP channel case 3 [12] without Doppler effect is used, corresponding to a multi-path fading with four path components (gain, delay): (0 dB, 0 ns); (-3 dB, 261 ns); (-6 dB, 521 ns); (-9 dB, 781 ns).

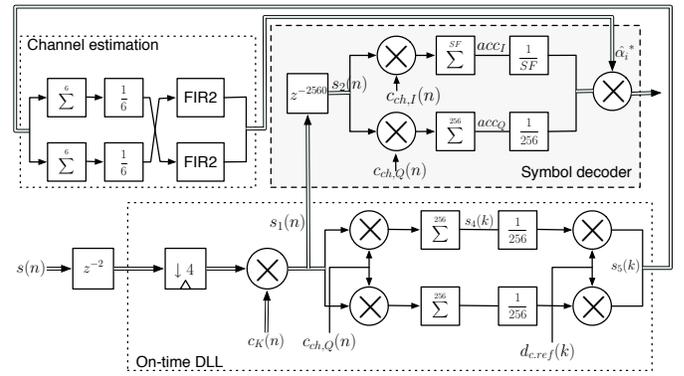


Fig. 7. Data flow graph of a symbol decoder in the WCDMA receiver.

B. Symbol detection

1) *Range estimation*: The flow graph of one finger of a rake receiver is presented in Figure 7. In the finger, the correlation between the input signal and the codes is used to amplify useful signal to detect the transmitted symbol. The correlation process increases the range of useful signal, but not that of noise. An approach is then proposed to determine more accurately the data dynamic range. Before the correlation process, the whole useful signal plus noise is considered. After this process, only the useful signal is taken into account when calculating the dynamic range.

In equation (16), the received signal is:

$$s(n) = \sum Rx_k = \sum a_k Tx_{-\tau_k} + ni_k \quad (18)$$

where ni_k is interference plus noise, which can be considered gaussian with variance σ^2 . Assuming that a user has one DPDCH channel, from (15), $d_t C_{ch}$ take values in $\{\pm 1 \pm i\}$. Thus $Tx = d_t C_{ch} S_G \in \{\pm 2, \pm 2i\}$. In our simulations, Tx is normalized into $\{\pm 1, \pm i\}$, hence its power is 1. By definition, $SNR = 1/\sigma^2$.

The increase of users in a cell rises the interferences. These interferences are processed as noises and thus increase the SNR. To take account of the different cases, a few number of communications in a cell with good transmission conditions and a great number of users with bad transmission conditions, a great range of SNR is considered (0 to 25 dB).

The input $s(n)$ consisting of signal plus noise is normalized to have the maximum amplitude of both real and imaginary parts one. Because a gaussian noise σ^2 has 99.7% of its values in $[-3\sigma, 3\sigma]$, assuming the real and imaginary parts of $\sum a_k Tx_{-\tau_k}$ are in $[-1, 1]$, the input is considered in $[-1 - 3\sigma, 1 + 3\sigma]$. The normalization process is thus implemented by dividing the input by $1 + 3\sigma$ then cut off by 1.

Multiplication with complex scrambling code $c_K(n)$ results in doubling the amplitude. Each real and imaginary part is then multiplied with OVSF code, which does not change absolute value. Averaging in 256 chips results in an important attenuation of noise, which leads the

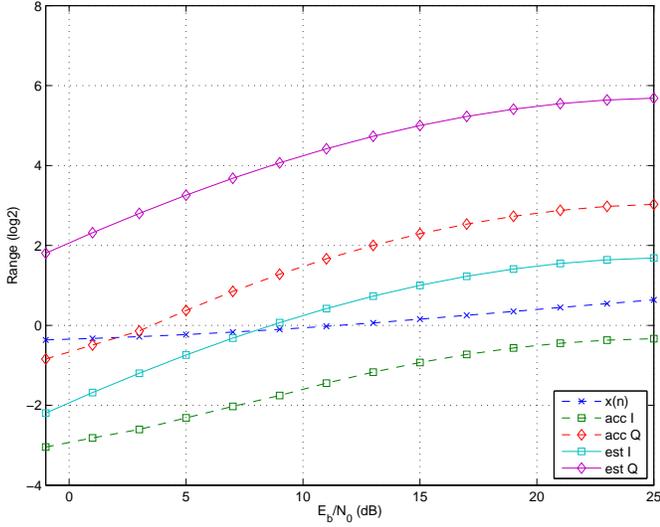


Fig. 8. Estimated and simulation based values of data range in different channel conditions for the symbol decoder.

output to signal-only. Thus, to evaluate the dynamic range after the correlation, only the useful signal is taken into account. Both real and imaginary parts of $s_5(k)$ are in $[-\frac{2}{1+3\sigma}, \frac{2}{1+3\sigma}]$. As a result, the channel estimation coefficient $\hat{\alpha}_i$ is in $[-\frac{2}{1+3\sigma}, \frac{2}{1+3\sigma}]$.

Multiplication with real OVFS code does not change dynamic range. Then, accumulation results – with the same explication as above – in $[-\frac{4 \cdot SF}{1+3\sigma}, \frac{4 \cdot SF}{1+3\sigma}]$ for the real part and in $[-\frac{4 \cdot 256}{1+3\sigma}, \frac{4 \cdot 256}{1+3\sigma}]$ for the imaginary part.

Estimated and simulation based values are presented in Figure 8. The acc_I and acc_Q data are studied as they have the largest dynamic range and the largest variation when SNR changes. In both simulation and estimation, there are a difference of 3 bits between 0 dB and 15 dB, 4 bits between 0 dB and 25 dB.

It is noticed that there are a difference from one to two bits between estimated and simulated results. This difference is explained by the channel model used in the simulation. If a single path channel model, for example, is used, the difference is less than 1 bit. Moreover, the analytical estimations are more pessimistic.

2) *Precision analysis:* Similar to (11), the total quantization noise in a symbol decoder can be presented by the sum of quantization noises in each step. Suppose the same bits N are used in every quantization, some calculations show that:

$$\sigma_{Q,I}^2 = \frac{2^{-2N}}{12} \times 9 \cdot SF \quad (19)$$

$$\sigma_{Q,Q}^2 = \frac{2^{-2N}}{12} \times 9 \cdot 256 \quad (20)$$

The normalization of the signal has an impact on noise: $\sigma'^2 = \sigma^2 \times \frac{1}{(1+3\sigma)^2}$. The total noise $\sigma_{Q,I}^2 + \sigma'^2$ has then less dependance on σ (thus, SNR). In fact, it is showed in Figure 9 that about 10 bits are sufficient to approximate

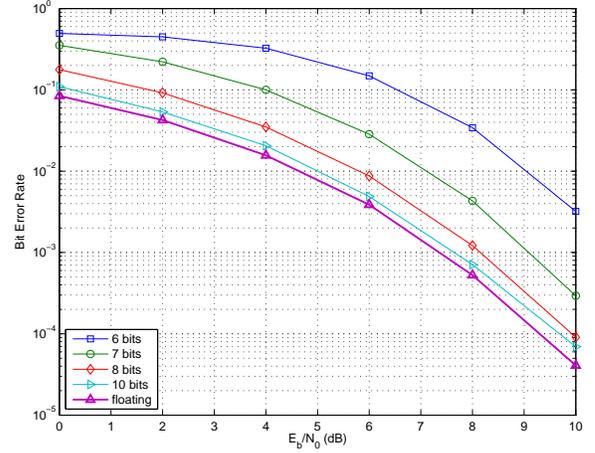


Fig. 9. Simulation based values of BER in different channel conditions for the symbol decoder.

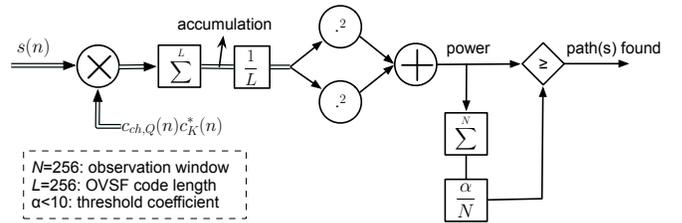


Fig. 10. Data flow diagram of a PDP path searcher

floating-point precision when E_b/N_0 varies between 0 dB and 10 dB.

In conclusion for the rake receiver, the optimised word-length of the output is equal to 16 bits at 25 dB and 12 bits at 0 dB. Thus, between the two SNR values 4 bits can be saved.

C. Path searcher

A path searcher (PS) with power delay profile (PDP) algorithm is now studied. This module analyses in temporal windows of a chip length the correlation between the input signal and the code generated inside the receiver. This PS module achieves the coarse-grain delay synchronization and then the fine-grain synchronization will be carried-out by the Delay Locked Loop (DLL) inside each finger. The processing is focused on the control channel and the unknown complex amplitude a_k is removed by computing the real and imaginary part and then taking the module.

The signal flow graph is presented in Figure 10. First, the PS module computes the correlation between the input signal and the code associated with the control channel. Then, the square module of the correlation is computed and this value is compared to an adaptive threshold $t(l)$. A path is detected if this value is greater than the threshold. The adaptive threshold is proportional to the average of all the correlation values.

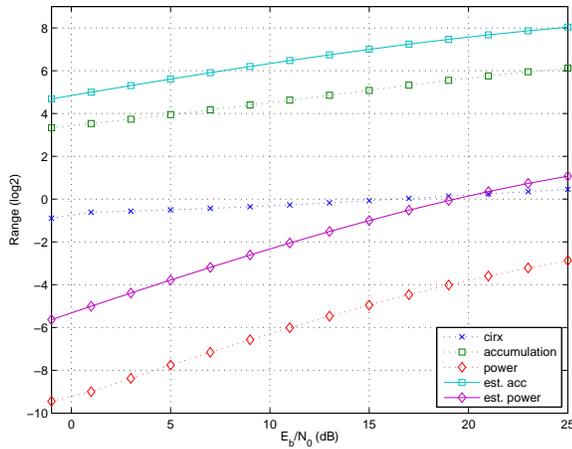


Fig. 11. Estimated and simulation based values of range for the Path Searcher.

1) *Range estimation*: Firstly, the filtered received data RX is normalized into $[-1, 1]$ with the same method that the one presented in IV-B.1. It is then multiplied with complex conjugate of spreading code $C_{ch}S_n^*$ and results in $[-2, 2]$ for each real and imaginary part. Accumulation along with N_W symbols (N_W : correlation window size) – only the signal is summed up significantly – results in $[-\frac{2N_W}{1+3\sigma}, \frac{2N_W}{1+3\sigma}]$, and then averaging into $[\frac{-2}{1+3\sigma}, \frac{2}{1+3\sigma}]$. All these analyses are summarized by the data flow graph of Figure 10.

The estimated and simulation based dynamic range of each value is presented in Figure 11. Both estimated and simulation based results have a difference of 3 bits in accumulation value between 0 dB and 25 dB, and a difference of 6 bits in power profile.

It is noteworthy that estimated and simulated results differ of 1 or 2 bits. This depends on the channel model and is due to fact that the Nyquist filter is not taken into account, which can slightly modify the values.

2) *Precision analysis*: In the Path Searcher, quantization noise of each power profile is chi-square distributed with variance of:

$$\sigma_{Q,PS}^2 = \frac{q^4 N^2}{9} = \frac{2^{-4N} L_c^2}{9} \quad (21)$$

where $L_c = 256$ the length of OVFSF code in DPCCH channel.

This PS module is based on the decision theory and classical criterions are used to analyze the performances. The *misdetctions* corresponding to the non-detection of an existing path and the *false-alarms* corresponding to the detection of a non-existing path are measured.

The results are obtained with a monte-carlo approach. Simulations are performed for different Rayleigh channel models, each having four paths. For each kind of Rayleigh channel, 500 experiments are carried out. In Figure 13, the average number of paths which have been missed are

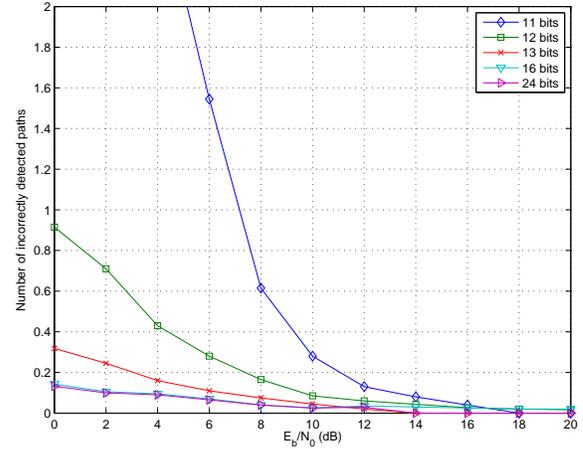


Fig. 12. Number of false-alarms for the Path Searcher.

reported. In Figure 12, the average number of non-existing paths which have been detected are reported.

Given that the precision is limited for small word-length, a lot of power values are coded with a value of 0 and thus the average value $t(l)$ is smaller than in the infinite precision case. Consequently, the number of false-alarms increases when the fractional word-length decreases. The number of misdetections decreases with the fractional word-length. This phenomenon appears surprising at first sight, but is due to the threshold reduction when the fractional word-length is decreased. From these results two conclusions can be drawn. The threshold $t(l)$ depends of the SNR even in the case of infinite precision. The reduction of the computation accuracy modifies the threshold $t(l)$ value, and, in our case, the misdetection is too important for low SNR and low computation accuracy. Thus, the parameter α must be adapted according to the SNR value and the fixed-point specification. This adaptation of the parameter α according to the SNR will allow the improvement of the misdetection and false-alarm probability.

V. CONCLUSION

In this paper, the concept of energy consumption reduction by adapting the fixed-point specification is addressed. The concept and the target architecture were presented. The results show that the global number of bits required to limit the degradation of the BER depends on the SNR at the receiver input. For the rake receiver the difference is around 4 bits between different values of the SNR. For the path searcher, the false-alarm and misdetection probability are used as performance metric. The difference is around 6 bits between the extreme values of SNR. This great difference is due to the square operation which emphasizes the phenomenon. For the future work, the energy consumption associated to each fixed-point specification

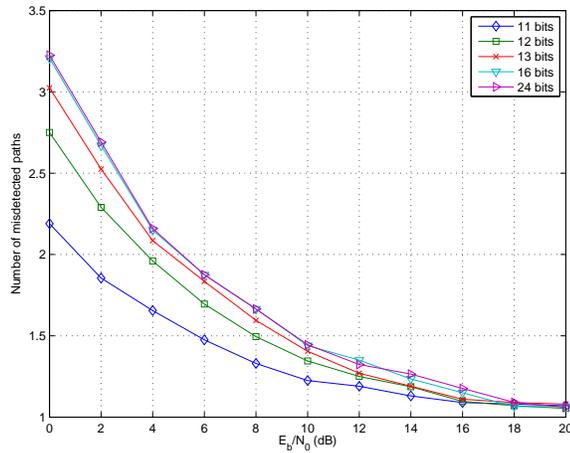


Fig. 13. Average number of misdetections for the Path Searcher.

will be determined to evaluate the gain in terms of energy consumption of our approach.

REFERENCES

- [1] W. Strauss, "Hanging up on analog and flexing Wireless/DSP muscles," Forward Concepts, Tech. Rep., 2008.
- [2] B. Evans, "Modem Design, Implementation, and Testing Using NI's LabVIEW," in *National Instrument Academic Day*, Beirut, Lebanon, June 2005.
- [3] J. Eyre and J. Bier, "The evolution of DSP processors," *IEEE Signal Processing Magazine*, vol. 17, no. 2, pp. 43–51, March 2000.
- [4] W. Strauss, "DSP chips take on many forms," DSP-FPGA.com Magazine, March 2006.
- [5] J. Eyre and J. Bier, "DSPs court the consumer," *IEEE Spectrum*, vol. 36, no. 3, pp. 47–53, 1999.
- [6] R. Rocher, D. Menard, N. Herve, and O. Sentieys, "Fixed-Point Configurable Hardware Components," *EURASIP Journal on Embedded Systems*, vol. 2006, no. 1, pp. 20–20, 2006.
- [7] D. Novo, B. Bougard, A. Lambrechts, L. Van der Perre, and F. Catthoor, "Scenario-based fixed-point data format refinement to enable energy-scalable software defined radios," *Design, Automation and Test in Europe, 2008 (DATE '08)*, pp. 722–727, March 2008.
- [8] S. Yoshizawa and Y. Miyana, "Tunable word length architecture for low power wireless OFDM demodulator," *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, May 2006.
- [9] D. Pauluzzi and N. Beaulieu, "A comparison of SNR estimation techniques for the AWGN channel," *Communications, IEEE Transactions on*, vol. 48, no. 10, pp. 1681–1691, Oct 2000.
- [10] M. Bhardwaj, R. Min, and A. Chandrakasan, "Power-aware systems," in *34th Asilomar Conference on Signals, Systems and Computers*, December 2000.
- [11] K. Tachikawa, *W-CDMA Mobile Communications System*. Wiley, 2002.
- [12] 3GPP, "TS 25.104 V8.2.0: Base Station radio transmission and reception (FDD)," 2008.

Motion Aware Slicing for H.264 Selective Visual Encryption

Marc LENY

Institut TELECOM,
TELECOM & Management
SudParis, ARTEMIS Dept
9 rue Charles Fourier,
91011 Evry Cedex, France
marc.leny@fr.thalesgroup.com

Françoise PRÊTEUX

Institut TELECOM,
TELECOM & Management
SudParis, ARTEMIS Dpt
9 rue Charles Fourier,
91011 Evry Cedex, France
francoise.preteux@it-sudparis.eu

Cédric LE BARZ

Thales Communications,
146 bd Valmy,
92704 Colombes Cedex, France
cedric.lebarz@fr.thalesgroup.com

Abstract— The paper describes a novel approach for image representation in compressed videos. The proposed hierarchical structure directly provides a semantic object partitioning of the video sequence. In addition, we present in detail how such a representation can be integrated within the current standard as MPEG-4 AVC/H.264. The dedicated implementation yields a similar – yet restricted - stream structure provided a specific encoder. The overall sequence partitioning can be used for several challenging applications as videosurveillance and video protection. This paper focuses on a selective visual encryption which provides a fully compliant H.264 stream in which moving objects appear ciphered unless the appropriate deciphering key is used.

I. INTRODUCTION

In domains such as videosurveillance or sign-language sequences, the most important part of visual information can be reduced to a small region inside the image. You can choose to focus on mobile vehicles or pedestrians in the first case, on the hands and head of the signer in the second one. However, pointing directly one of these elements inside a video stream is currently a difficult if not impossible task considering the current standard. One of the aims of MPEG-4 is to provide Universal Multimedia Access [10], which can in our case mean being able to decode specifically a targeted object inside a video sequence. If such a result can be obtained through transparency layers each containing one object, a single video stream cannot encapsulate all these elements.

Enabling to encode independently the objects inside a video stream makes it possible to support new applications. The growth of multimedia data exchanges brings up once more the problem of confidentiality and privacy. In videosurveillance, a security operator has to watch tens of videos at the same time. The increasing number of cameras usually implies new operators. But security clearance is not always easy to obtain. If persons and vehicles inside the stream were ciphered precisely enough so that the scene could be understood, privacy issues would not be a problem anymore. Totally decrypted stream would only be available providing the appropriate decryption key.

In this paper, we first introduce a new video stream hierarchy that provides a semantically higher interpretation of the

content in Section II. From this new representation modelling, we also suggest new tools exploiting this description. Even if such a structure is not directly implementable in a current standard, we develop through Section III an MPEG-4 AVC/H.264 compatible implementation requiring some restrictions to the proposed approach. Section IV exposes an application being made possible by this specification: the selective visual encryption uses this independent object definition to blur the moving objects inside videosurveillance sequences. If the process follows the previous specifications, it is possible to cipher an already compressed stream anywhere on the transmission channel. A dedicated encoder implementation makes it possible to reach similar results now with H.264 [9]. Finally Section V gathers concluding remarks and perspectives of future work.

II. SEMANTIC OBJECT ORIENTED SLICING

We will here expose the limitations we encountered with the current standards when dealing with videosurveillance systems. Up to now our algorithms processed MPEG-2 or MPEG-4 part 2 streams, and this article exposes our studies when we started investigating H.264. Surveillance brings up specific data and requirements, that we wanted to integrate in our new H.264 implementation, but that the standard could not directly address. These observations led us to define a new approach to image hierarchy, based on semantic visual objects. The structure led to new tools, providing several improvements to the current standards.

A. Limits of the current slice partitioning

Video compression standards up to H.264 provide a framework for both independent visual objects through layers and image partitioning through slices and slices groups, yet these two domains remain different and were not defined to be directly compliant one with another.

First, independent objects as defined for scene composition are implemented as different parts of a stream, consisting of videos, texts, avatars, etc. But if each object is coded independently from the others, visualising two objects implies de-

coding two sequences. There is not a unique compressed stream containing both objects.

Secondly, when dealing with slices and slices groups inside a single video stream as a H.264 sequence, dynamic change of the partitioning tends to be tricky. Considering you can define for each image the shape of each object, H.264 does not define any adapted procedure to fit these. Current decoders will therefore expect an Intra coded image after a Picture Parameters Set unit containing an MBAmapping (Macroblock Allocation map) change. The objects can each be defined inside a slices group, using an MBAmapping. H.264 main profile only allows one slices group, making this statement impossible, but the baseline and extended profiles can contain up to height slices groups. However, considering a videosurveillance scene, one group being allocated to the background, you could only have seven independent objects in an image, which can be problematic depending on the activity in the watched site.

Inside the slices groups, slices can be defined only as a sub-portion of the macroblock stream, implying you can't choose to assign a slice to any sub-part of an object. This could be convenient for example to reconstruct only a small background rectangle behind a car after an occlusion.

B. The proposed approach

1) *Definitions:* To explain the present hierarchy of the video stream, we first have to define some terms as we consider these:

We call "object" a connected set of pixels, blocks or macroblocks which possesses its own movement inside the video sequence compared to the other objects. Dealing with videosurveillance, it can refer to a vehicle, a pedestrian, an animal, etc.



Fig. 1 Slice definition

An object group is then defined through a common descriptor chosen by the user when the stream is encoded. Such a descriptor can be low or very-low level (colour, size, direction of the movement, etc.) or medium/high level (person/vehicle/background, spatial localisation in the sequence, etc.). An object group can therefore be the red objects, the ones moving to the left or even every people inside the image.

A slice is defined as a sub-portion of an image, as opposed to a sub-portion of the stream in the current standards: up to H.264, a slice is identified inside the stream by the VLC word "first_mb_in_slice". This definition implies that from a macroblock, a slice will occupy the following entire lines of the

image, inside the current slices group, until the next slice starts. It is not possible to define any sub-part of a slices group as a slice. In the suggested approach, a slice can be defined inside the MBAmapping or by to extreme macroblocks as for a bounding box (Figure 1). The term "slice" is used as a reference / continuity to the MPEG definition, but for a better understanding, it should be interpreted as an object part in our approach.

This new slice definition allows a partial reconstruction after an occlusion for example. As shown in the Figure 2, two objects are defined, one being the car, the other the background. The car is reconstructed by motion estimation through one slice. The background however is composed of two slices: most is obtained by copying the previous image, but the part right behind the car has to be reconstructed as it was not defined in the previous image. We therefore instantiate a slice, only corresponding to this area, which allow a very fast identification and reconstruction of the two sub-parts of the background.

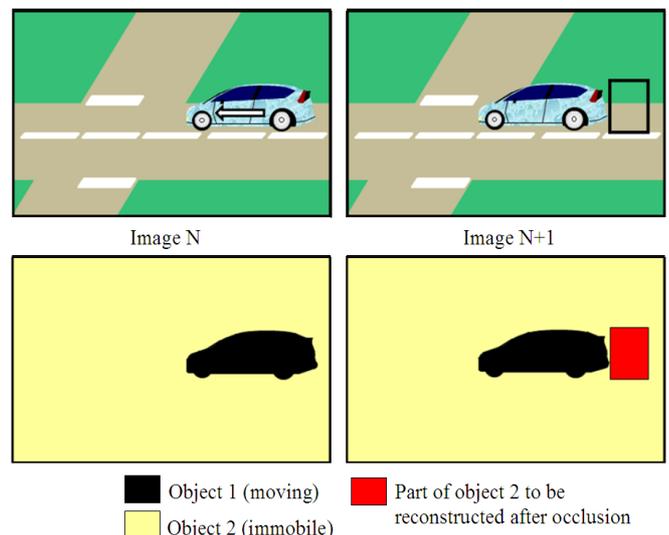


Fig. 2 Reconstruction after occlusion

2) *Hierarchy and stream structure:* Having defined these elements, we can now introduce the image hierarchy we suggest (Figure 3). An image contains one or several objects groups, without number limit to foresee future high definition videosurveillance cameras that might shoot several tens of objects at the same time. By "no limit on the number of objects", we mean that if several profiles are defined, even the base one should be able to contain over a hundred objects, and higher profiles might deal with thousands of objects (case of a very high definition camera inside an airport hall for example). Each objects group can contain one or several objects presenting a common attribute as chosen by the user at the encoding process. Finally, each object can contain one or several slices, which are defined by the encoder to optimise reconstruction.

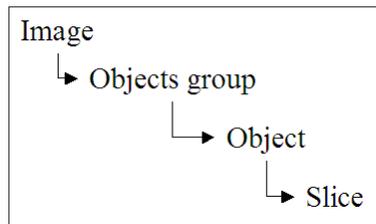


Fig. 3 Image hierarchy

In comparison with one of the underlined limit of the current standards, there is now only one video stream, containing a sequence structured as above. Figure 5 shows the video streams corresponding to the scene presented in the sequence Figure 6 and the tree Figure 4.

Each of the hierarchy elements possesses its own header, which contains the descriptors used through the encoding process to determine the slices / objects / objects groups, providing an efficient tool for fast navigation inside the stream. Descriptors shall include normalised ones, as the MPEG-7 visual descriptors, but also enable proprietary ones for wider or specific applications. The segmentation suggested is based on motion, as declared by the object definition. From this seg-

mentation, several descriptors are extracted for each object, and will then be used for the classification process (supervised or not) which provide the objects groups. The overall object declaration process leads to a motion indexation.

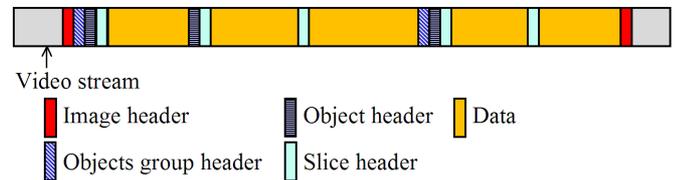


Fig. 5 Stream structure

Two objects groups, corresponding to the mobile objects and the background, are represented in the Figures 4 and 6). The first contains two objects (two cars). The first object is defined through a single slice, while the second one is made of two slices (front and back of the car). The second object group contains one object which is composed of two slices (the background already present in the previous image and the part needing a specific reconstruction after an occlusion).

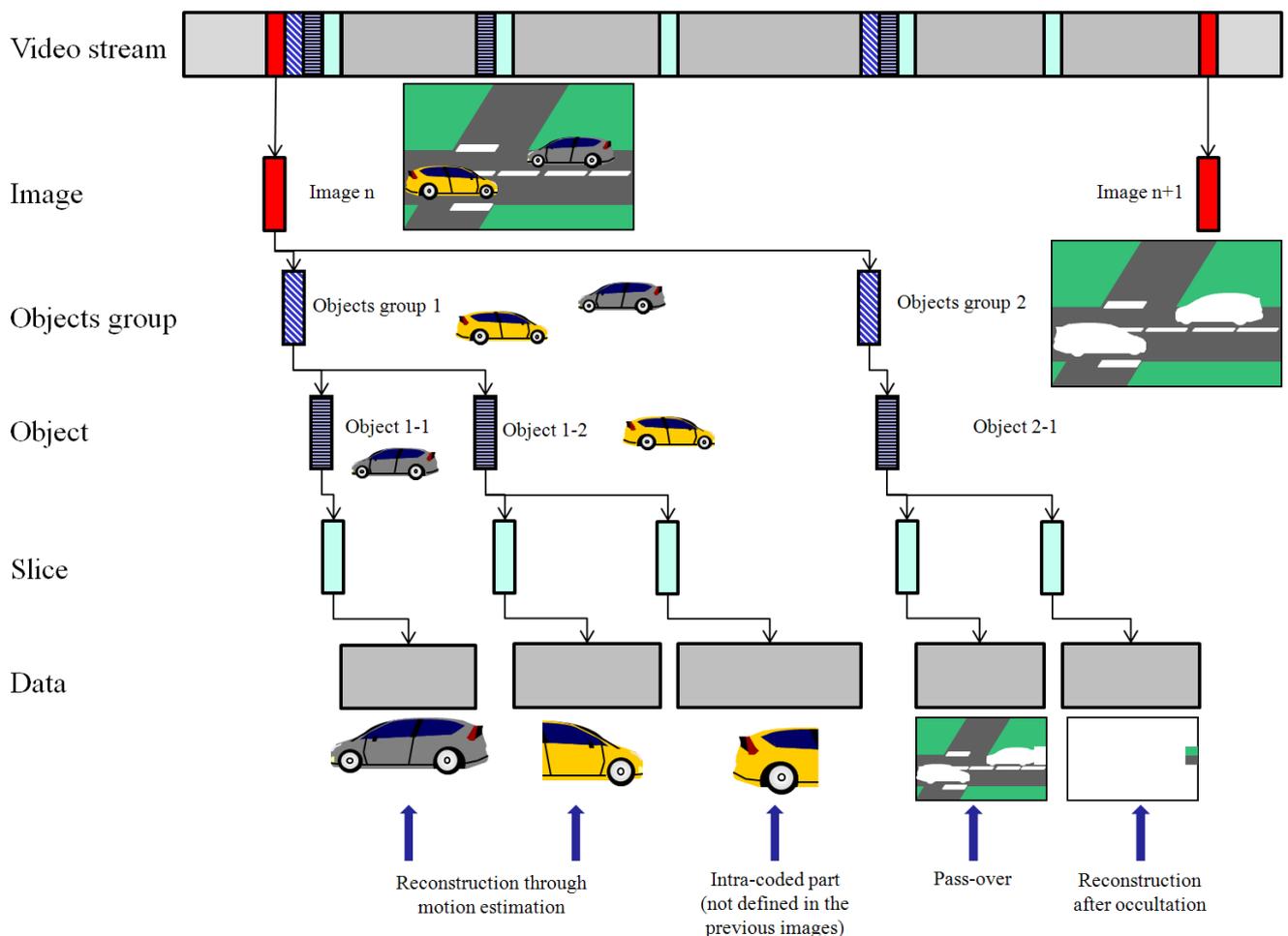


Fig. 4 Structure tree.

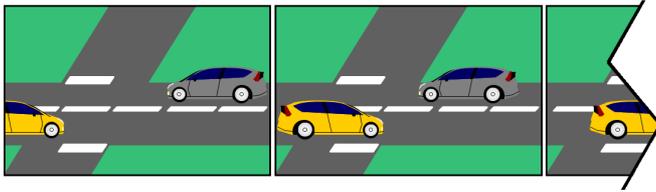


Fig. 6 Sequence (the yellow car is getting inside the frame from the left, while the grey one goes from the right to the left).

The tree shown in figure 4 corresponds to the image in the middle of figure 6 and to the stream of figure 5. Object 1-2 is made of two slices as the front of the car can be reconstructed by motion compensation while the rear – not present in the previous frame – is intra-coded. The same phenomenon can be seen for object 2-1, for which the previously occluded background is reconstructed differently from the rest of it.

3) *Encoding, compression and tools.* To allow a totally independent reconstruction of the objects, these have to be reconstructed without pointing to one another. Therefore, the motion compensation is restricted inside an object. If the objects are defined precisely enough, the compression impact of such a constraint is minimal: there is no need to point out of an object to obtain the matching blocks. This aspect requires the prediction of each object position inside an image before the image is compressed. It can be achieved by a tracking solution combined with a matching algorithm. For example a Kalman filter will predict the expected position of the objects detected from the previous images to the new one. A refinement, through background subtraction for example, will improve the shapes and identify new objects inside the image. The object declaration and position will be identified through an MBAMap similar to the one in H.264 and will be contained inside the image header. The map itself can be compressed by motion compensation, allowing a minimal impact on compression ratios. The encoder/decoder will have to take into account the dynamic changes of the MBAMap.

Using this slice definition, we also introduce the notion of *pass-over*. With H.264, a skipped macroblock will be reconstructed considering the motion prediction of the neighbouring blocks. To recopy exactly the same blocks as the reference image, a motion vector equal to zero has to be coded. The pass-over corresponds to this null vector, providing a shorter syntax combined with the possibility of multi-reference, which we defined as multi-reference pass-over. In many fields as in videosurveillance, objects move in front of an immobile background. The area that was occluded can now be reconstructed from any previous image where it was defined, as illustrated by figure 7. When several objects are moving at the same time, the multi-reference allows the pass-over to point to a different frame for each previously occluded slice. The slices as defined above also optimise the approach, as a single pass-over reference can be provided for the slice. There is no need for a macroblock per macroblock declaration as in H.264.

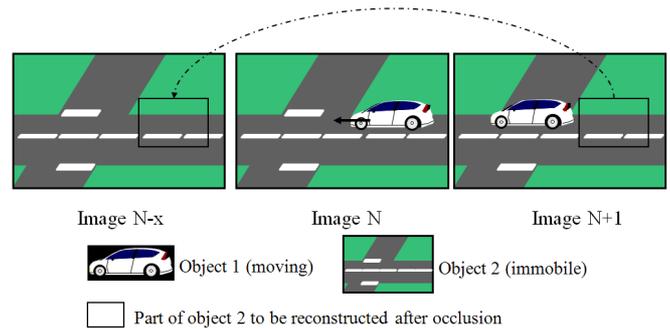


Fig. 7 Multi-reference pass-over.

4) *Advantages.* The proposed image hierarchy allows an independent object decoding and therefore display without any reconstruction problem. The non-decoded objects will leave flat areas. This can be optimum depending on the targeted applications: for instance, only vehicles are required for automatic registration plate identification; only the background and people inside a sub-portion of a video have to be decoded if the user zoomed inside a sequence.

The new structure was also designed to provide a new multimedia access. It is possible to directly access a given level of hierarchy, i.e. to the parts of the sequence considered as significant, by navigating quickly inside the image / objects group / object headers. This navigation can be achieved through several approaches. If the user knows what he is looking for (e.g. a red object), he will instantiate the decoding process specifying a decoding process limited to the corresponding objects. The navigation can also be interactive: a parsing process gets the descriptors inside the headers and the user specifies the ones he wants to be taken into account. The interactive decoding process can also be done by point and click selection on the sequence to indicate which parts of the sequence the user wants to go on visualising.

Having the descriptors defined inside the headers also provide a powerful tool for indexation, therefore for requests, from a unique stream. The metadata are coded inside the compressed video and do not require a dedicated encoding and/or transmission. The process implementation makes it possible to use these data without having to include other tools. The overall structure allows a semantic navigation, based on the definition of a hierarchy or an object category, instead of the previous possibilities which were restricted to spatial or temporal sub-portions of a sequence.

If a sequence is coded using for each image only one object group containing one object made of one slice, the compression process is similar to the one obtained via the current standards, which implies a comparable compression ratio.

III. H.264 IMPLEMENTATION

With new investigations led by the MPEG experts group as MPEG-A Surveillance Multimedia Application Format (SMAF), a compression standard dedicated to or at least optimised for videosurveillance might appear. If the suggested structure might later be proposed for standardisation, the cur-

rent ones made us design a specific H.264 encoder/decoder that tends to the hierarchy provided some restrictions.

A. Preliminary remarks

As exposed before, one of the first problems we have to face here is the dynamic slicing compatibility. The H.264 standard does not describe or require a specific implementation concerning this point. The only detailed fact is that for an MBAmapping update, a Picture Parameters Set unit has to be sent first. Usually the PPS is used for quantification parameters, the number of reference frames, the weighted bi-prediction etc. Considering this, most decoders expect an Intra coded image after a PPS, even if the standard does not specify it. Our use of the PPS and dynamic MBAmapping being quite unusual or even unexpected, most of the decoders will not be able to process the sequence. It is yet possible to design a fully compliant decoder taking into account these specific sequences.

We restrict the detailed image structure to fit the possibilities provided by H.264, as shown in Table I. The *image* level remains as it was, and a new image is identified by the frame number (*framenum*) index. The *objects group* level is not used. We use the *slices group* to define the *object* level, inside which the slices we suggested are replaced by the H.264 slices definition.

TABLE I
HIERARCHY EQUIVALENT IN H.264

New approach	H.264
Image	Image (access unit)
Objects group	None
Object	Slices group
Slice / object Part	Slice

Many tools or advantages described before are therefore not available or not as powerful as expected when restricted to H.264. The pass-over without the new slice definition is discarded and replaced by the current macroblock per macroblock declaration of the standard. The hierarchical structure combined with the descriptors cannot be inserted directly inside the compressed video stream. If provided, these elements will have to be inserted inside dedicated Network Abstraction Layer (NAL) units, preferably considering the types 30 and 31 (*undefined*, and not used by the RTP protocol).

H.264 restrictions dealing with slices group also imply using the *baseline* and *extended* profiles. The *main* profile only allows one slices group, making the expected stream definition worthless. The *baseline* and *extended* ones can provide up to height slices groups, which is still restrictive considering the semantic use of these, but can however deal with seven mobile objects in the scene at the same time (the eighth slices group being allocated to the background).

B. Encoding process

Dealing with videosurveillance, there is no defined start of a sequence, which discards the initialisation problems. However,

a one Group of Pictures (GoP) initialisation time is sufficient to start the following procedures.

1) *Detecting the moving objects.* This task can be achieved through many algorithms. However to maintain a fast computing process, as well as low CPU time and memory needs, we suggest to use analysis in the compressed domain. This approach, as detailed in [1.2.5.6.7.8], exploits the information contained in the transform coefficients (integer transform for H.264) and/or the motion vectors to detect the mobile objects inside the compressed video stream. This approach allows an object segmentation function analysing up to 500 images per second (720x576 4:2:0 on a 2.66 GHz core). The detection of the moving objects is achieved on an image per image basis, which is relevant considering videosurveillance uses Intra and Predicted frames to avoid the delay generated by Bidirectional frames. Then a Munkres matching algorithm combined with a Kalman filter are run to follow the object trajectory over several frames and to predict the position of each object inside the frame to be encoded.

Some criticism could be objected concerning the lower segmentation resolution implied by the analysis in the compressed domain. Intrinsically these algorithms provide a block or macroblock sensitivity; however refinements can be achieved to obtain a pixel precision, as described in [1]. Anyhow, we need this segmentation to define slices group, which means only a macroblock resolution is relevant.

2) *Dynamic slicing and compression of the current image.* We now have one image to compress and we know where the mobile objects are and their shape at a macroblock resolution. The next step is to define the slices groups and the MBAmapping. To be as close as possible to the new image hierarchy, we associate one slice group with one object, but if more than seven mobile objects are present in the image, it is possible to have one slices group containing two or more objects. In this case, the descriptors will be used to group preferably objects having the highest similarity (e.g. cars together). The additional NAL units can be used to store the descriptors obtained from the analysis in the compressed domain.

The compression is then performed almost normally: for each object the H.264 standard compression is achieved, but is restricted inside the object only, using its occurrences inside previous images matched by the algorithm which was used to predict the object position. Considering the object segmentation is precise enough, this particularity will not affect much the compression ratio as one object can usually be predicted from itself in the reference images. It also tends to the independent object coding inside a unique stream expected from the suggested image hierarchy.

C. Resulting stream

The compressed video obtained is H.264 compliant, although using a dynamic slices change which might not be taken into account by any decoder. Yet the background is coded independently from the moving objects. These can also be accessed one by one, provided no more than seven are presents in the current image. If more are inside the same image, they can be accessed group by group, each providing objects

with the highest similarity found during the compression process.

As we suggested when defining the image description hierarchy, this new structure can be used to decode independently the different objects inside a videosurveillance stream, but it can also make many other applications possible.

IV. SELECTIVE VISUAL ENCRYPTION

When dealing with videosurveillance, a recurrent problem is linked to the privacy of the people. Streams can be intercepted, people waiting for a granted access to a company in a security office can overlook the monitors, etc. Some work focused on ciphering H.264 videos, and now we propose to enhance them with selective encryption.

A. H.264 encryption

The underlying idea is to provide a tool that can create a visually ciphered video which is compliant with any H.264 decoder. C. Bergeron in [3] described a method based on AES [4] to obtain such a results by analysing which bits of the Variable Length Code (VLC) words inside a H.264 stream can be altered without denaturing the stream. Knowing these bits, an encryption key can be applied only on these, resulting in the expected ciphered sequence.

In our case, we only want to cipher mobile object inside a stream, making it possible to understand the scene without the decryption key and maintaining privacy over people and vehicles. Without all the above dedicated implementation of an H.264 encoder, it would be possible to identify the moving objects and cipher them with no more attention. The problem would in this case come from the prediction which is not restricted inside each object. The Intra-coded images would display more or less the same results as the one we claim, but the further the image is from an Intra-coded one, the more spread the encryption will be, because of the intra-prediction used in each frame in H.264, altering the whole field of view.

B. H.264 selective encryption

Considering the image hierarchy description and the H.264 implementation described in sections II and III, we can now only apply the H.264 ciphering on the slices groups dedicated to mobile objects. Objects being independently coded inside the stream, there is no ciphered zone spreading inconvenient. The 30 and 31 NAL units, a priori containing the descriptors, will also be used to identify the slices groups which were ciphered so that the decoder can select on which part of the image to apply the decryption key.

The difference between the two encryption processes is illustrated by figure 8, while the overall process is shown in figure 10.

The proposed scheme works as well on videos where objects are small (wide angle camera in videosurveillance) or when they occupy most of the field of view, in case the result is close to the one encoding the whole sequence as in [3]. However, to exploits the possibilities offered by this selective

encryption, small objects allow contextual identification, as shown in Figure 9.

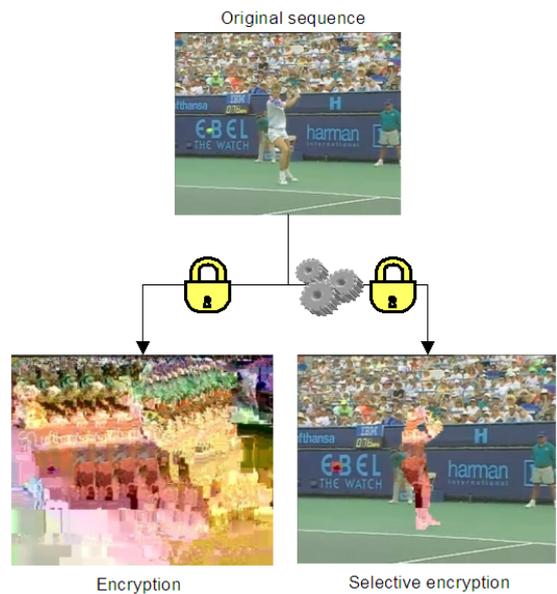


Fig. 8 Selectivity improvement.



Fig. 9 Results in videosurveillance (Top: Speedway sequence, Bottom: Caretaker [11] Corpus).

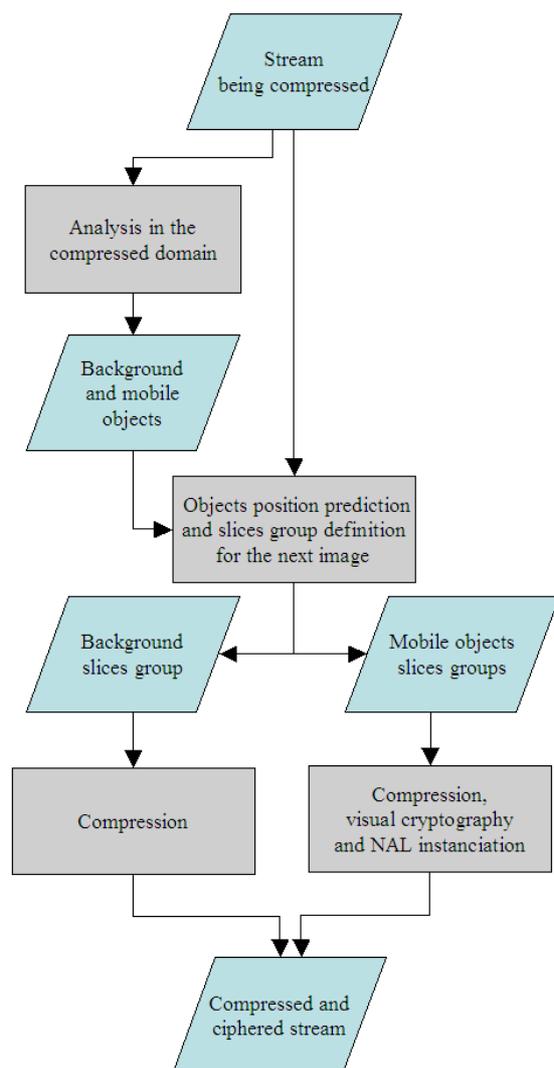


Fig. 10 Selective visual encryption algorithm.

V. CONCLUSIONS AND PERSPECTIVES

Many applications require a semantic knowledge of a video sequence, mainly the ones based on requests, as network research, videosurveillance, television archive centre, etc. The current video standards do not allow these tasks directly on a single stream. By not being able to organise semantically the data inside the sequence, it is impossible to directly apply functions adapted to different image areas. Aiming the regions considered as pertinent from the application focus allows processes such as encryption, watermarking or protection through redundancy. With the current standards, the most effective generic approach will decompress the image, determine the areas to be processed, apply the process, and finally recompress the image. The CPU and/or memory resources for such a result can become prohibitive when not impossible considering hi-definition videos are becoming a usual medium.

In this context, we suggested a new image definition hierarchy, which supports a full access to semantic objects inside a

video sequence by coding them independently in a single stream. As this structure cannot be implemented respecting the current standards, we also explained a restriction to this definition making it available and compliant with H.264. A first application we presented using this method uses these specific streams to select the mobile objects and encrypt them to enhance videosurveillance sequences with privacy respect. Our future work will focus on implementing this H.264 restriction of the new hierarchy. We will measure the results and validate our approach experimentally. Considering the compression impact, our implementation will be compared to the MPEG-4 scene representation using H.264 sub-streams, as well as unique H.264 streams using the "box-out" and "foreground with left-over" slice group mapping. Considering the descriptor aspect of the structure, we will test the results dealing with recall on request and partial decoding of a selected object.

As many applications become possible through this structure, we will also investigate the range of its perspectives and if needed detail the sub-parts that might require a extended specification.

ACKNOWLEDGMENT

The authors would like to thank Erwann RENAN and Didier NICHOLSON who worked on the new image description hierarchy and its applications for Thales Communications

REFERENCES

- [1] Wei Zeng, Jun Du, Wen Gao, Qingming Huang, "Robust moving object segmentation on H.264/AVC compressed video using the block-based MRF model", *Real-Time Imaging*, v.11 n.4, pp.290-299, August 2005
- [2] Zhi Liu, Yu Lu, Zhaoyang Zhang, "Real-time spatiotemporal segmentation of video objects in the H.264 compressed domain", *Journal of Visual Communication and Image Representation archive*, Volume 18, Issue 3, pp.275-290, June 2007
- [3] C. Bergeron, C. Lamy-Bergot, "Compliant selective encryption for H.264/AVC video streams", *Proc. of the International Workshop on Multimedia Processing (MMSP'05)*, pp. 477-480, Shanghai, China, Oct-Nov 2005.
- [4] *Advanced Encryption Standard*, National Institute of Standards and Technology, FIPS-197, November 2001.
- [5] M. Leny, D. Nicholson, F. Prêteux, "De l'estimation de mouvement pour l'analyse temps réel de vidéos dans le domaine compressé", *Proc. GRETSI*, pp. 1173-1176, Troyes, France, Sept 2007.
- [6] M. Leny, F. Prêteux, D. Nicholson, "Statistical motion vector analysis for object tracking in compressed video streams", *Proceedings of SPIE Electronic Imaging*, Volume 6812, San Jose, USA, 2008.
- [7] M. Coimbra, "Compressed Domain Video Processing with Applications to Surveillance", *Ph.D. Thesis*, Department of Electronic Engineering, Queen Mary, University of London, 2004.
- [8] Lan Dong; Zoghلامي, I.; Schwartz, S.C., "Object tracking in compressed video with confidence measure", *Proc. International Conference on Multimedia and Expo (ICME 06)*, pp. 753-756, Toronto, Canada, July 2006.
- [9] ITU-T Recommendation H.264 and ISO/IEC 14496-10AVC, "Advanced video coding for generic audiovisual services," version 3: 2005.
- [10] Yong Wang; Jae-Gon Kim; Shih-Fu Chang; Hyung-Myung Kim, "Utility-Based Video Adaptation for Universal Multimedia Access (UMA) and Content-Based Utility Function Prediction for Real-Time Video Transcoding", *IEEE Transactions on Multimedia*, Volume 9, Issue 2, pp. 213-220, Feb. 2007.
- [11] Caretaker: IST European project - Content Analysis and Retrieval Technologies to Apply Knowledge Extraction to massive Recording - <http://www.ist-caretaker.org>

Power Consumption Model at Functional Level for VLIW Digital Signal Processors

Mostafa E. A. Ibrahim⁽¹⁺²⁾, Markus Rupp⁽¹⁾, and S. E.-D. Habib⁽²⁾

⁽¹⁾Institute of Communications and RF Engineering-Vienna University of Technology, Austria

⁽²⁾ Electronics and Communication Department Faculty of Engineering-Cairo University, Egypt

Email: {mhalas,mrupp}@nt.tuwien.ac.at, seraged@ieee.org

Abstract—In this contribution the modeling of power consumption for the VLIW processor TMS320C6416T is presented taking into account typical software algorithms in signal processing. The modeling is performed at the functional level making this approach distinctly different from other modeling approaches in low level technique. This means that the power consumption can be identified at an early stage in the design process, enabling the designer to explore different hardware architectures and algorithms. Some typical signal processing algorithms are used for the purpose of validating the proposed model. The estimated power consumption is compared to the physically measured power consumption, achieving a very low resulting average estimation error of 1.75% and a maximum estimation error of only 3.6%.

I. INTRODUCTION

Many applications in special areas such as hand-held computation, tiny robots, and guidance systems in automated vehicles are powered by batteries of low rating. In order to avoid frequent recharging or replacement of the batteries, there is significant interest in low-power system design. Very Long Instruction Word (VLIW) Digital Signal Processors (DSP) are the most worthy choice for such an application domain because of their optimal performance at low power [1], [2].

The importance of the power constraints during the design of embedded systems has continuously increased in the past years, due to technological trends toward high-level integration and increasing operating frequencies, combined with the growing demand of portable systems. This has led to a significant research effort in power estimation and low power design [3].

Power measurement tools are available only for the lower levels of the design, at the circuit level and to a limited extent at the logic level. These tools are very slow and impractical to use to evaluate the power consumption of embedded software since the application power consumption would only be known at the very last stage of the design process.

In this paper, an approach for modeling the power consumption of a VLIW DSP, from the software point of view, is presented. The contribution of this work aims to precisely estimate the power consumption of the core processor while running a software algorithm at an early stage in the design process. The targeted DSP is the TMS320C6416T (for the rest of the paper it is referred to as C6416T for brevity) from Texas

Instrument. This processor features the highest-performance among the fixed-point DSPs of the C6000 DSP platforms.

The rest of the paper is organised as follows: Section II presents an overview of several existing power consumption modeling techniques for general purpose processors. A general overview of the target architecture is presented in Section III. It is followed by a detailed description of the functional level analysis for the targeted architecture in Section IV. The proposed power consumption model is verified in Section V and the reliability of the estimation is demonstrated. Finally, Section VI summarizes the main contributions of this paper.

II. RELATED WORK

Recent approaches to model the power consumption of DSPs can be separated into two main categories: hardware level models and instruction level models. Hardware level models calculate power and energy from detailed electrical descriptions, comprising circuit level, gate level, register transfer (RT) level or system level. Instruction level models deal only with instructions and functional units from the software point of view and without electrical knowledge of the underlying architecture [4].

Traditional methodologies perform power estimation at low abstraction levels such as circuit, gate or RT level [5], [6], micro-architectural-level simulation [7], [8], [9]. While providing excellent accuracy; these methodologies are slow and impractical for analyzing the power consumption at an early design stage. Moreover, these methodologies require the availability of lower level circuit details or a complete Hardware Description Language (HDL) design of the targeted processor, which is not available for most of commercial off-the-shelf processors [9]. Several instruction level estimation models have been proposed. These can be classified into Instruction Level Power Analysis (ILPA) and Functional Level Power Analysis (FLPA).

An instruction level power model for individual processors was first proposed by V. Tiwari [10]. By measuring the current drawn by the processor as it repeatedly executes distinct instructions or distinct instruction sequences, it is possible to obtain most of the information that is required to evaluate the power consumption of a program for the processor under test [10]. Power is modeled as a base cost for each instruction plus a circuit state overhead that depends on neighboring instructions. The base cost of an instruction can be considered

This work has been funded by the Christian Doppler Laboratory for Design Methodology of Signal Processing Algorithms, as well as the comet funded the K-project: Embedded Computer Vision.

as the cost associated with the basic processing needed to execute the instruction. An experimental method is proposed by the authors of [10] to empirically determine the base and the circuit overhead costs. In this experimental method, a program containing an infinite loop consisting of several instances of the given instruction is used. The average current drawn by the processor core during the execution of this loop is measured by a standard off-the-shelf, dual-slope integrating digital multimeter.

Much more accurate measuring environments have been proposed to precisely monitor the instantaneous current drawn by the processor instead of the average current. One of these approaches has used a high-performance current mirror, based on bipolar junction transistors as current sensing circuit [11]. Another approach, to reduce the spatial complexity of instruction-level power models, is presented in [12]. Therein, inter-instruction effects have been measured by considering only the additional energy consumption observed when a generic instruction is executed after a No Operation (NOP) instruction.

The ILPA based methods exhibit usually a small margin of error, typically 2 to 4 percent. However, these methods have some drawbacks. One of these drawbacks is that the number of current measurements is directly related to the number of instructions in the Instruction Set Architecture (ISA), and also the number of parallel instructions composing the very long instruction in the VLIW processor. The problem of instruction level power characterization of K-issue VLIW processor is $O(N^{2K})$ where N is the number of instructions in the ISA and K is number of parallel instructions composing the VLIW [13]. Also they do not provide any insight on the instantaneous causes of power consumption within the processor core, which is seen as a black-box model. FLPA was first introduced by J. Laurent et al. in [14].

The basic idea behind the FLPA is the distinction of the processor architecture into functional blocks like Processing Unit (PU), Instruction Management Unit (IMU), internal memory and others [14]. At first, a functional analysis of these blocks is performed to specify and then discard the non-consuming blocks (those with negligible impact on the power consumption). The second step is to figure out the parameters that affect the power consumption of each of the power consuming blocks. For instance, the IMU is affected by the instructions dispatching rate which in turn is related to the parallelism degree. In addition to these parameters, there are some parameters that affect the power consumption of all functional blocks in the same manner such as operating frequency and word length of input data [15].

By means of simulations or measurements it is possible to find an arithmetic function for each block that determines its power consumption depending on a set of parameters. For the determination of these arithmetic functions for each functional block, the average supply current of the processor core is measured in relation with the variation of each parameter. These variations are achieved by a set of small programs, called scenarios. Such scenarios are short programs written

in assembly language and consisting of unbounded loops with a body of several hundreds of certain instructions that individually invoke each block. The power consumption rules are finally obtained by curve-fitting the measurements values [15].

The parameters that affect the power consumption for each functional block can be extracted from the assembly code generated by the Integrated Development Environment (IDE). Some parameters cannot be extracted directly from the assembly code, such as the execution time and the data cache miss rate. Therefore, at least one simulation is required to obtain these parameters with the aid of the profiler.

The functional level power modeling approach is applicable to all types of processor architectures. Furthermore, FLPA-modeling can be applied to a processor with moderate effort and no detailed knowledge of the processors architecture is necessary [16].

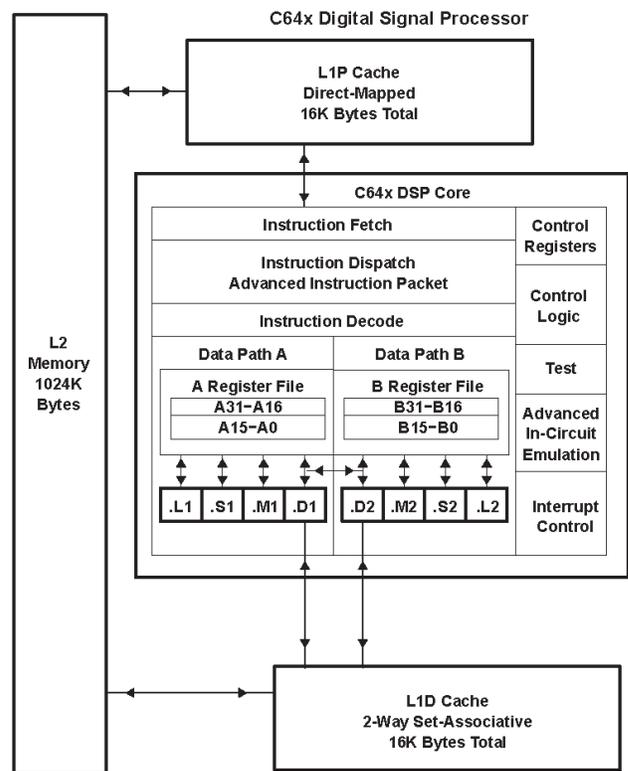


Fig. 1. C6416T block diagram.

III. TARGET ARCHITECTURE

A block diagram of the C6416T CPU is shown in Fig. 1. The CPU contains a program fetch unit, an instruction dispatch unit, an instruction decode unit, two data paths each of four functional units, as well as 64 32-bit registers. The program fetch, instruction dispatch, and instruction decode units can deliver up to eight 32-bit instructions to the functional units every CPU clock cycle. The processing of the instructions occurs in each of the two data paths (A and B). The CPU also has a 32-bit, byte-addressable address space and a 256 bit read-only port to access internal program memory as well

as two 256-bit ports (read and write) to access internal data memory. However, the internal L2 memory is unified for data and program, the L1 memory is organized into separate data and program caches [17].

This DSP is considered as a complex processor architecture since it features a deep pipeline (11 stages) and can execute up to eight parallel instructions per cycle.

IV. MODELING METHODOLOGY FOR C6416T

After applying the FLPA, the C6416T architecture is subdivided into six distinct functional blocks (clock tree, instruction management unit, processing unit, internal memory, L1 data cache and L1 program cache) as shown in Fig. 2. The parameters that affect the power consumption for the determined functional blocks are also shown in Fig. 2. The C6416T fetches instructions from memory in fixed bundles of 8 instructions, known as fetch packets. The instructions are decoded and separated into bundles of parallel-issue instructions known as execute packets.

The dispatching rate α represents the average number of execution packets per fetch packet. The processing rate β stands for the average number of active processing units per cycle. The internal memory read/write access rates express the number of memory accesses divided by the number of required clock cycles for executing the code segment under investigation. Finally the data cache miss rate λ corresponds to the number of data cache misses divided by the total memory accesses.

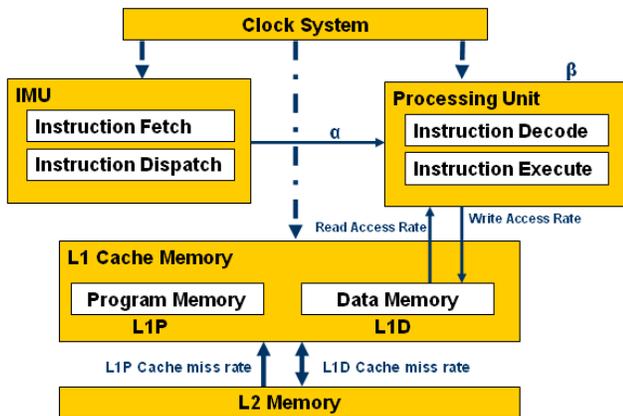


Fig. 2. Functional level power analysis for C6416T.

A. Experimental setup

In our setup, the DSP core voltage is 1.2 V and the operating frequency ranges from 600 MHz to 1200 MHz. The arithmetic functions in the following sections IV-B, IV-C, IV-D, describe the current, drawn by the DSP core at operating frequency of 1000 MHz. All measurements are carried out on the DSP Starter Kit (DSK) of the C6416T manufactured by Spectrum Digital Inc. There are three power test points on this DSK for DSP I/O current, DSP core current and system current. The Code Composer Studio (CCS3.1) from Texas Instruments is used as the IDE.

Several assembly language scenarios have been developed to separately stimulate each of the functional blocks. All scenarios consist of unbounded loops with a body of more than 1000 instructions, to avoid the effect of branching instructions on the measured current. First of all, the effect of the operating frequency on the power consumption is determined. The operating frequency linearly affects the current drawn by the DSP core and hence, also linearly affects the power consumption of the processor. Figure 3 shows the relation between the operating frequency and the current drawn by the DSP core.

For demonstration purposes the process of determining the power consumption rules for IMU, PU and L1 data cache functional blocks is presented.

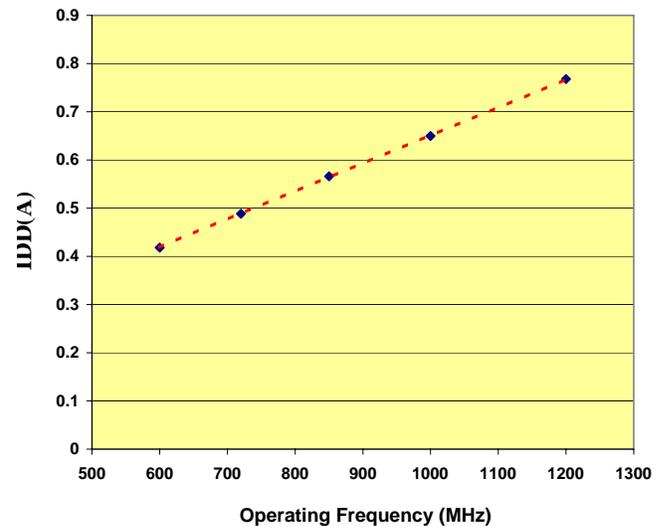


Fig. 3. Model function of the C6416T clock tree.

B. IMU power consumption model

The IMU unit consists of two main sub-units which are the instructions fetching unit and the dispatching unit. The C6416T processor fetches eight instructions per cycle as one fetch packet. The dispatch unit then subdivides this fetch packet into execution packets. Since the C6416T has eight functional units, it is capable of simultaneously executing up to eight instructions. Consequently, the dispatch unit can divide the fetch packet into one (maximum parallelism) to eight (sequential) execution packets. Therefore, it is obvious that the dispatch rate is the only parameter that affects the power consumption of the IMU.

The proposed scenario should not invoke any other functional unit but the IMU. Hence, the scenario is composed of an unbounded loop with more than 1000 NOPs. As the NOP instruction does not require any processing unit for its execution. The scenario varies the dispatch rate (number of fetch packets / number of execution packets) from 0.125 to 1.0.

Figure 4 indicates the characteristics of the current drawn by the core processor with a varying dispatch rate. By curve fitting the measurement values in Fig. 4 the arithmetical function in (1) is obtained.

$$IDD_{IMU} = -0.0918\alpha^2 + 0.284\alpha + 0.0603 \quad (1)$$

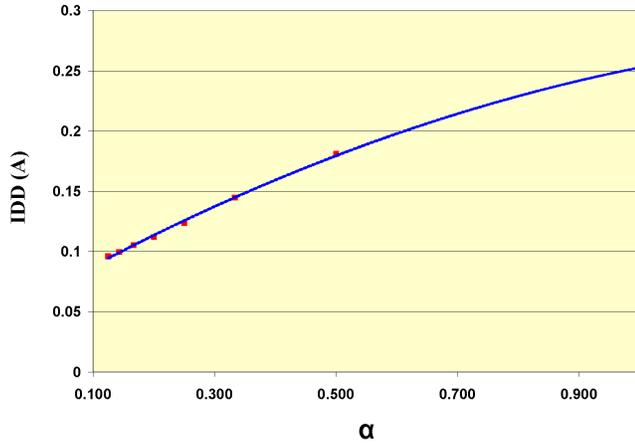


Fig. 4. Measured IMU current values vs. model function of (1).

The quality of the fitting process is measured by the value R-squared (R^2): A number from 0 to 1, which is the square of the residuals of the data after the fit. This value expresses what fraction of the variance of the data is explained by the fitted trend line. It reveals how closely the estimated values for the trend line correspond to the actual data. A trend line is most reliable when its R^2 value is at or close to 1.0 [18]. Since the R^2 value for the arithmetic function in (1) equals 0.9994 that means (1) is an excellent fit for the curve values in Fig. 4.

The arithmetic function in (1) does not consider the effect of the pipeline stalls. Many reasons cause the pipeline to stall. For instance, one data cache miss stalls the pipeline for at least 6 cycles. Hence, the arithmetic function in (2) is presented to account for the pipeline stall effect.

$$IDD_{IMU} = (-0.0918\alpha^2 + 0.284\alpha + 0.0603)(1 - PSR) \quad (2)$$

where PSR stands for pipeline stall rate which can be expressed as the number of pipeline stall cycles divided by the total cycles required for executing the code segment under investigation.

C. PU power consumption model

The data path of the C6416T consists of eight functional units. These functional units can work simultaneously, if the dispatch unit succeeds to compose an execution packet with eight instructions. The dispatch rate can be used as the affecting parameter for the PU power consumption. But, the fact that the NOP does not require any PU for its execution convinced us that another parameter yields a better description of the PUs. The new parameter is the processing unit rate which expresses the average number of active processing units per cycle. Another important parameter that affects the processing unit power consumption is the word length of the data operands. In the C6416T the word length varies from 8 bits to 32 bits. Thus, in our model 16 bit word length has been chosen to be the typical word length.

More than 1000 different instructions compose the scenario that varies the processing unit rate. That is to account for the inter-instructions effect. The current measured from the DSK

is the sum of the clock tree, IMU, and the PU currents. Headed for attaining the current drawn by only the PU, the IMU and clock tree currents are subtracted from the measured current.

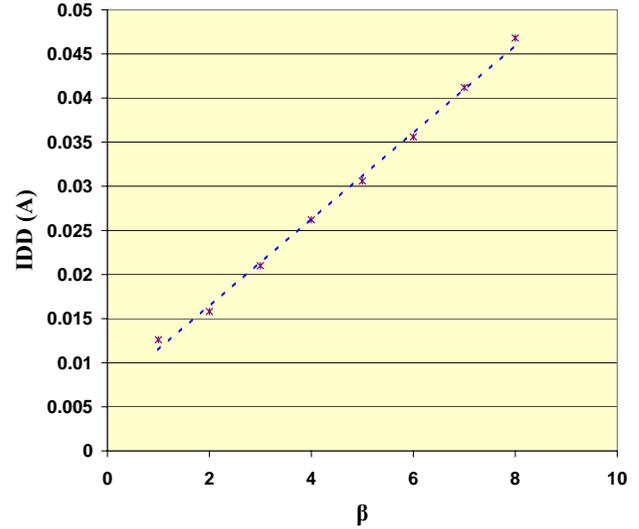


Fig. 5. Measured PU current values vs. model function of (3).

Figure 5 depicts the effect of varying the number of active PU per cycle on the current drawn by the core processor. The arithmetic function that best fit the curve in Fig. 5 is a linear equation as shown in (3)

$$IDD_{PU} = (-0.0049\beta + 0.0065)(1 - PSR) \quad (3)$$

The arithmetic function in (3) resulted in R^2 value of 0.9982 that means it is the best fit for the curve values in Fig. 5.

Compared to other functional units such as clock tree or the IMU, it is clear that the PU does not significantly contribute to the total power consumption of the core processor. It is important to mention that the scenario for invoking the PU does not include any memory instructions. The internal memory operations are handled in a separate scenario.

D. L1 data cache power consumption model

The L1 data cache functional block represents the flow of data from the L1 data cache to L2 memory and vice versa. Different scenarios are prepared to stimulate the effect of the data cache miss.

The data cache miss rate is used as the affecting parameter for the L1 data cache functional block. Taking into account the fact that L1 data cache is two-way associative cache, a scenario that varies the number of data cache misses per a fixed number of memory accesses has been developed. In this scenario, arbitrary data are pre-loaded into both blocks of set 0. To force a data cache miss, data from certain addresses in the L2 memory, which must be mapped into set 0 blocks, are loaded to L1 data cache. The addresses of the new data to be loaded are different from those already in set 0. Hence, a data caches miss occurs.

Figure 6 shows the effect of varying the data cache miss rate on the current drawn by the core processor. The best arithmetic

function that fit the measured values in Fig. 6 is obtained as indicated in (4) with R^2 value of 0.9909.

$$IDD_{L1D} = (-2 \cdot 10^{-5} \lambda^2 + 0.0041 \lambda)(1 - PSR) \quad (4)$$

The arithmetic function in (4) is a quadratic-polynomial. This arithmetic function differs from the corresponding linear function that was proposed in [16] for the cache functional block. The squared-function yields better description for the L1 data cache block due to the fact that L1 data cache pipelines the cache misses, to decrease the resulting pipeline stalls. The proposed model in [15] did not separately investigate the effect of data cache misses instead it is included in the processing unit functional block.

More details regarding the way in which the functional blocks were stimulated can be found in [19].

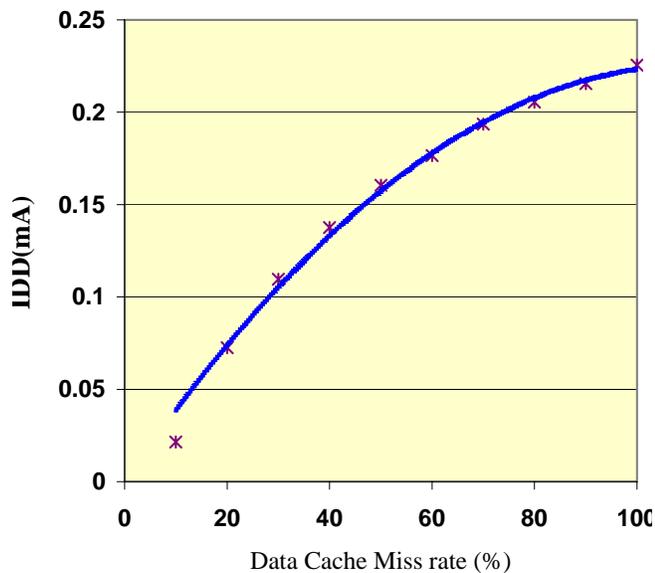


Fig. 6. Measured L1 data cache current values vs. model function of (4).

V. VALIDATION

For purpose of validating the proposed power consumption model of the VLIW processors the following benchmarks examples are presented. A finite impulse response (FIR) filter that uses 240 coefficients is a common signal processing benchmark. The FIR benchmark is found in the Texas Instruments signal processing library for C64x processor family. The dot product for two arrays of length 128 and input data of 16-bits is also one of the typical signal processing operations. Two more benchmarks are used from the image processing library of Texas Instruments the Sobel filter of window 3×3 and the image threshold are applied for an input image of size 256×32 (Columns \times Rows). The input data for all the previously mentioned benchmarks are located in the internal data memory.

First of all, all optimization options which are included in the CCS3.1 are turned off because these optimization options affect the speed or the code size only and are not dedicated to power optimization. The second step is to compile the

benchmarks. From the generated assembly files the required parameters for the model are calculated with the aid of the CCS3.1 profiler for the parameters that can not be estimated statically such as the data cache miss rate. For instance, the processing unit rate which is defined as the average number of active processing units per cycle is calculated from the assembly code. The parameter β is the result of dividing the number of processing units (equals the number of instructions excluding the NOP) by the number of cycles per code iteration.

Figure 7 presents the result of the estimated power consumption versus the measured one for the above mentioned benchmarks. The average estimation error is 1.75% and in the worst case is 3.6%.

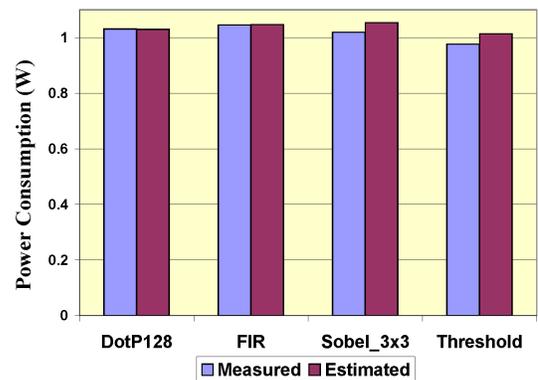


Fig. 7. Estimated vs. Measured power consumption of the C6416.

VI. CONCLUSION

In the presented paper different power consumption approaches that are applied on various levels of abstraction have been recapitulated. A functional level power analysis technique has been applied to the commercial off-the-shelf VLIW processor C6416T. The processor architecture has been divided into several functional blocks. The parameters that affect the power consumption of each functional block have been determined. These parameters have been calculated from the generated assembly code of the IDE. The inter-instructions as well as the pipeline stall effects have been investigated in our proposed model. The power consumption has been estimated for several signal and image processing benchmarks. The estimated power consumption is compared with the physically measured power consumption and a very low resulting average error of 1.75% is realized. A maximum estimation error of only 3.6% is achieved. There are some open issues to be studied in the future, for example, the effect of parameters estimation quality on the power estimation using the proposed model.

REFERENCES

- [1] N. Zafar Azeemi and M. Rupp. Multicriteria low energy source level optimization of embedded programs. In *Tagungsband zur Informationstagung Mikroelektronik 06 IEEE Austria*, pages 150–158, Vienna, Austria, October 2006.
- [2] N. Zafar Azeemi and M. Rupp. Energy-aware source-to-source transformations for a VLIW DSP processor. In *The 17th ICM 05*, pages 133–138, Islamabad, Pakistan, December 2005.

- [3] William Fornaciari, Paolo Gubian, Donatella Sciuto, and Cristina Silvano. Power estimation of embedded systems: A hardware/software codesign approach. *IEEE Trans. Very Large Scale Integr. Syst.*, 6(2):266–275, 1998.
- [4] Chris J. Bleakley, Miguel Casas-Sanchez, and Jose Rizo-Morente. Software level power consumption models and power saving techniques for embedded dsp processors. *Journal of Low Power Electronics*, 2(2):281–290, 2006.
- [5] T. Chou and K. Roy. Accurate estimation of power dissipation in CMOS sequential circuits. *IEEE Trans. Very Large Scale Integr. Syst.*, 4:369–380, September 1996.
- [6] Charlie X. Huang, Bill Zhang, An-Chang Deng, and Burkhard Swirski. The design and implementation of PowerMill. In *ISLPED '95: Proceedings of the 1995 international symposium on Low power design*, pages 105–110, New York, NY, USA, 1995. ACM.
- [7] Ping-Wen Ong and Ran-Hong Yan. Power-conscious software design-a framework for modeling software on hardware. *Journal of Low Power Electronics, IEEE Symposium*, pages 36–37, October 1994.
- [8] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The design and use of SimplePower: a cycle-accurate energy estimation tool. In *DAC'2000: Proceedings of the 37th conference on Design automation*, pages 340–345, New York, NY, USA, 2000. ACM.
- [9] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. *SIGARCH Comput. Archit. News*, 28(2):83–94, 2000.
- [10] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Power analysis of embedded software: a first step towards software power minimization. *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers.*, pages 384–390, November 1994.
- [11] S. Nikolaidis, N. Kavvadias, P. Neofotistos, K. Kosmatopoulos, T. Laopoulos, and L. Bisdounis. Instrumentation set-up for instruction level power modeling. In *PATMOS '02: Proceedings of the 12th International Workshop on Integrated Circuit Design. Power and Timing Modeling, Optimization and Simulation*, pages 71–80, London, UK, 2002. Springer-Verlag.
- [12] B. Klass, D. E. Thomas, H. Schmit, and D. F. Nagle. Modeling inter-instruction energy effects in a digital signal processor. In *Power Driven Microarchitecture Workshop in conjunction with International Symposium Computer Architecture*, June 1998.
- [13] Mariagiovanna Sami, Donatella Sciuto, Cristina Silvano, and Vittorio Zaccaria. An instruction-level energy model for embedded VLIW architectures. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 21(9):998–1010, 2002.
- [14] J. Laurent, E. Senn, N. Julien, and E. Martin. High level energy estimation for DSP systems. In *PATMOS'01: Proceedings International Workshop on Power And Timing Modeling and Optimization and Simulation*, pages 311–316, September 2001.
- [15] Eric Senn, Nathalie Julien, Johann Laurent, and Eric Martin. Power consumption estimation of a C program for data-intensive applications. In *PATMOS'02: Proceedings of the 12th International Workshop on Integrated Circuit Design. Power and Timing Modeling, Optimization and Simulation*, pages 332–341, London, UK, 2002. Springer-Verlag.
- [16] J. Von Livonius, H. Blume, and T. G. Noll. FLPA-based power modeling and power aware code optimization for a trimedia DSP. In *Proceedings of the ProRISC Workshop*, Veldhoven, Netherlands, 2005.
- [17] Texas Instruments. *TMS320C6416T, Fixed Point Digital Signal Processor, Datasheet*, November 2003. SPRS226J.
- [18] Marco Bianchi, Martin Boyle, and Deirdre Hollingsworth. A comparison of methods for trend estimation. *Applied Economics Letters*, 6(2):103–109, February 1999.
- [19] Mostafa E. A. Ibrahim, Markus Rupp, and Hossam A. H. Fahmy. Power Estimation Methodology for VLIW Digital Signal Processor. In *Proceedings of the Asilomar08 conference on signals, systems and computers*, Asilomar, CA, US, October 2008.

Data Cache-Energy and Throughput Models: A Design Exploration for Overhead Analysis

Muhammad Yasir Qadri

Department of Computing and Electronic Systems
University of Essex, CO4 3SQ, UK
yasirqadri@acm.org

Klaus D. McDonald-Maier

Department of Computing and Electronic Systems
University of Essex, CO4 3SQ, UK
kdm@essex.ac.uk

Abstract— Embedded processors, particularly for low power applications, can benefit from optimization. The memory system is a fundamental candidate for optimization and has been investigated for some time. Cache memories are one of the most complex memory structures which require particular mathematical models for analysis in order to have efficient optimization. This paper proposes energy and throughput models for a data cache, which could be used for overhead analysis for various cache types with relatively small amount of inputs. These models analyze the overheads on an application basis, thus providing the hardware and software designer with the feedback vital to tune the cache or application for a given energy budget. The models are suitable for use at design time in the cache optimization process for embedded processors considering time and energy overhead.

I. INTRODUCTION

The recent drive towards low power processing has challenged the designers and researchers to optimize every component of the processor. However optimization for energy usually comes with some sacrifice on throughput, and which may result in overall minor gain.

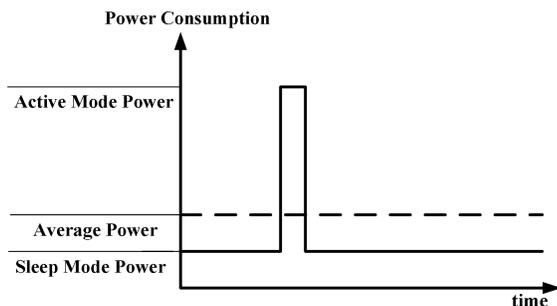


Fig. 1 Power consumption of a typical battery operated processor (adapted from [1])

Fig. 1 shows the operation of a typical battery operated embedded system. Normally, in such devices, the processor is placed in active mode only when required, otherwise it remains in a sleep mode. An overall power saving (increased throughput to energy ratio) could be achieved by increasing the throughput (i.e. lowering the duty cycle), decreasing the peak energy consumption, or by lowering the sleep mode energy consumption. This phenomenon clearly shows the interdependence of energy and throughput for overall power saving. Keeping this in mind, a simplified approach is proposed that is based on energy and throughput models to analyze the impact of a cache structure in an embedded processor per ap-

plication basis. The remainder of this paper is divided into five chapters. In the following two chapters related work is discussed and the energy and throughput models are introduced. In the fourth and fifth chapters experimental environment and results are discussed, and the final chapter forms the conclusion.

II. RELATED WORK

The cache energy consumption and throughput models have been the focus of research for some time. Wen-Tsong et al. [2] present an algorithm to find optimum cache configuration based on cache size, the number of processor cycles and the energy consumption. Their work is an extension of the work of Preeti et al. [3, 4] on data cache sizing and memory exploration. The energy model by Wen-Tsong et al. though highly accurate, requires a wide range of inputs like number of bit switches on address bus per instruction, number of bit switches on data bus per instruction, number of memory cells in a word line and in a bit line etc. which may not be known to the model user at an earlier stage. Another example of a detailed cache energy model was presented by Milind et al. [5]. These analytical models for conventional caches were found to be accurate to within 2% error. However, they over-predict the power dissipations of low-power caches by as much as 30%. CACTI (cache access and cycle time model) [6] is an open-source modelling tool based on such detailed models to provide thorough, near accurate memory access time and energy estimates. However it is not a trace driven simulator, so energy consumption resulting in number of hits or misses is not accounted for a particular application. Chen et al. in [7] have presented a framework for data locality prediction, which can be used to profile a code to reduce miss rate. The framework is based on approximate analysis of reuse distance, pattern recognition, and distance-based sampling. Their results show an average of 94% accuracy when tested on a number of integer and floating-point programs from SPEC and other benchmark suites. Extending their work Xipeng et al. in [8] introduce an interactive visualization tool that uses a three-dimensional plot to show miss rate changes across program data sizes and cache sizes. Another very useful tool named RDVIS as a further extension of the work previously stated was presented by Beyls et al. [9, 10]. Based on cluster analysis of basic block vectors, the tool gives hints on particular code segments for further optimization. This in effect provides valuable feedback to the programmer to improve temporal locality of the data to increase

hit rate for a cache configuration. The following chapter presents the proposed cache energy and throughput models, which can be used to identify an early cache overhead estimate based on a limited set of input data. These models are an extension of the models previously proposed by Qadri et al. in [11].

III. THE D-CACHE ENERGY AND THROUGHPUT MODELS

The cache energy and throughput models given below strive to provide a complete application based analysis. Thus they facilitate the tuning of a cache and an application according for a given power budget. The overhead can be obtained by taking the ratio of energy/throughput with cache to the corresponding values without cache.

A. Energy Model

If E_{read} and E_{write} is the energy consumed by cache read and write accesses, $E_{leak(std)}$ the leakage energy of cache structure in stand-by mode, $E_{c \rightarrow m}$ the energy consumed by cache to memory accesses, E_{mp} the energy miss penalty and E_{misc} is the Energy consumed by other instructions (which do not require data memory access), then the total energy consumption of the code E_{total} in Joules [J] could be defined as

$$E_{total} = E_{read} + E_{write} + E_{leak(std)} + E_{c \rightarrow m} + E_{mp} + E_{misc}. \quad (1)$$

Further defining the individual components,

$$E_{read} = n_{read} \cdot E_{dyn.read} \cdot \left[1 + \frac{r_{miss}}{100}\right], \quad (2)$$

$$E_{write} = n_{write} \cdot E_{dyn.write} \cdot \left[1 + \frac{r_{miss}}{100}\right], \quad (3)$$

$$E_{c \rightarrow m} = E_m \cdot (n_{read} + n_{write}) \cdot \left[1 + \frac{r_{miss}}{100}\right], \quad (4)$$

$$E_{mp} = E_{idle} \cdot (n_{read} + n_{write}) \cdot \left[P_{miss} \cdot \frac{r_{miss}}{100}\right], \quad (5)$$

where n_{read} is the number of read accesses, n_{write} the number of write accesses, $E_{dyn.read}$ the total dynamic read energy for all banks, $E_{dyn.write}$ the total dynamic write energy for all banks, E_m the energy consumed per memory access, E_{idle} the per cycle idle mode energy consumption of the processor, r_{miss} the miss ratio (in percentage), and P_{miss} is the miss penalty (in number of stall cycles).

It is worth noting that $E_{leak(std)}$ depends upon the time for which cache is not accessed; and could be calculated using the following equation

$$E_{leak(std)} = P_{leak} \cdot t_{cache-idle}, \quad (6)$$

where $t_{cache-idle}$ is the total time for which cache was idle in seconds.

B. Throughput Model

Due to the concurrent nature of cache to memory access time and cache access time, their overlapping can be assumed. If t_{cache} is the time taken for cache operations, t_{mem} the time saved from memory operations, t_{mp} the time miss penalty and t_{misc} is the time taken while executing other instructions which do not require data memory access; then the total time taken by an application with a data cache could be estimated as

$$T_{total} = t_{cache} - t_{mem} + t_{mp} + t_{misc}. \quad (7)$$

Furthermore,

$$t_{cache} = t_c \cdot (n_{read} + n_{write}) \cdot \left[1 + \frac{r_{miss}}{100}\right] \text{ and} \quad (8)$$

$$t_{mp} = t_{cycle} \cdot (n_{read} + n_{write}) \cdot \left[P_{miss} \cdot \frac{r_{miss}}{100}\right] \quad (9)$$

where t_c is the time taken per cache access and t_{cycle} is the cycle time in seconds [sec]. If leakage energy is considered, then $t_{cache-idle}$ could be calculated by subtracting the time for which cache was active ($t_{cache-active}$) from the total time for which processor was active, where

$$t_{cache-active} = t_{cache} + t_{mp}. \quad (10)$$

IV. THE EXPERIMENTAL ENVIRONMENT

To analyze the above given models, AVR ATmega644P microcontroller architecture was used as the target evaluation platform for its being a non-cache system [12]. This particular processor is a part of Atmel's picoPower family of energy efficient processors. UCLA's AVRORA (AVR Simulation and Analysis Framework) [13] was used to obtain coarse-grained code profile and cycle count information. The AVRORA's code profile is termed coarse-grain because it gives cycle counts and number of calls based on sections of the software; however it does not provide the exact count of memory accesses as required for the mathematical models proposed above. Furthermore, the AVRORA is based on ATmega128 ISA; and the Call and Return instructions of ATmega644P require one cycle more than the ATmega128. To overcome these problems a software tool, the Essex AVR Code Profiler was designed. The Essex AVR Code Profiler uses the output from AVRORA code profiler, along with the object dump file, in order to generate instruction level profile for a given code. CACTI 4.2 [6] was used for estimating cache and data memory access time and energy consumption. As the models analyze the cache overhead based on a specific applications rather than provide generic results; applications based on MiBench [14] and the MSP430 competitive benchmarking [15] were selected for analysis purpose.

1) *BasicMath*: The basic math application is a part of automotive suite of MiBench. It performs cubic function solving, integer square root and angle conversions from degrees to radians.

2) *QSort*: The QSort application is also a part of automotive suite of MiBench. It sorts a large array of strings into ascending order using the quick sort algorithm.

3) *FFT*: This benchmark is a part of telecommunication suite of MiBench. It performs a fast Fourier transform on an array of data. Fourier transforms are used in a wide variety of digital signal processing applications to find the frequencies contained in a given input signal.

4) *Dijkstra*: The Dijkstra benchmark is a part of network suite of MiBench. It constructs a large graph in an adjacency matrix representation and then calculates the shortest path between every pair of nodes using repeated applications of

Dijkstra's algorithm. The Dijkstra's algorithm is a well known solution to the shortest path problem.

5) *MatrixMul*: It is a part of MSP430 competitive benchmarking suite. The application multiplies a 3 x 4 matrix by a 4 x 5 matrix.

6) *FIR Filter*: It is a part of MSP430 competitive benchmarking suite. It calculates the output from a 17-coefficient tap filter using simulated ADC input data.

The above benchmarks were carefully chosen with particular reference to typical low power application scenarios ranging from sensor networks to automotive and digital signal processing. For the above cited applications the Essex AVR Code Profiler generated the instruction cycles distribution results as shown in Fig 2. The instructions are classified as Call, Push/Pop, DMStore (data memory store), PMStore (program memory store), DMLoad (data memory load), PMLoad (program memory load), branch, jump and miscellaneous (i.e. instructions which do not require data memory access).

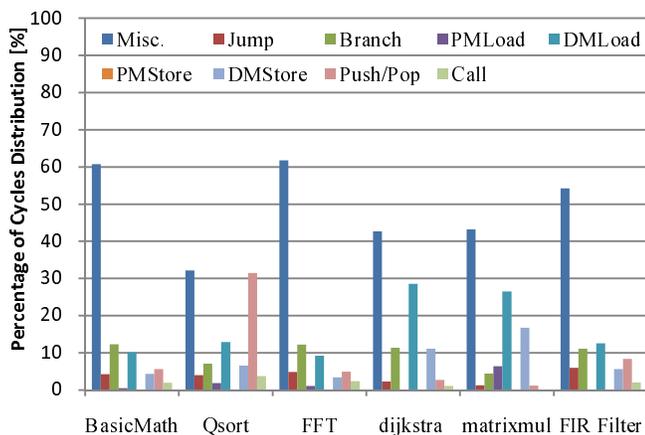


Fig. 2 Instruction Cycle distribution as per classification.

To identify cache access time, energy per read and write access; the CACTI cache modeller was used for a 512 bytes, 4-way set associative cache with the parameters shown in Table I.

TABLE I
CACTI CACHE INPUT PARAMETERS

Cache Parameter	Value
Number of banks	2
Total cache Size [bytes]	512
Size in bytes of a bank	256
Number of sets per bank	8
Associativity	4
Block Size [bytes]	8
Read/Write Ports	1
Read Ports	0
Write Ports	0
Technology Size [nm]	355
V _{dd} [Volts]	2.618

The cache fabrication technology was selected as 355nm which is the same as of ATmega644P [16]. The size of the cache was selected to be realistically in par with the actual ATmega644P data memory size, i.e. 4 KB. It is interesting to note that, the power dissipation of a direct mapped cache could be about 30% of a same sized four way set associative cache [17], so it can result in greater energy efficiency. However, typically a direct mapped cache has the highest miss rate, as the miss rate tends to decrease with the increase in the cache associativity [18]. Thus a 4-way set associative cache presents a viable compromise with energy efficiency and miss rate. Furthermore, if the cache is gated as proposed in a scheme by Michael et al. [19], stand-by leakage energy could be reduced up to 97% of active leakage energy. Thus assuming the same structure in this case, E_{leak} could be ignored for simplification purpose. Also the particular version of CACTI used does not provide leakage power data for 355 nm process technology.

V. RESULTS

For the given cache parameters, the cache modeller resulted in access time of 2.83ns, and a total dynamic read and write energy of 0.34nJ, and 0.068nJ respectively. As the ATmega644P energy consumption per memory access information is unknown, it was conservatively assumed to be the same as the per cycle energy consumption of the processor i.e. 1nJ at 2V and 1MHz [12]. The cache miss penalty was assumed to be 10 cycles. This assumption was based upon the one made by Hiroyuki et al. [20], however practically the cache miss penalty varies according to cache organizations and its miss handling policies. The code profiler resulted in the data depicted in Table II.

TABLE II
CODE PROFILER OUTPUT

Application	Memory Read Operations	Memory Write Operations
BasicMath	91087797	46282571
QSort	16922	13500
FFT	23034	10126
Dijkstra	215852	87288
MatrixMul	490	268
FIR Filter	116587	67009

The energy and throughput models results are detailed in Fig. 3 to 8, with corresponding graphs of power efficiency in cycles/watt. The graphs at the left show energy and time overhead versus miss rate for the particular benchmark applications. The overhead is obtained by taking the ratio of energy/throughput with cache (by using the proposed models) to the corresponding values without cache. To analyze the dependence of energy and throughput overheads on miss rate, a complete range of miss rates is considered. This in turn would help in rationalizing the feasibility of the particular cache structure for some range of miss rates. The graphs at the right show the power efficiency against miss rate, giving average power consumption statistics for the complete range of miss rates.

With the data listed in Table II, the energy models results showed that, the QSort application gave the least Power Efficiency (see Fig. 4b). Also in Fig 4a, the energy overhead for QSort varied from a minimum of 0.99 to a maximum of 2.8, and the time overhead varied from a minimum of 0.044 to a maximum of 4.76 for miss rates varying from 0 to 100%. It can be inferred from the results that, for miss rates below 15% data cache is feasible in terms of power efficiency for applications like QSort, with the assumed miss penalty. While on the other hand all the other applications except matrixmul showed data cache feasibility for miss rates below 20% (see Fig. 3, 5, 6, 8). For matrixmul the cache appears to be feasible for miss rates below 25% (see Fig. 7). It must be also be noted that these results are particular to the ATmega644P microcontroller and cannot be generalized for other platforms. The least power efficiency of QSort is due to its highest ratio of memory access instructions to non-memory access instructions as compared to any other application, as shown in Fig 2. It can be said that an increase in energy overhead and decrease in time overhead can

be observed as this ratio increases, below the miss rate threshold point (where energy and throughput overhead both have a value greater than 1). Generally, for miss rates below that threshold value (which in the case of QSort is 20% and for all others is typically 30%) the energy overhead is greater than the time overhead, and the phenomenon tends to reverse above this threshold. A greater power efficiency is observed below this threshold value and the cache becomes feasible for miss rates lower than that.

The applications selected for benchmarking were not optimized for miss rate reduction; furthermore the specific miss rate is also unknown. The proposed models were analyzed for several applications because in practice an embedded system has to run single (or a set of few) application(s) for its whole life. Thus it is important to carefully select a cache configuration and to perform code optimization in order to reduce the miss rate.

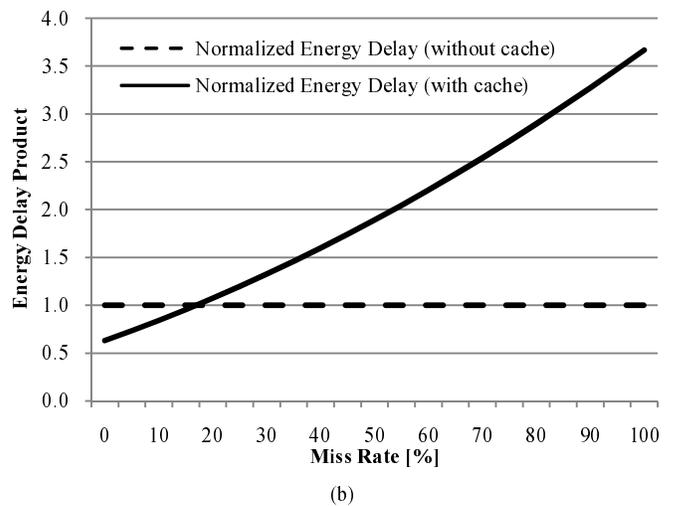
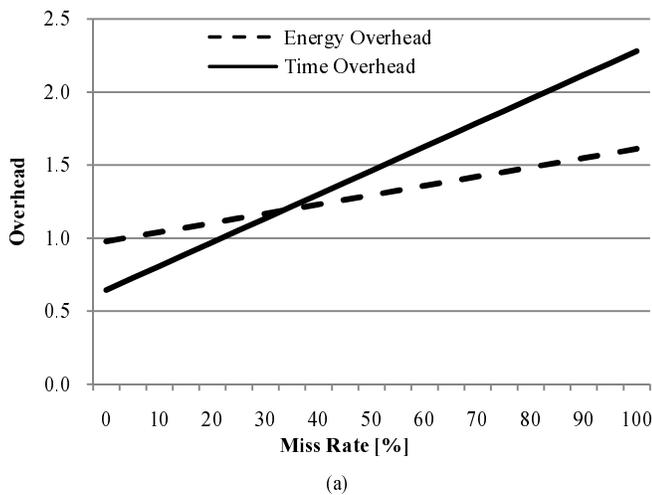


Fig. 3 BasicMath (a) Energy and Time overhead vs. Miss Rate. (b) Energy-Delay product vs. Miss Rate

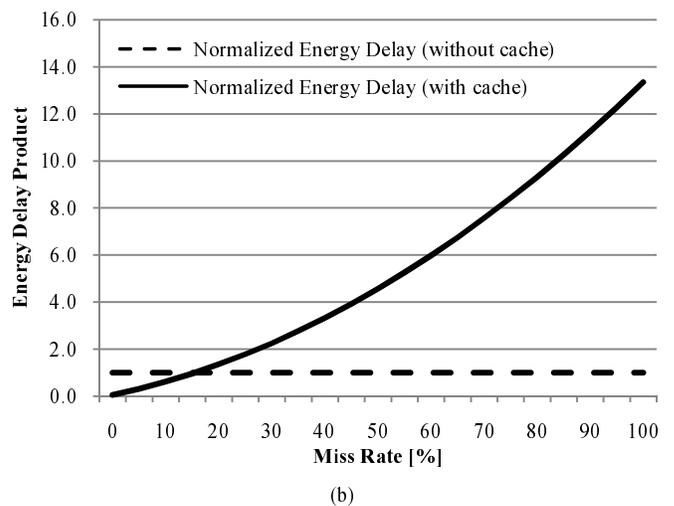
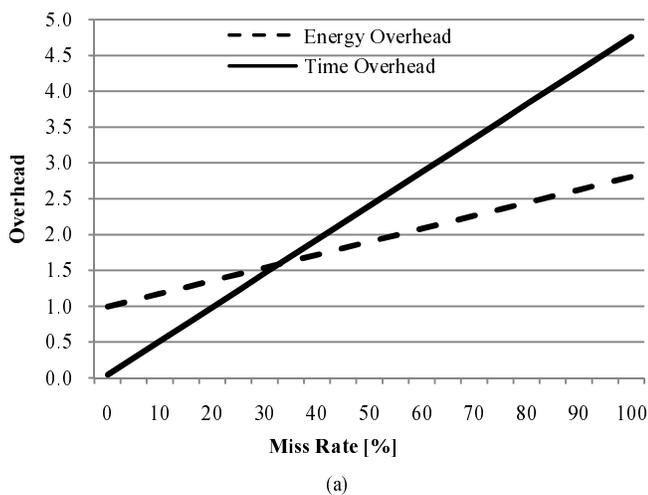
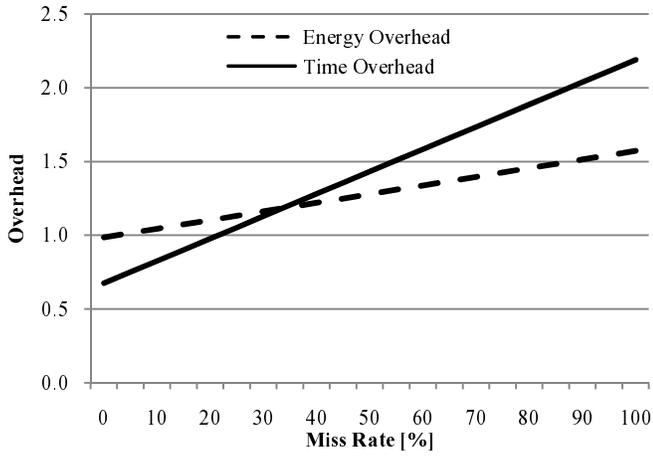


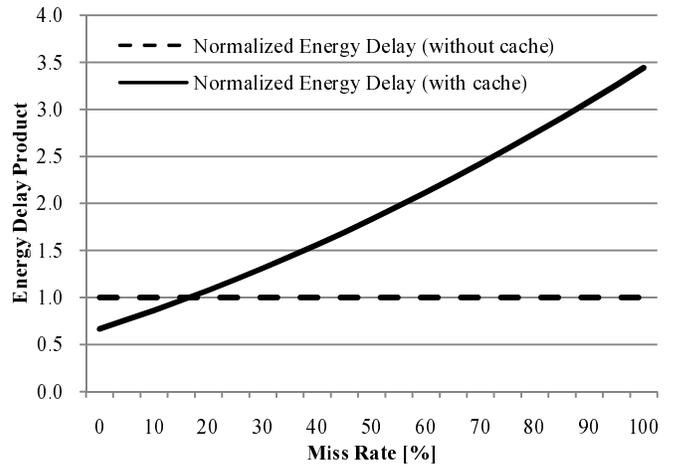
Fig. 4 QSort (a) Energy and Time overhead vs. Miss Rate. (b) Energy-Delay product vs. Miss Rate

DASIP 2008

November 2008

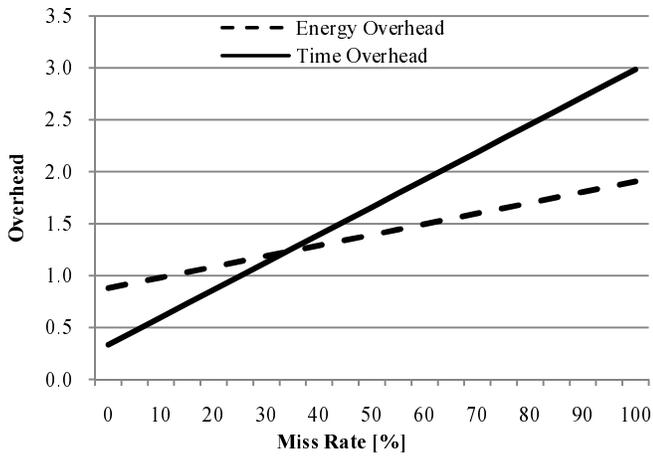


(a)

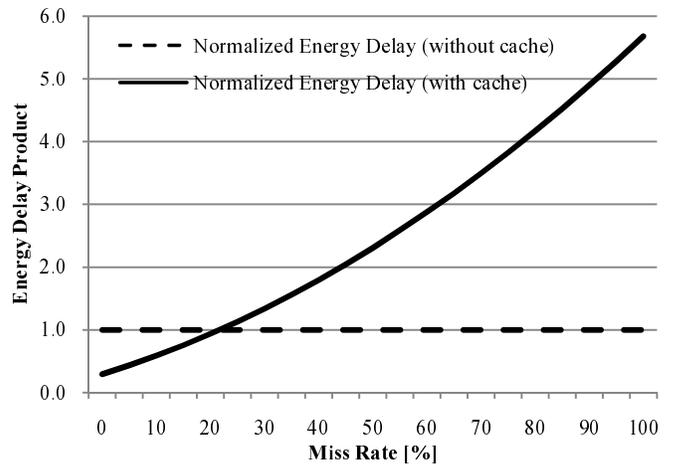


(b)

Fig. 5 FFT (a) Energy and Time overhead vs. Miss Rate. (b) Energy-Delay product vs. Miss Rate

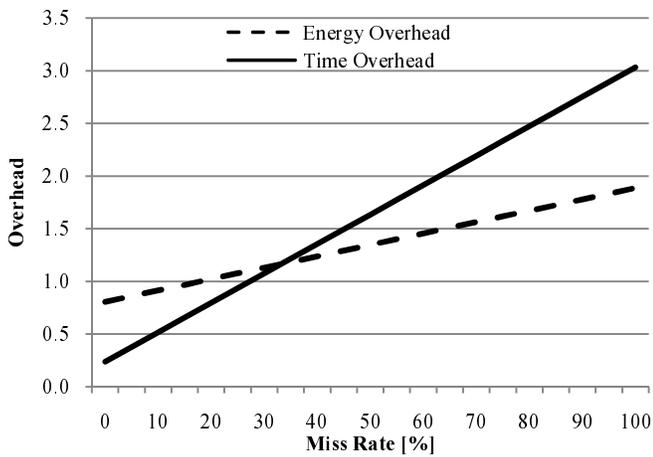


(a)

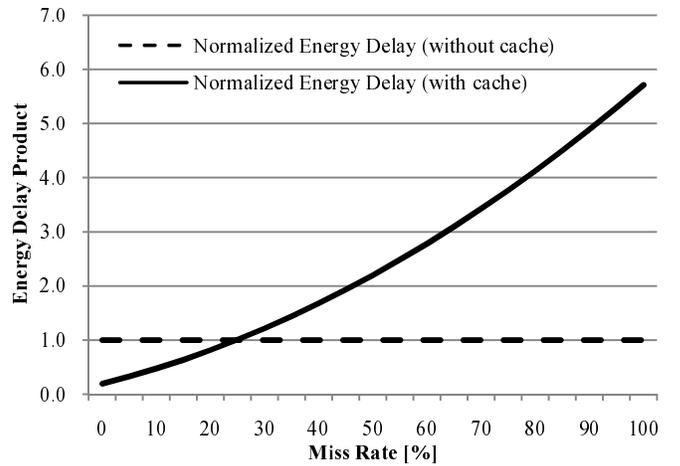


(b)

Fig. 6 Dijkstra (a) Energy and Time overhead vs. Miss Rate. (b) Energy-Delay product vs. Miss Rate



(a)



(b)

Fig. 7 MatrixMul (a) Energy and Time overhead vs. Miss Rate. (b) Energy-Delay product vs. Miss Rate

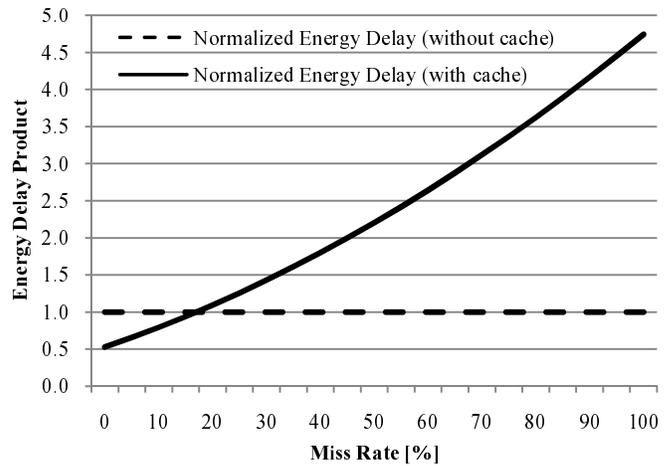
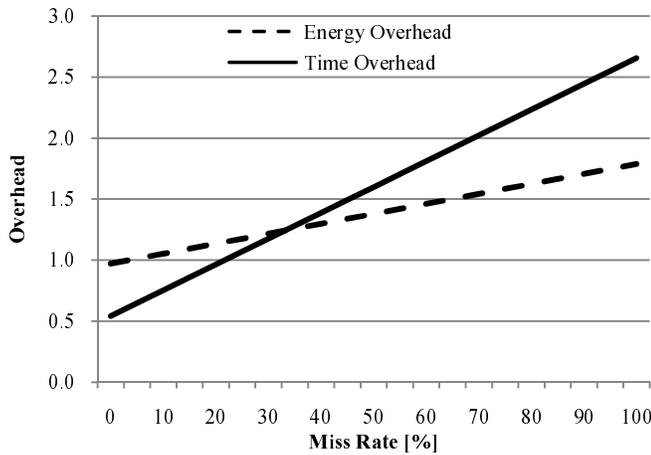


Fig. 8 FIR Filter (a) Energy and Time overhead vs. Miss Rate. (b) Energy-Delay product vs. Miss Rate

A design exploration strategy for finding optimal cache configuration and code profile is shown in Fig. 9. At first the miss rate prediction is carried out on the compiled code and preliminary cache parameters. Then several iterations may be performed to fine tune the software to reduce miss rates. Subsequently, the tuned software goes through the profiling step. The information from the cache modeller and the code profiler is then fed to the energy and throughput models. If the given energy budget along with the throughput requirements is not satisfied, then the cache parameters are to be changed and the same procedure is repeated. This strategy can be adopted at design time to optimize the cache configuration and decrease the miss rate of a particular application code.

