PROTOTYPING EMBEDDED DSP SYSTEMS – FROM SPECIFICATION TO IMPLEMENTATION

Zoran Salcic

Department of Electrical and Computer Engineering, University of Auckland, 20 Symonds St., Private Bag 92019, Auckland, New Zealand
email: z.salcic@auckland.ac.nz

ABSTRACT

When embedding DSP systems, a designer must meet not only functional but also many other requirements, such as low resource requirements, real-time constraints and low power consumption. The design flow used in prototyping such systems usually includes high-level languages for specification capture and then involves a number of traditional steps until the system is synthesized. Those steps often require manual interventions, as they lack coherence and integration. New technologies, such as reconfigurable systems based on FPGA technology and novel application specific tools offer a path towards rapid prototyping of such applications in Systems on Programmable Chip. In this paper we explore the features of those technologies and discuss their weaknesses, and present an approach to embedded systems implementation based on a new multiple-processing element computing architecture.

1. INTRODUCTION

Digital signal processing (DSP) algorithms are used in ever increasing number of embedded real-time applications that include systems as simple as consumer products to very sophisticated control systems. Prototyping of those systems include a number of steps, which usually start with high-level description and simulation using common engineering tools such as MatLab and Simulink for at least a part of the application. Then, they proceed with lower level abstraction modelling and use of simulation and synthesis tools, with the final aim of implementing a target system in an electronic form such as System-on-(Programmable)-Chip or So(P)C. The common characteristics of these applications are increasing algorithm complexity, high data and sampling rates, and combination of data-stream processing and reactivity on random external events. Such systems can be classified as heterogeneous embedded system. Also, they rarely allow software-only implementation on a selected processor (general-purpose or DSP processor) platform. More commonly, they require very sophisticated partitioning of the application in its "hardware" and "software" part to achieve system specification requirements. Those requirements include functional and timing characteristics, and power consumption constraints, which are often incorporated into a cost function that may include some other design goals (resource usage requirements, cost and time to market).

One of the major issues in embedded DSP systems prototyping and implementation is number of abstraction levels used to express the same functionality. Each of these description levels introduces modification of the algorithms that may influence the final outcome. For example, we often start with the algorithm functional validation and refinement using high-level tools or programming languages, typically with floating-point arithmetic that does not incur any negative effects on the final results. However, the next steps that aim at the algorithm implementation require further algorithm modifications in order to meet capabilities of the employed architecture (typical example is the use of fixed-point arithmetic as the only realistic option). This immediately requires repeated verification before the final step of algorithm synthesis for the target platform. Target platforms include technologies such as ASICs, FPGAs, general-purpose processors and DSP processors. The final step involves either transformation into synthesisable hardware description (behavioural or register transfer level, RTL) or machine code of the target processor. The later transformation steps usually include some further design goals, such as resource/area minimisation and power reduction.

A number of attempts have been made to automate prototyping of embedded DSP systems by raising the abstraction level of the initial algorithm specification and automating the translation process [4, 5, 8, 9]. Most of them target hardware designers, some algorithm designers, but the major missing step is the tools that would target system designers. The attempts to close the gaps between system, algorithm and hardware designers are exemplified by a number of tools based on programming languages, hardware description languages and system-level languages. The generated architectures synthesised with these tools often do not meet industrial requirements. Other attempts are made by incorporating initial system description into standard Simulink [12] framework. The synthesised architecture is often very close to the input description, which is nothing else but another structural representation similar to traditional datapaths in RTL descriptions. It is limited by a library of available design and processing blocks that are included into the Simulink library. The designer may need to add own designed block in some cases. Also, it addresses primarily processing of data which are sampled from the external environment on a regular basis, but does not deal with random events that change flow of control.

Figure 1 illustrates complexity of embedded DSP systems design and implementation and issues involved in the design flow which influence final system implementation.

In section 2 we discuss performance measures and benchmarking for usually used for DSP systems. Section 3 describes a number of complex applications and illustrates issues and possible solution approaches in their prototyping. of These issues are analysed and discussed further in the subsequent sections. Also, we briefly describe our proposed approach to embedded DSP systems in which we start from a new architecture level that propagates back to synthesis and specification levels by directly supporting features of systems-level design on low, hardware level.