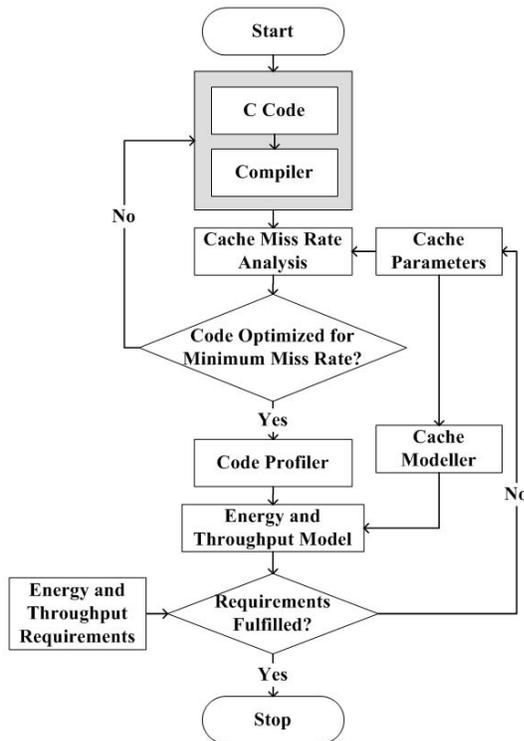


Fig. 9 Proposed design cycle for optimization of cache and application code

VI. CONCLUSION

In this paper straightforward mathematical models were presented that can provide an early estimate of a data cache energy and throughput overheads for a particular application. A design exploration strategy was also put forward to facilitate the identification of an optimal cache configuration and code profile for a target application. It can be inferred from the results that a cache structure is not always beneficial in terms of power efficiency for any application in low power embedded systems. However if the code is optimized for minimum possible miss rate which is significantly lower than the threshold defined in previous chapter, then a cache becomes a good choice for increased throughput and energy efficiency. In future work the presented models are to be analyzed for a soft core configurable processor, so that a fair estimate of their accuracy can be found. Also miss rate prediction strategies could be explored to further these analyses.

ACKNOWLEDGMENT

This research is in part supported by the UK Engineering and Physical Sciences Research Council (EPSRC) under Grants EP/C005686/1, EP/C014790/1 and EP/C54630X/1.

REFERENCES

- [1] A. M. Holberg and A. Saetre, "Innovative Techniques for Extremely Low Power Consumption with 8-bit Microcontrollers (White Paper)," Atmel Corporation 2006.
- [2] S. Wen-Tsong and C. Chaitali, "Memory exploration for low power, embedded systems," in Proceedings of the 36th ACM/IEEE conference on Design automation New Orleans, Louisiana, United States: ACM, 1999.
- [3] P. Preeti Ranjan, D. D. Nikil, and N. Alexandru, "Architectural exploration and optimization of local memory in embedded systems," in Proceedings of the 10th international symposium on System synthesis Antwerp, Belgium: IEEE Computer Society, 1997.
- [4] P. Preeti Ranjan, N. D. Dutt, and A. Nicolau, "Data cache sizing for embedded processor applications," in Proceedings of the conference on Design, automation and test in Europe Le Palais des Congrès de Paris, France: IEEE Computer Society, 1998.
- [5] B. K. Milind and G. Kanad, "Analytical energy dissipation models for low-power caches," in Proceedings of the 1997 international symposium on

- sium on Low power electronics and design Monterey, California, United States: ACM, 1997.
- [6] S. T. David Tarjan, Norman P. Jouppi, "CACTI 4.0," HP Laboratories Palo Alto June 2006.
- [7] D. Chen and Z. Yutao, "Predicting whole-program locality through reuse distance analysis," SIGPLAN Not., vol. 38, pp. 245-257, 2003.
- [8] S. Xipeng and S. Ahren, "Miss Rate Prediction Across Program Inputs and Cache Configurations," IEEE Trans. Comput., vol. 56, pp. 328-343, 2007.
- [9] K. D. H. Beyls, E., Platform-Independent Cache Optimization by Pinpointing Low-Locality Reuse vol. 3038/2004: Springer, 2004.
- [10] K. D. H. Beyls, E.; Vandeputte, F., RDVIS: A Tool that Visualizes the Causes of Low Locality and Hints Program Optimizations vol. 3515: Springer., 2005.
- [11] M. Y. Qadri and K. D. M. Maier, "Towards increased power efficiency in low end embedded processors: Can cache help?," in UKEF, Southampton, UK, 2008.
- [12] "ATmega644P Datasheet."
- [13] L. T. Ben, K. L. Daniel, and P. Jens, "Avrora: scalable sensor network simulation with precise timing," in Proceedings of the 4th international symposium on Information processing in sensor networks Los Angeles, California: IEEE Press, 2005.
- [14] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop: IEEE Computer Society, 2001.
- [15] K. V. Greg Morton, "MSP430 Competitive Benchmarking," Texas Instruments SLAA205B, July 2006.
- [16] "Qualification By Similarity for Microcontrollers/AVR Products," ATMEL Corporation, San Jose, CA October 12th, 2007 2007.
- [17] "International Technology Roadmap for Semiconductors," SEMATECH, Semiconductor Industry Association, Austin, Texas 1999.
- [18] Z. Chuanjun, V. Frank, and N. Walid, "A highly configurable cache architecture for embedded systems," SIGARCH Comput. Archit. News, vol. 31, pp. 136-146, 2003.
- [19] P. S.-H. Michael, Yang; Babak, Falsafi; Kaushik, Roy; T. N. Vijaykumar, "Gated-Vdd: a circuit technique to reduce leakage in deep-submicron cache memories," in Proceedings of the 2000 international symposium on Low power electronics and design Rapallo, Italy: ACM, 2000.
- [20] T. Hiroyuki and Y. Hiroto, "Code placement techniques for cache miss rate reduction," ACM Trans. Des. Autom. Electron. Syst., vol. 2, pp. 410-429, 1997.

Using Traceability for Reverse Instance Transformations with SiTra

Seyyed Shah, Kyriakos Anastasakis, Behzad Bordbar
 School of Computer Science
 University of Birmingham, Edgbaston, B15 2TT. UK.
 Email: {szs|kxa|bxb}@cs.bham.ac.uk

Abstract—Model Driven Development and the core concept of Model Transformation has gained wide acceptance especially when used with UML languages. Model Transformations are used to map models in one language to another and can be used to transform a design model into an implementation or for analysing a design model to identify faults. However, transformations are a one time bridge and the instances of transformed models can not be automatically mapped back into the original language. This is despite the fact that a mapping must already exist between the two models. This mapping is represented in the *trace* of the transformations execution. Tracing is a feature of several transformation frameworks where by the source of every destination element is recorded.

Tracing has originally been applied for *change propagation* in chains of transformation and in debugging Model Transformations. In the current paper we present a novel use of Model Transformation tracing: for reverse instance transformation. That is, the automatic transformation of instances of the destination model back into instances of the source model. We demonstrate the method in a case study of UML2Alloy, a complex transformation from UML to the Alloy analysis language. In this case study, A UML Class Diagram is transformed in to its equivalent Alloy form. The presented method *automatically* transforms analysis (instances) of the Alloy model, back into UML-Object Diagram form that are valid instances of the original UML Class Diagram.

I. INTRODUCTION

Model Driven Development (MDD) [34] aims to promote the role of modeling in software development. Models in the context of MDD are captured in machine-readable representations, using languages which are widely adopted by the software industry [35]. Hence, it is possible to communicate such models to various parties and reuse them. This results in lower software production cost and shorter development cycles. Central to the MDD is the idea of automated transformation of models via tools, commonly known as Model Transformation Frameworks, that execute transformation automatically [4], [23], [1], [31]. A typical Model Transformation framework accepts three inputs, a metamodel of the source language, a metamodel of destination language and a specification of the transformation which maps the model elements of the source to the destination. Then, for any given model complying to the metamodel of the source the tool executes the transformation resulting in the creation of an instance of the metamodel of the destination.

In complex applications domains, MDD can also be used to create multiple models of the systems automatically to bridge

the gap between technical spaces [22]. For example, MDD can be used to create analysable models from a design model [21], [3]. In such cases, the result of the analysis must also be transformed back to be presented to the designer. In this paper we demonstrate that traceability is an important issue in such Model Transformations. In effect traceability is a mechanism for recording the link between the source and target model elements [11], [29]. Establishing such links allows defining the reverse transformations automatically.

This paper reports on our current research on extending the Simple Transformer (SiTra) [1] with traceability capabilities. SiTra is a simple and lightweight implementation of an extensible Transformation Engine. SiTra has been successfully applied to model transformation in various application domains[2], [8], [7]. The paper also reports on a case study involving application of new version of SiTra to a complex Model Transformation from UML to Alloy [7].

The paper is structured as follows, the next section gives a brief background to the subject at hand. The concepts of Model Driven Development and Model Transformation, tracing transformations and the transformation framework: SiTra. Next we shall describe the problem followed by an outline of the solution in terms of the architecture and algorithm, as well as how SiTra was modified to accommodate traceability. Finally we present an example Model Transformation in UML2Alloy that applies the solution to a specific problem, followed by some discussion.

II. BACKGROUND

A. Model Driven Engineering

Model Driven Architecture (MDA) [25], a flavour of MDD which is initiated by the Object Management Group (OMG). MDA makes use of Meta Object Facility (MOF) [28] which describes the metamodels. Metamodels are themselves *models*, from which models of the system are instantiated. MOF can be compared to EBNF, which is used for defining programming languages grammars. As a result, MOF is a blueprint from which MOF Compliant metamodels are created.

Figure 1 depicts an outline of MDA and the process of Model Transformation. A number of Transformation Rules are used to define how various elements of one metamodel (source metamodel) are mapped into the elements of another metamodel (destination metamodel). The process of Model Transformation is carried out automatically via software tools

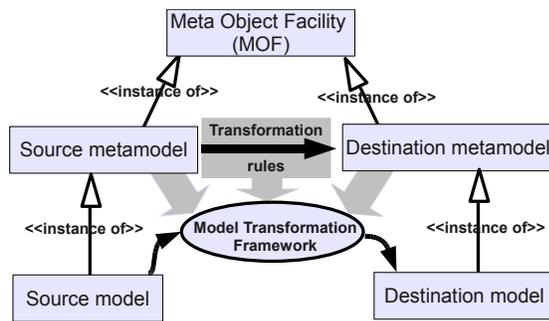


Fig. 1. An Overview of MDD

which are commonly referred to as Model Transformation Frameworks [37], [1], [13]. A typical Model Transformation Framework requires three inputs: source metamodel, destination metamodel and Transformation Rules. For any instance of the source metamodel, a *Transformation Engine* executes the rules to create an instance of the destination metamodel.

The central concept of MDD is Model Transformation [32], the mechanism for bridging technical spaces. Model transformations take as input one or more source models that conform to source language and translate them into one or more target models in the context of a destination language. Judson et al. [20] propose that transformations can be applied from one of two dimensions; vertical, that change the context for example Design to code or horizontal, as analysis for example UML2Alloy.

B. Model Transformation Traceability

Traceability is a feature in a Model Transformation Engines that keeps record of which element(s) in the source model maps to which element in the destination. Bondé et al. [5] apply the traceability to *change propagation*, thus if the source changes slightly, the change can be reflected in the destination without re-running the entire transformation. Change propagation is most useful when several successive transformations are applied to a model, so the models can be made interoperable. Moreover, the ability to trace the source of an element has been used in debugging of Model Transformation [16]. Thus, traceability support is a desirable part of a Transformation Engine feature set and for developer support.

Jouault [19] identifies two groups of model transformation traceability strategies, automatic or manual tracing. Automatic tracing requires no manual intervention by the developer, the trace information is recorded transparently during the transformation. Manual tracing, as the name suggests, requires explicit tracing rules be defined as part of the Model Transformation to be traced. Each method has its merits, automatic tracing requires little developer intervention and leads to less cluttered transformations; manual tractability gives the developer control over what information is traced. For further classification of tracing strategies in frameworks, see [11]. It is clear that integrated traceability support in the form of automatic tracing is a desirable feature for a Model Transformation framework.

The requirements for traceability information vary between frameworks. Vanhooft and Berbers [36] encode traceability information into a *UML Profile for Traceability*, where a model of the transformation is extended with trace information. The approach is taken so that large Model Transformations can become smaller transformations as part of “Transformation Chains”. The smaller transformations are more axiomatic so have dependencies on preceding transformations outcomes and rely on trace information. The Kermeta [12] framework with a similar aim of enabling chains of transformations, defines a distinct meta-model of traceability information. So a model of the trace is populated at transformation time. The OMG’s QVT [29] Model Transformation specification aims for a generic approach to transformation traceability, defining the *Trace Class* and *Trace Instance* entities. Trace instances are created with the appropriate information during transformation. The precise form of traceability support across languages depends on the motivation for adding traceability to a framework.

C. Simple Transformer (SiTra)

There are a wide range of languages available to specify MDD Transformation Rules [4], [13], [29]. Such languages, which are mostly extending [30], not only provide strong constructs for the specification of Model Transformations, but also are supported by Model Transformation Frameworks for executing the transformations. However, none of these languages are widely adopted by the academic community or industrial tool vendors. Much anticipated Query View Transformation (QVT) by OMG is now finalised [29] and is expected to result in a unifying language for specifying transformations. To execute a specification of a Model Transformation in the above languages, they must be transformed into lower-level languages such as Java.

In a large project, it is possible to divide the specification and implementation of Model Transformations between two different groups of people who have relevant skills. In the case of smaller groups of developers and newcomers to MDD, the combined effort involved in becoming an expert in the two sets of skills described above is overwhelming. In particular, the steep learning curve [24] associated with current MDD tools, such as [10], [9], [14], is an inhibitive factor in the adoption of MDD by even very experienced programmers. Simple Transformer (SiTra) [1] is a simple and lightweight Model Transformation Framework aiming to use Java for both writing Model Transformations and providing a minimal environment (the execution engine) for transformation execution. SiTra consists of two interfaces, the *Rule* interface, which user defined mapping rules have to implement and the *Transformer* interface, which provides the skeleton of the methods that carry out the transformation. The authors of SiTra provide a simple implementation of the *Transformer* interface and to use SiTra for simple transformations. The modeller only needs to define the transformation rules by implementing the *Rule* interface, which consists of three methods: *check()*, *build()* and *setProperties()*. If the rule is applicable for the *source* element in question, the *check()* method of the rule interface returns

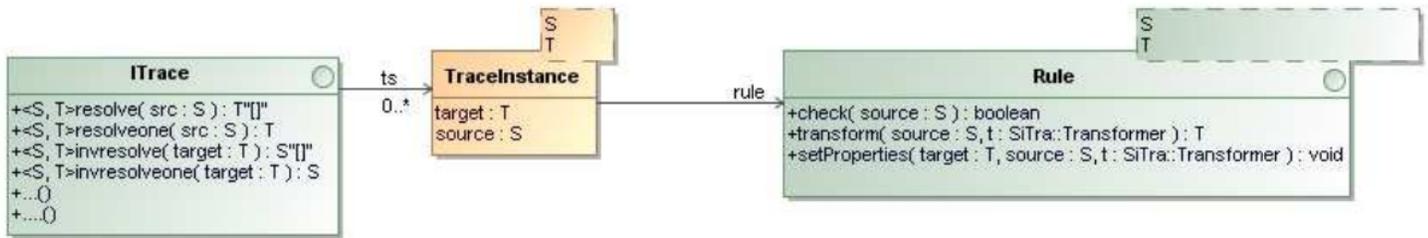


Fig. 2. A Model of the Tracing Mechanism

true and the *build()* method is executed. The *build()* method generated the target model element. The *setProperties()* is used to set the attributes and links of the newly created target element. SiTra has been successfully applied to Model Transformation in various application domains[2], [8], [7]. For further details on SiTra please refer to [1].

III. OUTLINE OF THE APPROACH

A. Description of the Problem

Often a Model Transformation is used to produce a model of a target language as an intermediate step in a process. For example, a model transformation may be used to transform a model from a source language *A* to a target language *X*, to take advantage of more advanced tool support in *X*. In such a case, the toolset of the target language is used to process (e.g. analysis, refactoring) the produced model. The results of this processing need to be interpreted in the domain of the original language *A*. If the transformation between the source and the target languages is not bidirectional this is not a simple task. In this section we present an algorithm, which uses the tracing information of a Model Transformation to interpret the results of a process on the target model, using concepts of the source language, in a unidirectional transformation. Moreover we propose extending the SiTra framework with support for traceability, to implement our method.

B. Architecture and Algorithm

Figure 3 depicts an overview of the method proposed. On the metalevel (*M2*) elements of the *Source* and *Target* metamodels (*A* and *X* respectively) are mapped using the Model Transformation *T*. On the model level (*M1*) if a source model *B* is given as input to the transformation a target model *Y* will be automatically generated. The execution of the Model Transformation will also generate a set of traces *T'*, which record how each element of the source model is mapped to an element of the target model. If *Z* is an instance of the target model on the *M0* level, using the trace information *T'*, we can construct *C*, which is an instance of the source model *B*.

The algorithm to extrapolate *C* is as follows. For all elements in *Z*, find the source of the parent element from *T'* and create an instance of that element. Once all elements in *Z* are applied to the algorithm, the resulting instance model *C* is produced. This is a valid instance model of *B* at *M0*-level in the source language, created automatically. The algorithm is shown in Figure 4.

C. Tracing Support in SiTra

In this section we describe how the SiTra Model Transformation engine is modified to add traceability support. SiTra, was developed by Akehurst et al. [1] as a simple Object Oriented Transformation Engine, in principle based on a modified visitor pattern. The engine has been modified to support both Model to Model (*M2M*) and Model to Text (*M2Text*) transformation tracing.

The MOF Queries Views and Transformations (QVT) specification [29] defines a tracing mechanism that can be used to trace which source metamodel elements are mapped to which target metamodel elements and the inverse. Based on the tracing mechanism of the QVT specification, we have developed and implemented an extension to the SiTra Transformation Engine. Figure 2 depicts our model for tracing *M2M* transformations.

More precisely, our tracing consists of an interface (*ITrace*), which holds a collection of *TraceInstances* (*ts*). Each *TraceInstance*, represents a mapping between a source and a target model element, through a SiTra rule. An implementation of the *ITrace* interface, provides a number of methods to query the *ts* collection. More specifically the *resolve* method, queries the *ts* collection, and returns all target instances that have been created during the transformation, from the *src* instance. Likewise, *resolveone* should return only the first instance of the target element that has been created during the transformation from the *src* instance. The method names preceded with 'inv' (i.e. *invresolve*), perform the inverse (i.e. return the source elements that have been mapped to the target element passed as a parameter). The QVT specification, also defines a number of additional methods that can be used to query the trace model.

To populate this tracing model, we have extended the implementation of the *transform* method of the SiTra distribution. Our implementation ensures that, for each rule being executed, a trace is recorded (i.e. a *TraceInstance*), which keeps track of the instance of the SiTra rule responsible for the transformation, the instance of the source metamodel provided as input to the rule and the instance of the target metamodel produced by the rule.

A similar tracing model has been implemented in SiTra for Model to Text (*M2Text*) transformations, which are usually defined in order to transform the abstract syntax model elements to a textual notation. Each *TraceInstance* of the *M2Text* trace model maps an element of the metamodel of the languages,

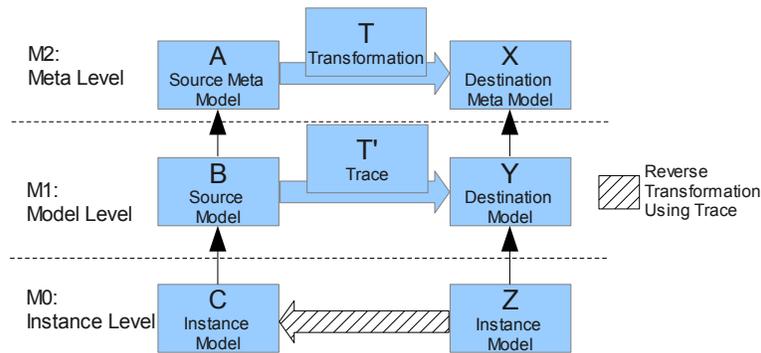


Fig. 3. Architecture of the Proposed Solution

Input: Instance Model: Z, Trace: T', Model: Y, Model: B
Output: Instance Model: C
foreach *element z in model Z* **do**
 find the class element of z, y from the model Y
 query T' using y to find class element b, the source of y
 using z, create c in instance model C
end

Fig. 4. Algorithm for Reverse Instance Transformation

to a range in the generated text model. The range is identified by the rows and columns it occupies in a text file.

IV. CASE STUDY: UML2ALLOY

To demonstrate the method, we apply the approach to a model transformation from UML class diagrams enriched with OCL constraints [35], to the Alloy language [18]. Alloy is an increasing popular textual language based on first-order logic and fully automated analysis capabilities. The transformation from UML to Alloy is implemented as part of a tool called UML2Alloy [3], which uses the SiTra transformation framework. UML2Alloy has successfully been applied to the analysis of agile manufacturing [7], E-Business applications [6] and Security in E-Commerce systems [15].

UML is now widely accepted for designing systems before implementation, allowing better modelling as a part of the development process. The idea behind UML2Alloy is to allow the designer to specify the system using UML and simultaneously harness the analysis capabilities of the Alloy language to identify possible faults in the created design via the Alloy language. More precisely, the Alloy language is supported by a tool called Alloy Analyzer, which is able to automatically simulate an Alloy model by creating an arbitrary instance of the model. Additionally the analyser offers the ability to debug overconstrained models [33], by locating the statements which are responsible for the inconsistent model.

Figure 5 depicts an overview of the problem addressed in this paper in the context of UML2Alloy. The UML2Alloy transformation maps elements of the UML and OCL metamodel to elements of the Alloy metamodel, which has been developed using the Alloy grammar [3]. If a UML model is given to the UML2Alloy implementation, it can automatically

create an Alloy model. This Alloy model can then be automatically analysed, by exploiting the Alloy Analyzer API. If the model is consistent the Alloy Analyzer will create an arbitrary instance of the model in XML form. If the model is overconstrained, however, it will return the lines and columns of the statements responsible for the inconsistency.

In both in simulation and analysis we need to represent the results of the outcome of the Alloy Analyzer, from the Alloy language concepts back to the UML domain. If the analyser provides an instance of the model, we need to represent the instance in terms of a UML Object Diagram [35]. If, on the other hand, the analyser returns the regions in the Alloy text file that are responsible for an overconstrained Alloy model, we need to locate the original UML model elements responsible for the inconsistency and present them to the user. Since the transformation from UML to Alloy is not bidirectional [34] (e.g. the multiplicity constraints of a UML association are mapped to an Alloy *fact*, but an Alloy *fact* does not necessarily correspond to UML association multiplicity constraints), we need to employ the technique presented in the previous section to carry out reverse instance transformation. How this technique is used in UML2Alloy, is described in the next section with the help of an example.

In order to take advantage of the analysis capability of the Alloy language, for the designer who wishes to use *only* UML, we will employ the tracing techniques presented in the previous section. Figure 6 shows the proposed solution, in terms of Alloy. Given a UML2Alloy transformation and trace, it is possible to transfer the outcome of the analysis into the UML form. In the case where the model is simulated to generate instances, we transform the instance back into UML Object Diagram, following the algorithm shown in

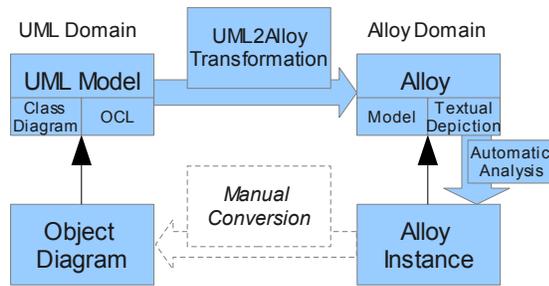


Fig. 5. UML2Alloy Case Study: Problem of Reverse Instance Transformation

Figure 4. In this case, for every instance element in the Alloy Instance model, find the signature in the Alloy Model. From the Trace, find the source Class in the UML model of the Signature. Finally instantiate the Class using the data from the Alloy instance in a UML Object Diagram. If repeated for all elements in the Alloy instance, an Object Diagram representation of the Alloy instance is created. When there is an inconsistency found in the model by Alloy Analyzer, the position in the text model is given. This position can be traced back to the cause in the UML, using the same algorithm.

In the next section contains a brief introduction to the Alloy language. Following this, there is an example model and model transformation in UML2Alloy where the outcome of the analysis are transferred back into UML form.

A. The Alloy Language

In this section we present a brief overview of the Alloy language [18] and supporting tool, the Alloy Analyzer. The Alloy language was designed for automatic reasoning and analyse of software systems. The five high-level constructs (termed paragraphs) in the language are Signatures, Facts, Predicates, Functions and Assertions. Multiples of these are used in the construction and analyse of a model. There is also the Run sentence, required to initiate analysis of a model. For the definition of models, Signatures, Functions and Facts in analogy to UML are classes, methods and constraints respectively. For analysis there are Predicates, used to simulate the models' properties with sample instances. Assertions are used in analysis in attempt to (in)validate a particular property of a model by producing counter-examples. The notion of model scope is important as Alloy's underlying logic is First-Order. As first order logic is undecidable in the general case, a scope must be set at execution time to bound the analysis space.

Using the Alloy Analyzer, a model in the Alloy language automatically be analysed and simulated. Models in the Alloy languages are translated into a series of boolean expressions. This form is suitable for analysis via off-the-shelf satisfiability (SAT) solvers. Simulation of the Alloy model is used to generate valid instance models.

The process is that the Alloy Analyzer automatically translates an Alloy model to a SAT formula that can be analysed by SAT solvers. In the case of assertion, the statement is negated and the Alloy Analyzer tries to find an instance of the model

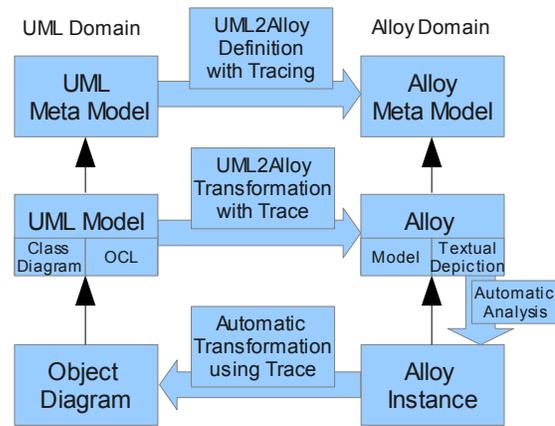


Fig. 6. UML2Alloy Case Study: Proposed Solution Using Tracing

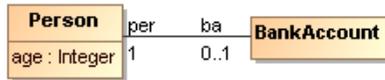
that conforms to the negated statement. If it can find one, this is a counterexample, if it cannot find one, the assertion may be valid. In the case of simulation, it searches for cases for an expression evaluates which to true, this expression will encode an arbitrary instance of the model.

V. EXAMPLE

In this section we introduce the example model and transformation to demonstrate the tracing mechanism. The UML model (Figure 7) is transformed into the Alloy model (Figure 8) using the UML2Alloy transformation in SiTra. The first part of each model (Figures 7a, 8a) is consistent in either language; conforming instances can be created by hand in UML or automatically in Alloy. We use the first models (Figures 7a, 8a) to demonstrate reverse instance transformation. The second part of each model (Figures 7b, 8b) extends on the first valid part to add contradictory constraints, making the original model inconsistent. The inconsistent model has no valid instances in either language and this property of the models is used to demonstrate the reverse instance tracing, to discover the source of the inconsistency in the original UML. The textual Alloy models presented here are refactored slightly to aid readability and for purposes of brevity.

We use a two-class, UML model shown in Figure 7 as the example in this section. According to the model, one *Person* can be associated either one or no *Bank Accounts*. The *Person* class has a single integer attribute representing age. The first part of the model (Figure 7a) is legal UML, valid instance of this model can be created by hand, as UML Object Diagram. In the second (Figure 7b), invalidating part of the model, two OCL constraints are added to the model specifying purposely contradictory constraints about the age of a *Person*. Both parts of the models are transformed into the Alloy model, shown in Figure 8.

The Alloy model (Figure 8a) consists of two signatures (*sig*), *Person* and *Bank Account*, here the keyword *some* enforces the existence in an instance model. *Person* has the two atoms, *age* is a relation to one integer and *ba* is a relation one or no (*lone*) *Bank Account*. The first fact paragraph *fact*{



(a) Valid Elements of Model

```

context Person inv first:self.age>20
context BankAccount inv second:self.per.age<18
  
```

(b) OCL Constraints (Introducing Contradiction)

Fig. 7. Example UML Model

$per = \sim ba$ }, signifies that ba and age form a single and the same relation from *Bank Account* to *Person* and vice-versa. Simulation of the first part of the model should yield sample instances using the Alloy Analyzer. In the second part of the Alloy model (Figure 8b) two facts constrain the model in contradictory way; that is the age of a Person must be both less than eighteen and greater than twenty.

In this example, mapping between the two models is performed by the UML2Alloy tool automatically, we explain the mapping here for the sake of clarity. The UML classes *Person* and *Bank Account* are transformed into signatures of the same name. The *age* attribute in person becomes the integer field *age* in the *Person* signature. The association is transformed into fields in the respective signatures, named *per* and *ba* and facts representing multiplicity. The OCL constraints are transformed into facts in the Alloy model. In UML2Alloy the above is a two step process, the first transformation is to a MOF like model of Alloy which is mapped via a one-to-one transformation into the Alloy textual form.

In the following two sections, we use the example to demonstrate the traceability from two perspectives. Firstly, given valid models (Figures 7a, 8a), transform the Alloy instances back into UML form. Secondly, given an invalid UML model (Figure 7), trace back the element(s) causing the inconsistency in the UML, as discovered in the Alloy model (Figure 8a) Alloy Analyzer.

A. Example One: Reverse Instance Transformation

The Figure 9, shows the output of analysis from the Alloy Analyser in XML form, our method will automatically create an Object Diagram form of this instance. It Depicts a sample of the output Analysis of the Alloy Analyser. In this partial sample, there is an instance (atom) of the *Person* signature from the Alloy model, labelled as *Person\$0*. There are also two fields, that form tuples (relations) from *Person\$0* to an integer (5) for the *age* atom. The second *BankAccount\$2* via the *ba* atom, instances of the *Person* signature's atoms.

Reverse instance transformation, this context, is taking an Alloy instance model and transforming it to back into the original UML form. Although this may seem a straight forward problem, possibly solved by transformation, often times there is a difference in semantic expressiveness of elements between the source and destination language. Reverse instance transformation is not possible in all cases using only the source and destination model as the transformation can change from a specific to a general one.

```

some sig Person{
  age : one Int,
  ba : lone BankAccount}
  
```

```

some sig BankAccount{
  per : one Person}
  
```

```

fact{ per = ~ba }
fact{per in BankAccount lone->one Person}
fact{ba in Person one->lone BankAccount}
  
```

(a) Alloy Model, Valid Segment

```

fact{all p : Person | int p.age > 20}
fact{all b : BankAccount | int b.per.age < 18}
  
```

(b) Alloy Model, Contradictory Segment

Fig. 8. Alloy Model Resulting from Transformation of Figure 7, using UML2Alloy

We demonstrate the issue in the example in Figure 9 where it is not possible to know the origin (in the UML model) of the Alloy instance atom marked *age*. This is because an Alloy field could have been mapped from a UML attribute or a UML association. The issue becomes more apparent in the instance (Figure 9) where the original association between *Person* and *Bank Account* has become a pair of Alloy atoms *ba* and *per*. In Alloy, these atoms are semantically equivalent to the *age* atom and thus indistinguishable for instance creation, without trace information.

For this example we assume the Model transformation of the consistent UML model in Figure 7a into the consistent Alloy model Figure 8a. We shall create a UML representation of the instance (Figure 9), which was created by analysis of the Alloy model (Figure 8a). The method of finding the origin in UML using the algorithm in the case of Figure 4 is as follows. Take the instance tuple *age* that maps the atom *Person\$0* (Figure 9) to the *Int* 5, find the parent element in the Alloy model (Figure 8). The parent element is the atom *age*, in the signature *Person*. The origin of this element is found by querying the trace, the origin here is is the attribute *age* of *person*, in the source UML model (Figure 7). At this point, it is possible to instantiate the *age* attribute in the UML Object Model of *Person\$0*, using data from the Alloy tuple i.e. with a value of 5.

B. Example Two: Model Inconsistency Tracing

Due to design errors it may be possible to define an inconsistent model, such as the model shown in Figure 8b. Discovering the cause of design inconsistencies in a model is highly desirable. A UML model can be transformed into Alloy and simulated to find inconsistency using the Alloy Analyzer. An inconsistent model in Alloy will result in no instance being generated. However the offending sentence(s) will be know from the analysis in the Alloy model, but not the original UML. The Alloy Analyser uses UnSat Core [17] feature of the SAT solver to find the line and column numbers of the inconsistent portion. This analysis does not

```

...
<sig name="Person" extends="univ">
  <atom name="Person$0"/>
</sig>

<field name="age">
  <type> <sig name="Person"/> <sig name="Int"/> </type>
  <tuple> <atom name="Person$0"/> <atom name="5"/> </tuple>
</field>

<field name="ba">
  <type> <sig name="Person"/> <sig name="BankAccount"/> </type>
  <tuple> <atom name="Person$0"/> <atom name="BankAccount$2"/> </tuple>
</field>
...

```

Fig. 9. Example One: Partial XML Output from Alloy Analyzer

apply directly to the UML model, however our method is able to trace the source of an element in the original UML and thus the originating cause of inconsistency. UML2Alloy allows inconsistency to be uncovered via analysis in Alloy and our method uncovers the root cause of the inconsistency in the original model, using the trace.

Our solution will trace the root cause of inconsistency in a UML model, after it has been transformed and analysed using Alloy. The method is as follows, assume the Model Transformation from the inconsistent UML in Figure 7 (combined) to the Alloy representation of the same in 8. The Alloy Analyser will uncover the inconsistency and the place (line(s), column(s)) at which the problem occurs in the Alloy model. To find the cause of the inconsistency in UML, first identify the element in the model causing the error. In this case, the UnSat Core gives the expressions from the two facts in 8b as the cause. To find the error in the UML Model, query the trace using the given ranges returns the result in OCL form, shown in Figure 10 and the location in the source model. This will highlight the two conflicting statements in the UML form for the developer to maneuver as necessary. The strength of the method is apparent in a large model with potentially many complex or similar constraints, as it is possible to pinpoint precisely which statement caused the inconsistency. We have shown that using our method to reverse transform elements in the destination model, it is possible to find the source of an inconsistency in the original UML model.

```

[self].age>20
[self].per.age<18

```

Fig. 10. Example Two: Result of Text to Model Tracing

VI. RELATED WORK

A common motivation for traceability in the literature has been to support chains of transformation, sequences of transformations with dependency, as found in [12], [36], [19]. The approach used in these methods is to add specific rules for traceability information and create a separate model of traceability as output of transformation. This model is then

used as input for further transformation stages. So the tracing approach for chains of transformations with dependency is fundamentally different from ours. In [19], an important discussion is made about generating trace information from a generic perspective based on an approach similar to [36], [12]. Our approach generates trace information automatically which is used internally in the context of the original transformation. We have demonstrated that it is possible to generate useful trace information automatically, without requiring specific rules in the original transformation.

The extension of the SiTra implementation to support tracing of model transformations, has been inspired by the QVT standard [29]. The QVT standard provides a number of methods to query the transformation traces, while the transformation is being executed (i.e. provides the ability to access target objects created from source objects and the inverse). Since the QVT specification does not provide any guidelines on how to implement the standard, the SiTra extension presented in this paper, which follows the QVT naming convention for methods, can be considered as an implementation of the tracing mechanism of the QVT standard.

Additionally the SiTra extension to provide tracing for model to text transformations is based on another OMG standard, the MOF Models to Text Transformation language specification [27]. In particular, the metamodel of the standard provides the ability to trace from which model element, a block of text has been created. An implementation of model to text tracing is presented in [26]. Our metamodel for model to text tracing is similar to the metamodel presented in [26]. However, these methods are more general than our extension to SiTra, since they address issues like synchronisation and change propagation between a model and its textual representation, if either the model or the textual representation changes. Our model to text transformation is simpler, since the generated Alloy textual model, is used as an intermediate step in order to be able to exploit the Alloy Analyzer API.

VII. CONCLUSION AND FUTURE WORK

This paper presents a implementation *Traceability*, a mechanism for recording the link between the source and target

model elements in Model Transformation Frameworks. Traceability is often used for the management of Model Transformation process for change propagation or debugging purposes. In this paper, Traceability produces the reverse transformation automatically at the instance level. Such applications of Traceability are essential for horizontal Model Transformations such as UML2Alloy.

The presented approach, which is inspired by QVT, is formulated as an algorithm. The SiTra framework is extended to a new version to do both model-to-model and model-to-text transformations, while the algorithm is implemented to allow traceability in both cases. The paper also reports on a case study based on the model transformation in UML2Alloy, which uses the new version of SiTra. Using a brief example, the process of Model Transformation carried out via UML2Alloy is described and the role of tracing mechanism is explained.

In the current implementation, only the binary transformation rules are traceable. So to handle ternary rules, they must be modified to create multiple binary rules. Discovery of methods of extension to ternary rules remains a subject for future research. There is a clear scope for extending the current framework to fulfill the original objectives of Traceability such as change propagation and debugging. This also remains an area for future research.

REFERENCES

- [1] David Akehurst, Behzad Bordbar, M. J. Evans, W. G. J. Howells, and Klaus D. McDonald-Maier. SiTra: Simple transformations in java. In Oscar Nierstrasz, Jon Whittle, David Harel, and Gianna Reggio, editors, *MoDELS*, volume 4199 of *Lecture Notes in Computer Science*, pages 351–364. Springer, 2006.
- [2] DH Akehurst, O. Uzenkov, WG Howells, and KD McDonald-Maier. Compiling UML State Diagrams into VHDL: An Experiment in Using Model Driven Development. *ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (formerly the UML series of conferences)*, Genova, Italy, 2007.
- [3] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. UML2Alloy: A Challenging Model Transformation. In G. Engels, B. Opdyke, D.C. Schmidt, and F. Weil, editors, *ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems*, volume 4735 of *LNCS*, pages 436–450, Nashville, USA, 2007. Springer.
- [4] J. Bézivin, F. Jouault, and D. Touzet. An Introduction to the ATLAS Model Management Architecture. *Research Report LINA,(05-01)*, 2005.
- [5] Lossan Bondé, Pierre Boulet, and Jean-Luc Dekeyser. Traceability and interoperability at different levels of abstraction in model transformations. In *Forum on Specification and Design Languages, FDL'05*, 2005.
- [6] Behzad Bordbar and Kyriakos Anastasakis. MDA and Analysis of Web Applications. In *Trends in Enterprise Application Architecture (TEAA) 2005*, volume 3888 of *Lecture notes in Computer Science*, pages 44–55, Trondheim, Norway, 2005.
- [7] Behzad Bordbar and Kyriakos Anastasakis. UML2Alloy: A tool for lightweight modelling of Discrete Event Systems. In Nuno Guimarães and Pedro Isaias, editors, *IADIS International Conference in Applied Computing 2005*, volume 1, pages 209–216, Algarve, Portugal, February 2005. IADIS Press.
- [8] Behzad Bordbar, Gareth Howells, Michael Evans, and Athanasios Staikopoulos. Model transformation from owl-s to bpel via sitra. In *ECMDA-FA*, pages 43–58, 2007.
- [9] Codagen Technologies Corp. Codagen architect.
- [10] Borland Software Corporation. Borland together architect.
- [11] K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Syst. J.*, 45(3):621–645, 2006.
- [12] J.R. Falleri, M. Huchard, and C. Nebut. Towards a traceability framework for model transformations in Kermeta. *ECMDA-TW Workshop*, 2006.
- [13] F. Fleurey, Z. Drey, D. Vojtisek, and C. Faucher. Kermeta language. *Reference manual*, Internet: <http://www.kermeta.org/docs/kermeta-manual.pdf>, 2006.
- [14] F. Flore. MDA: The Proof is in Automating Transformations Between Models, 2003.
- [15] Geri Georg, Indrakshi Ray, Kyriakos Anastasakis, Behzad Bordbar, Manachai Toahchoodee, and Siv Hilde Houmb. An Aspect-Oriented Methodology for Developing Secure Applications. *Information and Software Technology. Special Issue on Model Based Development for Secure Information Systems*. Accepted for publication.
- [16] Mark Hibberd, Michael Lawley, and Kerry Raymond. Forensic debugging of model transformations. In *MoDELS*, pages 589–604, 2007.
- [17] D. Jackson, M.I.T.I. Shlyakhter, and M. Sridharan. alloy: a logical modelling language. *Proceedings of 3rd International Conference of B and Z Users (ZB 2003): Formal Specification and Development in Z and B, Turku, Finland*, page 1, 2003.
- [18] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, London, England, 2006.
- [19] F. Jouault. Loosely Coupled Traceability for ATL. *Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability, Nuremberg, Germany*, 2005.
- [20] S.R. Judson, D.L. Carver, and R. France. A Metamodeling Approach to Model Refactoring. *Submitted to UML03*, 2003.
- [21] Soon-Kyeong Kim. *A Metamodel-based Approach to Integrate Object-Oriented Graphical and Formal Specification Techniques*. PhD thesis, University of Queensland, Brisbane, Australia, 2002.
- [22] I. Kurtev, J. Bézivin, and M. Aksit. Technological Spaces: an Initial Appraisal. *CoopIS, DOA*, 2002, 2002.
- [23] M. Lawley and J. Steel. Practical Declarative Model Transformation With Tefkat. *MoDELS Satellite Events*, pages 139–150, 2005.
- [24] G.A. Lewis and L. Wrage. Model Problems in Technologies for Interoperability: Model-Driven Architecture. Technical report, Carnegie Mellon University, Software Engineering Institute, 2005.
- [25] J. Miller and J. Mukerji. Mda guide version 1.0.1. Technical report, OMG, 2003.
- [26] Gøran K. Olsen and Jon Oldevik. Scenarios of traceability in model to text transformations. In *ECMDA-FA*, pages 144–156, 2007.
- [27] OMG. *OMG: MOF Model to Text Transformation Language*. OMG, 2004.
- [28] OMG. *Meta Object Facility (MOF) 2.0 Core Specification*. OMG, 2004.
- [29] OMG. *MOF QVT Final Adopted Specification*. Object Modeling Group, June 2005.
- [30] Mark Richters and Martin Gogolla. On formalizing the uml object constraint language ocl. In *ER*, pages 449–464, 1998.
- [31] Louis M. Rose, Richard F. Paige, Dimitrios S. Kolovos, and Fiona Polack. The epsilon generation language. In *ECMDA-FA*, pages 1–16, 2008.
- [32] Shane Sendall and Wojtek Kozaczynski. Model transformation: The heart and soul of model-driven software development. *IEEE Software*, 20(5):42–45, 2003.
- [33] Ilya Shlyakhter, Robert Seater, Daniel Jackson, Manu Sridharan, and Mana Taghdiri. Debugging overconstrained declarative models using unsatisfiable cores. In *Proceedings of the 18th IEEE International Conference on Automated Software Engineering, Montreal, Canada*, pages 94–105. IEEE Computer Society, 2003.
- [34] T. Stahl, M. Voelter, and K. Czarnecki. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.
- [35] OMG UML. 2.0 superstructure final adopted specification. *OMG Document reference ptc/03-08*, 2, 2003.
- [36] B. Vanhooff and Y. Berbers. Supporting Modular Transformation Units with Precise Transformation Traceability Metadata. *ECMDA-TW Workshop, SINTEF*, pages 15–27, 2005.
- [37] M. Volter. OpenArchitectureWare 4. *Bericht, openArchitectureWare*, 2007.

Dataflow/Actor-Oriented language for the design of complex signal processing systems

Christophe Lucarz^{*}, Marco Mattavelli^{*},
 Matthieu Wipliez[†], Ghislain Roquier[†], Mickaël Raullet[†],
 Jörn W. Janneck[‡], Ian D. Miller[‡] and David B. Parlour[‡]
^{*}Ecole Polytechnique Fédérale de Lausanne (Switzerland)
[†]IETR Laboratory - UMR CNRS 6164 - Rennes (France)
[‡]Xilinx Inc. - San Jose (U.S.A)

Abstract—Signal processing algorithms become more and more complex and the algorithm architecture adaptation and design processes cannot any longer rely only on the intuition of the designers to build efficient systems. Specific tools and methods are needed to cope with the increasing complexity of both algorithms and platforms. This paper presents a new framework which allows the specification, design, simulation and implementation of a system operating at a higher level of abstraction compared to current approaches. The framework is based on the usage of a new actor/dataflow oriented language called CAL. Such language has been specifically designed for modelling complex signal processing systems. CAL data flow models expose the intrinsic concurrency of the algorithms by employing the notions of actor programming and dataflow. Concurrency and parallelism are very important aspects of embedded system design as we enter in the multicore era. The design framework is composed by a simulation platform and by Cal2C and CAL2HDL code generators. This paper described in details the principles on which such code generators are based and shows how efficient software (C) and hardware (VHDL and Verilog) code can be generated by appropriate CAL models. Results on a real design case, a MPEG-4 Simple Profile decoder, show that systems obtained with the hardware code generator outperform the hand written VHDL version both in terms of performance and resource usage. Concerning the C code generator results, the results show that the synthesized C-software mapped on a SystemC scheduler platform, is much faster than the simulated CAL dataflow program and approaches handwritten C versions.

I. INTRODUCTION

With the unbounded increase of signal processing systems complexity made possible by both the advances in algorithm theory and by new generations of silicon technology, digital systems designers need new tools for the design of efficient systems employing "reasonable" design resources. Increasing the level of abstraction has always been a solution to appropriately handle the increasing complexity of systems design. Transistors, gates, VHDL and Intellectual Property (IP) blocks are example of different abstractions layers which have been successfully introduced in the past with the attempt of easing the design of more and more complex systems. In the authors opinion the current new challenge is not only to add a new abstraction layer, but also to close the gap between the specification and the implementation layers. Such gap, in the recent years, has been tried to be filled by using C/C++ reference implementations of the specifications. However, the

path from such specifications expressed as generic C/C++ reference SW to gate design has shown to be harder to be efficiently accomplished. One of the main reasons of such difficulty is the fact that C/C++ do not provide the operators that enable to naturally express parallelism, data flows and other fundamental architectural features without adding a huge amount of low level programming details. In other words the operators are defined at a level that is by, far too, low compared to the one at which a designer would like to express his architectural ideas. Also other approaches and methods aiming at closing the gap from specification to synthesis have yet to deliver on their promise. Several methodologies based on using UML, different variants of C/C++ (i.e streamC [1]) combined with SystemC/TLM library at different levels of the design flow are used. Some of them make use of exploration tools, simulators and code generators which can output hardware and software code specific to FPGA or other processor platforms, mainly using VHDL and C. Section II briefly reviews these different approaches and methods for building digital systems.

This paper presents a new design framework, currently in its early phase of development, based on CAL language, an actor and dataflow oriented language designed for the specification of complex signal processing systems. The new framework includes: a simulator of the system specified by CAL, a hardware generator for the direct synthesis of HDL and a software generator for the synthesis of C from CAL. The paper explains why and how this framework is a very promising approach for the design and development complex heterogeneous processing systems.

Section I introduces the CAL language, a dataflow and actor-based language that constitutes a very interesting candidate to support a design flow for designing complex signal processing systems. The major features of the new approach are simplicity, conciseness and expressiveness. Modelling systems in a concise and simple way is a good starting point, but at the same time enabling the automatic generation of efficient code is an extremely challenging step. Sections IV and V show how efficient hardware and software code can be generated directly from CAL language system descriptions.

Section VI discusses the reasons for which CAL language and the associated framework are a good approach to support a design flow for building complex heterogeneous systems.

Section VII concludes the paper.

II. RELATED WORK

This section provides a brief overview of the current design flows based on UML, SystemC, and C/C++. A comprehensive overview of the state of the art of system level modeling can be found in [2].

Although UML was originally conceived for modelling large software systems, Kukkala et al. [3] developed a design flow for multiprocessors Systems on Chip (SoC) by using UML 2.0 models as the starting point of the design flow. UML models (i.e. application, platform and mapping models) are written according to the experience of the designers. However, there is limited support for the elaboration of UML specifications. When building a system from monolithic C/C++ specifications (such as one of the the MPEG reference softwares for instance) UML models must be completely written by hand. This task is very time-consuming and error prone. Furthermore, according to [4], “*UML 2.0 lacks both a reference implementation and a human-readable semantic*”.

Modelling HW using a high level description languages such as SystemC can be a solution for representing functionality, concurrency, communication, software and hardware components at different (system) levels of abstraction. A language modelling complex systems should also provide a support for analysis and synthesis. SystemC is mainly used for simulation because it is not synthesizable in its whole generality. In practice, it can be reduced for ensuring the use of a synthesizable subset. The problem is that this subset of the language is placed at a quite low level of abstraction. Simulation capabilities at a high level of abstraction are an interesting feature of SystemC, but during the implementation step, the designer is forced to re-write the code using only the synthesizable subset. Such operation is a time consuming and error-prone task. In a nutshell, it is hard to use SystemC as a high level design language because in general it is not possible to implement directly systems just developing at high level the system specification.

Several tools propose a direct conversion of C code into VHDL. C-based methodologies for modelling systems lack concurrency and a concept of time. Hardware is inherently parallel and time is essential to represent its behaviour accurately, especially for real-time embedded systems. The main problems of using C as a Hardware Description Language are discussed in detail in [5] and [6]. In conclusion, the lack of intrinsic concurrency and the concept of time as well as the way communication mechanisms are written in imperative languages are the main limitations and drawbacks of C or similar languages for hardware representation.

III. CAL A LANGUAGE FOR ALGORITHM AND ARCHITECTURE SPECIFICATION

The more important features that system designers would like to find in a design language would be the capability of representing both algorithm and architecture at a single level of abstraction. Unfortunately, it is difficult to specify an algorithm

at a high level of abstraction and at the same time being able to deduce from such description an efficient low level representation. How can high level and low level constructs be combined into a single language? Furthermore, such language should support full simulation of the system behaviour. Thus, designers have been forced to use several languages to represent complex algorithms at different levels of abstraction during the entire design flow. Generic C, architectural C, SystemC, synthesizable SystemC is a possible first stage of a design methodology. The main drawback of such multistage approaches is how to implement the conversions between all these languages avoiding resource consuming hand re-writing [4]. Compilers often support only a subset of the languages, making the different conversions between the languages a headache! Thus, the reduction of the number of languages and abstraction levels composing a design flow is a fundamental issue. Ideally, the unique language transformation should be a direct translation from the high level language down to an implementation language such as C, VHDL or Verilog.

Abstract languages do not often support simulation capabilities and architectural representations are available at too low levels of abstraction. Standard imperative languages do not express easily parallelism. These facts reduce considerably the set of possible candidates. The new actor/dataflow oriented language called CAL has been specifically developed so as to support the features discussed above.

A. CAL language

CAL is a dataflow and actor oriented language that has been recently specified and developed by one of the authors of this paper as a subproject of the Ptolemy project at the University of California at Berkeley. The final CAL language specification has been released in December 2003 [7]. CAL describes algorithms by using a set of encapsulated dataflow components called *actors* communicating with each others. An actor is a modular component that encapsulates its own state. The state of any actor is not shareable with other actors. Thus, an actor cannot modify the state of another actor. The only interaction an actor has with another actor is only thought inputs and outputs ports. The behaviour of an actor is defined in terms of a set of *actions*. The operations an action can perform are to consume (read) input tokens, to modify internal state, to produce output tokens. The topology of the connections between actors input and output ports constitute what is called a network of actors. It is expressed by using an XML dialect known as network language (NL) that also includes the possibility to includes attributes (i.e. parameters) that may be different for the instantiation of the same (parametric) actor in a network of actors. Each action of an actor defines the kind of transitions that internal states can undergo and actions can be fired under specific conditions: (1) the availability of input tokens, (2) the value of input tokens, (3) the state of the actor or (4) the priority of that action. In an actor, the operations are executed sequentially. The execution of actions follows the following cycle:

- 1) Determine, for each action, whether it is enabled, by

testing all the conditions specified in that action. It depends on the availability of token(s) at the requisite input(s), the value of input tokens, the state of the actor and the priority of each action.

- 2) If one or more actions are enabled, pick one of them to be fired next;
- 3) Fire that action.
- 4) Go to (1).

The transitions of an actor are purely sequential: actions are fired one after another. At the network level, the actors are completely independent and can work concurrently, each one executing their own sequential operations. Figure 1 illustrates a CAL model in general and the reader can find examples of CAL actors in figures 4 and 5(a).

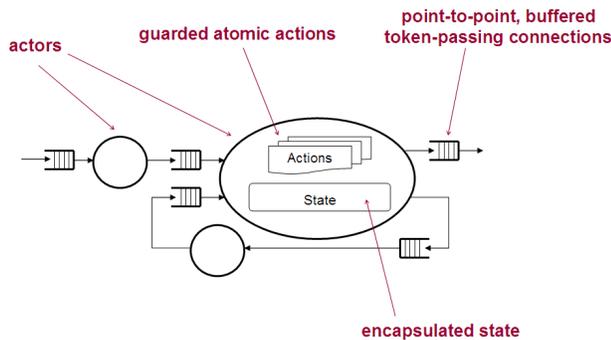


Fig. 1. Illustration of a CAL model

The selection order and the firing conditions for actions form the core of the design of an actor. CAL provides a number of constructs for describing action selection, which include *guards* (conditions on the values of input tokens and/or the values of actor state variables), a *Finite State Machine* and *priorities* (actions may be related to each other by a partial priority order). The order of execution of the actors is in general not known *a priori*. CAL provides a great flexibility to schedule action execution according to the particular requirements and constraints of the targeted device chosen for the final implementation.

CAL language very naturally allows also hierarchical system design. Each actor can be specified as a network of actors. This approach facilitates modularity, where the internal specification of any actor can be modified without impacting other actors.

B. Goals and features of CAL language

Ease of use CAL is a true programming language and not an intermediate format to automatically generate code. The notation is convenient to write, with consistent syntax rules, keywords, and structures.

Minimal semantic core CAL language is built on a very small set of semantic concepts. It simplifies compiler construction when transforming any program into an equivalent program in the core language. Thus for the compilation to any given platform, a code generator is needed for the specific implementation language.

Implementation independence and retargetability The first target for CAL actors was the Ptolemy II platform [8]. A complete framework (Open DataFlow) is currently being developed for simulating CAL networks and for generating hardware and software code [9]. OpenDF is built under the Eclipse environment. It makes these tools available for a large set of operating systems. CAL models are technology and architecture agnostic. Thus, it makes possible to design and simulate models very quickly using [9]. Designers do not need special hardware or libraries to design their own system in CAL. The implementation of CAL models is done by means of appropriate hardware and software code generators described in more details in the following sections..

C. Framework infrastructure

Figure 2 presents the general framework infrastructure. Once the CAL model is defined, the user can simulate it using the OpenDF simulator [9]. The user can generate software code and hardware code from the CAL model. These code generators are detailed in section IV and V respectively.

IV. SOFTWARE SYNTHESIS

A. Semantics of CAL dataflow

The system behavior of a dataflow program is determined by the interactions between actors (i.e. exchange of data tokens). Such interactions are governed by a Model of Computation (MoC) that defines which scheduling policies can be used to fire actors. The CAL language is not related to any particular dataflow MoCs. Indeed, several forms of dataflow exist to interpret the network from the general dataflow process network (PN) [10] model with multiple firing rules to the more restrictive synchronous dataflow (SDF) one [11], [12]. CAL extends the model in [10] by :

- 1) state within actors,
- 2) multiple overlapping (non-joinable in the parlance of [10]) firing rules, and
- 3) priorities among firing rules.

CAL actors often contain multiple actions and priorities, FSM or guards that lead to state-dependent or conditional execution. Therefore most of the CAL actors in the library are closer to the PN model which is then chosen (as a first milestone) for developing the tool described in this paper.

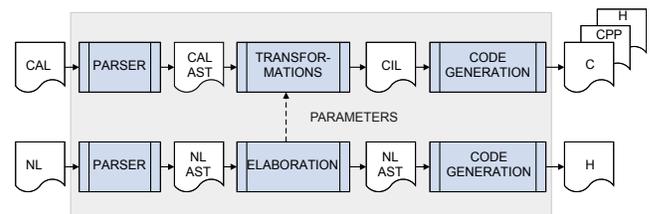


Fig. 3. Cal2C compilation process

When the PN model is chosen to interpret CAL networks, any environment which supports multithreading may be chosen for implementation. For instance, it can be done

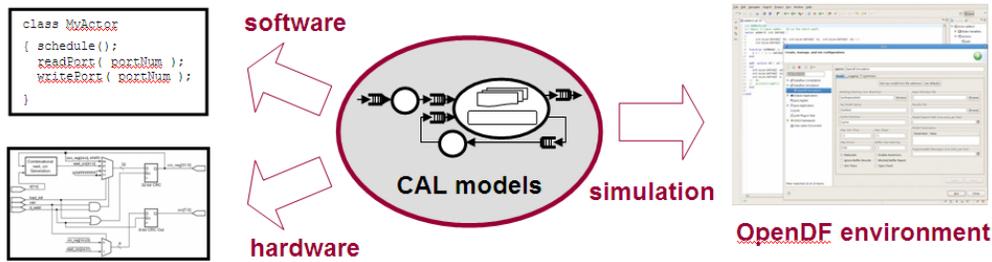


Fig. 2. The framework infrastructure

using POSIX threads by translating CAL actors into threads and by replacing connections with FIFOs. However, it is necessary to develop a scheduler and low-level considerations such as communication or actual scheduler implementation, make this solution time-consuming and error prone. Another approach for PN implementation is provided by the SystemC¹ standard whose simulation environment permits high-level programming, well-adapted to functional verifications. A PN-oriented SystemC model is expressed as a network of modules communicating with each other via blocking FIFOs (with an additional way to support token availability). Note that Cal2C does not intend to use SystemC as an hardware description language, but only as a convenient PN modeling and simulation environment. Moreover, this work is a first step to an efficient "pure C" code generation (closer to an SDF model than a PN one). In short, the software synthesis from a network of CAL actors produces several files as illustrated in Figure 3. Each NL network is translated into a header file where FIFOs, modules and sub-networks are instantiated and connected. CAL actor translation is done in 2 different parts: the actor code (actions, functions, procedures) to express the functionality and the action scheduler (priorities, FSMs, guards) to control the execution. Finally an additional file is created to instantiate the top network and to launch SystemC simulation.

B. C code generation: Actors

Actor code translation: Translating the CAL actor code produces a single C file wherein functions, procedures and actions are translated. however, C language and compilers impose limitations on the translated code. For example, (1) distinct functions should have different names to avoid linking problems, even if they are originally placed in different actors. Another challenge is (2) the difference of programming paradigm between the source and the target language: CAL allows functional constructs that have no direct equivalent in C. The action translation process starts with an Abstract Syntax Tree (AST) issued from the CAL source code, and modifies it as needed to satisfy the previous requirements. Function names are prefixed with the actor name to prevent any potential name clashes; actor parameters are replaced by their values (when constant) or transformed to local variables otherwise; actions are converted to functions where input and

```
ac: action AC:[i] => OUT:[ saturate( o ) ]
var
  int (size=SAMPLE_SZ) v =
    ( quant * ( lshift( abs(i), 1) + 1 ) ) - round,
  int (size=SAMPLE_SZ) o =
    if i = 0 then 0 else if i < 0 then -v else v end end
do
  count := count + 1;
end
```

(a) CAL "ac" action in the "Inversequant" actor

```
void Inversequant_ac(
  struct Inversequant_variables *_actor_variables ,
  int i , int *out )
{ int v, o ;
  int _call_6, _call_9;
  int _if_7, _if_8 ;
  _call_6 = Inversequant_fun_abs(_actor_variables, i);
  v = _actor_variables->quant * (( _call_6 << 1 ) + 1)
    - _actor_variables->round;
  if ( i == 0 ) {
    _if_8 = 0;
  } else {
    if ( i < 0 ) {
      _if_7 = - v;
    } else {
      _if_7 = v;
    }
    _if_8 = _if_7;
  }
  o = _if_8;
  (_actor_variables->count) ++;
  _call_9 = Inversequant_fun_saturate(_actor_variables, o);
  *out = _call_9;
}
```

(b) C translation of the "ac" action in the "Inversequant" actor

Fig. 4. C translation of a CAL action

output patterns become parameters. Finally, actor declarations are ordered by dependencies between locals, so that a variable or a function is defined before being used. At this point, the Cal2C generator converts the AST to λ -calculus, applies Damas-Milner \mathcal{W} -algorithm [13] to it, and augments the AST with the type information returned. Types are necessary for correct C code generation, and type-dependent transformations. Functions that return lists are "in-lined", and list sizes are computed. The transformed CAL AST is expressed in the C Intermediate Language (CIL) [14], where CAL functional constructs are replaced by imperative ones. C code is generated by calling the pretty-printer module included in the framework.

An example of the CAL actor code translation process is illustrated on figure 4. Figure 4(a) reports an "action"

¹<http://www.systemc.org>

extracted from the *Inversequant* actor of the RVC reference MPEG4-SP decoder, and figure 4(b) reports the corresponding generated C code. The resulting code presents a function whose name is composed of the actor name and the action name (requirement (1)). Its parameters are the same of the action's, with an additional pointer to a structure containing the actor variables. The **if** expression has been transformed to assignments of temporary variables (**_if_7**, **_if_8**) (requirement (2)). As a matter of fact, function calls have also become assignments of temporary variables (**_call_6**, **_call_9**) because CIL semantics requires it. The action output expression is translated as a pointer parameter whose value is written at the end of the C function. The synthesized C code shown in figure 4(b) is quite close to the original CAL code and results also reasonably readable by "humans".

Action scheduling: An action scheduler is created to control the action selection during execution. Priorities, guards, token consumption rates and FSM have to be translated to this end. Determining the overall order of action execution is required to have a consistent evaluation of actions that can be fired. Priorities are resolved by sorting actions in a total order and by adding a *if-then-else* statement around actions wherein the condition is given by the availability of input tokens and the guards conditions. FSMs are resolved using *switch-case* statement. Finally, the generated file consists of a thread with an infinite loop wherein its body consists of the result of the previous transformations and actions are replaced with their corresponding C functions.

For instance, a downsampler by N is illustrated figure 5(a). It could be written in different and also simpler forms, but in this form the actor enables to highlight key features of the action scheduling translation. The synthesized C++ code is illustrated in figure 5(b).

C. SystemC code generation: Networks

Expressing a network of actors specified by NL in SystemC is relatively straightforward: actor or network instantiations are transformed into module instantiations. There are two semantic differences however: FIFO channels, while implicit in NL, must be explicitly created in SystemC. Broadcast of data from a source to several sinks is transparent in NL, but requires additional logic in SystemC, namely a generic broadcast module.

D. Generation of C code from a CAL description: IDCT and MPEG-4 Simple Profile decoder

So as to validate the correctness of the Cal2C code generation, the first case study is a two-dimensional inverse DCT. The IDCT is a component of all MPEG standard video decoders and is fully specified by the new Finite Precision IDCT Specification [15] based on [16]. The algorithm consists of applying one-dimensional IDCT along the row and column axis of an 8×8 pixel block. The network is composed of 2 input ports, 1 output port and 5 different actors; one actor can be instantiated several times in NL. Incoming tokens from *IN* port are inverse-quantized coefficient and a token from

```

actor downsampler (N) In => Out :
  count := 1;
  pass: action In: [x] => Out: [x] end
  done: action => guard count = N do count := 1; end
  skip: action In: [x] => do count := count + 1; end

  schedule fsm pass:
    copy ( pass ) --> discard;
    discard ( done ) --> copy;
    discard ( skip ) --> discard;
  end
  priority
    done > skip;
  end
end

```

(a) CAL downsampler

```

struct downsampler_vars {
  int count;
  int N;
};
void downsampler::process() {
  int fsm_state, _call_6, _call_7, _out_1;
  struct downsamplerN_vars _actor_vars;
  _actor_variables.count = 1;
  fsm_state = 1;
  while (1) {
    switch (fsm_state) {
      case 1:
        _call_6 = In->get();
        downsampler_pass(&_actor_vars, _call_6, &_out_1);
        Out->put(_out_1);
        fsm_state = 2;
        break;
      case 2:
        if (_actor_vars.count == _actor_vars.N) {
          downsampler_done(&_actor_vars);
          fsm_state = 1;
        } else {
          _call_7 = In->get();
          downsampler_skip(&_actor_vars, _call_7);
          fsm_state = 2;
        }
        break;
    }
  }
}

```

(b) Synthesized action scheduler

Fig. 5. Action scheduling (FSM) of CAL actor

SIGNED enables to specify to the "clip" actor if incoming coefficient are signed or not. The first row of Table I shows the number of files of the respective programs. In the code generation process, an actor first is converted into a C file, then into a C++ and finally into a header file. A network of actors simply is translated into a header file while the corresponding C++ file becomes the "main" file. The second row exhibits the corresponding source lines of code (SLOC). The testbed consists of applying a stimulus (streamed by a C-code reference software of MPEG-4 SP decoder) to the top network and verifying the response against an expected result (from the CAL description simulated compared to a "golden reference" streamed by the C-code reference software).

Another synthesized model of a more complex CAL dataflow network simulated with the Open Dataflow environment has also been used to validate the Cal2C tool. The compilation process has been successfully applied to the full MPEG-4 Simple Profile dataflow model written by the MPEG

IDCT	CAL	NL	C	C++	H
Number of files	5	1	5	6	6
Code Size (SLOC)	131	25	324	386	107

TABLE I

CODE SIZE AND NUMBER OF FILES AUTOMATICALLY GENERATED FOR THE IDCT

RVC working group. Table II shows that the synthesized C-software is faster than the simulated CAL dataflow program (20 frames/s instead of 0.15 frames/s), and close to real-time for a QCIF format (25 frames/s) on a standard PC platform. It is interesting to note that the model is scalable: the number of macro-blocks decoded per second remains constant when dealing with larger image sizes.

MPEG4 SP decoder	Speed kMB/S	Code size kSLOC
CAL simulator	0.015	3.4
Cal2C	2	10.4

TABLE II

MPEG4SP DECODER SPEED AND SLOC

The MPEG4 SP dataflow program is composed of 27 actors. An actor composing a network of actors can be instantiated several times. For instance there are 42 actor instantiations in the MPEG-4 SP dataflow model. The number of SLOC generated is shown in Table III. All of the generated files are successfully compiled by gcc. For instance, the “Parser-Header” actor inside the “Parser” network is the most complex actor with multiple actions. The translated C-file (with actions and state variables) includes 1043 SLOC for actions and 1895 for action scheduling. The original CAL file contains only 962 lines of code as a comparison.

MPEG4 SP decoder	CAL	NL	C	C++	H
Number of files	27	9	27	28	36
Code Size (kSLOC)	2.9	0.5	5.8	3.7	0.9

TABLE III

CODE SIZE AND NUMBER OF FILES AUTOMATICALLY GENERATED FOR MPEG4 SP DECODER

V. HARDWARE SYNTHESIS FROM CAL DATAFLOW MODELS

In the current version of the HDL generator, each actor is translated separately into HDL and is connected with FIFO buffers in the resulting RTL descriptions. Consequently, no cross-actor optimizations are employed at the current level of development of the tool.

If two actors connected in this manner are specified to belong to different clock domains, an asynchronous FIFO implementation is selected, otherwise a synchronous FIFO is used for compactness of the implementation. Actors interact with FIFOs using a handshake protocol, which allows them to detect when a data token is available or when a FIFO is full.

The translation of each CAL actor into a hardware description follows a three-step process: (a) Instantiation, (b) Precompilation and (c) RTL code generation.

A. Instantiation

The elaboration of the network structure yields a number of actor *instances*, which are references to CAL actor descriptions along with actual values for its formal parameters. From this, instantiation computes a *closed* actor description, i.e. one without parameters, by moving the parameters along with the corresponding actual values into the actor as local (constant) declarations. It then performs constant propagation on the result.

B. Precompilation

After some simple actor *canonicalization*, in which several features of the language are translated into simpler forms, precompilation performs some basic source code transformations to make the actor more amenable to hardware implementation, such as e.g. inlining procedures and function calls. Then the canonical, closed actors are translated into a collection of communicating threads.

In the current implementation, an actor with N actions is translated into $N + 1$ threads, one for each action and another one for the *action scheduler*. The action scheduler is the mechanism that determines which action to fire next, based on the availability of tokens, the guard expression of each action (if present), the finite state machine schedule, and action priorities.

To facilitate backend processing, for both hardware and software code generation, the threads are represented in static single-assignment (SSA) form. They interact with the environment of the actor through asynchronous token-based FIFO channels. Their internal communication happens through synchronous unbuffered signals (this is, for instance, how the scheduler triggers actions to fire, and how actions report completion), and they also have shared access to the state variables of the actor.

C. RTL code generation

The final phase of the translation process generates an RTL implementation (in Verilog) from a set of threads in SSA form. The first step simply substitutes operators in expressions for hardware operators, creates the hardware structures required to implement the control flow elements (loops, if-then-else statements), and also generates the appropriate muxing/demuxing logic for variable accesses, including the Φ elements in the SSA form.

The resulting basic circuit is then optimized in a sequence of steps.

- (a) **Bit-accurate constant propagation.** This step eliminates constant or redundant bits throughout the circuit, along with all wires transmitting them. Any part of the circuit that does not contribute to the result will also be removed, which roughly corresponds to dead code elimination in traditional software compilation.

- (b) **Static scheduling of operators.** By default, operators and control elements interact using a protocol of explicit activation—e.g., a multiplier will get triggered by explicit signals signifying that both its operands are available, and will in turn emit such a signal to downstream operators once it has completed multiplication. In many cases, operators with known execution times can be scheduled statically, thus removing the need for explicit activation and the associated control logic. In case operands arrive with constant time difference, a fixed small number of registers can be inserted into the path of the operand that arrives earlier.
- (c) **Memory access optimizations.** Arrays are mapped to Block RAMs (BRAM) for FPGA implementation. These usually small RAM blocks (typically 18 kBits) are distributed across the FPGA, and can be ganged up to form larger memories, or a number of small arrays may be placed into one BRAM. Furthermore, BRAMs usually provide two or more ports, which allows for concurrent accesses to the same memory region. Based on an analysis of the sizes of arrays and the access patterns, the backend maps array variables to Block RAMs, and accesses to specific ports.
- (d) **Pipelining, retiming.** In order to achieve a desired clock rate, it may be necessary to add registers to the generated circuit in order to break combinatorial paths, and to give synthesis backends more opportunity for retiming.

	Size slices, BRAM	Speed kMB/S	Code size kLOC	Dev. time MM
CAL	3872, 22	290	4	3
VHDL	4637, 26	180	15	12
Improv. factor	1.2	1.6	3.75	4

Fig. 6. Hardware synthesis results for an MPEG-4 Simple Profile decoder. The numbers are compared with a reference hand written design in VHDL.

Figure 6 shows the quality of result produced by the RTL synthesis engine for a real-world application, in this case an MPEG-4 Simple Profile video decoder. Note that the code generated from the high-level dataflow description actually outperforms the VHDL design in terms of both throughput and silicon area for a FPGA implementation.

VI. DISCUSSION

In this paper it has been shown that CAL, at the same abstraction level, can yield system specifications for direct SW and HW generation. This is made possible by several interesting properties which not only provide appropriate answers at the problem of finding a language that specify systems at high level, but that can also be extremely useful for facing the challenges of next generation digital systems.

Expressing concurrency The intrinsic capability of CAL operators and construct to describe and easily fit concurrent problems make CAL actor-modeling an excellent fit for system

design of parallel algorithms and streaming or data dominated applications.

Compactness The MPEG-4 Simple Profile decoder has been manually implemented in different languages such as C/C++, VHDL and CAL. The full implementation is composed of approximately 4000 lines for CAL, 15000 lines for VHDL, 4100 for an optimized version in C/C++. It shows how concise CAL is, but at the same time concurrency and data dependencies are clearly exposed and by raising the level of abstraction of constructs and operators, CAL needs less lines of code to fully describe a given algorithm. Conversely CAL is not overloaded with implementation details and such feature is a clear advantage. CAL focuses only on the description of the algorithm itself and how data is generated and consumed by the different components. Implementation details such as scheduling of the operations are let to code generators.

Analysis CAL language allows the analysis of an actor and networks of actors. For the definition of the actors, CAL provides statically analyzable information about their behavior, such as the number of tokens it produces and consumes in each firing, the necessary conditions for their firing, on what depend those conditions... These information are very useful for effectively schedule, compose, and implement those actors in the final implementations.

Portability CAL specifies algorithms and defines their associated data flow models in a concise and clear way. CAL models are completely independent from final implementation. Thanks to this independency, it eases the integration, exchange and the development of actors. Different implementation for several targets can be built easily from these models. The encapsulation property of the actors is very convenient: global variables do not exist, making the integration of external actors (not written by the designer himself) in the design much easier.

Simplicity of actor design CAL offers a compact, clear and precise semantics, which is tailored to the constraints of actor design and thus facilitates readability and maintainability of the actors. The ease of programming is also necessary to make the language accepted in the scientific community.

Hardware and Software code generation The CAL language is a good starting point for both hardware and software code generation.

The results presented in this paper and in [17] show that efficient hardware code can be generated directly from CAL models. The results show the quality of results produced by the RTL synthesis engine for a real-world application (MPEG-4 Simple Profile video decoder). The code generated from the high-level dataflow description actually outperforms the VHDL design in terms of both throughput and silicon area.

C code can also be generated from CAL models. The results presented in this paper and in [18] show significant improvements compared to CAL dataflow simulation with the Open Dataflow environment [9]. The compilation process has been successfully applied to the same MPEG-4 Simple Profile video decoder. The synthesized software (10600 lines of code) is faster than the CAL dataflow simulated (about 20 frames/s instead of 0.15 frames/s), and close to real-time for

a QCIF format (25 frames/s) on a standard PC platform. It is interesting to notice that the model is scalable: the number of macro-blocks decoded per second remains constant when dealing with larger image sizes.

CAL is particularly well suited for describing signal processing systems which are intrinsically data-driven. It is not by chance that CAL language has been chosen by the ISO/IEC standardization organization in the new MPEG standard called "Reconfigurable Video Coding (RVC)" (ISO/IEC 23001-4 and 23002-4). RVC is a framework allowing users to define a multitude of different codecs, by combining together actors (called coding tools in RVC) from the MPEG standard library that contains video technology from all past standards (i.e. MPEG-2, MPEG-4 etc, etc). The reader can refer to [19] for more information about RVC. CAL is used to provide the reference software for all coding tools of the entire library. The essential elements of the RVC framework, besides the tool library, include a Decoder Description expressed in an XML dialect which describes in the architecture of the decoder by specifying the connections between the different actors, a Bitstream Schema (BS) which describes the structure, the organization of the data in the bitstream and implicitly defines the parser needed for the specific decoder reconfiguration.

VII. CONCLUSION AND FUTURE WORK

This paper describes a framework based on CAL data flow language that includes a simulator and SW and HW code generators. CAL data flow models results particularly efficient for specifying signal processing systems in a very synthetic form compared to classical imperative languages. Moreover CAL models can be developed to describe architectural features of the desired implementation, thus enabling the designer to work for both algorithm and architecture specification at the same level of abstraction. Hardware and software code generators then provide the implementation of the actors and associated network of actors on different targets (processors or FPGA). CAL succeeds in unifying different levels of abstraction in a single layer at which specification, design space exploration and efficient implementation can be developed. CAL is very expressive and concise. It exposes clearly the intrinsic parallelism of the algorithm by means of the notion of encapsulated actor processing and explicit data dependencies in the actor network. The first promising results obtained by this framework for modelling systems and generating software/hardware implementations, and the recent adoption by ISO/IEC MPEG of CAL as a the new specification formalism for the library that covers in modular form all video algorithms from the different MPEG video coding standards, shows that CAL is an appropriate language for supporting design flows aiming at building complex heterogeneous systems from high level system specifications. Concerning future works and extension of the framework they include the evolution of the software and hardware code generators in terms of the extension of the subset of CAL language supported by the two code generators, the development of scheduling tools beyond

the SystemC scheduler for mapping on multicore platforms and the evolution of the current Open DataFlow environment with more accurate and extended profiling and debug tools.

REFERENCES

- [1] "Streamc language specification," <http://graphics.stanford.edu/streamlang/streamc-3-6-00.pdf>.
- [2] Alberto Sangiovanni-Vincentelli, "Quo Vadis, SLD? Reasoning About the Trends and Challenges of System Level Design," *Proceedings of the IEEE*, vol. 95, pp. 467–506, 2007.
- [3] Petri Kukkala, Mikko Setälä, Tero Arpinen, Erno Salminen, Marko Hannikainen, and Timo D. Hamakainen, "Implementing a wlan video terminal using uml and fully automated design flow," *EURASIP J. Embedded Syst.*, vol. 2007, pp. 20–20, 2007.
- [4] D. Thomas, "Mda: revenge of the modelers or uml utopia?," *Software, IEEE*, vol. 21, pp. 15–17, 2004.
- [5] G. De Micheli, "Hardware synthesis from c/c++ models," in *Design, Automation and Test in Europe Conference and Exhibition 1999. Proceedings*, 1999, pp. 382–383.
- [6] G. De Micheli, D. Ku, D. Ku, F. Mailhot, and T. Truong, "The olympus synthesis system," *Design & Test of Computers, IEEE*, vol. 7, pp. 37–53, 1990.
- [7] Johan Eker and Jörn Janneck, "CAL Language Report," 2003, ERL Technical Memo UCB/ERL M03/48.
- [8] J. Eker, J.W. Janneck, E.A. Lee, Jie Liu, Xiaojun Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Yuhong Xiong, "Taming heterogeneity - the ptolemy approach," *Proceedings of the IEEE*, vol. 91, pp. 127–144, 2003.
- [9] "Open dataflow sourceforge project," <http://opendf.sourceforge.net/>.
- [10] Edward A. Lee and Thomas M. Parks, "Dataflow Process Networks," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 773–801, May 1995.
- [11] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. Comput.*, vol. 36, no. 1, pp. 24–35, 1987.
- [12] Shuvra S. Bhattacharyya, Praveen K. Murthy, and Edward A. Lee, "Synthesis of embedded software from synchronous dataflow specifications," *J. VLSI Signal Processing Systems*, vol. 21, no. 2, pp. 151–166, 1999.
- [13] Luis Damas and Robin Milner, "Principal type-schemes for functional programs," in *Proceedings of POPL '82*, 1982, pp. 207–212.
- [14] G. C. Necula, S. McPeak, S. P. Rahul, and W. Weimer, "CIL: An Infrastructure for C Program Analysis and Transformation," in *Proceedings of CC 2002*, Apr. 2002, pp. 213–228.
- [15] ISO/IEC FDIS 23002-2:2007(E), "Information technology – MPEG video technologies – Part 2: Fixed-point 8x8 inverse discrete cosine transform and discrete cosine transform," 2007.
- [16] C. Loeffler, A. Ligtenberg, and G. Moschytz, "Practical Fast 1-D DCT Algorithms with 11 Multiplications," in *Proceedings of ICASSP '89*, Feb. 1989.
- [17] Jörn W. Janneck, Ian D. Miller, Dave B. Parlour, Marco Mattavelli, Christophe Lucarz, Matthieu Wipliez, Mickael Raulet, and Ghislain Roquier, "Translating dataflow programs to efficient hardware: an mpeg-4 simple profile decoder case study," in *Design, Automation and Test in Europe (DATE)*, Munich, Germany, 2008.
- [18] Matthieu Wipliez, Ghislain Roquier, Mickael Raulet, Jean-Francois Nezan, and Olivier Dforges, "Code generation for the MPEG reconfigurable video coding framework: from CAL actions to C functions," in *IEEE International Conference on Multimedia & Expo (ICME)*, Hannover, Germany, 2008.
- [19] Christophe Lucarz, Marco Mattavelli, Joseph Thomas-Kerr, and Jörn Janneck, "Reconfigurable media coding: A new specification model for multimedia coders," in *IEEE Workshop on Signal Processing Systems*, 2007, pp. 481–486.

Automatic Allocation of Redundant Operators in Arithmetic Data path Optimization

Sophie Belloeil, Roselyne Chotin-Avot, Habib Mehrez and Alix Munier-Kordon
 University Paris VI, LIP6/SOC Laboratory
 4 place Jussieu,
 75252 Paris Cedex 05, France
 Sophie.Belloeil@lip6.fr

Abstract

Redundant operators such as adders and multipliers increase performance (timing and area) of high computational digital circuits. Mixing redundant operators with classical ones is nevertheless complex for circuit designers who might not have necessarily the required arithmetic knowledge. In this context, Computer Aided Design tools considering redundant arithmetic are of interest.

In this paper, several algorithms based on graph theory are described. They replace some classical operators of a design with redundant ones to minimize the overall timing. Several real life experiments are presented.

I. Introduction

Redundant operators such as adders and multipliers have very good performances considering time and area [7], [10], [14]. Using those operators in VLSI circuit design can thus appear advantageous, enabling architecture optimizations and consequently further improvements as for circuits performances. Several hand-made implementations have been done using those architectures, leading to good results [5], [6], [9], [16]: improvement up to 35% for the frequency with an area overhead bounded by 11% for a Discrete Cosine Transform Operator for example.

Mixing classical and redundant arithmetics in an explicit way can nevertheless appear quite tedious to non initiated designers, for whom, furthermore, the rapid pace of technological evolution puts a great "time to market" pressure. Such a pressure on design cycle combined with strict performance constraints make the automation of the introduction of redundant arithmetic in circuit design more and more useful, bringing it more accessible. There has

therefore been an extensive research work on the introduction of redundant arithmetic in logical synthesis [11], [12], [15]. However, they focus mainly on using only Carry-Save adders, and choose to transform subtractions and multiplications into additions. They do not address the possibility of using redundant multipliers as well as the Borrow-Save representation.

We have already proposed an approach based on pattern matching techniques in [2]. We also have demonstrated the interest of using the Borrow-Save representation in [3]. However, the pattern matching approach is not the most appropriate one for handling circuits which have operators with multiple fanouts. Algorithms have been developed for it to be handled, but increase a lot the time to perform the optimisation.

In this paper, we present a new approach based on graph theory. The two criteria considered are again the timing and the area minimization. Two heuristics solving this optimization problem are developed. The first one modifies classical operators into redundant ones as much as possible like the pattern matching approach. The second one uses a cost function in order to choose the best allocation possible for each operator. Furthermore, these two algorithms consider Borrow-Save architectures to handle subtractions.

Several designs, such as a FIR filter, a Distance Computation Unit (DCU), a Fast Fourier Transform (FFT) butterfly and a Discrete Cosine Transform (DCT) are optimised, using our two algorithms.

The remainder of this paper is organized as follows: Section 2 contains a global description of the redundant arithmetic and the associated architectures. In Section 3, we describe our redundant optimization algorithms. Our experimental results are presented in Section 4. Section 5 is our conclusion.

II. Redundant arithmetic

A. Mixed arithmetic

Redundant arithmetic involves two number representations [1]:

- **Carry-Save** representation: a digit is defined by $cs_i = cs_i^0 + cs_i^1$ with $cs_i \in \{0, 1, 2\}$ so that a number is considered as the sum of two terms: $CS = CS^0 + CS^1$.
- **Borrow-Save** representation: a digit is defined by $bs_i = bs_i^0 - bs_i^1$ with $bs_i \in \{-1, 0, 1\}$ so that a number is considered as the subtraction of two terms: $BS = BS^0 - BS^1$.

The abbreviations CS and BS are commonly used for Carry-Save and Borrow-Save representations, as well as NR and R for respectively Non Redundant and Redundant representations.

The sole use of redundant arithmetic in data path description is not conceivable for several reasons. Firstly, we must preserve the NR representation of the inputs/outputs of the circuits. Secondly, we have to deal with non-arithmetic operators such as multiplexors, boolean operators, etc... Classical and redundant arithmetics have therefore to be compatible. A new arithmetic is presented, called **mixed arithmetic**, defined as the combination between classical and redundant representations. This involves:

- 1) having at disposal every arithmetic operator accepting both R and NR inputs/outputs: the three representations of classical, redundant and mixed adders are presented in Figure 1 for example.
- 2) being able to convert one representation to the other: for example, the conversion $CS \rightarrow NR$ is the addition between the two terms composing the CS number.

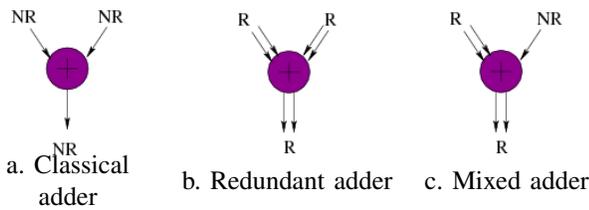


Fig. 1. Adders representation

B. Carry-Save architectures

1) *Adders*: The architecture of a **redundant adder**, adding two CS numbers, is presented in Figure 2. A similar architecture exists for a **mixed adder**, adding a CS number and a NR number, such as shown in Figure 3. Both adders

provide a CS output made of the effective sum and the carries ($Output = S + C$).

These architectures show the main benefit of redundant arithmetic: it allows to suppress the carry propagation in the computation of an addition. The time to perform an addition of two numbers is thus constant, independent of the number of digits. Addition being an essential operator, the potential benefit of using redundant/mixed adders is significant.

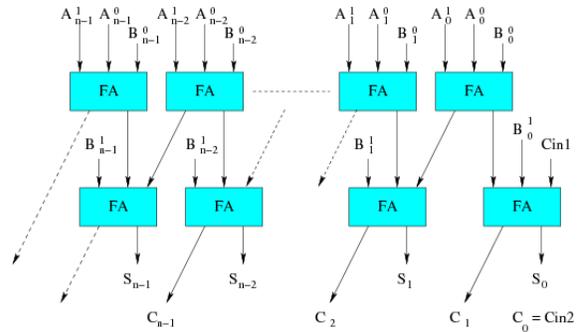


Fig. 2. Implementation of a redundant adder

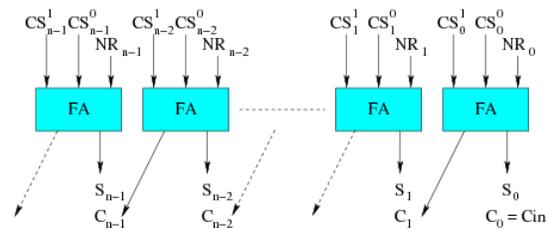


Fig. 3. Implementation of a mixed adder

2) *Multipliers*: The multiplier architecture frame (Figure 4) is divided into four parts [7]: (1) recoding (for inputs in R form), (2-3) partial products + sum, (4) conversion (to obtain the output in NR form). The second and third parts are mandatory; they represent the effective computation of the multiplication. The first and last parts are optional, depending on the representations of the inputs/output. The computations made are then $Output = (CS^0 + CS^1) * NR$ for mixed multipliers and $Output = (CS_0^0 + CS_0^1) * (CS_1^0 + CS_1^1)$ for redundant ones.

Since the output of the Wallace tree [14] (part 3) is in CS form, allowing the output of the multiplier to be in CS form generates the suppression of the final conversion. CS multipliers are therefore bound to have a better critical time than classical ones, but a bigger area because of the input recoders (especially with two R inputs).

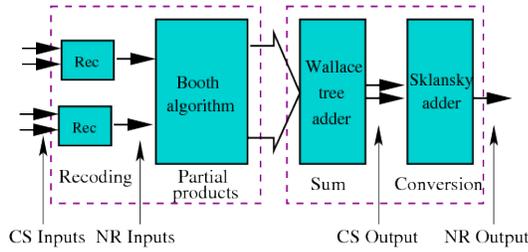


Fig. 4. Multiplier

C. Borrow-Save architectures

1) *Adders*: BS architectures allow to perform subtractions. They have the same frame as the CS ones (cf. the mixed adder architecture in Figure 5). Using the BS representation, a mixed adder (adding a BS number and a NR number) consists in computing $Output = C + S = BS^+ - BS^- + NR$. A redundant adder (adding two BS numbers) consists in $Output = C + S = BS_0^+ - BS_0^- + BS_1^+ - BS_1^-$.

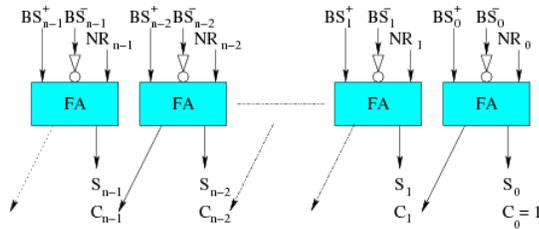


Fig. 5. Borrow Save mixed adder

The three different ways to implement the computation $(a-b) + (c-d)$ are shown in Figure 6: the NR architecture (Sklansky adders [14]: subtractions transformed into additions using the two's complement) in Figure 6.a, the CS architecture (same treatment as for subtractions) in Figure 6.b and the BS one in Figure 6.c. We have compared in [3] the use of those architectures, in terms of time and area¹. The results presented show that the BS architectures performances are in the same order of magnitude as the CS ones.

2) *Multipliers*: BS architectures have the same frame as the CS ones [7]. The difference is in the recoding part. The computations made are then $Output = S + C = (BS^+ - BS^-) * NR$ for mixed multipliers and $Output = S + C = (BS_0^+ - BS_0^-) * (BS_1^+ - BS_1^-)$ for redundant ones.

We have compared in [3] the use of the different architectures to perform a computation involving subtractions

¹Note that for all the tests presented, we used the place and route tools of the Cadence CAD System using the Alliance [8] CMOS Standard Cell Library in 0.35μm

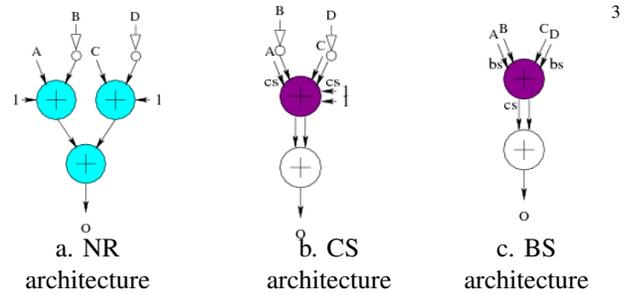


Fig. 6. Computation of $(a - b) + (c - d)$

and multiplications: $(a - b) * (c - d)$. As before, three architectures have been compared: the NR architecture (Sklansky adders and a classical multiplier), the CS one and the BS one. The results show that the BS architecture is faster and smaller than the CS one.

D. Advantage of the Borrow-Save representation

It has been shown in [3] that the interest of using the BS representation is better emphasized when subtractions are inside arithmetical chains. Let us consider the computation $(a + b) - (c + d)$. The three possible architectures to implement this computation are presented in Figure 7. We can see in Figure 7.b that, when using the CS architectures, the introduction of an inverter in the arithmetical chain prevents from using a redundant operator, leading to a non optimal design. Using the BS architecture avoids that issue, such as shown in Figure 7.c.

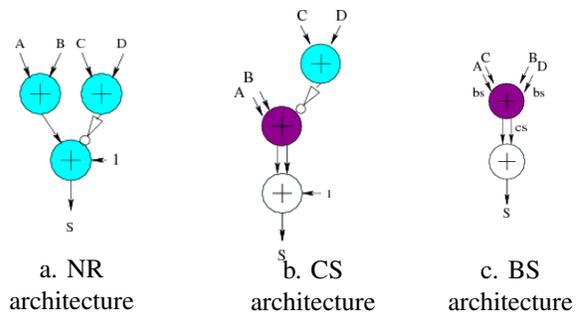


Fig. 7. Computation of $(a + b) - (c + d)$

The timing and area performances of these implementations are presented in [3]: using CS architectures results in 25.7% area improvement and 6.7% timing deterioration whereas using BS architectures result in 53.6% area improvement and 27.5% timing improvement, compared to the classical implementation (Figure 7.a). The BS implementation is therefore 94% smaller and 32% faster than the CS one.

III. Automatic optimization

Our aim is to take advantage of redundant arithmetic in order to improve circuits timing. Our CAD tools are part of a classical VLSI design flow and take place just before logical synthesis. We have developed two algorithms which, from a functional specification of a circuit described with the Stratus language [4], modify the specifications of the different operators and interconnections of the circuit, while preserving its behavior and inputs/outputs. After this process, the VLSI design flow remains unchanged.

This Section is organized as follows. First, some definitions are introduced. Secondly, our algorithms are described. Last but not least, the implementation of the final circuit is presented.

A. Definitions

Our optimization algorithms are based on graph theory. We therefore consider our circuits as graphs to use algorithmic tools to optimize them.

1) *Arithmetic computation graph*: An arithmetic datapath can be modelled using a directed acyclic graph $G = (V, \mathcal{A})$ such that:

- 1) the set of operators (arithmetic or not), inputs and outputs of the circuit represents the set of vertices V of the graph G ,
- 2) all signals of the circuit represent the set of arc $(x, y) \in \mathcal{A}$.

For any vertex $x \in V$, $\Gamma^-(x)$ denotes the set of direct predecessors of x ,

$$\Gamma^-(x) = \{y \in V, (y, x) \in \mathcal{A}\}.$$

2) *Allocation function*: An allocation function is defined in order to distinguish the signals in redundant representation from the others. The reasons a signal can not be put in redundant representation are if it is an input/output of the circuit, or if it is an input/output of a non arithmetical operator. Let $V_o \subset V$ be the set of vertices from V corresponding to binary arithmetic operators. Elements from V_o may be implemented using classical, mixed or redundant operators. Elements from $V - V_o$ correspond then to inputs, outputs or non arithmetic operators. The set of arcs that may be implemented using a redundant representation is then $\mathcal{A}_o = \{(x, y) \in \mathcal{A}, x \in V_o, y \in V_o\}$.

An *allocation* is a function $a : \mathcal{A} \rightarrow \{0, 1\}$ such that,

- 1) $\forall (x, y) \in \mathcal{A} - \mathcal{A}_o, a(x, y) = 0$;
- 2) $\forall (x, y) \in \mathcal{A}_o, a(x, y) = 1$ if (x, y) is in redundant representation, $a(x, y) = 0$ otherwise.

For any feasible allocation a , the set of arcs implemented using a redundant representation is $\mathcal{R}(a) = \{(x, y) \in \mathcal{A}_o, a(x, y) = 1\}$.

B. All in redundant algorithm

The *all in redundant* algorithm moves all arcs from \mathcal{A}_o into redundant representation, i.e. for any arc $e = (x, y) \in \mathcal{A}_o$, $a(e) = 1$ so that $\mathcal{R}(a) = \mathcal{A}_o$.

In other words, this algorithm can be stated as follows: *Given a graph G of arithmetic computations, classical arithmetic operators are transformed, when possible, into their mixed or redundant form, with the preservation of the functionality of the design.*

C. Optimal allocation algorithm

1) *Motivations*: Transforming systematically each arithmetical operator into its redundant form leads to good results but is not necessarily the optimal approach to obtain the best timing. The computation in Figure 8 emphasises this issue:

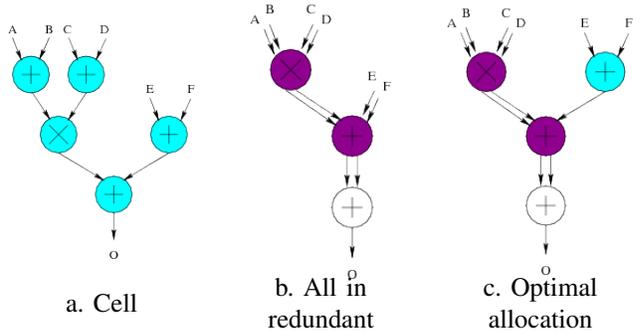


Fig. 8. Timing optimal allocation

- the critical path of the classical implementation contains two classical adders and one classical multiplier (cf. Figure 8.a),
- the *all in redundant* algorithm reduces the critical path to one redundant multiplier, one redundant adder and one classical adder (cf. Figure 8.b),
- since a classical adder is faster than a redundant multiplier, the addition between E and F can be performed in parallel with the multiplication without altering the critical path. Not transforming the adder results in changing the redundant adder in the critical path into a mixed adder. The critical path is therefore smaller (cf. Figure 8.c).

Table I presents the performances of the different architectures of Figure 8. They show an increase of the area and an improvement of the timing. The improvement of the timing being our main objective, those result confirm that, with an optimal allocation, the optimization of the timing is better than with the *all in redundant* allocation, with an area overhead.

width	Area (mm ²)			Time (ns)		
	a	b	c	a	b	c
8	0.24 ref	0.21 -9.8%	0.24 0%	60.5 ref	54.7 -9.6%	50.6 -16.4%
16	0.66 ref	0.61 -8%	0.68 +1.9%	81.64 ref	65.71 -19.5%	61.71 -24.4%

TABLE I. Computation of $(a+b)*(c+d)+(e+f)$

2) *Cost of an allocation function:* In order to find the best allocation choice for the operators, a cost function which evaluates each operator is defined, such as follows.

Let a be an allocation function such as defined in III-A.2. The cost of any arc $e = (x, y) \in \mathcal{A}$ represents the cost of its input vertex and therefore depends on $a(x, y)$ considering the 8 possibilities presented by Figure 9. This cost, noted $\mathcal{C}(a, (x, y))$, is computed as follows:

- 1) if $a(x, y) = 1$, then the arc (x, y) is implemented using a redundant representation. Its cost depends on the representation of the two inputs arcs of x following the cases (a), (b1), (b2) and (c) in Figure 9.
- 2) if $a(x, y) = 0$, then the arc (x, y) is implemented using a classical representation,
 - a) if $x \in V - V_o$, the cost of (x, y) is a constant independent from a ,
 - b) if $x \in V_o$, the cost of (x, y) depends on the representation of the two input arcs of x following the cases (d), (e1), (e2) and (f) in Figure 9.

In case of arithmetical operators (cases 1 and 2.b), the cost is based on the complexity of the corresponding architecture, for example: 1 for a mixed adder, 2 for a redundant adder, $\log_2(n)$ for a slansky adder, ...

Let $\mathcal{P}(G)$ denotes the paths of G . The cost of any path $\nu \in \mathcal{P}(G)$ is $\sum_{e \in \nu} \mathcal{C}(a, e)$. The cost $\mathcal{C}(a)$ of an allocation is the maximum cost of a path from G , so

$$\mathcal{C}(a) = \max_{\nu \in \mathcal{P}(G)} \sum_{e \in \nu} \mathcal{C}(a, e).$$

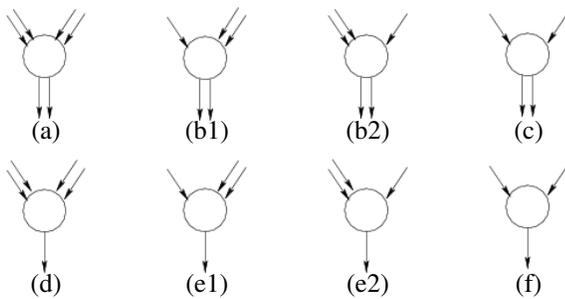


Fig. 9. All cases of connections

3) *Allocation functions for in-trees:* We suppose here⁵ that the arithmetic data-path graph is an in-tree denoted by τ . So, every vertex $x \in V$ has one successor in G .

4) *Algorithm:* The *optimal allocation* algorithm minimizes the cost function using dynamic programming. Let the arc $e = (x, y) \in \mathcal{A}$ and let $\tau(x)$ be the sub-tree of τ rooted in x . Let also $\mathcal{O}(e, 1)$ be a set of arcs from $\tau(x)$ in a redundant representation for an optimal solution for the graph $\tau(x)$ with the constraint that $a(e) = 1$. On the same way, let $\mathcal{O}(e, 0)$ be a set of arcs from $\tau(x)$ in a redundant representation for an optimal solution for the graph $\tau(x)$ with the constraint that $a(e) = 0$.

Sets $\mathcal{O}(e, 0)$ and $\mathcal{O}(e, 1)$ are built recursively as follows:

- 1) If $x \in V - V_o$, then every allocation function verifies $a(e) = 0$ and thus $\mathcal{O}(e, 1)$ is not defined. If x is an input vertex, we set $\mathcal{O}(e, 0) = \emptyset$. Otherwise, all the elements from $\Gamma^-(x)$ must be classical operators and $\mathcal{O}(e, 0) = \bigcup_{y \in \Gamma^-(x)} \mathcal{O}((y, x), 0)$.
- 2) Otherwise, $x \in V_o$ is a binary arithmetic operator and has exactly two inputs vertices denoted by y_1 and y_2 .
 - For $\mathcal{O}(e, 0)$, we fix $a(e) = 0$ and the output of x is implemented using a classical representation. Then, $\mathcal{O}(e, 0)$ is the minimum cost solution between the four following alternatives:
 - a) the arcs (y_1, x) and (y_2, x) are implemented using a redundant representation, so the first alternative is $\mathcal{O}((y_1, x), 1) \cup \mathcal{O}((y_2, x), 1) \cup \{(y_1, x), (y_2, x)\}$ (case (d) of Figure 9);
 - b) the arc (y_1, x) is implemented using a classical representation, and the arc (y_2, x) is implemented using a redundant representation, so the second alternative is $\mathcal{O}((y_1, x), 0) \cup \mathcal{O}((y_2, x), 1) \cup \{(y_2, x)\}$ (case (e1) of Figure 9);
 - c) the arc (y_1, x) is implemented using a redundant representation, and the arc (y_2, x) is implemented using a classical representation, so the third alternative is $\mathcal{O}((y_1, x), 1) \cup \mathcal{O}((y_2, x), 0) \cup \{(y_1, x)\}$ (case (e2) of Figure 9);
 - d) lastly, arcs (y_1, x) and (y_2, x) are implemented using a classical representation and the fourth alternative is $\mathcal{O}((y_1, x), 0) \cup \mathcal{O}((y_2, x), 0)$ (case (f) of Figure 9).
 - $\mathcal{O}(e, 1)$ is evaluated on the same way by considering that the output of x is implemented using a redundant representation. Four alternatives are investigated following cases (a), (b1), (b2) and (c) of Figure 9.

The optimum solution is $\mathcal{O}((x, y), 0)$ with y , the root of τ (since $y \in V - V_o$).

D. Implementation

Once the used algorithm has determined the best representation of each arc, the corresponding optimized circuit has to be created. It can be feasible only if, for each operation (addition, subtraction, multiplication), an architecture exists for every possible case of connection.

Table II presents all possible cases and the corresponding architectures. Three architectures exist for each operation (the classical one, the mixed one and the redundant one), which is sufficient to handle all the cases (a conversion $CS \rightarrow NR$ is done if the output of a R operator has to be NR). Comments can nevertheless be done upon the two cases marqued with a ”*”. In these cases, a null value has to be added in order to use redundant operators. The operation wanted is indeed $NR - (CS^0 + CS^1)$ which is equal to $NR - CS^0 - CS^1$. This operation can be implemented only by transforming it into $(NR - CS^0) + (0' - CS^1)$.

IV. Experimental results

The tests performed are meant to show the usefulness of the redundant arithmetic. We have therefore made, for each benchmark, an implementation using classical arithmetic only, and one or several implementations using our algorithms. We present the performances of the circuits, in terms of timing and area, with and without optimizations.

The first two benchmarks presented (FIR and DCU) have been optimized with the two different optimization algorithms. They can be modelled using trees and are therefore supported by the *optimal allocation* algorithm. The other two benchmarks (FTT butterfly and DCT) can not be modelled using trees, the optimizations presented result then from the *all in redundant* algorithm.

1) *FIR Filter operator*: The filter architecture is shown in Figure 10. Three implementations are done: the classical one, the one resulting from the *all in redudant* algorithm, and the one resulting from the *optimal allocation* algorithm. Since this design contains no subtraction, both algorithms use the CS representation.

The results (from a filter with 8 registers and 16 bits datas) are summarized in Table III. Thoses results show that both algorithms optimize timing and area. They also show that the *optimal allocation* algorithm optimizes better the timing, but less the area.

Architectures obtained with the different algorithms are shown in Figure 11 (exemple with 4 registers):

- Figure 11.a presents the result of the *all in redundant* algorithm, in which all arithmetical operators are transformed into their redundant form.