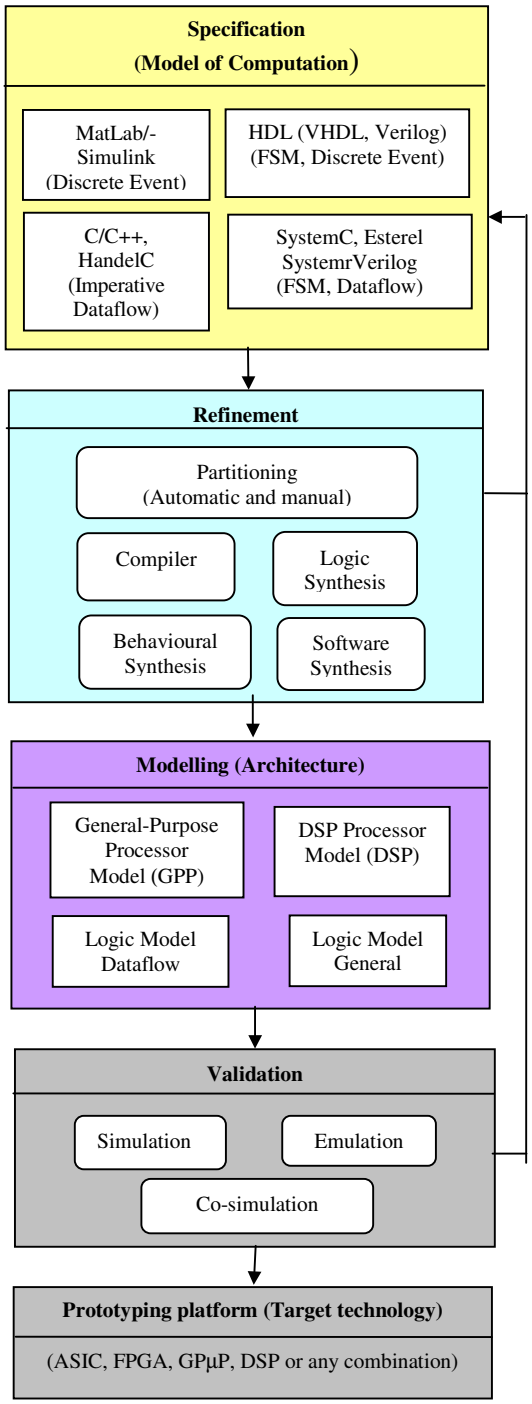

Figure 1 Design methodology and design flow

2. PERFORMANCE MEASURES AND BENCHMARKING

When implementing embedded DSP applications we usually face the problem of mapping an algorithm from its high-level language representation into one that executes in real-time. Performance characteristics and indicators largely vary depending not only on the nature and number of operations that have to be performed, but also on the implementation platform which will be used.

Traditionally, some simple parameters have been used to describe requirements of DSP applications and, then, to measure performance of implementation platform. The most common units, such as MIPS (millions of instructions per second), MOPS (millions of operations per second) and MACS (multiply-accumulates per second), are sometimes misleading as they successfully describe only a part of DSP implementation platform performance. For example, MAC operations, quite common in many DSP algorithms, have variations regarding the arithmetic (fixed-point or floating-point or complex numbers) and do not take into account how operands are acquired. Furthermore, some implementation platforms offer parallel execution of the same operation, but MACS measure disregards that. In addition, all above measurement indicators ignore other facts such as memory usage, power consumption, and application execution time, which are all important when choosing a DSP implementation platform.

Another common benchmark is using the system to complete a certain application. This measurement can also take the memory usage and energy consumption into account. However, the result of application benchmark only indicates performance of a specific application; for another application, the result may be quite different. The performance is also related to experience of different system designers. This benchmark is used to measure system but not the particular platform performance.

Instead of using above measures, some believe the better approach is to define a set of realistic, standard benchmarks and compare execution speeds of implementations using both simple "algorithm kernel" functions (such as real and complex block FIR filter, real single-sample FIR filter, IIR filter, vector dot product, vector add, Viterbi decoder, convolutional encoder, 256-point in-place FFT etc) and entire applications or portions of applications (such as speech coders) to evaluate the performance of implementation platforms. Using algorithm kernels is often suitable because most of DSP algorithms are well defined and functionalities are precisely specified. Another justification is that such DSP algorithms are most computationally intensive portions of DSP applications.