TABLE II. Architectures

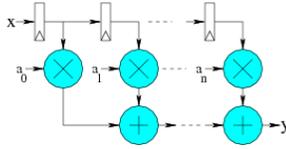


Fig. 10. FIR filter operator

	Classical	All redundant	Optimal
Area (mm^2)	3.97 ref	2.94 -25.92%	3.34 -15.83%
Time (ns)	134.39 ref	109.58 -18.47%	87.95 -34.56%

TABLE III. Results of the FIR filter

- Figure 11.b presents the result of the *optimal allocation* algorithm, in which the instantiation of several classical multipliers leads to mixed adders in the critical path instead of redundant adders. This choice can be made because redundant representations are not necessary in this case to improve the timing: the classical multipliers remain faster than the computation made on the other input of the adders in the critical path.

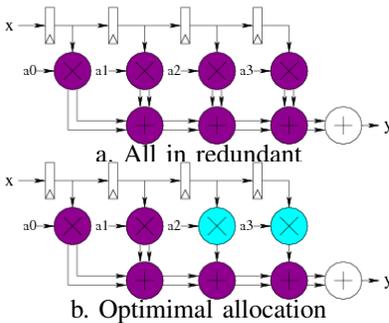


Fig. 11. FIR filter optimizations

2) *DCU operator*: The DCU architecture [5] is composed of two parts, such as shown in Figure 12: the first one computes the distances $(A_i - B_i)^2$, the second one performs the sum of these distances.

The results, summarized in Table IV, present the optimizations of both algorithms. This operator containing subtractions, each algorithm is used twice, once using the CS architectures, once using the BS ones. Those results show that the *optimal allocation* algorithm results in the same architecture as the *all in redundant* one: in this case the *optimal allocation* is indeed to transform all the arcs into redundant representation. They also show that the BS architectures produce a better timing and a better area than the CS ones.

3) *FFT butterfly*: The *butterfly* is the elementary cell composing the Fast Fourier Transform [16]. Its architecture

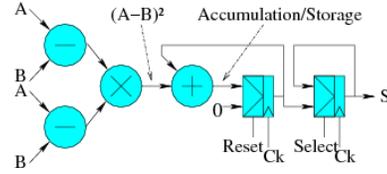


Fig. 12. DCU operator

	Classical	All Redundant		Optimal	
		CS	BS	CS	BS
Area (mm^2)	0.24 ref	0.29 +24.27%	0.24 0%	0.29 +24.27%	0.24 0%
Time (ns)	65.87 ref	56.72 -13.89%	54.44 -17.36%	56.72 -13.89%	54.44 -17.36%

TABLE IV. Results of the DCU operator

is shown in Figure 13. In this Figure, complex numbers are used, and we have: $X = A + w.B$ $Y = A - w.B$ where $w = \text{Cos} + i.\text{Sin}$.

The results, from the *all in redundant* algorithm with CS architectures and BS ones, are summarized in Table V. Once again, the BS architectures result in better performances than the CS ones.

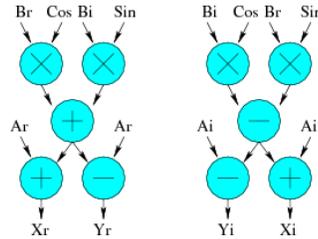


Fig. 13. FFT butterfly operator

	Classical	All Redundant	
		CS	BS
Area (mm^2)	0.77 ref	0.68 -10.91%	0.62 -19.54%
Time (ns)	63.68 ref	66.24 +4.02%	56.42 -11.39%

TABLE V. Results of the Butterfly operator

4) *1-D DCT operator*: The architecture of the DCT is shown in Figure 14. It represents the the Loeffler Signal Flow Graph [13] which computes the 1-D DCT of 8 pixels in only one cycle.

The results are summarized in Table VI, resulting again in better performances for the BS architectures. Let us compare those implementations with the ones resulting from our previous approach in [2] and [3]: the optimal algorithm produces better area (up to -4.6%) and timing (up to -1.7%). In addition, the optimal algorithm is performed up to 84% faster than the pattern matching technique.

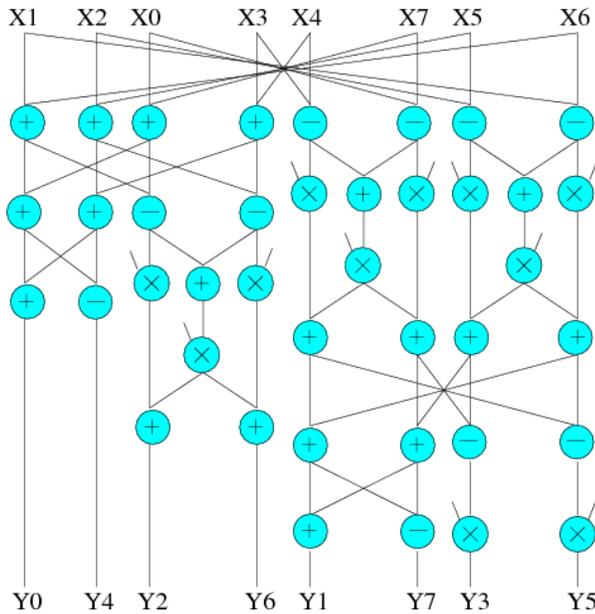


Fig. 14. 1-D DCT operator

	Classical	All redundant		Pattern Matching	
		CS	BS	CS in [2]	BS in [3]
Area (mm ²)	3.96 ref	3.91 -1.23%	3.73 -5.82%	4.1 +3.5%	3.85 -2.8%
Time (ns)	115.7 ref	100.25 -13.35%	95.99 -17.04%	100.53 -13.1%	97.65 -15.6%

TABLE VI. Results of the DCT operator

V. Conclusion

In this paper, two algorithms have been presented, which use automatically redundant arithmetic in order to optimize high computational digital circuits. The first algorithm does a systematic transformation of arithmetical operators into their redundant form. The second one does an optimal allocation for the timing. In addition, those algorithms use the Carry-Save architectures only or the Borrow-Save architectures also in order to optimize subtractions. We aimed at introducing the different solutions and outlining their pros and cons.

Our experimental results can be summarized as follows. First of all, a systematic transformation of arithmetical operators into their redundant form is not the optimal approach to obtain the best timing. An algorithm finding the best allocation for each operator (classical, mixed or redundant) seems like the best alternative. Second of all, this new approach based on graph theory seems better adapted to DAGs than the previous one.

We aim at testing our algorithms on other benchmarks in order to strengthen our conclusion that the *optimal allocation* algorithm leads to better results. Since this

algorithm is currently limited to trees, we also aim at extending its use to DAGs in order to be able to use it on more benchmarks.

References

- [1] A. Avizienis. Signed-digit number representation for fast parallel arithmetic. *IRE Trans. Electronic Computers*, 10:389–400, 1962.
- [2] S. Belloeil, R. Chotin-Avot, and H. Mehrez. Data path optimization using redundant arithmetic and pattern matching. In *Workshop on Design and Architectures*, 2007.
- [3] S. Belloeil, R. Chotin-Avot, and H. Mehrez. Arithmetic data path optimization using borrow-save representation. In *IEEE annual symposium on VLSI*, 2008.
- [4] S. Belloeil, D. Dupuis, C. Masson, J.-P. Chaput, and H. Mehrez. Stratus: A procedural circuit description language based upon python. In *International Conference on Microelectronics*, pages 275–278, 2007.
- [5] Y. Dumonteix, Y. Bajot, and H. Mehrez. A fast and low-power distance computation unit dedicated to neural networks, based on redundant arithmetic. In *International Symposium on Circuits and Systems*, volume 4, pages 878–881, 2001.
- [6] Y. Dumonteix, R. Chotin, and H. Mehrez. Use of redundant arithmetic on architecture and design of a high performance DCT macro-bloc generator. In *Conference on Design of Circuits and Integrated Systems*, pages 428–433, 2000.
- [7] Y. Dumonteix and H. Mehrez. A family of redundant multipliers dedicated to fast computation for signal processing. In *International Symposium on Circuits and Systems*, volume 5, pages 325–328, 2000.
- [8] A. Greiner and F. Pecheux. Alliance: A complete set of cad tools for teaching vlsi design. In *EuroChip Workshop*, 1992.
- [9] A. Guyot. Ocapfi: architecture of a vlsi coprocessor for the gcd and the extended gcd of large numbers. *Symposium on Computer Arithmetic*, pages 226–231, 1991.
- [10] A. Guyot, Y. Herrerós, and J.-M. Muller. Janus, an on-line multiplier/divider for manipulating large numbers. *Symposium on Computer Arithmetic*, pages 106–111, 1989.
- [11] T. Kim, W. Jao, and S. Tjiang. Arithmetic optimization using carry-save-adders. In *Design Automation Conference*, pages 433–438, 1998.
- [12] Y. Kim and T. Kim. Accurate exploration of timing and area trade-offs in arithmetic optimization using carry-save-adders. In *Conference on Asia South Pacific design automation*, pages 622–628, 2001.
- [13] C. Loeffler, A. Lightenberg, and G. Moschytz. Practical fast 1d-dct algorithms with 11 multiplications. In *Intl. Conf. On Acoustics, Speech and Signal Processing*, pages 988–991, 1989.
- [14] J. M. Muller. *Elementary Functions, Algorithms and Implementation*. Birkhauser Boston, 1997.
- [15] J. Um, T. Kim, and C. Liu. Optimal allocation of carry-save-adders in arithmetic optimization. In *IEEE International Conference on Computer-Aided Design*, pages 410–413, 1999.
- [16] A. Vacher, M. Benkhebbab, A. Guyot, T. Rousseau, and A. Skaf. A vlsi implementation of parallel fast fourier transform. *European Design and Test Conference*, pages 250–255, 1994.

High-Precision VLSI Architecture of Lifting-Based Forward and Inverse Wavelet Transforms

Andre Guntoro and Manfred Glesner

Department of Electrical Engineering and Information Technology

Institute of Microelectronic Systems

Technische Universität Darmstadt

Email: {guntoro,glesner}@mes.tu-darmstadt.de

Abstract—The richness of wavelet transformation is known in many fields. There exist different classes of wavelet filters that can be used depending on the application. In this paper, we propose a high-precision lifting-based wavelet processor that can perform various forward and inverse DWTs. Our architecture is based on $N \times M$ PEs that can perform either prediction or update on a continuous data stream in every two clock cycles. Contrary to other lifting-based processors, which focus on JPEG2000, our design is based on the fact that the wavelet transformations are not used only in the area of image processing and the wavelet filters cannot be represented as integer numbers. Therefore their computations cannot be satisfied by using integer arithmetics. For this purpose, IEEE 754-compliant floating-point arithmetics are used to compute the transformation. We also consider the normalization step that takes place at the end of the forward DWT or at the beginning of the inverse DWT. To cope with different wavelet filters, we feature a multi-context configuration to select among various DWTs. For the 32-bit implementation, the estimated area of the proposed wavelet processor with 2×8 PEs configuration in a $0.18\text{-}\mu\text{m}$ technology is 3.7 mm square and the estimated operating frequency is 361 MHz.

I. INTRODUCTION

The breakthrough in wavelet theory, which has been studied in the last two decades by many researchers [1], [2], [3], has delivered a solid background and has drawn much attention due to its attractive properties, especially regarding its time-frequency localization feature [4], [5]. Wavelets have been known in many fields such as mathematics, physics, and electrical engineering. Due to numerous interchanging fields, wavelets have been used in many applications such as image compression, feature detection, seismic geology, human vision, etc. Because different applications require different treatments, researchers have tried to cope with their own issues and implemented only a subset of wavelets suitable for their own needs such as one that can be found in image compression [6], [7], [8] and speech processing [9], [10], [11]. The power of wavelet tools is then limited due to these approaches.

In this paper, we propose a novel architecture to compute forward and inverse transforms of numerous DWTs (Discrete Wavelet Transforms) based on their lifting scheme representations. Most lifting-based wavelet processors are dedicated to compute biorthogonal (5,3) and (9,7) wavelet filters used in JPEG2000, where the coefficients can be represented as integers. Andra in [12] described a VLSI architecture to compute

(5,3) and (9,7) wavelet filters which required two adders, one multiplier, and one shifter in each row and column processor. The condition was as such that the prediction or the update constants of the actual and the delayed samples are equal (i.e. $c(1+z^{-1})$), which is one of the properties of this class of filter. Dillen in [13] described the combined architecture of the same filters for FPGAs. Seo detailed the arithmetic rescheduling of these biorthogonal wavelet filters with the aim to minimize the number of registers in [14]. Our new proposed architecture takes into account that the wavelet coefficients of an arbitrary wavelet filter and the corresponding wavelet transforms cannot be satisfied by using integer computation. Additionally, the lifting step representation of an arbitrary wavelet filter may have two different predict/update constants as the result of the polyphase decomposition. Their corresponding Laurent polynomials may have higher-order factors (i.e. $c_1 z^{-p} + c_2 z^{-q}$), which are common in various classes of wavelet filters such as Daubechies, Symlet, and Coiflet wavelet filters. The proposed architecture also considers the normalization step which takes place at the end of the forward DWT or at the beginning of the inverse DWT. In order to be flexible, the proposed architecture provides a multi-context configuration to choose between various forward and inverse DWTs.

The rest of the paper is organized as follows. Section II describes the second generation of wavelets. The proposed architecture, including the processing element, the floating-point multiplier and adder, the normalization, the configuration, and the context switch, are explained in Section III. Section IV discusses the performance of the proposed architecture and Section V summarizes our conclusions.

II. LIFTING SCHEME

Contrary to the filter approach, which separates the signal into low and high frequency parts and performs the decimation on both signals afterwards, the second generation of wavelets reduces the computation by performing the decimation in advance. The second generation of wavelets, more popular under the name lifting scheme, was introduced by Sweldens [15]. The basic principle of the lifting scheme is to factorize the wavelet filter into alternating upper and lower triangular 2×2 matrix. Let $H(z)$ and $G(z)$ be a pair of low-pass and high-pass wavelet filters:

$$H(z) = \sum_{n=k_l}^{k_h} h_n z^{-n} \quad G(z) = \sum_{n=k_l}^{k_h} g_n z^{-n}$$

where h_n and g_n are the corresponding filter coefficients. $N = |k_h - k_l| + 1$ is the filter length and the corresponding Laurent polynomial degree is given by $h = N - 1$. By splitting the filter coefficients into even and odd parts, the filters can be rewritten as:

$$\begin{aligned} H(z) &= H_e(z^2) + z^{-1}H_o(z^2) \\ G(z) &= G_e(z^2) + z^{-1}G_o(z^2) \end{aligned}$$

and the corresponding polyphase representation is:

$$P(z) = \begin{bmatrix} H_e(z) & G_e(z) \\ H_o(z) & G_o(z) \end{bmatrix}$$

Daubechies and Sweldens in [15], [16] have shown that the polyphase representation can always be factored into lifting steps by using the Euclidean algorithm to find the greatest common divisors. Thus the polyphase representation becomes:

$$P(z) = \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \prod_{i=n}^1 \begin{bmatrix} 1 & a_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ b_i(z) & 1 \end{bmatrix}$$

where K is the normalization factor and $a_i(z)$ and $b_i(z)$ are the Laurent polynomials which correspond to the updaters and the predictors of the lifting steps. Fig. 1 shows the arrangement of the lifting scheme representation. The Laurent polynomials $b_i(z)$ and $a_i(z)$ are expressed as predictor $P_i(z)$ and updater $U_i(z)$. The signal S_j is split into even and odd parts. Prediction and update steps occur alternately. The predictor $P_i(z)$ predicts the odd part from the even part. The difference between the odd part and the predicted part is computed and used by the updater $U_i(z)$ to update the even part. At the end, the low-pass and the high-pass signals are normalized with a factor of K and $1/K$ respectively.

By factoring the wavelet filters into lifting steps, it is expected that the computation performed on each stage (either a prediction or an update) will be much less complex. As an example, the famous Daub-4 wavelet filter with the low-pass filter response:

$$H(z) = \frac{1 + \sqrt{3}}{4\sqrt{2}} + \frac{3 + \sqrt{3}}{4\sqrt{2}}z^{-1} + \frac{3 - \sqrt{3}}{4\sqrt{2}}z^{-2} + \frac{1 - \sqrt{3}}{4\sqrt{2}}z^{-3}$$

can be factored into lifting steps:

$$P(z) = \begin{bmatrix} \frac{\sqrt{3}-1}{\sqrt{2}} & 0 \\ 0 & \frac{\sqrt{3}+1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 & -z \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & \sqrt{3} \\ -\frac{\sqrt{3}}{4} + \frac{2-\sqrt{3}}{4}z^{-1} & 1 \end{bmatrix} \begin{bmatrix} 1 & \sqrt{3} \\ 0 & 1 \end{bmatrix}$$

Since the finding of the greatest common divisors is not necessarily unique, the result of the Laurent polynomials may also differ. The Daub-4 and the biorthogonal (5,3) and (9,7) wavelet filters can be factored into lifting steps with a maximum degree of ± 1 [16] whereas Symlet-6 and Coiflet-2 (the lifting computations are not detailed here due to a page limitation) may have a $z^{\pm 5}$ factor on its Laurent polynomials.

III. PROPOSED ARCHITECTURE

The lifting-based forward DWT splits the signal into even and odd parts at the first stage. The split signals are processed by an alternating series of predictors and updaters (on some wavelet filters an updater may come before a predictor). On the final stage, the multiplication with the normalization factor takes place. The inverse DWT performs exactly everything backwards. It starts with the multiplication with normalization factor, continues with a series of updaters and predictors, and finishes with the merging of the outputs.

As the lifting scheme breaks a wavelet filter into smaller predictions and updates, the resulting predictor and updater can be limited to have a maximum Laurent polynomial degree of one. Nevertheless, the predictor or the updater of higher-order wavelet filters may have higher factors as well. Without loss of generality, we can formulate the predictor or the updater polynomial as:

$$l(z) = c_1 z^{-p} + c_2 z^{-q}$$

with polynomial constants c_0 and c_1 , and $|p - q| = 1$. This implies that on each stage (either as a predictor or an updater), two multiplications and two additions are performed. As an example, the first predict and update steps of Daub-4 can be written as:

$$\begin{aligned} \begin{bmatrix} s' \\ d \end{bmatrix} &= \begin{bmatrix} 1 & \sqrt{3} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s \\ d \end{bmatrix} = \begin{bmatrix} s + d \cdot \sqrt{3} \\ d \end{bmatrix} \\ \begin{bmatrix} s' \\ d' \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ -\frac{\sqrt{3}}{4} + \frac{2-\sqrt{3}}{4}z^{-1} & 1 \end{bmatrix} \begin{bmatrix} s' \\ d \end{bmatrix} \\ &= \begin{bmatrix} s' \\ d + s' \cdot \frac{-\sqrt{3}}{4} + s' \cdot \frac{2-\sqrt{3}}{4}z^{-1} \end{bmatrix} \end{aligned}$$

which perform one multiplication and one addition in order to solve s' and two multiplications and two additions to solve d' .

Taking into account that a predictor and an updater perform a similar computation, we propose a new wavelet processor which is based on NxM processing elements to cope with N-dimensional wavelet computations and M lifting steps. The N dimensions of our proposed wavelet processor can be interpreted in various ways. For example, the proposed wavelet processor with 2xM processing elements can perform two concurrent DWTs on a 2D image to compute the transformation on both row and column data at the same time. In case of a 1D signal, it can be used to double the performance by splitting the signal in half. Additionally, in case of wavelet packets, the proposed NxM architecture can perform N different transformations in a simultaneous manner.

A. Architecture of the Processing Element

The core behind our proposed architecture is the processing element (PE), which performs the prediction or the update in every two clock cycles. Taking into account that floating-point multipliers are expensive, in term of logic counts, and the PE receives two samples (s and d) at once, we have decided to lower the input rate by half. From the performance point of view, the processing rate of the PE will be equal to the processor speed and no longer twice as fast. This also implies

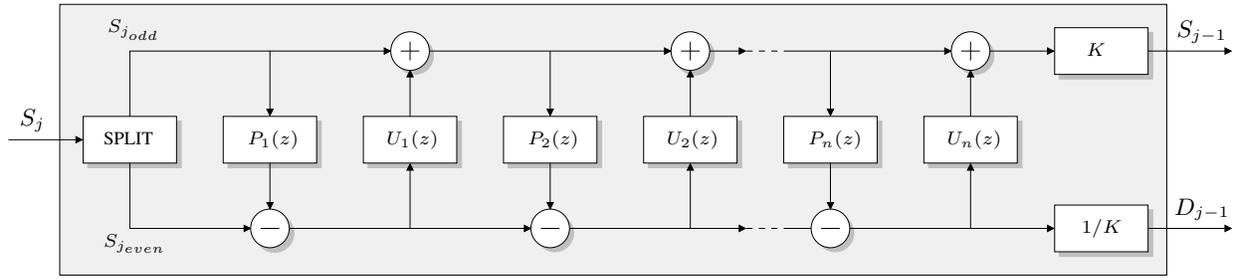


Fig. 1. Forward lifting steps.

that the bottleneck issues on the input and output ports with the memory will not occur. From the hardware implementation point of view, the PE requires only one multiplier. This optimization, as detailed later, is accomplished by multiplexing the operands of the multiplier inputs (the *multiplier* and the *multiplicand*).

The PE is a pipeline-based architecture in order to maximize the performance. Fig. 2 depicts the proposed PE. The PE has two selectors S1 and S2 to choose the prediction or the update samples that correspond to the factors p and q from the Laurent polynomial. Two constants C1 and C2 that represent the filter coefficients are defined and configured by the controller. By delaying the actual samples, selector S3 controls the prediction or the update that requires future samples. Selector S4 is a bypass selector. Because lifting steps of higher-order wavelet filters may require distance prediction or update samples, the maximum depth of the unit delay z^{-m} , which determines the maximum delay level, can be freely chosen during the design. Two unit delays are implemented on both input ports A and B. In order to reduce the number of registers needed for the unit delay, the unit delay on port A has two input selectors (i.e. S3 and the multiplexer output) and two outputs.

The PE is divided into 3 blocks. The first block organizes the input samples from both channels. The second block performs the floating-point multiplications on the samples that are selected by S1 and S2 with the constants C1 and C2. Two 2-level FIFOs on both input samples are implemented to compensate the multiplier delay. The last block performs the addition of three floating-point values. One 4-level FIFO is implemented to compensate the delay introduced by the adder.

B. Floating-Point Multiplier

As mentioned earlier, the PE utilizes only one floating-point multiplier which is time-shared in order to perform two multiplications. The first clock cycle performs the first multiplication (i.e. $C1 \times M1$) and the second cycle performs the second multiplication (i.e. $C2 \times M2$). M1 and M2 are the time-multiplexed output samples of the unit delay determined by the output of the multiplexer.

The floating-point multiplication follows the IEEE 754 standard [17]. Given are two floating-point numbers A and B:

$$A = (-1)^{S_a} \times M_a \times 2^{E_a} \quad B = (-1)^{S_b} \times M_b \times 2^{E_b}$$

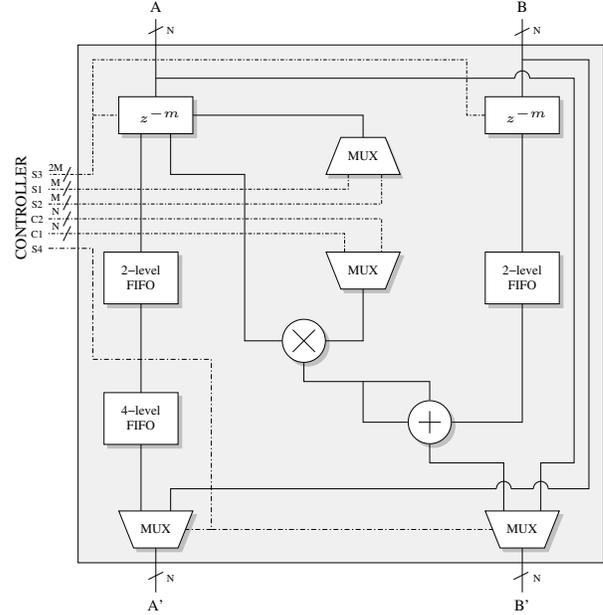


Fig. 2. Block diagram of the processing element.

To multiply two floating-point numbers, we need to multiply the mantissas M_a and M_b and add the exponents E_a and E_b . Thus, we have:

$$O = (-1)^{S_a \oplus S_b} \times (M_a \cdot M_b) \times 2^{E_a + E_b}$$

Several floating-point multiplier architectures are detailed in [18], [19]. Both architectures support only 32-bit single precision and 64-bit double precision formats. Our floating-point multiplier is a 2-level pipeline architecture and can be customized for other floating-point formats beside the standard single and double precision formats. Fig. 3 depicts the block diagram of the floating-point multiplier.

1) *Stage 1*: At the first stage, the resulting sign is resolved by the *sign block* (i.e. $S_a \oplus S_b$). The *multiplier block* inserts the hidden bit to both mantissas M_a and M_b and performs the unsigned multiplication. By taking into account the amount of the guard bits, the result of the multiplication will be truncated, resulting the fraction F . The amount of the guard bits that will be used by the *rounding block* is configurable. The *zero logic block* detects if one/both of the operands is/are zero. The *exponent add block* adds both exponents E_a and E_b .

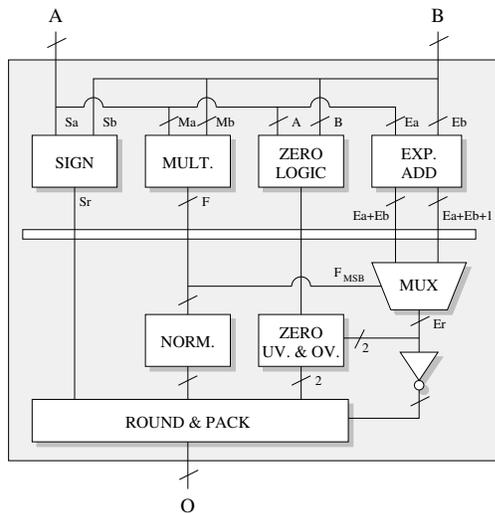


Fig. 3. Block diagram of the floating-point multiplier.

This block outputs two values to minimize the effort at stage 2. Because the exponents are in the biased-format, we can directly treat the operation as the normal unsigned addition. One guard bit is appended to the most significant bit of both exponents before the addition.

2) Stage 2: By examining the most significant bit (MSB) of the resulting fraction F , the *normalization block* normalizes the resulting fraction. The normalization is basically a shift-left by one operation. The same technique is applied to select the correct exponent E_r .

- $F_{MSB} = 0 \Rightarrow E_r = E_a + E_b$
- $F_{MSB} = 1 \Rightarrow E_r = E_a + E_b + 1$

To convert the resulting exponent E_r back to the biased-format, we only need to invert the exponent (i.e. $\overline{E_r}_{[MSB-1:0]}$). The *zero, underflow, and overflow block* checks if the result lays on the valid floating-point range.

- $Z = 1 \vee E_r_{[MSB:MSB-1]} = 00 \Rightarrow ZeroFlag = 1$
- $E_r_{[MSB:MSB-1]} = 11 \Rightarrow OverflowFlag = 1$

The *round and pack block* rounds the normalized fraction. Two rounding mechanisms are available: *rounding to zero* and *rounding to nearest*. The resulting sign, the rounded fraction, and the exponent are packed together. If the zero flag is set, the output will be cleared, and if the overflow flag is set, the output will be saturated.

C. 3-Input Floating-Point Adder

Contrary to the floating-point multiplier, the floating point adder requires more steps due to the algorithm complexity and the data dependency. As depicted in Fig. 4, to perform addition between two floating-point numbers, the following steps are performed:

- 1) Calculate the exponent difference.
- 2) Align the mantissa by shifting the mantissa with the lower exponent to the right.
- 3) Add/subtract both mantissas depending on the sign bits.

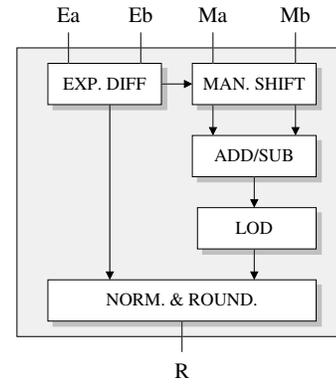


Fig. 4. Two-Input Floating-Point Adder with Leading-One Detector (LOD).

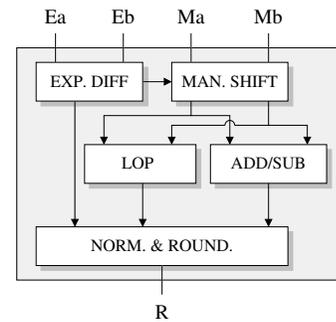


Fig. 5. Two-Input Floating-Point Adder with Leading-One Predictor (LOP).

- 4) Perform the Leading-One Detection (LOD¹) to determine the location of the first logic one.
- 5) Normalize and round the result.

In order to decrease critical paths, Leading-One Prediction (LOP²) was proposed in [20], [21], [22], [23] as a replacement of LOD, predicting the first occurrence of the logic one directly from the operands. Fig. 5 depicts the addition algorithm with LOP. The LOP works in parallel with the adder and it is based on the encoding tree which examines both inputs from left to right. The LOP ignores the possible carry or borrow that might occur on the addition/subtraction result. Thus, it leads to one-bit inaccuracy, which will be corrected during the normalization step. This is why it is more popular with the name *inexact LOP*. With additional complexity, papers in [24], [25] detailed the design of the *exact LOP* that predicts correctly the position of the leading-one. Therefore, it minimizes the effort on the normalization stage.

References [26], [27], [28] discussed the architectures of a floating-point adder. Kowaleski [26] described a 4-level floating-point adder that can be operated at 433 MHz in a 0.35- μm technology and Beaumont-Smith [28] detailed how to reduce the number of pipeline stages into three. In order to add three floating-point numbers, two inputs will be added first and the temporary result will then be added to the third input. This introduces a longer pipeline structure with the

¹LOD is also known as LZD (Leading Zero Detector).

²LOP is also known as LZA (Leading Zero Anticipator).

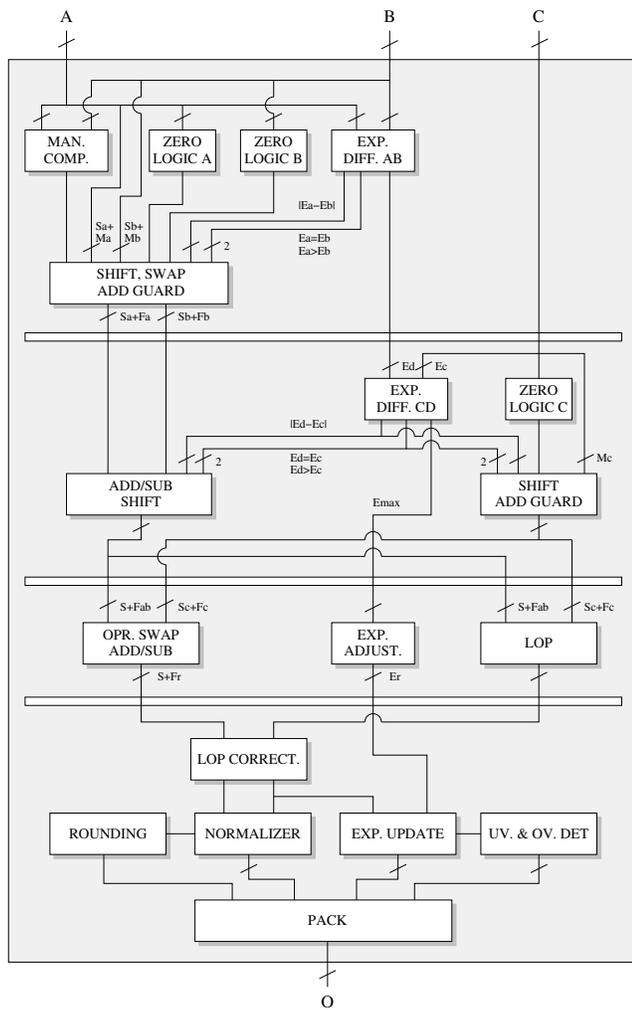


Fig. 6. 3-Input Floating-Point Adder.

normal approach (i.e. chaining two floating-point adders) and consumes more area.

To minimize the number of pipeline stages, we have developed a dedicated 4-stage 3-input floating-point adder. Fig. 6 depicts the block diagram of the adder.

1) *Stage 1:* At the first stage, the two inputs A and B are unpacked. The *mantissa comparator block* compares both mantissas M_a and M_b and outputs one decision bit (i.e. $M_a \geq M_b$) which will be used by the *shift, swap, and add guard block*. The *zero logic block* detects if the corresponding input is zero. The *exponent difference AB block* compares both exponents E_a and E_b . This block outputs four different values. The first three values (the shift count $|E_a - E_b|$ and the comparator results $E_a = E_b$ and $E_a > E_b$) will be used by the *shift, swap, and add guard block*. The fourth value (the temporary dominant exponent $E_d = \max(E_a, E_b)$) will be used by the *exponent difference CD block*.

Since the addition or subtraction is performed in a normal binary representation (i.e. unsigned number), it is necessary to sort/swap the operands (i.e. $M_a \leftrightarrow M_b$). We choose not to convert the numbers into signed, because LOP requires both

inputs to be unsigned. Additionally, we do not need to convert the final result back to unsigned during the normalization step, which would require one extra stage before the normalization. As the name implies, the *shift, swap, and add guard block* aligns the mantissa M_a and M_b to have the same exponent degree by shifting the mantissa with the smaller exponent to the right. The hidden bit and the guard bits are appended to the most significant bit and the least significant bit of both mantissas respectively. The number of bits used as guard bits can be freely chosen. Based on the exponent difference, three different cases are examined here:

- $E_a = E_b$: Depending on the output of the *mantissa comparator block*, both mantissa M_a and M_b will be swapped directly (i.e. $M_a \leftrightarrow M_b$) without performing any shifting.
- $E_a > E_b$: The mantissa M_b will be shifted to the right with the amount determined by the *exponent difference AB block* (i.e. $|E_a - E_b|$).
- $E_a < E_b$: The mantissa M_a will be shifted to the right with the amount determined by the *exponent difference AB block* (i.e. $|E_a - E_b|$). Both mantissas will be swapped afterwards.

In all of the cases, if a zero number is detected, the corresponding mantissa(s) will be set to zero. The outputs of the *shift, swap, and add guard block* are the sorted and extracted fractions F_a and F_b with their corresponding signs.

At this stage, the input C is not processed, thus two values that correspond to the prediction/update sample and the first multiplication result can be added first. The second multiplication result that comes on the next clock cycle will be unpacked at the second stage.

2) *Stage 2:* At this stage, the fractions F_a and F_b are added/subtracted depending on the sign difference (i.e. $S_a \oplus S_b$), resulting in the fraction F_{ab} . If the exponent E_c is greater than E_d , the result will be shifted to the right. These steps are performed by the *add/sub and shift block* with the shift parameter determined by the *exponent difference CD block*. Three different cases as at the first stage are also examined here.

The *shift and add guard block* prepares the mantissa M_c . If the exponent E_c is less than E_d , M_c will be shifted instead. The hidden bit and the guard bits are appended to M_c , resulting in fraction F_c . Finally if the *zero logic block* detects a zero number, F_c will be set to zero.

3) *Stage 3:* At stage 3, the *operand swap and add/sub block* swaps the operands F_{ab} and F_c if necessary (note that both operands have the same exponent). Afterwards, it performs the addition or subtraction. To minimize the logic counts and the logic levels on this stage, we have utilized the inexact LOP. The *LOP block* works parallel with the *operand swap and add/sub block* to predict the first occurrence of the logic one directly from the operands. Taking into account that one-bit inaccuracy might occur on the prediction, the *LOP block* prepares two values at the output to minimize the critical paths on the normalization stage.

Because three addition/subtraction arithmetic operations are involved, the final result might have an increase of exponent by two. The *exponent adjustment block* prepares the dominant exponent by simply adding two to the largest exponent (i.e. $E_r = \max(E_a, E_b, E_c) + 2$).

4) *Stage 4*: Because the *LOP block* may deliver an error within one-bit degree, the error has to be corrected. The error can easily be detected by looking at the LOP-index bit of the resulting fraction F_r . This step is performed by the *LOP correction block*. Additionally, this block also performs the pre-normalization by shifting the resulting fraction F_r to the left with the shift amount determined by the inexact LOP value. We have examined that correcting the LOP result in real time (by adding with one) will increase the critical path on the stage 4. This is why the *LOP block* on the stage 3 outputs two values. Therefore, we only need to choose the right value at the end.

Should the inexact LOP predicts the leading-one position falsely, the *normalizer block* corrects the pre-normalized fraction by shifting it to the left. Here we only need to perform one bit shifting. The rounding logic implements two rounding mechanisms: *rounding to zero* and *rounding to nearest*. Based on the corrected LOP value, the *exponent update block* updates the resulting exponent. The *underflow and overflow detector block* checks if the resulting exponent lays on the valid floating-point range. Finally, the sign, the normalized fraction, and the corrected exponent are packed together by the *pack block*.

D. Normalization

As normalization can take place at the end of the transformation in case of forward DWT or at the beginning of the transformation in case of inverse DWT, two special processing elements to handle this function are required. Neglecting the prediction/update process (addition between the actual sample and the predictor/updater values), normalization can be performed on the PE. We extend the functionality of the PEs that are located at the top and at the bottom of the proposed wavelet processor. Three additional multiplexers are needed to add the normalization factor unit into the PE. Fig. 7 shows the PE used at the top and at the bottom of the proposed architecture. By enabling S5 and setting S1 and S3 to zero, two inputs of the multiplexer before the multiplier correspond to the actual samples s and d (with the normalization factors $K = C1$ and $1/K = C2$). The first multiplication product passes through the multiplexer and the 1-level FIFO resulting in $s' = Ks$ (left side). The second multiplication product passes through the multiplexer resulting $d' = d/K$ (right side). The 4-level FIFO is split into 3-level and 1-level FIFOs, with the latter used to make the both outputs synchronized.

E. Controller

To cope with various lifting-based forward and inverse DWTs, we have separated the configuration-dependent parameters from the PE. Figs. 2 and 7 show how the inputs of the selectors and the multiplier constants are separately drawn on

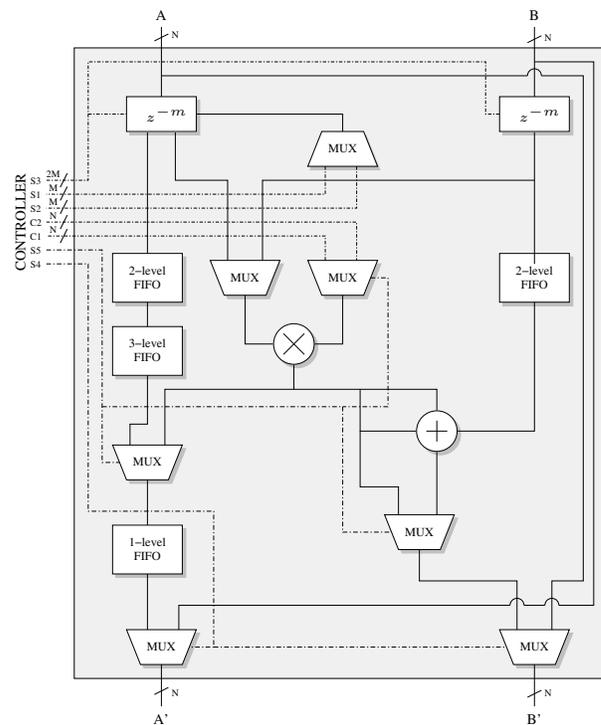


Fig. 7. Block diagram of the processing element which is located at the top and at the bottom.

the left side of the figures. To support different classes of wavelet filters, which require different types of configurations, we have implemented a multi-context configuration on each PE as depicted in Fig. 8. Each PE is assigned a row index as a unique ID for the configuration. Multiplier constants use the signal data paths to save the wiring cost whereas the selector configuration requires an additional controller path. The context switch is implemented as a memory module, where the address is controlled by the context selector, and the write enable signal is controlled by the output comparator. The active configuration can easily be selected by using this context-based controller to cope with various wavelet filters.

Benefits of a multi-context configuration are: (1) the proposed wavelet processor can be configured to perform the corresponding inverse DWTs in a very simple manner, (2) wavelet transformations that use longer wavelet filters can be computed by splitting the lifting steps, and (3) the issues regarding the boundary condition can be relaxed by utilizing special wavelet filters on the signal boundaries, which require less or no delayed/future samples (e.g. Haar wavelet), instead of exploiting the periodicity or mirroring of the signal.

IV. RESULTS AND PERFORMANCES

The proposed wavelet processor is based on modular and parametric approaches and is written in VHDL. Wavelet processors with 2x8 PEs to process two concurrent forward/inverse DWTs and eight lifting steps (including normalization), 8-level unit delays to support higher-order wavelet filters, and 16 available contexts to configure the transforma-

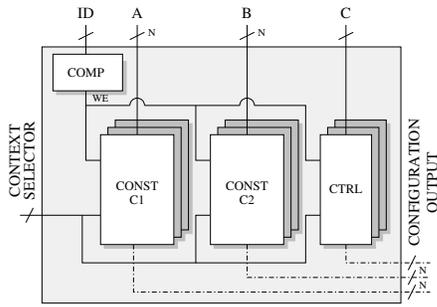


Fig. 8. Controller for the PE.

TABLE I
EST. AREA AND FREQUENCY OF A WAVELET PROCESSOR WITH 2X8 PEs.

Data Width	Est. Area (in mm ²)	Est. Freq. (in MHz)
16-bit (10p)	1.723	471.70
20-bit (14p)	2.195	425.53
24-bit (18p)	2.704	392.16
28-bit (21p)	3.206	371.75
32-bit (23p)	3.667	361.01

tions, are implemented and synthesized. Rounding to nearest with three guard bits is used on all floating-point arithmetics. Note that wavelet filters that require longer lifting steps can be split into several large steps with a maximum number of eight. The design is synthesized using 0.18- μ m technology. The estimated area and frequency of various data width implementations are reported in Table I. The value inside the bracket indicates the number of bits used for the precision. For the 32-bit configuration, the proposed wavelet processor consumes 3.7 mm² chip area and has a maximum operating speed of 361 MHz. As a comparison, Andra [12] with 16-bit integer architecture can only compute (5,3) and (9,7) filters and requires 2.8 mm² area with 200 MHz operating frequency. Table II summarizes the comparison. The other architectures, except ours, are integer-based with 16-bit data width.

To measure the level of correctness and to show the flexibility of our design, we perform three different DWTs on some predefined signals. Four different input signals ranged [-1;+1] with 1024 samples are used as references. These signals are forward and inverse transformed with Daub-6, Symlet-6, and Coiflet-2 wavelet filters, which do not have integer coefficients. The random signal has a uniform distribution. The lifting step coefficient of these wavelet filters are summarized in Table III. The SNR is computed using:

$$SNR_{(dB)} = 20 \times \log_{10} \left(\frac{\sum |signal|}{\sum |signal - result|} \right)$$

where *signal* corresponds the input vector and *result* corresponds the output of the forward and inverse transform. The SNR values of the different data width implementations are reported in Table IV and the corresponding maximum errors are reported in Table V. Depending on the data widths, SNR values vary between 46 dB and 144 dB, which are sufficient for most applications.

TABLE II
COMPARISON WITH OTHER LIFTING-BASED ARCHITECTURES.

Architecture	Operating Speed	Area	Filter
Andra [12]	200 MHz (0.18- μ m)	2.8 mm ²	(5,3) & (9,7)
Dillen [13]	110 MHz (FPGA)	-	(5,3) & (9,7)
Seo [14]	150 MHz (0.35- μ m)	-	(5,3) & (9,7)
Ours	361 MHz (0.18- μ m)	3.7 mm ²	Arbitrary

TABLE IV
SNR VALUES OF DIFFERENT DATA WIDTH IMPLEMENTATIONS (IN DB).

Source	Daub-6				
	16-bit	20-bit	24-bit	28-bit	32-bit
Sinusoid	56.38	78.27	100.68	118.41	134.81
Sawtooth	56.35	78.61	104.09	119.41	133.74
Step	72.01	86.77	110.81	119.41	128.29
Random	55.34	79.51	103.80	119.61	133.77
Source	Symlet-6				
	16-bit	20-bit	24-bit	28-bit	32-bit
Sinusoid	56.24	80.67	102.19	121.00	135.41
Sawtooth	55.46	81.76	103.02	120.26	135.61
Step	65.61	76.40	113.82	119.20	144.25
Random	45.95	70.10	93.47	112.06	123.43
Source	Coiflet-2				
	16-bit	20-bit	24-bit	28-bit	32-bit
Sinusoid	47.83	72.20	96.45	113.60	124.66
Sawtooth	47.53	71.84	94.83	112.96	125.31
Step	47.67	82.05	93.50	116.12	128.15
Random	47.63	71.49	94.93	113.47	124.44

The total latency on each PE is 7 clock cycles. One clock cycle is used by the input registers, 2+1 by the multipliers (two multiplications take place), and 3+0 by the adders (one cycle is *stolen* from the multiplier). Additional sample latency (two clock cycles per future sample) will add-up to the total latency on the PEs, which require this feature. The PE that is configured as a normalizer has a latency of 4 clock cycles only.

For the NxM wavelet processor, the total time needed to compute an L-stage forward/inverse DWT is:

$$T = T_d L + \frac{2S}{N} (1 - 0.5^L)$$

where *S* is the signal length and $T_d = M \times T_{PE}$ is the circuit delay with T_{PE} as the PE latency delay.

V. CONCLUSIONS

We have proposed a novel architecture to compute floating-point arithmetic-based forward/inverse DWTs. The proposed wavelet processor is based on NxM PEs and can accept continuous data streams. It can also be configured easily to support higher-order lifting polynomials as a result of the factorization of higher-order wavelet filters. To cope with different wavelet filters, the proposed architecture includes a multi-context configuration so that users can easily switch between transformations (including their inverses). Additionally, the proposed architecture takes into account the normalization step that occurs at the end of the forward DWT or at the beginning of the inverse DWT. The proposed wavelet processor is capable to receive continuous data streams and compute

TABLE III
LIFTING COEFFICIENTS OF DAUB-6, SYMLET-6, AND COIFLET-2 WAVELET FILTERS.

Type	Daub-6		Symlet-6		Coiflet-2	
Updater	2.425 z^0		-0.227 z^0		-2.530 z^0	
Predictor	0.079 z^{-1}	-0.352 z^0	-1.267 z^{-1}	0.216 z^0	-0.240 z^{-1}	0.342 z^0
Updater	-2.895 z^1	0.561 z^2	0.505 z^1	-4.255 z^2	3.163 z^1	15.268 z^2
Predictor	-0.020 z^{-2}		0.045 z^{-3}	0.233 z^{-2}	0.006 z^{-3}	-0.065 z^{-2}
Updater			-18.389 z^3	6.624 z^4	-63.951 z^3	13.591 z^4
Predictor			0.144 z^{-5}	-0.057 z^{-4}	0.001 z^{-5}	0.002 z^{-4}
Updater			-5.512 z^5		-3.793 z^5	
Normalizer	0.432	2.315	-0.599	-1.671	0.108	9.288

TABLE V
MAXIMUM ERROR OF DIFFERENT DATA WIDTH IMPLEMENTATIONS.

Source	Daub-6				
	16-bit ($\times 10^{-2}$)	20-bit ($\times 10^{-3}$)	24-bit ($\times 10^{-5}$)	28-bit ($\times 10^{-6}$)	32-bit ($\times 10^{-6}$)
Sinusoid	0.5371	0.3357	1.7166	4.5299	0.6556
Sawtooth	0.7324	0.5798	2.0981	3.5763	0.8941
Step	0.1953	0.0916	0.7629	1.9074	0.4768
Random	0.4883	0.3052	1.9073	3.3379	0.7152
Symlet-6					
Sinusoid	0.4883	0.2441	2.2888	2.3842	0.3576
Sawtooth	0.4883	0.4578	4.9591	2.8610	1.3113
Step	0.3906	0.2747	1.3351	4.0531	0.2980
Random	2.5513	1.4343	8.2016	9.0003	2.7269
Coiflet-2					
Sinusoid	1.1231	0.8240	4.0054	5.9605	2.5630
Sawtooth	1.8066	0.9460	7.4387	7.3910	2.2053
Step	0.9766	0.3662	6.6757	3.5085	1.1921
Random	1.8555	1.1597	7.5340	9.5367	2.2650

the transformation in every two clock cycles. Using 0.18- μm technology, the estimated area of the proposed wavelet processor with 32-bit configuration is 3.7 mm^2 and the estimated operating speed is 361 MHz.

REFERENCES

[1] I. Daubechies, "The wavelet transform, time-frequency localization and signal analysis," *IEEE Trans. on Information Theory*, vol. 36, pp. 961–1005, 1990.

[2] S. Mallat, Ed., *A Wavelet Tour of Signal Processing*. Academic Press, Incorporated, 1998.

[3] Y. Meyer, *Wavelets and Operators*. Press Syndicate of the University of Cambridge, 1992.

[4] A. Bultheel, "Wavelets with applications in signal and image processing," 2003.

[5] Y. T. Chan, *Wavelet Basics*. Kluwer Academic Publishers, 1994.

[6] C. Christopoulos, A. Skodras, and T. Ebrahimi, "The JPEG2000 still image coding system: an overview," *IEEE Trans. Consumer Electron.*, vol. 46, no. 4, pp. 1103–1127, 2000.

[7] S. Grgic, K. Kers, and M. Grgic, "Image compression using wavelets," in *Proc. of the IEEE Intl. Symposium on Industrial Electronics, ISIE '99*, K. Kers, Ed., vol. 1, 1999.

[8] R. Calderbank, I. Daubechies, W. Sweldens, and B.-L. Yeo, "Lossless image compression using integer to integer wavelet transforms," *Proc. of the Intl. Conference on Image Processing*, vol. 1, pp. 596–599, 1997.

[9] J. Agbinya, "Discrete wavelet transform techniques in speech processing," in *Proc. of the IEEE TENCON. Digital Signal Processing Applications, TENCON '96*, vol. 2, 1996, pp. 514–519.

[10] B. Carnero and A. Drygajlo, "Perceptual speech coding and enhancement using frame-synchronized fast wavelet packet transform algorithms," *IEEE Trans. Signal Processing*, vol. 47, pp. 1622–1634, 1999.

[11] Y. Kaisheng and C. Zhigang, "A wavelet filter optimization algorithm for speech recognition," in *Intl. Conference on Communication Technology Proc., ICCT '98*, vol. 2, 22–24 Oct. 1998, p. 5.

[12] K. Andra, C. Chakrabarti, and T. Acharya, "A VLSI architecture for lifting-based forward and inverse wavelet transform," *IEEE Trans. Signal Processing*, vol. 50, no. 4, pp. 966–977, 2002.

[13] G. Dillen, B. Georis, J. Legat, and O. Cantineau, "Combined line-based architecture for the 5-3 and 9-7 wavelet transform of JPEG2000," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 9, pp. 944–950, Sept. 2003.

[14] Y.-H. Seo and D.-W. Kim, "A New VLSI Architecture of Lifting-Based DWT," *Lecture Notes in Computer Science*, vol. 3985/2006, pp. 146–151, 2006.

[15] W. Sweldens, "The Lifting Scheme: A New Philosophy in Biorthogonal Wavelet Constructions," *Wavelet Applications in Signal and Image Processing*, vol. 3, pp. 68–79, 1995.

[16] I. Daubechies and W. Sweldens, "Factoring Wavelet Transforms into Lifting Steps," *J. Fourier Anal. Appl.*, vol. 4, no. 3, pp. 245–267, 1998.

[17] IEEE Standard Board and ANSI, "IEEE standard for binary floating-point arithmetic," 1985, IEEE Std 754-1985.

[18] G. Even, S. Mueller, and P.-M. Seidel, "A dual mode IEEE multiplier," in *Proc. of the Second Annual IEEE Intl. Conference on Innovative Systems in Silicon*, 1997.

[19] W. Krandick and J. Johnson, "Efficient multiprecision floating point multiplication with optimal directional rounding," in *Proc. of the Symposium on Computer Arithmetic*, 29 June–2 July 1993, pp. 228–233.

[20] R. Kershaw, L. Bays, R. Freyman, J. Klinikowski, C. Miller, K. Mondal, H. Moscovitz, W. Stocker, L. Tran, W. Hays, J. Boddie, E. Fields, C. Garen, and J. Tow, "A programmable digital signal processor with 32b floating point arithmetic," in *Proc. of the IEEE Intl. Solid-State Circuits Conference. Digest of Technical Papers*, 1985.

[21] E. Hokenek and R. K. Montoye, "Leading-zero anticipator (LZA) in the IBM RISC System/6000 floating-point execution unit," *IBM Journal of Research and Development*, vol. 34, pp. 71–77, 1990.

[22] J. Bruguera and T. Lang, "Leading-one prediction scheme for latency improvement in single datapath floating-point adders," in *Proc. of the Intl. Conference on Computer Design: VLSI in Computers and Processors*, 5–7 Oct. 1998.

[23] H. Suzuki, H. Morinaka, H. Makino, Y. Nakase, K. Mashiko, and T. Sumi, "Leading-zero anticipatory logic for high-speed floating point addition," *IEEE J. Solid-State Circuits*, vol. 31, no. 8, pp. 1157–1164, Aug. 1996.

[24] J. Bruguera and T. Lang, "Leading-one prediction with concurrent position correction," *IEEE Transactions on Computers*, vol. 48, pp. 1083–1097, 1999.

[25] G. Gervig and M. Kroener, "Floating-Point Unit in Standard Cell Design with 116 Bit Wide Dataflow," in *Proc. of the 14th IEEE Symposium on Computer Arithmetic*, 1999.

[26] J. Kowaleski, G. Wolrich, T. Fischer, R. Dupcak, P. Kroesen, T. Pham, and A. Olesin, "A dual execution pipelined floating-point CMOS processor," in *Proc. of the IEEE Intl. Solid-State Circuits Conference Digest of Technical Papers. 43rd ISSCC*, 8–10 Feb. 1996, pp. 358–359,473.

[27] P.-M. Seidel and G. Even, "Delay-optimized implementation of IEEE floating-point addition," *IEEE Transactions on Computers*, vol. 53, pp. 97–113, 2004.

[28] A. Beaumont-Smith, N. Burgess, S. Lefrere, and C. Lim, "Reduced latency IEEE floating-point standard adder architectures," in *Proc. of the 14th IEEE Symposium on Computer Arithmetic*, 14–16 April 1999.

An Image Compression System for Earth Observation Satellites

Tanya Vladimirova

Surrey Space Centre
Dept. of Electronic Engineering
University of Surrey
Guildford, UK
t.vladimirova@surrey.ac.uk

Guoxia Yu

Surrey Space Centre
Dept. of Electronic Engineering
University of Surrey
Guildford, UK
g.yu@surrey.ac.uk

Abstract—On-board image data compression is an important feature of satellite remote sensing payloads. This paper proposes a new image compression system for Earth observation satellites. A lossless image compression scheme is developed, which is based on the CCSDS lossless data compression recommendation for space applications. The scheme features novel techniques such as a multidimensional prediction methods enabled by a new scanning scheme and an embedded radiometric calibration technique, which result in highly efficient compression of panchromatic and multispectral satellite images. A configurable model for hardware implementation of the proposed algorithm in the form of a soft IP core is developed, which gives users high degree of flexibility. Three typical configurations are implemented and evaluated in terms of implementation resources. The IP core is tested on a FPGA prototyping board featuring low-power consumption. The compression core is integrated into a reconfigurable System-on-Chip platform for payload data processing and control, enabling real-time image compression on board satellites.

I. INTRODUCTION

Earth observation (EO) satellites require transmission to ground of an extensive amount of imaging data. The data transmission capability of the onboard equipment these days is reaching several times that of the downlink circuit capacity provided by present satellites. Data compression is believed to be a solution to the “Bandwidth Versus Data Volume” dilemma of modern spacecraft. Therefore compression is becoming a very important feature in payload image processing units of EO satellites [1].

There are several types of redundancy in an image, such as spatial redundancy, statistical redundancy, and human vision redundancy. Removing these types of redundancy is how the process of compression is achieved. Spatial decorrelation methods, like prediction or transformation, are usually employed to remove the spatial redundancy. Spectral redundancy, which is specific for multispectral satellite images, can be removed via inter-band spectral correlation. Prediction based [2] and Discrete Cosine Transform (DCT) based [3] compression techniques have dominated the field of on-board image compression. However, in recent years, the usage of Discrete Wavelet Transform (DWT) based techniques [4] is increasing

for their outstanding compression performance at a low bit-rate. Prediction based techniques are still very popular, for they provide the most effective way to achieve lossless data compression, which is a necessary requirement for most remote sensing missions.

The Consultative Committee for Space Data Systems (CCSDS) has recommended the CCSDS - Lossless Data Compression (CCSDS-LDC) algorithm [5] for use on board spacecraft. CCSDS-LDC is a low complexity algorithm which features low memory and power usage. Its error-resilience functionality is important for the hostile space environment. To stop error propagation, a sample is periodically kept uncompressed as a reference. So data before and after a reference sample are compressed independently. Therefore error is constrained in a small region, called Independent Compression Region (ICR). Being a 2-D type of data, an image can be compressed further, by using a 2-D prediction scheme, instead of the default 1-D scheme. However, compression of a current pixel will depend on neighbour pixels of previous lines, which is contrary to the featured coding independency, assuming one line of data is fairly larger than one ICR, which is true for most Earth observation remote sensing tasks.

Recent fast advances in Field Programmable Gate Arrays (FPGA), such as high clock frequencies and parallel processing capabilities, have made them a preferred platform for digital signal processing (DSP). FPGAs have been widely used in space missions, ranging from control and data processing tasks in satellites to Mars rovers [6]. Reconfigurable hardware like FPGAs crosses the boundary between software and hardware applying hardware description languages (HDL) such as VHDL or Verilog to program and redefine the hardware architecture. A wide variety of soft Intellectual Property (IP) cores are distributed in synthesizable HDL format. However, the alteration of these HDL codes to suit different application scenarios, is extremely difficult even to experienced hardware engineers. Nowadays, high level languages (HLL) like C or Matlab are used to capture the data processing model. A class of EDA tools is emerging, which can be employed to convert automatically from HLL to Register-Transfer Level (RTL) HDL, or straightaway to FPGA configuration bit stream. This

additional procedure is developed to speed up the design cycle, and to let designers concentrate on the algorithmic optimization and architectural exploration.

This paper proposes a real-time on-board compression system for satellite remote sensing imagery targeted at FPGA implementation. The paper is structured as follows. Section II introduces a new on-board image compression system. Section III discusses a novel lossless image compression algorithmic scheme based on CCSDS-LDC. Section IV presents performance evaluation results. Section V and VI detail the design of a lossless image compression hardware accelerator and its incorporation as a peripheral IP core in a system-on-a-chip (SoC).

II. ON-BOARD IMAGE COMPRESSION SYSTEM OVERVIEW

Future satellite missions will be capable of carrying out intelligent on-board image processing such as image classification and change detection, which are important for remote sensing applications. Image classification is used primarily for cloud detection. Clouds are a common problem in Earth observation using optical imagery, being effectively unwanted "application noise". The performance of compression can be improved as a result of cloud editing. Change detection analysis can help to improve the transmission bandwidth by sending to ground only the part of the image, which contains the identified changes. The incorporation of a fine-grained tiling scheme in the compression process, has proved to increase the resilience of image compression algorithms to single-bit errors [7].

Automatic band-to-band registration is the most critical pre-processing requirement for imaging tasks performed on board satellites, such as data fusion, inter-band coding, change detection, spectral signature based classification, etc.. The performance of these tasks depends on the robustness and accuracy of band registration. The a priori information of band misalignment is not accurate enough, due to spacecraft shaking or attitude changes.

In current on-board compression systems, bands of multispectral images are processed independently, which is called intra-band coding. In this case, the intrinsic spectral redundancy of multispectral images, which can be removed via spectral decorrelation, still exists in the compressed file. Although spectral decorrelation has been actively investigated in the literature, it has not been applied to on-board applications yet. Two methods are used to achieve spectral decorrelation - Inter-band Prediction or (Kahunen-Loeve Transform) KLT [8]. KLT, which is considered the optimum method to spectrally decorrelate MS data, is a topic of active research at the moment; however KLT cannot so effectively decorrelate images with a number of bands lower than four. The KLT transform will have a role to play on board future disaster monitoring missions, as they are expected to have an increased number of MS bands.

Fig.1 shows the functional block diagram of the proposed system for compression of panchromatic and multispectral

images. The image data provided by the cameras are processed serially, tile-by-tile. First the data are subjected to pre-processing routines, which improve the compression or enable the systems to make intelligent decisions about the compression process. The envisaged pre-processing tasks are radiometric calibration, registration, change detection and image classification. Radiometric calibration processing is applied to the raw data in order to take into account the sensor radiance quality. As the parameters of the radiometric calibration are changing with time, this functional block should be made reconfigurable.

The image compression block consists of a spectral decorrelation unit and a 2-D image compression unit, which provides both lossless and lossy compression. An encryption block is included too in case the compressed image data need to be transferred in encrypted format. The processed data could be stored in the on-board mass memory or sent to the downlink module, from where they are downloaded to the ground station. All of the processing units can be bypassed, so raw images could be transferred to ground, which is required in some cases. It is envisioned that a radiation-tolerant high density SRAM-based FPGA will be used as the central processing component. This is due to the advantages of SRAM-based FPGAs such as high processing performance and reconfigurability.

In the rest of the paper we discuss the design of a lossless compression intellectual property core for the implementation of the imaging system in Fig.1 as an FPGA-based system-on-a-chip.

III. LOSSLESS COMPRESSION ALGORITHM

This section presents a new lossless compression scheme for satellite images based on CCSDS-LDC. First the proposed scanning scheme and multidimensional prediction methods are introduced and then the CCSDS-LDC mapper and encoder are described. The scanning method achieves coherence between the regional independence coding and multidimensional predictions. An embedded Brightness Difference Compensation (BDC) technique increases the efficiency of the proposed system further for push-broom type of sensors.

A. Vertical Scanning Technique

Normally image data are read in a raster scan (RS) order, in which the first pixel of each line is taken as the reference sample. Hence ICR is just one horizontal line of data. The scan method, named Peanno-Hilbert (PH) scan, is believed to be the optimal scanning technique reducing 2-D spatial correlation to 1-D correlation [9].

To enable a 2-D prediction without affecting the ICR coding independency, a new vertical scan (VS) is proposed, which has a "V" shape. This scan goes down vertically, and turns from the start again after N pixels. N is 16, as it is the number of samples in the smallest compression unit. Therefore a 2-D prediction can be carried out using previous vertical line(s), while residing within one ICR [12].

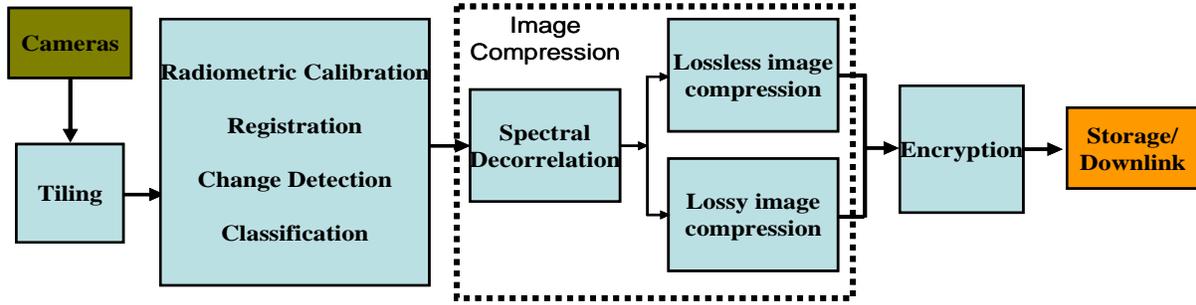


Fig. 1 Block diagram of the proposed on-board image compression system

B. Multidimensional Prediction Methods

The 2-D Gradient-Adjusted Prediction (GAP) technique uses the context gradient information to predict the intensity of the current pixel [10]. The GAP pixel neighbourhood pattern is rotated by 90 degrees in the vertical mode so that the value of the current pixel is predicted by using two pixels above and five pixels in two previous vertical lines of the image. Sometimes the spatial correlation is much more significant than the spectral one, while in other cases the spectral correlation will dominate, which can be established by comparison of the spectral and spatial gradients. To take into account both of these correlations, a 3-D extension to GAP is proposed.

Linear CCD image sensors used in push-broom imaging payloads, have different offset and shift registers, and hence difference in brightness for the even and odd column pixels. Thus the lesser correlation between odd and even column pixels will suppress the compression performance. In [7,11] a method called Brightness Difference Compensation (BDC) is reported, which is able to bring 5.5% further data reduction on JPEG-LS. BDC is applied to images on a tile-by-tile basis with a tile size of 512 by 512 pixels. The proposed VS scan needs to buffer only 16 lines of image data, while BDC needs 512 lines. Here by adapting BDC to the proposed scanning scheme, we apply an embedded BDC, which means that it is inserted into the GAP technique. So in order to get a better prediction, the WN , W , and WS pixel values are compensated through the embedded BDC method. To predict a pixel P_{ij} in horizontal line i and vertical line j using neighbour pixels in vertical lines $j-1$ and $j-2$ the embedded BDC is applied to all the pixels $P_{i,j-1}$ in vertical line $j-1$ using the equation below:

$$P_{i,j-1} = P_{i,j-1} + (\text{mean}(P_{i,j-4}) + \text{mean}(P_{i,j-2})) / 2 - \text{mean}(P_{i,j-3}) \quad (1)$$

where $P_{i,j-1}$, $P_{i,j-2}$, $P_{i,j-3}$ and $P_{i,j-4}$ are pixels in the previous four vertical lines and “mean” is the mean of all the 16 pixels in the line, as defined in the proposed scanning scheme.

C. CCSDS Lossless Compression Standard

In May 1997, the consultative committee for space data systems (CCSDS) published a recommendation for a lossless data compression standard, which is an extended Rice algorithm

with added two low-entropy coding options [5]. The architecture of the CCSDS-LDC is shown in Fig.5. The pre-processor block consists of the proposed scanning and multidimensional prediction techniques described in sections III.A and III.B above and the CCSDS-LDC mapper, which is introduced below.

The purpose of the CCSDS-LDC mapper is to map the prediction residual to a non-negative integer, in an effective way based on the shape of its Probability Distribution Function (pdf). Assuming the current pixel value is x_i , and the predicted value is \hat{x}_i , then the difference between them is the prediction error Δ_i . After the prediction stage a mapper takes the residuals and maps them into non-negative integers, through the following equation:

$$\delta_i = \begin{cases} 2\Delta_i & 0 \leq \Delta_i \leq \theta \\ 2|\Delta_i| - 1 & -\theta \leq \Delta_i \leq 0 \\ \theta + \Delta_i & \text{otherwise} \end{cases} \quad (2)$$

where $\theta = \min(\hat{x}_i - x_{\min}, x_{\max} - \hat{x}_i)$; x_{\min} is the minimum possible value and x_{\max} is the maximum possible value.

The extended Rice encoder converts the mapped prediction residual into an encoded bit sequence y . The entropy coder is a collection of variable-length codes operating in parallel. The coding option achieving the highest compression is selected, and the option ID bit pattern is enclosed and forwarded [12].

IV. COMPRESSION PERFORMANCE EVALUATION

For the purpose of evaluating the performance of the proposed CCSDS-LDC lossless compression scheme different versions of the algorithm are modelled in Matlab. The CCSDS-LDC based algorithms are compared with the state-of-the-art lossless compression algorithm, JPEG-LS [13]. The size of ICR is set to 128 x 16 pixels, which applies to JPEG-LS as well. Compression results in terms of compression ratio (CR) are derived using natural and satellite test images.

Table 1 shows performance results on four popular natural images for JPEG-LS and different combinations of CCSDS-LDC with the three scanning methods RS, PH, VS and the 2-D GAP technique. The results show that the differ-

ent scanning schemes have similar performance, but an extra 2-D GAP processing step brings significantly better performance, which exceeds that of JPEG-LS.

TABLE 1
COMPRESSION RATIO BASED ON STANDARD TEST IMAGES

Methods Images	JPEG-LS	CCSDS-LS			
		RS	PH	VS	VS+GAP
Goldhill	1.57	1.53	1.52	1.5	1.64
Lena	1.74	1.6	1.67	1.71	1.77
Mandrill	1.17	1.26	1.24	1.21	1.3
Peppers	1.62	1.56	1.6	1.59	1.67
<i>AVE</i>	1.53	1.49	1.51	1.5	1.6

Five panchromatic images, captured from the Surrey Satellite Technology Ltd. (SSTL) Beijing-1 small satellite, are selected as test images. The images, which represent different geographical areas with varied terrain, have 4 m ground sample distance (GSD) and a size of 6144 by 6144 pixels. The Beijing-1 panchromatic imager is of a push-broom type, so we could compare the performance of the proposed embedded BDC with that of its complex rival BDC. JPEG-LS with and without BDC and different combinations of CCSDS-LDC with BDC, EmbeddedBDC, RS, VS, and GAP are evaluated. Table 2 shows that the performance results of CCSDS with VS, BDC and GAP in column 6 are comparable to the results of JPEG-LS with BDC in column 2, which are much better than JPEG-LS (column 1). By comparing the results in columns 6 and 7 it can be seen that the embedded BDC technique only slightly underperforms BDC, however it reduces the buffer memory size 32 times. Table 2 also shows that the results of the proposed solution (column 7) are much better than the results achieved by CCSDS-LS with RS (column 3), nearly doubling the compression ratio only with an extra memory buffer of 16 lines of image data and a combination of a simple scanning scheme and 2-D GAP prediction. It can be concluded from Table 2 that the proposed approach is the most efficient

scheme for lossless data compression with constrained error propagation functionality.

For evaluation of the compression performance of multispectral images, a MS image from the NASA Landsat7 satellite is used, which features varied terrain as shown in Fig.2. Six out of eight available spectral bands are used (B1, B2, B3, B4, B5, B7), as they are sharing the same GSD of 30 m. Usually the individual band images are not aligned accurately and therefore inter-band image co-registration [14] is required before the compression could be performed. Before the compression of the current band, the previous band is already compressed and taken as the reference band, while the first band is compressed using the intra-band mode. Then according to the derived displacement between these two band images, the reference one is translated and re-sampled with a bilinear model.



Fig.2. The multispectral test image (Copyright NASA)

Compression results based on the multispectral test images in terms of CR are shown in Table 3, where different combinations of methods are evaluated. The results in columns 1, 2 and 4 are derived band by band, without any inter-band coding technique. They are included here for comparison purposes. The results in columns 3 and 5 are based on the difference of the current band with the reference one, which is referred to as "BandDiff" [15]. It can be seen from Table 3 that the results in the last column, derived with the proposed 3D-GAP technique, give the best performance out of all.

TABLE 2
COMPRESSION RATIO BASED ON PANCHROMATIC TEST IMAGES

Methods Images	JPEG-LS		CCSDS-LDC				
	none	BDC	RS	RS+BDC	VS		
					BDC	BDC+GAP	EeddedBDC+ GAP
	1	2	3	4	5	6	7
D001	3.48	3.64	1.93	3.2	3.13	3.68	3.63
D002	4.49	5.32	1.73	4.69	4.87	5.07	4.94
D003	2.84	2.94	1.87	2.6	2.52	2.98	2.96
D004	3.47	3.68	1.88	3.2	3.29	3.63	3.61
D005	2.71	2.78	1.83	2.39	2.29	2.73	2.74
<i>AVE</i>	3.4	3.67	1.85	3.21	3.22	3.62	3.58

TABLE 3
COMPRESSION RATIO BASED ON MULTISPECTRAL IMAGES

Methods Images	CCSDC-LDC					
	JPEG-LS	RS	RS+BandDiff	VS		
				2D-GAP	BandDiff+ 2D-GAP	3D-GAP
	1	2	3	4	5	6
B1	2.04	2.25	2.25	2.39	2.39	2.39
B2	2.07	2.41	2.39	2.55	2.47	2.68
B3	1.90	2.17	2.34	2.32	2.43	2.46
B4	2.23	2.74	2.38	2.70	2.49	3.02
B5	1.94	2.42	2.42	2.59	2.54	2.65
B7	1.84	2.19	2.36	2.31	2.46	2.46
AVE	2.002	2.362	2.355	2.477	2.461	2.609

V. LOSSLESS IMAGE COMPRESSION HARDWARE ACCELERATOR

The lossless compression IP core is developed using a clearly defined algorithmic data flow taking into account how it will be translated into HDL and how it will operate on the FPGA. After the floating-point simulation is successfully carried out in Matlab, AccelDSP is used to analyze the design, and translate it into a fixed-point design. The AccelDSP software is a Matlab based tool from Xilinx, which allows DSP engineers to transform a Matlab model into a hardware design that can be implemented using a Xilinx FPGA or other technologies. Its most interesting feature is that a synthesizable RTL design can be generated automatically from a floating-point m-code model. The automatic testbench generation is another valuable feature. The tool also could invoke HDL simulation tools, synthesis tools, and implementation tools.

The architecture of the IP core design is shown in Fig.3. Image data are scanned from the buffer using the scanning method proposed in section III.A. The Scan module is basically a memory address generator, which reads the image data from RAM using a sequence of generated addresses. The Embedded BDC and GAP techniques, described in section III.B, are in one module, eBDC+GAP. GAP requires pixel values from two previous vertical lines. And only pixels in the first line need embedded BDC, which will smooth out this GAP related region. For multispectral images in Band Interleaved by Line (BIL) format or Band Interleaved by Pixel (BIP) format the extra effort to implement 3-D GAP requires only the spectral prediction and a controlled selection between the spectral and spatial prediction. Afterwards, the prediction residuals are mapped to non-negative integers. The coding length of each option is computed and the shortest one is found. Subsequently the entropy coder sends out the compressed bit stream using the chosen coding option along with the option ID. The compressed code is given at the output in

bytes with enable signals. There is a dedicated control logic module (Control Unit), which generates control signals to each block to ensure seamless operation. The byte formatter converts the variable-length code into byte output, with output enable signal.

Achieving maximum efficiency for a particular application is a relatively simple task as it can be accomplished via configuring the model at the algorithmic level. The configurable parameters of the IP core include: the number of bits per pixel, N , the block size, J , the number of blocks of each reference sample interval, R , and the number of blocks of each segment, S [5]. As R depends on the actual application very much, it can also be configured after the implementation with the dedicated 'load' and 'RefIntervalValue' interfaces. These parameters are adjusted according to different imaging scenarios and custom requirements.

Typical configurations of the compression IP core design outlined above have been evaluated. An example is presented here in which the pixel bit-length parameter, N , is varied configuring three designs, labelled A, B, and C, to operate on 8-bit, 12-bit, and 16-bit pixel samples, respectively. The converted RTL designs are synthesized and implemented targeting three different devices from the Spartan-3, Virtex-4 and Virtex-5 series of Xilinx FPGAs. Table 4 summarises the implementation resources of the three designs and of an 8-bit reference JPEG-LS implementation [16] in terms of number of slices at the maximal operating frequency. The results are obtained after place and route, but with normal optimization effort for designs A, B and C. It can be observed from Table 4 that the proposed design is more economical than the reference design and a high throughput of more than 150 Msamples/second could easily be achieved with the Virtex-4 and Virtex-5 chips.

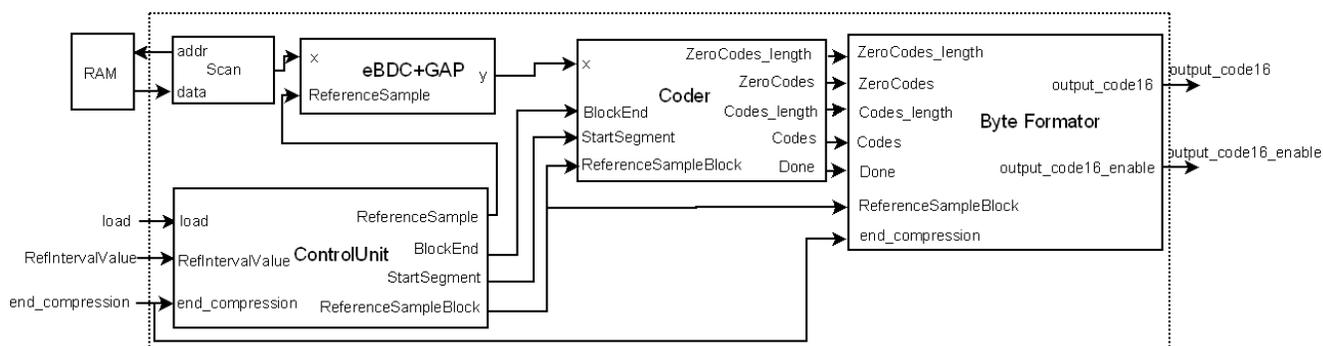


Fig. 3 Lossless image compression IP core design architecture

TABLE 4
IMPLEMENTATION RESOURCES

Designs	Design A 8 bits/pixel	Design B 12 bits/pixel	Design C 16 bits/pixel	CAST JPEG-LS 8 bits/pixel
Spartan-3e	3,610 Slices	5,503 Slices	7,129 Slices	5,636 Slices
1200e-5	@ 113 MHz	@ 75 MHz	@ 75 MHz	@ 64 MHz
Virtex-4	3,454 Slices	5,465 Slices	7,059 Slices	5,591 Slices
LX25-12	@ 224 MHz	@ 188 MHz	@ 166 MHz	@ 112 MHz
Virtex-5	1,760 Slices	2,571 Slices	3,386 Slices	2,182 Slices
LX30-3	@ 282 MHz	@ 250 MHz	@ 210 MHz	@ 166 MHz

The implementation of Design A was tested on the ZestSC2 FPGA prototyping board [17]. In this prototyping system, the host personal computer (PC) communicates with the FPGA and the data memory on the board through a USB interface. First the host PC writes the image data to the data memory, then it downloads the compression IP core to the FPGA and initiates the reading and compressing of the image data, which is followed by transmission of the compressed data back to the PC through the USB interface.

The power consumption of the compression core is estimated using the Xilinx XPower tool. Table 5 summarises the results, which are obtained for idle operation and compression of the Lena image with VCD data generated from Modelsim simulations. As it can be seen from Table 5 the static power is dominating while the dynamic power is only 20-30 mW at 48 MHz. The 3rd and 4th columns in Table 5 contain values obtained after optimization of the design for low power using a gated clock technique. This has achieved reduction of the dynamic power consumption by 28% and 21% in idle state and compressing the Lena image, respectively. As a result of that, the total power consumption over 1 million pixel (Mpixel) throughput is estimated to be 4.5 mW/Mpixels/second, which is more than three times lower than the power consumption of the 3.3 V ASIC implementation of CCSDS-LDC in [18], the value of which is reported as 15 mW/Mpixels/second. The maximal throughput of 80 Msamples/second of this ASIC is also much lower than the results shown in Table 4.

TABLE 5
ESTIMATED POWER CONSUMPTION

IP Core Power 3SP2000@48MHz (mW)	Idle	Lena	Idle (Opt.)	Lena (Opt.)
Dynamic	20.5	29.8	14.6	24.54
Quiescent	188.2	188.3	188.1	188.24
Total	208.7	218.8	202.7	212.77

The power consumption measured on the ZestSC2 prototyping system when compressing the Lena image is around 500 mW, which is in agreement with the power results estimated via XPower as the board has a number of other components in addition to the FPGA.

VI. SYSTEM-ON-CHIP DESIGN FOR PAYLOAD DATA PROCESSING AND CONTROL

The image compression accelerator described in section V is integrated as a peripheral module in a system-on-a-chip design [12, 19,20], which is implemented in an FPGA chip. The SoC is intended to be used as a payload controller on board a small satellite.

A. SoC Architecture for Real-Time Image Compression

The SoC design is targeted at the Xilinx Virtex series of FPGAs. The central processing unit (CPU) is the LEON3 microprocessor, which is a SPARC V8 soft intellectual property core written in VHDL [21]. The SPARC V8 is a RISC architecture with typical features like large number of registers and few and simple instruction formats. However, the LEON3 IP core is more than a SPARC compatible CPU. It is also equipped with various peripherals that interconnect through two types of the AMBA bus (AHB and APB), e.g. Ethernet, SpaceWire, PCI, UART etc. The SoC is an AMBA centric design and subsystems of the OBC can be added to the LEON3 processor providing that they are AMBA interfaced. The AHB is a high-performance system bus and provides high-bandwidth operations. On the other hand, APB is a simple and low-power extension to the AHB bus.

Fig.4 shows the diagram of the SoC architecture, including the LEON3 central processing unit, configuration access port (ICAP) related modules, imaging related modules, memory controllers, and other peripherals. The imaging sub-system handles the raw image data generated by an optical camera and consists of a camera controller, an image compression module and a Direct Memory Access (DMA)/Mass Memory controller. Inside the imaging sub-system, the camera controller also acts as a data router, which could distribute image data to downlink, Mass Memory or the image compression module as follows:

- The first route is to the downlink for real-time capture and download.
- The second route is to the compression IP core for real-time image compression, where the compressed data can be transferred through DMA to downlink for download or Mass Memory for storage.
- The third route is to the Mass Memory for storage of raw images. Through DMA, the image compression core could also compress the stored raw data from the Mass Memory and then transfer the compressed data back to the Mass Memory or directly to the downlink.

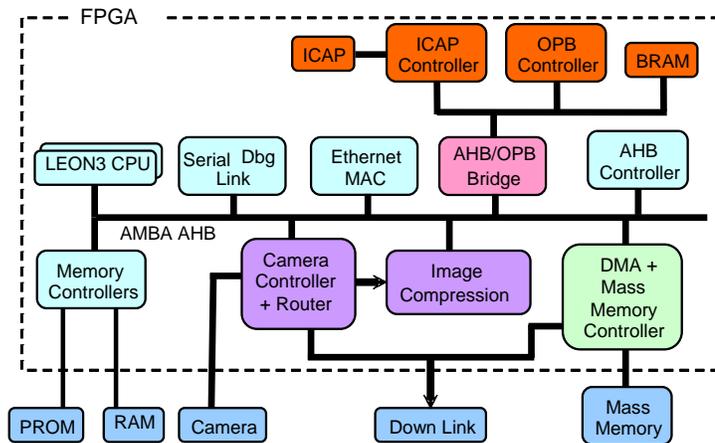


Fig.4 SoC architecture of the payload data processing & control unit

B. SoC Demonstration System

The SoC is implemented in Virtex-4 LX60 on the Avnet evaluation board [22]. The SoC architecture employed by the demonstration system is presented in Fig.5. The image data are downloaded from a PC through an UART and the AHB bus to the DDR memory on the board as this board has no camera input. A dedicated image bus allows the compression core to be clocked with frequency of 100 MHz which is higher than the LEON3 processor frequency of 70 MHz. The DDR controller switch is under the control of the LEON3 processor. The interactions among the LEON3 processor, the compression core and the memory during one compression round are described as follows:

Step (a): LEON3 supplies the compression core with three pieces of data: the starting address of the image data (SADD),

the number of the image pixels (LIMG), and the starting memory address where the compression core will write back the compression codes (DADD).

Step (b): Then LEON3 hands the memory control over to the compression core. The memory can be accessed by both LEON3 (AMBA Bus) and the compression core (Image Bus), but not at the same time. The switch is controlled by LEON3.

Step (c): LEON3 sends the "start compression (SC)" command to the compression core. The compression core starts to read from memory, carries out the compression and finally writes back the compression codes.

Step (d): When the compression core completes the compression, it will write into registers to let LEON3 know the end of the compression (EOC) and the length of the compression codes (LCC).

Step (e): LEON3 takes over the memory control again, if desired, for example to check the compression results.

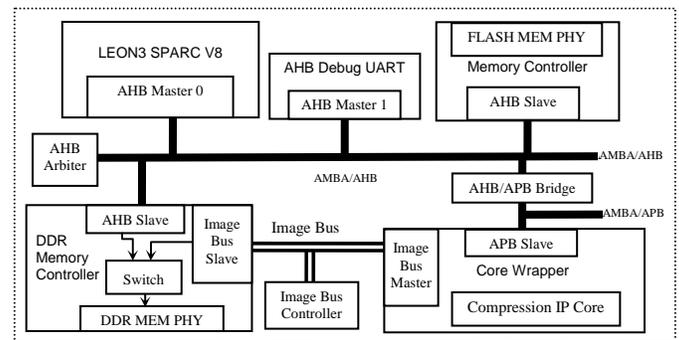


Fig.5 SoC demonstration system architecture

In this demonstration system, the LEON3 debug support monitor GRMON [21] is used to write/read registers or memory data. The actual compressed data stored in the DDR memory after writing the compressed code back to GRMON are compared with the simulated compressed data in Matlab. It is found that the compressed data generated from Matlab and the hardware implementation are exactly the same, as expected.

VII. CONCLUSIONS

Based on analysis of developing trends in current on-board compression systems, a new architecture of an on-board image compression system for future remote sensing missions is proposed. The architecture features intelligent pre-processing and spectral decorrelation to account for future needs of remote sensing.

A new efficient real-time on-board lossless image compression scheme is also proposed, which is suitable for remote sensing panchromatic and multispectral satellite images. A new lossless image compression design is introduced, which is a combination of 2-D prediction and independency coding by using a scanning scheme beforehand. This new design could increase the compression ratio by around 93%, with being only slightly more complex. Its performance is better than the state-of-the-art JPEG-LS, under the same conditions. The system is based on the recommended by CCSDS lossless compression

algorithm for space applications, which features low complexity. The pre-processing stage consists of a new scanning scheme, a modified 2-D prediction method, and a novel 3-D extension prediction technique for multispectral images.

A high-level configurable IP core design is developed which is able to generate the most efficient hardware implementation for a particular application scenario. The FPGA implementation of the IP core consumes low power and its compression ratio is higher than that of the state-of-the-art lossless compression algorithm JPEG-LS. To the best of the authors' knowledge this is the first FPGA based CCSDS-LDC implementation reported in the literature so far.

The compression core is integrated into a SoC platform for satellite payload processing and control. It is attached as a peripheral to the LEON3 processor via the AMBA bus. Demonstration work based on a Virtex-4 FPGA has provided a proof of correctness for the compression core operation and the real-time compression capability of the SoC.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the provision of satellite images from SSTL and DMC International Imaging for the experimental results in this paper.

This research is sponsored by the University of Surrey, an ORS PhD award and EPSRC grant EP/C546318/01.

REFERENCES

- [1] T. Vladimirova, M. Meerman, and A. Curiel, "On-board compression of multispectral images for small satellites", in *Proc. IEEE International Geoscience and Remote Sensing Symposium, IGARSS 2006*, 2006, pp. 3533-3536.
- [2] C. Lambert-Nebout and G. Moury, "A survey of on-board image compression for CNES space missions," in *Proc. IEEE International Geoscience and Remote Sensing Symposium, IGARSS 1999*.
- [3] H. Hihara, M. Sato, K. Fukasawa, H. Sase, Y. Osawa, and N. Ito, "High speed image data compression processor for advanced land observing satellite (ALOS)," in *Proc. 2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering, TENCON '02*, 2002.
- [4] N. Ismailoglu, O. Benderli, S. Yesil, R. Sever, B. Okcan, O. Sengul, and R. Oktem, "GEZGIN & GEZGIN-2: Adaptive real-time image processing subsystems for earth observing small satellites," in *Proc. 1st NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2006*, 2006.
- [5] *CCSDS Lossless Data Compression, Recommendation for space data system standards*, vol. 121.0-B-1: CCSDS, 1997.
- [6] S. J. Visser, A. S. Dawood, and J. A. Williams, "FPGA based satellite adaptive image compression system," *Journal of Aerospace Engineering*, 2003, vol. 16, pp. 129-137.
- [7] T. Vladimirova, A. Steffens. "Compression of multispectral images on-board observation satellites", *Proc. International Conference "Space, Ecology, Safety" (SES'05)*, 2005, vol.1, pp. 105-110.
- [8] J. A. Saghri, A. G. Tescher, and J. T. Reagan, "Practical transform coding of multispectral imagery," *Signal Processing Magazine, IEEE*, vol. 12, pp. 32-43, 1995.
- [9] S. Atek and T. Vladimirova, "A new lossless compression method for small satellite on-board imaging", *WSEAS Transactions on Mathematics*, 2002, vol. 1, no. 1-4, pp. 171-176.
- [10] X. Wu and N. Memon, "Context-based, adaptive, lossless image coding," *IEEE Transactions on Communications*, 1997, vol. 45, pp. 437-444.
- [11] G. Yu, T. Vladimirova, and M. Sweeting, "A new automatic on-board multispectral image compression system for LEO Earth observation satellites," in *Proc. 15th IEEE International Conference on Digital Signal Processing*, 2007, pp. 395-398.
- [12] G. Yu, T. Vladimirova, X. Wu, M. N. Sweeting, "A new high-level reconfigurable lossless image compression IP core for space applications", *Proc. 3rd NASA/ESA Conference on Adaptive Hardware and Systems AHS-2008*, 2008, pp. 183-190.
- [13] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: principles and standardization," *IEEE Transactions on Image Processing*, vol. 9, August 2000.
- [14] G. Yu, T. Vladimirova, and M. Sweeting, "Autonomous band registration for on-board applications," in *Proc. IEEE International Conference on Signal Processing and Communications*, 2007, pp. 1327-1330.
- [15] P.-S. Yeh, "Multispectral Prediction: a two-step predictor," *Personal Communication*, 2007.
- [16] CAST (2007) JPEG-LS Encoder Core — XILINX FPGA Implementation Results. [Online]. Available: http://www.cast-inc.com/cores/jpegls_e-xilinx.htm.
- [17] Orange Tree Technologies (2007) FPGA USB Boards - ZestSC2. [Online]. Available: http://www.orangetreetech.com/fpga_board_zestsc2.html.
- [18] P.-S. Yeh, "Implementation of CCSDS lossless data compression for space and data archival applications," in *Proc. Space Operations Conference*, 2002.
- [19] T. Vladimirova and M. N. Sweeting, "System-on-a-chip development for small satellite on-board data handling," *Journal of Aerospace Computing, Information, and Communication, AIAA*, vol. 01, pp. 36-43, January 2004.
- [20] T. Vladimirova and X. Wu, "On-board partial run-time reconfiguration for pico-satellite constellations," in *Proc. 1st NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2006*, 2006, pp. 262-269.
- [21] J. Gaisler, "GRLIB IP Library User's Manual (Version 1.0.4)," Gaisler Research, 2005.
- [22] Avnet Electronics (2007) 20 Xilinx Virtex-4 LX Evaluation Kit. [Online]. Available: <http://www.avnet.com/>

A Design Space Exploration Flow for FPGA Implementation of Intensive Signal Processing Applications

Sébastien Le Beux, Philippe Marquet and Jean-Luc Dekeyser
LIFL, INRIA Lille-Nord Europe and University of Lille
France
Email: Sebastien.Le-Beux@polymtl.ca

Abstract—Manipulating configurable resources like FPGAs in a co-design framework has become essential: especially since FPGAs may efficiently implement parallel systematic signal processing tasks. Nevertheless, such implementations are usually manually implemented at low level. Our proposition is to provide, on one side, a high level modeling of an application and, on other side, tools to automatically generate tuned VHDL code from these high level models. This paper then introduces a flow able to fit a parallel application onto a FPGA according to the FPGA characteristics, and to map this application onto the FPGA. Each step of the flow requires specific details of the FPGA that realize the implementation, several views of the same FPGA are therefore introduced: BLACK BOX, QUANTITATIVE and PHYSICAL. From the modeling of the application and a view of the FPGA, the flow automatically generates the VHDL code of the initial application and a constraint file that guides the synthesis tools.

I. INTRODUCTION

Current Systems on Chip (SoCs) are heterogeneous and integrate computing, storage, communication and interface resources. The latest generation SoCs often include reconfigurable resources, such as FPGAs, providing flexibility. A FPGA allows the realization of a computing, storage or a communication resource. Manipulating these reconfigurable resources in a SoC design framework has become essential and specific methodologies are proposed (see [2] for example).

The Gaspard co-design framework [6] is more specifically oriented towards the co-design of parallel software and hardware. It identifies the parallelism included in regular constructions such as application loops or repetitive constructions of hardware elements. Gaspard, in its first version, is able to program processor based architectures, but does not allow configuration of reconfigurable resources. In this paper, we propose a Gaspard extension targeting FPGA configuration.

The main challenge of this extension is to use the same description when targeting a processor or a FPGA based architecture. For a processor based architecture, tasks are scheduled and executed on the processor. However, for a FPGA based architecture, a hardware implementation of the application onto the FPGA is realized. The required knowledge of the FPGA characteristics depends on the precision of the required implementation: few details for a first rough design, more details for a tuned RTL design and even more details for a tuned FPGA mapping. Therefore, we identify three different views for the same FPGA, that are associated to the desired precision: BLACK BOX, QUANTITATIVE and PHYSICAL. This paper presents our design space exploration flow and the associated FPGA views.

The paper is organized as follows. The way our modeling of applications allows a factorized expression of parallelism is introduced in Section II. Section III presents our design space exploration flow that manipulates FPGA according to different views. Section IV presents significant results concerning utilization of our flow for intensive signal processing applications. Section V presents related works for high level synthesis and loop transformation methodologies that target a hardware execution. Finally, Section VI concludes this work and gives perspectives.

II. CURRENT GASPARD FRAMEWORK

The Gaspard co-design framework and its main concepts are introduced. We first describe the expression of repetitions in Gaspard. Then we provide an example.

A. ARRAY-OL: *Expression of Repetitions*

ARRAY-OL [10], [3] (Array-Oriented Language) is a language specialized in the description of systematic signal processing applications. These kind of applications are characterized by a huge number of data manipulated by a set of regular tasks. ARRAY-OL allows the description of the *task parallelism* and the *data parallelism* that compose an application. The *task parallelism* expresses dependencies between tasks contained in the application, providing a structural aspect of the application. Each task is described using *data parallelism* in the form of a set of inputs and outputs *patterns* consumed and produced by repeated iterations of the task. These sets of *patterns* tile the input and output arrays of the task. The *patterns* are defined through an *origin* vector, a *paving* and a *fitting* matrix. The *origin* defines the address in the array of a first *pattern*. The *fitting* defines the shape of the *patterns*. The *paving* defines the iteration of the patterns on the arrays. A formal description of the tilers and some examples are provided in [3]. Elementary tasks are black boxes, such as IP, consuming input *patterns* to produce output *patterns*. Elementary tasks represent atomic computations executed in ARRAY-OL.

Gaspard uses the ARRAY-OL concepts in order to express parallelism. Moreover, some ARRAY-OL extensions have been proposed and are also managed in the Gaspard framework. From the ARRAY-OL concepts and its extensions, Gaspard generates different programming languages that allow simulation, execution or synthesis of an application mapped onto an architecture. By generating synchronous languages (Lustre [5] or Signal [26]), it is possible to validate a design application and to detect, for example, dead locks. The automatic generation of procedural languages (e.g. Fortran and C languages) makes possible the execution of concurrent processes onto multiprocessor architectures. The SystemC language allows simulations at different abstraction levels of both application and architecture. The VHDL language makes the synthesis possible, this part of Gaspard environment is presented in this paper.

B. *Application Description in UML*

UML (Unified Modeling Language) is commonly used in model driven engineering community. A *profile* is a UML extension and is a set of *stereotypes* (used to specialize UML classes) and *tagged values* (used to add attributes to these classes). One of the main interests for using UML is that many tools support this language and the extension mechanism (*i.e.* profile). These tools offer opportunity to model a design in a quick and flexible manner. Numerous standard profiles aid to model design in specific fields like the MARTE [29]

profile which is dedicated for Modeling and Analysis of Real-Time and Embedded systems. MARTE provides mechanisms to represent intensive signal processing applications and to express parallelism in a factorized way. Data dependencies expression in MARTE are based on ARRAY-OL, this is a result of the contribution of Gaspard developers to the development of MARTE profile. Moreover, MARTE is under finalization standardization. For all these reasons, Gaspard applications are modeled with MARTE UML profile.

C. Motivating Example: Image Filtering

We illustrate the ARRAY-OL application design in Gaspard using the academic image filtering example. The objective is to correlate well known filters (Gauss, Croix, Sobel, Prewitt, etc.) with pictures. This example is interesting and relevant because of the bi-dimensional arrays and computations being managed.

Figure 1 represents a UML model of the picture filtering application, all the potential parallelism of this application is expressed in a factorized manner. Component *PictureFiltering* represents the application, it is composed of the input port *InputPicture* and the output port *OutputPicture*. Both ports correspond to bi-dimensional data arrays having a shape (i.e. multiplicity) $[(M, N)]$ (M and N represent size of the picture in this example, we use 4×4 pixels in order to illustrate data dependencies later on. Component instance *task* represents atomic computations in this example which is repeated 2×2 times. Its input port *InPattern* (with shape $[(3, 3)]$) reads data in the input picture thanks to a connector with stereotype $\ll\text{Tiler}\gg$ that expresses data dependencies according to ARRAY-OL concepts we introduced above (*origin*, *paving* and *fitting*). In the same manner, output port *OutPattern* (which is a scalar because the elementary task produces a single pixel in this application) writes data onto output picture according to another tiler.

Figure 2 represents this application without factorization of the parallelism in order to illustrate the powerful expression of parallelism available with MARTE profile. The top of the figure illustrates the repetitive iteration of a parallel task, while the bottom side illustrates one iteration at the data parallelism level. At this level, the *patterns* construction issued from tilers appear. Further information is provided in caption of Figure 2. As opposed to usual sequential loops, the repetition specification does not induce any artificial scheduling for task execution and data transaction.

D. FPGA Integration in Gaspard

Several processor based execution models have been introduced in Gaspard to compile an ARRAY-OL application, allowing execution on processors. With the introduction of FPGA in the Gaspard framework, it becomes necessary to study the feasibility for an ARRAY-OL hardware and dedicated execution model. Especially, we have to manage the multi-dimensional arrays and repetitions specific to ARRAY-OL, to efficiently transform this model into hardware, and to define a VHDL code generation from this hardware model.

Our flow performs two efficient transformations: FOLDING and UNFOLDING, which modify loops and repetitive structures in order to optimize a hardware implementation according to an FPGA. Transformation does not introduce control in the datapath computing resource. Control is introduced in the interface with the sensors. Therefore, the static task scheduling is still expressed in the ARRAY-OL language, keeping all the parallelism expression provided by this language.

III. DESIGN SPACE EXPLORATION FLOW

The ARRAY-OL application modeling detailed in the previous section is not directly executable on a FPGA since the repetition ex-

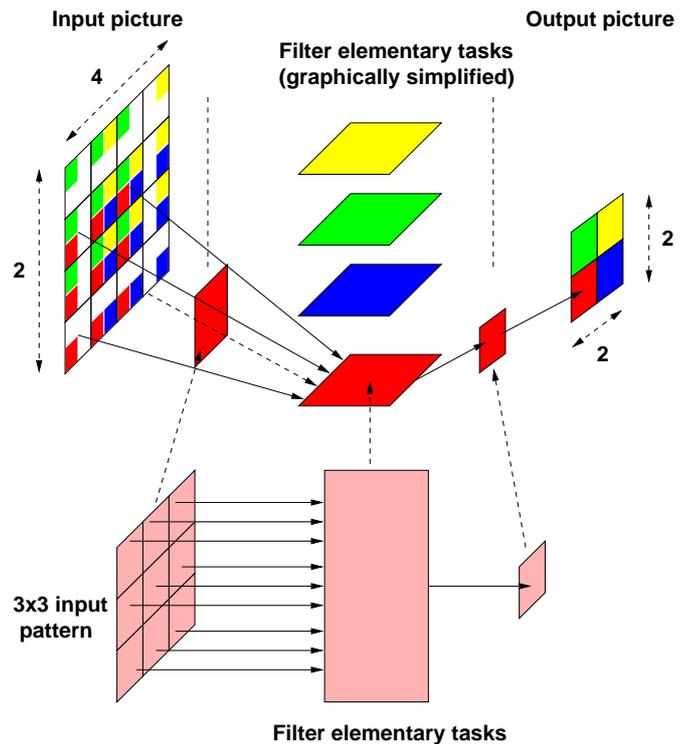


Fig. 2. Image filtering application. The top of the figure illustrates a repetitive iteration for the image filtering application. This application consumes an 4×4 input picture, on the left hand side, and produces an 2×2 picture, on the right hand side. Parallel iterations of the task are illustrated in the top center. The color of each task iteration identifies the corresponding data consumed and produced in the input and output image. The bottom side of the figure represents one task, a 3×3 consumed pattern (read in the input image) and the output pattern is composed of a single pixel, which is written in the output image. Both consumed and produced patterns result from the utilization of a tiler.

pression is factorized, furthermore the target FPGA characteristics are not considered. We present a flow that transforms the initial ARRAY-OL application into hardware, allowing a FPGA implementation. By introducing the FPGA characteristics in the flow, we enhance the application hardware implementation. The flow is illustrated in Figure 3 and is composed of the four following steps:

- the **connection** step analyzes the data dependencies expressed with the *tilers* and generates the appropriate hardware connections;
- the **computation** step adjusts the application implementation with the FPGA, and maximizes its computation power;
- the **routing resources** step adds routing elements on IO when necessary;
- the **code generation** steps generates the VHDL code, which is synthesizable onto a FPGA.

Our design space exploration flow automatically produces a hardware design according to an initial application model. The first step directly produces a hardware design, with no FPGA characteristics consideration. The second and third steps compare the computing resources and the IOs required for a hardware implementation with respect to the resources offered by the FPGA. According to these results, application refactorings are triggered, providing the most efficient hardware implementation that fits onto the target FPGA. For this purpose, two different views of a FPGA are introduced: a

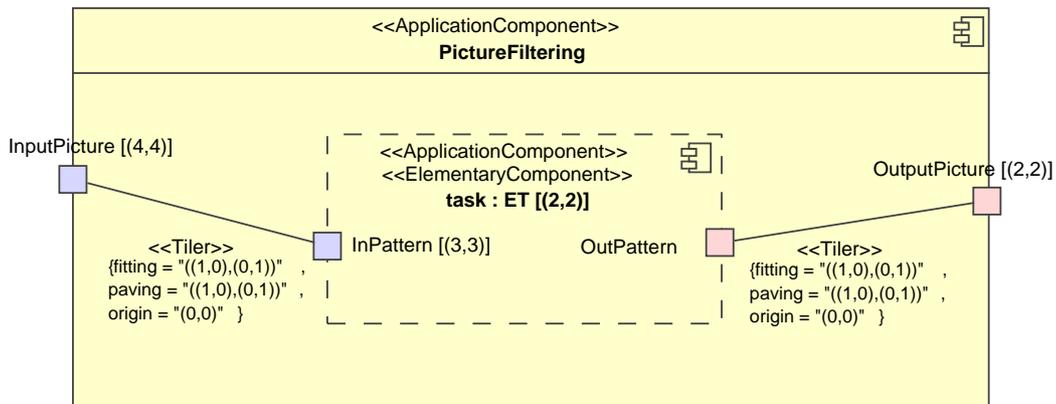


Fig. 1. UML model of the picture filtering application. In this application, Data parallelism is expressed with shape [(2, 2)] above *task* component instance and with connectors stereotyped as <<Tiler>>.

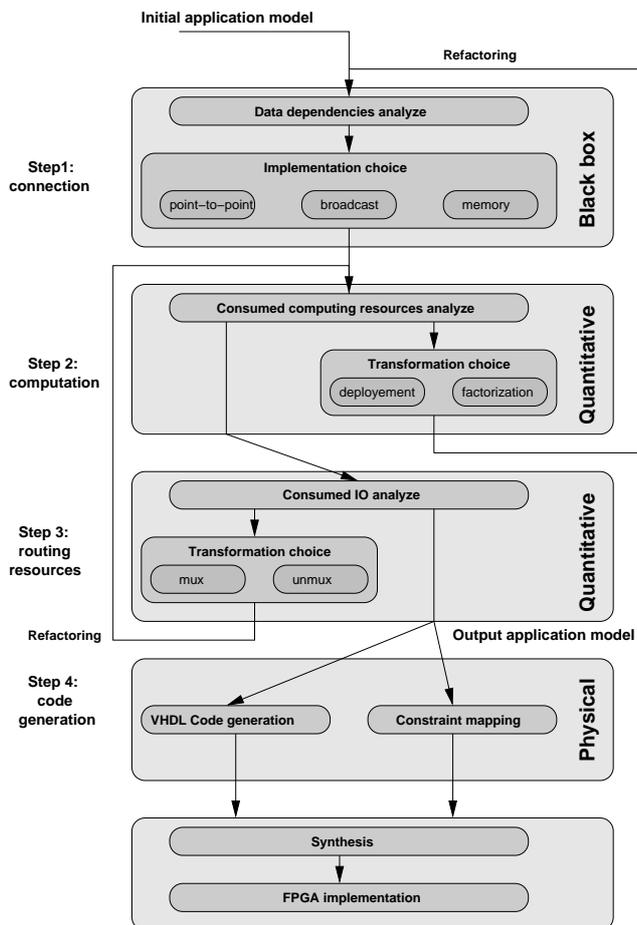


Fig. 3. Our automatic design space exploration flow with the corresponding BLACK BOX, QUANTITATIVE and PHYSICAL views. It is integrated in the Gaspard framework and is composed of four steps. It aims to transform the initial application model into hardware, to optimize this hardware implementation and to generate the corresponding VHDL code. From the generated code, we use external tools to realize the RTL simulation, the synthesis effort and the FPGA implementation.

BLACK BOX view and a QUANTITATIVE view. Our design space exploration flow also generates VHDL code and a user file that constraint mapping of generated hardware design onto FPGA. For this purpose, the FPGA PHYSICAL view is introduced.

A. Connection Step

In ARRAY-OL, exact data dependencies are expressed by the *tilers*, providing synchronizations between tasks. Technically, *tilers* define the way data are read into an array to produce a *pattern*, through *origin*, *paving* and *fitting* matrix. We refer the reader to [3] for details on the analytical formulation. For a dedicated hardware execution, that does not support dynamic data transaction, *tilers* are compiled before the execution in order to create a static connections topology. The aim of this step flow is to provide such hardware connection topology for data dependencies expressed in the initial application.

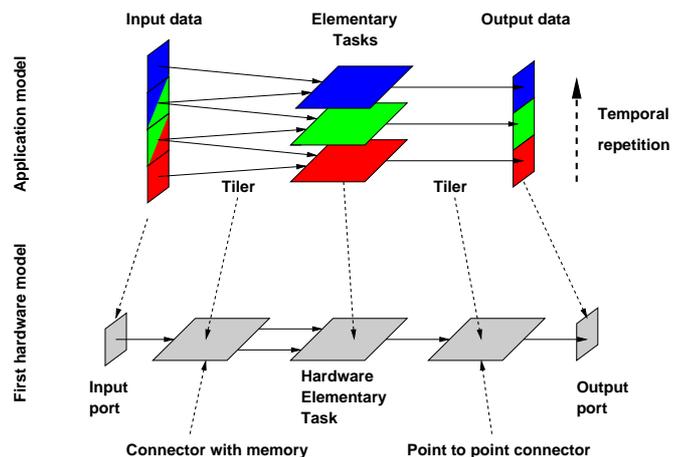


Fig. 4. From the application input model to its hardware implementation using *tiler* connector. According to the top left-hand side of the figure, data are read two times on the temporal dimension. Thus, a “connector with memory” is necessary, and a shift register is automatically created. Looking at the output, we observe that each produced data is written only one time, allowing us to not store the data and to use “point to point connectors”. The temporal dimension around the task is compiled as a single hardware task.

In the connection flow, the data flow described in ARRAY-OL with *tilers* is transformed into hardware elements and blocks. These

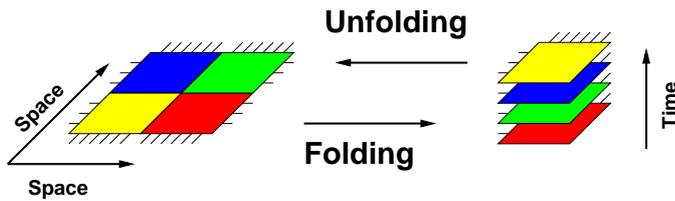


Fig. 5. Transformation that enables application refactoring. The UNFOLDING transformation increases the number of computing resources exploited for a hardware implementation, while the FOLDING one reduces this number.

hardware elements provide a precise quantification of the hardware resources required for an implementation. However, the resulting implementation does not take the FPGA characteristics into account.

The BLACK BOX view provides the simplest view of a FPGA. While implementing a design onto FPGA, no regards to the FPGA details are considered. This view allows to model complex and heterogeneous architectures that includes reconfigurable components [27], provides a fast and direct mapping of a design and allows to manipulate FPGAs with unknown characteristics. However, while some FPGAs are powerful enough to implement a given design, others could fail during the synthesis effort: it introduces an important lost of time. On the same way, a given design could not be implemented in the most efficient manner. The BLACK BOX view is used in the first step of our flow.

B. Computation Step

The computation step evaluates the computing resources required for a particular implementation. If this evaluation is not compatible with hardware resources, a transformation (refactoring) of the application model is required. Both synthesis and estimation provide the computing resources required for an implementation. In order to perform a fast design space exploration, we use estimation results proposed in [24] for each elementary task contained in the application model. This estimator provides an atomic estimation, not a global one, since transformed tilers and the repetition mechanism, which introduces multiple instances of one task, are not managed. Exploring the application model, we perform this global estimation and compare this result with the target FPGA characteristics. Therefore, we obtain a percentage of FPGA area required for an implementation. Two transformations allow to “refactor” the application at the model level. The UNFOLDING transformation increases the implementation size, while the FOLDING one reduces it. Therefore, it is feasible to fit the number of used resources in order to implement the most powerful executable solution on a particular FPGA. Figure 5 illustrates the FOLDING and the UNFOLDING transformations.

ARRAY-OL transformations and estimation tools have been developed by the team, and we are currently working on linking our design flow with those tools. We refer the reader to [9] and [3] for the ARRAY-OL transformation description and the way they may be driven from the Gaspard environment.

The QUANTITATIVE view provides a list of each resource included in a FPGA. Each kind of resource (i.e RAM-512, RAM-1k, DSP blocks, IOs), that composes a FPGA cell appears in this view. However, configurable wires are not considered because of their high complexity, heterogeneity and density. Not considering such wires may introduce a failure during synthesis effort because of the potentially high connection congestion. However, current FPGAs provide enough configurable wires to support a high density connection design. The QUANTITATIVE view makes possible the application

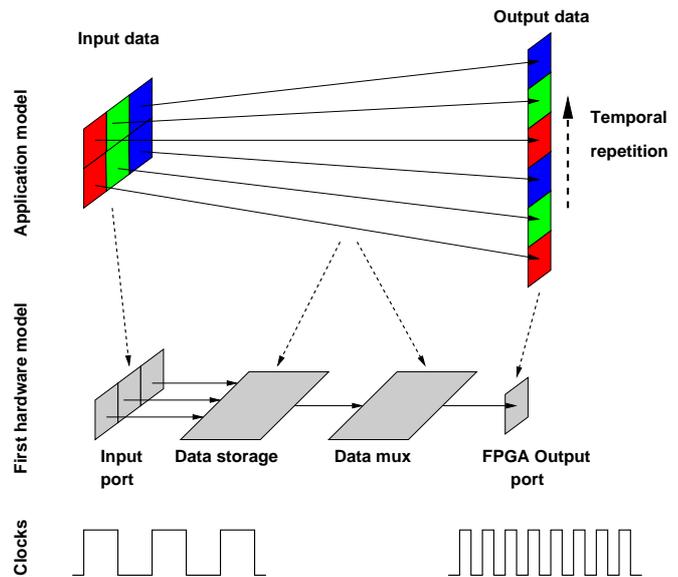


Fig. 6. Data multiplexing toward routing resources. The top of the figure illustrates the data that are read on the spacial and temporal dimensions, and written on only the temporal dimension. For each data input clock cycle, three data are produced on the spacial dimension. They are stored and cyclically sent to the output through a multiplexer. The data output clock cycle is three times higher than the data input one.

tuning, by performing an implementation that corresponds to the available FPGA resources. The QUANTITATIVE view is used in the 2nd and 3rd steps of our flow in order to perform refactoring.

The computation step of the flow compares the area required for an hardware implementation with the one offered by the target FPGA. This step performs application refactoring if necessary. However, it does not compare the inputs and outputs ports required by the implementation with the number of ports provided by the FPGA.

C. Routing Resources Step

Recent FPGAs contain hundreds of inputs and outputs (IOs) ports dedicated to user manipulation. For most part of the application, there are enough IOs. In systematic signal processing applications, where data can be produced by multiple identical sensors, like in multi-dimensional detection applications, the numbers of IOs is a restrictive parameter. A way to consume less IOs is to multiplex the data streams before entering data into FPGA. Inside FPGA, the data streams are de-multiplexed, offering possibilities to realize all computation in a parallel way. This data multiplexing/de-multiplexing is called routing elements in this study, and is illustrated in the Figure 6.

According to the modification introduced by the routing resources, the consumed computing resources are necessarily modified. The design flow then iterates to the second step to take this modification into account (Figure 3).

D. Code Generation Step

When the third step is successfully completed, the VHDL code generation is launched. The generated code allows to simulate and synthesize the resulting application. With a synthesis tool, a FPGA can be configured. This code generation has been successfully validated with several relevant applications inspired from industrial situations, such as the correlation algorithm detailed in [24], matrix multiplications, and the image filtering application taken as an illustrative example in this paper. The automatically generated

VHDL code keeps the parallelism expression provided at a high level description. Moreover, multidimensional arrays manipulation are managed.

Our flow also considers the file constraint code generation. While the generated VHDL code can be used in either synthesis tool, the constraint file does not. To constraint a mapping, the component instances and the number of required specific cells and the position in the grid are manipulated. Manipulating the component instances in the VHDL generated code is possible thanks to the hierarchy kept from the initial application. The mapping of a design onto a FPGA physical view provides an intermediate virtual view, very similar to Lagadec *et al.* [22]. We gather the FPGA cell to provide a virtual architecture that matches the application grain. Therefore, each module contained in the initial application that is clearly identified in the VHDL code is also now clearly identified on the FPGA as a gathering of cells. The generated placement constraint file is provided to the synthesis tool with the VHDL code.

The PHYSICAL view is the most precise view manipulated in this study. BLACK BOX and QUANTITATIVE views are constructed from the PHYSICAL one. PHYSICAL view considers a FPGA like a bi-dimensional grid of cells. Each cell functionality is known (like in the QUANTITATIVE view), and is linked to its position in the grid. With whole individual cells, a very precise view of a FPGA topology is constructed. Current heterogeneous FPGA architectures can be modeled, and mapping heuristics described in [16], [32], [31] can be used. According to the data dependencies constraints, the PHYSICAL view makes possible the design mapping onto a FPGA. The aim of this mapping, constructed from high level evaluation of data dependencies, is to provide a more efficient placement than the one provided by the FPGA mapping tool, which does not consider data dependencies issued from hierarchy, like demonstrated by Moeller in [19]. For this purpose we integrate in our flow, a mapping constraint file that guide the mapping issued by the synthesis tools.

E. Outcomes

In this section, we have shown the different steps necessary to generate an efficient hardware implementation of an application according to a FPGA. Thanks to VHDL code generation, we are able to synthesize and implement the resulting hardware design onto FPGA. The next section illustrates our approach with application example.

IV. CASE STUDY

This section illustrates correctness and efficiency of our flow dedicated to intensive signal processing applications. For this purpose, a correlation algorithm is studied, followed by another application that use dual correlation algorithms.

A. Correlation Algorithm

Correlation is a well known and frequently used algorithm in intensive signal processing. Eq. 1 provides the detailed formulation of the studied algorithm. This algorithm is composed of multiplications and additions that can be executed in a parallel manner, therefore, all potential parallelism is extracted in a UML model detailed in the following. The overall UML model of the correlation algorithm is not fully presented, instead we focus on the interesting part of the model.

$$C_{cy}(j) = \sum_{i=0}^{1023} c(i) \cdot y(i+j) \quad (1)$$

1) *High Level Model:* Top of Figure 7 illustrates component *AdditionTree* that realizes sum in correlation algorithm. Input port *inAdditionTree* is composed of 1024 data (the data to sum), output port is a scalar value (result of the sum). The 10 component instance represent the 10 pipeline stage of tree topology used to realize the sum. The bottom of the Figure 7 represents the 8th component instantiated in the pipeline stage. This component, *AddStep8*, expresses data parallelism. Input port *inA8* is composed of 8 data, output port *outA8* is composed of 4 data and elementary task *a8* is repeated 4 times to realize all the necessary computation.

2) *Synthesis Results of the Generated VHDL Code:* This section illustrates the synthesis results of the automatically generated VHDL code under Quartus tool from Altera. The top of Figure 8 illustrates the 6 last stages of the sum and clearly illustrates the behavior of the tree topology and reduction of data arrays from a pipeline stage to another. Bottom of Figure represents synthesis results for the 8th pipeline stage. Marks **1** and **5** correspond to input and output ports, marks **2x** and **4** identify generated components that resolve data dependencies initially expressed with tilers. Mark **3x** represent the four elementary tasks that realize additions in a parallel manner.

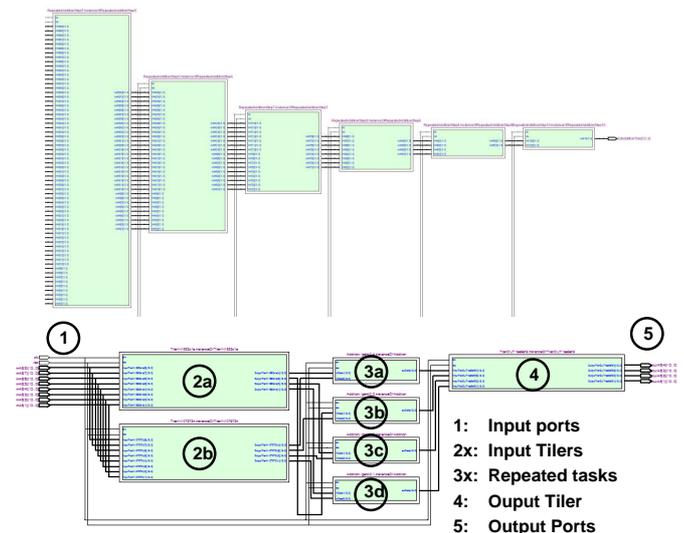


Fig. 8. Synthesis results: top illustrates task parallelism in sum, bottom illustrates data parallelism.

3) *On-line Demos:* The overall algorithm has been modeled in UML and the automatically generated VHDL code has been validated by simulation and synthesis. Three on-line videos are provided in [18], the first video¹ illustrates the complete UML model of the correlation algorithm, the second² illustrates compilation within our design space exploration flow and the third one³ represents simulation and synthesis results of the generated VHDL code.

B. Dual Correlation Algorithm

The aim of this case study is to illustrate the interest of the FPGA PHYSICAL view. Our approach is illustrated with an application that shares data, like described in Eq. 2: data $y(i+j)$ are shared to compute both $C_{cy}(j)$ and $C_{dy}(j)$ correlation. Similarly to the single correlation described above, a UML model was developed.

¹<http://www2.lifl.fr/west/DaRTShortPresentations/correlation-magicdraw.avi>

²<http://www2.lifl.fr/west/DaRTShortPresentations/correlation-eclipse.avi>

³<http://www2.lifl.fr/west/DaRTShortPresentations/correlation-vhdl.avi>

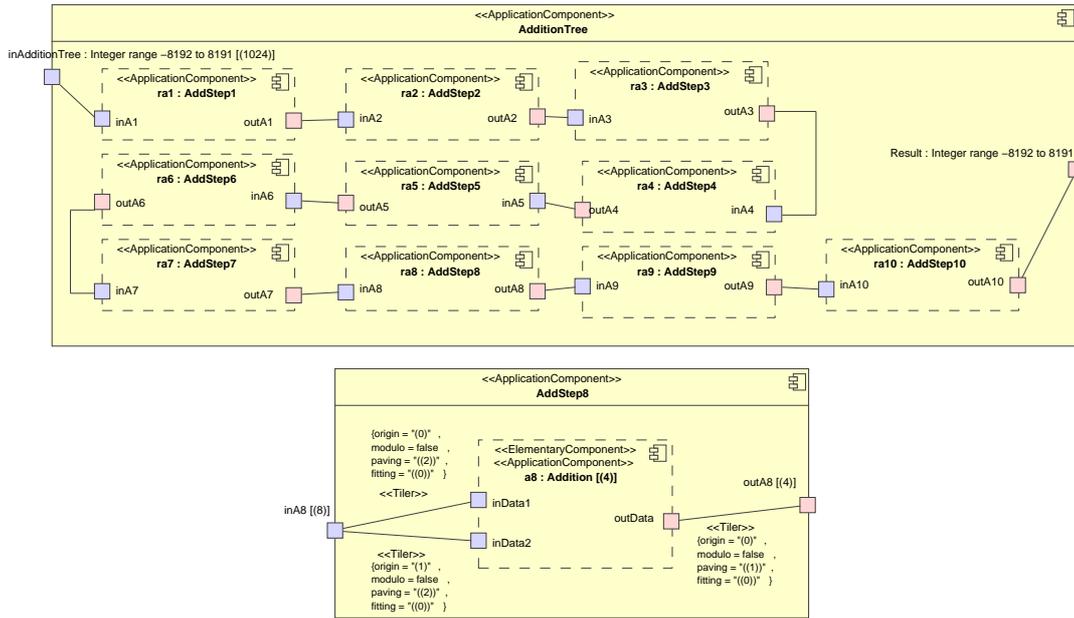


Fig. 7. UML model: the top of the figure illustrates sum in correlation algorithm at task parallelism level, the bottom of the figure illustrates one pipeline stage at data parallelism level.

$$C_{cy}(j) = \sum_{i=0}^{1023} c(i) \cdot y(i+j) \tag{2}$$

$$C_{dy}(j) = \sum_{i=0}^{1023} d(i) \cdot y(i+j)$$

1) *Temporal Data dependencies*: Expressing the overall application in UML, our design space exploration flow finds that data dependencies issued by $y(i+j)$ are efficiently implemented in hardware with shift register. Figure 9 represents partial synthesis result of the produced design that manages these data dependencies. The left side represents the entry in the shift register and the right side represents the several outputs that are connected to shifted data. As opposed to space dependency, such temporal data dependencies management is costly in term of FPGA resources because it consumes CLBs.

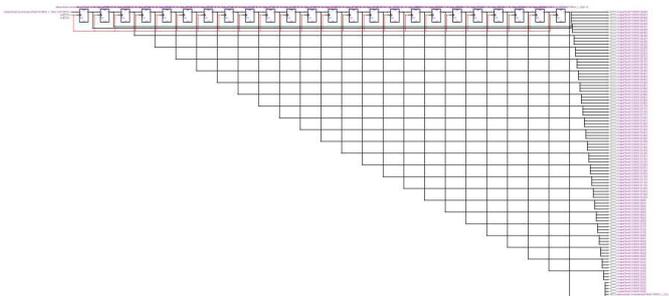


Fig. 9. Synthesis results for the generated shift register.

2) *Placement*: While providing to the synthesis tool the VHDL code generated for the overall application, we observe that the application is mapped in the center of the FPGA, with no regards

to the shared data and the module decomposition that appears in the VHDL code, as shown in Figure 10. The unconstrained solution does not provide a structural implementation, disabling the opportunity to exchange a module functionality. Using a file constraint, we gather shared data in high density FPGA columns, with the computation units around them. In the case study, shared data are issued from generated shift register as described above. Computation of the both correlation are placed around shift region dedicated to shift register.

The structure contained in the VHDL code perfectly matches the structure expressed for this constrained implementation. The synthesis results have demonstrated the equivalent performance for both implementations, with a decrease of the area consumed by the constrained one. This area consumption decrease is linked to the high density implementation required by the constraint mapping. The main advantage of the constraint implementation is the module decomposition that provides an opportunity to enhance the implementation efficiency in some cases, to modify the module functionality and, therefore, to introduce partial reconfiguration in our flow.

C. Outcomes

This section illustrates the interest of the several FPGA views in our design space exploration flow. The BLACK BOX view offers an opportunity to generate a first hardware implementation of a correlation algorithm. Details on optimization thanks to the QUANTITATIVE view can be found in [25]. We illustrate the PHYSICAL view with an intelligent placement of hardware resources that produce shared data by two different algorithms. The generated user file allows to constraint floorplaning during synthesis with Quartus tool.

The PHYSICAL view allows to identify the functionality of each FPGA cell and its associated configuration at a given mapping. Until now, Gaspard applications are expressed using the ARRAY-OL model, which is purely data-flow. Using an extension that introduces a control flow in the data flow [21], [12], we identify, in a design, the activity of an application (i.e active, inactive). Therefore, according to the

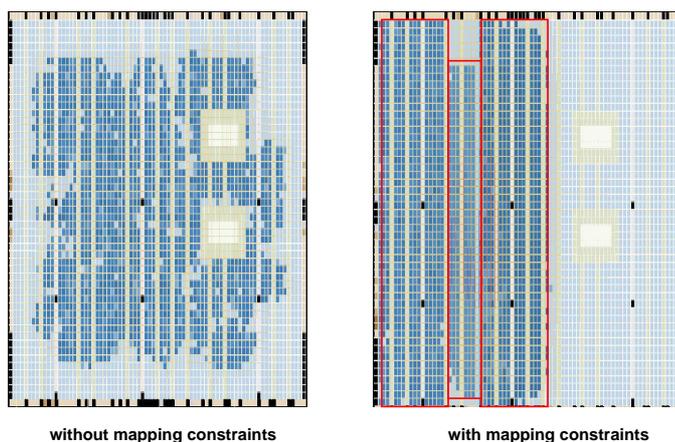


Fig. 10. Synthesis results with and without mapping constraints. Without constraints, the synthesis tool provides a design placed in the middle of the FPGA, with no regards to the application data dependencies. Using constraints, we are able to gather shared data in a single high density FPGA column with the computations units around them.

PHYSICAL view and the control flow, we believe that it is possible to dynamically and partially reconfigure an FPGA [17].

V. RELATED WORKS

Several related works which aim at optimizing a hardware implementation of loop based applications have been proposed.

In [8], Dias *et al.* transform a factorized graph, which expresses the repetition of a task in a compact way. An implementation is obtained by successive defactorizations of the factorized specifications. A heuristic based on a cost function that depends on the consuming resources, the data-rate and the latency, automatically chooses the most efficient defactorized solution. In [20], Kaouane *et al.* extend this work and automatically generate a hardware implementation. Delocalized control units, placed close to the controlled data-path, schedule the execution according to synchronization frontiers. However, while the defactorization transformation is available, the factorization is not.

Guillou *et al.* manage multi-dimensional scheduled uniform recurrence equations with a dedicated controller [13], in the context of the MMAAlpha environment [28]. The controller, generated from a polyhedron scanning method, performs data transactions from local memories and generates enable signals that control the computation. Devos *et al.* describe an extension of CLoog (Chunky Loop Generator) [1], a tool that statically schedules the execution of an application according to its polyhedral model [7]. The extension partially generates the VHDL that describes the hardware controller and the connection with the execution units, improving data locality. The described extension does not take the implementation constraints into account.

Moeller [19] demonstrates that VHDL synthesis and place and route tools do not optimally place a systolic structure onto FPGA. With a manually constraint placement, Moeller enhances the FPGA implementation.

During the last few years, the trend in high level synthesis field is to generate HDL (VHDL or Verilog) code from C or C like code (e.g. Handel-C). Many industrial and academic tools are developed in this topic. One can note C-TO-HARDWARE [23] from Altera, CODE-

VELOPER⁴ from Impulse, DK DESIGN SUITE⁵ from CELOXICA, CATAPULT C⁶ from MENTOR GRAPHICS, etc. STREAM-C [11], ROCCC [14], [4], SA-C [30], SPARK [15], etc. are academic tools that target HDL code generation from C language. There are many others tools but our aim is not to introduce each one, instead, we discuss about them in a general manner. Using C code to generate hardware design is, for sure, an interesting idea because it offers opportunity to work with a well known language and at higher abstraction level than RTL. However, this is not reliable without some major inconveniences.

- textual description is certainly adapted to “little” applications but becomes difficult to handle for hierarchical applications that manage both tasks and data parallelism. Hierarchy appears like function, task parallelism like successive operation and data parallelism like loops. The development of a complex application requires the designer to keep in mind the overall application in a an exact way. Modification of an application requires to handle and move piece of code in the program which is also one of the major problems while working at RTL level. A graphical representation does not have these inconveniences.
- data dependencies are expressed within the index which make multi-dimensional arrays accesses management difficult and error prone. Moreover, a program can become quickly unreadable with such multi-dimensional access.
- C language expresses potential parallelism of an application with sequential loops. Data parallelism extraction from “C to HDL” tools is therefore fastidious and tedious while it could be directly expressed by designer.
- the last point is more philosophical and concerns the claim to use C language because “known by everyone”. In fact, tools often require pragma annotation or extra information close to the C program in order to express parallelism for instance. Pragma annotation syntax depend on each tool: like for any new language, designer has to learn these pragma syntax and the way they are used in the program. Moreover, the overall syntax of C is generally not managed (pointer are often prohibited for instance). In conclusion, a designer does not handle C code but a subset and a constrained C code.

Our proposition differs from these researches in several aspects. Our application modeling takes into account whole applications and not only some academic nested loops. Our application modeling, which is based on ARRAY-OL and its extension [3], is oriented towards the description of systematic signal processing applications. It particularly includes powerful construction that allows to denote all the potential parallelism of an application: data dependencies are explicitly given by the way of high level constructions. Rough data dependencies between arrays will lead to a task parallelism execution while fine grain data dependencies between array elements will lead to a data-parallel execution. Compared with others, our models includes multi-dimensional arrays and our tools include refactoring of arrays that are able to identify temporal and spacial dimensions in arrays. A last key point is the use of UML as input language. This offers the opportunity to graphically model applications and to express, in a unified language and in standardized way, task parallelism, data parallelism and hierarchy.

⁴http://www.impulsec.com/C_to_fpga.htm

⁵<http://www.celoxica.com/products/dk/default.asp>

⁶http://www.mentor.com/products/esl/high_level_synthesis/catapult_synthesis/

VI. CONCLUSION

This paper presents an approach to manipulate reconfigurable resources such as FPGAs in the Gaspard co-design environment. Our design space exploration flow is composed of four steps. The first step identifies data dependencies in the application specification. The second and the third steps compare the computing resources and the IOs required for a hardware implementation with respect to the resources offered by the FPGA. According to these results, application refactoring are triggered, providing the most efficient hardware implementation that fits onto the target FPGA. The last step generates the VHDL code.

In order to perform efficient FPGA implementations of parallel applications modeled at a high level, our flow uses three views that allow to manipulate a FPGA according to the designer requirements: the BLACK BOX view for a first rough design, the QUANTITATIVE view for a tuned hardware design and the PHYSICAL view for a FPGA mapping constraints. The PHYSICAL view is the most detailed view considered in this study. It allows to map an application onto an FPGA in a modular way, taking into account data dependencies and application structure. We believe that FPGA implementation issued from such mapping constraint could benefit from high level application modeling and data dependencies expression. Moreover, this mapping module composition may be used to introduce partial reconfiguration and a regular mapping.

The future works will extend the flow to the partial reconfiguration and the regular mapping. The introduction of a control flow into our data flow language should allow to perform partial and/or dynamic reconfigurations.

REFERENCES

- [1] C. Bastoul, Code generation in the polyhedral model is easier than you think, in: PACT'13 IEEE International Conference on Parallel Architecture and Compilation Techniques, Juan-les-Pins, France, 2004.
- [2] T. Beierlein, D. Fröumlhlich, B. Steinbach, Model-driven compilation of UML-models for reconfigurable architectures, in: 2nd RTAS Workshop on Model-Driven Embedded Systems (MoDES '04), Toronto, Ontario, Canada, 2004.
- [3] P. Boulet, Array-OL revisited, multidimensional intensive signal processing specification, Research Report RR-6113, INRIA (Feb. 2007). URL <http://hal.inria.fr/inria-00128840/en/>
- [4] B. Buyukkurt, Z. Guo, W. A. Najjar, Impact of loop unrolling on area, throughput and clock frequency in ROCC: C to VHDL compiler for FPGAs, in: S. B. . Heidelberg (ed.), Reconfigurable Computing: Architectures and Applications, vol. 3985/2006, 2006.
- [5] P. Caspi, D. Pilaud, N. Halbwachs, J. Plaice, LUSTRE: a declarative language for real-time programming, ACM Press, 1987.
- [6] DaRT Team LIFL/INRIA, Lille, France, Graphical Array Specification for Parallel and Distributed Computing (GASPARD2), <https://gforge.inria.fr/projects/gaspard2/> (2008).
- [7] H. Devos, K. Beyls, M. Christiaens, J. Van Campenhout, D. Stroobandt, From loop transformation to hardware generation, in: Proceedings of the 17th ProRISC Workshop, Veldhoven, 2006.
- [8] A. Dias, C. Lavarenne, M. Akil, Y. Sorel, Optimized implementation of real-time image processing algorithms on field programmable gate arrays, in: 4th Int. Conf. on Signal Processing, ICSP'98, Beijing, China, 1998.
- [9] P. Dumont, Spécification multidimensionnelle pour le traitement du signal systématique, Thèse de doctorat (PhD Thesis), Laboratoire d'informatique fondamentale de Lille, Université des sciences et technologies de Lille (Dec. 2005).
- [10] P. Dumont, P. Boulet, Another multidimensional synchronous dataflow: Simulating Array-OL in ptolemy II, Research Report RR-5516, INRIA (Mar. 2005). URL <http://www.inria.fr/rrrt/tr-5516.html>
- [11] J. Frigo, M. Gokhale, D. Lavenier, Evaluation of the streams-c c-to-fpga compiler: an applications perspective, in: FPGA '01: Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays, ACM Press, New York, NY, USA, 2001.
- [12] A. Gamatié, E. Rutten, H. Yu, P. Boulet, J.-L. Dekeyser, Synchronous modeling of data intensive applications, Research Report 5876, INRIA (Apr. 2006).
- [13] A.-C. Guillou, P. Quinton, T. Risset, Hardware synthesis for multi-dimensional time, in: IEEE 14th International Conference on Application-specific Systems, Architectures and Processors (ASAP 03), The Hague, The Netherlands, 2003.
- [14] Z. Guo, B. Buyukkurt, W. Najjar, K. Vissers, Optimized generation of data-path from c codes for fpgas, in: DATE '05: Proceedings of the conference on Design, Automation and Test in Europe, IEEE Computer Society, Washington, DC, USA, 2005.
- [15] S. Gupta, N. Dutt, R. Gupta, A. Nicolau, SPARK: a high-level synthesis framework for applying parallelizing compiler transformations, in: Intl. Conf. on VLSI Design, 2003.
- [16] M. Handa, R. Vemuri, A fast algorithm for finding maximal empty rectangles for dynamic FPGA placement, in: Design, Automation and Test in Europe Conference and Exhibition, DATE'04, Paris, France, 2004.
- [17] Imran Rafiq Quadri and Samy Meftali and Jean-Luc Dekeyser, An MDE Approach for Implementing Partial Dynamic Reconfiguration in FPGAs, in: 16th International Conference on IP Based System-on-chip, IP'07, Grenoble, France, 2007. URL [IP07.pdf](http://www.lifl.fr/ip07.pdf)
- [18] INRIA, DaRT short presentations and demos, <http://www.lifl.fr/west/DaRTShortPresentations/> (2007).
- [19] T. J. Moeller, Field programmable gate arrays for radar front-end digital signal processing, Ph.D. thesis, Massachusetts Institute of Technology (May 1999).
- [20] L. Kaouane, M. Akil, Y. Sorel, T. Grandpierre, From algorithm graph specification to automatic synthesis of FPGA circuit: a seamless flow of graph transformations, in: 13th international conference on Field-Programmable Logic and Applications, FPL'03, Lisbon, Portugal, 2003.
- [21] O. Labbani, J.-L. Dekeyser, P. Boulet, E. Rutten, UML2 profile for modeling controlled data parallel applications, in: FDL'06: Forum on Specification and Design Languages, Darmstadt, Germany, 2006.
- [22] L. Lagadec, D. Lavenier, E. Fabiani, B. Pottier, Placing, routing, and editing virtual FPGAs, in: 11th International Conference on Field-Programmable Logic and Applications, FPL'01, Belfast, Northern Ireland, UK, 2001. URL <http://www.springerlink.com/content/rvx89nt37g7ln6wr/>
- [23] D. Lau, O. Pritchard, P. Molson, Automated generation of hardware accelerators with direct memory access from ansi/iso standard c functions, in: FCCM '06: Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'06), IEEE Computer Society, Washington, DC, USA, 2006.
- [24] S. Le Beux, V. Gagne, E. Aboulhamid, P. Marquet, J.-L. Dekeyser, Hardware/software exploration for an anti-collision radar system, in: The 49th IEEE International Midwest Symposium on Circuits and Systems, San Juan, Puerto Rico, 2006.
- [25] S. Le Beux, P. Marquet, J.-L. Dekeyser, A design flow to map parallel applications onto FPGAs, in: 17th IEEE International Conference on Field Programmable Logic and Applications, FPL, Amsterdam, Netherlands, 2007.
- [26] P. Le Guernic, J. Talpin, J. Le Lann, Polychrony for system design, Journal for Circuits, Systems and Computers, Special Issue on Application Specific Hardware Design.
- [27] MORPHEUS project, Multi-purpose dynamically reconfigurable platform for intensive heterogeneous processing, <http://www.morpheus-ist.org/>.
- [28] Project Inria-CNRS COSI, ALPHA home page, <http://www.irisa.fr/cosi/ALPHA/>.
- [29] ProMarte partners, UML Profile for MARTE, Beta 1, <http://www.omg.org/cgi-bin/doc?ptc/2007-08-04> (Aug. 2007).
- [30] R. Rinker, M. Carter, A. Patel, M. Chawathe, C. Ross, J. Hammes, W. A. Najjar, W. Böhm, An automated process for compiling dataflow graphs into reconfigurable hardware, IEEE Trans. Very Large Scale Integr. Syst. 9 (1) (2001) 130–139.
- [31] J. Tabero, J. Septién, H. Mecha, D. Mozos, Task placement heuristic based on 3D-adjacency and look-ahead in reconfigurable systems, in: 11th Asia and South Pacific Design Automation Conference, ASP-DAC 2006, Yokohama, Japan, 2006.
- [32] R. Tessier, Fast placement approaches for FPGAs, ACM Trans. Des. Autom. Electron. Syst. 7 (2) (2002) 284–305.

Lossless Multi-mode Interband Image Compression and its Hardware Architecture

Xiaolin Chen, C. Nishan Canagarajah, Jose L. Nunez-Yanez

Department of Electrical and Electronic Engineering

University of Bristol, UK

Email: {Xiaolin.Chen, Nishan.Canagarajah, J.L.Nunez-Yanez}@bristol.ac.uk

Abstract—This paper presents a novel Lossless Multi-Mode Interband image Compression (LMMIC) scheme and its hardware architecture. The approach detects the local features of the image and uses different modes to encode regions with different features. Run-mode is used in homogeneous regions, while ternary-mode and regular-mode are used on edges and other regions, respectively. In regular mode, we propose a simple band shifting technique as inter-band prediction and a gradient-based switching strategy to select between intra-band or inter-band prediction. The advantage of LMMIC is to adaptively switch among modes and predictors in different regions, to avoid complicated calculations in traditional segmentation and inter-band coefficients. This feature enables the hardware amenability. Experimental results show that LMMIC achieves superior compression ratios as well as high throughput. It also provides the benefits of enabling encoding any number of bands and easy access to any band.

I. INTRODUCTION

The rapid advances of multimedia technology generates huge amount of image data. Most of them are multispectral images, which contain more than one spectral bands. For instance, color images, often displayed as RGB, BMP, or TIF format, have at least three bands. In remote sensing, the LANDSAT 7 images have seven bands, and the AVIRIS hyperspectral images contain 224 bands. These images form the base of the widely used web mapping service, e.g. the Google Earth. In medical imagery, multispectral images also prevail. These images are normally compressed for transmission and storage. As many applications, e.g. remote sensing imaging, medical imaging, pre-press imaging and archiving [1], demand perfect reconstruction of images, lossless compression on multispectral images attracts increasing interests. Also for applications that need to transmit image data instantly after acquisition, real-time compression is desirable. To this end, we aim to design an efficient hardware amenable lossless interband image compression scheme, with the capability of real-time processing.

Unlike gray-scale image, multispectral image has not only spatial but also spectral redundancy. Moreover, the existence of multiple bands suggests two problems worth of concern - to encode any number of bands and to access whichever band. As spatial coding techniques have been extensively studied [2][3][4], recently a lot of research focuses on removing the spectral redundancy. Popular interband coding techniques include vector quantisation [5][6][7], Discrete Cosine

Transform [8], Discrete Wavelet Transform [9], and vector-lifting schemes [10]. These coding techniques are efficient in reducing spectral redundancy, but their high computational complexity and often jointly encoding several bands (e.g. 16 bands in [9]) are obstacles for hardware implementation and real-time processing. On the other hand, predictive coding does not only perform well in removing spatial redundancy but also spectral redundancy. Wu extended Intra-band CALIC [2] to Inter-band CALIC [1], which offers one of the best inter-band compression results in literature but requires complex inter-band correlation coefficients calculation and context formation. SICLIC [11] is a simple and efficient coder based on LOCO-I [3], but its 3-band joint-run mode, while giving good bit rates, constrains it from encoding any number of bands.

To relieve these problems, we propose a Lossless Multi-Mode Interband image Compression (LMMIC) scheme. The proposal of this scheme is inspired by the concept of segmentation. Segmentation, in a general sense, is to partition an image into multiple regions in order to change the representation of an image into something meaningful or easy to analyze. However, traditional segmentation, e.g. statistical model-based methods [12] and graph-based methods [13], is too complex to implement in real-time compression. The novelty of our scheme is to apply the idea of segmentation to group pixels with similar features and use different modes to encode them. We propose a multi-mode strategy, where a new ternary-mode is designed to detect and encode the edges, and a run-length coder [3] is to encode the homogeneous regions. The rest of the image, say the texture regions, is coded by a regular-mode which consists of intra-band and inter-band prediction. We propose a simple band shifting technique for inter-band prediction and adapt the Gradient-Adjusted Prediction (GAP) from CALIC [2] for intra-band prediction. A new gradient-based switch is also designed to select the better predictor. This multi-mode strategy applies to all bands in an image. The proposed scheme does not only offer excellent compression ratios, but also the distinctive feature of the flexibility of encoding any number of bands. The multi-mode strategy and switching method make LMMIC hardware amenable. Note that the proposed scheme in this paper is for general purpose (e.g. space, medical, archiving) images with more than one spectral bands but not specifically geared for hyperspectral images. Since some of the techniques for hyperspectral images make specific use of the structure of these images, for example,

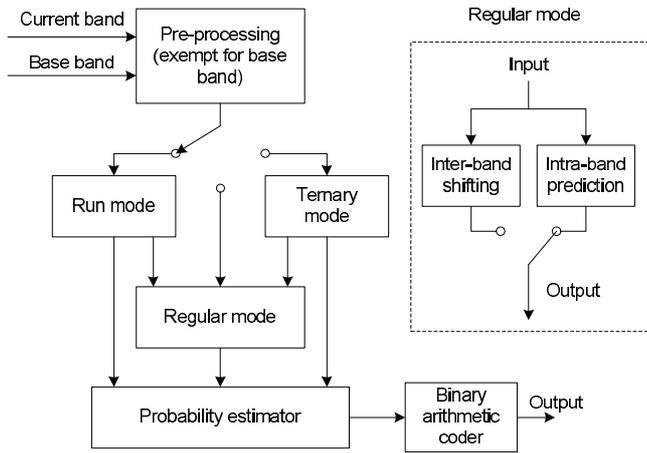


Fig. 1. Schematic of the proposed image compression system

by including a band ordering process [14] or by clustering a number of bands [9], we do not include those highly specialized and not necessarily hardware amenable methods in our comparison study. However, we acknowledge that refining our techniques for specific use with hyperspectral images is an interesting topic for further research, and its findings will be reported elsewhere.

The paper is organized as follows. In Section 2 we present an overview of the proposed scheme. In Section 3 and 4, we explain the details of the multi-mode strategy and the intra/inter-band switching method, followed by the hardware architecture in Section 5. We show the performance comparison with other state-of-the-art schemes in Section 6 and conclude our work in Section 7.

II. AN OVERVIEW OF LMMIC

An image contains many features, such as smooth regions, edges, texture etc. The complexity of an image is an obstacle for compression, thus segmentation (also referred to as region-based methodology) is a viable approach to help with distinguishing these features. The lossless image compression method TMW [15], which achieves the best gray-scale image compression ratio so far, uses segmentation to analyze the image in the first pass. Shen [16] proposed SLIC to combine the region-growing algorithm in segmentation with lossless compression for medical images. Ratakonda [17] used multiscale segmentation in encoding general images and achieved very good compression results. However, they are both complex and two-pass schemes so cannot well meet the real-time processing requirement. Due to the complexity and the nature of segmentation, we skip the conventional methods but apply its concept, by using a simple online checking of neighboring symbols to detect different image features and using different modes to encode these features. This is the idea that our scheme is based on.

Fig. 1 shows the schematic of LMMIC. Firstly, one band is chosen as base band, for instance band G in RGB images, or the first band received in the sequence. Then a pre-processing

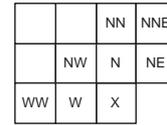


Fig. 2. Neighboring symbols of the current symbol

stage simply subtracts the base band from the current band to get the band difference, as in

$$Band_{diff} = Band_{curr} - Band_{base} \quad (1)$$

Thus each band is only coupled with the base band and no multi-band joint encoding is needed. This allows the flexibility of compressing any number of bands and easy access to any bands. Then a three-way switch enables the system to choose among run-mode, ternary-mode and regular-mode according to the local feature of where the input symbol is. Among these three modes, run-mode and ternary-mode are set to have higher priority and are checked first. Symbols that fail to be encoded in run-mode or ternary-mode are encoded in regular-mode. The diagram in Fig. 1 shows that regular-mode consists of intra-band and inter-band prediction working in parallel, and a switch enables the adaptation. While the base band is directly applied on these three modes, for other bands, only the band difference is applied on run-mode and ternary-mode but both the current band and the band difference are applied on regular-mode, when it is activated. After prediction, higher order redundancy of the image residue is removed by context modeling, which is done in a similar manner with the one in CALIC [2], with some simplification and modification for the intraband and interband case respectively. The output prediction errors and context information are processed by a probability estimator and a binary arithmetic coder [18]. The proposed scheme is a symmetric encoder, as the decoding is just the opposite procedure of encoding. Therefore, both encoding and decoding has the same level of complexity. In the next section, we explain the details of each working mode.

III. MULTI-MODE STRATEGY

Due to the inter-band correlation in images, subtracting the base band from the current band often generates more homogeneous region in the band difference, which is beneficial for compression. On the other hand, many of the image features, such as edges, tend to still remain in the band difference. Therefore, instead of the original band, we apply the band difference on the run-mode and the ternary-mode, and in areas with more complex texture, we can select between intra-band or inter-band prediction, which will be detailed in the next section. In this section, we present how each mode works and the conditions for entering each mode.

A. Run-mode

Run-length coding is simple and efficient in grouping identical symbols [3]. It encodes the occurrence, also called *run*, of a symbol if it appears continuously. We use run-length coding to encode the homogeneous regions of the image. Fig. 2 shows

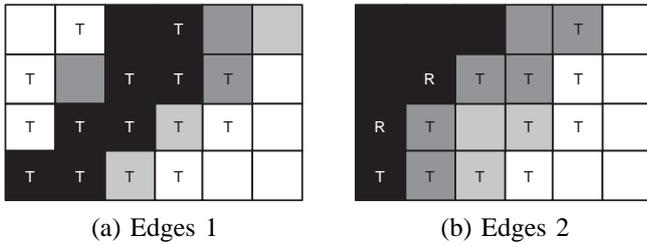


Fig. 3. Areas where ternary-mode is performed

the neighboring symbols of the current symbol X according to their geographical positions. When $W = N = NW = NE$, the current symbol is assumed to be in a homogeneous region and is tried to be encoded in run-mode. If X is identical to W , the run-length increases by one; otherwise “run” stops and the current run-length is encoded. The latter case means that encoding symbol in run-mode is unsuccessful, so the symbol is encoded in regular-mode.

B. Ternary-mode

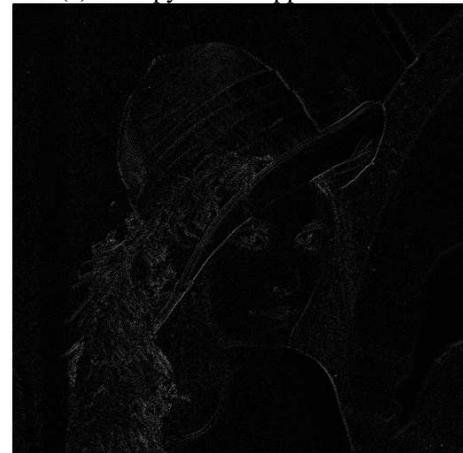
Ternary-mode is inspired by the binary-mode in CALIC, which works on the binary area where there are only two different symbols in the neighborhood, e.g. black and white texts. However, unlike CALIC, our ternary-mode is based on the idea of edge detection. In areas where a sharp edge occurs, shown in Fig. 3 (a), symbol values at the two sides of the edge are usually different; also, where a less sharp edge occurs, as in Fig. 3 (b), symbol values tend to be changing gradually. In both cases, we assume that within a small neighborhood of the current symbol, say the seven neighboring symbols in Fig. 2, there are no more than three distinctive symbols and the ternary-mode is triggered. Fig. 3 shows the areas that the ternary-mode is performed. T indicates the symbols encoded by ternary-mode, while R indicates run-mode, and the color is the gray-level of the symbols. The figure tells that edge areas can be largely covered by this mode. In ternary-mode, symbol W is represented as s_1 , while the second and third distinctive symbols are represented as s_2 and s_3 respectively. In other word, the current symbol can be denoted by their order of appearance given the condition that the checking of the context is always conducted in the same order. The current symbol is encoded by

$$T = \begin{cases} 0, & \text{if } x = s_1; \\ 1, & \text{if } x = s_2; \\ 2, & \text{if } x = s_3; \\ \text{escape}, & \text{otherwise.} \end{cases} \quad (2)$$

“Escape” happens when encoding in this mode fails. It is a way of switching among modes. The alphabet size for encoding in this mode is only four so lower entropy is obtained. Ternary-mode also works as a “backup” of the run-mode in smooth but not exactly homogeneous regions. Fig. 4 are the residue images of “lenagrey” after GAP and LMMIC respectively. The zero order entropy of the residue image after GAP is 4.39bpp while it is 4.31bpp after LMMIC. Fig. 5 (b) shows the regions



(a) entropy = 4.39bpp after GAP



(b) entropy = 4.31bpp after LMMIC

Fig. 4. Sample residue images after GAP (a) and LMMIC (b)

where different modes perform, comparing with the original image (a). Run-mode works on the grey homogeneous regions; ternary-mode works on the white regions, which often lie on the edge of the homogeneous regions, some smooth regions or clear edges; regular-mode works on the dark regions, which are mostly texture or noisy areas.

C. Regular-mode

Regular mode is triggered, either when the entry conditions for run-mode and ternary-mode cannot be met, or when encoding in other modes fails. Regular mode consists of intra-band and inter-band prediction, which is selected according to the local features adaptively. Details of the inter-band prediction and switching strategy are discussed in next section.

IV. GRADIENT BASED SWITCHED INTRA/INTER-BAND PREDICTION AND CONTEXT MODELING

We design a simple band shifting technique for inter-band prediction, and adapt the GAP from CALIC for intra-band prediction. However, the performance of inter-band prediction depends on the band correlation. In the case of strong band



(a) original image “bike3”



(b) different modes indication on image “bike3”, where grey area indicates run-mode; white dots indicate ternary-mode; dark area indicates regular-mode.

Fig. 5. Original image (a) and image indicating where different modes applied (b)

correlation, inter-band prediction is preferred, otherwise intra-band prediction is selected. A gradient based switching method is proposed for the selection.

A. Band shifting for inter-band prediction

In the regions where bands are strongly correlated, symbol changes in one band often happen in another band. Fig. 6 shows the plots of one line in band G and band B in the image “peppers”. It is clear that the dot plot from band G has a similar trend with the dash plot from band B. We intend to shift the reference band G to a position that is close to the current band B so that only a small difference between the current band and the shifted reference band needs to be encoded. There are a lot of possible ways to predict the value

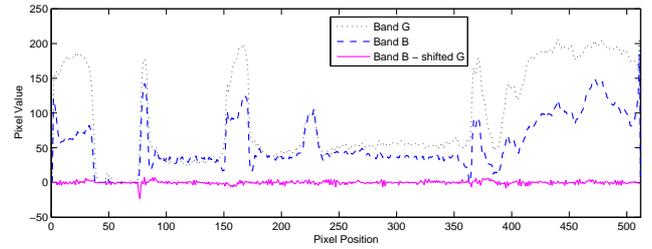


Fig. 6. Plots of one line in band G and band B and their difference after shifting

for band shifting. Since this band shifting is only used when band correlation is strong, the band difference tend to be very small. Considering the level-set preserving property of the simple median predictor, we use it to predict the band shifting. We denote the band difference at position W , N , NW as W_diff , N_diff , NW_diff , and calculate the band shifting by

```

if NW_diff >= max(N_diff, W_diff)
    shift_band = min(N_diff, W_diff);
else if NW_diff <= min(N_diff, W_diff)
    shift_band = max(N_diff, W_diff);
else
    shift_band = N_diff + W_diff - NW_diff;
end

```

The solid plot in Fig. 6 shows that this prediction method successfully generate a zero-mean band difference between the current band and the shifted reference band.

B. Gradient-based switching

As the performance of the two predictors varies in different regions of an image depending on the spatial and spectral correlation, it is critical to design a suitable selecting strategy to decide which one to use. As we aim at designing a hardware amenable scheme, complex calculation of inter-band correlation coefficients has to be avoided. We propose a simple switching method according to the local horizontal and vertical gradients, which are calculated by

$$\begin{aligned} dh &= |W - WW| + |N - NW| + |N - NE| \\ dv &= |W - NW| + |N - NN| + |NE - NNE| \end{aligned} \quad (3)$$

where dv is the vertical gradients and dh is the horizontal one. When calculating the inter-band gradients, W , N , NW , NE , NN , WW , NNE are substituted by the inter-band difference at the same positions. Inter-band gradients indicate how closely the two bands change in the same way. In addition to the gradients, the previous prediction error is taken into account to evaluate how well the predictor performs in the local area. Therefore, for both intra-band and inter-band prediction, we calculate

$$S = dv + dh + |E_w| \quad (4)$$

where E_w means the prediction error at position W . The predictor that gives smaller S is selected to encode the current symbol. Experiments on the test images used in the results section show that this simple method makes 70% or above right decisions.

C. Context Modeling

Context modeling is to group the prediction residue based on local contexts in order to obtain a lower conditional entropy. In the proposed scheme, context is formed in a similar manner with CALIC [2] but is simplified to reduce the memory usage. We take 6 context symbols (W, N, NW, NE, NN, WW) to compare with the predicted value \hat{X} to obtain a texture pattern t , representing the local texture feature. Also, to indicate the activity of errors in a context, a coding context is generated with the local gradients dv, dh and a previous prediction error e of W . The coding context is then quantized into 8 levels to form a coding context index. Combining the texture pattern and the coding context, a set of 512 compound contexts are formed with 6 bits texture pattern t and 3 bits coding context index. In the case of interband context formation, context symbols are replaced by the band difference at the same position. These contexts are also used to generate an bias cancellation of the predictor, which will be discussed in the next section. The 8 coding contexts are used to calculate the occurrence probability of symbols in the probability estimator presented in Section V-B.

V. HARDWARE ARCHITECTURE

Hardware amenability is one of the priorities in the design of the proposed scheme. Therefore, as previously described, the whole procedure, including the prediction and mode switching, only involves a limited number of addition and shift, and only need to buffer three lines in two bands. In this section we propose the suitable hardware architecture to support the proposed compression scheme, which includes the architecture of the prediction and context modeling module, probability estimator and binary arithmetic coder. As the binary arithmetic coder is adopted from [18], we will not explain in detail here.

A. Prediction and Context Modeling Module

To further optimize the hardware architecture, the data flow of the prediction and context modeling module is achieved with two pipelines running in parallel, as shown in Fig. 7. Line 1, indicated by the flow on the left, operates on the current symbol and yields the symbol “runs” or symbol order or prediction error with the selected mode for the probability estimator; Line 2, indicated by the flow on the right, updates the data from Line 1, calculates the prediction value and the context index for the next symbol under the selected mode. The advantage of dividing the procedure into two parallel pipelines is, while not violating the sequential constraint, to halve the execution time and hence obtain higher throughput. As the arithmetic calculation part is introduced in previous section, we will explain the rest, which is managing the context

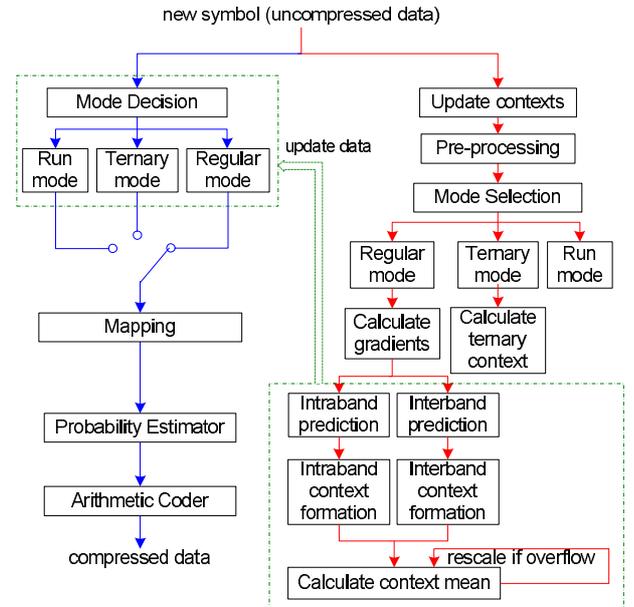


Fig. 7. Data Flow of the prediction and context modeling module

memory and the error feedback technique for bias cancellation, below in this subsection.

In the compression process, we need to store 3 lines of image pixel values in memory as context. To avoid reading in 3 lines every time when a line finishes, we use 3 lines' memory to store the pixel values and 3 pointers to indicate symbol locations in each line. At the end of processing each line, the 3 pointers to each line rotate in a certain order so that the oldest line is discarded and the newly formed line is saved.

An error feedback technique is used in the prediction step to adjust the bias in prediction in certain context. As prediction can not be accurate, the mean of errors \bar{e} is assumed to be the most probable prediction offset error in each context, and hence is a good observation of the bias of the predictor. We improve the prediction by adding on this term. It is calculated by

$$\bar{e} = \text{sum} / \text{count} \quad (5)$$

where sum and count are the sum and occurrence count of errors in the context, respectively. The calculation of mean requires arbitrary division, which is not desirable in hardware implementation. To solve the problem, we store the count with only 5 bits ($2^5 - 1 = 31$). When the count reaches its maximal value 31, it is halved by right-shifting one bit; meanwhile sum is halved so as to maintain the mean \bar{e} . Thus we only need 13 bits ($2^5 \times 2^8 = 2^{13}$, assuming the error range is 0 to $2^8 - 1$) plus one sign bit to store the sum of errors safely. Experimental results prove that this rescaling technique slightly improves the compression ratio by “aging” the observed data. However, division is always a difficult problem in hardware, especially when the dividend can be as large as 13 bits. To make this division practical, we bound the dividend sum by 10 bits for two reasons: firstly,

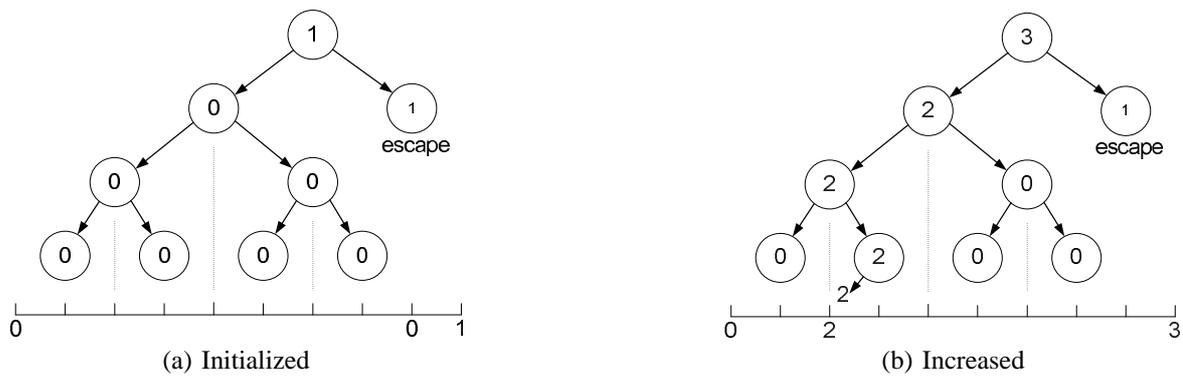


Fig. 8. Simplified tree structure of the probability estimator

experiments on our image test set show that the chance of the *sum* being larger than 1023 is less than 0.001%; secondly, extraordinary large errors tend not to reflect the true behavior of the context because prediction errors should be moderately small given an adequate predictor. Therefore, we use the most significant bits of the divisor *count* in the division, with the dividend being rescaled accordingly to maintain the same result. Consequently, we only need a lookup table of 1KByte ($2 \times 512 = 1024$) to complete fast division. Although the result of division is only an approximation, it does not affect the compression performance in our experiments.

B. Probability Estimator

As the prediction residue generated by the prediction and context modeling module has a multiple-symbol alphabet (e.g. for 8 bits per pixel, there are $2^8 = 256$ symbols), they cannot be processed in a binary arithmetic coder directly. We propose a probability estimator to adaptively calculate the probability of symbol occurrence in each context in a SRAM, and to decompose the data into bit level. Thus it enables the application of a simple and efficient binary arithmetic coder and hence results in full pipelining and high throughput.

The main part of the probability estimator is a tree structure model stored in a SRAM. Each context is represented by a balanced binary tree with 2^n (n is the bits per pixel) nodes associated with each symbol in the alphabet. A number of bits are used to store the symbol frequency count in each node. Initially, all the symbols in the alphabet are assigned an equal probability, and the whole range of the probability is 1. When one symbol is received, the value of the corresponding tree node increases to reflect the probability distribution of symbol occurrence. We demonstrate the working theory of the model with a simplified tree structure, as shown in Fig. 8, where $n = 3$. Firstly, in Fig. 8(a), each tree node is assigned to 0, and the whole range of the probability is assigned to “*escape*”. “*Escape*” here means coding is not successful. Then in Fig. 8(b) a symbol “2” (“010” in binary) arrives, so the counts for symbol “2” is increased by 2, or any specified value, and the probability of symbol “2” becomes $2/3$. This is reflected by the value of the tree nodes over the total value of the tree root. In this way, to encode the symbol “2”, we only need to

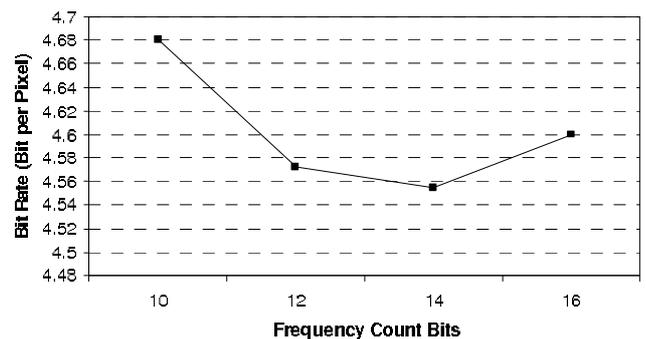


Fig. 9. Average Bit Rates under Different Probability Precision

encode the left or right decision when the symbol comes down through the tree.

As mentioned in previous section, there are 8 coding contexts, corresponding to 8 “dynamic” trees and one “static” tree for coding the *escape* symbols. *Escape* happens when a valid probability of a symbol cannot be found, e.g. when its probability is 0, in which case the symbol is “escaped” to the “static” tree and is sent as it is. *Escape* is undesirable as it does not achieve any compression. It takes place when some symbol frequency counts reach the maximal frequency count, e.g. 14 bits for $(2^{14} - 1)$, in which case all the symbol frequency counts in the tree will be halved. Consequently, the counts of symbols that have not been seen before will be rescaled to 0, resulting in *escape* when those symbols occur later for the first time. Therefore, the frequency count bits have to be carefully chosen. Experimental results of average compression bit rates under different frequency count bits are shown in Fig. 9. When the maximal frequency count is too small, more *escapes* are likely to happen; when the maximal frequency count is too big, the “aging” effect of observed data is less. Therefore, we choose 14 bits according to the result of Fig. 9.

Fig. 10 shows a simplified diagram of the probability estimator. There is a SRAM memory where the probability of symbols in each coding context is stored, and a SRAM to store the total root value of each context tree. The incoming symbol, which is the prediction error, is shifted and compared with

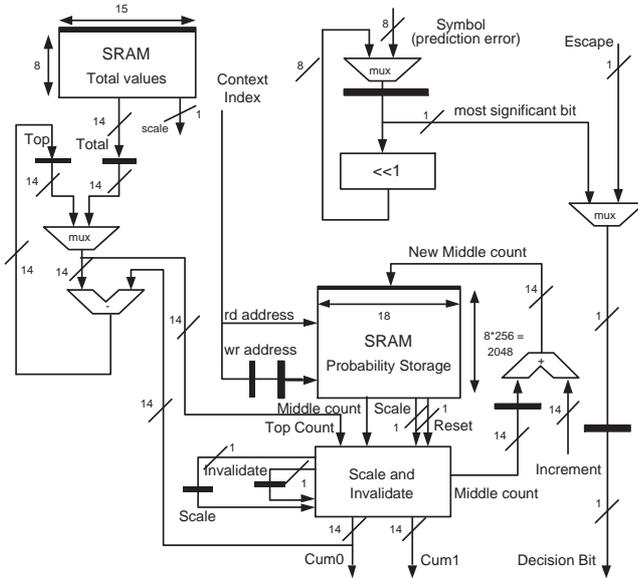


Fig. 10. Architecture of the probability estimator

the value “1” to get the decision bit (0 or 1, as left or right). Meanwhile, the context index is read in the corresponding context tree, where the *middlecount* represents the current symbol and the *totalcount* represents the total value in the context. So the probability is calculated as

$$p = \frac{\text{Middle Count}}{\text{Total Count}} \quad (6)$$

The probability is output as *Cum0* and *Cum1* as a binary sequence. This module maps the probability data into a set of binary decisions from the root to the leaves through each context tree. The binary arithmetic coder is driven by these decision bits and the probability data. It is multiplication-free, resulting in an improved clock ratio of the system. One decision bit is processed per clock cycle, and hence 8 cycles are needed for encoding one byte.

VI. PERFORMANCE COMPARISON

The experimental result in terms of compression ratio is presented in this section. We choose a set of standard 3-band RGB images, a 4-band CMYK image “park” and a 7-band LANDSAT 7 image “coastal” from CCSDC (the Consultative Committee for Space Data Systems). The RGB images include continuous-tone images (“cats”, “water”, “lena” and “peppers”), compound images (“cmpnd1” and “cmpnd2”) and synthesized images (“bike3”). We compare the proposed compression scheme with JPEG2000 [19], intra-band CALIC [2], IB-CALIC [1] and SICLIC [11]. The results of IB-CALIC and SICLIC are extracted from [11]. Some results are absent due to the unavailability of the programs. JPEG2000 is the current standard for image compression. The results of IB-CALIC are one of the best in literature in terms of general inter-band image compression, but not hyperspectral image compression, which is not the scope of our proposed method either. And SICLIC is a good trade-off between compression

ratio and complexity. Table. I shows that LMMIC outperforms JPEG2000 and intra-band CALIC by 10% and 14%, respectively. It is superior than SICLIC on average, though slightly inferior than IB-CALIC which has higher computational complexity. Since the inter-band coding in LMMIC only couples two bands, it has the flexibility of compressing images with any number of bands and easy access to any bands. The proposed hardware architecture together with the binary arithmetic coder enables the system to process one bit per clock cycle, which translates into a throughput of around 123Mbits/s on a Xilinx Virtex 4 FPGA.

VII. CONCLUSIONS

An original Lossless Multi-Mode Interband image Compression (LMMIC) scheme is proposed. The concept of segmentation is well ingrained in this scheme to deal with different regions in the image adaptively. The simple and efficient band shifting technique and the switching strategy successfully remove the inter-band redundancy. Experiments show that LMMIC achieves highly competitive compression ratios and provides the flexibility of compressing any number of bands as well as easy access to any bands, which are not offered by many other schemes. The complexity of the scheme is strictly controlled and hardware amenability is maintained. The proposed corresponding hardware architecture proves the achievement of high throughput.

ACKNOWLEDGMENT

The authors would like to thank the support from EPSRC under grant EP/D011639/1.

REFERENCES

- [1] X. Wu and N. Memon, “Context-based lossless interband compression – extending CALIC,” *IEEE Trans. Image Processing*, vol. 9, no. 6, pp. 994–1001, June 2000.
- [2] X. Wu and N. Memon, “Context-based, adaptive, lossless image coding,” *IEEE Trans. Commun.*, vol. 45, no. 4, pp. 437–444, Apr. 1997.
- [3] M. J. Weinberger, G. Seroussi, and G. Sapiro, “LOCO-I: A low complexity, context-based, lossless image compression algorithm,” in *Proc. Data Compression Conference*, Mar. 1996, pp. 140–149.
- [4] X. Li and M. T. Orchard, “Edge-directed prediction for lossless compression of natural images,” *IEEE Trans. Image Processing*, vol. 10, no. 6, pp. 813–817, June 2001.
- [5] M. J. Ryan and J. F. Arnold, “The lossless compression of aviris images by vector quantization,” *IEEE Trans. Geosci. Remote Sens.*, vol. 35, no. 3, pp. 546–550, May 1997.
- [6] J. Mielikainen and P. Toivanen, “Improved vector quantization for lossless compression of AVIRIS images,” in *Proc. XI Eur. Signal Process. Conf.*, Sept. 2002.
- [7] G. Motta, F. Rizzo, and J. A. Storer, “Compression of hyperspectral imagery,” in *Proc. Data Compression Conference, DCC’03*, Mar. 2003, pp. 333–342.
- [8] P.L. Dragotti, G. Poggi, and A.R.P. Ragozini, “Compression of multi-spectral images by three-dimensional SPIHT algorithm,” *IEEE Trans. Geosci. Remote Sens.*, vol. 38, no. 1, pp. 416–428, Jan. 2000.
- [9] X. Tang, W. A. Pearlman, and J. W. Modestino, “Hyperspectral image compression using three-dimensional wavelet coding,” in *Proc. SPIE*, Jan. 2003, vol. 5022, pp. 1037–1047.
- [10] A. Benazza-Benyahia, J. C. Pesquet, and M. Hamdi, “Vector-lifting schemes for lossless coding and progressive archival of multispectral images,” *IEEE Trans. Geosci. Remote Sens.*, vol. 40, no. 9, pp. 2011–2024, Sept. 2002.

TABLE I
BIT RATES (BITS PER PIXEL PER BAND) COMPARISON ON SELECTED SCHEMES

image	JPEG2000	CALIC	IB-CALIC	SICLIC	LMMIC
cmpnd1	1.44	1.21	1.02	1.12	1.05
cmpnd2	1.30	1.22	0.92	0.97	0.97
cats	1.75	2.49	1.81	1.85	1.82
water	1.41	1.74	1.51	1.45	1.44
lena	4.53	4.40		4.46	4.23
peppers	3.41	4.62		3.25	3.35
bike3	5.17	4.21		4.41	4.29
average	2.72	2.84		2.50	2.45
park	5.72	5.39			5.30
coastal	2.89	2.68			2.62

- [11] R. Barequet and M. Feder, "SICLIC: a simple inter-color lossless image coder," in *Proc. Data Compression Conference, DCC'99*, Mar. 1999, pp. 501–510.
- [12] J. Kim, J. W. Fisher, A. Yezzi, M. Cetin, and A. S. Willsky, "A non-parametric statistical method for image segmentation using information theory and curve evolution," *IEEE Trans. Image Processing*, vol. 14, no. 10, pp. 1486–1502, Oct. 2005.
- [13] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *Internation Journal of Computer Vision*, vol. 59, no. 2, pp. 167–181, Sept. 2004.
- [14] S. R. Tate, "Band ordering in lossless compression of multispectral images," *IEEE Trans. Comput.*, vol. 46, no. 4, pp. 477–483, Apr. 1997.
- [15] B. Meyer and P. E. Tischer, "TMW - a new method for lossless image compression," in *Proc. Int Picture Coding Symp.*, Oct. 1997.
- [16] L. Shen and R. M. Rangayyan, "A segmentation based lossless image coding methods for high-resolution medical image compression," *IEEE Trans. Med. Imag.*, vol. 16, no. 3, pp. 301–307, June 1997.
- [17] K. Ratakonda and N. Ahuja, "Lossless image compression with multiscale segmentation," *IEEE Trans. Image Processing*, vol. 11, no. 11, pp. 1228–1237, Nov. 2002.
- [18] J. Nunez-Yanez and V. Chouliaras, "A configurable statistical lossless compression core based on variable order Markov modeling and arithmetic coding," *IEEE Trans. Comput.*, vol. 54, no. 11, pp. 1345–1359, Nov. 2005.
- [19] D. S. Taubman and M. W. Marcellin, *JPEG2000 Image Compression Fundamentals, Standards and Practice*, Norwell, MA: Kluwer, 2002.

A Clustered Architecture for Scaling Video Processing Across Multiple Machines and Multiple Cores

Timothy Harris ^{#1*5}, Tughrul Arslan ^{%6}, Bruce Devlin ^{#2}, Joe Diggins ^{#3}, Iain Lindsay ^{%7}, Kaaren May ^{#4}

[#] *Snell & Wilcox Ltd., UK*

¹ tim.harris@snellwilcox.com

² bruce.devlin@snellwilcox.com

³ joe.diggins@snellwilcox.com

⁴ kaaren.may@snellwilcox.com

^{*} *Institute for System Level Integration, UK*

⁵ tim.harris@sli-institute.ac.uk

[%] *Edinburgh University, UK*

⁶ Tughrul.Arslan@ee.ed.ac.uk

⁷ Iain.Lindsay@ee.ed.ac.uk

Abstract—This paper presents the design considerations and performance results from implementing clustering to accelerate a video processing algorithm in a commercial product.

Index Terms—clustering, standards conversion, video processing, benchmarking

I. INTRODUCTION

In 2007 Snell & Wilcox brought its new Alchemist IP product to market. The product made the company's video standards conversion technology available as software for the first time. Initial customer feedback was that the processing was too slow (60 to 80 times run-length¹), and so a distributed architecture was implemented to allow the software to take advantage of a cluster of off-the-shelf PCs[1][2]. After the implementation, it became apparent that the architecture would also allow the software to take advantage of multi-core processors. This paper presents the design decisions made to reach the final architecture and the performance characteristics of that architecture.

Video standards conversion is used to convert between the different video standards used internationally (e.g. NTSC and PAL). The primary difference between the standards is the frame-rate of the video, which can be either 50Hz or 60Hz². The processing algorithm that was clustered for this work, performs frame-rate conversion from 50Hz to 60Hz and 60Hz to 50Hz.

The challenges of clustering the Alchemist IP product included:

- The video processing algorithm had temporal dependencies which meant that each frame of the video sequence could not be processed separately.
- The video quality of the output was a key feature of the application, so it had to be maintained.
- The Alchemist IP product was undergoing active development at the time, so the changes to the application had to be minimal.

¹This means that a 10 second video clip would take 600 to 800 seconds to process.

²For historical reasons the frame-rate of the video in the US is actually 59.94Hz.

II. UNCLUSTERED ALCHEMIST IP ARCHITECTURE

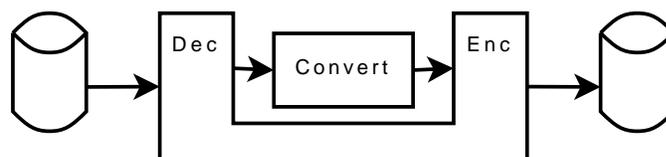


Figure 1. The flow of video through the Alchemist IP product with the main process decoding and encoding, while communicating with a frame-rate conversion process.

The dataflow through the Alchemist IP product has the following steps: decoding video from a compressed input file, sending the uncompressed video to the frame-rate conversion algorithm, encoding the resulting uncompressed video and then writing out to a new file (see Figure 1). The frame-rate converter runs in a separate process and communication between the converter and the rest of the product is through a pair of TCP/IP sockets. In the Alchemist IP software the majority of the processing time is spent in the frame-rate converter.

The processing algorithm in the Alchemist IP performs motion compensated frame-rate conversion[3]. The algorithm was originally designed for real-time processing and implemented in hardware using FPGAs. This has led to an algorithm that is processor intensive and contains temporal dependence.

III. CHOOSING A CLUSTER ARCHITECTURE

When designing the clustering architecture, different arrangements for the data flow were considered. The system was designed to handle multimedia content, which tends to involve large amounts of data. The overhead of moving the large amounts of data was considered to be the primary factor in determining the efficiency of the system[4].

In a share-nothing cluster (see Figure 2) each node reads a chunk of the input data, does all the processing required and then writes a chunk of the output file. The share-nothing cluster arrangement follows the “move the processing, not the data” philosophy. The share-nothing cluster arrangement

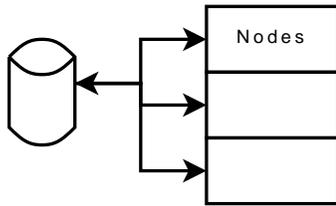


Figure 2. A share-nothing cluster where each processing node communicates directly with the storage device.

has three major advantages: efficiency (the data is only moved twice, minimising network bandwidth usage), scalability (scales directly by adding nodes with no single-node bottlenecks) and ease of deployment (each node is identical). Unfortunately the share-nothing cluster arrangement cannot be used in all circumstances. An example of this is when the data store is not accessible to all the nodes, or if the data processing requires access to the entire input file (for temporal processing). Another disadvantage to the share-nothing cluster is that it requires a complex control system to ensure the entire input file is processed and that the output from each node is in the correct place in the overall output.

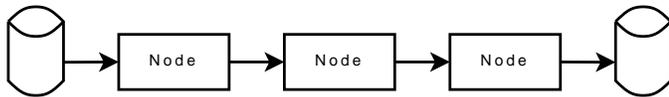


Figure 3. A cluster where each processing step is undertaken on a different node.

In the sequential cluster arrangement (see Figure 3), each node performs a single task on the input file and then passes its results onto the next node. The sequential arrangement moves the data between nodes, so it is the exact opposite to the share-nothing cluster arrangement. The main advantage of the sequential arrangement is that existing media processing code can be used without change. Each processing node processes the entire file and can perform as much temporal processing as required (e.g. multi-pass encoding). The main problem with the sequential arrangement is its lack of efficiency due to large communications overheads compared to the share-nothing arrangement. The extra overhead is due to the data copies which have to take place between nodes in the sequential arrangement. Also, balancing the workload across all the nodes can be difficult, making scalability expensive, as one of the nodes will always be the bottleneck of the system. Another issue with the sequential arrangement is that when dealing with failed nodes the entire file will need to be reprocessed, whereas in a share-nothing cluster arrangement only a single chunk of the input would need to be reprocessed.

The head-node arrangement (see Figure 4) is where a single node is responsible for all file I/O, handles splitting the work over the other nodes in the cluster and joining together the results. A benefit of using the head-node arrangement is that all temporally dependent processing (e.g. encoding and multiplexing) can be done in the head node using existing code. Also the head-node arrangement will work in environments where only one machine has access to the input and output file data stores.

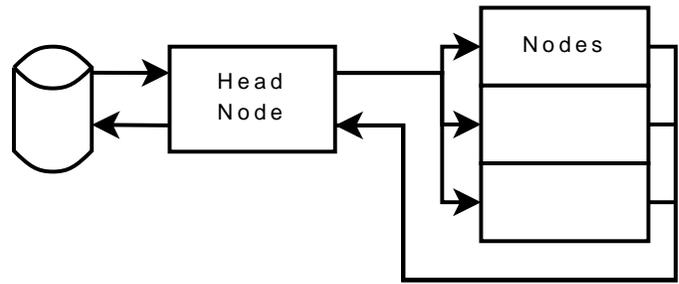


Figure 4. A cluster where all communications with the storage device is through a single node.

The disadvantages of using the head-node arrangement are that the head node is vulnerable to being the single point of failure and will eventually become the bottleneck to throughput as the cluster grows.

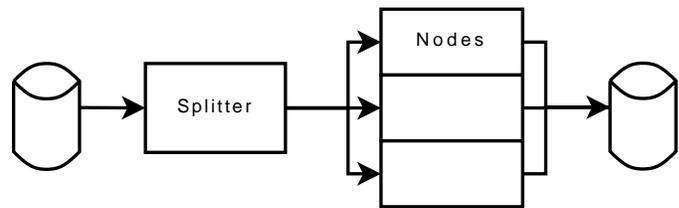


Figure 5. A cluster where the data is split across multiple nodes by a splitter node.

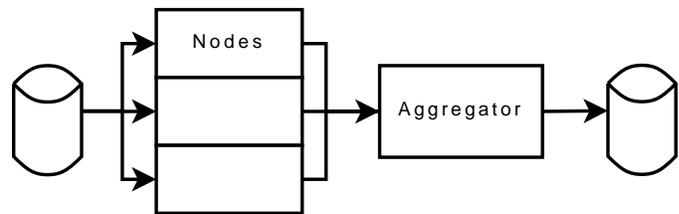


Figure 6. A cluster where the output of the cluster is aggregated by a final node.

There is a continuum of possible cluster arrangements between the share-nothing and the sequential arrangements. The head-node arrangement can be thought of as a midpoint in that continuum. Examples of other points on that continuum include the splitter and aggregate arrangements as shown in Figures 5 and 6. Part of the challenge of designing a system to utilise a cluster is deciding where in the continuum it should reside. The factors to consider include the processing being undertaken (both in terms of temporal dependency and workload), the location of the data being accessed, the number of machines in the cluster, whether the cluster is going to grow in size and the effect on any existing code being utilised. It might well be desirable to vary the arrangement of the system as and when the processing task being undertaken changes.

For the Alchemist IP the head-node arrangement was chosen by the author. The key factors in the decision included:

- The requirement that the clustering needed to fit into the existing code base. Most of the application code runs on the head node, with the processor intensive frame-rate converter being clustered across the other nodes.

- Only a small part of the system was being clustered, so we can afford to rewrite that part of the system to be as close to a share-nothing cluster as possible.
- The algorithm being clustered was very processor intensive, so the head-node bottleneck was not an issue.

IV. IMPLEMENTING THE CLUSTER ARCHITECTURE

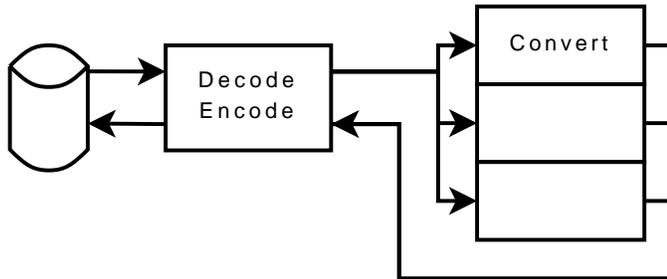


Figure 7. A clustered Alchemist IP with the main application running on the head node and the frame-rate conversion running on the rest of the cluster. This figure is a combination of the dataflow from Figure 1 and the cluster architecture from Figure 4.

The dataflow of video through the clustered Alchemist IP software has the decoding, splitting, joining and encoding all on the head node and the frame-rate conversion is performed on the rest of the nodes (see Figure 7). Uncompressed video is carried between the head node and the rest of the cluster. This decouples the frame-rate converter from the various I/O formats handled by the Alchemist IP product and allows the bulk of the application to live on the head node, while distributing the processor intensive part.

To implement clustering in the existing code base some new code was required to split the given video sequence into chunks, send the chunks to multiple frame-rate converters and join the output into the final output sequence.

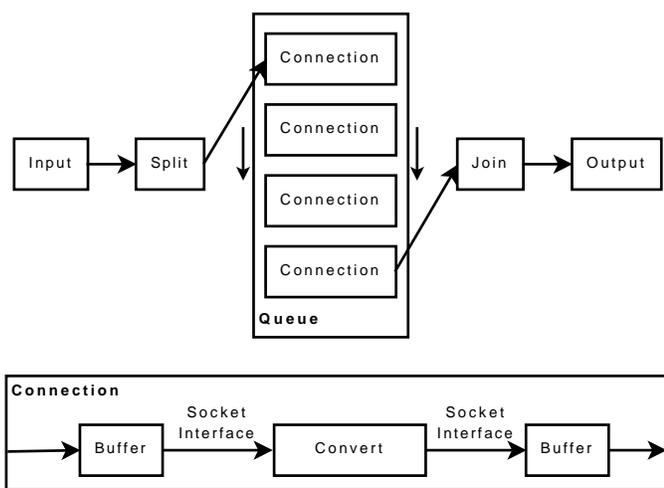


Figure 8. Cluster data-flow diagram. This diagram shows how the splitting and joining code utilised a shared queue of active connections to ensure the output data was produced in the correct temporal order. A new Connection was created for each chunk the video sequence was split up into.

The Connection class handles the connection to the frame-rate converter and the buffering. When splitting the given video

sequence into chunks each chunk was buffered and fed to a frame-rate converter using an instance of the Connection class. The results were buffered so they could be correctly sequenced into the output. Each instance of the Connection class handled one chunk of the input sequence. Figure 8 shows how the processing and queuing worked to keep the output data in the correct order.

Buffering was required in the Connection class for several reasons. The clustering code had data delivered to it and did not have control over the source of the data (i.e. the code was not able to seek through a file). So, in order to allow parallel operations, the data had to be buffered (either to memory or disk) before it was sent to the algorithm for processing. As the clustering code had to produce a single, temporally-contiguous, output, the output from each converter had to be buffered until all the results were read out of the previous Connection. The amount of buffering required caused clustering to be a memory intensive task for the head node. The buffering could have been reduced, but this would have required a significant redesign of the existing system.

V. VIDEO SEQUENCE SPLITTING

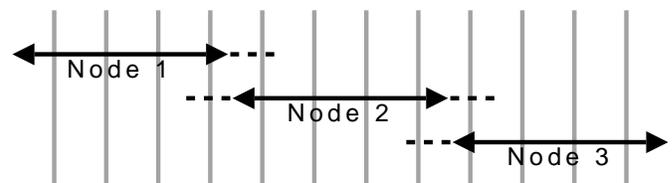


Figure 9. Splitting a sequence into 4-frame chunks to send to various processing nodes. Each grey line is a frame in the input video sequence. The dotted lines show the overlapping frames sent to multiple nodes (pre-roll and post-roll).

The input video sequence was split into chunks as shown in Figure 9. The number of frames in each chunk could be varied to optimise the utilisation of the cluster. In order to maintain the quality of the output from the frame-rate converter, it was necessary to send extra frames of video before and after each chunk, known as pre-roll and post-roll. These overlapping frames helped to overcome the temporal nature of the frame-rate converter’s algorithm. The frame-rate converter’s pre-roll and post-roll sizes were determined through algorithm analysis and a set of tests to be two frames of pre-roll and one frame of post-roll. The Peak Signal to Noise Ratio (PSNR) tests described in Section VI-C checked whether the quality of the video was being maintained when using the clustering code. The technique described above for spreading the work among the nodes is very different to traditional clustering techniques for image and video processing[5], [6], [7], [8], [9]. This is because the frame-rate conversion algorithm requires more than one frame of data and cannot be segmented spatially.

VI. EXPERIMENTS

Several experiments were carried out on the newly clustered Alchemist IP application. These included experiments to determine the number of frames per chunk to use, to determine how the clustered application behaved on a multi-core machine

and to check the clustering was not affecting the output picture quality of the application.

A. Frames per Chunk Experiment

The number of frames per chunk was a significant parameter to the clustering of this application. Ideally the system should be setup to use the smallest number of frames per chunk that does not excessively impinge on performance. These tests were designed to find the optimum number of frames per chunk. The 60Hz to 50Hz tests were run on a 10 machine cluster and the 50Hz to 60Hz tests were run on the same cluster using only 7 machines. The tests used 5000 frames per input sequence. The author tested both 50Hz to 60Hz and 60Hz to 50Hz conversions to determine whether there was any benefit in making the number of frames per chunk equal to the input or output frame rate.

As the graph in Figure 10 shows, beyond 30 to 40 frames per chunk, performance is almost flat and then beyond 70 frames per second the processing time starts to rise. The rise in processing time and variation in processing time, above 70 frames per chunk, was probably due to the increasing memory usage on the head node. The graph shows that a 20-30% performance improvement was achieved by choosing the optimum number of frames per chunk. The graph also shows that the optimum number of frames per chunk was not affected by the input and output frame-rate.

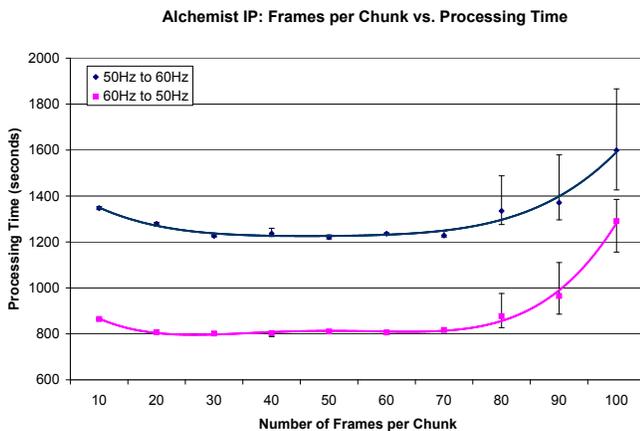


Figure 10. How job time varied with frames per chunk for two common frame-rate conversions. The results demonstrate that beyond 30 frames per chunk the returns in increased performance diminished and above 70 frames per chunk processing time increased significantly.

B. Multi-core Experiments

The effect of Moore’s Law on processor manufacturers has moved from increasing clock speeds to increasing the number of processor cores per chip. This means that the same techniques for distributing processing across machines, can be applied to speeding up the application on a single machine. An experiment was undertaken to test how the clustering code performed on a single machine with multiple cores. The experiment used a 5000 frame sequence, varied the number of nodes used (all nodes were run on the local machine) from 1 to

6 and varied the frames per chunk from 10 to 90 frames. Each test run was repeated 3 times to determine the average and variability of the results. This experiment was run on a dual-processor, dual-core machine (giving a total of four cores).

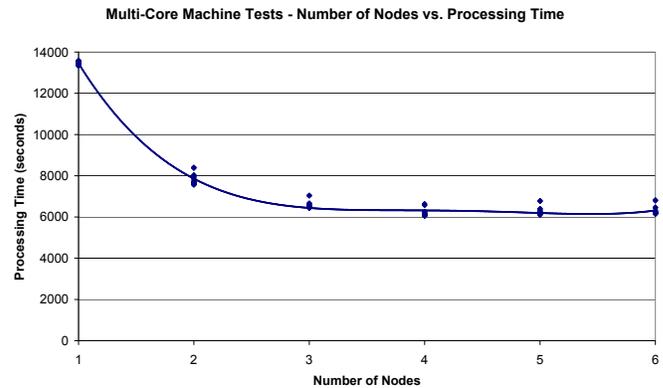


Figure 11. How the number of nodes affected processing time when running on a four-core machine. The apparent variance in the measurements is because the frames per chunk was varied for each number of nodes (see Figure 12). The graph shows a distinct leveling-off in processing time when the number of nodes was one less than the number of cores in the machine. A 2.2 times speed-up was achieved, going from 0.3 frames per second to 0.8 frames per second.

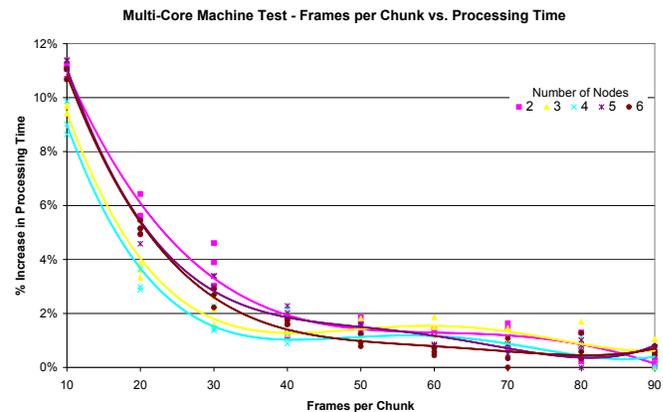


Figure 12. How varying the frames per chunk affected the processing time when running the clustered Alchemist IP on a single four-core machine. The processing time is displayed as a percentage increase from the minimum processing time for that particular number of nodes. The graph shows that system performance increased with frames per chunk, but that there were diminishing returns going beyond 50 to 60 frames per chunk.

Figure 11 shows the speed-up that occurred as more nodes were used on the multi-core machine. The graph shows that the performance leveled out as the number of nodes was one less than the number of cores in the machine. This result makes sense as one of the cores was being used for IO, decoding and encoding. Figure 12 shows how varying the frames per chunk sent to each node affected the job time. The graph shows that the overall trend was independent of the number of nodes and that there were diminishing returns over 50 to 60 frames per chunk. Looking at the results, the variation among the repeats was under 1%, which can be regarded as insignificant.

C. Checking output quality

To check that the clustering did not affect the quality of the result, a test was devised to run both the clustered and unclustered versions of the Alchemist IP and take a PSNR measurement between the two outputs (using the unclustered output as ground truth). PSNR was chosen as it was simple, quick to implement and widely used. The PSNR calculation used the following equations:

$$PSNR = -10 \log_{10}(MSE)$$

$$MSE = \frac{\langle (p - q)^2 \rangle_{\text{across the whole picture}}}{255^2}$$

$p = 8\text{-bit luminance value for pixel from ground truth picture}$

$q = 8\text{-bit luminance value for pixel from comparison picture}$

$$\langle PSNR \rangle = -10 \log_{10} \langle MSE \rangle$$

Testing with a sequence of 100 frames, using 10 frames per chunk and 3 servers the measured PSNR in the luminance channel was 123dB over 239 output frames (100 input frames create 239 output frames when converting from 50i to 59.94p). Looking in more depth at the output, only one frame was different between the clustered and unclustered outputs and that was only a few pixels with slightly different values. The test was repeated with a couple of longer sequences, which were chosen as they were difficult sequences to standards convert. The results were:

- Average PSNR for the sequence was 78dB over 359 output frames.
- Average PSNR for the sequence was 89dB over 2035 output frames.

The resulting output was inspected and only in a few cases were the differences in the frames visible (see Figure 13). In these cases the differences were determined to be inconsequential and were orders of magnitude below the artifacts produced by frame-rate conversion. In conclusion the clustering had almost no perceivable effect on the resultant pictures.

VII. FUTURE WORK

There are several follow-on investigations which may be undertaken. These include:

- Incorporating mezzanine compression between the head-node and the processing nodes to reduce the network bottle-neck. This is anticipated to become important when the system starts to be used to process high-definition video.
- Investigating various techniques for load-balancing the system when using heterogeneous nodes.
- Developing a cluster architecture expert system to implement all the various architectures described in Section III. The system would choose the best achievable architecture for the current job.

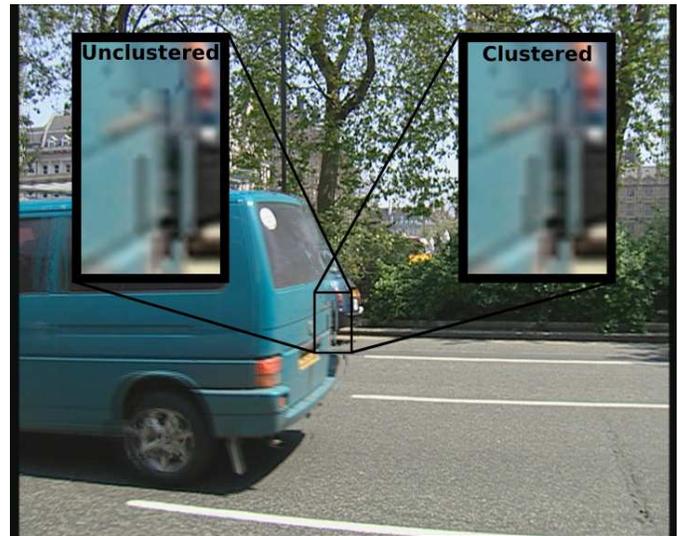


Figure 13. Worst-case difference between the unclustered and clustered outputs of the Alchemist IP. The picture above highlights the section of the output frame with the most visible artifact caused by clustering. The highlight on the left is from the unclustered output, while the highlight on the right is from the clustered output. The main visible difference is the grey vertical line in the middle. In the unclustered version the line is straight, in the clustered version the line is slightly staggered to the right.

VIII. CONCLUSIONS

Different clustering architectures and how they apply to different system configurations were investigated. A head-node architecture was implemented. Experiments on that architecture determined that significant speed-ups can be achieved by using a simple head-node architecture. The experiments determined that the number of frames per chunk affects the processing time by up to 10% and on a multi-core machine the number of nodes used should be one less than the number of cores. Speed-ups on the multi-core machine were achieved with no modifications to the application code. In a real-world application, the clustered Alchemist IP was slightly more than three times faster than the unclustered version on an eight-core machine.

The author's main recommendation for implementing algorithms which will scale across the available cores and machines is to make the processing occur in a single thread, and let the system architecture deal with the variability in the number of cores and machines.

ACKNOWLEDGMENTS

The author would like to thank his industrial supervisors Bruce Devlin, Dr. Joe Diggins and Dr. Kaaren May, his academic supervisors Dr. Iain Lindsay and Prof. Tughrul Arslan and his colleagues Violet Snell, Mike Knee and Dawn Frost for all their input into this project. The research was funded by Snell & Wilcox Ltd. and the EPSRC, under the Engineering Doctorate programme at the Institute for System Level Integration.

REFERENCES

- [1] R. Biesemeyer, "Media-friendly microprocessor architectures and tools," *SMPTE Journal*, vol. 111, no. 6, June 2002.

- [2] Digital Rapids, "Stream transcode manager - a deeper look," January 2007, a white paper from Digital Rapids covering job management software.
- [3] M. Knee, "Standards conversion for international HDTV content exchange," *SMPTE Journal*, vol. 116, no. 4, April 2007. [Online]. Available: <http://www.smpte.org/journal/?p=96>
- [4] G. Kola, T. Kosar, and M. Livny, "A fully automated fault-tolerant system for distributed video processing and off-site replication," in *NOSSDAV '04: Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM, 2004, pp. 122–126.
- [5] K. Shen, G. W. Cook, L. H. Jamieson, and E. J. Delp, "An overview of parallel processing approaches to image and video compression," in *Image and Video Compression, Proc. SPIE 2186*, M. Rabbani, Ed., 1994, pp. 197–208. [Online]. Available: citeseer.ist.psu.edu/shen94overview.html
- [6] J. M. Squyres, A. Lumsdaine, and R. L. Stevenson, "A cluster-based parallel image processing toolkit," in *Proceedings of the SPIE — The International Society for Optical Engineering*, vol. 2421, 1995, pp. 228–239. [Online]. Available: citeseer.ist.psu.edu/article/squyres95clusterbased.html
- [7] U. V. Vinod and P. K. Baruah, "Mpiimgen - a code transformer that parallelizes image processing codes to run on a cluster of workstations," in *CLUSTER '04: Proceedings of the 2004 IEEE International Conference on Cluster Computing*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 5–12.
- [8] Y. He, Y. He, I. Ahmad, and M. Liou, "Real-time distributed and parallel processing for mpeg-4," in *Proc. IEEE International Symposium on Circuits and Systems ISCAS '98*, I. Ahmad, Ed., vol. 3, 1998, pp. 603–606 vol.3.
- [9] K. Shen, L. A. Rowe, and E. J. Delp, "A parallel implementation of an mpeg1 encoder: Faster than real-time!" *Proceedings of SPIE Conference on Digital Video Compression: Algorithms and Technologies*, vol. 19, no. 2, pp. 281–296, – 1995. [Online]. Available: citeseer.ist.psu.edu/shen95parallel.html

Automatic Dynamic Structural-level Pipelining in Reconfigurable Processors

Mark Muir¹, Nazish Aslam²,
Ioannis Nouisias¹, Adam Major¹,
Tughrul Arslan^{1,2}, Iain Lindsay¹

⁽¹⁾The University of Edinburgh
Mayfield Road, Edinburgh
United Kingdom, EH3 9JL
mark.muir@ed.ac.uk

⁽²⁾Institute for System Level Integration
Alba Centre, Livingston
United Kingdom, EH54 7EG

ABSTRACT

This paper describes a technique for automated dynamic structural-level pipelining of programs targeting dynamically reconfigurable processors with very short reconfiguration times. These architectures are particularly suited to streaming applications, whose primary market are low-cost, high-volume consumer products such as the image signal processor for digital cameras in modern mobile phones. These reconfigurable processors open up the ability for vendors to differentiate their products by providing their own algorithms. To minimise area (and thus cost), it is important that vendors have the ability to tailor the resources of the core to their needs. Therefore, the process of application development is that of hardware/software co-design. As part of a high-level software tool chain, we present an optimisation technique that can be added to the compiler to significantly improve the throughput of applications, by pipelining tight loops (kernels) which perform the majority of the work in streaming applications. This allows more complex algorithms to be deployed whilst still meeting the available timing budget. The timing constraint is determined automatically, in a manner which maximises throughput within a given resource budget.

1. INTRODUCTION

The choice of platform for many modern digital signal processing tasks in embedded systems is often limited to application-specific integrated circuits (ASICs), since off-the-shelf programmable architectures such as DSPs and microprocessors cannot meet the throughput requirements, whereas reconfigurable hardware such as field-programmable gate arrays (FPGAs) require too much area and power. However, for applications that demand an element of reprogrammability, streaming processors (such as those offered by Ambric [1] and SPI [2]) are becoming an increasingly attractive solution, which improve on throughput by providing multiple processing elements/cores with an interconnect structure suited to streaming. However, these processing elements—usually based on regular DSP designs—often equate to significant silicon area. Alternatively, coarse-grained dynamically reconfigurable architectures (DRAs) offer a high degree of parallelism, sufficient to achieve high throughput [3][4]. Thus fewer cores are required for a given application, leading to a much lower area overhead. These coarse-

grained architectures, if given the ability to control their own reconfiguration, can be reconfigured very rapidly (e.g. millions of times per second), in order to achieve control flow similar to a regular microprocessor. This paper focuses on maximising the performance of programs running on a single core. However, the techniques can be directly applied to programs running on additional cores in a complete streaming application.

Coarse-grained DRAs, such as instruction cell based processors [5][6], provide a high degree of instruction chaining inside the core, by allowing arbitrary connections to be made between the various functional units via a configurable routing network. This allows quite complex data paths to be rendered onto the fabric and executed in a single configuration. This makes these architectures particularly suitable to stream processing, as fewer fetches from program memory are required. Performance is optimised by attempting to match the size of each kernel (inner loops where most of the execution time is spent) to the available resources, allowing them to fit into a single configuration context. This allows the configuration to persist for many clock cycles, operating on new data on each cycle. This increases throughput, since no time is spent having to reconfigure the core between successive iterations. It also decreases power consumption, as the configuration only needs to be fetched from program memory (or cache) once—upon first entering the kernel—rather than on every iteration. However, the resulting data paths can often have a long critical path, leading to poor temporal utilisation of the functional units, since they have to wait until all functional units have completed before operating on the next batch of data, which limits the throughput.

Pipelining provides a way of starting to operate on a new batch of data before an old one has completed. Thus, this allows the functional units of multiple stages of the kernel to be active concurrently; each operating on a different batch of data. Others have devised loop pipelining techniques for reconfigurable architectures [7, 8, 9], where successive iterations of the loop are replicated in hardware, and offset from each other to deal with any data dependencies between the iterations. These are most suitable for large reconfigurable architectures with much longer reconfiguration times, where there are sufficient resources for the entire loop body to be replicated many times. This paper elaborates on and extends work in a previous paper where structural-level pipelining techniques were shown to be applicable via software to rapidly reconfigurable/programmable architectures supporting operation-chaining. The technique allows complete kernels that were mapped to a single configuration context, to have their critical path length decreased by the addition of pipeline stage registers. Pipeline fill-

ing and flushing are achieved through dynamic reconfiguration.

The contribution in this work is the ability to automate the tasks of identifying configuration contexts which could benefit from pipelining, and choice of critical path constraint. In particular, in order to reduce power in the target architectures, the master clock frequency is kept as low as possible. Configuration contexts are allowed to persist for multiple clock cycles, until their critical path has completed. Pipelining reduces the critical path, so as a result, the quantisation introduced by the master clock frequency affects pipelined contexts more. Therefore, it is important to minimise the wasted time between the critical path stabilising and the next master clock cycle. The automatic pipelining algorithm demonstrated here attempts to do this.

Section 2 reviews existing pipelining techniques, and relevant software optimisation techniques. Section 3.1 describes an algorithm to perform pipeline stage allocation, and section 3.2 shows how properties of dynamic reconfiguration can be used to fill and flush the resulting pipeline. Section 3.3 details how the task can be completely automated. Section 4 shows the result of applying this technique to a real-life kernel used in image processing.

2. PREVIOUS WORK

For architectures that support instruction chaining, scheduling involves mapping as many dependent and independent data paths into as few configuration contexts as possible [10]. Independent data paths run in parallel, so the time for which a configuration persists is determined by the maximum critical path length of these data paths. If sufficient functional unit resources are available, loops can be optimised by loop unrolling [11]—i.e. placing multiple iterations as independent data paths in the same configuration. This allows multiple iterations to begin and end at once. This does not change the original critical path length, yet can increase the throughput. The throughput is determined by the critical path length of a loop iteration and the number of iterations that can be performed at once. During each execution of the loop configuration context, data propagates through the operation chains until the final result is ready. This means that the functional units involved in that chain are only performing useful work for a fraction of the time. This is where structural-level pipelining of these data paths comes in—to artificially reduce the critical path length by allowing new iterations to begin without waiting for the completion of previous iterations.

Various approaches of pipelining data paths have been proposed [12]. These require that the designer specifies a throughput constraint, in order to allow the algorithm to best make the choice between throughput and the area overhead each pipeline stage introduces. These approaches describe various algorithms for the task of pipeline stage allocation, applied to a number of different levels in a design. On reconfigurable architectures such as FPGAs, custom pipelines can be rendered as part of the configuration, leading to significant increases in throughput [13]. The previous work on dynamically pipelining DRAs [14] proposed a technique where pipelining would be performed based on a critical path constraint provided by the application developer. The work here elaborates on this technique, and looks into more detail on the real-life performance. Extensions are proposed to automate the choice of critical path constraint, and to maximise the real-life throughput.

3. DYNAMIC PIPELINING

Conventional structural-level pipelining can be applied to single configuration context kernels with long critical data paths, in or-

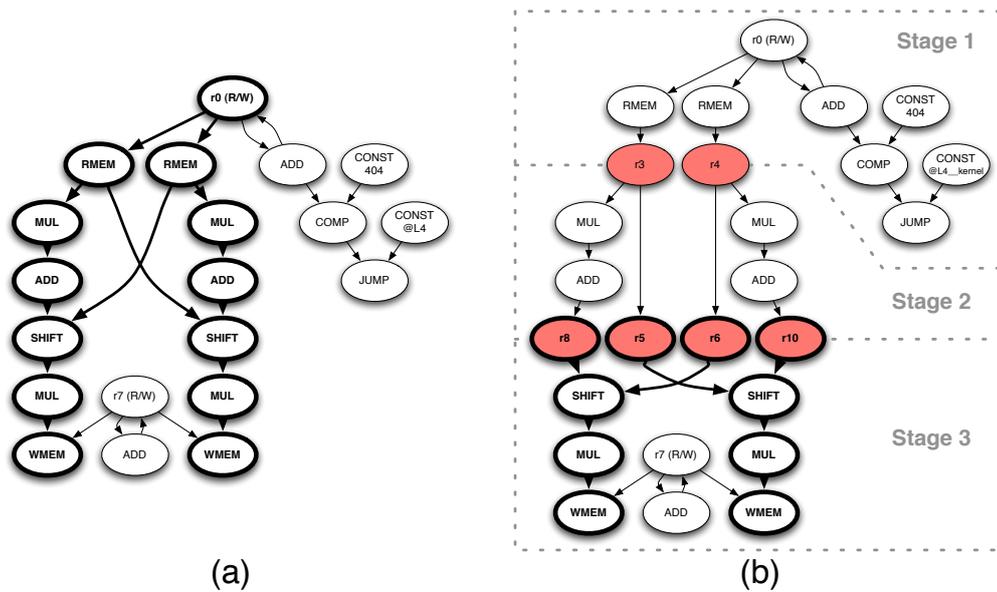
der to reduce the critical path, and thus increase throughput. This is done as part of the configuration—i.e. pipelines tailored to the particular kernel are rendered onto the core at runtime. This is done using existing register resources in the core to delay values for a single execution cycle, allowing values to be bridged across pipeline stage boundaries.

Structural pipelining is applied to the kernel basic block by first assigning each operation in the original data flow graph to a pipeline stage. Then, registers are introduced to store values over boundaries between pipeline stages. Only those values that are used in later pipeline stages are stored. A new register is needed for each value for each pipeline stage boundary over which it must persist. Figure 1 shows an example kernel before and after structural-level pipelining. The example includes only simple feedback chains consisting of a simple increment of the value of a register, however more complex feedback chains are also possible.

3.1 Pipeline stage allocation

First, constraints are defined between operations, where the order of execution is important. Examples include ‘same stage or earlier’ constraints between operations reading from input registers and operations that have those same registers marked as global output registers, and ‘same stage or earlier’ constraints between data memory read operations and potentially aliasing data memory write operations. All operations in a feedback chain must be placed in the same pipeline stage, since such chains require single-step total latency in order to keep the pipeline full. The algorithm for assigning pipeline stages to each operation is as follows:

- Identify the ‘jump’ operation, and all of its dependencies. Save this in a set—the ‘jump chain’ set.
- Create the ‘remaining’ set—a record of those operations yet to be assigned to a pipeline stage. This is initially populated with all the operations except for those in the ‘jump chain’ set.
- Define the constraints:
 - Add ‘same stage or earlier’ constraints between operations reading from input registers, and operations that have those same registers marked as global output registers.
 - Add ‘same stage or earlier’ constraints between data memory read operations and potentially aliasing data memory write operations.
 - Add ‘same stage or earlier’ constraints between volatile operations of the same kind, to ensure that they still appear in their original order.
- Detect feedback chains:
 - Identify all the operations that are part of each feedback chain, and record them in a set for each chain. These shall be referred to as the ‘feedback’ sets. No operation in a feedback set may be assigned to a pipeline stage until all the operations in that set are ready to be assigned.
- Create an ordered list of pipeline stages, initially consisting of a single entry. Each entry contains the set of operations that have been assigned to that pipeline stage.
- For each operation in the ‘remaining’ set:
 - Create a temporary set containing this operation and any operations in the same ‘feedback’ set (if one exists).
 - Determine whether any of the operations in the temporary set have any successors that are also in the ‘remaining’ set. If they do, then the temporary set is not ready, so discard it and move on to the next operation in the ‘remaining’ set.