3. APPLICATION EXAMPLES AND SPECIFICS

The more advanced applications tend to use even more complex algorithms, which are often defined using high-dimension matrices and operations on matrices in the data-driven part of the application. Those operations can be nicely described in high-level languages such as MatLab

[12], but their implementation on embedded platform differs considerably. Those applications include decision making based on current computation and on random events that occur in the surrounding environment. Thus, they introduce irregularities into otherwise very regular structures that describe datapaths or algorithms that perform data processing. The requirements of such applications are illustrated on a number of examples that follow.

- **FPGA-based target tracking computer** [14]. A number of adaptive filters (Kalman, α-β, α-β-γ) implemented using direct scalar-based algorithm mapping on FPGA hardware and combination of customised processor and hardware accelerators to meet requirements of the application are used in an adaptive algorithm for target tracking as illustrated in Figure 2. Target position is predicted using different filters depending on estimation of the target on normal trajectory or in manoeuvre. Design methodology used traditional manual hardware/software partitioning and tools for development of hardware and software part of the system. The filters are implemented as hardware functional units and adaptivity is achieved by using software decisions. The achieved performance exceeds any software-only solution, but has a drawback that hardware part of the solution has to be redesigned and synthesised from the beginning if any modification is required.

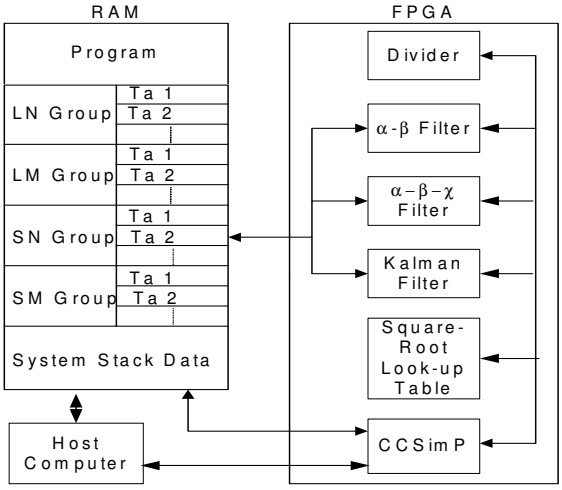

Figure 2 Adaptive Target Tracking Computer

- **Frequency measurement and relay.** Symmetry function [15] is used to determine peaks (reference points) of the power system signal and measure its frequency. The symmetry function requires very long word arithmetic and very fast multiply-and-accumulate operation, which is achieved by custom solution in the form of dedicated hardware circuit. Frequency calculation, and subsequently calculation of the rate of change of frequency, is based on the knowledge of the reference points. It is performed using software solution running on a simple general-purpose processor, as well as some other non-hard real-time functions. Frequency relay is Internet enabled and the link to Internet is also implemented in software. Algorithm development was performed using Matlab, the analysis of word length and accuracy using C/C++ program, and then the algorithm was translated into RTL VHDL design, which was directly synthesised for an FPGA implementation. Two soft core 32-bit processors (NIOS [ ]) are used in the system, one for frequency calculation and communication with the external environment via Internet, and the other one for testing and debugging purposes (e.g. generation of test inputs in real-time).

- **Self-tuning regulator** (STR) for second order systems [3] using recursive least squares (RLS) algorithm for system parameter identification and on-line system controller design as illustrated in Figure 3. A testbed has been developed in which MatLab or C++ model of the plant communicates with a parameterised model of the STR implemented on an FPGA and that communicates with the plant (process) simulator via a PCI-bus and shared memory. Parameter estimation circuit performs RLS algorithm using custom fixed-point or floating-point arithmetic units and MAC units. Number of these units, that may work in parallel, depends on the target performance of the STR. The approach enables very fast prototyping of the STR in a realistic testbed environment, where all STR functionalities are prototyped in FPGA device, and all system (plant) environment, as well as facilities for analysis of system behaviour are programs running on the usual PC platform.