1: Example kernel data flow graph, (a) before pipelining, (b) after pipelining (kernel loop context). The inserted pipeline stage registers are shown in red. The per-cycle critical path is shown in bold, and is shorter in (b), which allows for a higher throughput.

- Determine whether any constraints involving the operations in the temporary set involve operations that are also in the ‘remaining’ set. If they do, then the temporary set is not ready, so discard it and move on to the next operation in the ‘remaining’ set.
- Identify the latest pipeline stage where all the operations in the temporary set could be placed, according to their dependencies and constraints.
- Construct a configuration context containing all the pipeline stages constructed thus far, and calculate its critical path delay (including the reading from and writing to pipeline registers).
- Speculatively construct a configuration context containing all the pipeline stages constructed thus far, including the operations from the temporary set, placed in the previously identified pipeline stage. Calculate its critical path delay.
- If the critical path delay is different (i.e. increased), and the new delay exceeds the target, then move to the preceding pipeline stage (creating a new pipeline stage at the beginning of the list, if the chosen stage was the first in the list).
- Transfer the operations from the temporary set to the identified pipeline stage, and remove them from the ‘remaining’ set.
- Loop whilst the ‘remaining’ set is not empty.
- Add the operations from the ‘jump chain’ set to the first pipeline stage.

The algorithm is a form of list scheduling. Only operations whose predecessors (in the data path) have already been assigned a pipeline stage may be considered for insertion on each pass. In order to minimise the register count, operations should be placed in as late a pipeline stage as possible. Operations that must be placed in the same stage are dealt with together. Operations are considered for placement in the latest pipeline stage containing any of their predecessors. Then, the insertion point is moved towards later pipeline stages until all constraints have been satisfied. Once a valid insertion point has been identified, the critical path is calculated for the resulting (incomplete) configuration context with the operation in

that pipeline stage. If the critical path meets the target value, the operation is placed in that pipeline stage. Otherwise, the operation is added to the next pipeline stage (creating it if it does not exist).

The creation of dependencies ensures that the sequence of state changes is maintained, thus ensuring correct results. Assigning operations to a late a pipeline stage as possible aids to reduce the number of registers required. Once the pipeline stages have been determined, pipeline stage registers are assigned as follows:

- For each pipeline stage in sequence:
 - Assign a new register storing the value produced by each operation in all previous pipeline stages that needs to be stored for use in this or any later stage.

3.2 Dynamic initialisation and clean-up

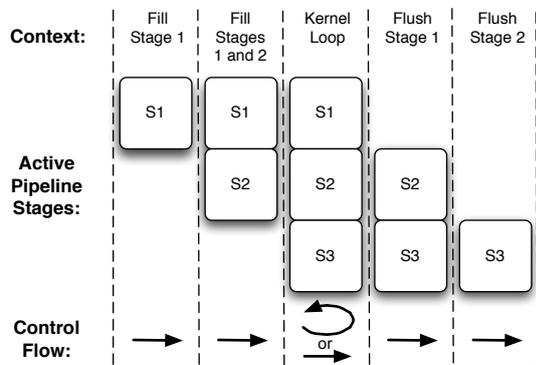
Normally, a pipelined design would require additional logic to take care of initialising the pipeline stages, or to suppress the operations in later pipeline stages until the previous stages have filled (predication), so that they do not operate on garbage. However, the pipelines in a coarse-grained DRA are themselves rendered as part of the configuration context. Provided that the configuration time is not significantly larger than the execution time of each step, dynamic reconfiguration can be used to render different configurations before the main kernel loop configuration, to fill successive stages of the pipeline, and similarly to flush the pipeline after exiting the kernel loop. This allows the kernel loop configuration to assume that the pipeline stages are always full. This provides a generic, purely software alternative to predication, which can be used as a fall-back when no hardware support exists.

Prologue: New configuration contexts are created to initially fill each successive stage of the pipeline. For n pipeline stages, $n - 1$ pipeline filling contexts are created.

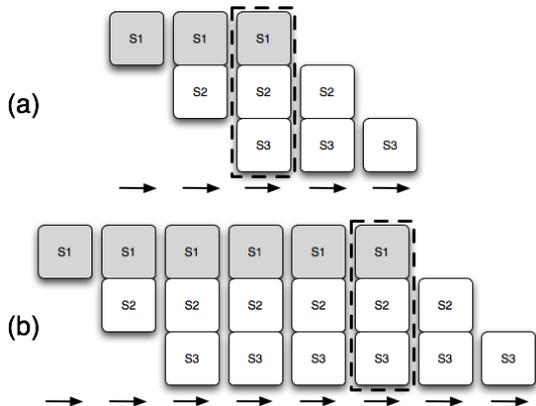
Loop: A single configuration context is created for the kernel loop, which includes all pipeline stages.

Epilogue: New configuration contexts are created to flush successive stages of the pipeline. For n pipeline stages, $n - 1$ pipeline flushing contexts are created.

The core is dynamically reconfigured to first perform pipeline initialisation, then reconfigured to execute the kernel loop, then finally reconfigured to flush the pipeline—as demonstrated in figure 2. This is similar to the epilogue and prologue in software pipelining [15].



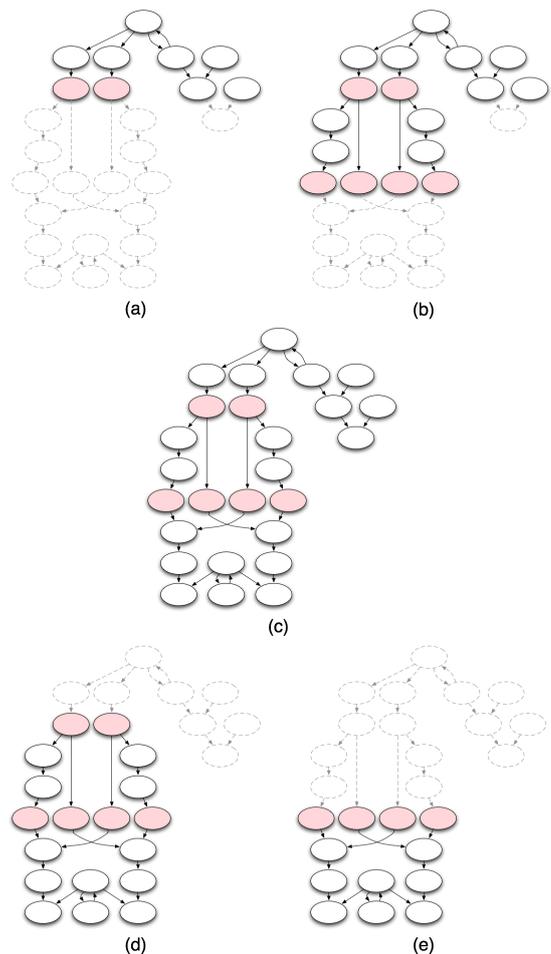
2: Control flow for a 3-stage pipelined kernel, showing which stages are active in each context (and moment in time). Execution flows from one context to the next, except in the kernel loop, which loops back to itself (holding the same context) until the end condition is satisfied.



3: Expanded control flow for the pipeline shown in figure 2 for (a) 3, and (b) 6 iterations. The point at which the loop termination condition should evaluate to true is shown by a dotted box. It can be seen in both cases that only the first stage has executed for the desired number of iterations by this point.

The configuration contexts generated for the kernel example from figure 1 is shown in figure 4. The use of separate special-purpose configurations alleviates the need for special logic for this purpose in the kernel loop configuration context, keeping its size down, and thus not compromising the potential parallelism in the core.

Figure 2 shows which stages of the pipeline are active during execution for a 3-stage pipeline. As the target architectures may not be state free (e.g. memory access), it is important to not allow any operation in any pipeline stage to operate on garbage, and to preserve the execution count. With the arrangement shown in the figure, all pipeline stages will be executed the same number of times irrespective of the number of iterations performed in the kernel loop.



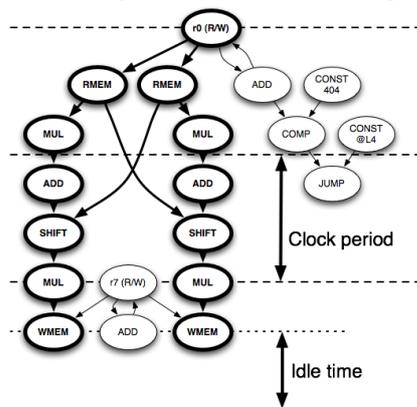
4: The sequence of configuration contexts created for the example kernel, (a) iteration 1—filling pipeline stage 1, (b) iteration 2—filling pipeline stages 1 and 2, (c) iterations 3 to $n - 2$ —pipeline full (loop), (d) iteration $n - 1$ —flushing pipeline stage 1, (e) iteration n —flushing pipeline stage 2.

Now consider the original kernel, where the ‘jump’ operation causes the loop to terminate after n iterations. In the pipelined kernel, we must ensure that the kernel loop terminates after n executions of the operations that calculate the loop termination condition; otherwise, the operations or operands would need to be modified to yield a different iteration count. Looking at figure 2, the minimum number of iterations possible in the pipelined design occurs when the kernel loop context executes only once. This corresponds to an iteration count equal to the number of pipeline stages (in this case 3). In order for the loop to terminate immediately, the operations that determine the loop termination condition must have been executed this number of times by the time the kernel loop context has been executed. This can only be achieved by placing these operations in the *first* pipeline stage. The same argument also applies for any higher iteration count. Figure 3 shows two examples, to highlight this point.

Placing the ‘jump’ in the first pipeline stage therefore requires that all of its dependencies are also placed in the first pipeline stage. Since the pipeline filling contexts (prologue) should always be executed in sequence (with no branching), the ‘jump’ operation is omitted from these contexts, even though it is in a pipeline stage

active in those contexts. Its dependencies are left in place, since their side effects are important—e.g. they could update the iteration counter whose value is used to determine the loop termination condition.

3.3 Automating the choice of timing constraint



5: Idle time resulting from the master clock. The shorter the critical path of the kernel, the more effect this has. This particularly affects pipelined kernels.

The arbitrary operation chaining supported by the target architectures leads to a great variation in critical path length in different configuration contexts, as paths can be constructed involving long chains of a varying number of cells, and each type of cell has a different combinatorial delay. Ideally, each iteration of the configuration context should be allowed to persist for the time required for the results to stabilise on the operation(s) that lie at the end of the critical path. In order to avoid the overhead of asynchronous logic, a master clock is normally used instead, and the iteration ends on the next master clock cycle after the last results have stabilised, as can be seen in figure 5. To minimise the resulting idle time between these two events, it is desirable to minimise the period of the master clock. However, high clock frequencies come at the cost of power consumption and area. Therefore, a suitable compromise has to be made.

Since pipelining reduces the critical path length of each iteration of the kernel loop configuration context, the quantisation introduced by the master clock frequency affects pipelined contexts more. Therefore, it is important to minimise the wasted time between the critical path stabilising and the next master clock cycle. This fact is used to aid the automatic choice of the timing constraint.

The timing constraint is initially chosen to be the minimum possible critical path length that a pipeline stage can consist of. This is determined by the length of certain data paths that cannot be split across pipeline stages. These include the jump condition logic determining when to finish the loop, and feedback loops that update a register or memory location (where that register or memory location is both read from and written to in the same kernel). The one with the longest critical path length is selected, and the value rounded up to the next integer multiple of the master clock period.

Then, pipeline stage allocation is performed using this critical path constraint. If a valid pipeline could be constructed, register allocation is performed. If there are sufficient registers available, then this pipeline geometry is used, since it will result in the highest possible iteration rate. Otherwise, the timing constraint is incremented

by one master clock period, and the process continues. A natural end point exists where this value reaches the critical path of the non-pipelined kernel. If reached, the context is left non-pipelined.

For completely automatic pipelining, feedback-directed optimisation is used. The program is first executed in a simulator prior to pipelining, and profiling information is fed back into the compiler. Basic blocks that loop to themselves are identified, and where sufficient resources exist in the core to map the entire block into one configuration context, these are potential candidates for pipelining. The number of consecutive iterations of each candidate is determined through the profiling results. The minimum consecutive iterations for a kernel defines the maximum depth to which it can be pipelined: the pipeline depth must not be less than the minimum execution count. This is used as a test during each iteration of the timing constraint selection algorithm, where a potential pipeline is checked for its depth not exceeding the minimum iteration count. If it does, then the geometry is considered invalid, and the algorithm continues with a larger timing constraint. To take into account the cost of loading the new configurations from memory, the minimum iteration count value is artificially reduced by an arbitrary count, to weigh the algorithm in favour of only pipelining loops with significant iteration counts.

4. APPLICATION TO STREAMING

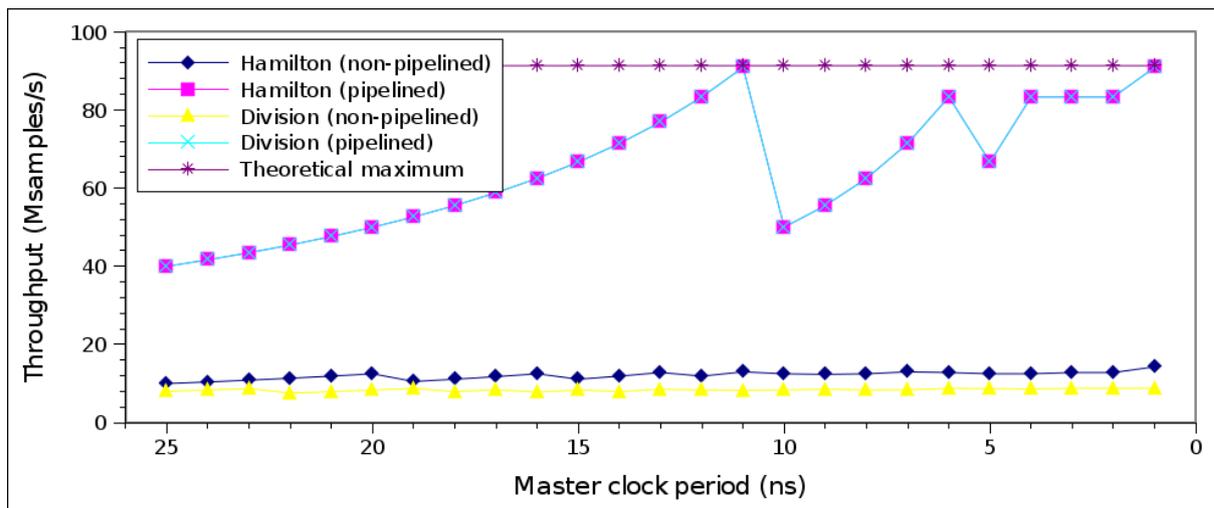
The algorithm described in this paper was applied to two real-life applications: a 7-line Hamilton demosaic filter [16], and a multiplication-based iterative software division algorithm. The demosaic involves interpolating missing colour components from the Bayer output of a colour filter array sensor. Division on a per-pixel level is used as part of many commercial noise reduction filters. Both are high-throughput tasks normally done on-chip as part of a custom image signal processing (ISP) pipeline, used in modern digital cameras and mobile phones. Both kernels were implemented on a reconfigurable instruction cell-based processor [5] (180nm timing figures), using the C language. Software optimisation techniques were used to reduce the main kernel in each case into a basic block small enough to fit onto the target architecture in a single configuration context. Both example kernels produce a single output pixel per iteration.

The performance of the pipelining for both cases is shown in figure 6, and some additional details are given for the Hamilton demosaic in table 1.

The main trend to notice is the ability for the maximum achievable iteration rate (after pipelining) to generally increase as the master clock frequency is increased. Since the same underlying data path is used in each case, the non-pipelined critical path length is constant. The iteration time of the non-pipelined data paths is just the critical path length rounded up to the next integer multiple of the master clock period. As the master clock period is decreased, the algorithm is able to produce a pipeline with a critical path closer to the theoretical minimum (as dictated by the indivisible data paths such as feedback loops, and the jump condition chain). However, the number of pipeline stages required to do this increases in a faster than linear fashion. This is due to quantisation: the error between the time taken for each data path fragment in each pipeline stage to complete and the closest integer multiple of the master clock frequency. As the pipeline stages get shorter, the relative size of the indivisible units being pipelined (i.e. the internal delays of each cell and section of interconnect) increases compared to the resolution of the master clock. The algorithm does well in minimis-

Master clock period (ns)	20.0	15.0	10.0	5.0	3.0	2.0	1.0
Pipeline stages	5	7	5	7	9	9	11
Pipeline stage registers	80	123	80	123	153	153	189
Min. possible constraint (ns)	10.95	10.95	10.95	10.95	10.95	10.95	10.95
Non-pipelined critical path (ns)	77.0	77.0	77.0	77.0	77.0	77.0	77.0
Pipelined critical path (ns)	19.8	14.65	19.8	14.65	11.55	11.55	11.00
Improvement in critical path	389%	526%	389%	526%	667%	667%	700%
Non-pipelined iteration time (ns)	80.0	90.0	80.0	80.0	78.0	78.0	77.0
Pipelined iteration time (ns)	20.0	15.0	20.0	15.0	12.0	12.0	11.0
Improvement in iteration time	400%	600%	400%	533%	650%	650%	636%
Pipelined throughput (MPixels/s)	50.0	66.6	50.0	66.6	83.3	83.3	90.9

1: Performance of the demosaic filter kernel before and after automatic pipelining, over a range of master clock periods. See section 4 for an explanation of the results.



6: Throughput before and after automatic pipelining, over a range of master clock periods, for two pixel-level code examples: Hamilton demosaic and iterative software division. The theoretical line shows what could be achieved if the master clock were of infinite frequency, based on the longest indivisible critical path (the iteration control logic in both of these cases).

ing this effect, and the percentage improvements with and without the effect of the master clock are relatively close in all cases.

The pipeline geometries constructed for each master clock frequency setting are shown in figure 7. Both examples show identical post-pipelining throughput (iteration rate), as both cases have the same longest indivisible critical path—corresponding to the iteration control (jump) logic (shown by the theoretical line in figure 6). There are no data dependencies or other constraints limiting the potential for pipelining in either example. If data dependencies, feedback loops, or other constraints were present, these would be reflected by a larger indivisible critical path. The shorter the indivisible critical path, the more important the behaviour of the automatic pipelining algorithm.

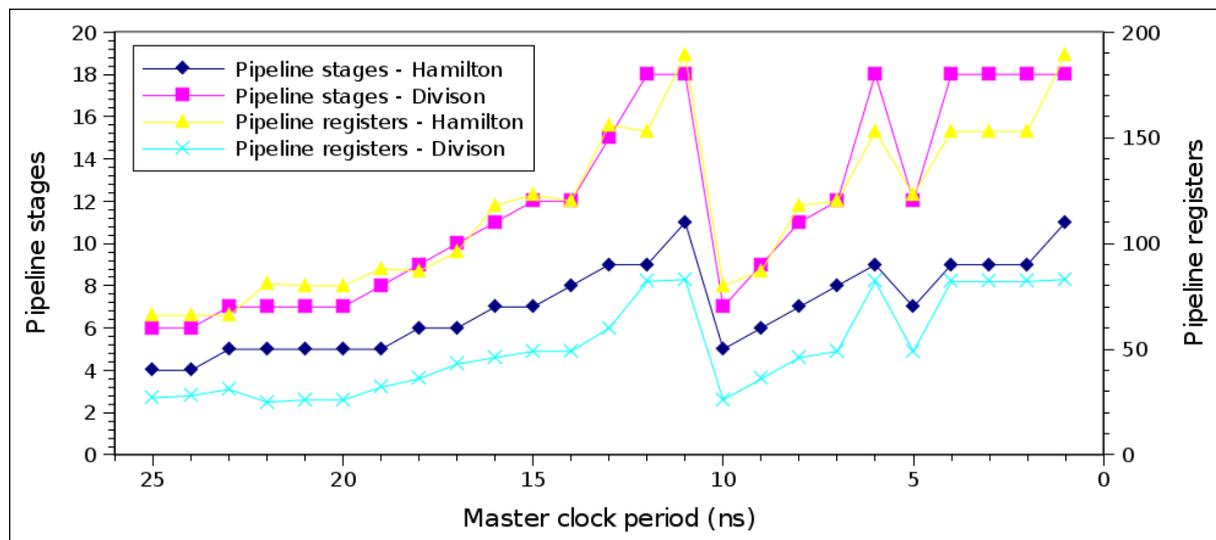
The resource-saving effect of the algorithm can be seen to come into effect each time the current integer multiple of the master clock frequency drops below the indivisible critical path length. This makes the iteration rate curve appear to wrap around each time it tries to cross the theoretical maximum iteration rate line. By extending the length of the pipeline stages up to the next master clock period, the number of registers is minimised, which avoids needless congestion on the interconnect. The reduction in the number

of pipeline stages reduces the configuration size and the latency, since fewer filling and flushing iterations need to be performed.

5. CONCLUSIONS

This work proposed an algorithm for automatically applying dynamic structural-level pipelining to single configuration context kernels running on dynamically reconfigurable arrays (DRAs). The technique is a form of feedback directed optimisation, where profiling information (consecutive execution counts) are used to determine which kernels will benefit from pipelining. Candidates with very low consecutive execution counts must not be pipelined too deeply. This is to ensure that the additional latency of pipeline filling and flushing is more than nullified by the decrease in total execution time for the pipelined kernel loop when the pipeline is full. This is only possible when the minimum possible iteration count is known. This is the case for pixel-level kernels in the ISP application domain, as the iteration count is typically the line size of the image.

An iterative approach is used to form an efficient pipeline, where the timing constraint is automatically chosen to be an integer multiple of the master clock frequency. The timing constraint is incre-



7: Pipeline geometry from automatic pipelining, over a range of master clock periods, for two pixel-level code examples: Hamilton demosaic and iterative software division.

mented until a valid pipeline can be constructed without encountering register starvation. The range of possible pipeline geometries is controlled by the availability of registers. Architectures with distributed registers will offer the best results, otherwise the bandwidth of the interface and/or additional combinatorial delays introduced by routing to and from a register file would likely outweigh any benefit. This makes the case for registers to be made available in the interconnect itself.

The algorithm was applied to a demosaic kernel of modest complexity and to a software division algorithm, leading to the possibility to pipeline to a significant depth. A performance increase of up to 7 times can be obtained for the demosaic example, and nearly 10 times for the division. As the pipeline gets deeper, the cost—in terms of register requirement and storage for pipeline filling and flushing contexts—increases more than linearly. As the critical path of the pipelined kernel gets smaller, the quantisation of the iteration rate caused by the master clock, gets increasingly worse. Inside the bounds of this quantisation, reducing the pipeline critical path (by increasing the number of pipeline stages) has no effect on the iteration rate. In these situations, extra resources would be introduced for no benefit. To avoid this, the proposed algorithm relaxes the critical path to take into account this quantisation, thus minimising the resource requirements for a given physically achievable iteration rate.

6. REFERENCES

- [1] M. Butts, A. M. Jones, and P. Wasson, "A structural object programming model, architecture, chip and tools for reconfigurable computing," in *FCCM*, 2007, pp. 55–64.
- [2] B. Khailany, T. Williams, J. Lin, E. Long, M. Rygh, D. Tovey, and W. Daly, "A programmable 512 GOPS stream processor for signal, image, and video processing," in *Solid-State Circuits Conference*, 2007, pp. 272–602.
- [3] A. Major, T. Arslan, et al., "H.264 decoder implementation on a dynamically reconfigurable instruction cell based architecture," in *International SOC Conference*, 2006, pp. 49–52.
- [4] Z. Khan, T. Arslan, et al., "Implementation of a real time programmable encoder for low density parity check code on a reconfigurable instruction cell architecture," in *Design Automation Conference, Asia and South Pacific*, 2007, pp. 583–588.
- [5] S. Khawam, I. Nousias, M. Milward, Y. Yi, M. Muir, and T. Arslan, "The reconfigurable instruction cell array," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 1, pp. 1–11, 2008.
- [6] "Loosely-biased heterogeneous reconfigurable arrays," U.S. Patent 20 050 257 024, 2005.
- [7] M. Weinhardt and W. Luk, "Pipeline vectorization," *IEEE Trans. Comp.-Aid. Des. Integ. Circ. and Syst.*, vol. 20, no. 2, pp. 234–248, 2001.
- [8] J. Liao, W. Wong, and T. Mitra, "A model for hardware realization of kernel loops," in *Field Programmable Logic, International Conference on*, 2003, pp. 334–344.
- [9] R. Rodrigues and J. Cardoso, "Pipelining sequences of loops—a first example," in *ARC, Workshop*, 2005, pp. 147–151.
- [10] Y. Yi and I. Nousias, "System-level scheduling on instruction cell based reconfigurable systems," in *Design Automation and Test in Europe, International Conference on*, 2006, pp. 381–386.
- [11] J. Sanchez and A. Gonzalez, "The effectiveness of loop unrolling for modulo scheduling in clustered VLIW architectures," in *ICCP Parallel Processing, International Conference on*, 2000, p. 555.
- [12] S. Bakshi and D. Gajski, "Partitioning and pipelining for performance-constrained hardware/software systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 7, no. 4, pp. 419–432, 1999.
- [13] S. Silva and S. Bampi, "Area and throughput trade-offs in the design of pipelined discrete wavelet transform architectures," in *Design Automation and Test in Europe, International Conference on*, 2005, pp. 32–37.
- [14] M. Muir, T. Arslan, and I. Lindsay, "Automated dynamic throughput-constrained structural-level pipelining in streaming applications," in *Design Automation and Test in Europe, international conference on*, 2008, pp. 1358–1361.
- [15] M. Lam, "Software pipelining: an effective scheduling technique for VLIW machines," in *ACM SIGPLAN conference on Programming Language design and Implementation*. New York, NY, USA: ACM Press, 1988, pp. 318–328.
- [16] R. Ramanath, W. Snyder, and G. Bilbro, "Demosaicking methods for bayer color arrays," *Electronic Imaging*, vol. 11, no. 3, pp. 306–315, 2002.

Design and Implementation of an Efficient Shape Analysis Engine on FPGAs

Dr. Khaled Benkrid*

The University of Edinburgh, School of Engineering and Electronics, King's Buildings,
Mayfield Road, Edinburgh EH9 3JL, Scotland, UK

*k.benkrid@ieee.org

Abstract

This paper presents the design and implementation of an efficient shape analysis engine on FPGAs. The engine implements the necessary front-end image processing operations for shape analysis, namely: image segmentation, object labelling and counting, and outputs the results in the form of objects with different labels or grey scale levels attached to each object. These can then be further processed to generate various shape-specific measures such as area, perimeter, skeletons and moments. This is illustrated in this paper in the calculation of object compactness, defined as the ratio of an object area to the square of its perimeter length. The paper gives the details of the algorithms used as well as their efficient hardware implementation, and shows that the resulting implementations on Xilinx Virtex-II FPGAs can easily process video images in real time.

1. Introduction

2D shape analysis provides an important mean for object recognition and location in digital images [1]. Indeed, 2D shapes represent a unique characteristic of many kinds of objects ranging from keys to spanners and fingerprints to chromosomes. The aim of any shape analysis algorithm is thus to identify a number of measures, unique to a particular shape. These include area, perimeter, skeletons and moments [2]. However, before measuring such features, it is important to segment the input image (into foreground and background) and label all objects in the resulting binary image in order to process them separately. Figure 1 presents a block diagram of these pre-requisite operations in a typical shape analysis system.

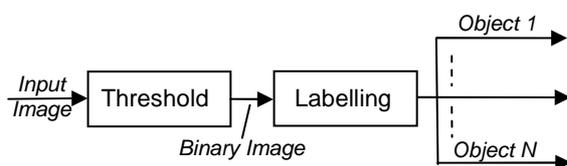


Figure 1. Block diagram of a shape analysis system front-end

As shown in Figure 1, the first operation on the input image is usually a threshold operation which separates the input image foreground from its background. The resulting image is usually binary i.e. it consists of only

'1' or '0' pixels, with 0's representing pixels belonging to the background, and 1's representing pixels belonging to image objects. After this, and in order to process different objects in the image separately, object labelling is performed [2]. The aim of this task is to allocate a unique pixel value (or label) to each connected component or object in the image. As a result, different objects can be processed separately according to their unique pixel value or label. This paper illustrates this back-end processing by measuring object compactness on the fly for all image objects. This is obtained by measuring each object area and perimeter and calculating object compactness, which is given by the following formula:

$$Compactness = Area / (Perimeter)^2$$

The resulting shape analysis system is thus illustrated in Figure 2. The paper presents an efficient hardware implementation of this basic shape analysis system using Field Programmable Gates Arrays (FPGAs). The following sections will detail the design of the main operations involved in Figure 2, namely image labelling, perimeter counting and compactness measurement, respectively. Implementation results on a Xilinx Virtex-II FPGA are then given and discussed. Finally, future work and conclusions are drawn.

2. Connected Component Labelling

Connected Component Labelling (CCL) is an important task in intermediate image processing with a large number of applications [3][4]. The problem is to assign a unique label to each connected component in an image while ensuring a different label for each distinct object as illustrated in Figure 3. Here, the input image is binary where 0 pixels refer to the background and non-zero pixels (1's) represent object pixels. Assuming square-shaped pixels, two pixels are considered to be connected, and thus part of the same object, if they lie next to each other horizontally, vertically or diagonally. Each pixel thus has eight possible neighbours. Figure 3.b presents the result of a CCL operation applied on the binary image of Figure 3.a. This assigns different labels (1, 2 and 3 in Figure 3.b) to each connected region or object. By assigning a unique label to each connected region, higher level image processing operations can identify, extract, and process each object separately.

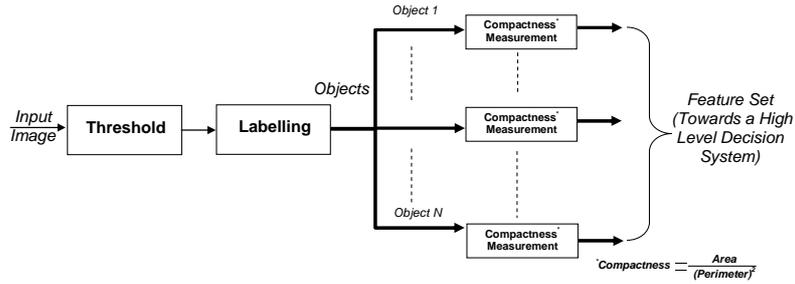


Figure 2. Block diagram of a basic shape analysis system

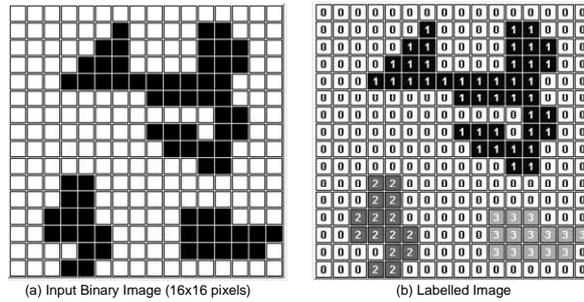


Figure 3. Connected Component Labelling: An example

Early hardware implementations of CCL were based on massively parallel approaches as they allocated one PE for each image pixel [5][6][7], which required a great deal of logic. As a result, such implementations were limited in the size of the image which can be labelled at once (e.g. 32x32). More recent hardware implementations are based on a neighbourhood operation where the result pixel of each neighbourhood calculation is fed back to the input image in time for the next neighbourhood calculation [8][9]. The algorithm used in this paper has been proposed in [10]. It is based on a 5-pixel neighbourhood operation and needs two scans of the image data in order to generate a completely labelled image. The following presents the details of this algorithm.

2.1 The CCL Algorithm

The CCL algorithm used in this paper consists of three pipelined steps:

- **First Pass (Initial labelling):** This step scans the original image applying a neighbourhood operation with the aim to assign the same label to connected pixels. However, this first step does not always succeed in assigning a single label to all object pixels due to possible object concavities, as will be illustrated below. As a result, and in order to link different labels which belong to the same object in a subsequent scan, a *labels connection table* is built in this step. The following step analyses this table.
- **Analysis of the Labels Connection Table (LCT):** In this step, all label connections are analysed in view of generating a final label correspondence table, or Look-Up-Table. The latter is used in the

final step (second pass) to assign a single label to each object in the image.

- **Second Pass (Final Re-labelling):** This final step scans the intermediate image generated from the first pass, and uses the Look-Up-Table generated in the second step to assign a single label value to all pixels belonging to the same object in the image.

The following explains these three steps in detail.

2.1.1. First Pass (Initial Labelling)

This step applies a ‘non-zero minimum’ neighbourhood operation on the image, using the template given in Figure 4.

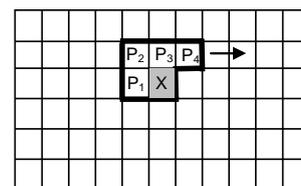


Figure 4. First Pass neighbourhood template

Scanning starts from the top left corner of the image and ends at the bottom right corner in a horizontal manner. Moreover, pixels outside the image boundaries are assumed to be background pixels i.e. 0’s. During each neighbourhood operation, each result pixel is stored back in the original image in time for the next neighbourhood calculation. At each template position, the source image pixel X and its neighbourhood pixels P₁, P₂, P₃, and P₄ are analysed in order to generate the new label of the current pixel X (= LX). This analysis is summarised by the following pseudo-code:

```

if (X==0) then
  // Here, X represents a background pixel, set result pixel to background i.e. 0
  LX=0;
else
  if (X!=0 and {Pi=0}i=1,2,3,4) then
  {
    // Here, X represents a foreground pixel with background neighbours. This is a potential
    // new object: assign the next available label that is not used (Next_Label) to this pixel
    LX= Next_Label;
    Next_Label = Next_Label +1;
    // Then, save this label value in the Labels Connection Table (LCT)
    LCT(LX)=LX;
  }
  else
  if (X!=0 and (P1!=0 or P2!=0 or P3!=0 or P4!=0)) then
  {
    // Here, X represents a foreground pixel with at least one other foreground neighbour. In this
    // case, the result pixel should be the minimum of all non-zero pixels in the neighbourhood
    LX = NON_ZERO_MIN (P1, P2, P3, P4)
    // If P1, P2, P3 or P4 have a different label from LX (the result of the non-zero minimum
    // operation) then these pixels should be re-labelled in a subsequent step of the algorithm
    // (step 3). For this to be possible, the connection between existing labels and the new label
    // should be stored in the LCT table. It can be proven, however, that there is only one single
    // different label in {P1, P2, P3, P4} equal to Max(P1, P2, P3, P4), thus:
    LCT(MAX(P1, P2, P3, P4)) = LX
  }
  }
  
```

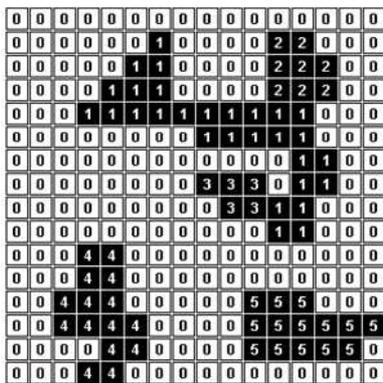
Figure 5 gives the result of the application of the above algorithm on the 16x16 binary image presented in Figure 3.a. Note that the first available label value (i.e. the initial value of Next_Label in the pseudo-code) is equal to 1. As can be seen, this pass does not assign a single label to all connected labels in an object. The following steps address this problem. But in order for this to be possible, this first step generates a Labels Connection Table (LCT) which captures the fact that some connected object pixels do not have the same pixel value or label, as a result of this first step. The latter captures this information by populating the LCT table with $LCT[j]=i$, where j is a label within the neighbourhood outlined in Figure 4, and i is the minimum value in the neighbourhood. If $j>i$ then any pixel labelled as j in the intermediate image generated by this step will be re-labelled as i in the next step. The content of the LCT table generated for the image sample given in Figure 3.a is shown in Figure 5.

As mentioned above, in addition to an intermediate, often partially labelled image, the first step generates a Labels Connection Table where if $j>i$ and $LCT[j]=i$, then label j should be re-labelled as i . However, if label j is also connected to another label k (i.e. if $k>j$ and $LCT[k]=j$) then k should not be replaced by j in the subsequent step, but rather by i . Such indirect connections can be of any length. Hence, it is necessary to update the content of the LCT before it can be used as a direct look-up table in the final step of the CCL algorithm. This consists in looping through the LCT table from $i=0$ to the maximum label used (Max_Label) and performing the following operation at each table position:

$$i = LCT[LCT[i]]$$

This will result in the elimination of some labels from the table. The remaining labels should thus be re-labelled in order to form a monotonically increasing sequence. The last label value gives the number of objects in the image.

The following pseudo-code summarises this second step:



LCT content:
LCT[1] = 1
LCT[2] = 1
LCT[3] = 1
LCT[4] = 4
LCT[5] = 5

Figure 5. Result image after the first pass of the CCL algorithm

2.1.2. Analysis of the Labels Connection Table

Figure 6 below illustrates the results of this step on the LCT table given in Figure 5.

```

// Initialise the next available label value to 1
NEXT_Label = 1;

for(i=0;i<=MAX_LABEL;i++)
{
  if (TAB(i) != i)
  {
    // Here labels TAB(i) and i are connected, re-label TAB(i)
    //pixels as TAB(TAB(i)) pixels
    TAB(i)=TAB(TAB(i));
  }
  else
  {
    // Here labels TAB(i) = i, replace TAB(i) by the lowest available
    // label value
    TAB(i)= NEXT_Label;
    NEXT_Label = NEXT_Label + 1;
  }
}
Number_Of_Objects = NEXT_Label;
  
```

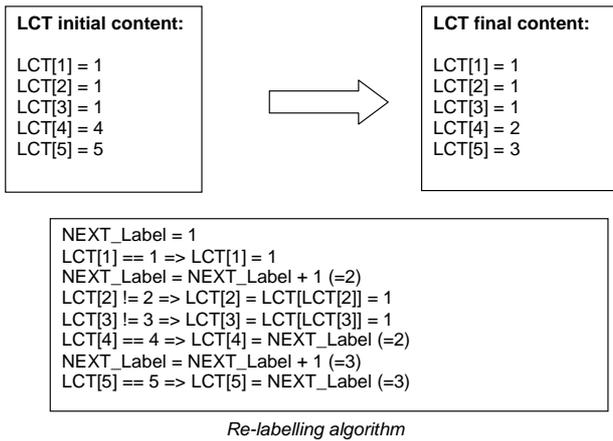


Figure 6. Illustration of the LCT re-labelling process

2.1.3. Second Pass (Final Re-labelling)

This final step consists in scanning the intermediate image generated from the first pass (referred to as **INTERMEDIATE** below) and using the LCT table as a Look-Up-Table to generate the fully labelled image i.e. for each pixel with coordinates (x,y) perform the following:

$$\text{Labelled_Image}(x,y) = \text{TAB}(\text{INTERMEDIATE}(x,y))$$

This is illustrated in Figure 7 in the case of the partially labelled image of Figure 5 above.

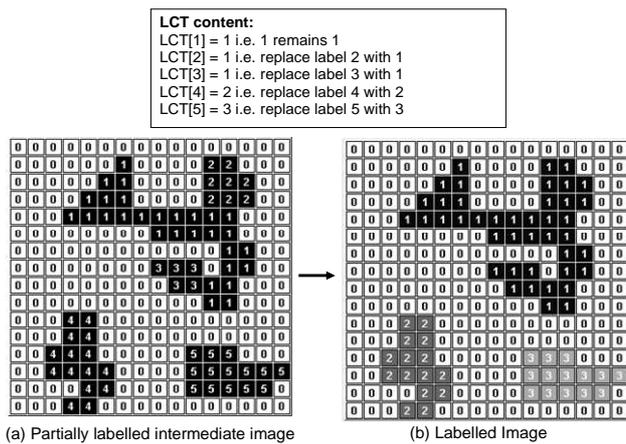


Figure 7. Connected Component Labelling: Final Phase

2.2. Hardware Implementation

Figure 8 presents a block diagram of a hardware architecture for the implementation of the above CCL algorithm. This hardware architecture was developed using a direct C-to-hardware language called Handel-C [11]. This ANSI C-like language is a subset of C, extended with CSP parallelism and communication primitives. One of the most fundamental constructs in Handel-C is the **par** statement which allows for separate parallel flows (the default being sequential flows) as illustrated in Figure 9.

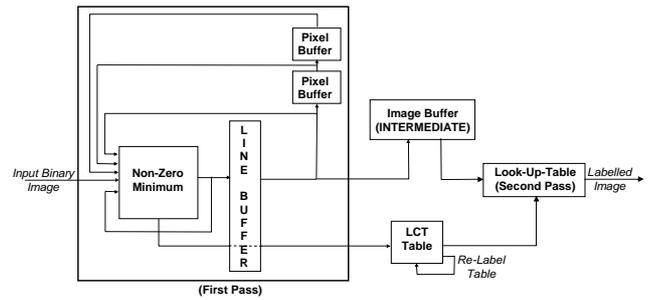


Figure 8. Hardware architecture for the CCL algorithm used

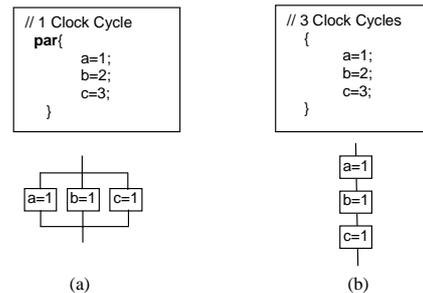


Figure 9. Parallel (a) vs. sequential (b) statements in Handel-C

Moreover, Handel-C has a simple synchronous timing model whereby each assignment takes precisely one clock cycle. This, alongside Handel-C's high level control flow statements (**if/else** statement, **while** statement, and **for** loop statement) and simple memory access, makes it relatively easier to use than other FPGA design languages and tools. This is particularly true for capturing control-intensive algorithms in Hardware. Furthermore, and in order to allow for concise and parameterised code, Handel-C provides code replicating constructs in the form of parameterised **par** and **seq** statements (for parallel or sequential code replication respectively). These statements' syntax is similar to that of a "for loop" as illustrated below for the case of replicated **par** statement.

```
// Replicated par
par (i=0; i<3; i++)
{
    a[i] = b[i];
}
expands to:
par
{
    a[0] = b[0];
    a[1] = b[1];
    a[2] = b[2];
}
```

Using such constructs allows for compact code listing as well as reuse of code through the use of *macro procedures*. Another advantage of the Handel-C language is that it is independent of the underlying hardware architecture as the same code can be used to target different FPGA architectures.

The hardware architecture presented in Figure 8 has been described in Handel-C in a modular manner using three main macro procedures corresponding to the three steps involved in the algorithm (non-zero minimum neighbourhood operation or first pass, LCT table re-

labelling, and the look-up-table or second pass phase), as illustrated in the code below:

```

par
{
// First Pass: Non-Minimum Neighbourhood Operation
Neighbourhood_operation(&Input_ImStream, &Intermediate_ImStream,
non_zero_min, [[-,0,0],[0,0,0]], &LCT);

// Re-label the Labels Connection Table
Re_label_table(&LCT);

// Second Pass: Look-Up-Table using LCT
Look-Up(&Intermediate_ImStream, &Labelled_ImStream, &LCT);
}
    
```

The first macro procedure **Neighbourhood_operation** takes the input and output stream handles (&Input_ImStream and &Intermediate_ImStream) as parameters, as well as the type of neighbourhood operation (e.g. non-zero minimum), template size and template coefficients. It returns a handle to the LCT table, which is used by the second macro (**Re_label_table**) in order to re-label the LCT table. Finally the **Look-Up** macro performs a second pass on the image (this time the intermediate image generated from the first pass) using the LCT table as a look-up table hence generating the fully labelled image (&Labelled_ImStream).

3. Perimeter Counting

Measurement of perimeters, areas, centroids and other shape related parameters of planar objects is an important task in industrial computerised vision systems [12] and is one measure used in object identification. The following will illustrate the problem of finding a reasonably accurate measure of the perimeter of an object. This will be addressed first in the case of a binary image with one single object. The case of multi-object images will then be deduced.

3.1. An algorithm for perimeter estimation

Consider a binary image 'Im' containing only one object. A contour of a digitised object is defined as a sequence of boundary pixels of the object as shown in Figure 10.c.

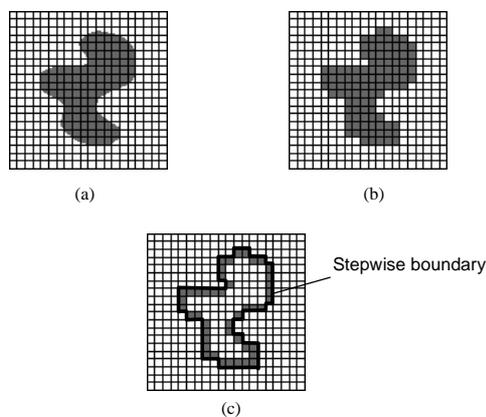


Figure 10. (a) Original object shape (b) Object shape after digitisation (discrete form) (c) object contour with its stepwise boundary

Another view of the object contour is the line between the object pixels and the background (the 'crack'). Encoding this line (a sequence of horizontal and vertical pixel edges) yields what is usually called the crack code of the digitised object boundary (identified as the bold line in Figure 10.c). Clearly the length of the latter contour is greater than the perimeter of the original object, especially for shapes with many corners: hence the problem of finding an efficient and accurate perimeter estimator.

One way of estimating the perimeter of a digitised object is to measure the number of vertical and horizontal cracks, and perform subsequent adjustments e.g. take the number of corners into consideration. Another approach is to approximate the real object boundary more accurately, and perform subsequent measurements on this approximated contour. In particular, it is common to represent the contour as a line passing through the centre of boundary pixels – i.e. as a sequence of horizontal, vertical and diagonal links [2]. Area measurements must of course take this into account, as this approach effectively shaves off a little of each boundary pixel of the object. Assuming square pixels, the perimeter is then estimated by:

$$Perimeter = No. of horizontal \& vertical links + (No. of diagonal links * \sqrt{2})$$

Hence, the number of horizontal, vertical and diagonal links in the contour needs to be counted. Before that, however, the perimeter itself needs to be found. A simple method for identifying the boundary pixels is to perform an 'Erode' operation on the image. This is a minimum neighbourhood operation with a template of size 3x3 as illustrated in Figure 11 below. The boundary pixels are those which were eroded, and can be found by subtracting the result from the original image, as shown in Figure 11.

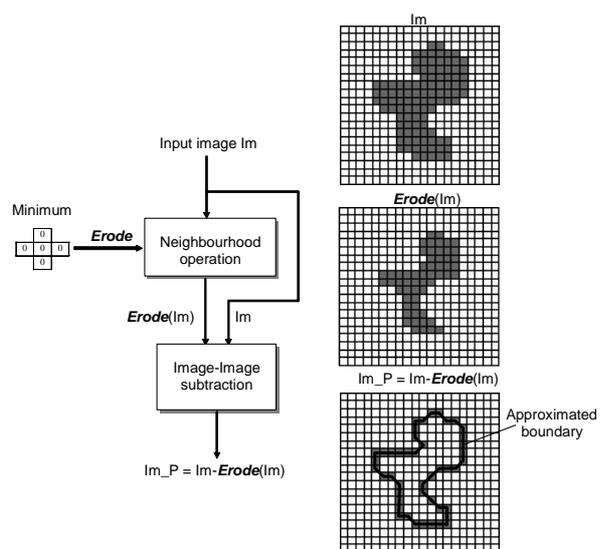
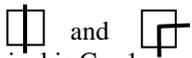
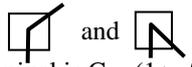
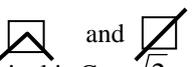


Figure 11. An edge finding algorithm for binary images

As mentioned above, a simple count of the number of pixel edges does not give an accurate measure of the perimeter (because of corners and diagonal edges). Instead, the contour is considered to be a sequence of links between the centres of adjacent boundary pixels (see *Im_P* in

Figure 11). However, rather than focus on the links (each of which straddles two boundary pixels), the individual contribution of each boundary pixel is considered. This contribution depends on the path which the contour follows through the pixel. Assuming a one-pixel wide perimeter, and an aspect ratio of 1.0 (i.e. square pixels), each edge pixel contribution to the perimeter can be classified into one of the following categories (or any of their rotations) [13]:

- (a)  in which case the contribution of the pixel is $C = 1$.
- (b)  in which case the contribution of the pixel is $C = (1 + \sqrt{2})/2$.
- (c)  in which case the contribution of the pixel is $C = \sqrt{2}$.

The perimeter is then given by:

$$\begin{aligned} \text{Perimeter} = & \text{No. of (a) pixels} * 1 \\ & + \text{No. of (b) pixels} * (1 + \sqrt{2})/2 \\ & + \text{No. of (c) pixels} * \sqrt{2} \end{aligned}$$

One way of classifying the contribution of edge pixels (assumed to be '1' against a background of 0's) is to convolve the whole binary image with the following window:

$$T = \text{Im_P} <\text{convolve}>$$

10	2	10
2	1	2
10	2	10

The result of this convolution at each pixel position enables the category of the corresponding edge pixel to be deduced:

Result (T[i,j])	category
5 or 15 or 7 or 25 or 27 or 17	(a)
13 or 23	(b)
21 or 33	(c)
anything else	no contribution

Table 1. Classification of different convolution pixel results

The make-up of these result pixels is shown in Figure 12 below.

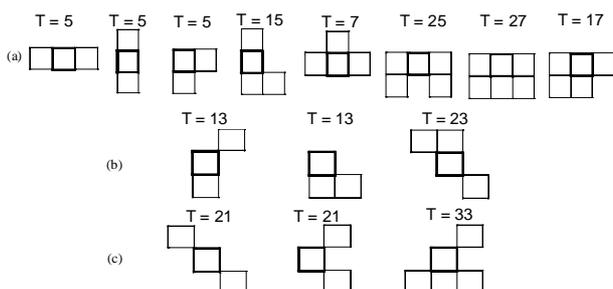


Figure 12. Different possibilities for edge segments

The perimeter will then be given by:

$$\begin{aligned} \text{Perimeter} = & \text{count}(T=5, 15, 7, 25, 27 \text{ or } 17) * 1 \\ & + \text{count}(T=13 \text{ or } 23) * (1 + \sqrt{2})/2 \\ & + \text{count}(T=21 \text{ or } 33) \text{ Pixels} * \sqrt{2} \end{aligned}$$

A block structure of the whole operation is given by Figure 13.

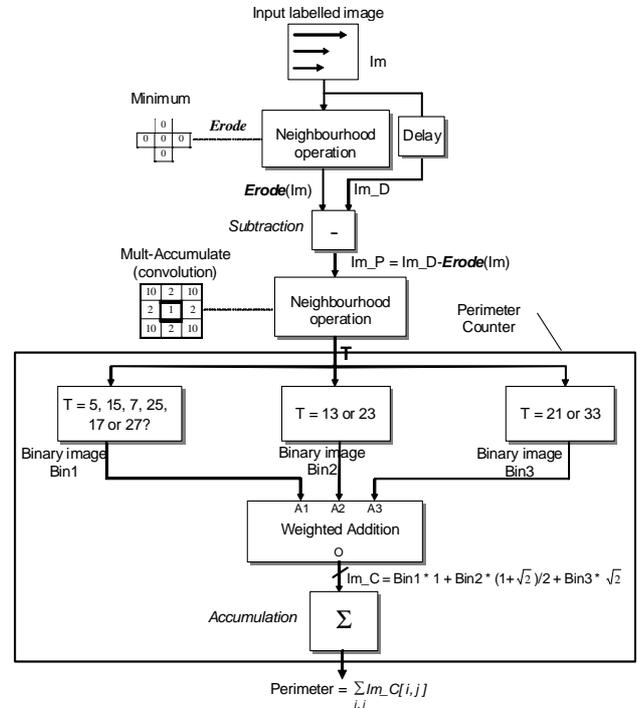


Figure 13. A block diagram of a perimeter estimator

3.2. The Case of a Multi-Object Image

In the shape analysis application (illustrated in Figure 2), the input to the feature measurement blocks (compactness measurement in this paper's case) is a labelled image. The latter consists of a discrete numbers of grey levels each corresponding to a distinct object in the image. The above perimeter algorithm could hence operate on every object in an image in parallel by making the perimeter counting process conditional on the pixels' grey level. That way, the perimeter of each object in the image could be counted in one single pass of the labelled image with no need for multiple passes, each for each object. The following subsection presents the corresponding hardware implementation.

3.3. Hardware Implementation

Figure 14 (at the end of this paper) gives a block diagram of the hardware architecture adopted for the above perimeter estimation algorithm. The input of this architecture is the output of the labelling process i.e. a multi-scalar image with each scale or grey level corresponding to one single object in the image. As in the case of the labelling architecture, this architecture has also been captured in Handel-C. The following gives the top level Handel-C code for the architecture.

```

par
{
// Erode Neighbourhood operation
Neighbourhood_operation(&Im, &Im2, add_min,
[[-,0,-],[0,0,0],[-,0,-]]);
// Subtract
Reduction_operation(&Im, &Im2, &Im_P, sub);
// Convolution operation
Neighbourhood_operation(&Im_P, &Im_P2, convolution,
[[10,2,10],[2,1,2],[10,2,10]]);
// Threshold
Point_Operation(&Im_P2, &Im_T, Array_of_Thresholds);
// Perimeter Count
// contribution = 1
ImageMultiScalarCount1(&Im_T, &Im_C1);
// contribution = (1 + sqrt(2)) / 2
ImageMultiScalarCount2(&Im_T, &Im_C2);
// contribution = sqrt(2)
ImageMultiScalarCount3(&Im_T, &Im_C3);
Affine_operation(&Im_C1, &Im_C2, &Im_C3, &Im3, Perimeter);
}

```

The final result i.e. the perimeter count for each object in the input image is stored in Array *Perimeter*.

4. Compactness Measurement

In order to measure the compactness of an object, both area and perimeter need to be counted. The previous section showed how to count object perimeters. Measuring an object's area is a simple matter of counting the number of pixels in the object. Object compactness can then be measured by evaluating the following equation:

$$Compactness = Area / (Perimeter)^2$$

Figure 15 (at the end of this paper) presents a block diagram of a hardware architecture for compactness measurement where the input image is fully labelled. Only an area count block and a block for the implementation of the compactness equation have been added compared to Figure 14's architecture.

The following gives the top level Handel-C code for this architecture. The final result, i.e. the compactness of each object in the input image, is stored in Array *Compactness*.

```

par
{
// Erode Neighbourhood operation
Neighbourhood_operation(&Im, &Im2, add_min,
[[-,0,-],[0,0,0],[-,0,-]]);
// Subtract
Reduction_operation(&Im, &Im2, &Im3, sub);
// Threshold
Point_Operation(&Im3, &Im_P);
// Convolution operation
Neighbourhood_operation(&Im_P, &Im_T, convolution,
[[10,2,10],[2,1,2],[10,2,10]]);
// Perimeter Count
// contribution = 1
ImageMultiScalarCount1(&Im_T, &Im_C1);
// contribution = (1 + sqrt(2)) / 2
ImageMultiScalarCount2(&Im_T, &Im_C2);
// contribution = sqrt(2)
ImageMultiScalarCount3(&Im_T, &Im_C3);
Affine_operation(&Im_C1, &Im_C2, &Im_C3, &Im3, Perimeter);
// Area measurement
Area(&Im3, Area);
// Compactness Measurement
Compactness(Perimeter, Area, Compactness);
}

```

5. Implementation Results

We have implemented the shape analysis architecture presented in Figure 15 on Xilinx Virtex-II FPGAs. For this, we used Celoxica's DK suite to compile our Handel-C descriptions into EDIF netlist. Xilinx ISE software was then used to generate an FPGA configuration/bitstream. The design flow is illustrated in Figure 16.

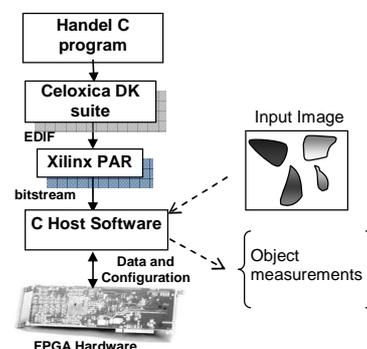


Figure 16. Handel-C design flow

Table 2 gives the hardware implementation results of the complete shape analysis engine described in Figure 2, as well as its major individual components, on the Xilinx XC2V6000-4 FPGA chip, for input images of 640x480 8-bit pixels. The resulting FPGA configuration has been tested on real FPGA hardware in the form of AlphaData ADM-XRC board. This shows clearly that the resulting implementation can easily achieve real time processing of video signals even at high image resolutions.

	Area	Speed	Throughput
Shape analysis engine	980 slices, 5 BlockRAMs, 1 Mult18x18	80 MHz	260 frame/sec
Labelling	540 slices, 4 BlockRAMs	80 MHz	260 frame/sec
Perimeter Counting	220 slices	80 MHz	260 frame/sec

Table 2. Hardware implementation results on a Xilinx XC2V6000-4 FPGA

6. Conclusion

This paper presented the detailed design and implementation of an efficient shape analysis engine on reconfigurable hardware. The engine can label objects present in video images and compute various morphological attributes related to each object separately, in real time. This has been illustrated in this paper in measuring the compactness of each object. Such information can serve as one of a number of features that can be used by a high level decision system e.g. for object recognition.

The core has been captured in a high level hardware language called Handel-C which makes it portable across various FPGA families. The resulting hardware implementation on a Xilinx XC2V6000-4 FPGA chip achieves a throughput rate of 260 frames per second for 640x480 8-bit/pixel images.

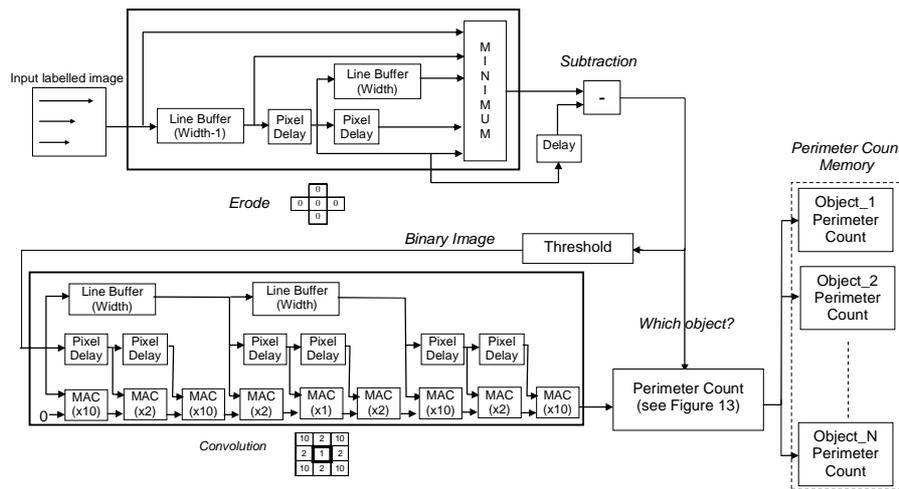


Figure 14. Hardware Architecture of the perimeter estimator

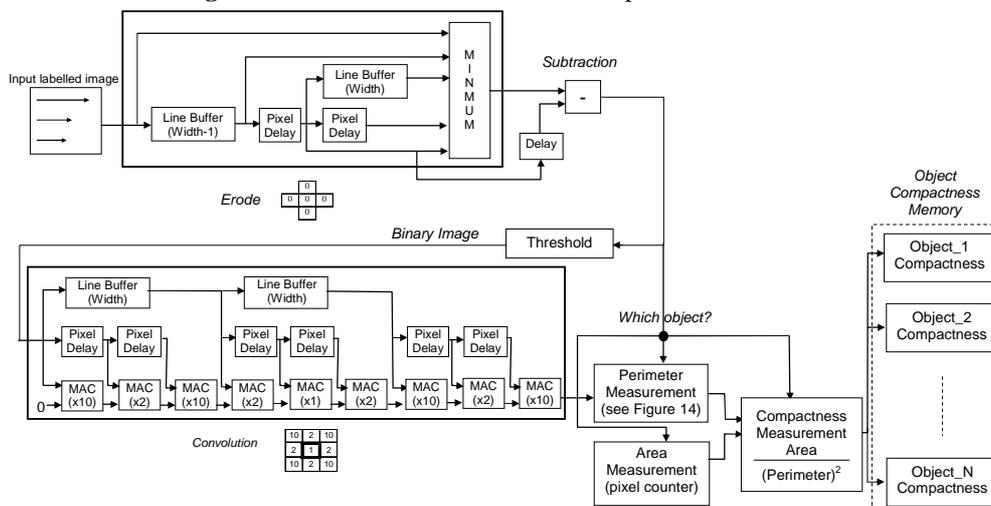


Figure 15. Hardware Architecture of the proposed compactness estimator

7. References

- [1] Ross J, 'The Image Processing Handbook', CRC Press, 1995.
- [2] Castleman K R, 'Digital Image Processing', Prentice Hall, 1996.
- [3] Milgram D L, 'Region extraction using convergent evidence', Computer Graphics and Image Processing, Vol. 5, No. 2, pp. 561-572, 1988.
- [4] Sanz L C and Petkovic D, 'Machine vision algorithms for automated inspection of thin-film disk heads', IEEE Transactions on Pattern Analysis and Machine Intelligence, 10(6), pp. 830-848, 1988.
- [5] Olariu S, Schwing J L and Zhang J, 'Fast component labelling and convex hull computation on reconfigurable meshes', Image and Vision Computing Journal, 11(7), pp.447-455, 1993.
- [6] Maresca M, and Li H, Lavin M, 'Connected component labeling on Polymorphic-Torus architecture', IEEE International Conference on Computer Vision and Pattern Recognition, Ann Arbor, pp. 951-956, 1988.
- [7] Mozef E, Weber S, Jaber J and Prieur G, 'Parallel architecture dedicated to image component labeling in O(nLog n): FPGA implementation', Proc. SPIE, Vol. 2784, pp.120-125, 1996.
- [8] Crookes D and Benkrid K, 'An FPGA implementation of image component labelling', Proc. SPIE Configurable Computing: Technology and Applications, Boston, 1999, pp. 17-23.
- [9] Benkrid, K, Sukhsawas, S, Crookes, D, Benkrid, A, 'An FPGA-Based Image Connected Component Labeller', Proc. of FPL 2003, pp. 1012-1015, September 2003.
- [10] Jabloski M and Gorgon M, 'Handel-C implementation of Classical Component Labelling Algorithm', Proceedings of the EUROMICRO Systems on Digital System Design, DSD 2004.
- [11] Agility Design Solutions, 'Handel C information sheets', <http://www.agility-ds.com>
- [12] Freeman H, 'Computer processing of line drawing images', Computer Surveys, Vol. 6, pp. 57-97, 1974.
- [13] Benkrid, K, Crookes, D, and Benkrid, A, 'Design and FPGA Implementation of a Perimeter Estimator', Proc. of IMVIP'2000, Belfast, September 2000, pp.51-57.

Multiresolution Analysis on Reconfigurable Hardware for Imaging

Abbes Amira

School of Engineering and Design
Brunel University, West London
email abbes.amira@brunel.ac.uk

Abstract— Multiresolution analysis is very useful in many image processing applications including intelligent information retrieval and medical imaging. In this paper, discrete wavelet transform has been explored for two applications namely latent semantic indexing and medical image segmentation. The slowest parts of the proposed systems have been also accelerated on reconfigurable hardware using the RC1000 FPGA prototyping board. Efficient implementations and architectures have been developed for Haar wavelet transform and A`Trous algorithm which have shown better performance compared to existing systems in place.

I. INTRODUCTION

Wavelets are a mathematical tool for hierarchically decomposing functions. They allow a function to be described in terms of a coarse overall shape, plus details that range from broad to narrow. Regardless of whether the function of interest is an image, a curve, or a surface, wavelets offer an elegant technique for representing the levels of detail present. This paper is intended to provide people working in imaging with some intuition for what wavelets can be used, as well as to present efficient reconfigurable architectures to accelerate this type of decompositions.

The first application addressed is latent semantic indexing (LSI) used in intelligent information retrieval. It is used as an attractive alternative to traditional keyword matching approaches due to its ability to address the problems arising from lack of co-occurrence, use of synonymous, near synonymous, and polysemous words as dimensions of document representations [1]. LSI is implemented in several stages. The first stage is to pre-process the database of documents. A term document matrix (TDM) is then generated which represents the relationship between the documents in the database and the words that appear in them. This is followed by a decomposition of the TDM using singular value decomposition (SVD) or QR factorization [1]. The document set is compared with the query

and the documents which are closest to the user's query are then returned. In this paper, a hybrid approach using Haar wavelet transform (HWT) and SVD is used to enhance the existing LSI based approach. TDM is visualized as an image as illustrated in Fig. 1 for Cochrane database, then image processing transforms and techniques are explored for processing the TDM and enhance the database content.

The second application addressed is medical image segmentation using multiscale statistical techniques as illustrated in Fig 2. It combines the A`Trous algorithm and a new Markov random field model (MRFM) to quantify and segment the tumour for radiotherapy planning and cancer diagnosis.

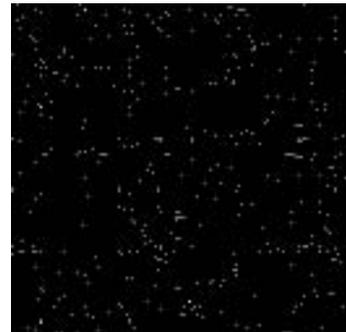


Fig 1. Cochrane TDM represented as a grey scale image. The Cochrane database contains titles of medical studies:
www.updatesoftware.com/publications/cochrane/

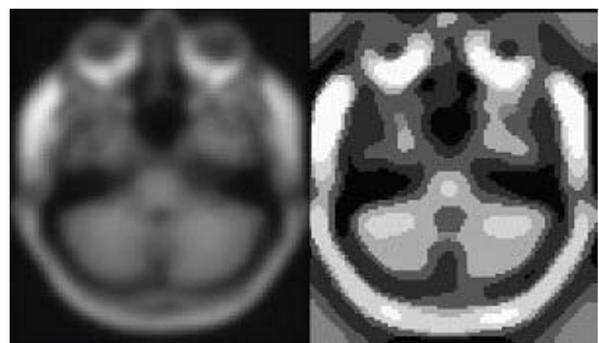


Fig 2. PET brain image segmentation using the system developed

A number of hardware acceleration techniques have been proposed for text retrieval [2][3] and medical image segmentation [4]. None of these approaches have gained significant attention due to continuous changes and improvement in information retrieval and medical imaging algorithms that make ASIC implementation infeasible. However, Field Programmable Gate Arrays (FPGAs) can be used as an accelerator for the basic building blocks in the system used in LSI for large databases and medical image segmentation. The reconfigurable nature, availability of C to hardware rapid development design flows, low cost and high performance nature due to massive parallelism capabilities makes a good case for offloading the slowest parts in these applications on FPGAs. The structure of the rest of the paper is as follows: The proposed system environments for selected applications are presented in sections 2 and 3 respectively. Implementation results are provided in section 4. Concluding remarks are stated in section 5.

II. A RECONFIGURABLE ENVIRONMENT FOR SVD/HAAR HYBRID APPROACH FOR LSI

Different components of the LSI system and the proposed hybrid technique based on SVD/HWT are shown in Fig. 3. Each database used is held as table in Microsoft Access. The tables are in the form: ID Title. The 'Title' field holds the document title from which the keywords are generated. The 'ID' field provides acts as a unique key for each entry in the table, allowing documents to be referenced easily. The Memos database is a small database consisting of nine Bellcore technical memos which is widely used as a testbench by many researchers dealing with LSI [1]. Other commercial databases that have been tested in this paper include Cochrane, eBooks and Reuters.

The database-style document table needs to be converted to a TDM. Before this can be achieved, pre-processing needs to be carried out on the document set. Punctuation and meaningless words need to be removed, and the keywords necessary for construction have to be extracted. The TDM for memos is shown in Table 1.

TABLE 1. TDM FOR MEMOS DATASET

	B1	B2	B3	B4	B5	B6	B7	B8	B9
Computer	2	1	0	0	0	0	0	0	0
Eps	0	0	1	1	0	0	0	0	0
Graph	0	0	0	0	0	0	0	1	1
Human	1	0	0	1	0	0	0	0	0
Interface	1	0	1	0	0	0	0	0	0
Minors	0	0	0	0	0	0	0	1	1
Response	0	1	0	0	1	0	0	0	0
Survey	0	1	0	0	0	0	0	0	1
System	0	1	1	2	0	0	0	0	0
Time	0	1	0	0	1	0	0	0	0
Trees	0	0	0	0	0	1	1	1	0
User	0	1	1	0	1	0	0	0	0

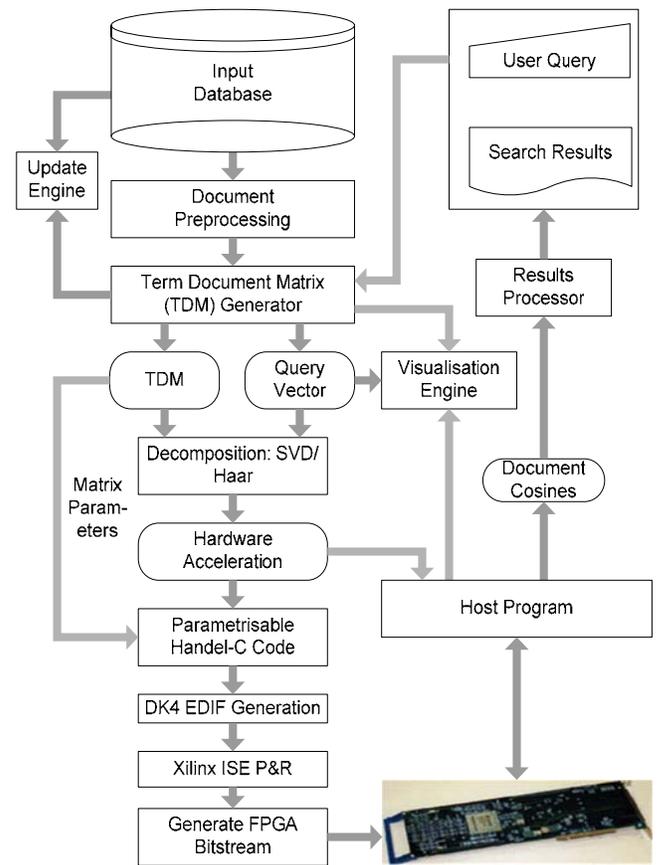


Fig 3. Proposed hybrid LSI system with hardware acceleration

The TDM is then generated from the overall keywords list which appears in the documents. Each row and column of the matrix are assigned to a term, and each a document respectively. In order for searches to be carried out, queries also have to be represented in vector form. This is achieved by the same process that is used to convert documents into columns in the TDM. Keywords are extracted from the query, and if a keyword also appears in the document set then the number of times it appears in the query is recorded in the same format as one of the document vectors in the TDM. This is followed by the SVD/Haar operations in order to perform two key functions: (i) extraction of semantic content from the search results, and (ii) removal of "lexical noise".

A commonly used approach in image processing is to combine different techniques in order to improve noise reduction. The comparison of the TDM to a gray scale image invites a similar technique. The system allows SVD and HWT techniques to be combined as below to investigate its effect on the TDM and the quality of the results. In image processing, the HWT can be used to remove noise from an image by means of thresholding in the multiresolution domain, resulting in a clean image on reconstruction. For our present purposes, the TDM can be considered as a gray scale image, usually a binary image (sparse TDM with 0, 1 probabilities). By applying the HWT and a

hard threshold to the TDM after the SVD has been performed, we can also improve the “noise” removal from our image, in this case lexical noise. Once the image is reconstructed, comparing the query with the approximated TDM should provide better results than the comparison with the original matrix. The SVD/ Haar hybrid sub-block is represented in Fig. 4.

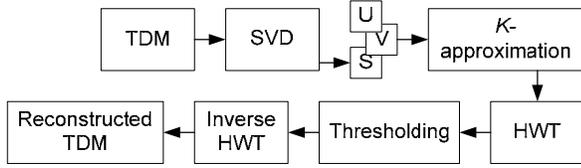


Fig 4. Hybrid SVD-HWT Decomposition

In addition, hardware acceleration of the slowest parts in the LSI process using the RC1000 FPGA prototyping board is illustrated in Fig.3.

A. SVD Applied to a TDM

A matrix M can be decomposed as $M = U * S * V^T$ where U is the singular row vectors of M , S is a diagonal matrix holding the singular values of M in ascending order V^T is the transpose of the singular value column vectors of M [2]. Matrix U can be considered to represent a term by concept matrix, matrix S as a concept by concept matrix and Matrix V as a concept by document matrix. The diagonal elements in S are stored in ascending order. The higher order values are larger and this means that they represent more of the semantic content of M (as illustrated in Fig 5). By contrast, the lower order values are small and can be viewed as “lexical noise”. The original TDM can be re-approximated by multiplying the three matrices back together again. However, if the lowest singular values of S are discarded, TDM can be re-approximated by:

$$M_a = U_k * S_k * V_k^T$$

where $k < r$, and

M_a = approximated TDM; U_k = first k columns of U ;

S_k = the new matrix of singular values, and

V_k^T = the first k columns of V^T [1]

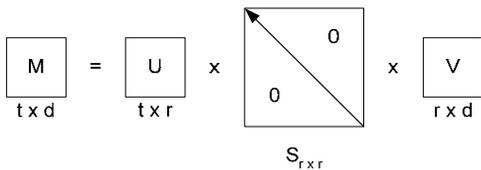


Fig 5. Diagonal Matrix S. The inner boxes represent singular values

B. The Haar wavelet transform

Popular discrete wavelets include Haar, Daubechies, Daubechies Biorthogonal, Coiflet, Symmlet etc. The Haar wavelet is one of the simplest wavelet transforms, and is of particular interest to us due to its unitary nature. The mother scaling function for the HWT is given by:

$$\psi(x) = \begin{cases} 1 & 0 \leq x < 1/2 \\ -1 & 1/2 \leq x < 1 \\ 0 & \text{otherwise} \end{cases}$$

Fast computation of the HWT can be performed by means of lifting, using the HWT matrix. The smallest matrix of size 2×2 is given by:

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

This mathematical equation for the HWT is expressed in pseudo code form as shown in Fig. 6.

procedure DecompositionStep (C: **array** [1. . h] **of** reals)

for $i \leftarrow 1$ **to** $h/2$ **do**

$C[i] \leftarrow (C[2i-1] + C[2i])/2$

$C[h/2 + i] \leftarrow (C[2i-1] - C[2i])/2$

end for

$C \leftarrow C'$

end procedure

procedure Decomposition (C: **array** [1. . h] **of** reals)

$C \leftarrow C/h$ (normalize input coefficients)

while $h > 1$ **do**

 DecompositionStep (C[1. . h])

$h \leftarrow h/2$

end while

end procedure

procedure StandardDecomposition (C: **array** [1. . h, 1. . w] **of** reals)

for $row \leftarrow 1$ **to** h **do**

 Decomposition (C[row , 1. . w])

end for

for $col \leftarrow 1$ **to** w **do**

 Decomposition (C[1. . h, col])

end for

end procedure

Fig 6. Pseudo code for Haar standard decomposition

III. A RECONFIGURABLE ENVIRONMENT FOR MEDICAL IMAGE SEGMENTATION

The proposed system for medical image segmentation is illustrated in Fig 7.

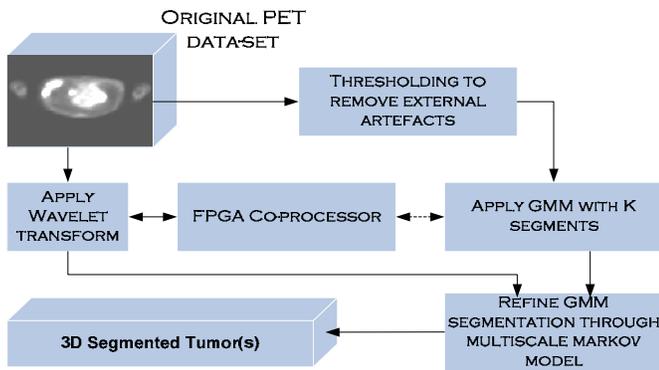


Fig 7. Proposed Segmentation System

The 3D image is acquired from PET scanner, before segmentation a thresholding technique is performed to remove artefacts and redundant data.

This type of procedure is suitable for this application due to the distribution of intensity values through images of this nature. The marginal segmentation achieved using a multiscale MRFM to segment the data based on the relationship of a voxel with its neighbours. In other words, the original 3-D image volume is decomposed using the A`Trous wavelet transform, which is performed on FPGA. The original image and selected wavelet scales are used in conjunction with a MRFM, which is considered through the utilisation of multivariate Gaussian density calculations.

IV. IMPLEMENTATION DETAILS AND RESULTS

Software simulations and reconfigurable hardware implementations have been carried out for both systems used in LSI and medical image segmentation.

A. LSI System

For the optimal value of k , the SVD-HWT hybrid approach returns one less result than the SVD approach. This is because the extra HWT step has filtered out the low frequency terms in the TDM, and has thus improved the relevance and quality of the search results. HWT post-processing can moderately improve the precision of the document set returned for a query at the expense of losing a small number of irrelevant document matches. Decomposition and query of the various databases using SVD, HWT and SVD-HWT hybrid decomposition engines delivers the following results in terms of software computational time:

Database	Average Execution Time (s)		
	SVD	Haar	Hybrid
Memos	0.018	0.011	0.029
Cochrane	0.430	0.408	0.775
Ebooks	42.45	4.134	45.28
Reuters	111.245	4.433	114.54

In order to provide hardware acceleration for the proposed LSI system, the slowest sub-blocks of the design have been prototyped on the Celoxica RC1000 board containing the Xilinx XCV2000E FPGA. A full explanation of the architectural and performance and FPGA implementation details for a modular and parametrisable SVD core developed by our research group can be obtained from [8]. The HWT hardware implementation proposed in this paper occupies 259 FPGA slices and operates at a maximum frequency of about 67 MHz. The hardware implementation of HWT clearly outperforms the software implementation by a speedup factor of 5.5 times, and it has been shown in [8] that the hardware implementation of SVD outperforms comparable optimised software implementations by a factor of 2:1.

B. Medical Image Segmentation

The multiscale MRFM developed requires significantly more computation than the spatial domain MRFM. For an image with dimensionality 128x128x117, the multiscale MRFM, takes approximately 25 min (C source) to compute, for a pre-determined number of clusters. Times are obtained using a single processor 2.4GHz Pentium 4, with 256MB of RAM.

The above A`trous algorithm architecture with the use of a convolution with a mask of 5x5 (the B3 spline) has been implemented for three resolution levels for a 32x32 8-bit input image due to the number of BRAMs available on the FPGA. The memory blocks RAM A and RAM B have been implemented as 32x16-bit wide dual-port BRAMs available in the Virtex-E FPGA device with no additional hardware complexity. In order to store 32x32 bit image and resulting wavelet coefficients 2x32 (32x16-bit) wide BRAMs have been used.

The row delay circuit in the architecture has been implemented by 32 word depth 16-bit FIFOs with the use of embedded BRAMs. It is also important to point that the B3 spline filter coefficients are all power of 2. Therefore, the multiplication of the pointwise summation and filter coefficients in (1-D) filters (P0, P1 and P2) is implemented as right-shift operations. This way the increase in hardware area in terms of FPGA slice has been prevented. Table 3 shows implementation results for the A`trous wavelet architecture, in terms of FPGA area occupied.

TABLE 2. SOFTWARE COMPUTATION TIMES DATABASES

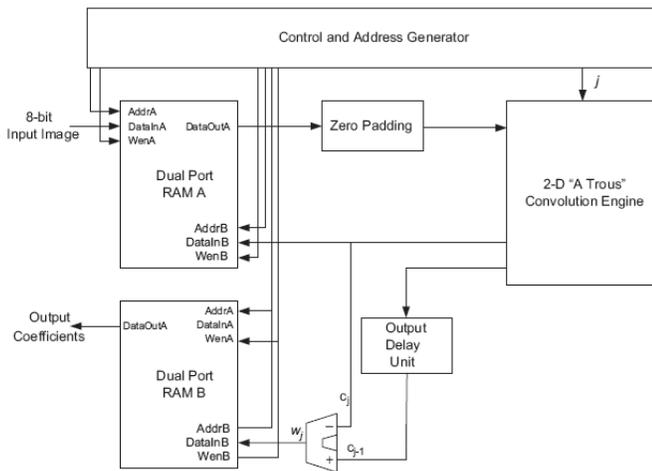


Fig 8. Proposed architecture for A`Trous algorithm

TABLE 3. HARDWARE IMPLEMENTATION OF A`TROUS ALGORITHM

Length	Area (Slices)	BRAMs	Speed (MS/s)	Computation time
32	276	68	87	3×32^2

V. CONCLUSIONS

In this paper, acceleration of multiresolution analysis algorithms was performed on reconfigurable hardware to address two applications in information retrieval and medical image segmentation respectively. An FPGA based hardware accelerator has been developed to provide computational speedup. It has been shown that the hardware implementation of the basis computational blocks of the LSI and medical imaging system clearly outperforms their software counterparts. Ongoing research involves the investigation and implementation of other multiresolution algorithms such as ridglet, curvlet...etc on the most recent FPGA devices.

Acknowledgment

I would like to thank Dr. Chandrasekaran, Dr. Uzun, Dr. Montgomery and Dr. Jaber for their contributions to this work.

VI. REFERENCES

[1] M.W. Berry, S.T. Dumais and G.W. O'Brien "Using Linear Algebra for Intelligent Information Retrieval" SIAM Review, pp 573 - 595 Volume 37, Issue 4, December 1995.

[2] S. K. Agun and O. Frieder, "HAT: A Hardware Assisted Top-Doc Inverted Index Component", Twenty-Sixth ACM SIGIR, July 28 - Aug. 1, 2003.

[3] Michael Freeman, "Hardware acceleration of information retrieval", *Euromicro Conf./DSD Symposium*, August 2005.

[4] P. Dillinger, J. F. Vogelbruch, J. Leinen, S. Suslov, R. Patzak, H. Winkler, K. Schwan "FPGA based Real-Time Image Segmentation for Medical Systems and Data Processing" IEEE Transactions on Volume 53, Issue 4, Aug. 2006 Page(s): 2097 - 2101

[5]. D. Montgomery, A. Amira and F. Murtagh "An Automated Volumetric Segmentation System Combining Multiscale And Statistical Reasoning" Proceedings of the IEEE International Conference on Circuits and Systems, pp 3789-3792, May 2005.

[6] www.xilinx.com

[7] www.celoxica.com

[8] A. Ahmedsaid and A. Amira "Accelerating SVD on reconfigurable hardware for image denoising", Proc. of the IEEE Intl. Conf. on Image Processing, pp 259 - 262, vol. 1, Oct 2004.

A design and an implementation of a parallel based SIMD architecture for SoC on FPGA

Mouna Baklouti*[†]
and Mohamed Abid*

*CES Laboratory - Tunisia

[†]LIFL, INRIA Lille North Europe

University of Lille, France

Email: mouna.baklouti@lifl.fr

Philippe Marquet
and Jean Luc Dekeyser
LIFL, INRIA Lille North Europe
University of Lille, France

Abstract—Parallel machines are most often used in many modern applications that need regular parallel algorithms and high computing resources, such as image processing, signal processing, etc. Several system on chip design methodologies are studied in order to integrate such architecture.

The aim of this work is to propose an architecture based on IP reuse. A model of a parallel SIMD architecture called Massively Parallel Processor System on Chip is defined and discussed. An implementation on FPGA is also performed in order to proof its feasibility. The proposed architecture is generic and allows the adaptation of the PEs grid size and the memory length according to the application requirements. One original aspect of this project is the use of IP to build the machine. One novel approach is also the use of the same IP for both Array Control Unit and Processing Elements.

I. INTRODUCTION

Parallel machines are most often used in many modern applications that need regular parallel algorithms and high computing resources, such as image processing, signal processing, etc. In fact, SIMD (Single Instruction Multiple Data) architectures are powerful executers especially for data intensive calculations. This class of processors has been less and less used in the ninetenths because of its dramatically high fabrication cost.

However nowadays, present system design methodologies promote component based design and enforce the use of IP (Intellectual Property) blocks and industry standard interfaces [5]. This context substantially alleviates the design cost of a dedicated processor for a SIMD system. Significant evolutions of system design, silicon integration technology and growing application computing power requirement have also change the context. So, the integration of a parallel system on a single chip will be of prime interest for parallel applications.

Massively parallel machines make use of fine-grained computational units, called Processing Elements (PEs) working in parallel to speed up computation. They are connected together in some sort of simple network topology, which often is custom-made for the type of application it's intended for. In SIMD machines a main processor, called Array Control Unit (ACU), is responsible for fetching and interpreting instructions. The Array Control Unit processor issues arithmetic and data processing instructions simultaneously to the Processing

Elements, and handles any control flow or serial computation that cannot be parallelized. Each PE then performs the global operation on its own data. This work aims at proposing a model of a parallel SIMD architecture called Massively Parallel Processor System on Chip, and presenting one implementation on FPGA in order to proof the design feasibility. FPGA has been chosen for its reconfigurability which can be exploited to implement a reusable design, adjusted for specific applications.

One original aspect of this project is the use of IP to build a parallel machine. One novel approach is also the use of the same IP for both Array Control Unit and Processing Elements.

The remainder of this paper is organized as follows. The next section presents some significant works related to SIMD systems and their integration on a chip. It highlights also the use of IPs to design most of modern SoCs. Section 3 describes the MPPSoC architecture. Section 4 presents the MPPSoC design and its implementation on FPGA. It presents also the MPPSoC toolchain needed for its programming, and discusses experimental results by executing matrix multiplication algorithm. Finally, Section 5 concludes this paper with a brief outlook on future works.

II. RELATED WORKS

The history of SIMD machines began with the ILLIAC IV project [12], [13] started in 1962. Several other SIMD machines appeared such as the Maspar MP-1 [16] and the Thinking Machine CM [19], [3]. These machines were centered on the concept of a very large number of processing elements. However, in the end of ninetenths, SIMD has fallen by the wayside in the arena of commercial general-purpose computing. One primary reason attributed to the commercial failure of SIMD machines hinges on the rate of growth of microprocessor performance relative to the time to market for SIMD machines.

But the spread of special applications that require a great deal of independent data computation such as multimedia and graphics applications as well as video or sound and signal processing has lead to the appearance of small scale commercial SIMD variants in the marketplace. They are in the form of ISA multimedia extensions, including Intel's MMX

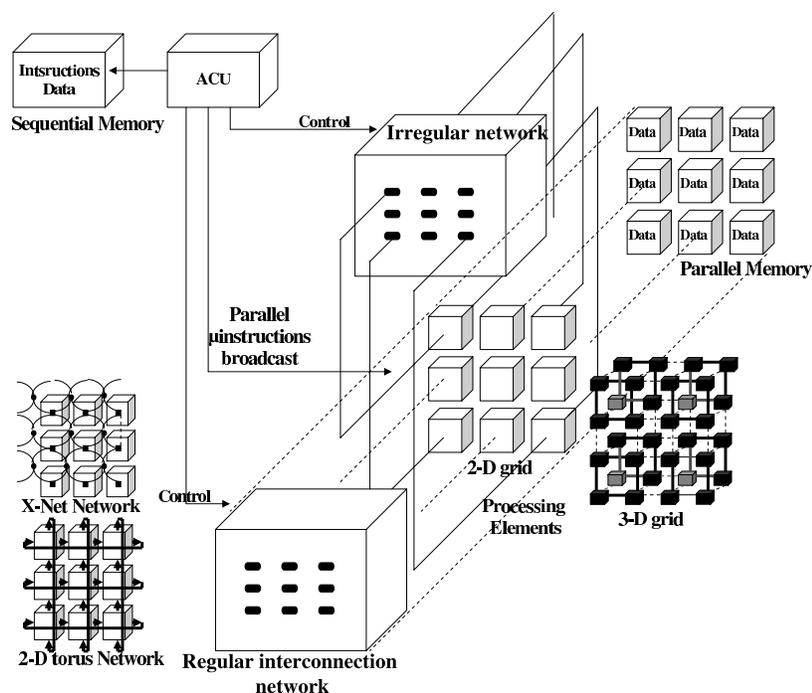


Fig. 1. The MPPSoC architecture (composed of an ACU that controls the whole architecture, an array of PEs organized in different topologies and connected to parallel memories, an irregular network and a regular network with various topologies).

and SSE, AMD's 3D-Now and Motorola's AltiVec, which are short vector SIMD architectures. In addition, with the recent great evolution of silicon integration technology, some major IP providers start designing special components for on-chip SIMD architectures acceleration; for instance, the Neon technology proposed by ARM [17], the PicoChip's PC101 array [9] and the Morphosys [6] architecture. However these architectures are dedicated to specific applications and do not treat the various needs of intensive parallel ones.

In [4], a project called CoMAP "Co-Design of Massively Parallel Processor Architectures" presents a new class of parameterizable coarse-grained reconfigurable architectures called Weakly Programmable Processor Arrays WPPA. The architecture proposed does not follow the SIMD model. In addition, this project focuses more precisely on implementing a reconfigurable NoC architecture connecting multiple processing elements.

According to all these works, there are no available SIMD massively parallel SoCs using really the recent integration technology capabilities especially in terms of IP reuse and being generic enough to suit various application needs.

In fact, design re-use and Intellectual Property (IP) trading should now be considered a necessity to face time to market constraints. So, IP reuse methodology has been introduced in SoC era to cope with very large and complex designs. It basically involves partitioning of the designs into smaller IP blocks with well-defined functionality that can be re-used across mul-

iple designs. Ideally, an IP core should be entirely portable, that is, able to easily be inserted into any vendor technology or design methodology. These cores may include embedded processors, memory blocks, or circuits that handle specific processing functions [15]. Different approaches attempt to ease IP integration today by defining design methodologies or techniques to solve specific problems. Indeed, designing parameterized IP cores with a well defined interface [2] allows cores to be quickly customized and integrated into multiple design projects.

Some EDA companies provide a set of tools that allows incorporating IP cores for high level specification and system cosimulation [10]. The VSIA [18] and OCP-IP [7] standards have produced specifications which dictate core interfaces to promote IP reuse and speed up integration. Other new standards, emerging today, include SPIRIT and SPRINT (Open SoC Design Platform for Reuse and Integration of IP) which are proposed to facilitate the IP integrator's reuse process, thus enabling a higher degree of automation across the SoC design process.

Thus, to obtain a short time to market, less costs, better performance and to provide an optimized and generic design, our goal is to build a massively parallel SIMD architecture based on IP reuse. In the following, we define our massively parallel architecture and we describe its different components.

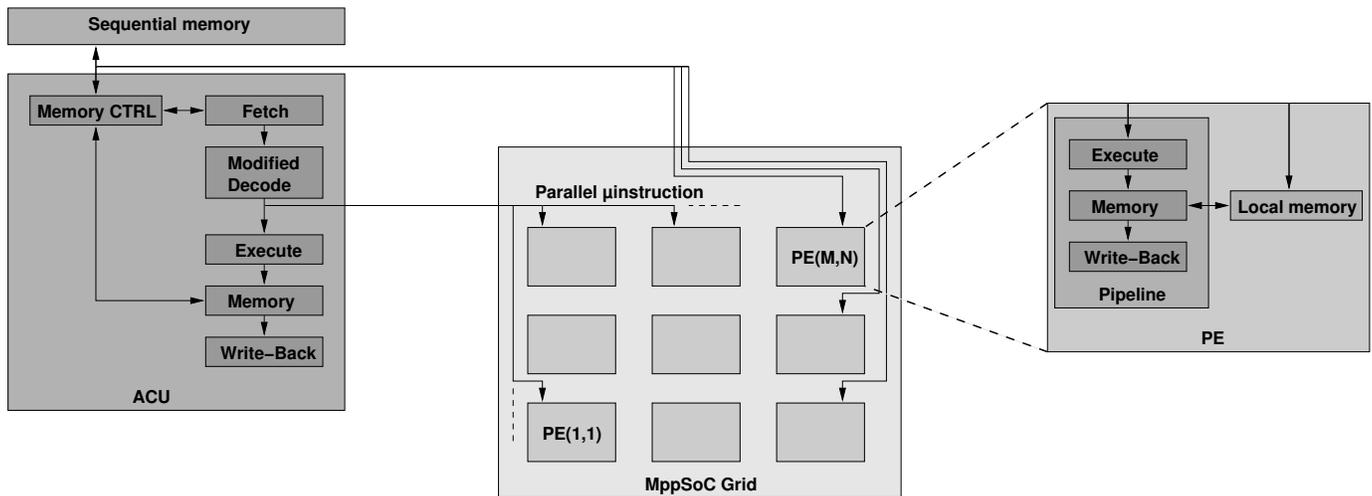


Fig. 2. MPPSoC implementation on FPGA: the left-hand side represents the ACU and its sequential memory. The 2D grid of PEs is represented in the center of the figure. The PE is build from the three last pipeline stages of the MIPS processor chosen for the implementation, and is connected to a local memory as shown on the right-hand side of the figure.

III. MPPSoC ARCHITECTURE

Our work revisits the SIMD concepts, expands and interprets them with modern requirements and technological constraints. We propose a Massively Parallel Processor System on Chip MPPSoC, close to the very well known MasPar [16]. The general basic MPPSoC architecture is composed of a number of processors (the PEs), organized in a defined topology such as a 2-D grid or a 3-D grid, working in a perfect synchronization. A small amount of local and private memory is attached to each PE. The latter is potentially connected to its neighbours, which are of number 8 if we use the regular network X-Net. Furthermore, each PE is connected to an entry of mpNoC, a massively parallel Network on Chip that potentially connects each PE to one another, performing efficient irregular communications. The mpNoC could be of different types like a crossbar or a multistage network for example. The system operating is organized by a control processor (the ACU, Array Control Unit) that accesses its own sequential memory which contains data and instructions. The ACU reads instructions and, according to the executed program, synchronously controls the whole MPPSoC architecture within these three ways: a) parallel instructions which are broadcasted to the grid of PEs which may be of different topologies, b) regular interconnection network control which establish its connection topology (torus, xnet,...) and c) the global router control which determines the route of the irregular communications. Figure 1 illustrates the global generic MPPSoC architecture [11].

There are two major differences between our system MPPSoC and the original MasPar [16]. They are related to memories accessing issues and processing elements. MPPSoC offers direct connections from the ACU to the parallel memories, which remains an original solution to provide fast communication. The second difference concerns processors. Both ACU and PEs are constructed from the same IP processor. Thus,

using the same IP, the dramatically high design cost linked to SIMD machines is considerably reduced. This solution remains, according to our knowledge, a completely novel approach in SIMD design field.

Adding to that, the originality of our work remains in designing one SIMD architecture MPPSoC, taking into consideration the applications and their various demands. The main parts like the ACU, the PEs and the interconnection networks will always be in the system, but they will be designed to be reusable and parameterized.

Below, we define the basic IP components of MPPSoC architecture.

A. Processors

Processors include both ACU and the Processing Element. Our work proposes to use an existing processor design as a base to reduce the time needed for implementing and testing a custom soft processor. We have also modified the instruction set by adding parallel instructions. The instruction set extension has been chosen to be compliant with the MIPS32 standard one.

The ACU is connected to its sequential memory, which contains data and instructions. It is build as a modified processor which produces a micro-instruction at the end of the decode stage. This micro-instruction is either executed locally by the ACU in the case of a sequential instruction, or is broadcasted to all the PEs in the case of a parallel instruction [11]. The new instructions added give mainly to the ACU the following functionalities: distinguish sequential and parallel instruction, check activity state of the PEs, and broadcast data to the PEs.

PEs are less complex than the ACU. In fact, they do not have fetch and decode stages in their pipeline. This significant gain will allow integrating a large number of PEs on a chip. We note that the PE write-back stage is controlled by the

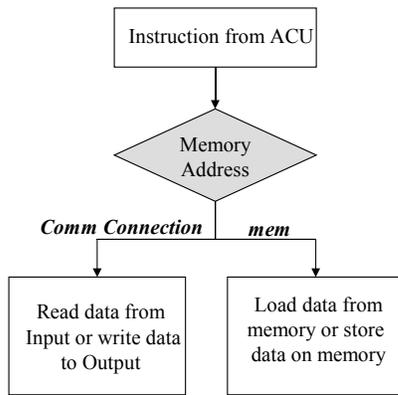


Fig. 3. Memory instruction flow diagram

activity bit of each PE: operations are realized but their results are not stored in registers or memory if the PE is inactive. Each PE is connected to its local data memory, to a regular interconnection network and to the global router. The PEs local memories are directly accessible from the ACU, keeping so the design flexible and allowing faster read/write access to local memories than transferring via PE.

Moreover, the basic design of the PE consists of a memory block, an I/O block and a processor unit containing the three execution stages fed with instructions via an instruction bus. So, data are received by the ALU either from the I/O, the memory or from the instruction word. If the data comes from the I/O, it is sent by another PE, received by the I/O block and forwarded to the ALU using the internal data bus. If the data is read from the memory it will be fetched from the block-RAM by the memory block and sent to the ALU via the same internal data bus. The memory block in the PE controls the on-chip block-RAM, which receives instructions from the ACU when data should be loaded from, or stored on the memory.

The functioning of the I/O is similar to that of memory. The only difference is that we deal with specific memory addresses which define the address range of the I/O block, as illustrated in figure 3. In fact, the I/O block is a specified part in the memory used for external communication i.e. the moving of data between PEs. Two separate buses are used for data input and output. The input and output buses are connected to the regular network making the topology responsible for which PEs that are neighbours, or to the irregular network to perform point to point communications to another PEs. So, the main function of the I/O block is to load and store data to the PE from the router. If it is an input instruction the I/O block stores the data from the input bus and forwards it using the data bus to the PE processor. If the instruction is an output, the data in the I/O block is immediately put on the output bus.

B. Interconnection networks

The choice of interconnection is of great importance in SIMD architectures. When constructing a SIMD system, a large part revolves around the interconnection network, and specifically its topology. In fact, different applications require

different interconnection networks for maximum performance. It is, therefore, necessary that the network is both effective and easy to modify to suit different applications. And one of the goals of our work is to be able to implement any kind of topology, from the simplest bus systems to a multistage network.

MPPSoC contains a neighbourhood network and a global router. The MPPSoC regular network allows inter-PE regular communications. It can provide connections to the neighbors in the four primary directions (north, south, east, west) like most parallel machines (2D mesh, torus, etc.) or to enable each PE to communicate directly with its eight nearest neighbours (four primary directions as well as the four diagonal directions). Moreover, point-to-point communications are more tedious and difficult. In fact, data transfers to and from the array are often a bottleneck in parallel systems. Consequently, a global router is also integrated on the MPPSoC. The global router connects all PEs and their peripherals together. This network allows point-to-point communications between any PE. It provides a way to send/receive data into parallel memories, offering an efficient communication way with peripheral. The counter part of such network is the conflict management, which increases data transfer latencies. This global router is generic and depend on the application needs. It is designed based for example on a crossbar IP or a multi stage network IP according to the MPPSoC architecture needs and performances.

After defining the all MPPSoC architecture, we performed an implementation of one configuration as a case study to proof the feasibility of our design. This FPGA implementation is described in the next section.

IV. DESIGN AND IMPLEMENTATION OF MPPSoC ON FPGA

A first hardware implementation of the MPPSoC was set as a feasibility study of the designed architecture. Targeting an FPGA platform was chosen for the sake of its, relative, simplicity and re-programmability. This implementation was a practical experiment of an IP reuse, which remains, according to our knowledge, a novel approach in SIMD field. Both the ACU and PE implementations were designed from the miniMIPS [8], a pre-existent MIPS implementation on FPGA. The choice of an FPGA implementation allows also a generic design that can be tuned with respect to the effective targeted application. Our prototype generated an FPGA configuration parameterized by the number of PEs and by the amount of local memory attached to each PE.

A. Implementation methodology

The current validated implementation, shown in figure 2, consists of two main components: ACU and a number of Elementary Units which derived from miniMIPS. Each elementary unit composes of processing element, its local memory and its chip select. So the first implementation is carried in hierarchical manner. In fact, the grid elements are included in a hierarchic unit called elementary unit. An intermediate

hierarchic level grouping several PEs has been also defined to help the synthesis tool and to facilitate the routing.

This MPPSoC FPGA implementation does not yet integrate interconnection networks. But different elements are defined to achieve such implementation. Our aim is to be able to integrate networks of different topologies depending on application requirements. The network should be scalable from a very basic topology such as a bus, to a complete all-to-all network like a crossbar. The different parts will be constructed as IP-blocks, which will be connected with a chosen network architecture using SoC design. The benefit of using IP-blocks is that the essential core of the system can easily be lifted out of the design and introduced into another one, which is important for reusability.

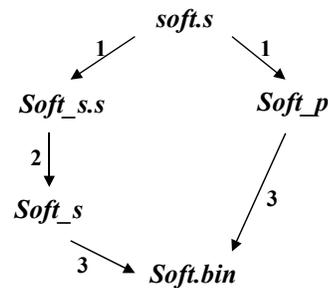
As the ACU can accede to PEs local memories, establishing the connection mode ACU – PE, we want also that any PE can communicate with any other by sending or receiving data (mode PE – PE) or will be able to send data to the ACU (mode PE – ACU). So an I/O block is designed to assure communication between PEs. The I/O block is connected to the network in one end and to PEs or peripheral units (in the case of global router) in the other. The network topology will determine how data is sent to and from PEs.

The interface between ACU and elementary units captures the micro-instructions, as presented by figure 2, signals to communicate with memories (sequential memory and parallel memories) and control signals. More precisely, connections between the ACU and the PEs are signals delivered from the ACU decode stage to the PE input execute stage. In fact, a parallel instruction sends the micro instruction to the PEs and a NOP in the ACU execute stage, while a sequential instruction performs things in reverse order.

For our first experiments, we target the Altera Stratix-2S180 FPGA that contains 179k logic elements (LEs), a number of dedicated memory blocks, and 96 DSP blocks. Our implementation is a VHDL code that can be used either as an input of the ModelSim Altera simulator, or as an input of the Quartus II synthesis tools [1].

In this work, we tested one MPPSoC prototype integrating 64 PEs placed in a 2D grid, with 4kB of memory per PE. After compilation and synthesis, our prototype implementation on FPGA has validated several results. Beyond the feasibility confirmation of the design, we have been able to identify a rough idea of the number of Logical Elements needed to implement a given MPPSoC configuration. In our case the whole MPPSoC design uses 69% of the FPGA resources. Furthermore, our implementation uses not only the LEs, but also the memory (24%) and DSP blocks (68%) of the FPGA. In fact, to create a large design on any platform it is important to use the components that are cheap in that technology.

The analysis of the FPGA surface used shows also that a PE is much smaller than the ACU. The maximum operating frequency of our design is 50 MHz. This frequency depends mostly on the frequency of the processor miniMIPS which functions at 50 MHz.



1. *generation soft_s.s soft_p soft.s*
2. *sde-as -EL -o soft_s soft_s.s*
3. *fusion soft.bin soft_s soft_p*

Fig. 4. Assembly steps (from a parallel extended assembler language we generate the binary compatible with the extended MIPS instruction set).

B. MPPSoC Toolchain

In order to program MPPSoC, a tool chain has been developed. From a data parallel extended assembler language (the extended language allows parallel expression and parallel variable manipulation), we are able to generate the corresponding binary compatible with the proposed extended MIPS instruction set and so compliant with MPPSoC.

Extended instructions include arithmetic and logical instructions and access memory instructions. However, branch instructions or system calls were not extended, because PEs can execute all MIPS standard instructions except those using PC register (instructions executed only by the ACU). We developed *generation* and *fusion* tools (written in C language). The script-shell *mppsocasembly* allows then to generate a binary executable from a source file in extended assembler, using also our tools as well as *sde-as*.

Generation Tool

Generation is the first tool to use for the generation of an executable file. The command used is: *generation dest_s_src dest_p src*. It permits, from an extended assembler source file *src*, to separate the MIPS code from the extended code. After execution, *dest_s_src* contains only the assembler *src* instructions compatible with MIPS. Compared with *src*, instead having the extended assembler additional instructions, we placed NOP instructions. The second file generated is *dest_p*. It contains so the specific extended instructions and it is created directly in binary form. This method allows us to get a MIPS assembler source file. Then after using *sde-as*, we get two binary files: the first is an executable file in elf form that does not contain the extended instructions, the second is the one containing the extended instructions of the program.

Fusion Tool

The second tool is named *fusion*. It allows us to obtain the binary executable file by typing: *fusion dest src_s src_p . src_s* and *src_p* are the two binary files obtained previously (by generation and *sde-as*). *dest* presents the final binary file. In this step, we copy the *src_s* file where NOP instructions are replaced by an instruction of *src_p*. This latter is therefore

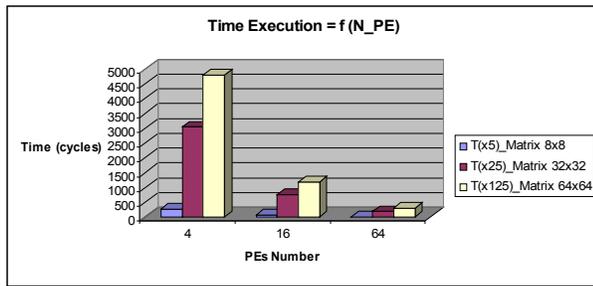


Fig. 5. Time execution of matrix multiplication on different MPPSoC versions parametrized by the PEs number

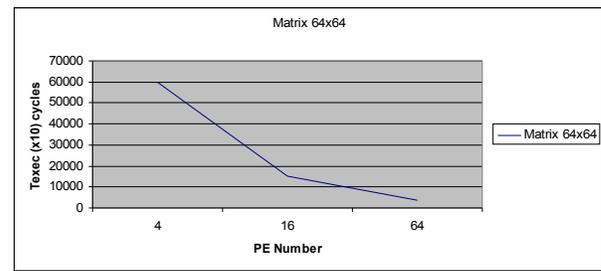


Fig. 6. Time execution of 64x64 matrix multiplication parametrized by the PEs number

read sequentially. The number of instructions in *src_p* is equal to that of NOP instructions in *src_s* file.

Mppocasassembly Tool

The third tool used is *Mppocasassembly* which is a simple script-shell allowing generating an extended executable directly from an extended assembler source file. It is used as: *mppocasassembly dest src*. It compiles the generation and fusion tools if necessary. Then, it uses *generation*, *sde-as* and *fusion*. The diagram shown in Figure 4 summarizes the assembly different steps.

Finally, the binary is then downloaded into instruction memory, itself downloaded into FPGA (via bit stream). The program is then executed by the VHDL MPPSoC design. The bit stream is provided by Quartus synthesis tool in few seconds since MPPSoC architecture synthesis is not necessary. This flexibility allows us to powerfully manipulate MPPSoC according to a particular program.

C. Experimental Results

In this section we will present experimental results when executing matrix multiplication with different matrix sizes on MPPSoC parameterized by the PEs number. In fact, matrix multiplication is commonly used on almost intensive data parallel algorithms. It is a very compute intensive task, but also rich in computational parallelism, and hence well suited for parallel computation.

In this work, we consider the basic algorithms for the special case of square $N \times N$ matrices that fit exactly on a N^2 processor machine. When N becomes large, block matrix multiplication is used to divide the matrix into smaller matrices blocks. The matrix multiplication is defined as $C=A*B$, where $c_{i,j} = \sum_k a_{i,k}b_{k,j}$ for all i,j .

At the end of the algorithm, each processor $P(i,j)$ will hold the element $c(i,j)$ of the product matrix. Two core strategies are employed in our implementations to minimize the ratio of memory accesses: accumulate results in registers for as long as possible to reduce write backs and re-use values in registers as much as possible.

We need to multiply two matrices to form a third. So, we consider the second matrix as a collection of column vectors, with each one assigned to a separate processor, one can then

perform the multiplication almost as fast as on the single vector.

In our experiments, we consider in a first step multiplication of two $N \times N$ matrices on N^2 PEs; and in a second step a 8×8 , 32×32 and 64×64 matrix multiplications are tested on 4, 16 and 64 PEs. So, in our algorithms we use matrix decomposition, when needed, based on the number of processors available. For example, with 4 PEs, we decompose our 8×8 matrix in 4 matrices blocks 4×4 . A 8×8 matrix multiplication necessitates that each processor performs 28 operations: 16 integer multiplications and 12 integer additions.

During execution, a processor calculates a partial result using the block matrices it currently has access to. When all multiplication is complete, the ACU reassembles the partial results to generate the complete matrix.

The figure 5 illustrates time execution, in cycles, to compute the multiplication of a 8×8 matrix, a 32×32 matrix and a 64×64 matrix using three different parallel architecture versions with 4, 16 and 64 PEs.

We see clearly that when increasing the number of processors performing the computation in the SIMD architecture, the matrix multiplication speeds up and we achieve so fast matrix-matrix multiplies. Test results show sustained high performance when the machine comprises more processors. This is due to the fact that each processor executes fewer operations to produce its result.

when considering a matrix multiplication 64×64 and by varying the PEs number, we notice that the time needed to perform computation is almost linear with the PE number as illustrated in Figure 6. In fact, with 16 PEs for example MPPSoC is 4 times faster than with 4 PEs. Therefore the speedup is significant and in line with the number of processors.

So, we have tested a matrix multiplication parallel algorithm and shown that it can be efficiently implemented in our first MPPSoC prototype. Due to task partitioning, there is no need for communications between processing elements. However, matrix multiplication application should be more efficient and easy when implementing it on a SIMD machine with interconnection networks [17]. Thus, our futur goal is to compare performances results when executing this application on MPPSoC with interconnection networks.

V. CONCLUSION

In this paper we presented a parallel SIMD architecture model for SoC, called Massively Parallel Processor System on Chip. It is composed by a sequential processor and controller, the Array Control Unit (ACU), a grid of Processing Elements (PEs) and regular and irregular networks. One design aspect distinguishes our proposal from the others by designing our architecture based on IP reuse. Using the same IP to provide both ACU and PEs, offers a large gain in term of development time. Our goal is to propose a methodology toward building MPPSoC generic architecture based on IP reuse.

In this paper, we presented a first FPGA implementation which is generic and allows the modification of the PEs grid size and the memory length according to the application requirements. We tested then our prototype by executing a parallel matrix multiplication application with different PEs grid size. The experimental results show high performances with the increasing number of processors and so with the most massively parallel architecture to multiply large matrices. This application will be more easily implemented considering communications between processors.

So, future works consist on integrating the interconnection networks in the MPPSoC architecture and execute more data parallel applications. When implementing MPPSoC networks, performance and reusability have to be addressed, in order to reduce the time needed for an inter-PE communication and to provide the topology suitable for the application respectively.

The ultimate goal is to build a generic MPPSoC with its different components that should be reusable and adjustable in several parameters such as bus width, memory size, number of PEs and network topology. This parameterization makes it possible to tailor the architecture for a specific application and thereby increasing its effectiveness.

REFERENCES

- [1] Altera, <http://www.altera.com>.
- [2] A. Sangiovanni-Vincentelli and J. Rowson, *Interface-based design*. In Proceedings of Design Automation Conference, 1997, 178-183.
- [3] Brewster A. Kahle and W. Daniel Hillis, *The Connection Machine model CM-1 architecture*, IEEE Transactions on Systems, Mans, and Cybernetics, 1989, 707-713.
- [4] D. Kissler, F. Hannig, A. Kupriyanov and J. Teich, *A Highly Parameterizable Parallel Processor Array Architecture*. In Proceedings of the IEEE International Conference on Field Programmable Technology, Bangkok, Thailand, 2006, 105-112.
- [5] ITRS, *International Technology Roadmap for Semiconductors*, Design, 2005 edition.
- [6] M. Lee, H. Singh, G. Lu, N. Bagherzadeh, F. J. Kurdahi, Eliseu M. C. Filho and V. C. Alves, *Design and implementation of the MorphoSys reconfigurable computing processor*, The Journal of VLSI Signal Processing, 2000, 147-164.
- [7] Open Core Protocol International Partnership (ocp-ip). <http://www.ocpip.org/>.
- [8] OpenCores, *miniMIPS overview*, <http://www.opencores.org/projects.cgi/web/minimips>.
- [9] P. Claydon, *picoChip: A massively parallel array processor*, In Embedded Processor Forum, San Jose, 2003.
- [10] P. Coussy, A. Baganne and E. Martin. *A design methodology for integrating IP into SOC systems*. In Custom Integrated Circuits Conference. Proceedings of the IEEE, 2002, 307-310.
- [11] P. Marquet, S. Duquennoy, S. Le Beux, S. Meftali and J. L. Dekeyser, *Massively Parallel Processing on a Chip*, Conference on Computing Frontiers, Proceedings of the 4th international conference on computing frontiers, Italy: Ischia, 2007.
- [12] R. L. Davis, *The Illiac IV processing element*, IEEE Transactions on Computers, 1969, 800-816.
- [13] R. Michael Hord, *The Illiac IV the first supercomputer*, Computer Science Press, 1982.
- [14] S. Duquennoy, S. Le Beux, P. Marquet, S. Meftali and J. L. Dekeyser, *MpNoC design: Modeling and simulation*, In 15th IP Based SoC Design Conference, France: Grenoble, 2006.
- [15] S.J.E. Wilton and R. Saleh, *Programmable logic IP cores in SoC design: opportunities and challenge*. In Custom Integrated Circuits, IEEE Conference, 2001, 63-66.
- [16] T. Blank, *The MasPar MP-1 architecture*, In Proceedings of the IEEE Compeon Spring, CA: San Francisco, 1990, 20-24.
- [17] T.Sorevik, P.Bjorstad, F.Manne and M.Vajtersic. *Efficient matrix multiplication on SIMD computers*. In SIAM Journal on Matrix Analysis and Applications, January 1992, 386-401.
- [18] Virtual Socket Interface Alliance Component Interface Standard. In VSI Alliance, <http://www.vsia.org/>.
- [19] W. Daniel Hillis, *The Connection Machine*, The MIT Press, Cambridge, 1989.

FPGA Design of ICA for real time Blind Signal Separation

M. OUNAS

Faculty of Electronics and
Informatics, USTHB.
BP 32 El Alia, Bab
Ezzouar, Algiers, Algeria
ounas_mouloud@yahoo.fr

S. CHITROUB

Faculty of Electronics and
Informatics, USTHB.
BP 32 El Alia, Bab
Ezzouar, Algiers, Algeria
s_chitroub@hotmail.com

**R. TOUHAMI and S.
GAOUA**

Faculty of Electronics and
Informatics, USTHB.
BP 32 El Alia, Bab
Ezzouar, Algiers, Algeria
gaoua_s,rtouhami@yahoo.fr

M. C. E. YAGOUB

School of Information
Technology and
Engineering (SITE)
Ottawa, Ontario Canada,
K1N 6N5
myagoub@site.uottawa.ca

Abstract— When physically implemented, Independent Component Analysis (ICA) algorithm can achieve a real time Blind Signal Separation (BSS). However, due to the limited size of the hardware device in microelectronics technology, several constraints can be encountered to reach the real time processing since the application of the ICA algorithm requires the consumption of a huge number of input signal samples. Hence, the system performance was degraded since we required the processing of an important number of memory circuits with faster hardware execution time. Therefore, in order to improve the hardware performances of the device, the authors proposed the sequential processing of one neuron hardware model based on Field Programmable Gate Array (FPGA) implementation. Such approach overcomes the consumption resources and the interconnections complexities of the FPGA architecture. Thus, an optimal hardware design can be proposed in which a maximum number of samples can be handled while maintaining high speed of hardware processing time. The proposed approach was demonstrated through an experimental study of TIMIT database exhibiting a hardware execution time of 3.3 μ s to process 10000 samples with 57 kHz of sample rates to separate two output independent signals from two input mixed signals.

I. INTRODUCTION

In the past, it was not possible to study the hardware implementation using complex algorithms of digital signal processing (DSP) in which we cannot exhibit a real time processing through hardware devices. Nowadays, the advance of Very Large Scale Integration (VLSI) technology over complex digital circuit design for hardware implementation has open the door to the design of Application Specific Integrated Circuit (ASIC) devices that provides high computation performance. Therefore, intensive research in DSP area has been made to resolve the problem of Blind Signal Separation (BSS) for real time processing. In fact, such applications have received a particular attention due to their efficiency in the area of communication systems, including wireless networks, and speech signals [1], [2]. Hence, it will be very interesting to study the application of complex BSS algorithms since the

impact of their performances over real time processing becomes physically feasible on hardware devices.

The BSS application is sometimes used interchangeably with the processing of Independent Component Analysis (ICA) algorithm, although, technically, BSS and ICA are different tasks. For simplicity, we used the instantaneous mixture of BSS algorithm, which is suitable for BSS application that yields to low computation cost. Usually, the ICA algorithm searches for a linear transform of mixed signal provided from multi-sensors array. But, the main task of ICA is often to eliminate the dependence computation of higher order statistics, in which the final target is to estimate as close as possible the independence of signal subsets [3], [4].

The computation of ICA algorithm requires a high number of iterations, thus a lot of arithmetic computation in the learning step of the neural network to adjust the updated weights. However, to overcome such high computation tasks required by the ICA algorithm, different authors used the mutual information approach. It can help reducing the computation cost over the theoretical computing of the gradient descent processing of the learning rule of equation [5], [6]. With the recent advances in programmable FPGA circuits, one can implement and quickly verify the hardware computation of the algorithm leading to low development time. Frequently, the hardware circuit designer is almost encountered with the limitation problem of the hardware resources in the FPGA technology since the processing of ICA algorithm uses a huge amount of FPGA memory blocks.

However, in order to improve the performances of the hardware ICA processing of the FPGA design, the authors combined in this paper the computation of implementation design and the sequential operation of the hardware neuron model. Such approach led to less complex hardware architecture [7], [8] with faster hardware execution time. Thus, the aim of the FPGA design is to reduce the hardware resource cost while improving the performances of its computation. Therefore, with a large number of both sample rates and computation iterations required to update weights, we can demonstrate the command of the processing block from a

simple implementation of a principal sequencer block, which can help control signals to process with the utilization of an optimal amount of data flow. In this context, a variety of related works have been done [9], [10], [11] to point out an alternative design which focuses on both the processing of high sample rates and faster hardware execution time, which presents the platform comparison of our results. In our hardware implementation, we introduced an FPGA design that uses higher frequency rates of system clock than Charoensak and Sattar [10], and lower sample rates than Kuo-Kai and Ming Huan [11]. Thus, several advantages become significant. In fact, an FPGA implementation of the Virtex II XC2V8000 from Xilinx allows a faster hardware design and can evaluate the number of sample rates to be used accordingly to the hardware execution time.

In section II, we introduced the computation of the implementation design based on the optimized ICA algorithm processing. Section III presents the digital circuit design. The synthesis simulation results of the FPGA hardware implementation is presented in section IV.

II. COMPUTATION OF IMPLEMENTATION DESIGN

The processing computation of the algorithm can be divided into three main steps: Computation of the update weight step, the accumulation computation of the convergence of update weight step and computation of the final weight step.

A. Computation of the update weight step

In this step, the weight W_{iteij} cycles were updated through a number of iterations, noted ite , until convergence. Then, the computation of the update weight will be as follow:

For $(i=1, 2, \dots, n)$ and $(j=1, 2, \dots, n)$;

$$W_{iteij} = W_{(ite-1)ij} + \mu(1 + \varepsilon)W_{(ite-1)ij} \quad (1)$$

where ε is a stop criterion parameter defined as

$$\varepsilon = \sum_{i=1}^n \varphi(u_i)u_i < \varepsilon_{\max} \quad (2)$$

ε_{\max} is a constant that fixes the correctness of the update weight. φ is the nonlinear learning function of the natural gradient learning equation algorithm that adjusts the update weight W_{iteij} of the neural network; u_i is the input synaptic of the neuron.

φ depends on the computation of the sigmoid function of the nonlinear activation function f of the neuron. It can be approximated by the piecewise linear function [12] as:

$$\varphi(u_i) = (1 - 2f(u_i)) \quad (3)$$

and:
$$u_j = \sum_{i=1}^n W_{(ite-1)ij}x_j \quad (4)$$

where x_j is the input signal sample.

After convergence, the updated weights will be:

$$W_{c_kij} = W_{(c_k-1)ij} + \mu(1 + \varepsilon)W_{(c_k-1)ij} \quad (5)$$

where $(1 \leq ite \leq c_k)$ and $(1 \leq k \leq M)$; M indicates the maximum number of the samples of the input signals and c_k the maximum number of iterations in the convergence step (for each sample k of the inputs). The update weight will be normalized to keep it less or equal to +1,

$$W_{c_kij} = \frac{W_{(c_k-1)ij} + \mu(1 + \varepsilon)W_{(c_k-1)ij}}{W_{c_kij \max}} \quad (6)$$

where $W_{c_kij \max}$ is the maximum value of the update weight before the normalization step.

B. Accumulation computation of the convergence update weight step

In this step, the iterative accumulation computation of the convergence of update weight at each input sample k is performed until the maximum input M sample given by

$$W_{a_kij} = W_{a_{(k-1)ij}} + W_{c_kij} \quad (7)$$

where $W_{a_{(k-1)ij}}$ is the previous accumulation of the updated weights. The computation of the final accumulation of the updated weights $W_{c_{ij}}$ is thus given by

$$W_{c_{ij}} = \sum_{k=1}^M W_{c_kij} \quad (8)$$

C. Computation of the final weight step

The final weight is given by the computation mean over the processing of the input signal samples. It will be as follow:

for $(i=1,2,\dots,n)$ and $(j=1,2,\dots,n)$;

$$W_{f_{ij}} = \frac{W_{c_{ij}}}{M} \quad (9)$$

III. DIGITAL CIRCUIT DESIGN

In this section, we illustrate the proposed methodology used for the digital circuit design, highlighting the optimization of the area size of the digital circuit to improve its performances. Thus, in order to save number of hardware resources that can reduce the area size of the digital circuit, we used a model to implement one neuron. Hence, the processing of the neurons in sequential operation can implement the ICA computation, which is related to the digital circuit design. Therefore, the three function blocks illustrating the application of the computation of implementation of the algorithm can be driven by a control block, in order to implement the neuron model. In Figure 1, the digital implemented circuit of one neuron is shown. It contains the implementation of the control block that command the implementation of the RAM block for weights

and inputs. Such block memorizes the information carried over unidirectional data, for the computation of the input synaptic and nonlinear function block, and over bidirectional data, for the computation of update weight block and the computation of the final weight block.

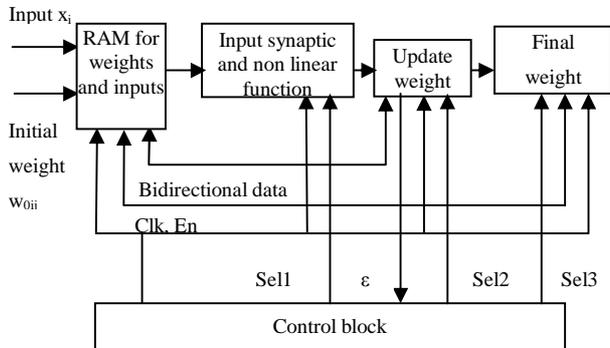


Fig.1 Digital circuit of one neuron implementation

We can transmit principally three signals from the implementation of the control block to process the selection for three demultiplexers implemented in the digital circuit of implementation of the neuron. The first one is implemented in the nonlinear function block to select the signal Sel1 that deviates the updated weights towards the block of the updated weights before convergence. The second one is implemented in the updated weights blocks before the convergence step to select the signal Sel2 that deviates the updated weights towards the final updated weights block after convergence, and the third one is implemented in the final updated weights block to select the signal Sel3 that deviates the updated weights stocking their final values in the memory of the RAM block. We implemented the normalization unit in the updated weights block to illustrate the final computation processing at each iteration of the input samples after convergence of the updated weights. Also, we can optimize the normalization unit by using a shift register that highly reduces its complexity. In the nonlinear function block, the digital circuit of the activation function of neural network is optimized when using a piecewise linear approximation to implement the nonlinear sigmoid function [12]. Furthermore, we can also reduce the complexity of the digital circuit of the neuron implementation when the adder and the multiplier illustrating the computation of the arithmetic processing are implemented with a fixed point number representation. In the arithmetic computation, we used the implementation of the Mac1 (Multiplier, Adder, and Accumulator) to compute the input synaptic in the nonlinear function block. The implementation of the Mac2, with the same components as the Mac1, illustrates the computation of the information signal ϵ transmitted towards the control block to either test the convergence of the updated weights or the computation of their adjustment.

From the digital circuit design of the control block, illustrated in Figure. 2, the diagram state of the control block

associated with the signal condition indicates the sequential operation of the computation.

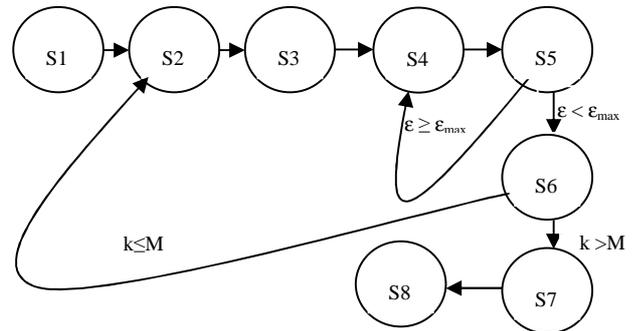


Fig. 2 Diagram state of the control block implementation

Hence, the block of the processing computation is commanded by eight main states, which are in turn controlled by two conditions. The first condition is used for the fifth state to test the information ϵ that illustrates the criterion convergence of updated weights, and the second condition is used for the sixth state to test the maximum number of the input samples.

Then, we give the function of each state as follows:

- State1 provides the signals that control the start up and the reset signals of the digital circuit of implementation.
- State2 provides the signals that control the adjustment of two address registers; the first one is used to adjust the input signal memory and the second one is used to adjust the updated weight memory that processes the first iteration of computation.
- State3 provides the signals that control the processing of the nonlinear function block and the updated weight block from the initial weight data in the first iteration step.
- State4 provides the signals that control the adjustment of the first address register for the input signal memory, and the adjustment of the address register for the updated weight memory to process the next iteration before the convergence step.
- State5 provides the signals that control the pipeline processing computation that processes the iterations illustrating the adjustment of both the updated weights for the present iteration and the value of the information ϵ computed for the next iteration (to adjust the updated weight step).
- State6 provides the signals that control the accumulation of updated weights after the convergence step for each sample of the input signals.
- State7 provides the signals that control the final value of the updated weights after the accumulation of the updated weight step.
- Finally, state8 provides the signals that stop the processing computation of implementation.

A. Digital circuit of the processing block

The different components of the digital circuit listed above are detailed in Figure.2

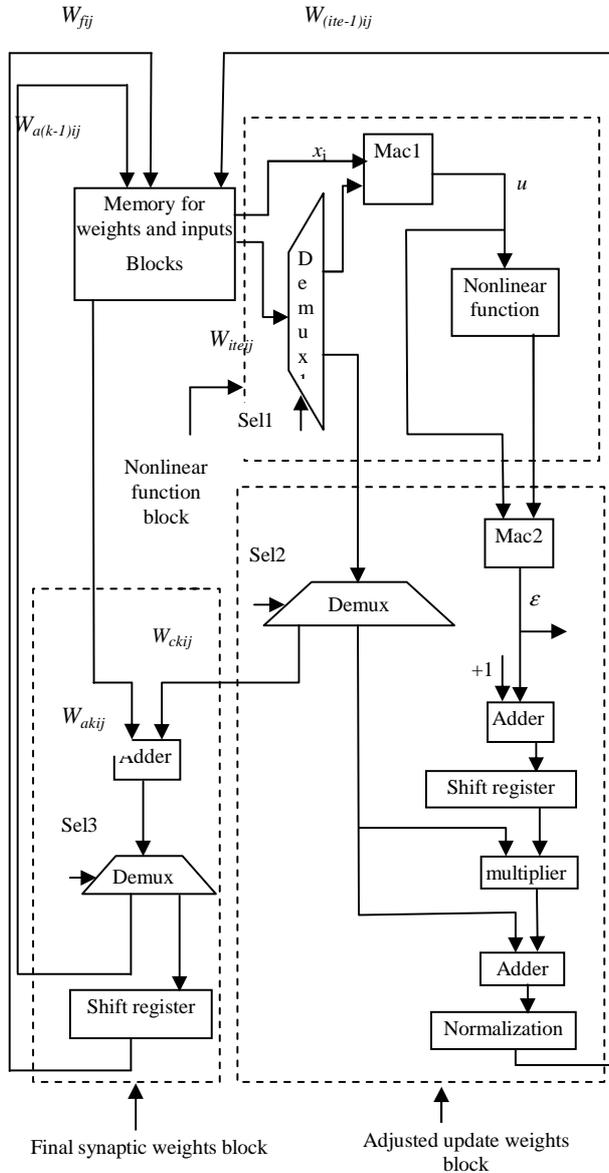


Fig.2. Digital circuit of implementation of the neuron

B. Digital circuit of the control block

To generate the control signals in the control block shown in Figure 3, we used the eight states. Thus, we implemented two flip-flops to control the states S1 and S2 and six counters to control the remaining states. To illustrate the overall control of all the states, the digital circuit of the control block used the implementation of the main counter (MOD8) called counter1. In this Figure, N is the number of the input signals and M the number of samples of each input signal. These parameters are

used along with the convergence information ϵ and ϵ_{max} to turn-on the digital circuit of the signal condition.

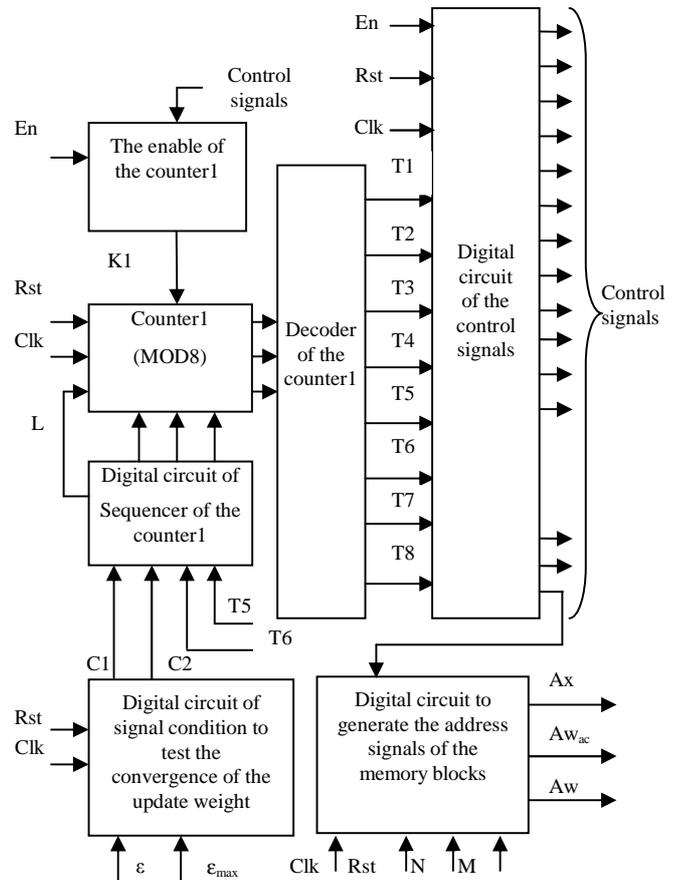


Fig.3. Digital circuit implementation of the control block

IV. FPGA DESIGN AND EXPERIMENTAL RESULTS

To demonstrate our approach, we used the FPGA software developing platform ISE 6.1 proposed by Xilinx. However, it has been found that 16 bits was the most suitable format of bit-width data for the fixed point binary number for both the input samples x_i and the synaptic weights W_{ij} . Thus, we implemented the digital circuits of the control block and the processing block to show the effects in terms of the limit size and the execution time. The hardware device we used to illustrate the FPGA design was the Virtex II XC2V8000. First, we generated the synthesis results of the hardware neuron model wrt the maximum size. We obtained a successful processing of the ICA learning algorithm based on the FPGA implementation of the neural network to adjust the updated weights from two input mixed signals of 10000 samples for TIMIT data base to separate two output independent signals.

A. System level of FPGA design

As for the FPGA design of the ICA algorithm, Du and Qi [9] implemented the hardware of the pICA algorithm for Dimensionality Reduction in Hyperspectral Images to process

2 input signals for system clock frequency up to 20.161 MHz. In 2005, Charoensak and Sattar [10] used the FPGA device of the Virtex-E family to implement a BSS algorithm that processes 8 kHz of sample rates and 64.4 MHz of system clock. Later, in 2006, Kuo-Kai and Ming Huan [11] used an FPGA design with the implementation of the FasICA algorithm in the Altera device, reaching 192 kHz of sample rates and 50 MHz of system clock.

Increasing the number of input samples will lead to more complex implementation knowing that the hardware can handle limited memory circuits.

Therefore, we studied the acquisition of the required input samples that can achieve a signal separation for each related work in FPGA design based on the ICA algorithm. In our FPGA design, the maximum required input signal that can be implemented is increasing with the number of memory circuits available (with respect to the FPGA capacity).

In Tables I and II, we compared the simulation results of our FPGA design with other published simulation results.

In order to evaluate exactly the performances of the obtained results, we illustrated the computation of the normalized learning time in Table III. It contains the implementation of the ICA algorithm based on ICNN [6] and the implementation of the FastICA algorithm [11]. We introduced another parameter to validate the computation of results; called the 'performance value', this parameter is proportional to the learning time and the number of input samples, and inversely proportional to the number of cost slices as well as the maximum frequency.

TABLE I
COMPARISON OF OUR SIMULATIONS WITH PUBLISHED RESULTS

Implementation	FPGA circuit	Cost (slices)	Learning time (μ s)	Sample rates (kHz)	Frequency Max (MHz)
Implementation of ICNN[6]	Xilinx Virtex XCV 812 E	12271	6.44	-	50
Implementation of pICA algorithm [9]	Xilinx Virtex V 1000	11318	-	-	20.161
Implementation of BSS algorithm [10]	Xilinx Virtex-E family	-	-	8	64.4
Implementation of FastICA algorithm [11]	Altera company	-	5.2	192	50
Implementation of our ICA algorithm	Xilinx Virtex II XC2V 8000	5500	3.351	57.53	185.580

TABLE II
DIFFERENT INPUT SAMPLES ACHIEVED IN FPGA IMPLEMENTATION OF ICA

FPGA circuit	Xilinx Virtex XCV 812 E	Xilinx Virtex V 1000 E	Xilinx Virtex-E family	Altera Inc.	Xilinx Virtex II XC2V 8000
Input samples	500	6000	6000	1000	10000

TABLE III
THE VALUES OF PERFORMANCES RESULTS

Implementation	FPGA circuit	Performance value
Implementation of ICNN[6]	Xilinx Virtex XCV 812 E	0.0052
Implementation of FastICA algorithm [11]	Altera Inc.	-
Implementation of our ICA algorithm	Xilinx Virtex II XC2V 8000	0.183

Then, the degree of the performance value of our implementation is 35.2 higher than the performance value of the ICNN implementation [6]. Consequently, the performance value demonstrates that we handled a maximum normalized number of input samples through an optimal value of the cost of slices with faster processing time. Such result is useful to reach a high speed of the hardware execution time when the target is the real time processing.

CONCLUSION

The FPGA design based implementation of one neuron model has reduced the area size of the digital circuit since we can get the BSS processing by sequential operation. Thus, we used fewer calculations in the ICA algorithm, mainly due to the computation of implementation design and the ICA model computation. Therefore, the most significant advantage of the digital circuit architecture is the use of one parameter that can be adjusted to control the updated weights in the learning process of the neural network. Furthermore, we significantly improved the hardware implementation performance that supports a maximum number of input samples that can maintain a high speed of hardware processing time. Hence, the performance results are satisfactory when the application was performed on speech signal separation. Thus, the BSS performance was not degraded when we utilized the FPGA design that can improve the processing time of the updated weight convergence. As future investigations, we plan to exploit the synthesis of the proposed FPGA design based implementation of the ICA algorithm to identify radio frequency signals. In fact, we have there a long processing time

to solve the anti-collision problem due to several radio wave signals.

REFERENCES

- [1] Andrej Cichocki, and Shun-Ichi Amari, *Adaptive Blind Signal and Image Processing, Learning Algorithms and Applications*, John Wiley, 2002.
- [2] Yuhon Hu, and Jenq-Nenghwang, *Handbook of Neural Network Signal Processing*, CRC Press, 2001.
- [3] Aapo Hyvarinen, Juha Karhunen, and Erkki Oja, *Independent Component Analysis*, John Wiley, 2001.
- [4] J.Bell, and J. sejnowski, "An Information –Maximization Approach to Blind Separation and Blind Deconvolution," *neural computation*, vol.7, pp.1129-1159, 1995.
- [5] Zhong Feng Li and Qiuhua. Lin, "FPGA Implementation of infomax BSS Algorithm with Fixed Point Number Representation," *International Conference on Neural Networks and Brains (ICNN&B'05)*, vol.2, pp.889-892, 2005.
- [6] A.B.Lim, R.C. Rajapakse and A.R. Omondi, "Comparative Study of Implementing ICNNs on FPGAs," *Proceedings International Joint Conference on Neural Networks*, pp.177-182, 2001.
- [7] M. Ounas, R. Touhami, and M.C.E Yagoub, "Low Cost Architecture of Digital Circuit for FPGA Implementation Based ICA training algorithm of Blind Signal Separation," in *Proc. of International Symposium on Signals, Systems, and Electronics, issse 2007*, Montréal, Québec, Canada, 2007.
- [8] M. Ounas, S .Chitroub, et R. Touhami, "Méthodologie de Conception d'une implémentation Matérielle d'un algorithme par l'Analyse en Composantes Indépendantes pour la séparation de signaux," *au 21ème Colloque International du Traitement du signal et des Images, GRETSI 2007*, Troyes, France, 2007.
- [9] H.T.Du and H.R.Qi, "An FPGA Implementation of Parallel ICA for Dimensionality Reduction in Hyperspectral Images," *Geoscience and Remote Sensing Symposium, 2004, IGARSS'04, Proceedings, 2004 IEEE International*, vol.5, 3257-3260, 2004.
- [10] C.Charoensak, and F.Sattar, "A Single- Chip FPGA Design for Real Time ICA Based Blind Source Separation Algorithm," in *Proc.IEEE International Symposium on Circuits and Systems*, pp. 5822-5825, May 2005.
- [11] Kuo-Kai Shyu and Ming Huan Li, "FPGA Implementation of FastICA based on Floating-Point Arithmetic Design for Real-Time Blind Source Separation," in *Proc.2006 International Joint Conference on Neural Networks*, July, pp.2785-2792, 2006.
- [12] H Amin., K.M. Curtis and B.R. Hayes-Gill, "Piecewise Linear Approximation Applied to Nonlinear Function of a Neural Network," *IEE Proc. Circuits Devices syst.*, 1997, 144 (6), 313-317.

High speed binary image processor for compact real time vision systems

Andreas Loos
Michael Schmidt
and

Dietmar Fey
Institute of Computer Science
Friedrich-Schiller-University
D-07743 Jena, Germany
Email: {loos,m.schmidt,fey}@cs.uni-jena.de

Jens Gröbel
Vision & Control GmbH
D-98527 Suhl
Germany

Email: jens.groebel@cs.uni-jena.de

Abstract— We present a fine-grain parallel processor chip which can be embedded in very compact machine vision systems, e. g. in 3d stacked die assemblies. Smart and fast vision systems are frequently required in industrial environments to automatically detect and inspect objects, e. g. on an assembly line. The chip die has a size of $5 \times 5 \text{ mm}^2$ and is manufactured using a $0.18 \mu\text{m}$ CMOS technology. The chip processes binary images with a maximum resolution of 320×240 pixels (QVGA) supplied by a separate closely-linked image sensor array.

It is possible to process an image in multiple cycles where a set of morphological operations can be subsequently combined depending on the image processing problem. In addition the chip is able to compute more complex user-defined programs, e. g. to skeletonize images. The output data can be a preprocessed image or projection representations in horizontal, vertical and diagonal direction, which reduces the data amount. Therefore a faster image post-processing is supported, e. g. to calculate object's momenta. The chip is driven by a 40 MHz clock. As result a base morphological operation including image in/output needs only $250 \mu\text{s}$. For even faster data in/output a ROI (region of interest) can be defined. Two standardized interfaces (JTAG, SPI) allow to parameterize as well as to program the circuit.

I. MOTIVATION AND PREFACE

The motivation to present that paper emerges from a clear tendency to substitute PC-based standard machine vision systems with smaller and faster embedded components. A number of different ways to process the data amount of image sensors are described in the literature, e. g. smart cameras based on high quality sensors and FPGAs [1], [2], asynchronous/synchronous techniques (ASPA) to process local operators introduced in [3] or smart cameras as embedded systems [4]. Our way to manage fast subtasks within image processing systems is to use application specific integrated circuits (ASICs) as basic platform. The advantages of ASIC based components confront with their main weakness: the inflexible and fixed instruction set. To meet that we present a so called ASIP

(application specific instruction set processor), which combines an instruction subset of a GPP (General Purpose Processor) with the speed of an ASIC [5]. Our work shown here bases mainly on [6], where we have presented a fine-granular parallel architecture to manipulate binary images in a fast and applicable way. In [7], [8] and [9] programmable grey value image processing architectures using higher level operators are introduced where the sensor is an integrated part of the circuits. By contrast we intend to separate the sensing and processing functionality due to the advantageously use of different CMOS process nodes for both sensor and processor as shown in [1] and [10]. Our design strategy can be summarized as follows: to integrate not as much as possible functionality but rather as much as required to fulfil the desired specifications. A certain subset of industrial applications are well defined concerning the illumination and the clearness of object and background pixel information. Therefore a complex grey value arithmetic is not required in each case.

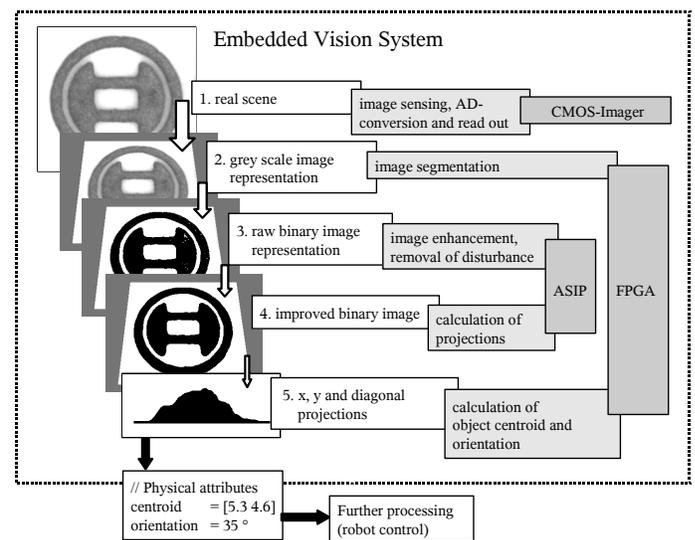


Fig. 1. Image processing flow

Before we explain some details of our system architecture we point out the characteristic data processing flow of the considered machine vision system where our processor chip will work. Figure 1 illustrates a generic procedure of the embedded machine vision environment. In the beginning a real scene is captured by a CMOS image sensor and converted to digital values (1). Afterwards the gray scaled image is segmented (2) and we receive a raw binary image representation. The quality of the image is improved by applying e. g. morphological filter operations (3). In the next step we will calculate diagonal, vertical and horizontal projections of the binary image (4). This means simply that pixels are counted what can be perfectly parallelized. This information can then be used in the next step to calculate the object's centroid and orientation, what is one of the most important tasks in industrial image processing. For step 1 we can use a commercial CMOS sensor as well as a sensor which was especially designed in our project. The steps 2 and 5 are solved by hardwired algorithms implemented in a low-priced medium class FPGA (field programmable gate array). An FPGA is the most practical solution to solve that, not only to compute the segmented image representation (such algorithms are highly application specific and should be configurable), but also to calculate the object's physical attributes like moments of first (centroids) and second order (orientation) what has to be performed in step 5. The steps 3 and 4 are calculated in our ASIP chip which is the core of the embedded real time vision system and which is the main task we present in this paper.

In particular the performance of the image processing system to be designed depends on the following parameters:

- total latency of the system *image sensor* – *image processor*
- the frame rate (time resolution)
- the number of the processed pixels (lateral resolution)
- the number of bits per pixel (color/gray scale depth)

These variables span a parameter space, where some combinations of them represent a certain application area. The application range of our imaging system covers high speed time resolution tasks (fast image sequences), short latencies, though a lower lateral resolution. The achievable color/gray scale depth depends on the used image sensor and its AD converter unit. In detail we aim to an image frequency of more than 500 Hz and latencies shorter than 5 ms from the start of the image capturing process until the availability of the results.

The rest of the paper is organized as follows. Chapter II presents the ASIP chip architecture, chapter III addresses the basic design characteristics and the chip layout. In chapter IV an embedded system for machine vision applications is introduced where our ASIP is a part of it. To illustrate the ASIP functionality an application example is given. The paper closes with some conclusions and an outlook.

II. CHIP ARCHITECTURE

In this section our chip architecture is described in a top down manner. We will start with a presentation of the main functional chip components at a glance (II-A). The most important of them are the processor core (II-B), the control unit (II-C) and the pixel counters (II-D).

A. The ASIP at a glance

Figure 2 shows a block diagram of our ASIP's architecture. The chip is separated into seven functional modules:

- Data IO (pad array)
- Boundary scan chain
- Control unit
- JTAG/SPI control
- Horizontal counter array (320 eight bit incrementers)
- Vertical counter array (240 nine bit incrementers)
- Core (main processing unit)

Before the chip can work properly it has to be initialized by writing programming data either onto the JTAG [11] or the SPI control interface (thick black arrows on the left). After the configuration is done the image processing procedure is performed in the following way:

A binarized input image is read in via the data input pads into the processing core either serially (slow mode) or in a parallel fashion (fast mode) using up to 16 channels (upper striped arrow). The control unit receives and interprets the configuration data (thick vertical arrow) in order to drive the computation modules. Hence the core module executes the image processing operations (thick horizontal arrow).

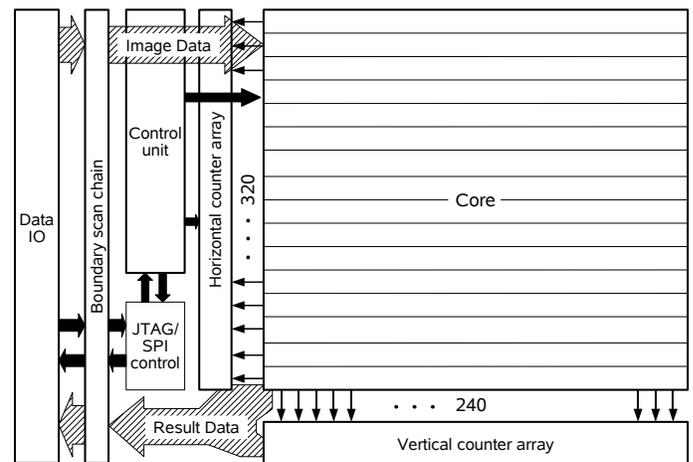


Fig. 2. Block diagram of the image processor

To count pixels in X, Y and 45 degree angle direction the horizontal and vertical counter arrays can be accessed (thin arrayed arrows). The counted results are required to determine the centroid and the orientation of objects. Either the processed image data (16 one bit channels) or the pixel counter results (17 one bit channels) are shifted out on the same (multiplexed) output ports (bottom striped

arrow), monitored by a protocol signal bus. The modules are described in more detail in the following subsections.

B. Image processing core

The core performs image preprocessing operations in a parallel fashion. To realize that the image to be processed has to be split into line-based clusters. Such a pixel cluster is serially processed by a PE whereas all clusters are processed in parallel. Figure 3 illustrates the basic core architecture. We call one PE including the corresponding storage resource (pixel and result registers 1 .. M) a multiplexed PE (MuxPE). Two register files are required to store both the original image and the result image. The reason is that each step to compute a single cluster pixel requires the neighbored pixels of the original image as input data. The PE (processing element) on the central position (with the ingoing black arrows) has signal inputs from the eight directly neighbored image pixel registers. The computation is subdivided into two steps: A new result pixel is calculated as result of a binary relation between the own pixel value and the neighbored pixels and stored into the respective result register.

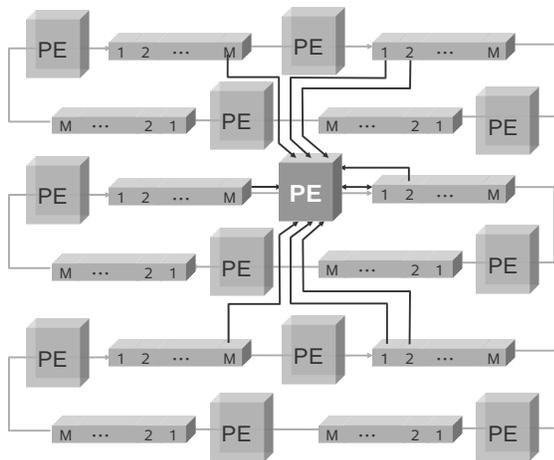


Fig. 3. Core architecture

The second step is a shifting operation, where the entire origin image and the result image is shifted to the next register cells in eastern direction. The procedure loops until all pixels in a cluster are processed.

During the computation process the two images (origin and processed image) are synchronously moved within the register files. To prevent data loss the pixel register output at the end of each processing line is directly connected to the first pixel register's input. Hence each processing line is folded to a ring on its middle, where the rings are stacked one upon the other. Further information and more details about the core architecture are given in [6].

To integrate the ROI functionality we use a scheme illustrated in Figure 4 which is easy to realize due to the folded processing lines. During the data read in the switch S_0 is in

position **o**. Before the computation it toggles to position **I** to form a closed ring. Between each multiplexed PE block (MuxPE with 16 bit register) one of the seven 2 to 1 multiplexors $S_1...S_7$ switches the data channel in order to the required horizontal image resolution in discrete blocks of 32 pixels. For the given example the closed switch S_5 provides the shifting of 160 pixels at all within the ring register. At maximum resolution (240 pixels) all switches have to be at position **o**.

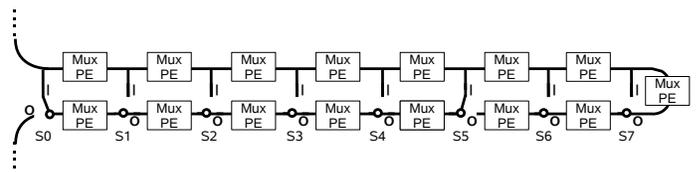


Fig. 4. ROI switching scheme

C. Control Unit

The basic function of the control unit is to generate and distribute control signals to drive the computing units (core and pixel counter arrays). To provide the basic morphological operations (e. g. edge detection or open-close loops) a standard instruction set is implemented. It can be accessed by loading single instruction words at the peripheral JTAG or SPI master controller. To perform more complex instruction sequences (e. g. the skeleton algorithm [12]) the control unit is free programmable by transmitting operation sequences via the JTAG or SPI interface. To store such a micro program a 512 byte register file is a fixed part of the control unit. The control unit is designed as a state-machine represented in Figure 5, triggered by the input image protocol signal *frame_valid*.

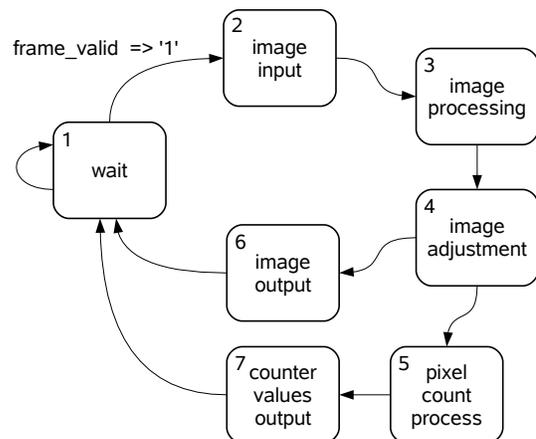


Fig. 5. Simplified state machine of the control unit

A new image macro cycle starts when the image enable signal goes to logic high level. The state machine leaves the wait status (1) and enables the core logic to store the image data (input mode, 2). After the read in the control unit switches to the processing mode (3), where the desired micro programs are executed to manipulate the image. A mechanism to adjust horizontal data offsets

(4) evoked by the processing tasks is provided before the image is put out or is committed to the pixel counters. All image processing macros can be supplemented with the pixel counting process to get the image projections (5). Each image macro step is finished either by the image (6) or the counter data output process (7).

D. Pixel counters

The pixel counter arrays are designed as parallel integer incrementers where each of it counts the pixels of one image row (vertical counter) and accordingly one image line (horizontal counter). All counters are enabled when the result image pixel has the logical value '1'.

To count in a diagonal fashion a scheme shown in Figure 6 is applied. The pixel field is shifted alternately in vertical and horizontal direction on each clock cycle. The horizontal count enable signal c_{hor_ena} controls the horizontal counters whereas its inverted logical value c_{vert_ena} controls the vertical counters. Hence one complete count process requires $2(N_x + N_y)$ steps where N_x is the horizontal and N_y the vertical number of pixels.

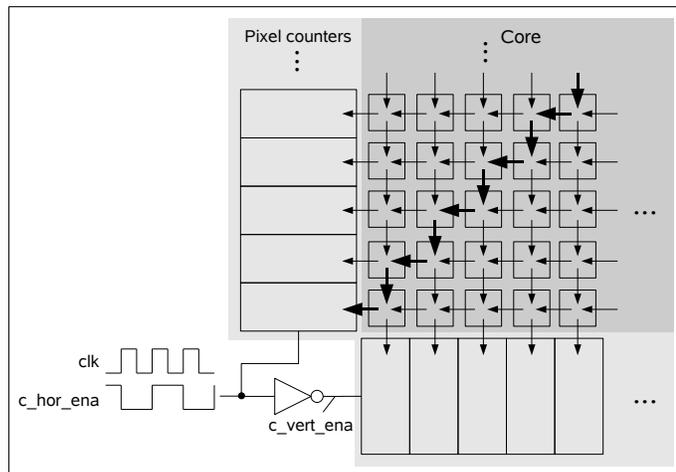


Fig. 6. Pixel counting scheme

III. DESIGN PARAMETERS AND CHIP LAYOUT

Table I shows the primary design parameters of the ASIP chip. The chip size of about 5 mm x 5 mm is the basic constraint, which affects the given image resolution and the number of pixels in a serially processed cluster (M , multiplex coefficient). For our chip we have to balance between an feasible image resolution, the given timing constraints, the limited chip area and the die costs because for prototyping purposes the fab supported only quadratic tiles of exactly 25 mm² as smallest area unit. Hence there were some difficulties to place the design onto the given chip die area shown in Figure 7. The main modules are located like Figure 2 demonstrates it.

We point out, that the core logic cells occupy more then 98 % of the available module area. The logic signal pads of the core limited design are located on the left chip side.

All remaining connections (at the top and bottom side) are at most core supply voltage pads.

Parameter	Value
Die size	
Die width	5 mm
Die length	5 mm
Bonding pad pitch	$\geq 64 \mu\text{m}$
Process node	UMC 0.18 μm 6m MIM
Number of pads	
IOs	54
Power supply	54
Power supply voltage	1.8 V (core), 3.3 V (IO)
Power consumption (only core, simulated)	≈ 1 W
Programming interfaces	JTAG, SPI
Number of processor elements	4800
Pixel multiplex factor	16
Clock rate	40 MHz
Image resolution (max.)	320 x 240

TABLE I
PRIMARY DESIGN PARAMETERS

Due to the lack of vertical space these pads could not be placed in a regular orientation. Unfortunately we could not use a standard package offered by the fab to enclosure the circuit because too many bond pads are located on the left circuit edge. To bond and cover the chip we access a project's co-partner specialized in electronic packaging. We prefer a COB (chip on board) technology, where the chip is directly fixed and bonded onto the base circuit board, together with the sensor chip die.

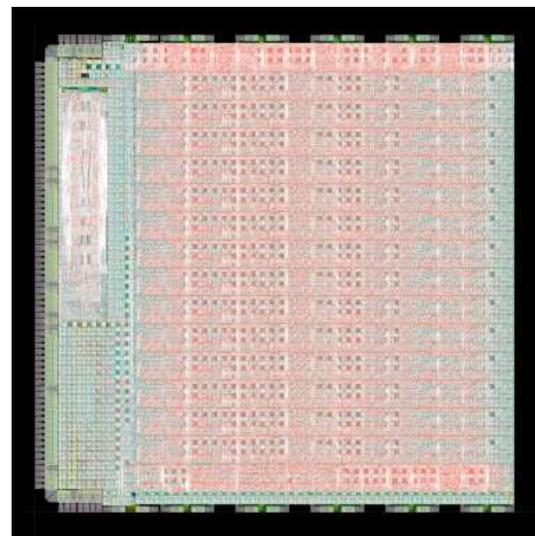


Fig. 7. Image processor chip (GDSII view)

The achieved processing time per image reflects the characteristics of our massive parallel architecture. Instancing

In state (8) some early image preprocessing like shading correction can be done and the pixel data stream will be further passed through to our ASIP architecture which is represented by compound state (9) in Figure 8. Finally a decision has to be made in state (10) whether the sample under test passes the predefined criteria or not.

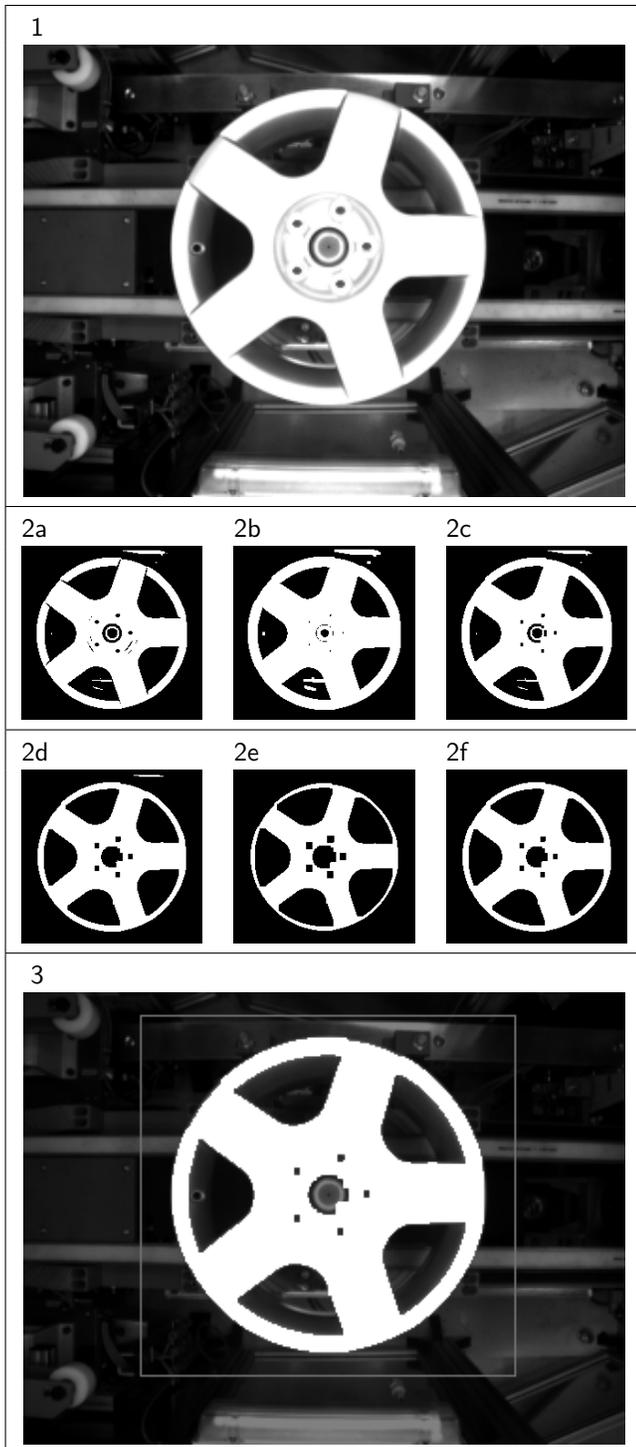


TABLE III

APPLICATION EXAMPLE (IMAGE 1A BY COURTESY OF V&C GMBH)

The solution will usually be presented as a logic PLC output value to achieve a high processing speed. Any possible severe error will be recognized and reported, causing the system to enter state (4) before it is doing some attempts to soft restart the system represented through a gray arrow pointing towards state (1). After some relaxing time the system is ready again to await the next external event.

Considering the entire system as a black box, the main overall system performance is mainly defined by the delay between a trigger input and the generated trigger output illustrated by black dotted arrow lines entering and leaving the image processing system. Table III gives a typical application example, how the functionality of our prototype chip can be used to inspect a rim. The example task shown here is to find out the positions of the five mounting holes (e. g. to automatically carry the rim to another production line in a save way). The presented images are the result of a chip simulation at system level.

The built-in ROI functionality of the sensor crops the image to the required size (sub-figure 2a). After the binarization with a fixed threshold some disturbances (holes in the rim shape, obsolete pixels on the background) have to be removed. The processor is able to receive the image with the same size in x and y dimension like the sensor has captured it. The result of a first erosion operation is sub-figure 2b; in 2c, 2d and 2e the results of three dilate operations are shown. Two erosion operations shrink the foreground object to its origin size (2f and 3). In addition sub-figure 3 illustrates an overlay of the raw image (1) with the computed image.

V. CONCLUSIONS AND OUTLOOK

In machine vision object detection and classification is a common application, e. g. to inspect automated product pipelines. The algorithms to segment images, to identify certain objects out from a set of objects known in advance, and to detect their position and their orientation within few milliseconds with cheap hardware is both an economically important and technically challenging task. To meet that we designed a massively parallel programmable ASIP processor chip which is suited for the integration in small embedded vision systems fulfilling real-time tasks. We solved this by a micro-programmable parallel on-chip architecture which allows e. g. the programming of fast image segmentation operations. This programmable structure is supported by additional counter resources to extract certain features like the moments of zeroth, first and second order to compute rapidly, area or centroid resp. orientation of detected objects.

This work shall highlight hardware implemented preprocessing methods to avoid too much redundant image information, which otherwise has to pass through mostly too expensive and often too slow standard PC-based com-

ponents. That way is possible since industrial scenes are often clearly defined in illumination and the constitution of objects. So we can reduce images to their essential information, at a fraction of the origin data amount. A two-dimensional gray scale image representation is mostly not needed anymore. The data reduction process begins at the image binarization. By determining the image projections only a quarter of the binary image data amount has to be submitted to the further computation process. The effect is a crucial reduction of the data transmission and computation times. The image resolution (320 × 240 pixels) of our prototyping chip is a consequence of the limited silicon area. The pixel resolution of later manufactured chips may be larger without that constraint. The generic design description would support this.

Further works in progress engage with bio-inspired algorithms [13] to determine objects centroids more efficiently, more precisely and even faster as the methods described above.

ACKNOWLEDGMENTS

This work presents results of the "PIXSTACK" project which is funded by the local government of Thuringia, Ministry of economics, technology and work (TMWTA).

REFERENCES

- [1] R. Mosqueron, J. Dubois, and M. Paindavoine, "High-speed smart camera with high resolution," *EURASIP J. Embedded Syst.*, vol. 2007, no. 1, pp. 23–23, 2007.
- [2] U. Muehlmann, M. Ribo, P. Lang, and A. Pinz, "A New High Speed CMOS Camera for Real-Time Tracking Applications," in *IEEE International Conference on Robotics and Automation*, 2004, pp. 5195–5200. [Online]. Available: Muehlmann2004.pdf
- [3] A. Lopich and P. Dudek, "ASPA: Focal Plane digital processor array with asynchronous processing capabilities," in *ISCAS*, 2008, pp. 1592–1595.
- [4] W. Wolf, B. Ozer, and T. Lv, "Smart Cameras as Embedded Systems," *Computer*, vol. 35, no. 9, pp. 48–53, 2002.
- [5] M. Wirthlin and B. Hutchings, "A Dynamic Instruction Set Computer," in *IEEE Symposium on FPGAs for Custom Computing Machines*, P. Athanas and K. L. Pocek, Eds. Los Alamitos, CA: IEEE Computer Society Press, 1995, pp. 99–107. [Online]. Available: citeseer.ist.psu.edu/wirthlin95dynamic.html
- [6] A. Loos, M. Schmidt, A. Graupner, D. Fey, and R. Schüffny, "A combined space-time multiplex architecture for a stacked smart sensor chip," in *Micro-Optics, VCSELs, and Photonic Interconnects II: Fabrication, Packaging, and Integration*. Edited by Thienpont, Hugo; Taghizadeh, Mohammad R.; Van Daele, Peter; Mohr, Jürgen. *Proceedings of the SPIE, Volume 6185, pp. 61850H (2006)*, ser. Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference, vol. 6185, Apr. 2006, pp. H1–H9.
- [7] G. Linan, A. Rodriguez-Vazquez, R. Carmona, F. Jimenez-Garrido, S. Espejo, and R. Dominguez-Castro, "A 1000 FPS at 128 x 128 vision processor UIT 8-Bit digitized I/O," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 7, pp. 1044–1055, 2004.
- [8] S. C. P. Dudek, "A General-Purpose 128x128 SIMD processor array with integrated image sensor," *Electronics Letters*, vol. 42, no. 12, pp. 678–679, 2006.
- [9] C.-C. Cheng, C.-H. Lin, C.-T. Li, S. C. Chang, and L.-G. Chen, "iVisual: an intelligent visual sensor SoC with 2790 fps CMOS image sensor and 205 GOPS/W vision processor," in *DAC '08: Proceedings of the 45th annual conference on Design automation*. New York, NY, USA: ACM, 2008, pp. 90–95.
- [10] A. Abbo, R. Kleihorst, V. Choudhary, L. Sevat, P. Wielage, S. Mouy, and M. Heijligers, "XETAL-II: A 107 GOPS, 600mW Massively-Parallel Processor for Video Scene Analysis," in *IEEE International Conference Solid-State Circuits Conference, 2007. ISSCC 2007*, 2007, pp. 270–602.
- [11] Anon., *IEEE Standard Test Access Port and Boundary Scan Architecture*. IEEE, 1993, ch. IEEE Std. 1149.1a.
- [12] R. Stefanelli and A. Rosenfeld, "Some Parallel Thinning Algorithms for Digital Pictures," *J. ACM*, vol. 18, no. 2, pp. 255–264, 1971.
- [13] D. Fey, M. Komann, F. Schurz, and A. Loos, "An Organic Computing architecture for visual microprocessors based on Marching Pixels," in *ISCAS*. IEEE, 2007, pp. 2686–2689. [Online]. Available: <http://dblp.uni-trier.de/db/conf/iscas/iscas2007.html#FeyKSL07>

Emerging Description Language Capabilities Matching Arithmetic Hardware Trends

Alex Zamfirescu
ASC

Palo Alto, California, USA
alex.zamfirescu@gmail.com

Abstract— Freeing the constraints that the languages meant to be used for software development and those used to describe hardware should be similar with respect to arithmetic capabilities, this paper attempts to motivate mixing dynamically changeable formats, in high level description languages. A perspective on the most often used computer formats, precedes the presentation of few arithmetic design trends, which leads to the creation of a map from desirable capabilities into HDL requirements. A few encouraging steps are then exemplified.

I. INTRODUCTION

One of the first options faced by the designer of a digital system including arithmetic computations is whether that system should be implemented in hardware or software. For many years the software implementation was more attractive, mainly due to rapid progress of compiler technology, the mass-production of microprocessor devices and the availability of a good number of software engineers. Demand for increased DSP computational throughput and low power considerations lead to gradual migration from the general-purpose processors to specialized programmable DSP devices, with the alternative to a microprocessor implementation being a custom digital hardware. The hardware approach improves speed and reduces the power consumed, but still suffers from the lack of mature high-level design tools. Development of expensive tools becomes more attractive when there is a better understanding of the problems they address, but the promise of fostering innovation and avoiding building commodities is preserved. This motivated the writing of this paper, which initially attempted the following:

- Identify major trends in the design of custom hardware for arithmetic algorithm implementation;
- Map those trends into desired capabilities for the methodologies based on popular hardware description languages (HDLs);
- Extract requirements for the HDL extensions,
- Discuss the first known steps in the right direction, made by standard organizations or private efforts.

Additionally, an attempt was made to clarify why the arithmetic support mandated for classic programming languages, and that required for hardware description languages (HDL) supporting system design and computer arithmetic optimisations is and should remain different. The first kind of support is driven by the need to hide the hardware

(platform) details from the language user, ensure portability, etc., while an HDL is best when it enables *convenient control and refinement* of the hardware details. On the other hand, poor HDL design equates software development language arithmetic requirements to those for languages describing hardware performing arithmetic computations. While this was less visible in the past, the massive recent progress in hardware capabilities, the trend to start the design above RTL, and the pressure to early-optimize designs for performance (including power), makes mandatory the clear understanding of the difference, and its manifestation in modern HDLs. Adding to that is the need to conveniently describe, and many times refine, re-configurations of arithmetic computations, graceful degradations, redundancy in computations, all enabled by the dramatic recent progress in re-configurable devices.

Early concerns were only about integer numeric types for synthesis [4]. Those were followed by the need to address floating and fixed-point types [1]. The possibility of run-time dynamic HDL fixed and floating point types, based on unified (across HDL languages) type descriptors, was flagged in a report in early 2006 [5]. A study of HDL arithmetic capabilities as they appeared in VHDL, Verilog, and ELLA was published in [2]. Today there are good ISO/IEC standards that describe the software-driven language requirements for arithmetic support [6]-[10]. An implementation of variable precision hardware building blocks described in HDL appears in [11].

The structure of this paper is as follows. A perspective on the most often used computer numeric values, and some widely impacting basic properties (section II), is followed by the presentation of few arithmetic design trends (section III), and their map into HDL requirements (section IV), section V covers a few encouraging steps already taken using examples. Conclusions and some ramifications are summarized in section VI. Mathematical notations are only used to avoid lengthy repetitions.

II. GENERAL OVERVIEW OF COMPUTER NUMERIC VALUES

Digital computer arithmetic deals with manipulating numeric values encoded into strings of bits, and belonging to particular subsets of \mathbb{Z} or \mathbb{R} , i.e. integer or real values. Of particular interest are integer *unsigned* values, integer *signed* values, *fixed-point* values and *floating-point* values. All

numeric values are in fact real¹ values, and the boundaries of the subsets are determined by the length of the string of bits used for encoding, and by the representation scheme. Using a bit string of length n , at most 2^n different values can be encoded, but the values may be either uniformly (equidistantly) or non-uniformly distributed. Throughout the paper notations that are closer to those used in the ISO/IEC standards [6-10] will be used. Some newer ones will be introduced. Five families of sets are first introduced, and summarized in table I. Then a few interesting properties are enumerated.

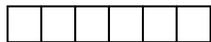
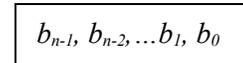
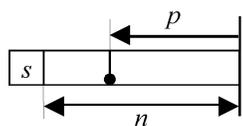
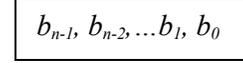
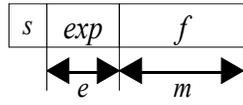
A. Families of Sets Containing Numeric Computer Values

Given a positive integer n , the sets \mathbb{U}_n and \mathbb{S}_n will contain all unsigned and signed values respectively, which can be encoded with n bits using positional binary and two's complement encoding respectively. Given $p \in \mathbb{Z}$, $\mathbb{Q}_{n,p}$ will denote the set of all fixed-point numbers which can be represented using a bit string of size n , where parameter p represents the displacement of the binary point from the LSB.

When sign-magnitude encoding is used one bit encodes the sign, while with two's complement the conventions are the same as for \mathbb{S}_n , and the value is multiplied by 2^{-p} . When two's complement representation is used for the fixed-point values we will denote their set by ${}^{2's}\mathbb{Q}_{n,p}$. Finally, the set $\mathbb{F}_{m,e}$ contains all floating-point number values, which can be encoded, using the binary representation mandated by IEEE Std. 754/854 [13] conventions, where m is the number of mantissa bits (significant field size), and e is the number of exponent bits.

Parameterised sets $\mathbb{U}_n, \mathbb{S}_n, \mathbb{Q}_{n,p}, {}^{2's}\mathbb{Q}_{n,p}, {}^b\mathbb{F}_{m,e}$ form families indexed by positive integers (n, m, e) , integer p , and the set $\{2, 10\}$ for b . The union of each such family will be denoted without indexes as $\mathbb{U}, \mathbb{S}, \mathbb{Q}, {}^{2's}\mathbb{Q}, {}^b\mathbb{F}$ or $[{}^b\mathbb{F}]$. Let the union of all these sets be \mathbb{VP} . When the size of the bit string available (or used) to encode the \mathbb{VP} elements is limited to n , the set will be denoted as \mathbb{VP}_n . Note that it could be shown that \mathbb{VP} includes \mathbb{R} . However, \mathbb{VP}_n is just a subset of the rational numbers².

TABLE I
COMPUTER ARITHMETIC SETS OF REAL NUMBERS

Five Set Families Forming \mathbb{VP}			
	Set Name	Bit Encoding	Value
\mathbb{U}_n	Unsigned	$2^{n-1} \dots 2^1 2^0$ 	$\sum_{i=0}^{n-1} b_i 2^i$
\mathbb{S}_n	Signed	$b_{n-1}, b_{n-2}, \dots, b_1, b_0$ 	$-b_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$
$\mathbb{Q}_{n,p}$	Sign-magnitude fixed-point		$\frac{\sum_{i=0}^{n-1} b_i 2^i}{2^p}$
${}^{2's}\mathbb{Q}_{n,p}$	Two's complement fixed-point	$b_{n-1}, b_{n-2}, \dots, b_1, b_0$ 	$(-b_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i) / 2^p$
${}^b\mathbb{F}_{m,e}$ $b \in \{2, 10\}$	Base b floating-point		$(-1)^s \frac{1}{b^e} \cdot f \cdot b^{exp}$
$[{}^b\mathbb{F}_{m,e}]$	Floating-point including special conditions	${}^b\mathbb{F}_{m,e} \cup \{-\infty, -0, +\infty, NAN\}$	

B. Some Computer Numeric Value Set Properties

There are a few obvious, properties involving the sets just defined. The most important ones are categorised and listed below.

1) *Same Family Set Inclusions:*

$$\begin{aligned} \mathbb{U}_n &\subset \mathbb{U}_{n+1} \\ \mathbb{S}_n &\subset \mathbb{S}_{n+1} \\ \mathbb{Q}_{n,p} &\subset \mathbb{Q}_{n+1,p+1} \\ {}^b\mathbb{F}_{m,e} &\subset {}^b\mathbb{F}_{m+1,e} \\ {}^b\mathbb{F}_{m,e} &\subset {}^b\mathbb{F}_{m,e+1} \end{aligned}$$

2) *Mixed Family Relations:*

$$\begin{aligned} \mathbb{U}_n &\subset \mathbb{S}_{n+1} \\ \mathbb{S}_n &= \mathbb{Q}_{n,0} \\ 0 &\in \mathbb{U} \cap \mathbb{S} \cap \mathbb{Q} \cap {}^{2's}\mathbb{Q} \cap \mathbb{F} \end{aligned}$$

3) *Fixed Point Sets Including Floating Point Sets:* The floating-point formats provide for a much wider dynamic range than fixed-point formats. Therefore, it is not expected

¹ These values are, restricted to be only rational numbers $\{p/q : p, q \in \mathbb{Z}\}$, for which q is a power of the base, but to simplify notation, we will refer to those rational numbers as just real numbers, especially when the fraction form is not relevant.

²A wider subset of the rational numbers is $\mathbb{VP}_n / (2^k - 1)$, and that is because any rational number can be written as $N / (2^p (2^k - 1))$ for some integers N, p, k .

that for comparative sizes any set from the \mathbb{F} family would be included in one from the \mathbb{Q} family. However, it is interesting to know what are the smallest numbers n and p for which ${}^2\mathbb{F}_{m,e} \subset \mathbb{Q}_{n,p}$, and whether they always exist. The following proposition gives the answer.

Proposition 1: For any pair (m, e) of positive integers, there is a pair (n, p) of integers, so that ${}^2\mathbb{F}_{m,e} \subset \mathbb{Q}_{n,p}$.

One simple proof is to find the smallest values satisfying the inclusion relation.

Those are:

$$n_{\min} = 1 + e \max + \text{abs}(e \min) + m$$

$$p_{\min} = \text{abs}(e \min) + m$$

where $e \max$ and $e \min$ are the maximum and the minimum values of the exponent of ${}^2\mathbb{F}_{m,e}$.

Example 1: Given ${}^2\mathbb{F}_{48,9}$, we find that $e \max = 255$

and $e \min = -254$, then $n_{\min} = 1 + 255 + 254 + 48 = 558$

and $p_{\min} = 255 + 48 = 303$.

Therefore, ${}^2\mathbb{F}_{48,9} \subset \mathbb{Q}_{558,303}$.

4) *Arithmetic Operations Closure:* Defining arithmetic operations $\circ \in \{+, -, *, /\}$ for sets \mathbb{X} with $\mathbb{X} \in \{\mathbb{U}_n, \mathbb{S}_n, \mathbb{Q}_{n,p}, {}^{2^s}\mathbb{Q}_{n,p}, {}^b\mathbb{F}_{m,e}, [{}^b\mathbb{F}_{m,e}]\}$ requires either that the result be considered in another set from the same family, the reduction of the domain to less than \mathbb{X}^2 , or the use of special extra artificial elements. For example, the $+$ operator can be evaluated for all pairs from \mathbb{U}_n^2 only if the result is considered in \mathbb{U}_{n+1} . This makes the correct definition for the addition in \mathbb{U}_n to be something like $+: \mathbb{U}_n^2 \rightarrow \mathbb{U}_{n+1}$. The set $\mathbb{U}_{n+1} - (+(\mathbb{U}_n^2))^3$ is in general mapped back into \mathbb{U}_n using either a convenient correspondence, or the mapping is left to be consequence of the hardware implementation. What was exemplified here for \mathbb{U}_n similarly occurs for all arithmetic operations and all sets \mathbb{X} . There are three cases to be considered for $\circ: \mathbb{X}^2 \rightarrow \mathbb{VP}$.

1. Result belongs to \mathbb{X} .

2. Result is between two elements of \mathbb{X} . In this case a rounding scheme has to be specified⁴, which designates the choice for one element to be the result of that specific application of the operation.

3. Result is not between two elements of \mathbb{X} . Special rules are specified about what is the result. This cases deal with pairs for which the operation is not defined, or it is outside of the range of \mathbb{X} because \mathbb{X} does not contain $\pm \infty$.

This is the semantic when the families (as categorized in Table I) are not mixed, and the best effort is made to provide a result in the set both operands belong to. While this could be a desideratum for some fixed type programming languages, simple examples show that with hardware which is performing computer arithmetic this is not always the case, result getting frequently outside of the initial operand set. Indeed, starting with the simple extra carry bit for addition, one can continue to imagine how more complicated operations on the significant field of floating point numbers place a partial result into a fixed point register (before normalization). Section III below identifies more cases where inter-mixing the sets is highly desirable.

Note: Such cross family operators become much simpler when executed directly by the hardware, compared to emulating/modelling that, from within current software-driven programming languages. The emulation task is not impossible, but it is tedious, and that's why the modern hardware description languages are moving towards direct specification of cross family operators. This is something needed for the progress of arithmetic computation, and maybe its evolution from the state where "software is slow-dancing in the rhythm of hardware", to the more desirable state where the "hardware generation is driven by the requirements enabling progress in computer arithmetic."

5) *Generalized Rounding:* One key new ingredient for this is the capability to work with operations $\circ \in \{+, -, *, /\}$ defined as $\mathbb{X} \times \mathbb{Y} \rightarrow \mathbb{V}$, where the three sets $\mathbb{X}, \mathbb{Y}, \mathbb{V}$ could be different, either from same family, but having other precision settings (sizes of the fields), or they could be sets from different families. An obvious requirement for, commutative operators, is that for any a and b , belonging to both \mathbb{X} and \mathbb{Y} , $a \circ b = b \circ a$. When the exact result is not in \mathbb{V} , its rounding should occur based on the same three rules specified in the previous paragraph, as they apply to set \mathbb{Z} . This generalized rounding should also be used as a basis for the (implicit or explicit) conversion between elements from the different sets. That way any particular real number could be "landed" in any particular set, meaning that any number from \mathbb{R} can be stored – exactly, rounded, or "re-routed" (case 3 above) into any format. Note that the association of a particular rounding scheme could be done at the lowest granularity that is the operation level.

This global overview of the unsigned values, integer signed values, fixed-point values and floating-point values, and the implications that mixing their sets in hardware operations is mathematically sound, feasible and comes hardware-natural,

³ The '-' sign represents here the set difference, while '+' denotes the name of the addition operator (as a function).

⁴ Like those classic and well known, to nearest, towards infinity, truncate, etc. The particular rounding scheme is not relevant for this discussion.

could mean nothing, without more compelling examples of why a language describing hardware should do that too.

III. MAJOR TRENDS

This section presents some actual trends in designing arithmetic hardware, and how the sets introduced in section II are mixable.

A. Exact Dot Product Hardware Solutions

The computation of dot products like

$$x \cdot y = \sum_{i=0}^n x_i \cdot y_i$$

plays an essential role in DSP, scientific computations, and in verification numerics.

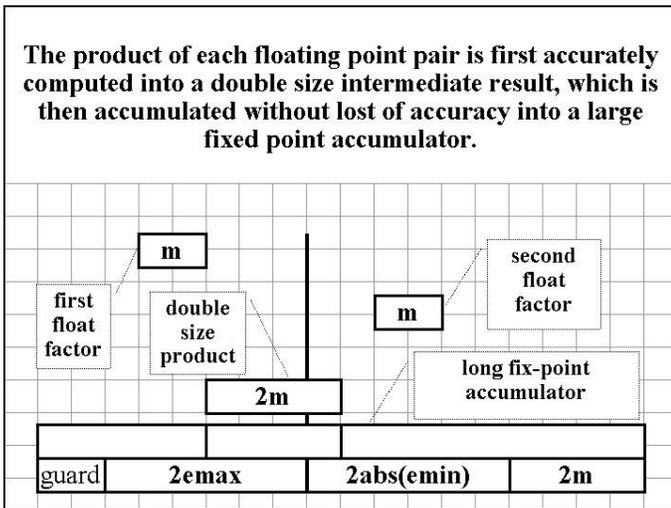


Fig. 1 Accurately computing the dot product.

Most DSP processors come with a special *multiply-accumulate instruction* to help accelerate the frequent computation of the dot product. Many scientific computations involving Hilbert spaces, norms, or matrix-based optimisations are also computing most of the time dot products. It is surprising that, the current general-purpose computers do not have yet a built in mode to help avoid errors that often occur when large $(x_i \cdot y_i)$ product pairs cancel after the contribution of a smaller value product was truncated.

A powerful algorithm for this task is based on a long accumulator [3]. The accumulator (see Fig. 1) is a fixed-point accumulator, which satisfies proposition 1, (i.e. it is wide enough to fit any double size floating point accurate product). A few extra guard bits are added to avoid overflow after the fixed-point addition. At the end of the accumulation phase the fixed-point accumulator is guaranteed to contain the exact dot product. If normalization occurs and the fixed-point accumulator content is read into a float some rounding may occur, but still the floating-point result is guaranteed to be the most accurate possible given the result format restriction.

Example 2: Example 1 from section II computed that ${}^2\mathbb{F}_{48,9} \subset \mathbb{Q}_{558,303}$. That means that dot product on vectors

from the space ${}^2\mathbb{F}_{24,8}^N$ can be accurately computed using a $\mathbb{Q}_{558+guard,303}$ fixed-point accumulator register, if $N < 2^{guard}$, and one or more $*$: ${}^2\mathbb{F}_{24,8} \times {}^2\mathbb{F}_{24,8} \rightarrow {}^2\mathbb{F}_{48,9}$ multipliers possibly working in parallel.

During the accumulation phase some long carry chains may occur. One technique is to use some hardware “all one” flags and efficiently jump the carry. The experimentation with this technique and/or other such optimisations has to be specified in the same HDL where the hardware simulation is performed. By being able to mix the sets, the designer and experimenter would be able to concentrate on just increasing performance.

B. Predictable Accuracy for Computed Functions

When the precision of the numeric value is fixed it is relatively easy to provide precise function results, because the optimisation is done once only, on known sizes. The support for variable precision brings the challenge to provide predictable accuracy for the computed functions. This can be done by selecting algorithms that are known to converge to a specified accuracy in a number of steps like the CORDIC, or by estimating for each result an upper bound and a lower bound.

C. Support for Interval Arithmetic

A strong trend towards accurate arithmetic computations is building around interval arithmetic. This is involving full bounded intervals of real numbers, instead of single values. Remarkably, intervals of real numbers can be defined precisely with the sparse sets of computer arithmetic values. Statements about numbers that are not representable can be made based on arithmetic executed on the intervals containing them, which are bounded by representable numbers. The semantic of the operation on intervals is that the result interval contains all the possible results for one operand from the first interval and the second operand from the second interval.⁵ However, the interval arithmetic has much wider ramifications in scientific computing, verification of crisp (one value) arithmetic, and proof of critical systems. Each interval operation ends up to be split in a set of cases, and in each case specific operations with rounding-up, or rounding-down are performed. The support for convenient rounding specifiable at the level of operation is therefore a requirement, which could enable the implementations of efficient interval arithmetic machines. The usual exceptions of floating-point arithmetic like underflow, overflow, division by zero, or invalid operation do not occur in interval arithmetic. While this arithmetic is not applicable to all problems, it is becoming more important, and a request to start a study group for a standard dealing with interval arithmetic was already submitted to the IEEE.

Modern computer arithmetic has also extended the precision of interval arithmetic by considering intervals of the form

⁵ EDA engineers are familiar with such techniques from the timing calculators.

$$\mathbf{x} = \sum_{i=1}^n x_i + (\epsilon_1, \epsilon_2)$$

where $x_i, \epsilon_1, \epsilon_2 \in {}^2\mathbb{F}_{m,e}$, \mathbf{x} (bold x) is an interval with boundaries $\epsilon_1 + \sum_{i=1}^n x_i$ and $\epsilon_2 + \sum_{i=1}^n x_i$, and the significants of all the x_i , and ϵ_i are non-overlapping if accumulated to an appropriate element from $\mathbb{Q}_{n,p}$. Computations with such intervals involve mixing floating point and fixed-point accumulators. An interesting generalization of the normalization operation would bring back the sum of float representation from a long accumulator. This operation would look for the leading one extract a mantissa of size m and build a corresponding floating point. An HDL useful procedure would take an element a of $\mathbb{Q}_{n,p}$, and return the first floating point x and $a - x$. The sizes of x could either be passed in or inferred directly from the sizes of a using again proposition 1. Note that the operation $a - x$ is a cross-family operation⁶. With availability of mixed operations the best choice for the hardware implementation of this kind of precise computation could be again well optimised, to take best advantage of the available hardware resources.

D. Migration between Floating and Fix Point Solutions

We can identify two phases in the ESL design. In a first phase designers concern is only with the numeric values, with the verification of the concept, and creation of a golden model. In a second phase focus is on the register sizes, and encoding, and tradeoffs between accuracy (quantisation errors, overflows etc.) and performance, area, and power are performed. Current HDLs do not support the two-phase approach to ESL design; users are forced to jump between software programs, which are designed for scientific computations and HDLs, or between custom C and HDLs. A particularly interesting phase, which is the transition of the design from floating-point to, fixed-point format, and more recently the decision about what format to use continue the implementation should be feasible just by changing the descriptors of the types within a single HDL, and not by forcing the transition of the whole design into another environment. Many times designers like to keep parts of a non-transited design running in one format and use the live stimulus to exercise the changed part. The recognised trend is to use more floating-point units and that adds to the decision complexity.

E. Design Exploration of Multiple Word Length Paradigm

In DSP when fixed-point is used there is a need to optimise the sizes of the registers, to avoid using larger than needed registers on one hand and to avoid unpleasant quantisation errors on the other. While this is done easily for

⁶ The operation is just a reset of a particular field. Another mode would just return an integer pointing to the last visited bit.

linear systems, by using analytical approaches and some heuristics, the non-linear case (much more often) encountered, has to be solved using also simulation. The current simulations start with a fixed size registers. That's why the optimisation loop included re-elaboration of the simulator, and in many cases a designer in the loop. The automatic, tight loop is feasible only if changes to formats are made at run time.

F. Synthesis with Low Power Constraints

During low power behavioural synthesis optimisation changes are made to a computation graph, to explore the design space. Those changes include size of the operands changes, value encoding, and operator binding. In order to cover a large design space decisions for the next best move are taken based on an estimation of a cost function, which takes into consideration area, timing, and dynamic and static power. To determine the timing and the dynamic power the activity is simulated and statistics are extracted. The time to execute classic re-elaboration of static objects for each small incremental change or exploratory move is prohibitive. A good solution is to provide the functionality in the HDL, which will allow changes of the object characteristics during the same elaborated simulation.

G. Base 10 Arithmetic

Recent IEEE Std. 754R include support for decimal floating point. With the known benefits of decimal arithmetic out of the question, the trend is that more devices supporting decimal arithmetic will be built. HDLs used to design and verify such systems are the first to provide the decimal arithmetic capabilities.

IV. REQUIREMENTS FOR HARDWARE DESCRIPTION LANGUAGES

To each trend discussed in section III, a set of desired HDL capabilities were associated. The rationale was that if the capability were provided in the HDL, more progress would be achieved in the direction of the trend. This is essential given the to fulfil the commercial requirement for the viability of the HDLs and the tools based on them. A clear requirement was then established for each capability. The summary of this exercise is provided in Table III.

TABLE III
ESSENTIAL HDL ARITHMETIC REQUIREMENTS

Modern Design Trends Reflected into HDL Requirements			
	Design Trend	HDL Desired Capability	Requirement
A	Design Exact Dot Product Computation in Hardware	Multiply pairs of floating point numbers into a double sized float, accumulate into a large fixed-point register, and use	Mixing different sizes and different kinds (fixed or floating point) of objects in operations and assignments, while observing the numeric value semantic.

		just one final rounding.	
B	<u>Predictable Functions</u>	Predictable accuracy for all precision ranges.	Provide HDL <i>predictable and accurate solutions</i> for all common algebraic and transcendental functions (i.e. sqrt, sin, cos, tan, log, sh, ch, etc.)
C	<u>Implement Interval Arithmetic Processors</u>	Operator driven rounding mode.	Run time <i>easy change of rounding mode</i> (specified in a procedure call, or flagged by special operator symbol in the HDL)
D	<u>Float-Fix Migration for DSP Design</u>	Manageable change between floating and fixed-point representations.	Provide capability (i.e. a descriptor) to <i>change numerical representations</i> without any code re-write.
E	<u>Multiple Word Length Exploration</u>	Run time tight loops including simulation and format <i>size changes</i> driven by automatic measurements.	Support <i>dynamic</i> (run time) <i>format size changes</i> .
F	<u>Power Constrained Synthesis</u>	Run time <i>numerical representation changes</i> during design convergence towards minimal power.	Support <i>dynamic</i> (run time) <i>representation changes</i> .
G	<u>Decimal Arithmetic</u>	Standard decimal floating point with customisable choices.	Support <i>decimal floating-point</i> arithmetic, and accurate verification mechanisms.

V.STEPS IN THE RIGHT DIRECTION

A few good steps to enhance the HDL capabilities to handle variable precision types, and provide for dynamic, run time changes of the type descriptor were taken by both standard organizations and the private sector.

This section briefly discusses the new enhancements of the VHDL language IEEE P1076 in ballot at the time of this writing, and will use the result computed in the example 1, section I, to present how computation of the dot product in the

space with 100 dimensions ${}^2\mathbb{F}_{24,8}^{100}$ using an accumulator from $\mathbb{Q}_{558,303}$ is described and simulated using Fintronic FinSimMath by extending a popular HDL, the Verilog® HDL.

A.Enhancements to VHDL

There is good news for the VHDL supporters doing arithmetic with fixed-point and floating-point types. It appears that the proposed new draft contains capabilities to declare types representing values from ${}^2\mathbb{F}_{m,e}$ and both $\mathbb{Q}_{n,p}$ and

2 's $\mathbb{Q}_{n,p}$. The sizes should be provided before elaboration, and cannot be changed during simulation. Conversion is explicit (requires always a function), and there is a good set of functions supported. From the requirements shown in Table III, A is feasible only with explicit conversions; B is supported but the burden to bring the result close to the desired value is left to the users who have to specify (by intelligent guessing) things like the number of required iterations; C is possible; and D, E, F, G are not possible. Note that fulfilling requirement G is not out of the reach, but it will require another specialized package.

B.FinSimMath Extensions to Verilog®

FinSimMath, an extension of Verilog for Mathematical computations is described in chapter 8 of FinSim's User's Guide [11]. Those extensions satisfy all requirements A, B, C, D, E and F from the table III. G is also feasible. These requirements are satisfied for all six families from listed in Table I.

A FinSimMath tutorial [13] provides running examples in extended Verilog that show how to:

- (1) modify during the simulation the format (floating point or fixed point), as well as the number of bits used for each field of the format in the model of a low pass filter,
- (2) perform without the need of explicit conversion functions arithmetic operations with operands of type complex (Cartesian or polar) or of matrices having elements of type complex, including the computation of the inverse of such matrices,
- (3) compute the pseudo inverse of matrices,
- (4) separate data from its location using high level constructs such as the "View as" and "InitM" constructs for multi-threading processing,
- (5) perform FFT, and fast autocorrelation,
- (6) exchange FinSimMath data between modules, and
- (7) monitor special conditions such as overflow or underflow.

Example 3: The example selected here shows how the dot product in ${}^2\mathbb{F}_{24,8}^{100}$, a real space with 100 dimensions is computed accurately using a long accumulator `tmp2` which was designed based on the results of Example 2. The full listing was given here just for information and the details of the descriptor specifications are all available online in chapter 8 of [13]. First lines included in the first for loop containing the code

```
tmp1 = v1[i]*v2[i];
```

is in fact performing an operation

$$* : {}^2\mathbb{F}_{24,8} \times {}^2\mathbb{F}_{24,8} \rightarrow {}^2\mathbb{F}_{48,9}$$

and the line

```
tmp2 = tmp2 + tmp1; performs
```

$$+ : 2^F_{24,8} \times 2^s Q_{608,304} \rightarrow 2^s Q_{608,304}$$

while the assignment `v = tmp2;` which follows after the loop has an explicit conversion from a fixed point value into a floating point value. The data for the example was chosen for a case where the result without using the long accumulator is wrong. This can be seen in the results printed by the simulator.

A. Code for Example 3

```

module top;
parameter SIZE = 100;
/*format */
`define TWOS_COMPLEMENT 1
`define SIGN_MAGNITUDE 2
`define FLOATING 3

/* rounding */
`define TO_NEAREST_INTEGER_IF_TIE_TO_MINUS_INF 1
`define TO_NEAREST_INTEGER_IF_TIE_TO_PLUS_INF 2
`define TO_NEAREST_INTEGER_IF_TIE_TO_ZERO 3
`define JUST_TRUNCATE 4
`define TO_ZERO 5
`define TO_INF 6
`define TO_MINUS_INF 7
`define TO_PLUS_INF 8

/* overflow */
`define SATURATION 1
`define NORMAL 2
`define WARNING 64

(* varprec = descriptor *)
reg [0:1] d1;
(* varprec = descriptor *)
reg [0:1] d2;
(* varprec = descriptor *)
reg [0:1] d3;

(* varprec = data *)
reg [0:32] V1[SIZE-1:0];
(* varprec = data *)
reg [0:32] V2[SIZE-1:0];

(* varprec = data *)
reg [0:283] tmp1;

(* varprec = data *)
reg [0:561] tmp2;

(* varprec = data *)
reg [0:32] v;

integer i;

initial begin
    $VpSetDescriptorInfo(d1, 8, 24, `FLOATING,
        `TO_NEAREST_INTEGER_IF_TIE_
TO_MINUS_INF,
        `SATURATION+`WARNING, 1);
    $VpSetDescriptorInfo(d2, 9, 48, `FLOATING,
        `TO_NEAREST_INTEGER_IF_TIE_
TO_MINUS_INF,
        `SATURATION+`WARNING, 1);
    $VpSetDescriptorInfo(d3, 304, 304,
`TWOS_COMPLEMENT,
        `TO_NEAREST_INTEGER_IF_TIE_
TO_MINUS_INF,
        `SATURATION+`WARNING, 1);

    $VpSetDefaultOptions(8, 24, `FLOATING,
        `TO_NEAREST_INTEGER_IF_TIE_

```

```

TO_MINUS_INF,
        `SATURATION+`WARNING, 1);

    $VpAssocDescrToData(V1, d1);
    $VpAssocDescrToData(V2, d1);
    $VpAssocDescrToData(v, d1);
    $VpAssocDescrToData(tmp1, d2);
    $VpAssocDescrToData(tmp2, d3);

    $InitM(V1, (($I1 == 0) ? -11 : 1+2**(-13)));
    $InitM(V2, (($I1 == 0) ? 9 : 1+2**(-13)));
    $PrintM(V1, "%k");
    $PrintM(V2, "%k");

    tmp2 = 0;
    for (i = 0; i < SIZE; i = i + 1)
        begin
            tmp1 = V1[i]*V2[i];
            tmp2 = tmp2 + tmp1;
        end
    v = tmp2;
    $display("using temporary registers: v = %k\n",
v);
    v = 0;
    for (i = 0; i < SIZE; i = i + 1)
        begin
            v = v + V1[i]*V2[i];
        end
    $display("without using temporary registers: v =
%k\n", v);
end
endmodule

```

B. Example 3 Simulation Results

```

Simulating until no event ...
V1[ 0]=11000010.011000000000000000000000
V1[ 1]=00111111.000000000000100000000000
V1[ 2]=00111111.000000000000100000000000
...
V1[99]=00111111.000000000000100000000000
V2[ 0]=01000010.001000000000000000000000
V2[ 1]=00111111.000000000000100000000000
V2[ 2]=00111111.000000000000100000000000
...
V2[99]=00111111.000000000000100000000000

using temporary registers:
v = 00111001.100011000000011000110000

without using temporary registers:
v = 00111001.10001100000000000000000000

Ending at time 0s.

```

VI. CONCLUSIONS

The tremendous progress in computer technology should be accompanied by extension of the mathematical capacity of the computer. A balanced standard of computer arithmetic should require that the basic components of modern computing (floating-point arithmetic, interval arithmetic, and an exact dot product) should be provided by the computer's hardware. We presented how those and many other DSP and hardware arithmetic design problems do benefit from enhancements to HDLs including dynamic variable precision.

ACKNOWLEDGMENT

The author wishes to acknowledge Dr. Alec Stanculescu for the help in developing and verifying the FinSimMath code example used in this presentation.

REFERENCES

- [1] A. Zamfirescu, "Floating Point Types for Synthesis," in *Proc. of VIUF Fall 2000 Workshop*, Orlando, Florida, Oct. 2000.
- [2] A. N. Zamfirescu, co-author, J. P. Mermet, editor, *Fundamentals and Standards in Hardware Description Languages*, Kluwer Academic Publishers, Norwell, MA, 1993.
- [3] U. Kulisch, W. Miranker, "The arithmetic of the digital computer: A new approach." *SIAM Rev.*, 28(1), pp.1-40, 1986.
- [4] A. N. Zamfirescu, "Numeric Types for Synthesis," in *Proc. of the VHDL International Users Forum*, Fall 1992.
- [5] A. Zamfirescu, "Modern Numeric Capabilities in Hardware Description Languages," [Online]. Available:
http://alex.zamfirescu.googlepages.com/num_May_10_2006.pdf
- [6] Enter <fix>ISO/IEC
- [7] Enter <fix>
- [8] Enter <fix>
- [9] Enter <fix>
- [10] Enter <fix>
- [11] Enter <fix>
- [12] R. Kirchner, U. Kulisch "Hardware support for interval arithmetic," *Reliable Computing*, Vol. 12:3, pp. 225–237, 2006.
- [13] FinSim's User's Guide, Fintronic USA, [Online] Available:
<http://www.fintronic.com/manual/simug10.1.pdf>
- [14] IEEE Std. 754/854 <fix>
- [15] FinSimMath Tutorial, Fintronic USA, [Online], Available:
<http://www.fintronic.com/finmath.html>

FPGA Implementation of Harmonic Currents Identification Algorithms using Neural Networks

S. R. Dzonde⁽¹⁾⁽²⁾, H. Berviller, J-P Blondé, F. Braun
⁽¹⁾Institut d'Électronique du Solide et des Systèmes (InESS)
 ULP/CNRS - UMR 7163
 Strasbourg, France
 herve.berviller@iness.c-strasbourg.fr

C. H. Kom
⁽²⁾Laboratoire d'Electronique, Electrotechnique,
 Automatique et Télédétection (LEEAT)
 University of Douala, Cameroon

Abstract—This work presents a Neural Networks based intelligent control design used for harmonics current identification as a part of an Active Power Filter (APF) control unit. APF rule is to compensate reactive power in low voltage power systems as well as eliminate current harmonics caused by non-linear loads. For their integration on FPGA (Field Programmable Gate Array), the algorithms developed and simulated under MatLab/Simulink[®] were converted in Altera Dsp Builder[®] (ADB) interface by overcoming some incompatibilities between the two environments. The experimental results and the material resources utilization ratio confirm the interest to continue the hardware implementation process of the whole APF control unit.

I. INTRODUCTION

Nowadays, APFs are one of the most advanced solutions to suppress harmonic currents and compensate reactive power, which presence in a distribution power system bring about serious problems such as transformer overheating, machine vibration, motor failures and higher line losses, etc. In presence of a nonlinear loads (rectifiers, inverters, AC regulators, etc.), harmonics are added on the source current thus creating a disturbed current. The injection of compensation currents in the electrical power supply by means of the APF's power circuit composed of an inverter and an output filter, allows returning to the initial current's shape.

The APF control unit is split in two main blocks: the first one assumes current harmonics identification in order to produce the required reference currents for the control algorithms, while the second one carries out the control method for injecting phase-opposite these currents in the electrical power system [1].

The technique used to obtain the reference currents, will have a decisive influence on the APF efficiency and performance. In [2], an evaluation of the techniques traditionally used for this purpose is proposed. It is possible to differentiate between those which operate in the time domain and in the frequency domain. The latter case includes the algorithms based on the Fourier theory. This technique is characterized by some drawbacks: the impossibility of having precise results in transient conditions and a large memory requirement to store the samples of the last fundamental period. There is a greater diversity among the techniques operating in the time domain, including the least square error technique and the Kalman filtering [3],[4]. Beyond their simplicity they cannot easily take into account noises and

errors, and they need an incompressible time-delay to convergence. The more powerful technique is with no doubt the Instantaneous Power Theory (IPT) [2], [5], [6]. The application of this technique requires an equilibrated and balanced voltage system provided by an auxiliary block usually achieved with a Phase-locked Loop (PLL). However, the IPT is not available for a specific harmonic current compensation.

For a few years, artificial intelligence techniques were applied to control APFs. For instance, ANNs by their learning capabilities allow them to constantly adapt themselves to any changes and thus to be very efficient. Indeed, current research proposes a unified neural approach of the entire adaptive harmonics compensation system, as in [5], [6]. The proposed approach is unified in the sense that it is only based on a single type of ANN: the Adaline neural network. This approach is motivated by a need of simplicity and flexibility in ANNs-based control strategies used in electrical systems, but also to optimize the hardware resources required in digital implementation. Thus, four intelligent control structures based on Artificial Neural Networks (ANNs), adapted to various constraints were developed in [5], with full satisfactions, supplanting the traditional approaches. The work presented in this paper concerns the implementation methodology for the first block assuming current harmonics identification.

On this subject, due to the parallelism and pipelining technologies, application specific hardware implementation can offer much greater speed than software implementation using DSP or microprocessors. There are essentially two types of technologies available for hardware design: full custom hardware design also well-known as Application Specific Integrated Circuit (ASIC) and semi custom programmable devices like Field-Programmable Gate-Array (FPGA). While it can offer highest performance, the ASIC can't be reconfigured. Implementing complex algorithms on reprogrammable device minimizes the time-to-market cost, enables easy and fast circuit modification and rapid prototyping through Hardware Description Language (HDL) [7], [13]. So FPGA should be the excellent choice for real-time harmonics mitigation.

The ADB provides an integrated environment where the designer can easily generate timing-optimized RTL code based on high-level Simulink[®] design descriptions. Moreover, it can generate synthesizable FPGA HDL, allowing multi-disciplined users to work at high level of abstraction in a

common workspace. Succeeding the simulations on this interface, a FPGA implementation is planned in order to evaluate the constraints required by such implementation to obtain similar results as those generated by MatLab/Simulink®. These constraints are mainly due to some incompatibilities between this software and the FPGA development environment, i.e. ADB, and on the other hand, to the FPGA development board characteristics.

The Neuronal modeling of the Active Power Filter is related in the next section. Section III is dedicated to the methodology required by the FPGA implementation due to the different design environments and finally simulations and experimental results are presented in section IV.

II. NEURONAL MODELING of the ACTIVE POWER FILTER CONTROL UNIT

A. Adaline Neural Networks

Originally, the formal neural networks are a mathematical attempt of human brain modeling [8]. Well-known ANNs, such as Multi-Layer Perceptrons (MLP) have proved to be able to solve various problems, but due to their relatively high complexity, they are not well suited for hardware implementation. In our application, we used the ADaptive LINear Element (Adaline) which is a particular case of MLP with a very simple architecture; only one neuron with a linear activation function and a vector entry type. Its efficiency has been proved in signals estimation applications and adaptive filtering. The learning process is carried out by minimizing the system's global error. In [9], another application of ANNs can also be found in position control of a robot hand.

The general network topology of an Adaline is described on Figure 1. The estimated signal $y(t)_{est}$ of $y(t)$ can be determined by the linear relation (1) :

$$y(t)_{est} = W^T(t)X(t) \quad (1)$$

where $W^T(t)$ represents the weight vector updated during the learning process and $X(t)$ the input vector.

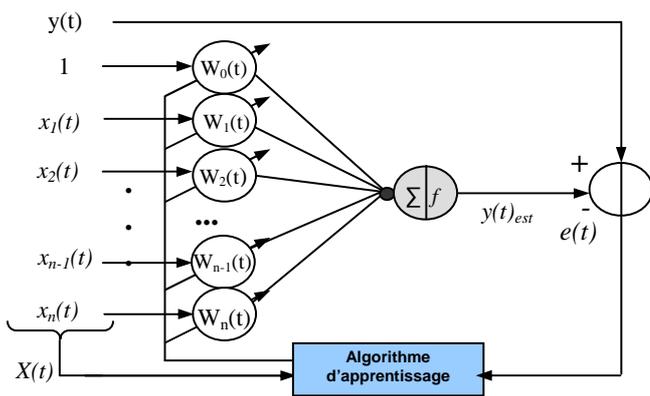


Fig. 1. Adaline topology

Adalines are trained by an online learning process based on the Δ -rule called LMS (Least Mean Squares). In the

following, we will use the Δ -rule in its simplest form by equation (2) whose convergence was proved in [10], [11].

$$W(t+1) = W(t) + \frac{\mu e(t)X(t)}{X^T(t)X(t)} \quad (2)$$

For most applications, the learning rate μ can be empirically determined without prejudice to the scientific rigor of the work.

B. APF Compensation Strategy

Figure 2 presents a system with a non-linear three-phase load and voltage supply, both simulated on MatLab/Simulink/Power System Blockset®. An APF is used to generate the compensation current. The non-linear load current i_c is the sum of the source current i_s and the compensation current i_{inj} . As shown in [12], the goal is to get a balanced supply current without harmonic. The APF control unit shows up in detail four blocks with specific functions:

- Power supply frequency estimation
- Direct voltage components extraction
- Harmonics current and reactive power identification
- The three-phase inverter control

The harmonics identification is firstly carried out with the traditional approach, then with the Adaline networks. Several control strategies were also evaluated, in particular the neuronal approach with identical simulation parameters.

The compensated signal i_s , resulting from this purely neuronal approach looks like a sine wave disturbed by a weak noise. Moreover, the on-line training of the neural network dynamically allows the compensation by following the fluctuations of the disturbances. With this approach, the measured Total Harmonic Distortion (THD) is of 0.82% instead of 8% with a traditional one.

This neuronal approach deals with:

- a variant of the IPT called *Real and Imaginary Instantaneous Powers* method based on Adaline for the identification of the harmonic currents
- the *Direct and Reverse neural control* for re-injecting these currents in the electrical power supply.

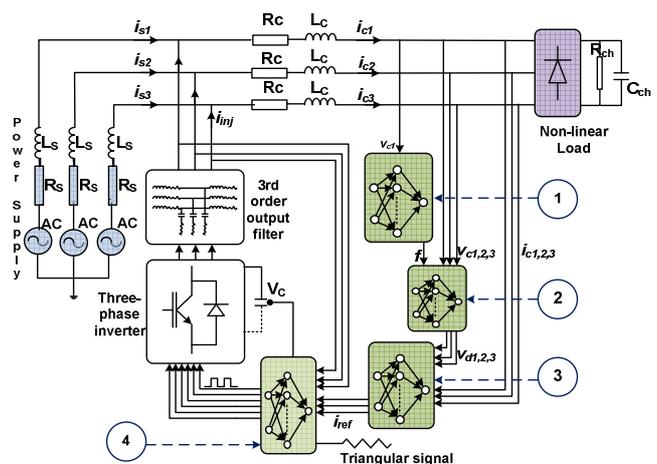


Fig. 2. General diagram of the APF compensation strategy

C. Identification method for harmonic currents

A periodic waveform can be decomposed by Fourier series into a sum of simple oscillating functions, namely sine and cosine. By applying it on the load current i_c , as in [5][12], we can obtain relation (3)

$$i_c(t) = i_{cf}(t) + i_{ch}(t) \tag{3}$$

where i_{cf} represents the fundamental current and i_{ch} the harmonic currents. Equations (4) and (5) present the details:

$$i_{cf}(t) = I_{11} \sin(\omega t - \alpha) + I_{12} \cos(\omega t - \alpha) \tag{4}$$

$$i_{ch}(t) = \sum_{n=2, \dots, N} [I_{n1} \sin(n(\omega t - \alpha)) + I_{n2} \cos(n(\omega t - \alpha))] \tag{5}$$

ω represents the fundamental frequency of the electrical power supply, α is an unspecified angle which can be equal to zero, I_{11} and I_{12} are the amplitudes of the sine and cosine components of the fundamental current, I_{n1} and I_{n2} are associated with the sine and cosine components of the n-order harmonic current. The harmonics identification is done by the same Adaline network on each phase according to the topology of figure 1. Therefore, the load current can be written in the matrix form (6):

$$i_c(t) = W^T(t) X(t) \tag{6}$$

Where: $W^T(t) = [I_{11} \ I_{12} \ I_{21} \ I_{22} \dots I_{n1} \ I_{n2}]$;

$$X(t) = \begin{bmatrix} \sin\omega t \\ \cos\omega t \\ \dots \\ \sin n\omega t \\ \cos n\omega t \end{bmatrix} \tag{7}$$

and $W(k+1) = W(k) + \mu \cdot e(k) \cdot X(k)$ (8)

By using the equation (8), a very simplified alternative of the Widrow-Hoff algorithm for weights updating, the fundamental current can be evaluated with (4) and the harmonic current by the equation (9) :

$$i_{ch}(t) = i_c(t) - i_{cf}(t) \tag{9}$$

$$= \sum_{n=2, \dots, N} [I_{n1} \sin(n\omega t) + I_{n2} \cos(n\omega t)]$$

In this way, this method allows the selective compensation of the harmonic currents.