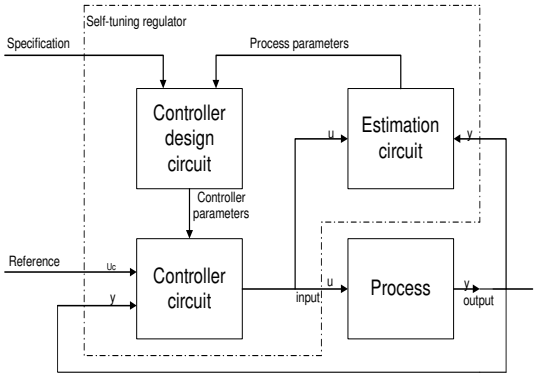

Figure 3 STR structure

- **Smart antenna for DS-CDMA** system using four and six antenna elements and least square direct spreading/despreading algorithm [10]. Full implementation of the adaptive algorithm with system-level design in MatLab, then Simulink and direct synthesis of majority of the system directly from Simulink (via VHDL) onto an FPGA. Matrix inversion is imple-

mented in the FPGA using systolic array architecture. Adaptive smart antenna beamformer has been integrated into a Simulink CDMA system simulation environment and verified as a part of the whole transmitter/receiver system.

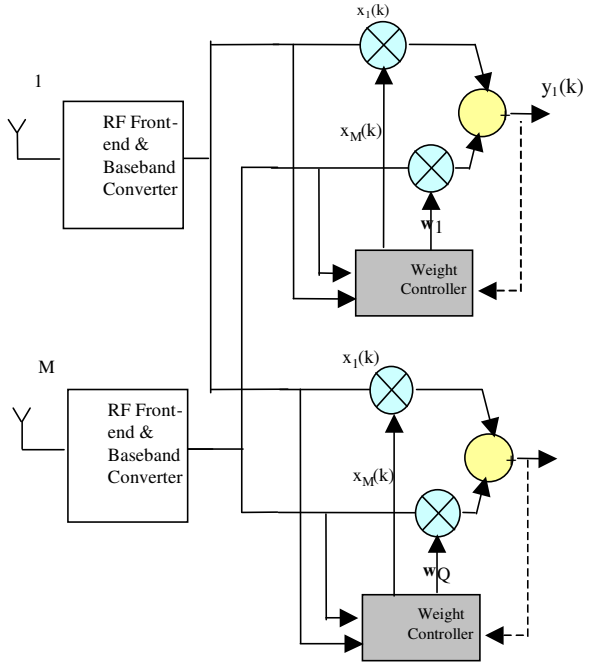

Figure 4 Smart antenna structure

- **Radial Basis Function Neural Network (RBFNN) smart antenna**. The system implements smart antenna beamforming based on direction-of-arrival estimation [20]. After modelling the application in MatLab and then C/C++, program profiling gave an insight into critical operations if the beamforming is performed using sequential execution. As it proved impossible to meet timing requirements, a number of accelerators have been designed by trading software for hardware in order to meet systems requirements. Solution is based on hardware/software partitioning and on a System-on-Programmable-Chip (SoPC), where FPGA-based hardware functional units are used to implement complex matrix operations, autocorrelation, and neural network implementation, and the rest of algorithm is implemented using software running on the softcore 32-bit microprocessor implemented in the same FPGA. The system is illustrated in Figure 5.

The common feature of all these applications is that their portions are well-defined DSP algorithms with regular processing of input signal samples. However, all of them also contain control-driven elements that require decision-making and even modification of the basic algorithms "on-the-fly". It was the main reason that software parts of the solutions become almost unavoidable.

A common part of the design methodology was to actually perform prototyping of the solution on an FPGA-based platform. Effective prototyping always included connection of the solution prototype with its environment which is most often simulated on a PC platform in the form of MatLab or programming language model, together with facilities for intermediate results collection, storage and on-line or off-line analysis. It proved to be invaluable, as prototypes gave the best insight into solution behaviour.