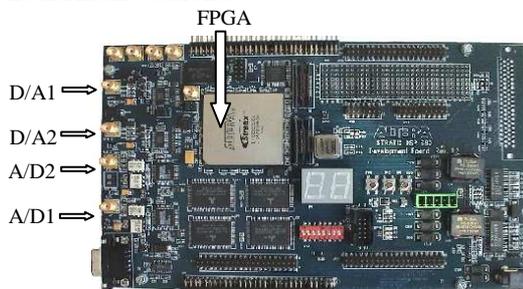


Fig. 3. The DSP development Kit

III. HARDWARE IMPLEMENTATION OF HARMONIC CURRENTS BLOCK using ANNs TECHNOLOGY

This research aims to implement an intelligent control design used for harmonic currents identification by using neural networks. Based on the Matlab/Simulink® developed design, a new version suitable for FPGA implementation is done through ADB environment. According to the reconfiguration and flexibility ability of FPGA, it can satisfy requirements for iterative learning and parallel processing of the neural networks.

The development kit of figure 3 used in our designs, integrates a Stratix® EP1S80B956C6 as FPGA, of Altera family, memories, A/D and D/A converters.

A. Compatibility Problems Between DSP Builder and Matlab-Simulink®.

During installation, ADB library is added to those of Matlab/Simulink®, and provides dedicated functional blocks (arithmetic, bus manipulation...). The SignalCompiler block, which is the heart of this interface, ensures the following functions successively, according to the Top/Down method.

- Conversion of the design into a VHDL program
- Analysis and logical synthesis
- Architectural study and RTL synthesis according to Quartus II software
- FPGA programming.

To achieve the design on ADB, it is possible that some needed functions were not available. For example, the ramp $f(t) = t$ has been realized with a Look-Up Table (LUT) associated with an "Increment Decrement" block. Moreover, sine and cosine blocks were also built with a LUT to constitute the input vector. The three-input multiplying block is carried out by cascading two two-input multipliers blocks, but this isn't without consequences on the data bus size.

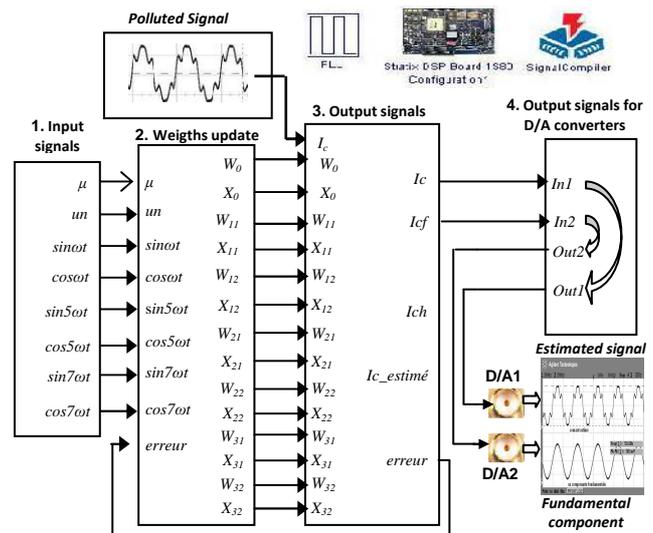


Fig. 4. Modular representation of harmonic currents identification design in ADB

B. FPGA Programming

In the algorithm's MatLab® version, the disturbed current is a signal composed by the following components:

TABLE I.
INITIAL CONDITIONS of the DISTURBED SIGNAL

Harmonic Components	Amplitude	Frequency(Hz)
Fundamental	100 (mA)	50
5-order harmonic	100/5 (mA)	5x50
7-order harmonic	100/7 (mA)	7x50

In addition, the learning rate is hold at 0.01, the Adaline weights are initialized to 0 and the sampling period is 5.10^{-5} s corresponding at a 20 kHz sampling frequency.

After a simulation time of 0.3sec, the weights are converging in the following way:

- for the fundamental $I_{11} = 100\text{mA}$ and $I_{12} = 0$
- for harmonic 5, $I_{21} = 20\text{mA}$ and $I_{22} \approx 0$
- for harmonic 7, $I_{31} = 14,28\text{mA}$ and $I_{32} \approx 0$

The observed learning phase is less than 0.08s.

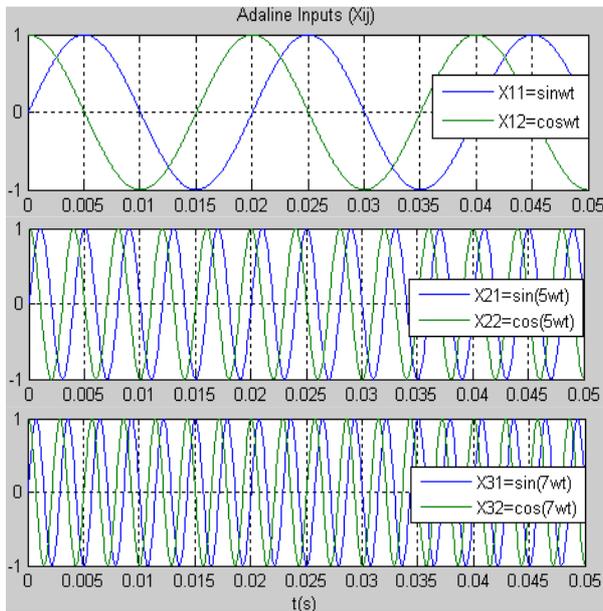


Fig. 5. Input signals of the Adaline

1) *Input signals generation module*: The used input vector is as follows:

$$X(t)=[1 \sin(\omega t) \cos(\omega t) \sin(5\omega t) \cos(5\omega t) \sin(7\omega t) \cos(7\omega t)]^T \quad (10)$$

Building this vector suppose:

- o to generate the time t with a unit ramp block.
- o to apply sine and cosine functions on the time t multiplied by values $(\omega, 5\omega$ and $7\omega)$.

With sampling periods of respectively 5.10^{-5} s and 5.10^{-8} s, simulations were done with fundamental current frequencies of 50Hz and 20Khz. Figure 6 shows Input signals waveforms of 20, 100 et 140Hz.

2) *Weights Update Module*: This module consists of 2-inputs adders, memory blocks, and self-built 3-input multipliers. Obtained simulation signals are given in figure 6 which indicate the evolution of the Adaline's weights values during learning and steady-state phases. During the approximately 0.06s long learning process, the W_{ij} weights vary quickly before being stabilized around 100, 0, 20, 0, 14.28 and 0 in accordance with the Matlab's® obtained values.

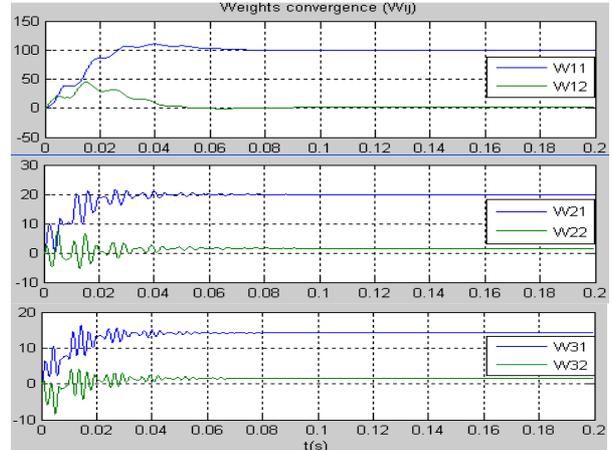


Fig. 6. Weights convergence of the Adaline

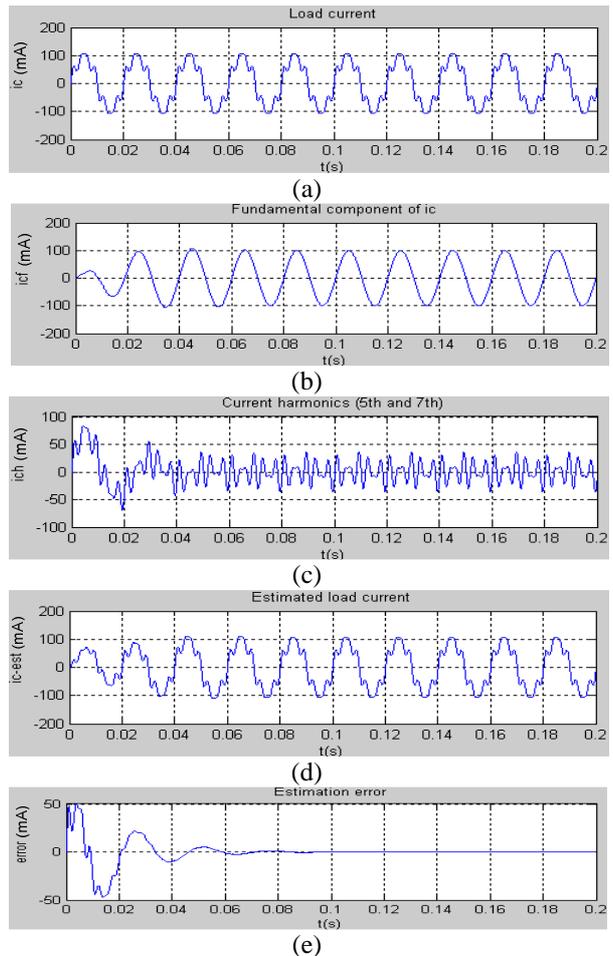


Fig. 7. Output signals from identification algorithm under ADB

3) *Output signals generation module*: Various signals generated by this module allow evaluating the effectiveness regarding signals estimation of the algorithm developed under ADB. Thus, figure 7 shows the disturbed load current (a) and its fundamental component (b). In addition, (c) presents the sum of the 5th and 7th harmonics extracted from (a), (d) the estimated current and (e) the estimation error. Therefore, multiplier and adder/subtractor blocks were used and a sub-module dedicated to the fundamental identification was developed.

4) *Adaptation module for output signals*: There are two 12-bits A/D signed format converters on the DSP board from figure 3 which allow to code values between -2048 and +2047. However, the two D/A converters are 14bits unsigned integer and can then receive 16384 positive values. Signal processing inside FPGA is done on wide bus size sometimes in signed fractional format. Hence, to obtain signals through D/A converters, we need to convert the data format to a 14 unsigned bit format. This is possible by using adders and especially the block "barrel shifter" of ADB interface.

IV. METHODOLOGY VALIDATION

A. Simulation results analysis

Figure 8 shows Matlab's[®] simulation results.

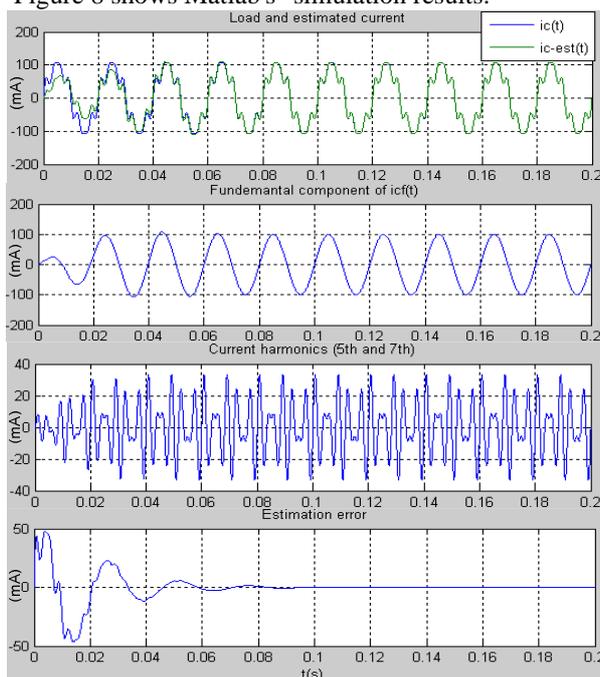


Fig. 8. Output signals from identification algorithm under Matlab[®]

We observe light differences on the harmonic signals especially during learning process before time 0.1sec. Nevertheless, they are the same in steady state (after 0.1s).

Moreover, figure 9 shows that the steady-state error is not really stable to 0 as under Matlab[®]. Fluctuations of $\pm 0.15\text{mA}$ remain even if they are negligible.

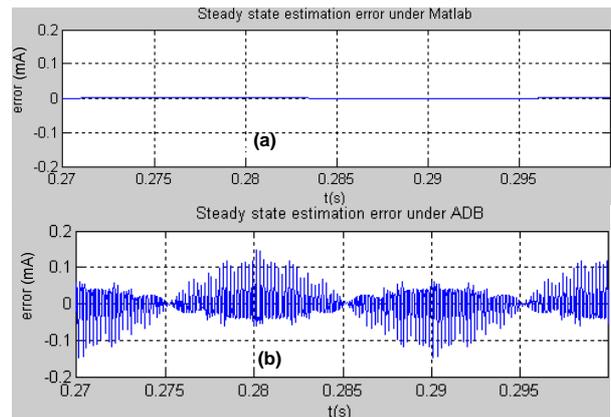


Fig. 9. Estimation error comparison under Matlab[®] and ADB - (a) Steady state estimation error under MatLab[®] (b) Details on steady state error under ADB

In addition, it is difficult to verify the frequency components by looking at an original signal, i.e in the time-domain. Converting currents in the frequency domain, the spectral decomposition shown in figure 10, represents the frequency content of the estimated current i_c and the identified harmonics current i_{ch} .

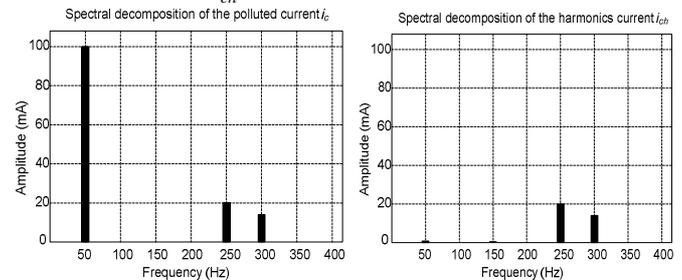


Fig. 10. Frequency content of the load current i_c and the harmonics current i_{ch}

According to figure 9, precision is a little bit affected for many reasons:

- Analog Digital conversion on the DSP board which is in signed bit format doesn't take into consideration fractional values of the input signal.
- The digital processing in fixed point is done through arithmetic blocks which, like the multiplier, double the data's output bus size when using two inputs of same size. The succession of calculations quickly increases the data bus size going beyond the limits of 51 bits.
- The D/A converters still impose to reduce the data buses size to be adapted to his 14 input bits.

B. FPGA Hardware implementation

We firstly proceeded to the *adaptation module for output signals* implementation while changing the amplification level. Thereafter, we embedded the *Input signals generation module* on the board. After those input/output blocks, we developed and implemented a *disturbed signal generation module*. We chose a 50Hz signal disturbed by the 5th and 7th order harmonics according to the low voltage power supply.

Since the weights *calculation* and the *output signals generation module* are not being able to be checked independently because of the estimation error feedback, we directly went through the whole design implementation which produced the attempted signals.

Figure 11 shows a 49,8Hz frequency disturbed signal of 575mV peak to peak from which the algorithm based Adaline extracts a 194mV peak to peak voltage representing harmonics current.

Harmonics identification during experimental phase has allowed an easy determination of the load's THD (at the 24% value) by using the estimated signal of figure 12. In addition we can observe that the fundamental component has 581mV peak to peak amplitude. The used resources by the implementation in the Stratix® EP1S80B956C6 FPGA are gathered in table II:

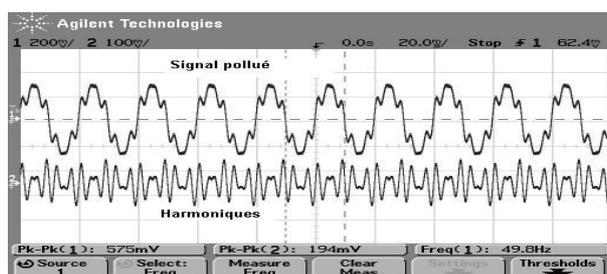


Fig. 11. Estimated disturbed signal and harmonics generated by the DSP board.

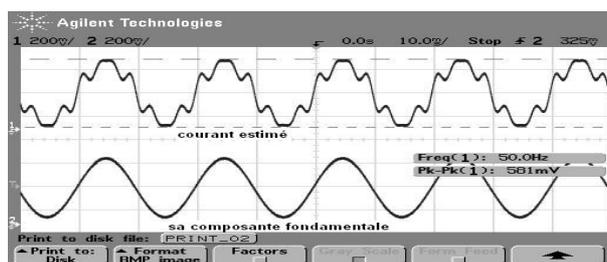


Fig. 12. Estimated disturbed signal and the fundamental component identified.

Table II shows an average use of 33% of the FPGA resources. In fact, this result includes the *disturbed signal generation module* which should come from a real low voltage power supply. Moreover, the use of the recent FPGA Stratix IV could offer more resources.

TABLE II.
USED RESOURCES on FPGA

Resources	Used	Total	Ratio in %
Logic elements	31.673	79.040	40%
Memory bits (RAM)	1.451.520	7.427.520	19%
Stratix pins	32	679	4%
9 bits Memory	1.290.240	7.427.520	17%
DSP blocks	176	176	100%
PLL	2	12	16.67%

V. CONCLUSION AND FUTURE WORKS

A hardware implementation of an intelligent current harmonics identification module based on neural networks has been presented. The design has been developed using the Altera DSP Builder®. Due to the limited resources available on an FPGA, fixed-point numeric representation which is more area-efficient than floating-point was preferred. According to the disturbed current estimation error during simulations, we observed that blocks parameterization, data bus sizes and self-built blocks used in the identification design didn't cause degradation on the compensation current.

This FPGA implementation validates the feasibility for neural networks implementation. With careful design, mapping the algorithm with adequate system performance is reachable despite the lack of resources. The designing modular approach will be extended towards the building of synthesizable and reusable blocks [13].

The proposed identification block will allow getting an excellent filter dynamic response to load variations. Although other methods for the control of active filters are widely known, the hardware implementation of the whole APF control unit with only neural networks will show with a source current THD of around 1% that ANNs are of efficient application in power electronics. The use of a FPGA target for such applications may improve the global performances due to its aptitudes for parallel processing.

REFERENCES

- [1] M. A. E. Alali, Y.-A. Chapuis, S. Saadate, and F. Braun, "An advanced common control method for shunt and series active compensators used in power quality improvement," in *Proc. Inst. Elect. Eng. -Electric Power Appl.*, vol. 151, no. 6, pp. 658-665, 2004.
- [2] L. Asiminoaei, F. Blaabjerg, and S. Hansen, "Evaluation of harmonic detection methods for active power filter applications," in *Proc. of the APEC'05*, March 6-10, 2005, Austin, Texas.
- [3] M. Su, D. Li, W. Yang, and H. Peng, "A novel adaptive predictive method, for harmonic detection," in *Proc. of the 4th China-Japan International Workshop on ITCA*, Hunan, China, 25-26 oct. 2005., pp 99-103.
- [4] V. Moreno, A. Pigazo, and R.I. Diego, "Reference Estimation Technique for active power filters using a digital Kalman algorithm," in *Proc. of the 10th International Conference on Harmonics and Quality of Power*, vol. 2, pp 490-494, 2002.
- [5] D. Ould Abdeslam, P. Wira., J. Mercklé, D. Flieller, and Y.A. Chapuis, "A unified artificial neural network architecture for active power filters". *IEEE Transaction on Industrial Electronics*, vol 54, no.01, pp 61-76, 2007.
- [6] D. Ould Abdeslam, P. Wira., J. Mercklé, Y.A. Chapuis, and D. Flieller, "Stratégie neuromimétique d'identification et de commande d'un filtre actif parallèle," *Rev. Int. Génie Électr.* 9, 2006, pp. 35-64.
- [7] Z. Shu, Y. Guo, and J. Lian, "Steady-State and Dynamic Study of Active Power Filter with Efficient FPGA-Based Control Algorithm," *IEEE Transactions on Industrial Electronics*, vol 55, no.04, pp 1527-1536, 2008.
- [8] Y. J. Chen and W. P. Du Plessis, "Neural network implementation on a FPGA," in *Proc. of the 6th IEEE Africon*, vol. 1, Oct.2-4, 2002, pp. 337-342
- [9] S. S. Kim and S. Jung, "Hardware Implementation of a real time neural network controller with Dsp and an FPGA," in *Proc. of the IEEE ICRA'04*, vol.5, pp 4639-4644, New Orleans-LA, 2004.

- [10] B. Widrow and E. Walach, *Adaptive inverse control, ser.* Information and System Science Series. Upper Saddle River, NJ: Prentice Hall Press, 1996
- [11] D. Ould Abdeslam, P. Wira, J. Mercklé, and D. Flieller, "Harmonic identification based on ANN : a comparative study," in *Proc. of the 9th international conference on engineering applications of neural networks*, EANN2005, pp 179-186, Lille, France, 2005.
- [12] J. R. Vasquez, P. Salmeron, J. Prieto, and F.J. Alcantara, "A new active power line conditioner for compensation in unbalanced/distorted electrical power systems," in *Proc. of the 14th PSCC*, June 24-28, Sevilla, 2002.
- [13] Y.A. Chapuis, J.P. Blondé, and F. Braun, "FPGA implementation by modular design reuse mode to optimize hardware architecture and performance of AC motor controller algorithm," in *Proc. of the 11th International Power Electronics and Motion Control Conference (EPE-PEMC 2004)*, vol. 1 pp. 134-142, Riga (Latvia), Sept. 2-4, 2004.

A modelling of the throughput of various architectures and its confrontation in the case of the AES algorithm implementation

Yves BERVILLER, Hassan Rabah and Serge WEBER
 Nancy Universités
 LIEN
 BP 239 - 54506 VANDOEUVRE lès NANCY Cedex
 Email : {first name}.{last name}@lien.uhp-nancy.fr

Résumé—In this paper we present a modelling of the processing throughput of a data processing architecture. We model various architectures, namely general purpose processors, *VLIW DSP*, *FPGA* with and without run-time reconfiguration. Based on this modelling, we can characterise various architectures before going through any implementation and make a selection based on the throughput requirements. We implemented the *AES* encryption algorithm in order to compare experimental results with our modelling. The model is quite accurate for the *FPGA* implementations, but requires some refinements for the processors, because the model always overestimates the throughput. But we can still use our modelling as the basis of a design methodology that can give some advices to a designer about the opportunity to whether use the run-time reconfiguration feature or not.

I. INTRODUCTION

One of the claimed goals of the run-time reconfiguration (*RTR*) feature allowed by some *FPGA* architectures is that this technique provides more flexibility than ASICs and more processing power than general purpose processors (GPP) [16]. The reasons given to justify this, are that more specialisation leads to more efficiency and a more generic architecture leads to less efficiency [15], ASICs and GPP being the extreme architectural cases. If this sounds, at least qualitatively, quite intuitive, from a quantitative point of view little work, has been done in order to prove it. In this paper, we will try to quantify at least one aspect of the more general efficiency : processing throughput. We have still proposed a modelling of the flexibility [11], but because the flexibility of an architecture is a multi-faceted metric it is difficult to compare practical architectures based on it. In contrast, throughput is easy to measure and is well defined, thus we propose to model it in an abstract way and then compare this modelling in the case of a signal processing application : the *AES* encryption algorithm.

The remainder of the paper is organised as follows : in section II we present the modelling used to assess the throughput of a *GPP*, a *VLIW DSP*, an *FPGA* and an *RTR FPGA*. Then, in section III we present briefly the *AES* algorithm. Section IV presents implementation results obtained on the various architectures. Section V presents a confrontation between the modelling and the results in this

particular application case and we conclude in section VI.

II. THROUGHPUT MODELLING

A. Throughput modelling of a *GPP* and a *VLIW DSP*

Here we make the very simple assumption that a *GPP* can load, apply a single processing instruction and output (or store) the resulting data whose size is the same than that of the data-path of the *GPP* at each clock cycle. This implies that if the processing algorithm needs N_P instructions in order to complete, the throughput will be N_P times lower than the one obtained with a single instruction. The duration of a clock cycle is $T_{GPP} = \frac{1}{Clk_{GPP}}$. Here again, one can object that the pipeline architecture of most *GPP* would give some improvements. This is true, and it is always possible to assume that all the pipeline stages are always fully supplied with data, thus providing an improvement factor equal to the number of pipeline stages available. Even with such a conservative assumption, it would be difficult to obtain a sustained throughput of one data word per clock cycle for an algorithm of more than a very few instructions. But we will see that there are generally many orders of magnitude between the throughput of a *GPP* and the one of an *FPGA*.

Hence, the data processing throughput Th_{GPP} of a *GPP* can be modelled by equation 1. We can see that this could explain the trend towards high width data-path, because the throughput is proportional to the width of the data-path (B_D). Of course, for a *VLIW* processor, the data-path has rarely the same size as instruction words have, but the throughput would still be proportional to the width of the data-path with the added benefits of an instruction level parallelism (β) greater than 1.

$$Th_{GPP} = Clk_{GPP} \cdot \frac{B_D}{N_P} \cdot \beta \quad (1)$$

B. Throughput modelling of an *FPGA*

In an *FPGA* we can theoretically choose any size for the data-path width. In a real case however, there are some limiting factors. First, there is a limited number of I/O pins available to the designer, secondly, and may be most importantly, the data

buses pushing and pulling data to and from the *FPGA* are generally not very wide. We assume that the data-path is B_O bits wide, that it is fully pipe-lined and that it operates at a clock frequency $Fclk_{FPGA}$. Hence the throughput Th_{FPGA} can be modelled like in equation 2. We can see that an important difference is that the complexity of the algorithm (i.e. the number of “instructions” to apply) does not affect the throughput in this case. This is in contrast with the *GPP*, where it was inversely proportional to the complexity.

$$Th_{FPGA} = Fclk_{FPGA} \cdot B_O \quad (2)$$

C. Throughput modelling of an RTR FPGA

In the case of run-time reconfiguration of an *FPGA*, the throughput modelling depends on four more parameters : the number N of temporal partitions, the size D that is the number of words in each data block processed between reconfigurations, the number n of pipeline stages in the data-path and the reconfiguration time T_{reconf} that is needed for reconfiguring the logic cells used in a temporal partition of the algorithm. Hence, we can model the average throughput, which is the number of processed bits per time unit, of an *RTR FPGA* implementation by means of equation 3. Thus, we can see a double overhead in the use of *RTR*. The first one is reduction by a factor of N of the throughput if we neglect the reconfiguration time. The second one is the value of T_{reconf} , that in practise, is very great compared with $Tclk_{FPGA} = \frac{1}{Fclk_{FPGA}}$, thus requiring a very big value for D in order to achieve an interesting throughput. The last case implies that n can generally be neglected with respect to D .

$$Th_{RTR\ FPGA} = \frac{D \cdot B_O}{N \cdot ((D + n) \cdot Tclk_{FPGA} + T_{reconf})} \quad (3)$$

III. THE AES ENCRYPTION ALGORITHM

Data security is a significant subject for which various algorithmic solutions have been proposed. In 2001, Advanced Encryption Standard (*AES*) was accepted as a *FIPS* (Federal Information Processing Standard) [10]. *AES* is an encoding algorithm intended to replace *DES*, which had already showed some safety weaknesses in data protection. In October 2002 *NIST* (National Institute of Standards and Technology) selected the Rijndael cipher developed by two Belgian cryptographers as the *AES* algorithm. Since then, many achievements on hardware and software had been proposed by combining various architectures. In general, various architectures have been used to apply the *AES* algorithm on hardware. They seek to satisfy two metrics important in digital systems : the throughput, and the area or the amount of hardware resources required to achieve this throughput. The throughput reached goes from 20 Mbps to 70 Gbps according to the technology and the architecture used as described in [3], [4], [5], [6], [7] and [8]. The technology of the circuits as well as the tools available for the design and the implementation of the algorithms have played a significant role to achieve a high throughput, but with a high cost in terms of resources used.

Nevertheless, the intrinsic parallelism of the algorithm is still well adapted to a hardware implementation.

A. Description of the algorithm

The *AES* is a block cipher with possible lengths for the blocks and the key of 128, 192 and 256 bits. The blocks to be encrypted and the key can have different lengths. The encryption is comprised of a variable number of rounds (determined by the key and block lengths) with each round containing four transformations : *ByteSub*, *ShiftRow*, *MixColumn* and *RoundKeyAddition* (in the last round, the *MixColumn* is omitted). An initial key is expanded to form an Expanded Round Key based on the number of rounds. Since *AES* is a symmetric cipher, decryption is just the inverse of the encryption. If more details are needed see [10], [1]. Figure 1 shows the operations required by the algorithm [9]. Thus, we can separate the *AES* algorithm into two distinct functions : the key expansion part and the cipher part.

B. The key expansion part of the AES

The *AES* algorithm takes the Cipher Key K , and performs a Key Expansion routine to generate a key schedule (i.e. the ten different keys that will be used later by the Cipher module). The Key Expansion generates a total of $Nb \cdot (Nr + 1)$ words : the algorithm requires an initial set of Nb words and each of the Nr rounds requires Nb words of key data. The resulting key schedule consists in a linear array of 4-byte words, denoted $[wi]$, with i in the range $0 < i < Nb(Nr + 1)$ [10], [1].

The data are arranged in a linear vector of words of 4 bytes, indicated by $[wi]$. The data are put to the algorithm through *dato_e(128bits)* and the result is provided at *dato_s*. Let us specify that *temp* is a variable of 32 bits wide and $w[i]$ is the line of a matrix that has a dimension of 4 by 4 bytes. The *RotWord* function takes a word of 32 bits $[a0, a1, a2, a3]$ as input, carries out a cyclic permutation, and returns the word $[a1, a2, a3, a0]$. *SubWord* is a function that takes on its entry a word of four bytes and applies a look-up matrix *S_Box* to each four byte to produce a new word. This matrix has a size of 256 data of 8 bits each. The constant *Rcon[i]* contains already defined values. For word indices that are integer multiple of Nk (number of 32-bit words comprising the Cipher Key), a transformation is applied to $w[i - 1]$, followed by an *XOR* with a constant iteration, *Rcon[i]*. The transformation is composed of a circular shift of the bytes in a word (*RotWord*), followed by a look-up of each byte in a word (*SubWord*). Figure 2 shows the block diagram of the execution of a single round for this module.

C. The cipher part of the AES

At the beginning of the Cipher module, the input is stored in the State array that has a size of 128 bits (16 bytes). Following the addition of the K_i key (the $i - th$ key), the State array is modified by applying the standard round ($Nr - 1$ times) and a final round, which does not include the *Mixcolumns* transformation. Finally, State is sent to the output. The various

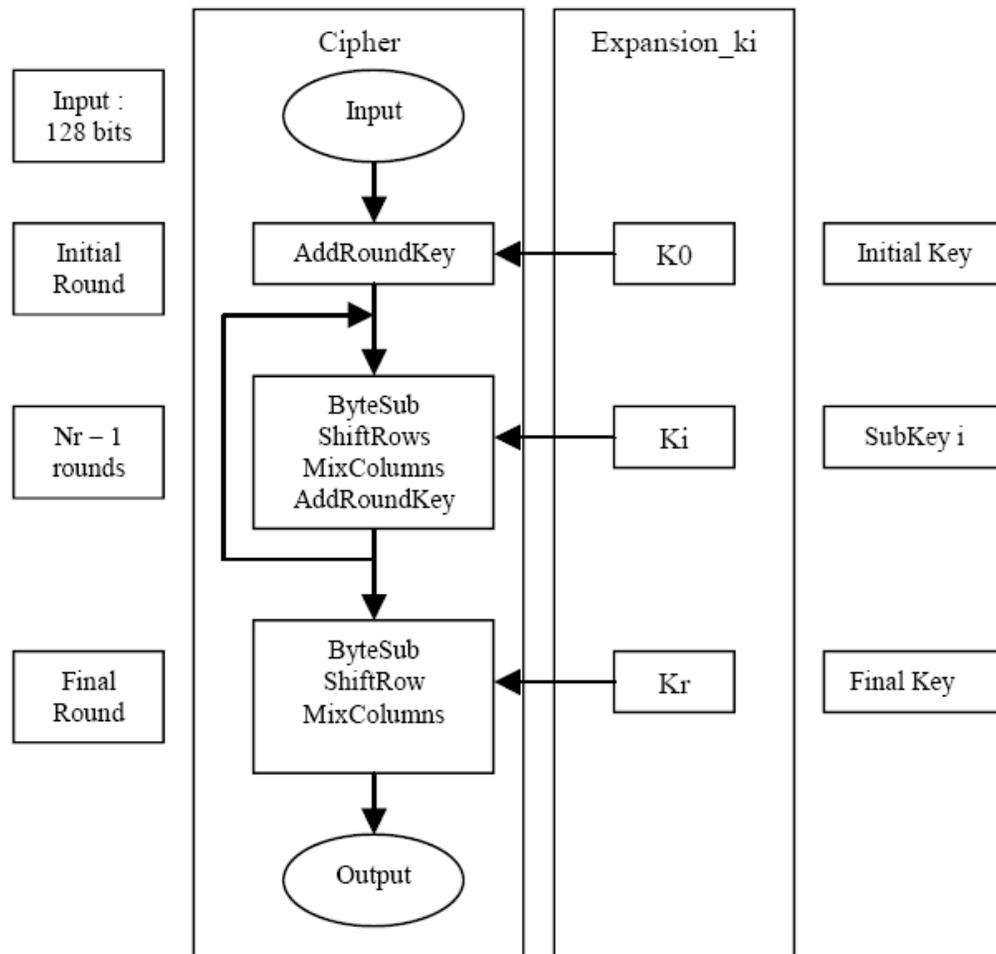


FIG. 1: Block diagram of the AES algorithm

transformations (*SubBytes*, *ShiftRows*, *MixColumns*, and *AddRoundKey*) that treat the State array are now described :

- *SubByte* is a non-linear function, operating independently on each byte from the State vector, known as a substitution box (*S - Box*).
- The *ShiftRows* function shifts the data (this function divides its input in 4 segments of 4 bytes each and makes a rotation towards the left of respectively 0, 1, 2, 3 bytes for segments 1, 2, 3 and 4).
- *MixColumns* is a function that transforms each byte of input into a linear combination of bytes. This function can be expressed mathematically as a matrix product in the body of Galois (28) [10]. This matrix multiplication uses multiplications in "finite fields" by two and three, that reduce to an *XOR* function and thus makes the architecture more efficient [2].
- The *AddRoundKey* transformation adds K_i (previously generated by the key expansion function) to State by use of an *XOR* operator. Each K_i is composed of Nb words. K_i is the k -th sub-key calculated by the algorithm starting from the main key K . The application of the *AddRoundKey* transformation in the Nr rounds of the

cipher, occurs when $1 < round \leq Nr$ [10].

Finally, as it can be seen on figure 3, the transformations *SubBytes*, *ShiftRows* and *MixColumns* are always used. So, the re-usability of these operators can be exploited here.

IV. IMPLEMENTATION RESULTS

A. The various architectures and implementations

We have implemented the AES 128 - 128 algorithm on three kind of processors and on one FPGA with three different architectural choices. This leads to six implementation results for which the measurement methods are briefly described :

- The *P4* implementation was made on a Pentium 4 processor by help of a *C* compiler. This processor has 512 *KBytes* of level 2 cache and runs at 2.4 *GHz*. The measures were made by the use of the system *timer* functions and correspond to the average of many results.
- The μB implementation was made on a *MicroBlaze* processor embedded in an *FPGA* and again in the *C* language. This processor runs at 100 *MHz* and had no cache memory. The measure was made by toggling a *I/O*

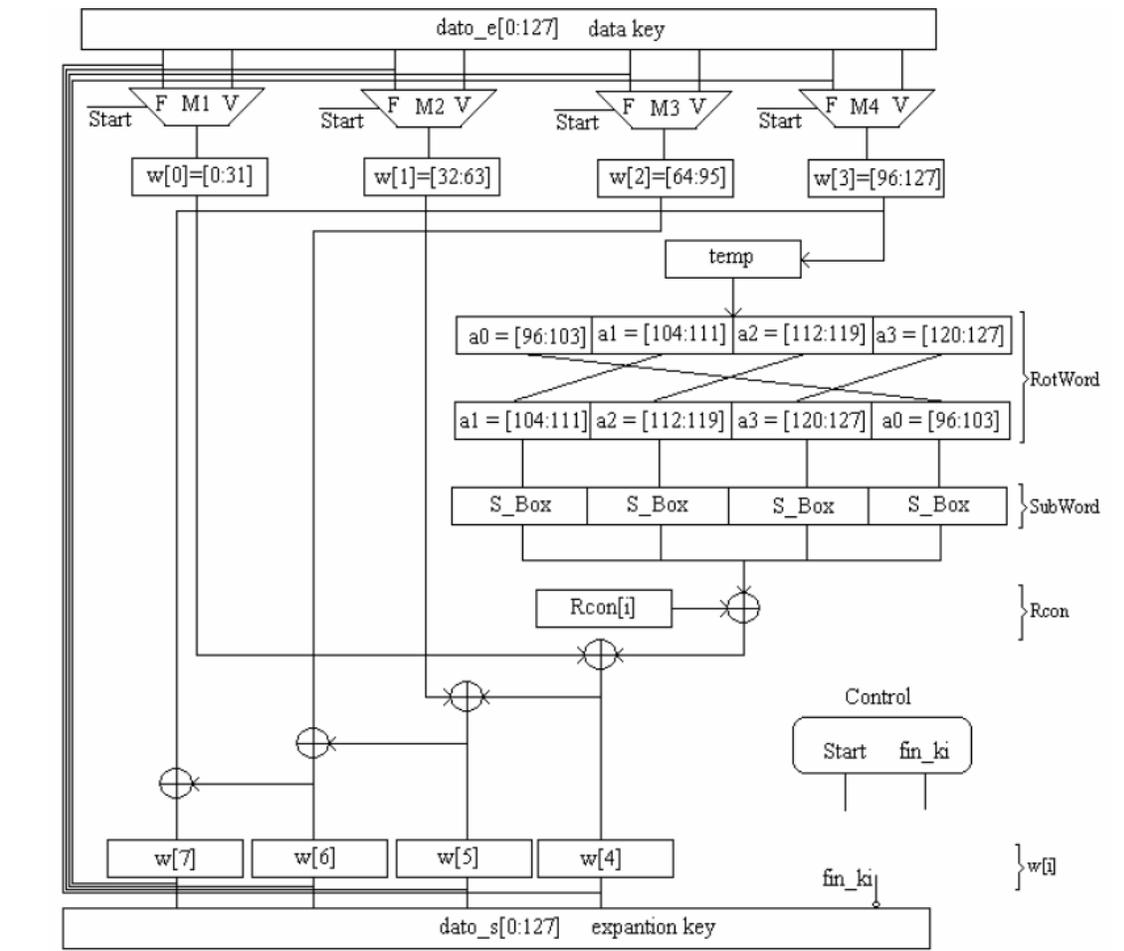


FIG. 2: The key expansion part of the AES algorithm

port bit at the begin and the end of the processing and an oscilloscope attached to this port.

- The *VLIW* implementation was again made in *C* on a *TMS320C6713 DSP*. This processor runs at 200 MHz has 4 KBytes of both instruction and data level 1 cache a shared 256 KBytes level 2 cache. We used the same measurement method as for the *MicroBlaze* processor.
- The *FPGA_Fact* implementation was made from a *VHDL* description on a *Virtex 2 FPGA*. The implementation kept the iterative nature of the *AES* algorithmic description. The measurement of the throughput was easy thank to the simulation work preceding the final implementation. The same is true for the next two *FPGA* implementations.
- The *FPGA_Unroll* implementation was made from a *VHDL* description on a *Virtex 2 FPGA*. The implementation took a modified description of the *AES* algorithm in order to remove its iterative nature. We unrolled the 10 iterations and this leads to a pipe-lined chain of replicated sets of operators.
- The *FPGA_RTR* implementation was made from a

TAB. I: Implementation results for the 6 implementations

Implementation	Throughput (Mbps)	Kernel "size"	Tconf (ms)
<i>P4</i>	41	1875 clocks	-
<i>μB</i>	2.6	1400 clocks	-
<i>VLIW</i>	16	400 clocks	-
<i>FPGA_Fact</i>	4200	618+1225 cells	2.3
<i>FPGA_Unroll</i>	44000	2725+11705 cells	18.2
<i>FPGA_RTR</i>	<4200	1285 cells	2.4

VHDL description on a *Virtex 2 FPGA* and used the partial reconfiguration feature of these devices in order to implement separately the two parts of the *AES* (the key expansion end the cipher itself). This introduce a new parameter because the mean throughput now depends on the length of data encrypted with the same key.

B. Results

Table I gives the results that we have obtained for our six implementations. The results for the *FPGA* are very high compared to the processors despite a quite low operating frequency (300 MHz), this is not really a surprise. It is mainly

TAB. II: Modelling results for the *AES* algorithm

Implementation	Throughput	N_P or B_O	Work Freq.
<i>P4</i>	468 Mbps	164 instr.	2400 MHz
μB	20 Mbps	164 instr.	100 MHz
<i>VLIW</i>	240 Mbps	164 instr.	200 MHz $\beta = 6$
<i>FPGA_Fact</i>	3840 Mbps	128 bits	300 MHz
<i>FPGA_Unroll</i>	38400 Mbps	128 bits	300 MHz
<i>FPGA_RTR</i>	<3840 Mbps	128 bits	300 MHz

due to the pipeline type of parallelism and the adaptation to the size of the data (128 *bits*). One must note that the result for *FPGA_RTR* is a kind of asymptotic value, because it corresponds to a case where the key does not change frequently. This means that the value of D in equation 3 is very big.

V. CONFRONTATION OF THE MODELLING AND THE EXPERIMENT

A. Modelling applied to the experimental cases

We apply the modelling described in section 2 for each corresponding case of the experiment and we obtain the results given in table II. There are two important parameters to characterise in order to use the model : N_P for the processors and T_{Clk_FPGA} for the *FPGA*. For N_P we enumerate the basic operations required by the *AES* algorithm and we came to a total of 164 instructions among them 92 are used by the key expansion part and 72 by the cipher part. For T_{Clk_FPGA} we used the data-sheet of the *FPGA* and applied a 40% penalty to the maximum frequency in order to take into account the routing between operators.

B. Discussion about the divergences

While the modelling is quite accurate for the *FPGA* implementations there are some issues in the cases of the processors. It is worth noting that the algorithm used here leads to a regular data-path with the output of an operator going straight to the input of the next one, except for the iteration, but since there is only one "feedback" path, there are not many routing congestion issues. This could explain that our rough 40% penalty leads to a quite accurate frequency estimation. Furthermore, we did take into account the fact that for the *FPGA_Fact* implementation, a result was output only every 10 clock cycles. This highlights a big weakness of our modelling, indeed it does not take into account the structure of the algorithm. This is quite a problem here because of the iterative nature of the *AES* description and this leads to a highly underestimated value of the number of clock cycles needed by the processing kernel. Thus if we correct the value of N_P in order to take into account the iterations, the new results (obtained easily by dividing the values from table II by 10) become closer to the modelling. But after correcting for this, the more inaccurate modelling concerns the *VLIW*. This is most probably due to the fact that the C description did not permit to the compiler to use all the potential parallelism

provided by the *DSP* architecture. Indeed, the *DSP* that we used has 8 processing units among which 6 contain an *ALU*, so one would expect to see at least a value of β equal to 6. But if we look at the results, we only see an effective value of 4. This has always been a problem with *VLIW* architectures. We will try to optimise the C description or even write the kernel in assembly language to see if one can obtain results more close to the modelling.

VI. CONCLUSION

In this paper we have presented a modelling of the throughput of various data processing architectures. Based on this modelling, we can characterise various implementations for the *AES* encryption algorithm on different kind of processors and various architectural choices on *FPGA*. We made six implementations of the *AES* on three types of processors and three on an *FPGA*. We compared the experimental results with our modelling and we obtained quite accurate results for the *FPGA* implementations. The processors implementation were less accurate, this is due in part to a not fine enough model (i.e. ignoring the structure of the algorithm) but also because of compiler issues. However, given some throughput constraints, a designer can quickly estimate which architectures are suitable for his needs. Then he can choose its final architecture based on other metrics such as the resource requirements, power consumption, ... or on more subjective criteria like development time or the ease of design modifications. Our future work concentrate on the refinement of this modelling and the modelling of the power consumption in order to help the designer to assess the advantages and drawbacks of run-time reconfiguration compared to other architectures very early in the design schedule.

RÉFÉRENCES

- [1] *Data Encryption Standard*, revised version issued as FIPS 46-3, National Institute of Standards and Technology, 1999.
- [2] M. McLoone and J.V. McCanny, *Rijndael FPGA implementations utilising look-up tables*, Journal of VLSI Signal Processing, July 2003, pp. 261-275.
- [3] A.J. Elbirt, W. Yip, B. Chetwynd, C. Paar, *An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists*, IEEE Transactions on VLSI Systems, Volume 9, Issue 4, Aug 2001 pp. 545-557.
- [4] F-X. Standaert, G. Rouvroy, J-J. Quisquater, J-D. Legat, *Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware : Improvements and Design Tradeoffs*, CHES 2003 LNCS 2779, pp. 334-350, 2003.
- [5] P. Chodowicz, P. Khuon, K. Gaj, *Fast implementation of secret-key block ciphers using mixed inner and outer-round pipelining*, Proceedings of the ACM/SIGDA international Symposium on Field Programmable Gate Arrays, FPGA'01 Monterey, CA, February 11-13, 2001.
- [6] A. Hodjat and I. Verbauwhede, *A 21.54 Gbits/s Fully Pipelined AES Processor on FPGA*, Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 308-309, 2004.
- [7] K. U. Järvinen, M. T. Tommiska and J. O. Skyttä, *A fully pipelined memoryless 17.8 Gbps AES-128 encryptor*, Proceedings of the Eleventh ACM/SIGDA international Symposium on Field Programmable Gate Arrays, FPGA '03 Monterey, CA, February 23 - 25, pp. 207-215, 2003.
- [8] P. Kancharla and D. A. Buell, *The Advanced Encryption Standard on the HC-36m reconfigurable computer*, Proceedings of the Military Applications of Programmable Logic Devices (MAPLD), Washington, DC, 9-11 September 2003.

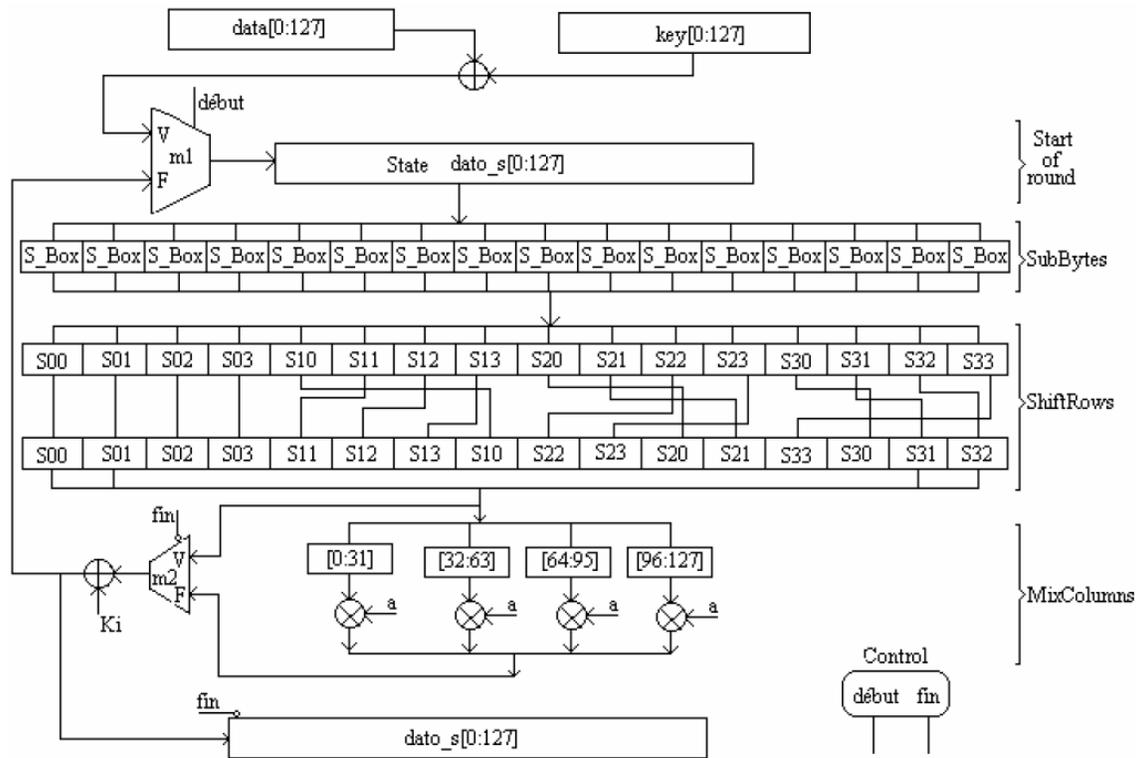


FIG. 3: The cipher part of the AES algorithm

[9] José de Jesús Angel, *AES Advanced Encryption Standard ; 1999*. URI : <http://computacion.cs.cinvestav.mx/~jjangel>

[10] Announcing the ADVANCED ENCRYPTION STANDARD (AES). Federal Information Processing Standards, Publication 197, 2001, AES page available via <http://www.nist.gov/CryptoToolkit>.

[11] Y. Berviller, O. Perez, S. Weber, *A modelling of the flexibility of an RTR-FPGA implementation in comparison to a software implementation*, DASIP 2007 European Workshop on design and Architectures for signal and image processing, Nov 27-29, Grenoble 2007.

[12] A. Ye and J. Rose, *Using bus-based connections to improve field-programmable gate-array density for implementing datapath circuits*, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on Volume 14, Issue 5, May 2006 Pages :462 - 473

[13] E. Ahmed, J. Rose, *The effect of LUT and cluster size on deep-submicron FPGA performance and density* Very Large Scale Integration (VLSI) Systems, IEEE Transactions on Volume 12, Issue 3, March 2004 Page(s) :288 - 298

[14] A. DeHon, *Comparing Computing Machines*, Proceedings of SPIE, pp. 124-133, November, 1998.

[15] A. DeHon, *DPGA-Coupled Microprocessors : Commodity ICs for the Early 21st Century*, FCCM94, IEEE workshop on FPGAs for custom computing machines, Napa CA, april 10-13 1994.

[16] K.Compton, S. Hauck, *Reconfigurable Computing : A Survey of Systems and Software* ACM Computing Surveys, Vol. 34, No. 2, June 2002, pp. 171-210.

[17] P. Brunet, C. Tanougast, Y. Berviller, S. Weber, *Hardware partitioning software for dynamically reconfigurable SoC design*, Proceedings of the 3rd IEEE International Workshop on System-on-Chip for Real-Time applications, 30 June-2 July 2003 Page(s) : 106 - 111

OveRSoC Graphical Design Environment

Mehdi AICHOUC*

Emmanuel HUCK*

*ETIS, CNRS, ENSEA

F-95000 Cergy-Pontoise

[Email]: firstname.name@ensea.fr

Benoît MIRAMOND*

Abstract— This document presents the OveRSoC Graphical Design Environment. The OveRSoC project objective is to develop an exploration and validation methodology of embedded real time operating systems for reconfigurable System-On-Chip platforms. Here, we describe an integrated graphical environment to explore, simulate, and validate the distribution of OS services on these platforms. Our work mainly focuses on developing an easy to use tool which supports graphical editions, SystemC source code generation, and functional/timed simulation.

The OveRSoC project deals with embedded systems featuring dynamically reconfigurable units. The goal is to develop a complete executable model of an RSoC platform (hardware and software) in order to bring out the services that an embedded real time operating system should provide. The project also proposes an exploration flow based on a flexible SystemC model of Real Time Operation System (RTOS). This RTOS model is a package of modular services [1]. To develop each service, an Object Oriented approach has been adopted and implemented using SystemC 2.2 library. This model allows the building of an application specific RTOS by assembling generic custom OS services basic blocks. The application is linked to the resulting OS from a POSIX standard API. The application and its custom OS are then deployed both in software and hardware. Finally, the entire platform is simulated using the SystemC kernel. The design methodology for the exploration of the OS services distribution is described in figure 1. The software aims to automate this methodology through the following main design steps:

Design of the platform: The design phase consists of choosing and instantiating the needed components into the graphical editor in order to assemble the OS services and distribute them into the RSoC execution units.

SystemC source code generation: After interconnecting all components and verifying the bindings, the structural source code of all the objects instantiated graphically into the platform are generated automatically.

Compilation and Simulation of the platform: To complete the creation of the platform, the parameterized structural SystemC description is combined with the behavioral source code of all component provided by the user. The global SystemC description is compiled and simulated.

Analysis of the simulation results: Graphical diagrams are produced to visualize the evolution of the system metrics dur-

ing the simulated time. This step helps the designer to evaluate the quality of the design. It acts as a decision guide for the exploration of the design solution space.

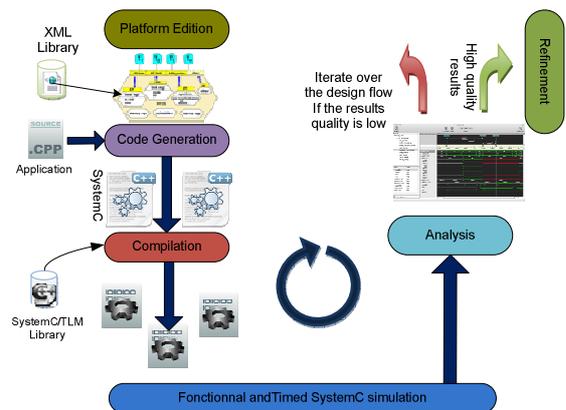


Fig. 1 OveRSoC Design Flow

The approach has been applied on a robotic vision application [2] in order to explore its hardware and software implementation. In this study case, encouraging results have been obtained. We are currently implementing the OveRSoC GME as a standalone application based on Eclipse Rich Client Platform. Standard project management functions like importation of API service or template components into standard provided library are supported as well as the creation of new platform models. The SystemC library and the C++ compiler can be customized by a preference menu. The software offers the possibility to add the new created platform to the pre-configured library. All data manipulated by the OveRSoC GME application are stored using a proprietary XML format focusing embedded software modelling. It is also possible to import external RTL hardware IP components only if they are previously stored in the XML format conformed to IP-XACT specification [3].

REFERENCES

- [1] E. Huck, B. Miramond, and F. Verdier, *Modular SystemC RTOS Model for Embedded Service Exploration*, First European Workshop on Design and Architectures for Signal and Image Processing (DASIP). Grenoble, France, 2007.
- [2] E. Huck, T. Lefebvre, M. Maillard, B. Miramond, and F. Verdier, *Using High-Level RTOS Models for HW/SW Embedded Architecture Exploration: Case Study on Mobile Robotic*, EURASIP Journal on Embedded Systems. 2008.
- [3] The Schema Working Group of the SPIRIT Consortium, IP-XACT v1.4: A specification for XML meta-data and tool interfaces, Mar 12, 2008.

MDE benefits for Real Time Operating Systems modeling

Yessine Hadj kacem¹, Adel Mahfoudhi^{1,2}, Mohamed Abid¹

¹National Engineering School of Sfax Road Soukra km 3,5
Computer & Embedded Systems Laboratory (CES)

B.P. : w -- 3038 Sfax TUNISIA

²Department of Computer Science, Science Faculty of Sfax
Road Soukra km 3,5 BP : 1171 -- 3000 Sfax TUNISIA

adel.mahfoudhi@fss.rnu.tn

mohamed.abid@enis.rnu.tn

Abstract—A large part of real time embedded systems RTES has to satisfy real time constraints and it usually employs Real Time Operating System RTOS. In order to decrease the design complexity of such systems, they need methods and tools based on high abstraction layers.

Currently, UML profiles are found to be an effective solution for the automatic RTES design. Unfortunately, they are poor in integrating RTOS modeling.

In the present work, a methodology based on model driven engineering MDE for RTOS design is introduced. The proposed approach aims at defining a platform independent model of RTOS. It suggests the implementation of statecharts relating to the process states, whose real time constraints can be checked by defining their semantic variants. The ultimate goal is the automatic generation of the code related to the scheduler.

Keywords— RTOS modeling, semantic variants, statecharts implementation, MDE, automatic code generation

I. INTRODUCTION

Designing ERTS has always been a challenge. Indeed, standards to facilitate the checking of system properties at a preliminary stage are progressing well based on different abstraction layers. With regard to the bottom layers, there exists many synthetic tools; the only problem relates to the CAD (Computer Aiding Design) of the highest level which is the concern of this work.

At present, UML (Unified Modeling Language) is having a growing interest in the software and hardware development. It represents a viable solution to decrease the complexity of ERTS design via UML profile. The ERTS complexity depends on the architecture deployment and requires runtime guarantees. Although ERTS requires a real time operating system, the existing profiles do not integrate the notion of RTOS modeling during the ERTS design, since they just focus on architecture and application modeling.

This paper presents our contribution to the RTOS modeling, based on model driven engineering. In fact, the main goals are the definition of a platform independent model of RTOS and the automatic generation of the code related to the scheduler. It suggests the implementation of statecharts relating to process states.

For the suggested models, the structure of the RTOS is described through a class diagram which includes the definition of operational semantics. Then, the behaviour of a task which constitutes the core of the RTOS is defined in order to ensure coherence between various diagrams UML. The temporal and transitional semantics of the statecharts relative to the various states of a real time process is defined in order to reach the model of task scheduling. These two independent platform models are integrated in MDE process to generate the code related to the scheduler.

This paper starts by providing a brief discussion about some related work in section two. Then, the proposed design methodology is described in section three, in which the models of the RTOS structure and the scheduler are introduced through the implementation of statecharts. The experimental results of an application example are provided in section four. At the end, some final conclusions with some future work are given.

II. RELATED WORK

After examining the specificity of each UML profile such as SPT (Scheduling, Performance and Timing) [15, 12], QoS/FT (Quality of Services & Faults Tolerance) [15] and MARTES (Modeling and Analysis of Real-Time and Embedded systems) [13], it is concluded that the focus was on the description of the material architecture and the application. These profiles are founded on an abstraction level higher than other approaches like ROOM, SDL, ADL, Petri Net. They also aim at the applications to data flow predominance rather than those to control. Even though these works briefly tackle the temporal aspect, they cannot cover the RTOS modeling. They are criticized for the lack of temporal and transitional semantics common to the models as well as the absence of tools which support them. In reality, these works have not enabled us to guarantee the reliability of the system yet; i.e., its determinism aspect. These models do not support the integration of real time characteristics sufficiently and therefore they do not consider the RTOS related to a specific architecture and application. The simulation approaches need a simulation time long enough to give a relatively reliable

sight of operation.

According to [16], RTOS modeling is based on two independent class diagrams: a diagram describing the structure and another describing the scheduler. They suffer from major limitations, namely the coherence between the used diagrams and the lack of temporal semantics definition. In fact, because it can not cover the temporal behaviour of the RTOS, the diagram used to characterize the scheduler is a static one. It must also be complementary to the structure model via a good expression of the follow-up of the real time process evolution. Therefore, a methodology assuring the coherence between the used diagrams and the support of scheduling model is important.

In [14], the author's work consists in developing a middleware in order to implement scheduling algorithms on a real platform. It just focuses on mapping and ordering tasks dynamically for platforms using a middleware layer between the application and the RTOS.

In [10], a model driven approach aims at proposing generic RTOS APIs (Application Programming Interfaces) and generating a fully-functional code by transforming generic RTOS APIs into RTOS specific APIs. This proposition can describe most of typical RTOS services but does not support real time task scheduling.

III. RTOS MODELING

The proposed methodology presents a step ensuring coherence between the various used UML diagrams and covering the behavioral aspect of the system as real time constraints. First, the model of the RTOS structure is defined. Then, a statechart diagram related to the state of a real time task is specified. After that, the temporal semantics presented by the statecharts [1] is given. While defining the semantic variation points of the statecharts, some techniques such as the reification and enumeration of the states and the events are applied. The model related to the RTOS structure corresponds to the source model during the stage of model transformation. During this stage, the scheduling model represents the target model.

After defining the RTOS models, temporal constraints such as deadline, duration, etc are specified using Object Constraint Language (OCL).

A. RTOS structure model

A class diagram is proposed for the description of the RTOS structure. It describes the major components of the RTOS.

The class diagram which is presented by Figure 1 is characterized by the following entities:

- Task: It is the most important component of the RTOS. A task must acquire a great number of information in order to manage their scheduling
- Event: It causes the change of a task state
- ISR: Interrupt Server Routine: It is the routine in charge of the interruption processing. In this context,

it makes the relay between the material interruption mechanism and the software one

- Alarm: Based on a meter, an alarm could activate a task, impose an event or activate an alarmCallback
- Counter: It presents a software/ hardware source for an alarm. It is an object intended for the recording of "ticks" coming from a timer
- Resource: This entity is used to coordinate the concurrent accesses to shared resources. It is similar to semaphores. It is also used for explaining resource management.
- MeanOfCommunication: It is an abstract interface which manages data between active objects [4]. The class ProtectedVar, which implements this interface, associates a mechanism of data protection (semaphore). In addition, LettreBox uses a file of messages. Besides, the read and write methods of this interface can update or get the value of the protectedVar class. This entity ensures data protection
- Watchdog: The ISR contains one or more watchdog timers. The watchdog could possibly provide debugging information
- Precedes: It illustrates the dependence of a task on another. It includes the definition of operational semantics [2].

Before presenting the scheduling model, it should be born in mind that each state of a running task on RTOS can take only one of the following values: {Waiting, Running, Ready, Suspended, Created}. As for the event, it has these values : {terminate, activate, start, wait, preempt, release, create}.

B. RTOS scheduling model based on statecharts implantation

The structure of the statecharts diagram is, nonetheless, given a precise specification [17]. It can not easily be understood. So, UML 2.0 Statecharts present some semantic variation points. These variation points [9] concern, principally, three aspects: the time management, event selection policy, and transition selection policy.

A set of approaches [8, 11] was proposed in the literature in order to define these semantics and implement the statecharts. The present work adopts the approach proposed by [6], whose technique is based on the enumeration and reification

The reification consists in the transformation of states into specific class hierarchy through the application of the design patterns.

A solution to separate the behaviour related to a state in an object, is to reify states through the use of the state pattern [7]. To reify and select the right transition events, the command pattern [7] is applied to the entity Task (See Figure 2). In order to ensure the progression of the automat, it is necessary to focus on the deterministic aspect of the system. It is also essential to determine the state running of the automat and the behavior to be adopted according to the event which has occurred.

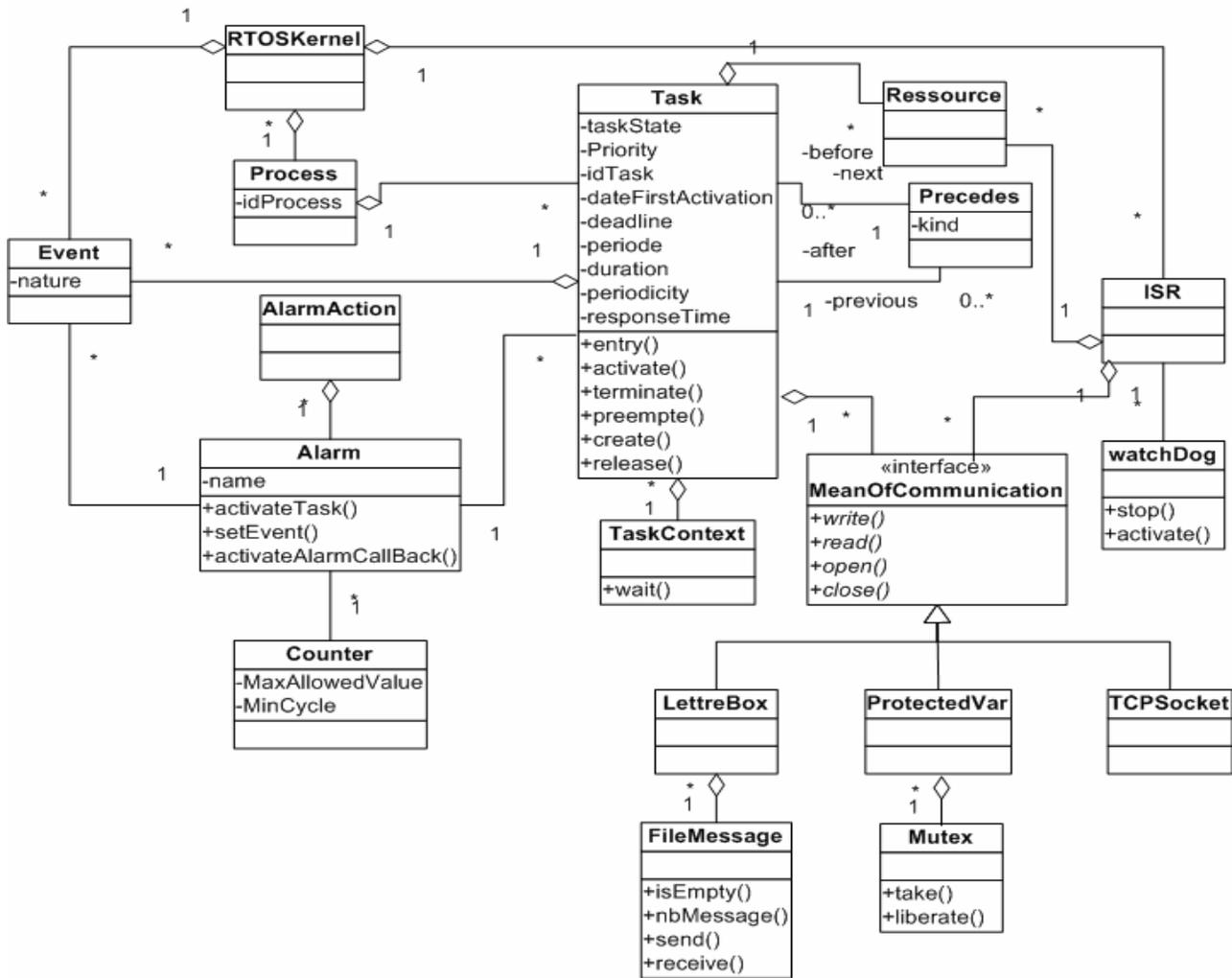


Figure 1: Static Model of the RTOS Structure

Regarding the enumeration of the states and the events, the code reacting the progression of the automat is localized in the method processEvent(). As for the enumeration of the states and the reification of the events, the code will be set out again between the method processEvent() and execute() of each class. As far as the reification of the states and the enumeration of the events are concerned, the code will be distributed between the method processEvent() and the method processEventPlay() of each class state. Finally, when the states and the events are reified, the code is distributed between the method processEvent() principal class, the processEvent() methods of the state class and the execute() methods of the class called event.

The last solutions based on enumeration and reification do

not allow the representation of the concept of file messages related to the automat progression. Time is not taken into account. To overcome this problem, the use of the Active-Object pattern is, therefore, essential since it is effective for the achievement of the various policies of parallelism as shown in Figure 2.

Following the reification application of the states and events, as well as the evolution illustration of the automat, the final model represented in Figure 2 will be considered as the target model during the transformation process.

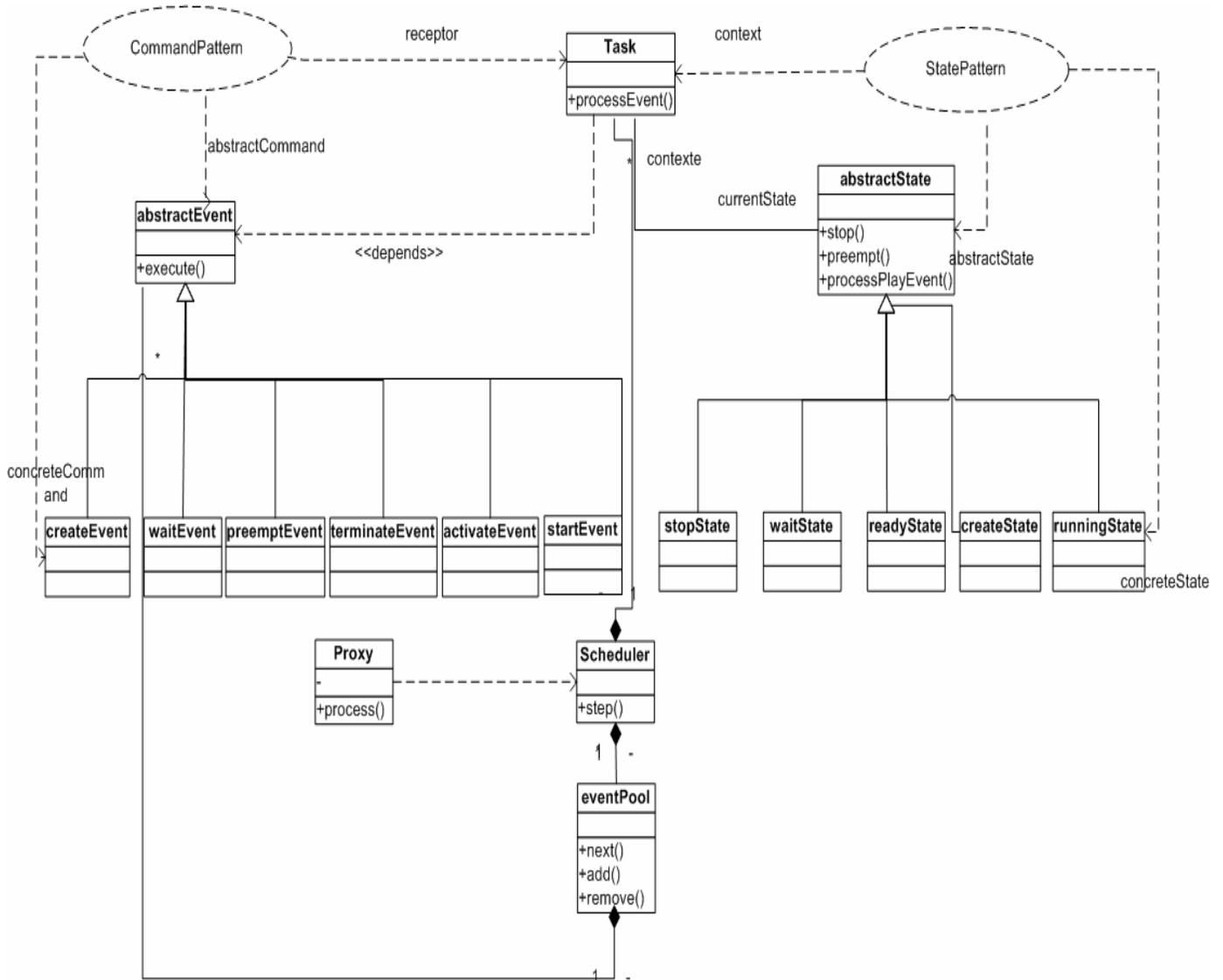


Figure 2: RTOS scheduler model

C. Specifying temporal constraints

In order to focus on software correction quality during model transformation, OCL is used to explain temporal constraints such as deadline and duration. Let us take the example of the attribute progress of the Task entity, whose value must always be lower than the deadline. This constraint is translated into OCL as shown in Figure 3.

```

context Task ::activate()
inv : periode = deadline
and Ci < deadline
and progress <= duration
pre: state = 'ready'
post: progress > progress@pre
and state = 'running'
    
```

Figure 3: Example of Temporal constraints specified with OCL

IV. CASE STUDY

The objective of this step consists in transforming an XML (Extensible Markup Language) source model obtained automatically from an UML source model to an XML target model. The model transformation is based on ATL (ATLAS Transformation Language) [5]. To describe the model transformation, the KM3 (Kernel MetaMetaModel) language is used. It makes it possible to define models according to meta-model MOF (Meta Object Facility) in a textual form.

The source model transformed corresponds to the diagram of class presented by Figure 1. The code corresponding to XMI (XML Metadata Interchange) based on XML offers a tree structure to our model by presenting the classes and the attributes in textual format.

To do these transformations, four tasks are taken with various characteristics. The scheduling of these tasks is made according to the scheduling algorithm Rate Monotonic [3].

The four tasks are specified in ecore format as shown in Figure 4.

```
<Task idTask="T1" taskState="Create" dateFirstActivation="0"
periode="6" Ci="1"
<Task idTask="T2" taskState="Create" dateFirstActivation="0"
periode="10" Ci="3" >
<Task idTask="T3" taskState="Create" dateFirstActivation="0"
periode="15" Ci="2">
<Task idTask="T4" taskState="Create" dateFirstActivation="0"
periode="20" Ci="3">
```

Figure 4: Tasks example

The model transformation consists of the following three major stages:

- Stage 1 (Scheduling analysis): It checks the schedulability conditions
- Stage 2 (Initialisation): It puts all task instances in the event pool and initializes the tasks and events states
- Stage 3 (Scheduling): It ensures tasks scheduling through time progressing

A. Scheduling Analysis

The basic schedulability conditions for Rate Monotonic were derived from a set of n independent periodic tasks with a fixed priority. A set of tasks is schedulable by the RM algorithm if (1) is verified where T_i is the task period, n is the number of tasks and C_i is the worst case execution time.

$$U = \sum_{i=1}^n \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1) \quad (1)$$

Using ATL, Equation (1) is specified as shown in Figure 5:

```
helper context RTOSStructure!Task1 def: Somme():
String=
RTOSStructure!Task1.allInstances()-
>collect(e|e.Ci.toInteger()/e.periode.toInteger())-
>sum()->toString();
helper context RTOSStructure!Task1 def:
ContraireRM(x:String):String=
x.toInteger()*2.exp(1/x.toInteger()-
1).toString();
helper context RTOSStructure!Task1 def:
ConditionRM(x:String):Boolean=
if self.Somme().toReal()<=
self.ContraireRM(x).toReal()
then true
else false
endif;
```

Figure 5: Schedulability conditions explained with ATL

B. Initialization

After checking schedulability condition, it is important to take instances of all classes of target model. The rule called Task2Task aims to transform Task source model elements to Task target model elements. It is described by Figure 6:

```
rule Task2Task{
from
s : RTOSStructure!Task1
to
w : RTOSSchudeler!Task (
idTask <- s.idTask,
taskState <- s.taskState ,
priority <- s.priority,
dateFirstActivation <-
s.dateFirstActivation,
deadline <- s.deadline,
duration <- s.duration,
periode <- s.periode,
Ci <- s.Ci,
Tsi <- s.Tsi,
progress <- '0')
}
```

Figure 6: Tasks initialization

After transforming all instances of the task source model into the target model one, it is important to use a helper named isCreate which selects all tasks with created state. In the same way, four other helpers are implemented in order to cover the other task states. IsCreate helper is represented by Figure 7.

```
helper context RTOSStructure!Task1 def:
isWait:Boolean=
if self.taskState='Wait'
then true
else false
endif;
```

Figure 7: isCreate Helper

The previous helper is used via a rule called Task2CreateState. The rules «task2WaitState», «task2RunningState », « task2StopState », «task2ReadyState » are implemented with the same manner as shown in Figure 8.

```
rule task2WaitState{
from
s : RTOSStructure!Task1 (s.isWait)
to
w : RTOSSchudeler!WaitState
(idTask<-s.idTask
)
}
```

Figure 8: Task2WaitState helper

After initializing all the tasks and putting them in the eventPool, the four tasks must be scheduled using Rate Monotonic algorithm with a fixed priority. These tasks are scheduled according to each time unit. To do that, it is important to follow the following steps:

For each moment, the file message Eventpool gives an idea about the existing tasks, their current attributes, specially the progress and the state ones.

C. Scheduling

According to Rate Monotonic, the task with the highest priority is extracted from the event pool. To find the highest priority, all the instances of this class are collected in a

sequence. These instances are sorted using the instruction `asSet()`. This operation returns a set containing the elements of the self collection. Order is lost from a sequence or an ordered set. The helper that defines the highest priority is shown by Figure 9.

```

helper context RTOSStructure!Task1 def :Maxtasks
:String=
RTOSSchudeler!EventPool.allInstances()-
>collect(e|e.priority.toInteger()).asSet()->select(e
| e >= 0)->last()->toString();
helper context RTOSStructure!Task1 def : MaxTask
:String=
{if
self.priority.toInteger()==self.Maxtasks.toInteger()
then self.idTask
else false
endif};

```

Figure 9: Extracting task with highest priority using ATL

During the last step, the task with the highest priority is running while the other ones are waiting or blocked. So, the class `RunningState` of the target model is instanced. Thus, after selecting the highest priority, the concerned task is defined via a helper called `check2` and it is used in the following rule and represented by Figure 10.

Finally, after executing all the necessary rules, the target model is written in `ecore` format. It contains the four tasks scheduled according to Rate Monotonic algorithm. This file can be translated to any specific platform, i.e., to any computer programming language.

```

rule RTOSModelingt(
from
s : RTOSStructure!Task1(s.check2)
to
w : RTOSSchudeler!shudeler(
idTask<-s.idTask,
priority<-s.priority,
dateFirstActivation<-s.dateFirstActivation,
deadline<-s.deadline,
progress<-'2'
),
c: RTOSSchudeler!RunnigState(
idTask <- w.idTask,
priority<-s.priority,
dateFirstActivation<-
s.dateFirstActivation,
periode <- s.periode,
duration<-s.duration,
deadline<-s.deadline)
)
helper context RTOSStructure!Task1 def :check2
:Boolean=
if self.priority ==self.MaxTasks
then true
else false
endif;

```

Figure 10: Instantiation of `RunningState` class

V. CONCLUSION

MDE is a well-known technique that has been successfully applied in ERTS design, especially for hardware and application modeling. In this paper, the mentioned technique is used for integrating RTOS modeling in high abstraction level. At this level, concepts undergo abstraction and are

independent of realisation and specific platform execution. The major contributions are the definition of independent platform RTOS models and the production of the code that ensures tasks scheduling.

MDA pattern based on statecharts implementation technique is very effective since it leads to the creation of scheduling model. During model transformation, scheduling model is considered as the target model while the structure model is the source one. The proposed approach is validated through a case study on Rate Monotonic algorithm.

Future work includes the focus on annotating used UML diagrams with reliability attributes to identify and recover system from failures.

REFERENCES

- [1] A. Cuccuru, C. Mraidha, F. Terrier, S. Gérard. Templatable Metamodels for Semantic Variation Points. *ECMDA-FA 2007*: 68-82
- [2] B. Combemale, S. Rougemaille, X. Crégut, F. Migeon, M. Pantel, C. Maurel. Expérience pour décrire la sémantique en Ingénierie des modèles. *IDM6 LILE 26 28 juin 2006*
- [3] C. L. Liu, James W. Layland, Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *Journal of the ACM (JACM)*, v.20 n.1, p.46-61, Jan. 1973
- [4] D. Thomas, C. Baron, B. Tondu. Ingénierie dirigée par les modèles appliquée à la conception d'un contrôleur de robot de service. *IDM6 LILE 26 28 juin 2006*.
- [5] Frédéric Jouault, Ivan Kurtev: Transforming Models with ATL. *MoDELS Satellite Events 2005*: 128-138
- [6] F. Chauvel and J. Jézéquel. Code generation from UML models with semantic variation points. In S. Kent L. Briand, editor, *Proceedings of MODELS/UML'2005*, volume 3713 of LNCS, pages --, Montego Bay, Jamaica, October 2005. Springer
- [7] Gamma, Erich, Helm, Richard, Johnson, Ralph, and Vlissides, John. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman Publishing Co., Inc., 1995.
- [8] G. Pinter and I. Majzik. Impact of Statechart Implementation Techniques on the Effectiveness of Fault Detection Mechanisms, *Proceedings of the 30th EUROMICRO Conference (EUROMICRO'04)*. 1089-6503/04 IEEE
- [9] Harel, David and Naamad, Amnon. The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293-333, October 1996.
- [10] Ji Chan Maeng, Dongjin Na, Yongsoon Lee, Minsoo Ryu: Model-Driven Development of RTOS-Based Embedded Software. *ISCIS 2006*: 687-696
- [11] L. Gomes, A. Costa. From Use Cases to System Implementation: Statechart Based Co-design, *Proceedings of the First ACM and IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE'03)*. ISBN 0-7695-1923-7/03 2003 IEEE.
- [12] M. Cruz Valiente, G. Genova, J. Carretero. UML 2.0 Notation for Modeling Real Time Task Scheduling. *Carlos III University of Madrid JOURNAL*
- [13] OMG Document Number: ptc/07-08-04. A UML Profile for MARTE, Beta 1 OMG Adopted Specification, August 2007
- [14] Peng Yang, Francky Catthoor: Dynamic Mapping and Ordering Tasks of Embedded Real-Time Systems on Multiprocessor Platforms. *SCOPES 2004*: 167-181

- [15] S. Bernardi and D. Petriu. Comparing UML Profiles for Non-functional. Requirement Annotations: the SPT and QoS Profiles, SVERTS 2004
- [16] Shourong Lu; Halang, W.A.; Gumzej, R. Towards platform independent models of real time operating systems. Industrial Informatics, 2004. INDIN '04. 2004 2nd IEEE International Conference on Date: 26-26 June 2004, Pages: 249 - 254
- [17] S. Kholgade, J.White, H. Reza: Comparing the Specification of a Near-Real Time Commanding System Using Statecharts and AADL. ITNG 2007: 355-360

Efficient Component-Based Face Recognition System for High Speed Devices

Naser Zaeri

Electronic and Electrical Eng.
Dept.
University of Surrey
GU2 7XH Guildford
United Kingdom
n.zaeri@surrey.ac.uk

Josef Kittler

Electronic and Electrical Eng.
Dept.
University of Surrey
GU2 7XH Guildford
United Kingdom
j.kittler@surrey.ac.uk

Abdallah Cherri

Electrical Engineering Dept.
Kuwait University
P.O.Box 5969 Safat
Kuwait
cherri@eng.kuniv.edu.kw

Abstract – In this paper, a new approach for face recognition using component-based linear discriminant analysis is proposed. The proposed method may find many applications in systems requiring very high recognition rate, high speed and low memory. Component-based approaches for face recognition have gained interests in order to compensate for face changes and other disadvantages of holistic methods. A methodical study and comprehensive experiments relating time consumption versus system performance are presented. This study relates the three factors (component size, number of components, and component location) on one hand, with the performance of the system and its speed on the other hand. Further, we demonstrate a solution to the problem of face occlusion using this method. The experimental results show that the proposed method enhances the performance of the system and achieves a substantial saving in the computation time when compared to other known methods.

Index Terms – Face recognition, high speed devices, component LDA, system performance

I. INTRODUCTION

Face recognition has received extensive attention because of its significant applications in many fields, ranging from biometrics, information theory, law enforcement and surveillance to smart cards and access control. Accurate automatic personal identification is now needed in a wide range of civilian applications involving the use of passports, cellular telephones, automatic teller machines, and driver licenses. Moreover, in video processing and internet applications, face image retrieval is an important factor for identification and verification purposes [1].

Despite the great advances in this field of research, face-based identification still poses many challenges because of the changes in viewpoint, colour, and illumination. Further, the majority of the systems designed to date are not robust to occlusion. Another important concern is that the feature representations of most of these methods are of high dimensionality. This last important disadvantage makes them inadequate for the implementation in systems that require efficient storage and high speed processing, such as cell-phone cameras and video surveillance systems. One of the important reasons beyond these disadvantages is that

most of the proposed systems are global or holistic approaches. In other word, they deal with the face image as a whole.

In order to compensate for face changes and other disadvantages of holistic methods, recently, researchers have focused more on component-based approaches to use flexible relations between the face components [5, 6, 7, 8]. The ability of component-based methods to not require a perfect view of all facial features offers many benefits. Nevertheless, there are still many shortages in this track of research. The processing and computational time of proposed methods are still not enough attractive to be implemented efficiently by limited memory and high speed processing devices' manufacturers. Most of these methods require many pre-processing steps and use large image component size, which yield many deficiencies. In addition, the selection of the components of other methods is done manually around pre-selected points on the face.

In general, component-based approach passes through two stages. In the first stage, the local features of the face image that comprise most of the importance are found using a set of characteristics and special well-defined criteria or function. Next, these local features are combined at a second stage to decide whether the input face image belongs to a given class. One of the main objectives of the component-based approach is to find the best set of components, including their locations, their numbers, and their sizes that can classify and identify the face image.

Ullman *et al.* [6] used both whole face images at intermediate resolution and local regions at high resolution for face verification, where components of various sizes were cropped at random locations of the images. Amit and Geman [9] employ small, localized and oriented edges and combine them with decision trees. Weber *et al.* [10] use localized image patches and explicitly compute their joint spatial probability distribution. Heisele *et al.* [7] introduced an algorithm that learns rectangular facial components around pre-selected points on the face, and each component is grown iteratively, and consequently it was computationally costly. Heisele *et al.* in [8] have restricted the location of the components to be within a pre-defined search region and they add the position of the detected pre-defined components as an additional input to the classifier. Using linear and second-degree

polynomial SVMs classifiers in their system makes their system computationally exhaustive. Kim *et al.* [5] apply component-based LDA onto facial components and offer a good representation for MPEG-7 face description.

In this paper, we present a novel method that aims to solving some of the problems mentioned above. Specifically we show that our proposed system is of high speed and it is attractive for domains demanding strict timing requirements with efficient recognition and verification rates, when compared to other systems. We present a study that relates the three factors (number of components, their sizes, and their locations) on one hand, with the performance of the system on the other hand. Further, we present a study and comprehensive experiments showing the relationship between the performance of the system and the time consumed by it. Furthermore, we will show that the proposed technique provides a solution to the occlusion problem in face recognition. From the above mentioned works, the closest one to what we are proposing is presented by Kim *et al.* [5]. However, in their study they do not explain how the facial components are chosen. Further, no criteria are mentioned on how to choose the *size* of component that should be used, or how many components are needed, or the proper *location* of these components. Therefore, we try to find the *best* component-based LDA for face representation by finding the best *size*, the *number*, and the *location* of components necessary to describe the face image efficiently. Moreover, the training time consumed by the method presented in [5] is very large as we will demonstrate in our experiments. This problem is efficiently overcome in our new proposed method. The extensive experimental results of testing and implementing the proposed approach are presented on two independent databases; the Olivetti Research Ltd. (ORL) database and the xm2vts, CVSSP – University of Surrey database. The organisation of the paper is as follows. Section 2 presents a brief mathematical background about the linear discriminant analysis (LDA) technique, since it is used in our proposed method. Section 3 discusses our proposed novel component-based method. Section 4 presents the experimental results and explores the relationship between the performance of the system and the time consumed by it. Finally, concluding remarks are given in Section 5.

II. LINEAR DISCRIMINANT ANALYSIS

The LDA has been exhaustively used in face recognition due to its optimal performance and time-efficient matching for multi-class face recognition. LDA emphasizes variations among different classes while the variations of the same class are de-emphasized [3, 4]. Consequently, the LDA method finds a direction on which data are well class-wise clustered. Assume that a data matrix $\chi = \{x_1, x_2, \dots, x_M\} \in \mathfrak{R}^{N \times M}$ is given, where $x_i \in \mathfrak{R}^N$ is a

N -dimensional column vector representing the face image and M is the number of images. Each vector belongs to one of C object classes $\{\chi_1, \chi_2, \dots, \chi_C\}$. Classical discriminant analysis aims to derive a transformation $W \in \mathfrak{R}^{N \times n}$ ($n \leq N$) which maps a vector x to $y = W^T x$, $y \in \mathfrak{R}^n$ such that the transformed data have maximum separation between classes and minimum separation within classes. The between-class S_B and within-class S_W scatter matrices in LDA for the c -class case are given respectively by

$$S_B = \sum_{i=1}^C n_i (m_i - m)(m_i - m)^T \quad (1)$$

$$S_W = \sum_{i=1}^C \sum_{x \in \chi_c} (x - m_i)(x - m_i)^T \quad (2)$$

where m_i denotes the class mean, m is the global mean of the entire sample set and n_i denotes the number of samples in class c . The attempt is to solve the generalised eigenvalue problem

$$S_B u_i = \lambda_i S_W u_i \quad (3)$$

by first projecting the image into a lower dimensional space using the PCA technique and then applying the LDA. This will ensure that the within class scatter matrix is non-singular. The optimal projection W_{opt} that maximises the ratio of the determinant of the between-class scatter matrix of the projected samples to the determinant of the within-class scatter matrix of the projected samples is found by the following

$$W_{opt}^T = W_{fld}^T W_{pca}^T \quad (4)$$

where

$$W_{pca} = \arg \max_W |W^T S_T W| \quad (5)$$

$$W_{fld} = \arg \max_W \frac{|W^T W_{pca}^T S_B W_{pca} W|}{|W^T W_{pca}^T S_W W_{pca} W|} \quad (6)$$

where S_T is the total scatter matrix.

III. COMPONENT-BASED RECOGNITION SYSTEM USING LDA AND SFFS

By considering local components of a face image (instead of the entire face image) for recognition, we may achieve higher correct recognition rates. Since the image variations on the component level is limited when compared to the variations within the entire image. Thus, by adding the efficient use of the LDA method to the components makes the component-based LDA approaches advantageous for many applications, such as:

- 1) **Time efficient applications:** our experiments show that the training time required for our proposed component-based method is much smaller than the others, as will be shown.
- 2) **Occlusion problem:** when a targeted image is occluded, we can divide the image into a number of blocks in order to exclude the occluded part of that image and consequently the recognition will be based on the other components, as will be shown shortly.

The most important issues of face recognition at the component level are finding the best size, location, and number of these components. Selecting the right size of a component is a difficult task and could depend on many factors such as: the number of images in the database, the size of these images, the variations in the input images, the required identification speed, and the required recognition rate. These factors influence the decision of choosing the best component *size*, *number*, and *location*, which are not independent. Fig. 1 shows our proposed component-based LDA method considering the above mentioned factors. A face image in the database is partitioned into a number (L) of components with an initial predefined size. The size and the necessary number of these components are then adapted and optimised by the algorithm that automatically finds the best presentation of the face image and consequently the best performance. These important components are automatically found through a search of the best number of sub-images (components) that can describe and eventually recognise the face using the Sequential Floating Forward Search (SFFS) algorithm [2].

The L components of the image undergo LDA transformations. In a similar way to the training images, the test (query) image is divided into L components and their LDA transformations are performed. Now, for every component of the test image we find the distance between its LDA transformation and the corresponding training LDA transformation.

In mathematical words, given a set of M training images $\{x_1, x_2, \dots, x_M\}$, then a set of LDA transformation matrices is extracted for each of them. First, all the images are partitioned into L facial components. The images (patches) of each component are represented in a vector form with the k^{th} component being denoted as $\{c_1^k, \dots, c_M^k\}$. Then, for the k^{th} facial component, the corresponding LDA transformation matrix W^k is computed. During testing, the L vectors $\{c^1, \dots, c^L\}$ that correspond to the facial component patches are extracted from a face

image x_i of the test data set. Next, a set of LDA feature vectors $y = \{y^1, \dots, y^L\}$ is extracted from the test image x_i using the corresponding LDA transformation matrices as

$$y^k = (W^k)^T c^k, \quad k = 1, \dots, L \quad (7)$$

Thus as shown in Fig. 1, a face image x_i is represented by a set of LDA feature vectors $\{y^1, \dots, y^L\}$. We have used the Euclidian distance in finding the minimum distance.

At this stage, we have a candidate for each component representing the identity of the test image. Finally, we find the maximum number of the components scoring the same candidate and recognise the test image as that person. This is basically a fusion process where we fuse the information coming from each component to produce a single score that represents the combined decisions of the components result. The fusion method that is used is the *Voting* method. This method outputs a score equal to the number of component classifiers that output scores above their respective thresholds. For example, if we have five classifiers and three of these classifiers give a score belonging to identity A , while the fourth classifier suggests identity B and the fifth classifier gives a result in favour of identity C , then based on the *voting* method the system decides that the image belongs to identity A .

As we mentioned earlier, the SFFS method is used in our method to find the best features since it was found to be one of the best feature selection methods [2]. The proposed system shown in Fig. 1 uses the SFFS algorithm to find the best location of a certain number of components that can give the highest possible recognition rate. If a pre-defined recognition rate is not achieved, the SFFS algorithm increases the number of components and the process is repeated again. In this regard, the SFFS algorithm includes new best features (here the feature is the component) that when added to the current feature set, the error rate is minimized. In addition, the SFFS algorithm excludes the worst feature during the selection process in order to further improve the selection of the feature set. Floating search methods of feature selection provide a performance that has been found to be very good compared with other suboptimal search methods and they are computationally much more efficient than the optimal method [2]. Briefly, the component selection using SFFS procedure can be explained as follows.

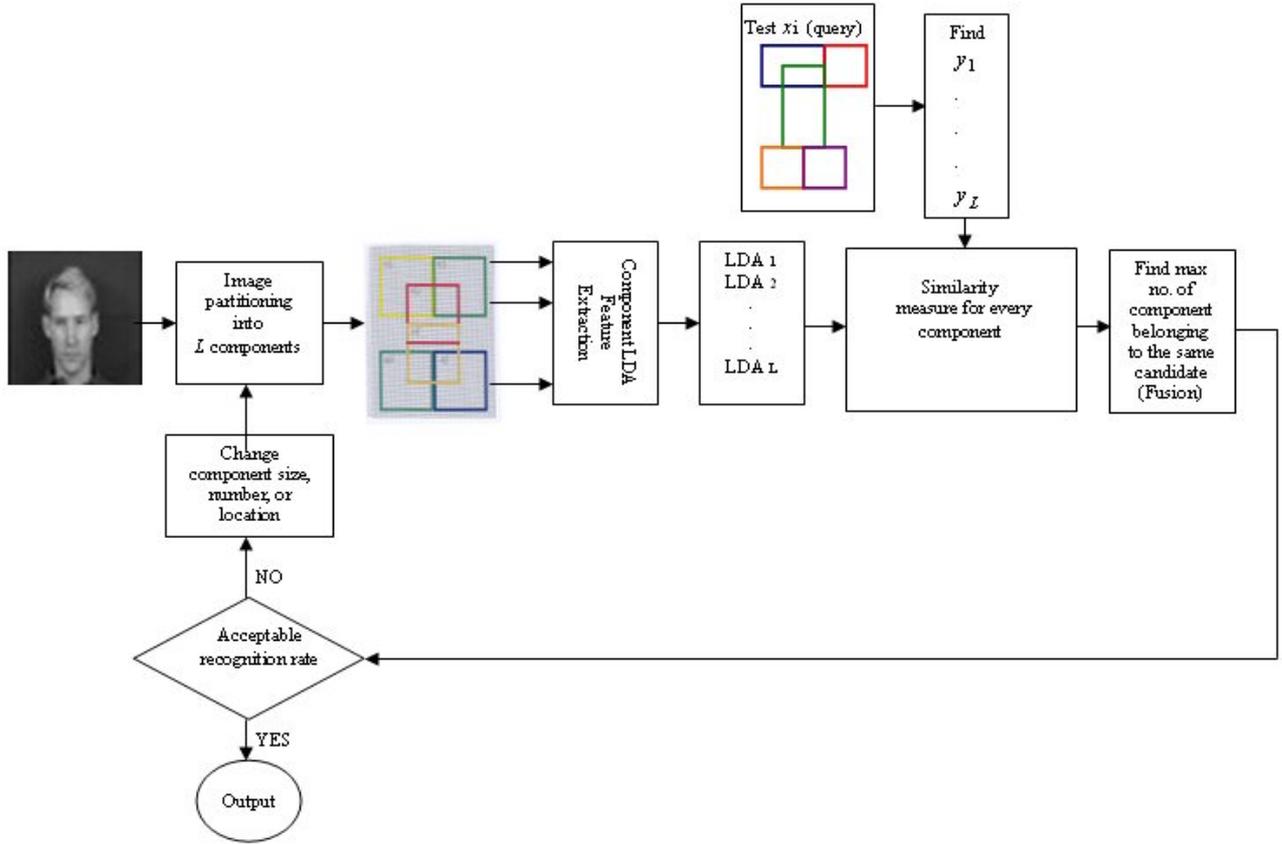


Figure 1. The proposed component-based LDA system for choosing the components size, number, and locations.

Suppose k features (components) have already been selected from the complete set of measurements $Y = \{y_j | j = 1, 2, \dots, D\}$ of D available features to form the set $X_k = \{x_i : 1 \leq i \leq k, x_i \in Y\}$ with the corresponding criterion function $J(X_k)$. In addition, the values of $J(X_i)$ for all preceding subsets of size $i = 1, 2, \dots, k - 1$, are known and stored.

Step 1. Select feature x_{k+1} from the set of available measurements, $Y - X_k$ to form feature set X_{k+1} such that $J(X_k + x_{k+1})$ is minimised with respect to x_{k+1} .

Step 2. Find a feature, \hat{x}_{k+1} , in X_{k+1} the removal of which will minimise $J(X_{k+1} - \hat{x}_{k+1})$ with respect to \hat{x}_{k+1} .

Step 3. If $x_{k+1} = \hat{x}_{k+1}$, then $k = k + 1$ and return to Step 1.

Step 4. If $x_{k+1} \neq \hat{x}_{k+1}$, then $\hat{X}_k = X_{k+1} - \hat{x}_{k+1}$.

Step 5. If $k = 2$, then $X_k = \hat{X}_k$, and return to Step 1.

Step 6. Find a feature, \hat{x}_r , in \hat{X}_k , the removal of which will minimise $J(\hat{X}_k - \hat{x}_r)$.

Step 7. If $J(\hat{X}_k - \hat{x}_r) > J(X_{k-1})$, then $X_k = \hat{X}_k$, and return to Step 1.

Step 8. If $J(\hat{X}_k - \hat{x}_r) \leq J(X_{k-1})$, then $X_{k-1} = \hat{X}_k - \hat{x}_r$, $k = k - 1$.

Step 9. If $k = 2$ return to Step 1, otherwise return to Step 6.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

We carried out experiments on two independent and different databases; the ORL and the xm2vts. Both sets include a number of images for each person, with variations in pose, expression and lighting. The ORL set includes 400 images of 40 different individuals where each individual is represented by 10. The system was trained using 4 images for each person from this set. The xm2vts set have 2360 images for 295 different individuals with each individual represented by 8 different images, where we have trained our system with 4 images [11]. Figures 2 and 3 show examples of the images from the xm2vts and the ORL databases used in the experimentation of our system, respectively. Example of an image and some of its corresponding components taken from different parts of the image are shown in Fig. 4. These components are candidates for

the system to choose the best ones that can give the best performance among the others, as it was explained in section 3.



Figure 2. Different images from 4 different sessions (each session with 2 shots) for a certain individual taken from the xm2vts database.



Figure 3. Examples of the images from the ORL databases.

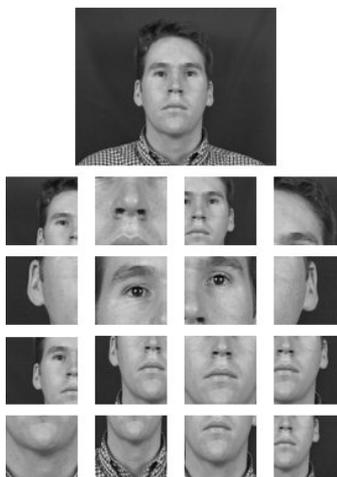


Figure 4. A demonstrative example of different components that correspond to a face image.

A. Occlusion Problem

As a first experiment, 4 components ($L = 4$) each of size 30×30 from the xm2vts database (2360 images) is used as shown in Fig. 5. The component-based LDA finds the individual recognition rate of each component, and then fuses the information of each component to produce a single *score* rate that represents the combined decisions based on a *voting* method. The final combined rate is shown in Table 1. It is worth mentioning that

although an excellent successful recognition rate is achieved, the time consumed in the training stage is very large (48.7 minutes). As such, we reduced the size of the components to 25×25 and the corresponding training time eventually reduced to 36 minutes. Nevertheless, this is still unsatisfactory for many practical implementations. Therefore, in the next subsections we will keep reducing the size of the components and at the same time try to keep the recognition rate as high as possible.

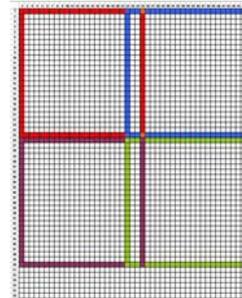


Figure 5. Example of four components face image.

Table 1. The results of the experiments related to Fig. 5 (4 components).

Comp. size	Success Rate	Training Time (minutes)
30×30	99.5%	48.7
25×25	99.2%	36

As a demonstration of the strength of our proposed technique, we have applied it to the occlusion problem in face recognition. If a targeted face image is occluded, as shown in Fig. 6, our approach achieves acceptable recognizing rate for this image. For the quarter-part occluded images applied to the xm2vts database, we have reached an ~93% successful recognition rate. When we applied our method to the half-part occluded images, we have achieved an ~80% successful recognition rate. Note that the occluded images are used for the testing ones.



Figure 6. Original, one-quarter, and one-half occluded image.

B. The Component Size and Location

Table 1 reveals the important fact that the performance, the speed of the recognition system, and the size of the components are closely related. For many practical applications, the speed of recognition is crucial. To achieve this objective our method reduces the size of the components as much as possible in order to decrease the amount of calculation needed. As a result, the issue of the component location arises, since

for smaller component sizes, empty uncovered regions of the images will be present. Further, these components should cover the *most important* regions of the face image. In this regard, we have reduced the image size to 20×20 using 4 components. Some initial selections of the 4 components that are proposed by the SFFS algorithm are shown in Fig. 7. Table 2 shows some examples of the results (the results correspond to xm2vts database). The recognition rate for an individual component is found to be above 50%. However, the overall success recognition rate for the whole component analysis approach can reach 81%. Note that the component number shown in Table 2 represents a specific case of a combined components layout which is generated by the SFFS algorithm. It is clear that some parts of a face are more descriptive and discriminative and need to be considered when generating face components. This emphasizes the importance of the location of the components that our approach focuses on.

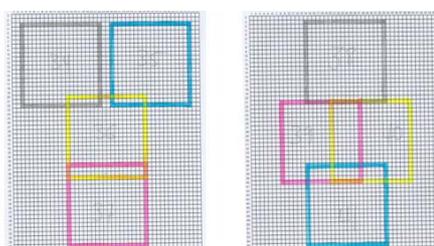


Figure 7. Some combinations of the four components face image of size 20×20.

Table 2. Sample results (from the xm2vts database) of the experiments (components) for images in Fig. 7.

Comp.	Success %	Fusion Rate	Comp.	Success %	Fusion Rate
34	52.4	81.4	38	51.2	76.6
35	55.4		39	56.6	
36	71.0		40	58.5	
37	63.4		41	63.4	

On the other hand, to further emphasize the important issue of the size of the selected components in a face image and in order to reach some conclusions, we have decreased the size of a component to 15×15. Figure 8 shows a sample of the component. The performance has dropped significantly (33.22% success rate was obtained). Increasing the number of components and keeping the component size unchanged did not give promising results. The highest rate that we have obtained was 51.6 % success rate for the case of 14 components.

Further, the experiments have demonstrated that increasing the component size has much better effect on the performance than increasing the number of the components. This was demonstrated in Table 1 for the cases of 30×30 and 25×25, and Table 2 for the case of 20×20 component size. However, the improvement of the performance will be obtained at the expense of speed and time efficiency. The execution time for the 4

components of size 20×20 is higher than the corresponding time for the 5 components of size 15×15.

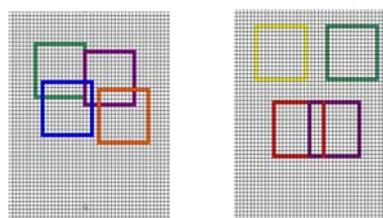


Figure 8. Four components face image of size 15×15.

C. Number of Component vs. Computation time and Performance Rate

The results of the previous experiments indicate that the number of components should be increased in order to improve the recognition rate and at the same time keep the components at a reasonable size. By increasing the number of the components, a higher overall recognition success rate is achieved. Fig. 9 shows a comparison between the recognition rates for different number of components (all of size 20×20). Note that by using 14 components, a success recognition rate of 98.5% is achieved.

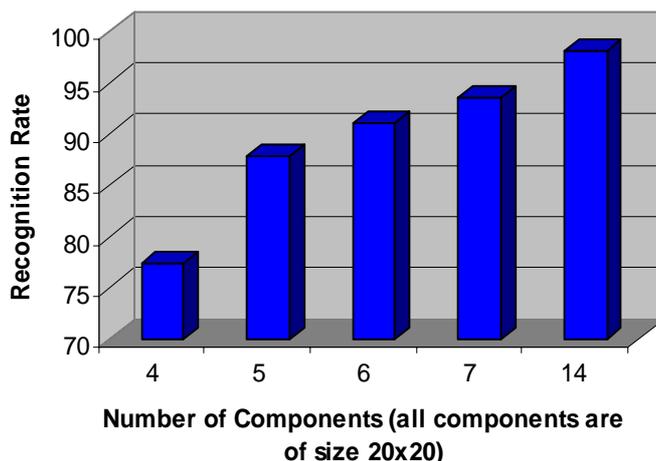


Figure 9. A comparison between the recognition rates for different number of components using the xm2vts database.

Table 3 shows the computational time (using 2.8 GHz – 512k RAM machine) needed to train the recognition system for different number of components and for the method presented in [5], when applied to the xm2vts database using 2360 images. Our method gives a success rate of 98.5% and needs a training time of only ~27 minutes, for the 14 components case; whereas the method proposed in [5] takes more than 134 minutes to achieve the corresponding high rate. In other word, our technique performs 5 times faster than the one proposed in [5]. Note that, although the 30×30 four-components method achieves 99.5% success rate, it takes almost 49 minutes to train the system in order to accomplish its task. Whereas the 20×20 14-components experiment achieves 98.5% success rate and needs only 27 minutes. It is important to mention that the computation times include the computation of LDA matrices.

Table 3. The training time consumed (in minutes) and the success recognition rate for the different number of components and the component-based method described in [5] using 2360 images of the xm2vts database.

	4 comp. -size 30x30	4. comp. -size 25x25	4. comp. -size 20x20	5 comp. -size 20x20	6 comp. -size 20x20	7 comp. -size 20x20	14 comp. -size 20x20	5 Comp. [5]	14 Comp. [5]
time (min)	48.7	36	10.1	14.5	16.8	19.3	27	112.1	134.6
Success Rate %	99.5	99.2	80	86.9	91.1	92.4	98.5	Not mentioned	98.5

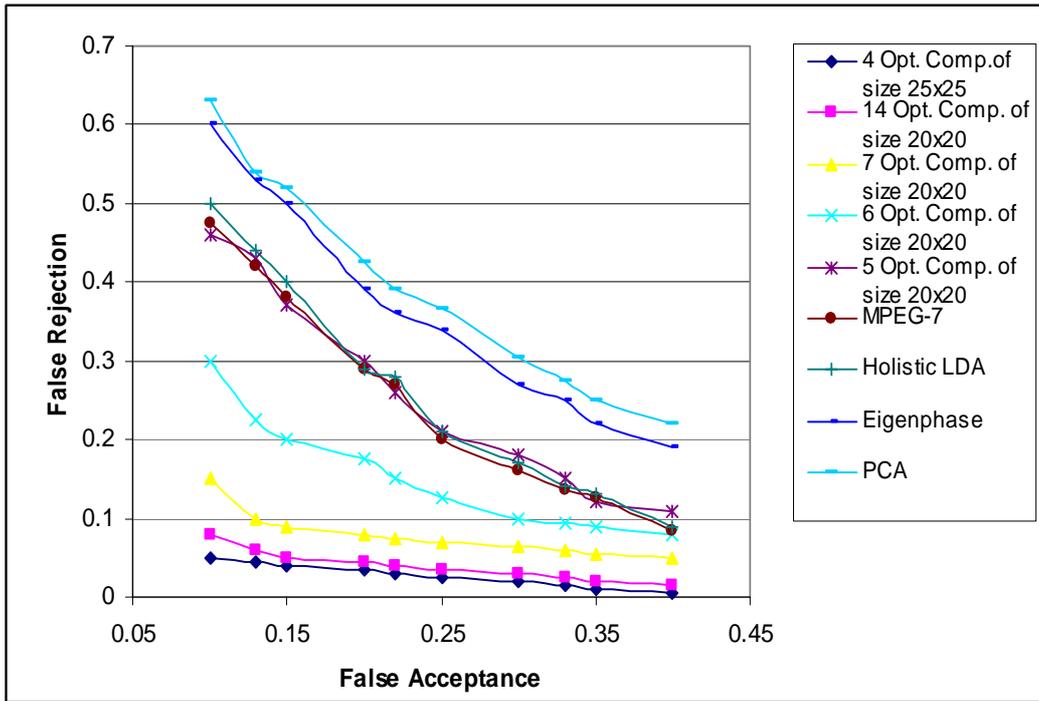


Figure 10. Comparison of the ROC curves of various methods for the ORL database.

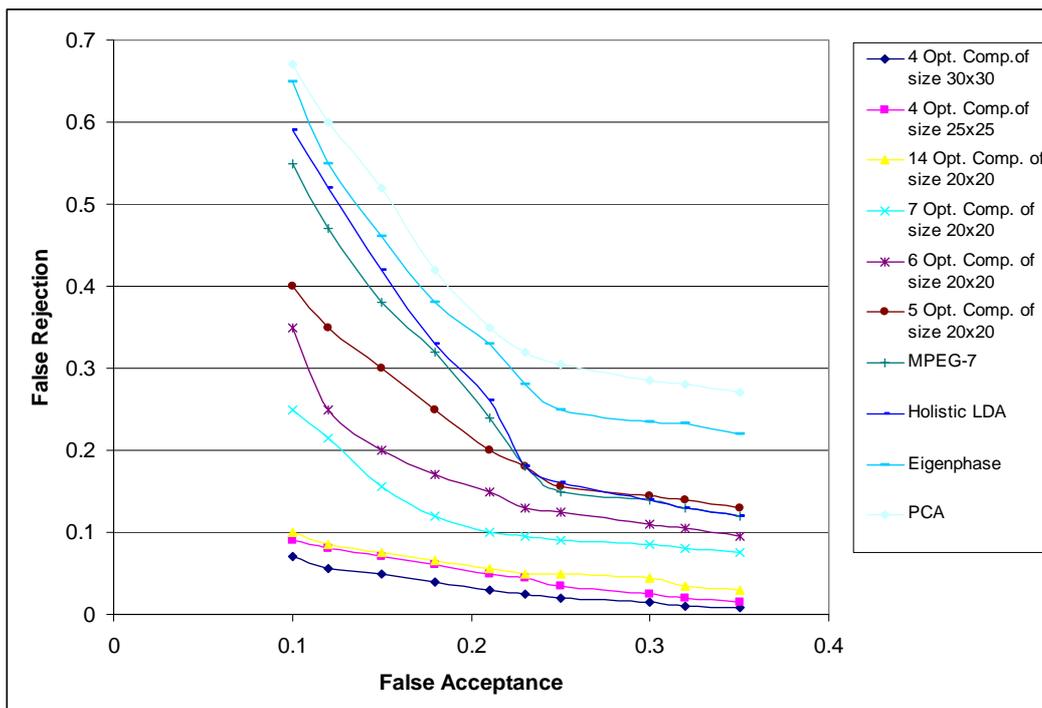


Figure 11. Comparison of the ROC curves of various methods for the xm2vts database.

In addition, the Receiver Operating Characteristic (ROC) curve is used to assess the performance of the proposed method and compared to different other techniques. The ROC curves for the ORL and xm2vts databases are shown in Figures 10 and 11, respectively. It is worth mentioning that these curves were also calculated for MPEG-7, Holistic LDA, Eigenphases, and PCA techniques as shown in the figures. From the Figures, we can comfortably state that the performance of our system is robust.

V. CONCLUDING REMARKS

In this paper, a new approach using the component-based LDA for face representation was presented. Our new approach relates the size, the number, and the location of the components in order to describe the face image efficiently, and consequently achieve the best recognition rate within the minimum possible amount of training time. We have experimentally demonstrated the existence of a trade-off between the component size, the number of such components, the processing time, and the recognition rate. In addition, thorough experiments have been conducted in order to decide a suitable component size which can give high performance and at an acceptable training time. For the images used in our databases (size of 56×46), it has been found that component size of less than 20×20 would not give good recognition rates. Moreover, we experimentally demonstrated that increasing the component size reflects directly on the processing time. In other word, a trade-off between the component's size and the time to process them must be decided. In addition, increasing the number of components increases both the recognition rate and the processing time. Moreover, our proposed method was compared to other published well known methods (PCA, Holistic LDA, Eigenphase, and MPEG-7) and it was shown that we have achieved much better performance. As such, the new technique can find applications where real-time processing and high speed performance are required.

REFERENCES

- [1] W. Zhao, R. Chellappa, P.J. Phillips, and A. Rosenfeld, "Face Recognition: A Literature Survey", *ACM Computing Surveys*, Vol. 35, No. 4, pp. 399 – 458, Dec. 2003.
- [2] E. C.-Paz, S. Newsam, and C. Kamath, "Feature Selection in Scientific Applications", *Proceedings of the Tenth ACM International Conference on Knowledge Discovery and Data Mining*, pp: 788 - 793, 2004.
- [3] K. Etemad and R. Chellappa "Discriminant analysis for recognition of human face images", *J. Opt. Soc. Am. A/Vol. 14, No. 8 pp. 1724-1733 - August 1997*.
- [4] P. Belhumeur, J. P. Hespanha, and D. J. Kriegman, "Eigenfaces vs. Fisherfaces: Recognition Using Class Specific linear Projection", *IEEE PAMI*, Vol. 19, No. 7, pages 711-720, July 1997.
- [5] T. Kim, H. Kim, W. Hwang, J. Kittler, "Component-based LDA face description for image retrieval and MPEG-7 standardization", *Image and Vision Comp.- Elsevier*, 2005.
- [6] S. Ullman, M. Vidal-Naquet, and E. Sali, "Visual Features of Intermediate Complexity and Their Use in Classification," *Nature Neuro-science*, No. 5, pp. 682 – 687, 2002.
- [7] B. Heisele and T. Koshizen, "Components for Face Recognition", *Conference on Automatic Face and Gesture Recognition, Seoul, Korea*, pp.153-158, 2004.
- [8] B. Heisele, T. Serre, and T. Poggio, "A Component-based Framework for Face Detection and Identification," *International Journal of Computer Vision*, November 2006.
- [9] Y. Amit and D. Geman, "A Computational Model for Visual Selection", *Neural Computation*, 11(7): 1691 – 1715, 1999.
- [10] M. Weber, M. Welling, P. Perona, "Unsupervised Learning of Models for Recognition", *In ECCV*, pp. 18 – 32, 2000.
- [11] K. Messer, J. Matas, J. Kittler, J. Luettin, G. Maitre, "XM2VTSDB: The Extended M2VTS Database", *Second International Conference on Audio and Video-based Biometric Person Authentication, Washington D.C.*, 1999.

Dataflow design of a co-processor architecture for image processing

Richard Thavot*, Romuald Mosqueron*, Mohammad Alisafae*,
 Christophe Lucarz*, Marco Mattavelli*, Julien Dubois†, Vincent Noel‡

*Ecole Polytechnique Fédérale de Lausanne (EPFL), GR-LSM, CH 1015 Lausanne Switzerland

Email: richard.thavot@epfl.ch

†Université de Bourgogne, Laboratoire LE2I, 21000 Dijon, France

‡AKAtech SA, Ecublens, Switzerland

Abstract—This paper presents the comparison of two design methodologies applied to the design of a co-processor dedicated to image processing. The first methodology is the classical development based on specifying the architecture by directly writing a HDL model using VHDL or Verilog. The second methodology is based on specifying the architecture by using a high level dataflow language followed then by direct synthesis to HDL. The principle of developing a dataflow description consists on defining a network of autonomous entities called actors, which can communicate only by sending and receiving data tokens. Each entity in the process of consuming and generating data tokens performs completely independent and concurrent processing. A heterogeneous platform composed by a SW processor and the designed HW co-processor is used to compare the results of the designs obtained by the two different methodologies. The comparison of the results shows that the implementations based on the dataflow methodology, not only can be completed with an important reduction of design and development time, but also enable efficient re-design iterations capable of achieving performances, which are comparable in efficiency to design obtained by hand written HDL.

I. INTRODUCTION

Since the 70's, the design of digital systems has known a continuous evolution. The technology used to realize silicon has been continually improving and thus the number of transistors available for digital design on the same silicon area has been increased. On the same line the complexity of algorithms and of circuits has followed a similar trend. Nowadays, the circuits complexity available on a chip has also generated another phenomenon. Several processing units such as processors, FPGAs and DSPs are available on the same heterogeneous platform. Nowadays, one of the major challenges is how to exploit all the processing resources available on such platforms for implementing complex applications, but using only limited developments and design resources. In other words, design productivity and efficient usage of the platform processing resources are the fundamental challenges of current and next generation designs. VHDL was introduced 20 years ago to simplify the design of the logic circuits by raising the abstraction layer avoiding the designer to work at the gate level. The dataflow methodology described in this paper, is based on CAL language[1] and on the synthesis of HDL directly from the dataflow model abstraction layer. Its introduction has exactly the same objective: raise the

abstraction layer of a design. Thus, the designer should not have to care about most of the low level implementation issues present in VHDL or Verilog, but rather focus on higher level architectural issues such as how efficiently dataflow through the different architecture components and how to partition and map the algorithm/processing elements on the different components of actual heterogeneous platforms. The aim of this work is to show how CAL design methodology can lead to efficient hardware designs within a shorter development time and lower design resource usage than classical HDL methods. In doing so, the paper introduces the essential concepts and elements of the new dataflow methodology based on writing networks of CAL actors [1], [2], [3] and compares the results of a design case using the new approach and the classical development at VHDL level.

A heterogeneous platform composed by a SW processor and a HW co-processor is used for the comparison. An image processing application partitioned into the SW and HW components is designed using CAL and finally compared to the implementation obtained by a classical HDL approach [4] [5] [6]. The main novelty of the approach is the possibility of specifying both SW and HW components, using the same language CAL, and then to generate automatically VHDL or Verilog at RTL level. Different versions of CAL models have been developed in order to explore the achievable performances of the automatic HDL generation tool.

The paper is organized as follows. Section II presents the dataflow concept and the "modus operandi" of the CAL language. Then, the heterogeneous platform (i.e. the smart camera) and the co-processor unit are presented in details in section III. The CAL dataflow model of the co-processor is presented in section IV. The implementation results of the different versions of the CAL dataflow models and the comparison with the hand written version in HDL are presented in section V. Finally, conclusions and future works are reported in section VI.

II. MODELING DATAFLOW SYSTEMS USING CAL

There are many applications that fit well the semantics of dataflow systems. An example is multimedia systems with flowing streams of data within processing blocks. Developing true dataflow models of such systems using general purpose

programming languages or hardware description languages is possible. However, the genericity of concepts and operators of these languages make the description of the models more complicated. This implies that the models are harder and more time-consuming to create and manipulate. It may be better if they were modelled directly using a specific dataflow language.

A. CAL language

CAL Actor Language is a language based on the Actor model of computation for dataflow systems. It provides many natural concepts to facilitate modeling of those systems [1]. A dataflow model expressed in CAL is composed by a set of independent "actors" and their connection structure. It makes a network of actors. An actor is a stand alone entity which has its own internal state represented by a set of state variables and it performs computations by firing actions. It has a set of input and output ports through which it communicates with other actors by passing data tokens. An actor must have, at least, one action to do computations. Actions execute (or fire) based on the internal state of the actor and depending on the availability and values of tokens at the input ports. An action may consume tokens from inputs, may change the internal state of the actor, and may produce tokens at the outputs. Action execution is modeled as an atomic component which means that no other action, of the same actor, can execute while an action is executing or interrupting any executing action. CAL provides scheduling concepts to control the executions order of actions inside an actor. CAL actors can be combined into a network of actors to build larger systems called network of actors. This is achieved by connecting the input and output ports of actors together to define the communication structure of the model (Figure 1). These communication channels are constituted by FIFOs, which in the CAL computation model have infinite size. Using CAL, designers can only focus on the model-

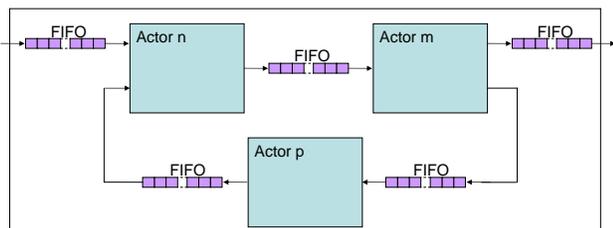


Fig. 1. A dataflow network with actors connected by FIFO channels.

ing of the dataflow system (actors and their communication topology) and do not need to care much about the low level of details to implemente the communication between actors (i.e. message passing protocols, queues, ...). The underlying computation model, simulation and synthesis system take care of all communication driven issues. However, it also provides to the designer the control over communication parameters such as length of queues and types of exchanged data. In this paper, we focus only on the issues related to the development

of a CAL model of HW accelerators and on the results of the implementations.

B. Workflow for CAL-based designs

One of the current challenges of designing embedded systems composed by mixed SW and HW components is the difficulty and the design efforts needed for specifying, modeling and implementing complex signal processing systems on a heterogeneous platform. CAL addresses this issue by unifying the hardware and software design and implementation process in a single flow. In a CAL-based design flow, the whole system is modeled and implemented in CAL. After that, designers can decide on HW/SW partitioning for the final implementation. A subset of the model can be used to generate synthesizable HDL code. The generated code can also be combined with existing HDL designs. Software can be generated in a similar manner based on the partitioning decision[3]. In such workflow, the partitioning between hardware and software can be easily modified since the same source is used for generating both parts. Figure 2 shows the complete CAL design flow for the implementation of a smart camera platform. It is composed of a general-purpose processor for running SW modules and a FPGA platform including specialized hardware.

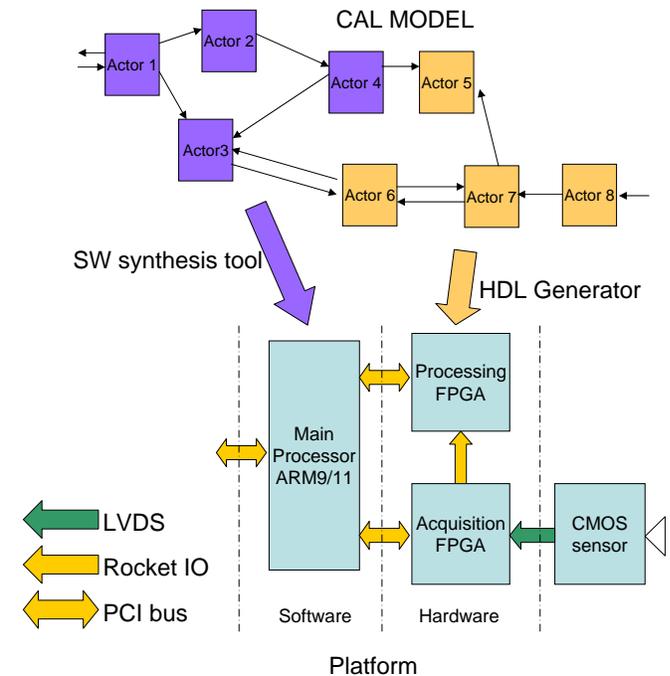


Fig. 2. Graphic representation on how partitions of the CAL dataflow models can be mapped on a heterogeneous SW and HW platform using synthesis tools.

III. SMART CAMERA PLATFORM

In this work, the test platform is a "smart camera" based on an embedded HW/SW co-processor designed and developed in a previous work [4]. Cameras with embedded co-processors enable the implementation of more powerful processing due

to the high degree of flexibility and to the clear task separation between the different units. The efficiency and the processing tasks have been tested and validated implementing a real application. This application is the detection and decoding of bar codes in a postal sorting application [5], described in more details hereafter. The whole co-processor has been specified and designed manually in VHDL. The high level of performance of the co-processor have been obtained exploiting the potential parallelism at the different stages of the processing. The communication and task controllers are rather complex due to the large variety of implemented functionalities. Therefore, obtaining an efficient model for direct HW synthesis by means of a high level dataflow description represents a real challenge. The system infrastructure of the SW and HW platform is presented below. The platform is composed of an embedded frame-grabber and is equipped, at different levels, of a processing unit for the image captured by the sensor. Figure 3 illustrates the main architectural components of the smart camera with embedded co-processor (Xilinx FPGAs) and processor (Nexperia). Two FPGAs are used to acquire and pre-process the image coming from the camera sensor. The main processor is in charge of the high-level processing tasks. The co-processor deals with the acquisition, pre-processing tasks specific to the application, and the lower-level tasks. These latter are characterized by processing regularity and a high level of parallelism. Figure 3 illustrates the main architectural components of the smart camera with the embedded co-processing stage.

In this section, the platform communication infrastructure is

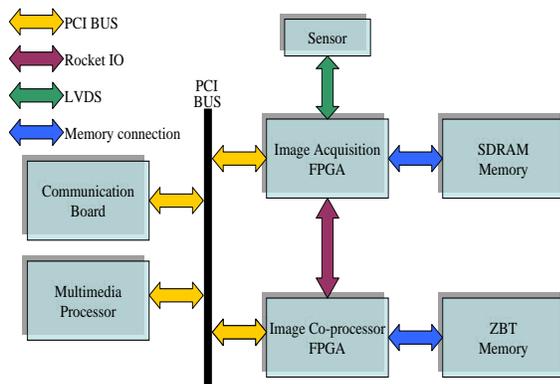


Fig. 3. Simplified smart camera description.

described showing that when building a CAL dataflow model it is not necessary to redefine the implementation of the existing hard-wired communication interfaces and busses.

A. The communication between the different platform components

The communication infrastructure in data dominated systems is an essential part in the development of an embedded system. The design choices aim at obtaining the largest achievable bandwidth between the four components of the platform. In developing the CAL dataflow model, "Core" or

"drivers" already written by the component vendors (Xilinx and Nexperia in this case) have been used. In this way, a reduction of the development time and resources were achieved. The components of the communication infrastructure are:

- Sensor => acquisition FPGA,
- Acquisition FPGA => pre-processing FPGA,
- Acquisition FPGA => processor,
- Pre-processor FPGA => processor.

The communication between the sensor and the acquisition part is specific for each image sensor. Consequently, the interface must be built accordingly. The connection between the acquisition part and the processing part is standard and independent from the sensor. In the described design case, the connection between the co-processor and the processor is implemented with a standard PCI bus. Hence, the co-processor is independent from the processor and could be used as embedded IP with any PCI system. The co-processor architecture can achieve full data rate transfer on the PCI bus. The communication between the two FPGAs is either a PCI communication (same bus and the processor master this communication channel) or a RocketI/O connection (serial high-speed connection specific to Xilinx component).

In FPGAs, all communications are built with the Core Generator tool of Xilinx, which yields optimized designs. In the processor, a driver is included in the dedicated component library. Thus, it is not necessary to redefine and to implement again these parts of each component in the CAL model.

For such reasons, only the core architecture of the co-processor FPGA is addressed in this design case study. For performance reasons, the pixels of the image from the sensor are transferred in words of 32 bits.

B. Co-processor description

The Co-processor is composed of four components as indicated in Figure 4. These components are:

- The interface with the configuration memory
- The co-processor manager
- The external memory controller
- The processing modules

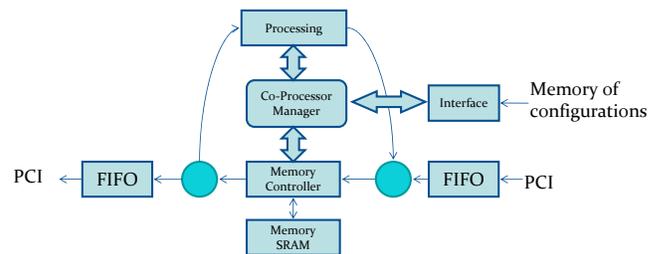


Fig. 4. High level architecture of the co-processor.

The interface connects the configuration memory and the co-processor manager. The configuration defines: the image size, the processing tasks and the start address in the external

memory where the results of the image processing are stored. The co-processor manager deals with the whole system by controlling all the components. The memory manager accesses the external memory to acquire an image according to the processing. The processing module performs the actual processing according to the desired configuration.

The processing modules that have been developed by writing VHDL are:

- Median filter 3x3,
- Transpose,
- Adaptive local binarization,
- High pass filter 11x1,
- Dilation 31x1,
- Sub sample by 4 in width and by 4 in height.

C. Bar code reading application

The postal sorting is a real-world example chosen to show the processing possibilities and the achieved level of parallelism of the system. The goal of this application is to detect and decode bar codes on letters, as shown in Figure 5, to enable automatic sorting at different stages of the logistic postal letter handling. This application is a good example of usage of all the processing possibilities of this heterogeneous platform. Some processing tasks are implemented by the co-processor and the others by the processor.

This section describes how the bar code is decoded. Different



Fig. 5. Application of the reading of a bar code.

processing steps are necessary to complete the process. In sequence, a bar code is detected by applying: a transposition, a high pass filter, a dilation, a sub-sampling, a "blobbing" and finally a decoding on the area where the bar code has been detected. The first processing stage is a transposition. The transposition rotates the image acquired line by line vertically of 90 degrees. As described in [6], a transposition is necessary because the other processing stages are specific to a horizontal reading. The first processing is a high pass filter. The high pass filter deletes the background and raises the white bar code. The resulting image is stored into the memory, but two others processing tasks are built into the FPGA co-processor. The two pre-detection tasks are the dilation and the sub-sampling. The first step dilates the bars of the bar code to build white areas. This step is necessary later to correctly identify the bar code location within the image. The second step reduces the image size without changing its content relevance. Then, the small image obtained is sent to the processor using less bandwidth on the PCI bus. These four tasks are all executed by the co-processor. The "blobbing" executed by the processor consists in locating the two or three largest "white" areas in the small image which provide the location of the bar codes. A command is sent

to the co-processor to send only the regions determined by the blobbing. These regions are taken into the image without background stored previously into the memory. In order to decode the bar code, the processor executes several 1-D FFT on lines oriented in different directions. In reality, only the part delimited by the rectangular region is read as illustrated in Figure 5.

IV. CO-PROCESSOR DATAFLOW MODEL

In this work, we want to compare the implementation of a subset of the smart camera platform in VHDL with its equivalent implementation in CAL. Two versions of the coprocessor implemented in CAL are used for the comparison. The first design is a exact transposition of the VHDL architecture into a CAL dataflow architecture. The comparison of this design with the VHDL version provides the information on how the CAL toolset is efficient in generating HDL code from a given dataflow architecture. In the second design, the coprocessor is a complete redesign of new architecture at CAL abstraction layer providing the same functionality of the VHDL model and the original design. Such new design exploits the properties of the CAL to HDL compiler toolset to reduce the on-chip area of the original design. Developing a CAL design by successive architectural refinements can be performed much more quickly compared to modifying a HDL design.

A. CAL co-processor design of the handwritten HDL architecture

This section describes the architecture of the original co-processor design (illustrated in Figure 6) and how it has been transposed into the CAL dataflow model. As shown in

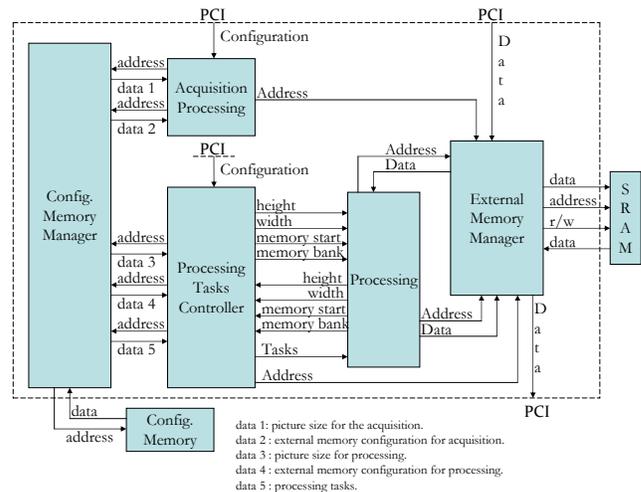


Fig. 6. CAL Dataflow HDL architecture of the co-processor.

the picture, "Acquisition Processing", "Configuration Memory Manager", "Processing Task Controller", "Processing" and "External Memory Manager" are architectural components that have been fully specified in CAL. Processing tasks have been created to support the described

application. Such processing tasks written in CAL are : High Pass Filter, Dilation, and Transpose. Their implementations in CAL respect the same architecture of the original design with just minor modifications to I/O ports. These actors are not explained hereafter.

The dataflow description of the co-processor is reported in Figure 6. The Acquisition Processing actor receives its configuration by the processor via the PCI bus. Then, it produces two addresses which are used to retrieve data in the configuration memory. When it receives *data 1* and *data 2* coming from the configuration memory, it produces addresses to store the image into the external memory. *data 1* is the image size and *data 2* contains the information about the localization of the read/write operation in the external memory (addresses). When the acquired image is completely saved in the SRAM, the Processing Tasks Controller actor receives its configuration data sets. Like the Acquisition Processing actor, it fetches *data 3*, *data 4* and *data 5* into the configuration memory. Then, it sends the list of the selected tasks to the actor processing and their associated configurations. Afterwards, the Processing Tasks Controller actor waits for the updated configurations produced by "Processing" actors. If number of processing tasks are realized, the resulting image is sent via the PCI bus. The Configuration Memory Manager and External Memory Manager actors are only used to switch the data or the addresses towards the right actors. The subsections below explains in details behavior of the different actors illustrated in Figure 6.

1) "Acquisition processing" actor: The actor "Acquisition Processing" is illustrated in Figure 7. Its function is to interpret the configuration sent by the processor via the PCI bus. These configuration data sets are stored temporarily into the Config. Memory. Configuration data are the size of the image and its place into the physical memory. In details, the "Acquisition Task Convertor" actor selects the right configuration data in the Config. memory thanks to the "Data Requester" actors. The "memory start convertor" and "Picture Size Convertor" actors send the following tokens to the "Address Generator" actor : *Height*, *Width*, *Start Address* and *Memory Bank*. This latter actor generates also addresses used to store the image into the external memory.

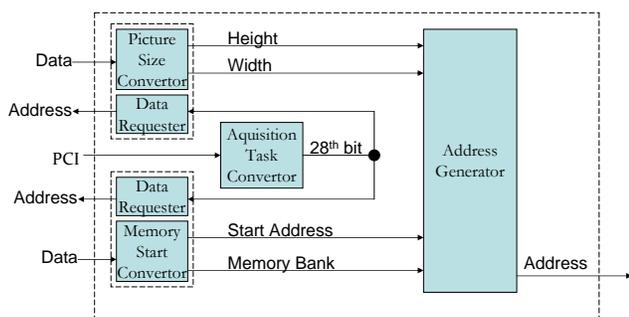


Fig. 7. Dataflow model of the "Acquisition Processing".

2) "Processing tasks controller" actor: The configuration of the processing which is sent by the processor is obtained with the same principle than the previous actor. These configurations are: size of the resulting image, place in the physical memory of the result, list and number of processing tasks. The four "Processing Tasks Manager" actors manage correctly the *Height*, *Width*, *Address Start* and *Memory Bank* parameters in function of the number of processing tasks. These actors communicate with the actor "Processing", described hereafter, via wires "underway" and "new". "underway" is the current value of each parameter and "new" the new parameters sent by "Processing" actors. Once processing tasks are finished, the final step is to send a resulting image to the processor. This last step is processed by the "Processing Manager" and "Address Generator" with the last configuration values.

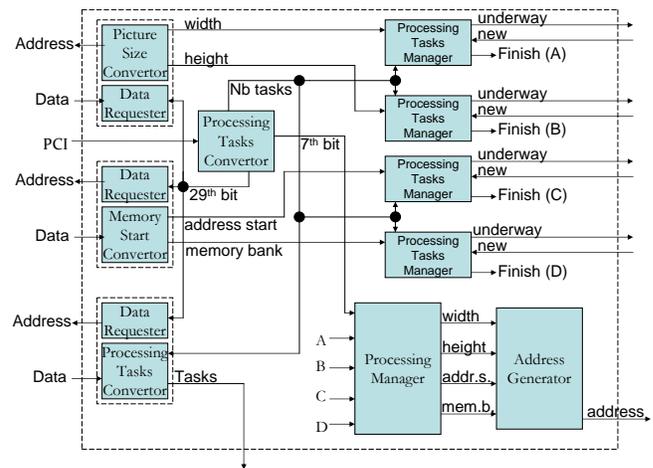


Fig. 8. Dataflow model of the "Processing Tasks Controller".

3) "Processing" network: "Processing" is a network of actors that performs various processing tasks on the images (Figure 9). As explained above, this network receives the current parameters into "Processing Manager" actors. Only one "Processing manager" actor reacts in function of the selected processing tasks. After, the right "Processing" actor (or network) receives the parameters and execute its task. When the processing is finished, parameters are updated and sent to the "Processing Task Controller". Each processing operation in this network (denoted by "Processing N") can be a single actor or a network of actors. In this design there are three processing operations: Transpose, High pass filter, and Dilation. These operation are partitioned into multiple actors which allow to perform actions in parallel.

4) "External Memory Manager" and "Configuration Memory Manager" actors: The "External Memory Manager" actor implements two functionality: read and write. The write function generates the addresses and the data. The read function generates the addresses and the data according to addresses shunting. The actor "Address Convertor" provides the information if the address is in read mode or in write mode. The "Configuration Memory Manager" actor implements a similar

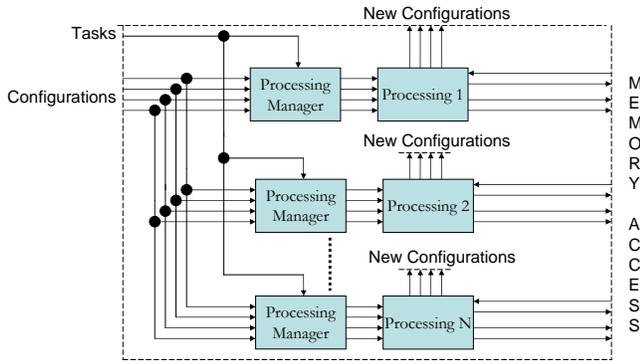


Fig. 9. Dataflow model of the "Processing".

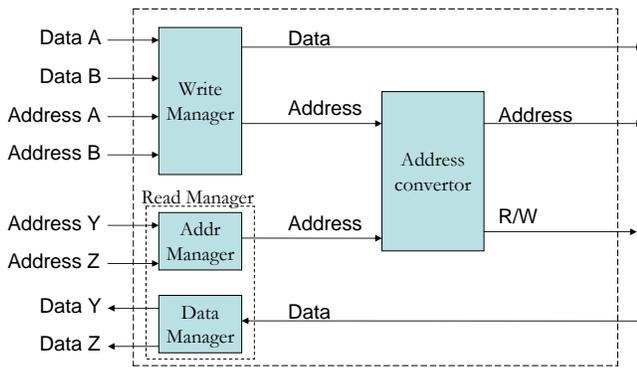


Fig. 10. Dataflow model of the "memory manager".

functionality which is the read part of the "memory manager" actor.

B. Redesign of the coprocessor

Beside the transposition of the architecture written from VHDL in CAL, described in the previous sections a new architecture has been completely redesigned directly in CAL with no more correspondences with the original VHDL reference architecture (Figure 11). Groups of actors that operate sequentially and do not benefit much from parallelism are merged into a larger actor. Such actors have almost the same functionality of all actors of the transposed architecture. In the new design, actors "Acquisition Processing", "Configuration Memory Manager", and "Processing Task Controller" are merged into a single actor, "Controller". This actor deals with commands sent via the PCI bus, reads and interprets configuration data stored in the configuration memory and controls the processing actors. It initiates appropriate processing operations and collects and stores results of processing. The actor "Memory Controller" provides the interface to the SRAM memory and arbitrates access requests made by various sources. A processing network includes three processing operations: High Pass Filter, Dilation, and Transpose. As above, each of these networks are redesigned by merging some of their actors into larger actors and thus the number of actors in each network is reduced compared to the original transposed design described

above. However, each processing operation implements the same functionality and it is equivalent to the original design. One objective of the new data flow redesign, where actors are merged into larger actors is to achieve better tradeoffs between area and throughput. Such possibility is achievable at the level of the CAL design. Since each actor when converted to HDL has an area overhead for reset circuitry, internal finite-state-machine, input and output token queues and circuitry for the communication channels, a non negligible resource overhead is required when too many actors are instantiated. If such actors do not usually work in parallel there is no throughput penalty in merging them and saving silicon area. This usually holds when dealing with many small actors with a small number of actions. On the other hand, due to the fact that actions of an actor cannot be executed in parallel, it must be considered that throughput could decrease as the number of actions in an actor increases. This suggests that for those parts of the system that mostly work in sequence and do present little or no parallelism, it is better to use a fewer number of actors to save resources. For parts with a relevant amount of parallelism is it better to use a higher number of actors to achieve higher throughput. The fact that such trade-offs can be developed at CAL level using a compact high level representation, constitutes a very attractive feature of CAL based design.

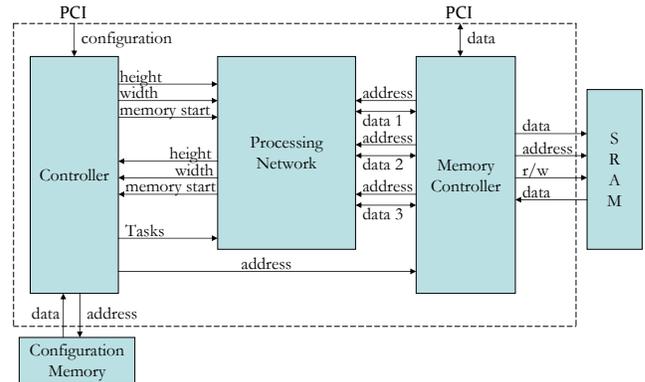


Fig. 11. New CAL Dataflow architecture.

V. RESULTS AND PERFORMANCES COMPARISON

In this section, the results of the synthesis in HDL and RTL of the models of the architecture developed in CAL dataflow language are reported and compared with the hand written model.

A. Results of dataflow language

When the dataflow model has been developed, it is simulated using the Opendataflow simulator [7] to check for the correct functionality. Its equivalent HDL code is generated using the tool described in [2] and then it is synthesized. The results of the synthesis is reported in Tables I and II for the original and revised design. Several variant configurations for the coprocessor have been tested to explore the performance

of the conversion tool and the appropriateness of the modeling methodology. Initially, a version of the co-processor without image processing functions is reported. This includes only functionality required for scheduling and dataflow controls which are highly adapted to CAL framework. As a result, both design have better area performance than the VHDL design while both preserve the same processing throughput.

In the redesigned coprocessor, the hardware overhead is reduced compared to the original design. This is due to the lower number of actors in the CAL model, since each actor instantiation implies overheads in the synthesized hardware (i.e. fifos and handshake protocols for each data token connection).

In terms of frequency and processing time, the maximum achievable frequencies for the original and the revised design are 90 MHz and 100 MHz respectively. The processing time at these frequencies for a test image of 1712×180 is about 0.85 ms and 0.77 ms. Both designs present a processing time of 1.54 ms at 50 MHz which is the working frequency for the example application.

Other scenarios experimented in this work include the usage of the processing network. In one case it includes only the high pass filter and in the other it includes the complete processing tasks (high pass filter, transpose, and dilation). Processing tasks introduce additional delay and as result the throughput is reduced compared to the scenario with no processing. The same considerations can be seen with the whole processing used in the design case application. For the original design the clock frequency is the same for the three scenarios. The revised design has higher clock frequency in the two first scenarios and has a frequency equal to the original design for the scenario with full processing. Nevertheless, all clock frequencies are completely in accord with the specification of the application. Processing delays for the original design are 3.57 ms at 50 MHz and 1.92 ms at 90 MHz and for the revised design are 7.30 ms at 50 MHz and 4.06 ms at 90 MHz. Throughput comparison between the two designs shows that both achieve the same performance in the first comparison scenario, but the original design performs better in scenarios with processing tasks included. The reason is that in the first scenario all tasks are performed sequentially and we do not benefit from having actors working in parallel. For processing operations, the potential parallelism is high so the original design which presents a large number of actors working in parallel yields a higher throughput at the cost of larger area.

B. Performances comparison

Table I, Table II and Table III reports the results of the two methodologies in terms of number of occupied slices, slice number of Flip Flops, number of four input LUTs, frequency and throughput. In these tables, the size of files that describe the same elements in VHDL and CAL is also reported.

Table I reports the results of the co-processors implemented in CAL. Table II reports the results of hand written VHDL as described in [6]. It has to be noticed that the PCI core interface is not taken into account by the results reported in the tables. The results show that all along the development of

the dataflow CAL model the hardware resources used in the FPGAs are also lower or nearly the same in the revised design than the one necessary by the handwritten VHDL design. A second interesting point is that the code size of CAL is by far smaller than the code size of VHDL with a factor ranging from 3 up to 10. However, even if such factors are already an excellent result, it can be noticed that supplying CAL with a library of basic functions similar to what in VHDL are Concatenation(), Bitselect() and similar low level library functions, such compactness factor can largely further improve. Moreover, CAL code is better structured and results are much easier to understand and analyze than an equivalent VHDL or Verilog design.

In terms of maximum achievable frequency, the hand written architecture remains better and consequently the achievable data throughput at the maximum frequency is higher. However, considering that the application requires processing at 50 MHz, when the two architecture work at the same frequency the throughput for the given application example is the same for both the original design and the revised design. However, in terms of HW resources, the revised design achieves better results compared to the original design and the handwritten design. Another important point is the reduction of the development time in CAL by a factor of about four compared to the hand written coding. Thus, with these results it is easy to say that the development time in dataflow is much faster than the one of a standard HDL development language.

Figure 12 summarizes the main components of a design: platform resource usage, design productivity and performance. For this application example size area is reduced in one of the case (redesigned CAL), the development time is considerably reduced by at least a factor four and the processing/data throughput is approximately the same for both methodologies. Moreover, the code size written for the same application is reduced by a factor of 3.

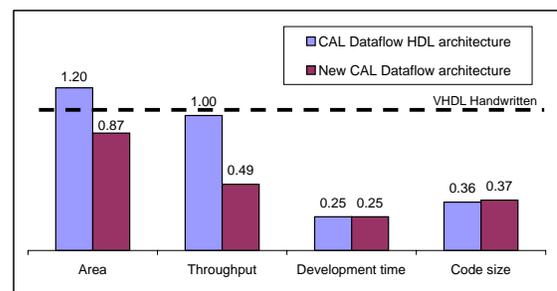


Fig. 12. Comparison to VHDL reference design.

VI. CONCLUSION & FUTURE WORKS

In this paper two design methodologies, a classical approach with handwritten HDL and a CAL dataflow development, are compared. The performance of dataflow approach and the performance of the conversion from CAL to HDL are evaluated. The results reported in section V are very promising. Beside a slight reduction of the maximum achievable frequency, other

	Number of occupied slices	Number of slice Flip Flops	Number of 4 input LUTs	Code size (kbyte)	Frequency MAX (MHz)	Processing Time(ms)/Throughput (image 1712x180/s)	
						At max Frequency	At 50 MHz
						Without processing module	949
High pass filter only	2193	1999	1846	39.0	≈90	0.85/1176	1.54/649
All processing modules	3127	2930	3956	65,7	≈90	1.92/520	3.57/280

Table I: CAL Dataflow HDL architecture results

	Number of occupied slices	Number of slice Flip Flops	Number of 4 input LUTs	Code size (kbyte)	Frequency MAX (MHz)	Processing Time(ms)/Throughput (image 1712x180/s)	
						At max Frequency	At 50 MHz
						Without processing module	469
High pass filter only	1,220	1,170	1,621	34.2	≈100	2.31/433	4.61/217
All processing modules	2,276	1,991	3,079	68.3	≈90	4.06/246	7.3/137

Table II: Dataflow revised design results

	Number of occupied slices	Number of slice Flip Flops	Number of 4 input LUTs	Code size (kbyte)	Frequency MAX (MHz)	Processing Time(ms)/Throughput (image 1712x180/s)	
						At max Frequency	At 50 MHz
						Without processing module	1,171
High pass filter only	1,639	2,779	2,376	162.0	≈125	0.61/1,639	1.54/649
All processing modules	2,623	3,855	3,740	182.9	≈125	1.37/730	3.57/280

Table III: Hand written VHDL results

design results are improved, particularly in terms of HW resources for the revised design and throughput for the original design. Moreover, development time and code size in both CAL model examples are reduced of a relevant factor, enabling interesting redesign iteration options.

Several improvements could be further applied to the methodology tested so far. The first is certainly to include a library of basic low level function to ease CAL code writing and considerably reduce the size of the code. The second is to continue the improvement and the optimization of the HDL generation tool as well as implementing extensions of the OpenDF framework functionality.

REFERENCES

[1] Johan Eker and Jorn Janneck, "CAL Language Report", Tech.Rep.ERL Technical Memo UCB/ERL M03/48, University of California at Berkeley, Dec. 2003.

[2] Jörn W. Janneck, Ian D. Miller, Dave B. Parlour, Marco Mattavelli, Christophe Lucarz, Matthieu Wipliez, Mickal Raulet, and Ghislain Roquier, Translating dataflow programs to efficient hardware: an MPEG-4 simple profile decoder case study, in Design Automation and Test in Europe (DATE), Munich, Germany, 2008.

[3] Christophe Lucarz, Marco Mattavelli, Matthieu Wipliez, Ghislain Roquier, Mickaël Raulet, Jörn W. Janneck, Ian D. Miller and David B. Parlour, "Dataflow/Actor-Oriented language for the design of complex signal processing systems", Workshop on Design and Architectures for Signal and Image Processing (DASIP08), Bruxelles, Belgium, November 2008.

[4] J. Dubois, M. Mattavelli, "Embedded co-processor architecture for CMOS based image acquisition", IEEE International conference of Image Processing (ICIP03), Volume 2, pp.591–594, 2003

[5] R. Mosqueron, J. Dubois M. Mattavelli "High Performance Embedded co-processor Architecture for Cmos Imaging Systems", Workshop on Design and Architectures for Signal and Image Processing (DASIP07), Grenoble, France, November 2007

[6] R. Mosqueron, J. Dubois and M. Mattavelli, "Smart camera with embedded co-processor : a postal sorting application", Optical and Digital Image Conference (SPIE08), Proceeding Volume 7000, Strasbourg, France, April 2008.

[7] "open DataFlow Sourceforge Project" <http://opendf.sourceforge.net/>.

Insights to Variable Block Size Motion Estimation by Design Space Exploration

Jani Boutellier
Machine Vision Group
University of Oulu
90014 Oulun yliopisto, Finland
Email: bow@ee.oulu.fi

Philip Brisk and Paolo Ienne
Processor Architecture Laboratory
École Polytechnique Fédérale de Lausanne
1015 Lausanne, Switzerland
Email: philip.brisk@epfl.ch, paolo.iemme@epfl.ch

Abstract—Numerous different implementations for H.264/AVC variable block size motion estimation have been proposed in the recent years to make this computationally challenging task more feasible for mobile devices with video encoding support. The variable block size motion estimation problem defined by the standard is complex and multidimensional, offering a wide variety of possibilities for efficient implementation. One of the most popular implementation architectures are systolic arrays.

In this paper we look at the full-search variable block size motion estimation problem on 1D systolic arrays from a high level by modeling the system with a software tool that enables design space exploration and cycle-accurate simulation. Our design space exploration tool has provided many interesting insights to the VBSME problem that give directions for making efficient designs.

I. INTRODUCTION

Motion estimation is a fundamental part of modern block-based hybrid video coders, such as in MPEG-4 video [1]. In block-based motion estimation a small part (template) of the present video frame is matched against a search area in a neighboring video frame in the hope that corresponding content can be found.

In the new h.264/AVC video coding standard the motion estimation is done by so-called variable block size motion estimation (VBSME), which is one of the most computationally demanding tasks of the encoding effort. In the last few years, numerous implementations have been proposed to make the computational burden lighter [2], [3]. These approaches can be divided into two main categories: approximative and full-search solutions.

An approximative algorithm for VBSME uses only a fraction of the available search space to find the best match for the template image data. This means that the globally best match cannot always be found, but there will be a sub-optimal solution with a considerably smaller computational effort. As an example, the algorithm described in [2] offers a 75% reduction in computational complexity with an image quality loss of only about 0.5dB (PSNR), which is visually hard to detect.

However, there are some occasions when the sub-optimal solution is not acceptable. In this case the motion estimation has to be done by a full-search method that searches all of the area within the limits or the search radius. Although the num-

ber of computations increases dramatically when compared to approximative solutions, there are fortunately a few features in the full-search approach that can make it computationally feasible. First of all, there is a considerable amount of redundancy in the computations. When utilized correctly, this means that the amount of data moved around can be reduced greatly. Second, the full search space makes the problem very regular, which promotes the use of simple and fast processing elements (PEs). Finally, the full-search solutions can be parallelized in many different ways, which opens further possibilities to reduce the latency of the algorithm.

Because of the aforementioned properties, the H.264/AVC VBSME problem has often been implemented in systolic array hardware. A systolic array architecture consists of a set of processing elements organized by a highly regular interconnection network (e.g., a mesh). During every clock cycle, each PE performs some computation and exchanges data with its immediate neighbors. The PE is typically controlled by a finite state machine (FSM) and the data flow is synchronous and regular. Both 1-dimensional (1D) systolic arrays and 2-dimensional (2D) solutions are used for VBSME. 1D systolic array architectures are typically used for portable electronics, where a premium is placed on power consumption and battery lifetime, rather than performance; 2D architectures are typically used for high-end systems, where a premium is placed on performance and power consumption is not a primary concern.

In this paper we describe a software tool for modeling 1D systolic array architectures that can be used for VBSME. The software tool has been written in C++ with the help of a general-purpose systolic array library. The software model can be used for design space exploration of VBSME solutions and it has given some insight on building efficient 1D systolic array architectures for VBSME.

In Section II we describe the general-purpose systolic array library that was used as a basis for our design space exploration tool. Section III describes the design space exploration tool in detail and Section IV describes the insights provided by our work.

II. SYSTOLIC ARRAY DEVELOPMENT LIBRARY

Our design space exploration tool was built on a systolic array library that is written in C++ in an object-oriented

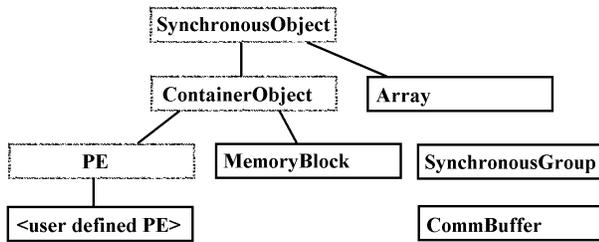


Fig. 1. The systolic array primitive classes.

manner. As a basis in the library are several primitive classes that are either instantiated directly by the designer, or used as base classes for custom design. The class hierarchy is depicted in Figure 1.

The library provides the primitive building blocks for a systolic array architecture, as well as the means to connect them. In addition to that, also convenient functions have been designed that help setting up meshes of PEs by a single function call. A feature worth mentioning is also the possibility of address auto-increment in memory blocks, which enables removing this mundane task from the actual processing loop.

Our library does not automatically map applications to a systolic array, but assumes that the designer has enough knowledge to do that. This also means that the synchronization of data streams from different inputs must be figured out by the designer.

The parent of all systolic array classes is *SynchronousObject* that provides a common clock to all objects that have a concrete counterpart in the system. *ContainerObject* is a virtual class that is derived from *SynchronousObject*. It is the parent class of all classes that can be used to store data. The final virtual class is *PE*, which contains the mandatory functionalities common to all kinds of processing elements. Our library does not offer any directly usable processing element classes, instead it is assumed that the user of the library derives a new class from *PE* and uses that for the application. After deriving a custom PE class, the user must program its actual behaviour in C++, which essentially means defining functionality between the input and output channels of the PE. The functionality between the input and outputs can be arbitrary, as long as it provides valid data to the outputs at each clock cycle and respects the width of the channel. Below is the class declaration of *PE*:

```

class PE : public ContainerObject
{
public:
    PE();
    ~PE();
    virtual void Process(int cycle) = 0;
    void ReadChannels(int cycle);
    void WriteChannels(int cycle);
    int AddInput(CommBuffer* source);
    int AddOutput(CommBuffer* sink);
};

```

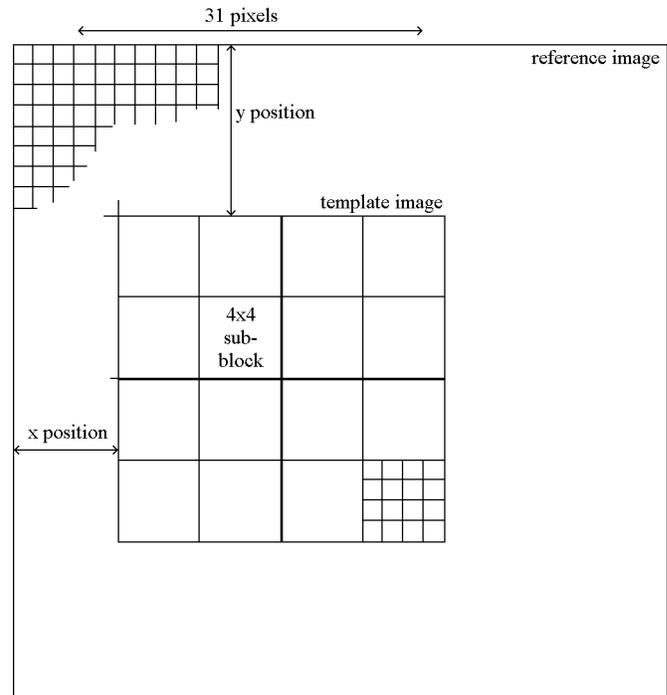


Fig. 2. The variable block size motion estimation problem.

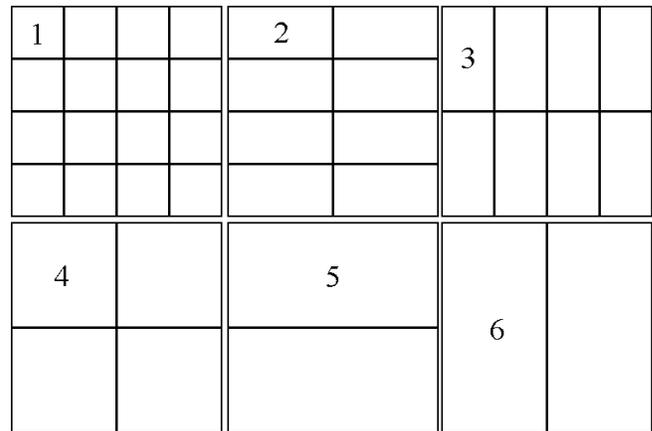


Fig. 3. The different block shapes. Shape 7 is not shown: it involves all sub-blocks.

MemoryBlock is a class that is derived from *ContainerObject* and it is used to represent memories that can be written or read by processing elements.

Commbuffer is a stand-alone class without parents that is used to represent a communication channel between classes that are derived from *ContainerObject*. *Commbuffer* is capable of storing the incoming and outgoing data, as well as checking that the data does not exceed the channel width or that one channel is not written twice within one clock cycle. Instances of the *Commbuffer* class are not aware of the inputs and outputs that are connected to it. That information is found in the *ContainerObjects* attached to it.

Array is a class that can contain processing elements and communication channels, and serves as a wrapper that repre-

TABLE I
USAGE EXAMPLE OF OUR LIBRARY.

Action	Explanation
-	Derive a custom processing element class myPE from <i>PE</i>
mysg = new SynchronousGroup();	Instantiate new <i>SynchronousGroup</i>
myarr = mysg.CreateArray();	Create a new systolic array object
myarr.CreatePEs(<i>myPE</i>)(1,8);	Instantiate 1 times 8 processing elements of type myPE to <i>myarr</i>
myarr.CreateChannels(0,8);	Connect PEs by 8-bit channels in direction 0 (left-to-right)
inmem = mysg.CreateMemoryBlock(size);	Instantiate new <i>MemoryBlock</i>
inmem.AddOutput(arr.GetPE(0,0).GetInput(0));	Connect dangling leftmost channel of the array to <i>inmem</i>
inmem.SetOutputStepping(0, 1);	Set memory address auto-increment to 1
outmem = mysg.CreateMemoryBlock(size);	Instantiate new <i>MemoryBlock</i>
outmem.AddInput(arr.GetPE(0,7).GetOutput(0));	Connect dangling rightmost channel of the array to <i>outmem</i>
outmem.SetInputStepping(0, 1);	Set memory address auto-increment to 1
for(i = 0; i < size; i++) mysg.Process();	Do cycle-by-cycle processing through the input data

sents the actual systolic array. It exists mainly to provide a convenient way to create and modify arrays of PEs. Finally, *SynchronousGroup* is the highest-level wrapper class that is intended to contain all the objects attached to the systolic array. *SynchronousGroup* essentially offers the shared clock functionality that is needed by all classes derived from *SynchronousObject*.

Table I shows a toy example of using the library. It does not contain all the details, but outlines the sequence of class instantiations and method invocations to set up and do some processing with a simple 1-D systolic array that has 8 PEs, and input and output memories.

A. Run-Time Behaviour of the Library

The cycle accurate simulation of the array is based on a three-phase functionality that is common to all objects inherited from *SynchronousObject*. Each phase is initiated by calling the respective function for the one and only *SynchronousGroup*, which recursively calls the same function in each object that it contains.

In the first phase of each clock cycle, ReadChannels() is called. For every object *O* the data values from the *CommBuffers* connected to the inputs are copied to the input data registers of *O*. During the Process() phase, object *O* is expected to fill its output registers – usually by modified contents of the input registers. For *MemoryBlock* objects, the Process() method is defined to be empty, since memories are not supposed to modify their contents. Finally, the simulated clock cycle ends by the WriteChannels() method, during which every object *O* copies the contents of output registers to the respective *CommBuffers*. This three-phase procedure makes the functionality cycle accurate, which means that the content of each memory, channel and PE can be exactly determined at each clock cycle.

The library offers convenient ways to debug systolic arrays. For example, the behaviour of custom PEs can be inspected easily by writing *cout* or *printf* -style calls to the Process()-method. Even hard-to-detect timing problems can be found quickly by outputting the PE contents each clock cycle

III. THE DESIGN-SPACE EXPLORATION TOOL

Our design-space exploration tool was initially built as a high-level model that expresses two different, previously built 1D systolic array architectures as different design points of one general model. The two architectures that span the design space were: *a part of* the work done by Yap and McCanny [3] and another 1D design that is still unpublished work [4]. Before explaining our Design Space Exploration (DSE) tool in detail, we define the VBSME problem and present the two designs [3], [4] that motivated our work.

A. VBSME Concepts

In the h.264/AVC VBSME algorithm the 16x16 pixel *template image* is compared to the 31x31 pixel *reference image* at all 256 *positions* that the 16 pixel search range allows (See Figure 2). The comparison is done by computing a sum of absolute differences (SAD) between the overlapping pixels of the template and reference image at each specific position.

The 16x16 pixel template image is divided into 4x4 *sub-blocks*, which are combined into variable size *blocks* in seven different ways (depicted in Figure 3). As the SAD results are computed for each of the 7 block combinations that contain various numbers of blocks, the result of this scheme is 41 SAD (16+8+8+4+2+2+1) results for *each* search position, giving a total of 10496 SAD results. The VBSME search algorithm must finally return the position that gave the best SAD result, for each of the 41 partial SADs.

In addition to these standard concepts, we define an additional concept called *cluster*. A cluster is a rectangular shape of 16 pixels and it is used to describe the order in which the template image pixels are processed. Clusters come in 5 different shapes, as depicted in Figure 7. Possible cluster shapes include 1x16, 2x8, 4x4, 8x2 and 16x1. As each template image consists of 16 clusters, also the processing order of *clusters* must be agreed upon. For example, the traditional raster scan [3] can be implemented with the cluster shape 16x1.

For every array architecture presented in this paper, the input data feed was organized in the same way. The organization is named *broadcasting reference frame data* [5] and is depicted in Figure 4.

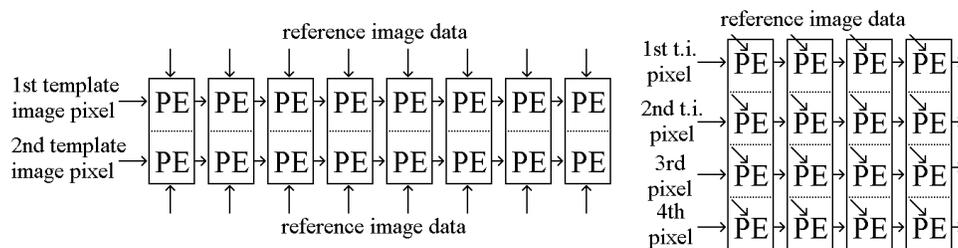


Fig. 4. The data input to the processing elements for configurations 8x2 and 4x4.

B. Implementation of the Architecture of Yap and McCanny

The architecture designed by Yap and McCanny [3] is depicted in Figure 5. It has a complex interconnection named the SAD Bus Network that connects two sets of processing elements. Most of the computations are done in the first set of PEs (labeled PE in Figure 5), whereas the second set of processing elements practically only serves as a set of comparators (labeled Min in Figure 5). As stated before, we used a part of the solution of Yap and McCanny as one initial design point while creating our DSE tool. The part that we used, was the array of processing elements labeled PE in Figure 5.

The PE data input and output schedules are not defined explicitly for the DSE tool. Instead, the DSE tool automatically computes a SAD production schedule after the array parameters have been set. For the array configuration that matches the work of Yap and McCanny, the resulting DSE PE schedules match exactly the ones defined in the paper [3].

The systolic array design of [3] was not followed outside the array named "PE" (see Figure 5), e.g., the SAD bus network is not reproduced by the DSE tool. Instead, the comparator functionality was implemented in the same fashion as in the other baseline work [4]. This is explained in the next subsection.

C. Improved 1D Systolic Array

The second design point that gave directions in creating our DSE tool was an internally developed architecture that dramatically simplifies the 1D systolic array design when compared to the work described in [3]. This design takes the comparator functionality that resides in a separate set of processing elements in [3], and moves them *inside* the primary set of PEs (shown in Figure 5). This eliminates the need of the complex SAD bus network.

This solution differs from the design of [3] also by an alternative organization of processing elements, which is depicted in Figure 6 under the name "4 PE pipeline". Moreover, the input order of the template image data (See Fig. 4) has been changed. We shall see in Section IV, how these changes affect the design. This architecture was designed on register transfer level before the work described in this paper was done.

D. Design Space Exploration Tool

As the two previous subsections already suggest, this design space exploration tool is actually a parameterized implemen-

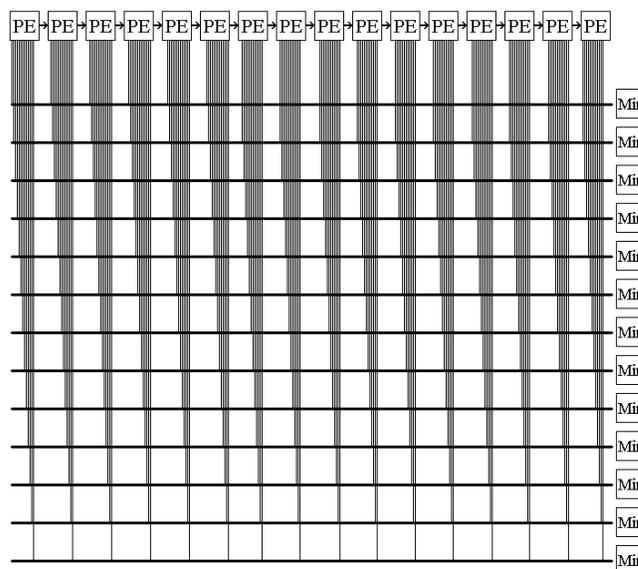


Fig. 5. The Yap and McCanny SAD Bus Network.

tation of our internally developed VBSME systolic array [4] that can also be transformed into the design described in Subsection III-B, and numerous other designs by simple parameter changes.

The DSE tool allows changing the *array configuration* (Fig. 6) and *template image cluster shape* (Fig. 7), but the throughput remains fixed¹. These two parameters affect the number of registers in the system, as will be shown later. In addition to these two parameters, the *cluster processing order* can also be varied freely in the exploration tool, but we used only one fixed cluster order for each cluster shape.

Changing the design parameters does not change the number of computations that are done inside a PE: each PE computes a single pixel difference on each clock cycle. However, there is a difference in the rate that SAD results for sub-blocks and larger blocks are produced.

Based on the variables mentioned above, the DSE tool computes a cycle-accurate schedule for all the SAD computations of the block matching and allocates register space for results that have to be stored temporarily. At this point it is worthwhile to mention that our tool correctly reproduces the schedules of

¹To be specific: the results are available in 4096 + L clock cycles, where L is the pipeline length: 1...16

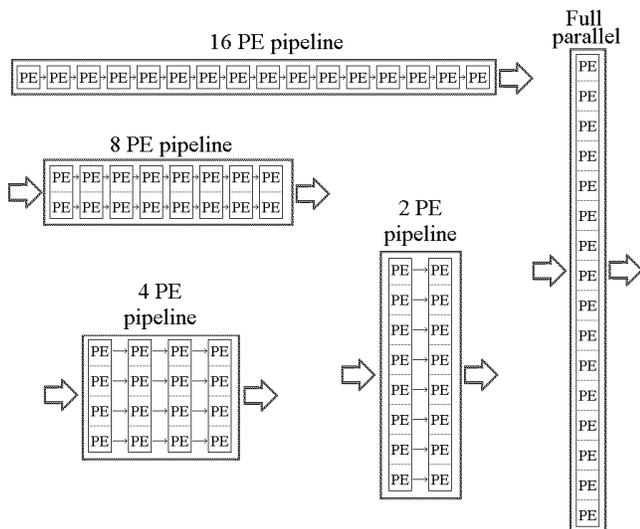


Fig. 6. The array configurations supported by the design space exploration tool.

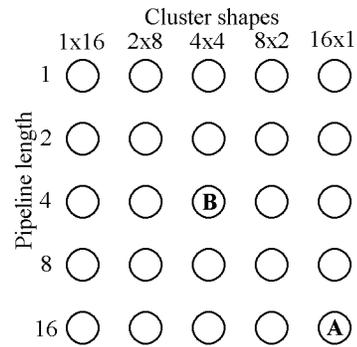


Fig. 8. The design space.

The key result that the DSE tool produces is the schedule for SAD computations that tells at which clock cycles new results emerge and have to be processed by the comparator hardware. Evaluation of the SAD schedules brings up the first insight provided by this work: it is desirable that the SAD results are produced as a smooth stream instead of periodic bursts. If the SAD results occur as sudden bursts, the PEs need to contain more register space to save the results before sending them for further processing. According to the DSE output, the *cluster shape* is the variable that affects this issue most, as can be seen in Table II. The cluster shape 4x4, which has also been used in [4], provides the smallest SAD production variance and thus, register space requirement. This makes sense intuitively, since 4x4 is also the shape of sub-blocks in H.264/AVC. Again, this speaks against the use of the traditional raster scan that has been used, e.g., in [3] and [5].

The length of the systolic pipeline (See Fig. 6) is related to the reuse of template image data. With the pipeline of length 1 (array shape 1x16), the template image pixels are never reused inside the array, but instead are read 16 times more often from the template image memory than with pipeline length 16 (array shape 16x1). This means that the memory bandwidth increases as the pipeline grows shorter: with pipeline length 1 the system requires 16 new template image pixels (= 128 bits) each clock cycle, whereas the configuration with pipeline length 16 requires only one template image pixel value (8 bits) per clock cycle.

When we assume that we would create a design that uses the cluster shape 4x4, we see from Table II that the number of registers per PE remains generally around 6 with all pipeline lengths. The reason behind this is that during the VBSME computations, it happens for every PE at some point that 6 SAD results are ready at the same clock cycle. Therefore, each of the PEs must have this minimum number of temporary register space. This leads to our second insight: since the register space requirement per PE remains roughly the same for all pipeline lengths, this speaks strongly against long pipelines. This issue is shown in Table III that shows the total number of registers used by each configuration. In addition to the temporary register space, we have also added the necessary inter-PE communication registers to the register count here.

IV. INSIGHTS PROVIDED BY THE DSE TOOL

Figure 8 shows the design space covered by our DSE tool. The vertical axis depicts the array configurations that are shown in Figure 6 and the horizontal axis depicts the cluster shapes (See Fig. 7). The letter A in the design space figure shows the design space position of the solution that was described in Subsection III-B. Respectively, the letter B shows where the solution of Subsection III-C lies in the search space. The other 23 design space points represent new solutions that have been brought up by this DSE tool. As we will later see, the DSE results point out that some of these designs might even be better than [4], [3], whereas others are most certainly worse.

Fig. 7. Template image pixel scanning order for cluster shapes 2x8, 4x4, 8x2 and 16x1. The scan order of 1x16 is not shown; it is that of 16x1, transposed.

[4] and [3], and thus validates the correctness of our model.

It is assumed that there is an 8-bit pipeline register for each parallel channel between PEs, and that temporary SAD results that are stored internally are 16 bits.

Related to the SAD register space, one might wonder if it would be possible to share the SAD registers between different PEs. With clever design and a small number of PEs, this could truly work, but it is evident that for longer pipelines this will raise complex interconnection issues such as the SAD Bus network in [3].

Another fact worth mentioning is that the cluster shape and processing order variables also affect the template image memory access patterns. The most simple memory access pattern is achieved with the cluster shape 16x1, which implies only a simple memory address incrementation each clock cycle. Other cluster shapes require more complex addressing, as Figure 7 shows. However, this particular issue can be compensated by the way how the template image is written to the memory – it is possible to write the pixels to the memory so that simple address incrementation can be applied to every cluster shape.

In a nutshell, the results indicate that longer pipeline lengths result in large numbers of registers (as well as other PE components) that are under-utilized. This speaks in favor of short 1..4 PE pipelines. Naturally, since our design space exploration tool does not provide a HW synthesis backend, it is not certain that a system like "Full Parallel" in Figure 6 would be feasible. Also, it is clear that the cluster shape (i.e. template image pixel processing order), should be set to 4x4, since it requires the least register space.

The DSE tool shows that there are potentially better array configurations than the ones that were known prior to building the DSE tool. It was known beforehand from our internally developed 1D array that the cluster shape 4x4 was better than the commonly used 16x1 raster scan. The design space exploration automated the non-trivial problem of determining schedules for the SAD calculations, which is strongly related to the register count required by the architecture.

As the systolic array library does not yet provide a possibility for hardware synthesis, it is not certain what kind of undiscovered drawbacks these newly found array configurations might have. This is a clear direction for future work.

V. RELATED WORK

Systolic arrays were popularized in the research community by Kung and Leiserson [6] in 1978. Historically, systolic arrays have been used for many different applications. Classic examples of algorithms that could easily parallelize on systolic arrays include matrix multiplication and evaluation of polynomial algorithms using *Horner's Rule*.

In the domain of video coding, systolic arrays have also been used elsewhere than in motion estimation: Chipper [7] has published a systolic array implementation of the inverse discrete cosine transform and Li *et al.* [8] have published a systolic array implementation of 2-dimensional interpolation.

A considerable amount of work has been done in the development of the ALPHA language [9], which enables the

TABLE II
NUMBER OF REGISTERS REQUIRED BY DIFFERENT CONFIGURATIONS, PER PE.

PL, Cluster shape	1x16	2x8	4x4	8x2	16x1
1	13	7	6	6	13
2	12	5	5	7	12
4	12	6	6	7	12
8	9	6	6	6	9
16	6	6	6	6	6

TABLE III
REQUIRED REGISTER SPACE IN BYTES FOR EACH SOLUTION IN THE DESIGN SPACE.

PL, Cluster shape	1x16	2x8	4x4	8x2	16x1
1	26	14	12	12	26
2	56	28	28	36	56
4	108	60	60	68	108
8	158	110	110	110	158
16	207	207	207	207	207

automatic synthesis of systolic algorithms. ALPHA follows an equational approach, where a program is expressed as a collection of single assignment equations. This is a major difference when compared to our approach, as well as the fact that ALPHA does not convey implicitly the notion of time.

A work close to ours [10] has been done as a part of the Aries project. This technical memo describes a C++ framework name *Sim* that is intended for cycle-based simulations. Compared to our work, Sim employs a lower level of abstraction.

Saraswat *et al.* have performed design space exploration [11] on the same VBSME problem as presented in this paper. They used a tool called DesertFD and their work considered a much wider problem that involved choosing one of two different sub-optimal search patterns and also synthesizing two solutions to actual hardware. Their paper also contains a good survey of recent DSE papers related to VBSME.

VI. DISCUSSION AND FUTURE WORK

We have presented a variable block size motion estimation design space exploration tool for 1D systolic arrays. It is based on a general-purpose systolic array library that has been written in C++. The DSE tool correctly reproduces two already existing designs and uncovers a variety of new designs, some of which indicate the possibility of improved performance when compared to the existing ones.

For future work it would be interesting to build a backend to the systolic array design library that would enable automatic hardware synthesis of the systems that have been designed with the library. Also, the design space exploration tool could be extended to encompass various throughputs and search radiuses.

REFERENCES

- [1] F. C. Pereira and T. Ebrahimi, *The MPEG-4 Book*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2002.

- [2] M.-J. Chen, G.-L. Li, Y.-Y. Chiang, and C.-T. Hsu, "Fast multiframe motion estimation algorithms by motion vector composition for the mpeg-4/avc/h.264 standard," *IEEE Transactions on Multimedia*, vol. 8, no. 3, pp. 478–487, June 2006.
- [3] S. Y. Yap and J. McCanny, "A vlsi architecture for variable block size video motion estimation," *IEEE Transactions on Circuits and Systems*, vol. 51, no. 7, pp. 384–389, July 2004.
- [4] H. Parandeh-Afshar, P. Brisk, and P. Jenne, "Private communication."
- [5] K.-M. Yang, M.-T. Sun, and L. Wu, "A family of vlsi designs for the motion compensation block-matching algorithm," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 10, pp. 1317–1325, Oct 1989.
- [6] H. Kung and C. Leiserson, "Systolic arrays (for vlsi)," *Technical Report CS 79-103, Carnegie Mellon University*, no. 33, 1978.
- [7] D.-F. Chipper, "A new systolic array algorithm for memory-based vlsi array implementation of idct with high throughput rate and low complexity," *Digital Signal Processing Workshop Proceedings, 1996., IEEE*, pp. 179–182, 1-4 Sep 1996.
- [8] C.-H. Li, C.-C. Chen, W.-C. Su, M.-J. Wang, W.-H. Peng, T. Chiang, and G.-G. Lee, "A unified systolic architecture for combined inter and intra predictions in h.264/avc decoder," in *IWCMC '06: Proceedings of the 2006 international conference on Wireless communications and mobile computing*. New York, NY, USA: ACM, 2006, pp. 73–78.
- [9] H. L. Verge, C. Mauras, and P. Quinton, "The alpha language and its use for the design of systolic arrays," *J. VLSI Signal Process. Syst.*, vol. 3, no. 3, pp. 173–182, 1991.
- [10] J. P. Grossman, "An efficient c++ framework for cycle-based simulation," in *Project Aries Technical Memo ARIES-TM-17*. Massachusetts Institute of Technology, 2002.
- [11] R. Saraswat and B. Eames, "On the use of desertfd to generate custom architectures for h.264 motion estimation," *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, 2008*, pp. 359–368, 31 2008-April 4 2008.

A Hardware Oriented Integer Pel Fast Motion Estimation Algorithm in H.264/AVC

Obianuju Ndili^{#1}, Tokunbo Ogunfunmi^{#2}

[#] *Department of Electrical Engineering, Santa Clara University
Santa Clara, California 95053 USA*

¹ ondili@scu.edu, ² togunfunmi@scu.edu

Abstract—Two approaches to motion estimation speed-up in H.264/AVC include designing fast motion estimation algorithms and accelerating motion estimation in hardware. Hybrid fast motion estimation algorithms emerged from an effort to maintain motion estimation speed-up without compromising rate-distortion performance. The best hybrid algorithms combine the use of various search patterns with motion vector prediction, in order to solve the “local minimum” problem. One such algorithm is the Simplified Unified Multi-Hexagon (SUMH) search also known as the Simplified Fast Motion Estimation (SFME) algorithm, a non-normative part of the H.264/AVC standard. The drawback however, of hybrid algorithms, is that they are intrinsically control-heavy and sequential and thus do not readily lend themselves to parallel processing on hardware. In this paper, we present hardware-oriented modifications to SUMH. In [10], we have proposed a flexible hardware architecture for the modified SUMH. Our experiments compare the PSNR performance and computation time of Full Search, SUMH and the modified SUMH algorithms. Our results show that the modified SUMH, while it achieves the desired goal of greater suitability for hardware implementation, its losses in terms of PSNR performance and computation time, are insignificant.

I. INTRODUCTION

Motion estimation (ME), is by far the most powerful compression tool in the H.264/AVC standard [1], [2], and it is generally carried out in two stages: integer pel and fractional pel. It features variable block sizes, quarter-pixel accuracy for the luma component (one-eighth pixel accuracy for the chroma component), and multiple reference pictures. However the power of ME in H.264/AVC comes at the price of increased encoding time. Experimental results [3], [4], have shown that ME can consume up to 80% of the total encoding time of H.264/AVC, with integer ME consuming a greater proportion. In order to meet real-time and low power constraints, it is desirable to speed up the ME process. Two approaches to ME speed-up include designing fast ME algorithms and accelerating ME in hardware.

Considering the algorithm approach, Yi et al. [5], proposed a fast ME algorithm known variously as the Simplified Unified Multi-Hexagon (SUMH) search or Simplified Fast Motion Estimation (SFME) algorithm. SUMH is based on UMHExagonS [4], a hybrid fast motion estimation algorithm. Yi et al. show in [5] that with similar or even better rate-distortion performance, SUMH reduces ME time by about 55% and 94% on average when compared with UMHExagonS and Fast Full Search respectively. In addition, SUMH yields a

bit rate reduction of up to 18% when compared with Full Search in low complexity mode. Both SUMH and UMHExagonS are non-normative parts of the H.264/AVC standard.

Considering ME speed-up via hardware acceleration, although there has been some previous work on VLSI architectures for variable block size motion estimation (VBSME), in H.264/AVC, the overwhelming majority of these works have been based on the Full Search Motion Estimation (FSME) algorithm. This is because FSME, presents a regular-patterned search window which in turn demands a less complex architecture – the memory organization, routing and control are all straightforward. The price to pay, however, for the simplicity of the FSME architecture is the greater amount of time (and hence power), consumed in ME.

Previous works on architectures for fast motion estimation (FME) [6] – [9], have been based on diverse FME algorithms.

Rahman et al. in [6] and Byeon et al. in [7] base their works on UMHExagonS. In [8], Chen et. al propose a parallel, content-adaptive, variable block size, four step search (4SS) algorithm, upon which their architecture is based. In [9], Zhang et al. base their architecture on the following search sequence: Diamond Search (DS), Cross Search (CS) and finally, fractional-pel ME.

In this paper, we present hardware-oriented modifications to SUMH. We also show the advantages and costs of these modifications, taken separately and as a whole. Our results (see Section IV), show that for the modified SUMH, the PSNR loss is 0.01 dB to 0.06 dB when compared with FSME, and 0.003 dB to 0.03 dB when compared with SUMH. In terms of percentage computational time savings, while SUMH saves 88.8% to 98.8% when compared with FSME, the modified SUMH saves 61.5% to 92.2% when compared with FSME. Finally, the percentage bit rate increase of the modified SUMH is 0.21% to 4.32% when compared with FSME, and 0.18% to 2.95% when compared with SUMH.

Thus the contribution of this work is to show that while the proposed modifications to SUMH make the algorithm for suitable for hardware implementation, the losses incurred in terms of PSNR performance, percentage computational time savings and average bit rate increase, are insignificant.

In this paper, we also present a brief overview of a flexible hardware architecture which we propose more fully in [10].

This hardware architecture is based on the modified SUMH algorithm that is described in this paper.

The rest of this paper is organized as follows: In Section II we summarize integer-pel motion estimation in SUMH. In Section III we present the hardware-oriented modifications we make to SUMH. In Section IV we present results of these modifications. Finally our conclusions are presented in Section V.

II. INTEGER PEL SUMH ALGORITHM

H.264/AVC uses block matching for motion vector search. Integer-pel motion estimation uses the sum of absolute differences (SAD), as its matching criterion. The mathematical expression for SAD is given in equation (1).

$$SAD(dx, dy) = \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} |a(x, y) - b(x + dx, y + dy)|$$

(1)

$$(MV_x, MV_y) = (dx, dy) \Big|_{\min SAD(dx, dy)} \quad (2)$$

In equation (1), $a(x,y)$ and $b(x,y)$ are the pixels of the current, and candidate blocks, respectively. (dx, dy) is the displacement of the candidate block within the search window. $X \times Y$ is the size of the current block. In equation (2) (MV_x, MV_y) is the motion vector of the best matching candidate block.

H.264/AVC features seven inter-prediction block sizes which are: 16x16, 16x8, 8x16, 8x8, 8x4, 4x8 and 4x4. These are referred to as block modes 1 to 7, as shown in Fig. 1. An up layer block [4], is a block that contains sub-blocks. For example, mode 5 or 6 is the up layer of mode 7, and mode 4 is the up layer of mode 5 or 6.

SUMH [5] utilizes five key steps for intensive search, integer-pel motion estimation. They are: cross search, hexagon search, multi big hexagon search, extended hexagon search and extended diamond search. For motion vector (MV) prediction, SUMH uses the spatial median and up layer predictors, while for SAD prediction, the up layer predictor is used. In median MV prediction, the median value of the adjacent blocks on the left, top, and top-right (or top-left) of the current block is used to predict the MV of the current block. Specific rules [11] govern special cases such as when the current block is on the edge of a picture or group of blocks. The complete flow chart of the integer-pel, motion vector search in SUMH is shown in Fig. 2.

III. HARDWARE ORIENTED SUMH ALGORITHM

The goal of our hardware-oriented modifications is twofold. First we seek to make SUMH less control-heavy and sequential. This way the algorithm is readily adaptable for parallel processing on hardware. Secondly, we seek to keep at a minimum, any PSNR losses and increases in the computation time. The following are the modifications we make to SUMH:

1. We consider only the zero displacement motion vector

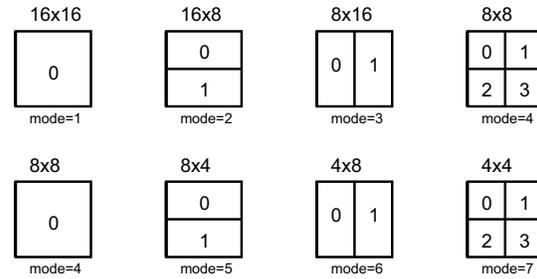


Fig. 1. The seven inter-prediction block sizes in H.264/AVC.

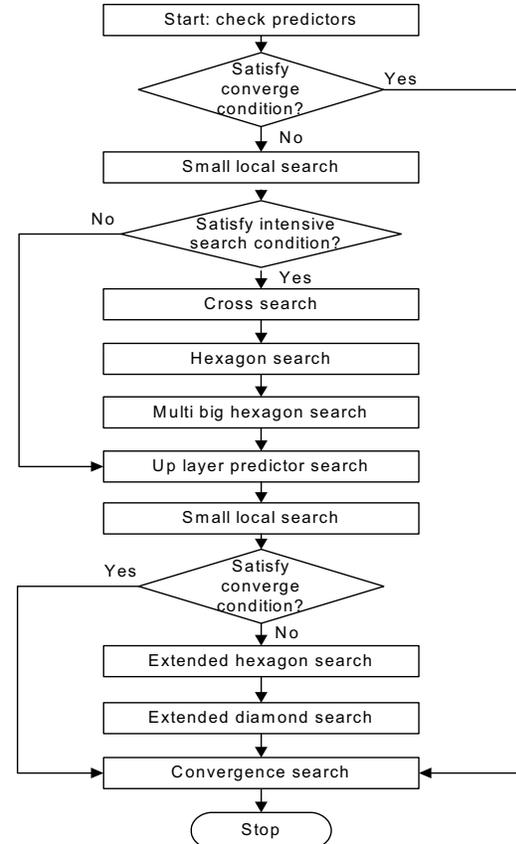


Fig. 2. Flow chart of integer pel search in SUMH.

i.e. $(MV = (0, 0))$, as the predictor at the start of the algorithm. By centering our initial search window around $MV = (0, 0)$, we break data dependency across neighboring CMBs. This enables parallel processing of multiple CMBs. Experiments in [12], [13] and [14], show that when the search window is centered around $MV = (0, 0)$, the average PSNR loss is less than 0.2 dB compared with when the median MV is also used. In Section IV we present results from our own experiments, which show the effect of using only $MV = (0, 0)$ as the motion vector predictor.

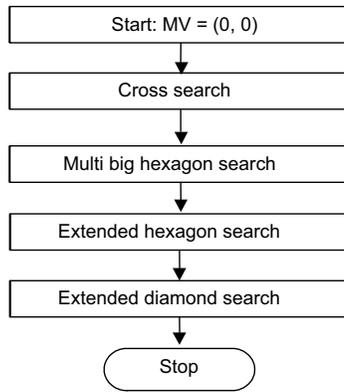


Fig. 3. Flow chart of modified integer pel search.

2. We consider that intensive search condition is satisfied. We do this in order to remove the decision control structures that make SUMH control-heavy and sequential. Another advantage of considering the worst case search path is that we greatly compensate for the PSNR losses that would otherwise have been incurred through our other modifications.

3. We skip the hexagon search to maintain regularity of the search patterns while avoiding greatly degrading the speed-up value and PSNR performance of SUMH. Section IV presents results from our experiments, which show the effect of skipping the hexagon search.

The complete flow chart of the modified integer-pel, motion vector search is shown in Fig. 3

In Fig. 4 we show our proposed scalable and configurable hardware architecture based on the modified SUMH. The details of the architecture can be found in [10]. However, we see from Fig. 4 that the architecture consists of N processing units (PUs), 2 Memory Banks that operate in ping-pong fashion, a Control Unit, a SAD Combination Tree, a Comparison Unit that compares all possible 41 SADs computed in parallel and a register that stores the final result. Fig. 5 shows a PU which consists of N processing elements (PE). A PU computes 16 4×4 SADs for one candidate macroblock (MB). Fig. 6 shows the SAD Combination Tree. It combines the outputs of a PU, then outputs the 41 possible SADs of one MB. Fig. 7 shows a PE which calculates and accumulates the absolute difference between two pixels. Fig. 8 shows a comparison element (CE), which consists of 41 N -input comparators.

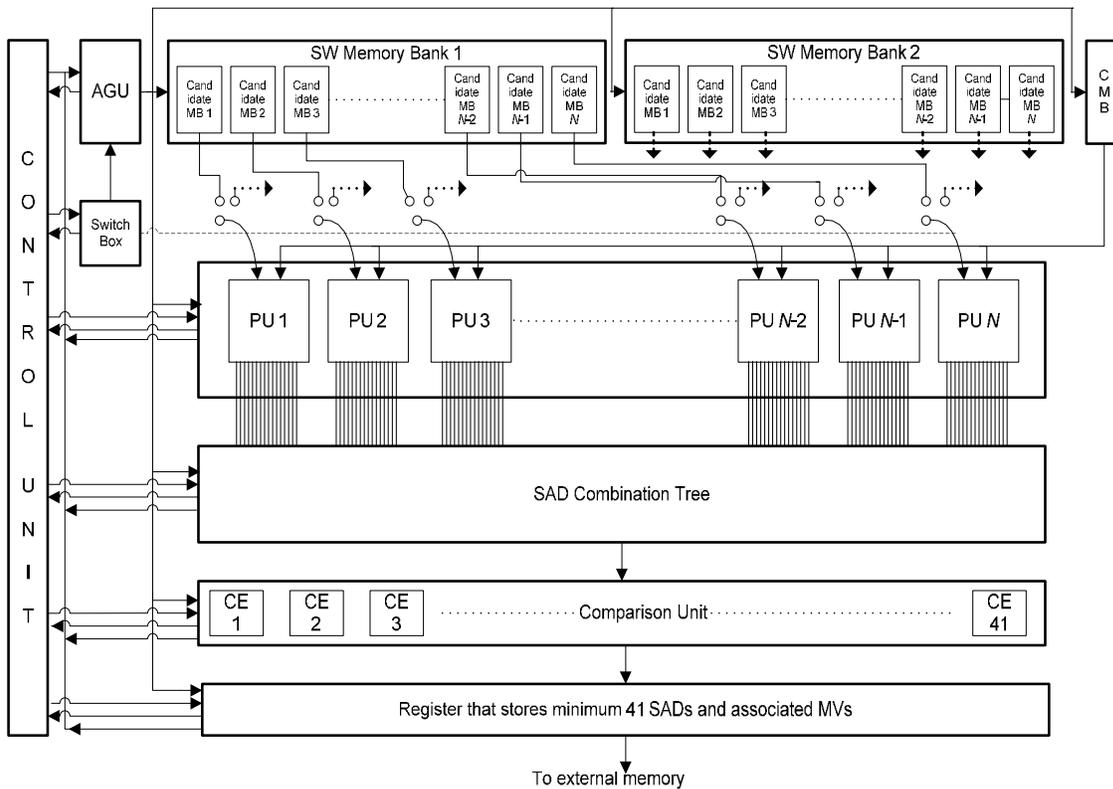


Fig. 4. The Proposed Architecture for Implementing the Modified SUMH.

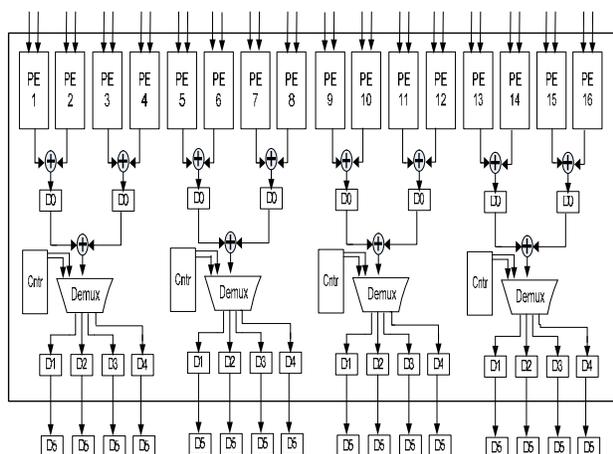


Fig. 5. The architecture of a Processing Unit (PU).

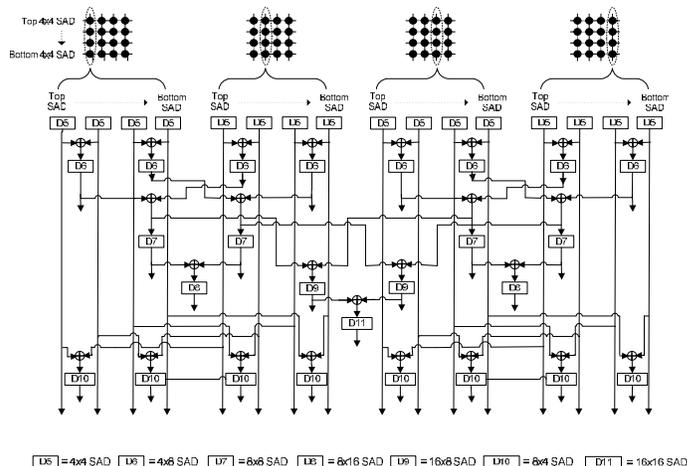


Fig. 6. SAD Combination Tree.

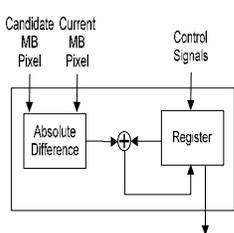


Fig. 7. Processing Element (PE).

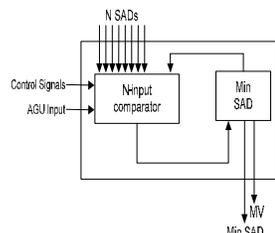


Fig. 8. Comparing Element (CE).

IV. EXPERIMENTAL RESULTS

Our experiments are done in JM 13.2 [15]. We use the following standard test sequences: “stefan” (large motion), “foreman” and “coastguard” (moderate to large motion), “silent” (small motion). We chose these sequences because we consider them extreme cases in the spectrum of low bit-rate video applications. The sequences are coded at 30 Hz. The picture sequence is IPPP with I-frame refresh rate set at every 15 frames. We consider 1 reference frame. The rest of our simulation conditions are summarized in Table I.

In Table II we present the average Y-PSNR losses incurred separately in setting the MV predictor to $MV = (0, 0)$, and in skipping the hexagon search. Fig. 9 shows curves that compare the rate-distortion efficiencies of Full Search ME, SUMH and the modified SUMH. In Tables III and IV, we

show a comparison of the speed-up ratios of SUMH and the modified SUMH. Table V shows the average percentage bit rate increase of the modified SUMH when compared with Full Search ME and SUMH. Table VI shows the average Y-PSNR loss of the modified SUMH when compared with Full Search ME and SUMH. Finally Table VII shows a comparison of the computational complexities for Full Search ME, SUMH and the modified SUMH.

From Table II, we see that there is a slightly less PSNR loss from skipping the hexagon search, than from setting the MV predictor to $MV = (0, 0)$. Note that although the individual modifications taken separately, each yield losses as large as 2 dB, the average PSNR loss for the modified SUMH is much smaller. This can be seen from Fig. 9 and Table VI. The reason for this is that the modified SUMH algorithm takes the most intensive search path. As a result of the intensive search, the individual losses introduced by the modifications in Table II, are greatly compensated for.

From Fig. 9 and Table VI we also observe that the largest PSNR losses occur in “foreman” sequence, while the least PSNR losses occur in “silent”. This is because the “foreman” sequence has both high local object motion and greater high-frequency content. It therefore performs the worst under a given bit rate constraint. On the other hand, “silent” is a low motion sequence. It therefore performs much better under the same bit rate constraint.

TABLE I
SIMULATION CONDITIONS

Sequences	QUANTIZATION PARAMETER	SEARCH RANGE	Frame Size	No. of Frames
Foreman	22, 25, 28, 31, 33, 35	32	CIF	100
Stefan	22, 25, 28, 31, 33, 35	16	CIF	90
Coastguard	18, 22, 25, 28, 31, 33	32	QCIF	220
Silent	18, 22, 25, 28, 31, 33	16	QCIF	220

TABLE II
AVERAGE Y-PSNR LOSS PER MODIFICATION

Modification	SET MV PREDICTOR AS $MV = (0,0)$	SKIP HEXAGON SEARCH
Foreman	0.0297 dB	-0.0038 dB
Stefan	1.0088 dB	0.9918 dB
Coastguard	2.0898 dB	2.0798 dB
Silent	0.8170 dB	0.8170 dB

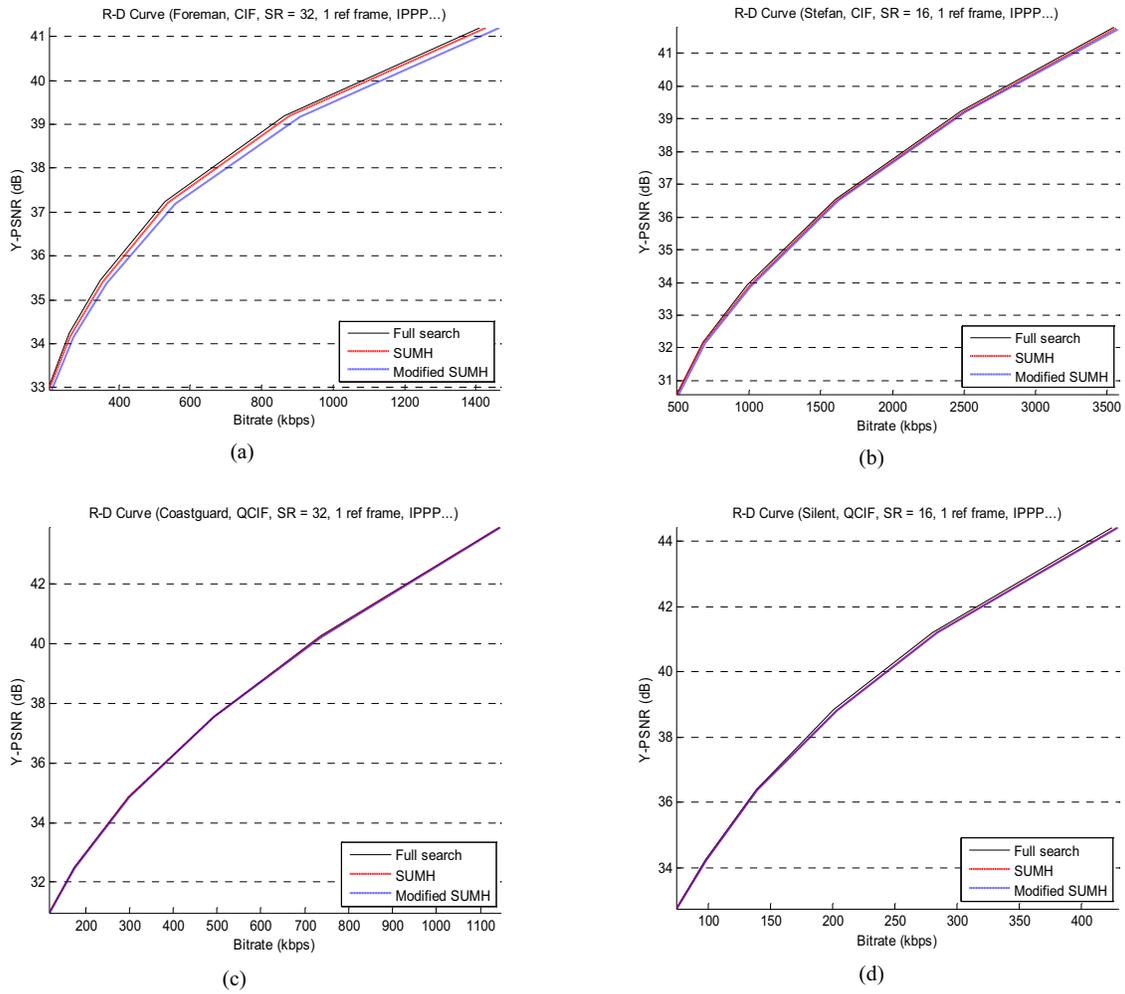


Fig. 9. Comparison of rate-distortion efficiencies.

TABLE III
COMPARISON OF SPEED-UP RATIOS

Quantization Parameter	18		22		25		28		31		33		35	
	SUMH	Modified SUMH												
Foreman			48.55	8.51	41.55	7.11	32.68	5.87	25.87	4.95	21.68	4.39	19.11	3.94
Stefan			15.35	4.64	13.16	4.32	12.20	4.04	10.67	3.76	10.05	3.76	8.96	3.18
Coastguard	86.34	12.75	70.12	10.99	58.05	9.42	43.62	8.23	36.04	6.87	30.10	6.10		
Silent	21.86	4.28	16.74	3.54	13.17	3.17	11.90	2.98	9.29	2.71	8.56	2.59		

TABLE IV
COMPARISON OF PERCENTAGE TIME SAVINGS

Quantization Parameter	18		22		25		28		31		33		35	
	SUMH	Modified SUMH												
Foreman			97.94	88.26	97.59	85.93	96.94	82.98	96.13	79.82	95.38	77.24	94.76	74.63
Stefan			93.48	78.47	92.40	76.88	91.80	75.27	90.63	73.42	90.05	70.23	88.83	68.58
Coastguard	98.84	92.16	98.57	90.90	98.27	89.39	97.70	87.85	97.22	85.45	96.67	83.62		
Silent	95.42	76.65	94.02	71.75	92.40	68.54	91.60	66.46	89.23	63.21	88.32	61.46		

TABLE V
AVERAGE PERCENTAGE BIT RATE INCREASE FOR MODIFIED SUMH

Compared With	Full Search	SUMH
Sequences		
Foreman	4.32	2.95
Stefan	1.99	1.24
Coastguard	0.21	0.20
Silent	1.08	0.18

TABLE VI
AVERAGE Y-PSNR LOSS FOR MODIFIED SUMH

Compared With	Full Search	SUMH
Sequences		
Foreman	0.0645 dB	0.0290 dB
Stefan	0.0265 dB	0.0082 dB
Coastguard	0.0132 dB	0.0072 dB
Silent	0.0117 dB	0.0037 dB

For Table III, we define the speed-up ratio as the ratio of the ME coding time of Full Search to ME coding time of the algorithm under consideration. From Table III we see that speed-up ratio increases as quantization parameter (QP) decreases. This is because there are less skip mode macroblocks as QP decreases. From our results in Table III, we further calculate the percentage time savings t for ME calculation, according to equation (3).

$$t = \left(1 - \frac{1}{r}\right) \times 100 \quad (3)$$

where r are the data points in Table III. The percentage time savings obtained are displayed in Table IV. From Table IV, we find that SUMH saves 88.8% to 98.8% in ME computation time compared to Full Search, while the modified SUMH saves 61.5% to 92.2%. Therefore, the modified SUMH does not incur much loss in terms of percentage time savings.

In our experiments we set rate distortion optimization to high complexity mode (i.e. rate distortion optimization is turned on), in order to ensure that all of the compared algorithms have a fair chance to yield their highest rate-distortion performance. From Table V we find that the average percentage bit rate increase of the modified SUMH is very low: 0.21% to 4.32% when compared with Full Search, and 0.18% to 2.95% when compared to SUMH. Here also, we find that the modified SUMH incurs very little degradation in terms of bit rate increase.

From Table VI we see that the average PSNR loss for the modified SUMH is very low. When compared to Full Search, the PSNR loss for modified SUMH ranges from 0.01 dB to 0.06 dB. When compared to SUMH, the PSNR loss for modified SUMH ranges from 0.003 dB to 0.03 dB. Once again, these are insignificant losses.

From Table VII we see that the complexity of the modified SUMH in terms of the number of subtractions, absolute value operations and additions, is in between the complexities of the best case SUMH search path and the worst case SUMH search path. However the complexity of the modified SUMH is much lower than that of Full Search ME.

As mentioned earlier, the reason why the modified SUMH algorithm achieves overall insignificant PSNR losses whereas the individual modifications in Table II yielded larger losses, is because the removal of the decision control structures causes us to do the intensive search every time. As a result of the intensive search, more accurate motion vectors are obtained.

V. CONCLUSION

In this paper we have presented hardware-oriented modifications to SUMH. From our experimental results, we found that the modified SUMH algorithm achieves very low PSNR loss. When compared with Full Search, the average PSNR loss ranges from 0.01 dB to 0.06 dB. When compared with SUMH, the average PSNR loss is 0.003 dB to 0.03 dB. In addition to achieving a low PSNR loss, the modified

TABLE VII
COMPARISON OF COMPLEXITY (SEARCH WINDOW SIZE = $-N$ to $N-1$)

	Number of Subtractions	Number of Absolutions	Number of Additions
1 Macroblock	256	256	256
Full Search	$256(2N \times 2N)$	$256(2N \times 2N)$	$256(2N \times 2N)$
Best case SUMH	256×4	256×4	256×4
Worst case SUMH	$256(4 + 2N + 6 + 4N + 16 + 4 + 6 + 4)$	$256(4 + 2N + 6 + 4N + 16 + 4 + 6 + 4)$	$256(4 + 2N + 6 + 4N + 16 + 4 + 6 + 4)$
Modified SUMH	$256(2N + 4N + 4 + 6)$	$256(2N + 4N + 4 + 6)$	$256(2N + 4N + 4 + 6)$

SUMH algorithm also retains a very good ME speed-up of 61.5% to 92.2% when compared with Full Search. The bit rate increase for the modified SUMH algorithm is also very low. When compared with Full Search, the modified SUMH algorithm has a bit rate increase of 0.21% to 4.32%. When compared with SUMH, the modified SUMH algorithm has a bit rate increase of 0.18% to 2.95%.

Therefore the modified SUMH can be used instead of SUMH (without much penalty), for ME in H.264/AVC.

REFERENCES

- [1] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560-576, July 2003.
- [2] G.J. Sullivan, P. Topiwala and A. Luthra, "The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions," *SPIE Conference on Applications of Digital Image Processing XXVII*, August 2004.
- [3] H.-C. Lin, Y.-J. Wang, K.-T. Cheng, S.-Y. Yeh, W.-N. Chen, C.-Y. Tsai, T.-S. Chang and H.-M. Hang, "Algorithms and DSP Implementation of H.264/AVC," *Proceedings of 2006 Asia and South Pacific Conference on Design Automation*, pp. 742 - 749, January 2006.
- [4] Z. Chen, P. Zhou, Y. He, "Fast Integer Pel and Fractional Pel Motion Estimation for JVT", JVT-F017, *Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG. 6th Meeting: Awaji Island, Japan*, Dec. 2002.
- [5] X. Yi, J. Zhang, N. Ling, and W. Shang, "Improved and simplified fast motion estimation for JM," JVT-P021.doc, *Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG. 16th Meeting: Posnan, Poland*, July. 24-29, 2005.
- [6] C. Rahman and W. Badawy, "UMHexagonS Algorithm Based Motion Estimation Architecture for H.264/AVC" *In Proceedings, Fifth International Workshop on System-on-Chip for Real-Time Applications*, 2005.
- [7] M.-S. Byeon, Y.-M. Shin, "Hardware Architecture for Fast Motion Estimation in H.264/AVC Video Coding," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E89-A, No. 6, June 2006.
- [8] T.-C. Chen, Y.-H. Chen, S.-F. Tsai, S.-Y. Chien and L.-G. Chen, "Fast Algorithm and Architecture Design of Low-Power Integer Motion Estimation for H.264/AVC," *IEEE Transactions on Circuits and Systems*, Vol. 17, Issue 5, pp. 568 - 577, May 2007.
- [9] L. Zhang and W. Gao, "Reusable Architecture and Complexity-Controllable Algorithm for the Integer/Fractional Motion Estimation of H.264," *IEEE Transactions on Consumer Electronics*, Vol. 53, Issue 2, pp. 749 - 756, May 2007.
- [10] O. Ndili and T. Ogunfunmi, "A Configurable Hardware Architecture for Integer Fast Motion Estimation in H.264/AVC," Submitted to 2009 *International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2009*.
- [11] "Editor's Proposed Draft Text Modifications for Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC), Draft 2", JVT-E022d2, Geneva, Switzerland, 9-17 October, 2002.

- [12] Y. W. Huang, T.-C. Wang, B.-Y. Hsieh, and L.-G. Chen, "Hardware Architecture Design for Variable Block Size Motion Estimation in MPEG-4 AVC/JVT/ITU-T H.264," *Proceedings of the 2003 International Symposium on CAS, ISCAS 2003*, pp. II-796 - II-799, May 2003.
- [13] Y. Song, Z. Liu, S. Goto and T. Ikenaga, "Motion Estimation Algorithm Modificaiton and Implementation in H.264/AVC," *Proceedings of the 2005 IEICE Workshop on CAS*, pp. 193 – 198, April 2005.
- [14] L. Deng, W. Gao, M. Z. Hu and Z. Z. Ji, "An Efficient Hardware Implementation for Motion Estimation of AVC Standard," *IEEE Transactions on Consumer Electronics*, Vol. 51, No. 4, pp. 1360 – 1366, November 2005.
- [15] (2008) H.264/AVC Reference Software JM 13.2. [Online]. Available: <http://iphome.hhi.de/suehring/tml/download/>