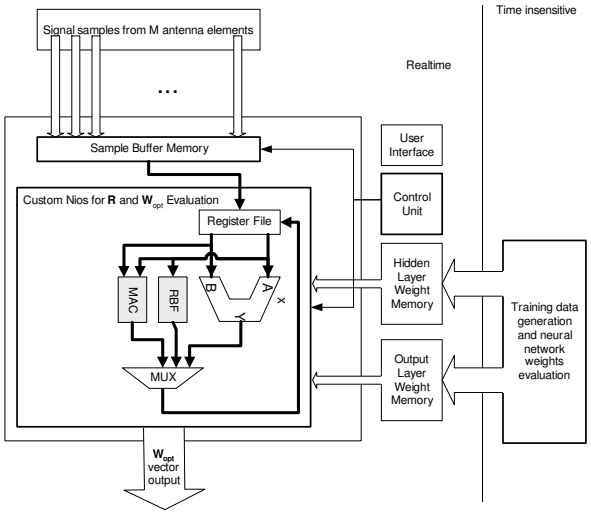

Figure 5 RBFNN smart antenna

4. TARGET PLATFORM PERSPECTIVE

Technology capabilities to integrate a few hundred millions of transistors on a single chip, with Moore's law still in action for at least medium-term, enable implementation of extremely complex systems that satisfy real-time application requirements. As the complexity of the algorithms increases, the development effort also increases. RTL level has already become too low for complex designs thus requiring transfer to behavioural description and synthesis [4, 5] or even further, system level description and synthesis [18, 19]. Target implementation technologies, such as ASICs and processor (DSP or general-purpose), do not satisfy requirements of many applications.

ASICs typically contain some non-programmable hardware (e.g. special-purpose algorithm engines). Some ASIC systems can be implemented on a single chip (System-On-Chip, SoC), with memory, peripherals, special I/O and they may also contain one or more processor cores. DSP cores also have been integrated into ASICs. ASICs incorporating a DSP core can integrate interface logic, peripherals, specialized datapaths, memory, and other functions onto a single integrated circuit. This approach provides opportunities for extremely high levels of integration, which can result in smaller products, lower production costs, improved performance, and low power consumption. As DSPs find their way into more products, production cost, and power consumption constraints, ASICs incorporating DSP cores are becoming very popular. How-

ever, despite the advantages of ASIC DSP system in terms of power consumption, speed and cost, the ASICs have long development cycles, and the development of chips is costly and time consuming. Another drawback of ASICs is that their hardware/software partitioning and integration are challenging compared with other platforms.

Today, many general-purpose processors (GPPs) have strong DSP capabilities. The very high performance of GPPs for PCs and workstations (Pentium4, PowerPC and Ithanium) make them increasingly suitable for DSP tasks. GPPs typically have much higher clock speeds than most DSPs; they have very good tool support for major processors; and their cost-performance can rival floating-point DSPs (Pentium4 probably is faster than any floating-point DSP currently available). For DSP applications, nearly all GPPs today include some support for signal processing, often in the form of SIMD instruction set extensions. The software development tools are also mature and powerful. Finally, the 32-bit GPP is easier to target for many non-DSP tasks. However, the efficiency of GPPs still cannot match DSP processors. Most of GPPs have high energy use with poor cost-performance compared to fixed-point DSPs. The GPPs are also weak on integration. Another drawback of GPPs is the unpredictable execution times making their use difficult for real-time applications.

Reduced Instruction Set Computer (RISC) processors use pipelining to minimise instruction cycle. The RISC processor has advantage of faster instruction execution and a number of high-speed instructions, which can be brought together to perform a complex DSP instruction. However, in order to achieve efficient RISC programming for DSP applications, the DSP algorithm needs to be split into many interleaved tasks or data streams to keep the pipeline full as much as possible.

DSP processors are the specific chips designed to implement typical DSP applications efficiently. Unlike other platforms, DSP processors have introduced many powerful architectural enhancements to make specific DSP algorithms run faster. These enhancements include MAC units, Harvard architecture to separate program and data accesses, pipelining, very long instruction word (VLIW), single instruction-multiple data (SIMD), efficient execution of loops and efficient accesses to typical data structures used in DSP algorithms. The DSP processor has the advantages of relatively low development cost, fast development cycles and relatively easy-program software tools. However, comparing with ASICs, DSP processor has higher power consumption.

FPGAs, as a more flexible technology have been successfully used in many DSP embedded systems. A qualitative comparison of different implementation technologies is presented in Table1. Many attempts have been done to combine those traditional technologies when implementing embedded solutions. Recently, FPGAs have enabled integration of both "software" implementation part of an application together with "hardware" parts in System-on-Programmable-Chip (SoPC) [1, 21]. This new opportunity allows the experimenting with new architectural solutions on low executable-level, which is ideal for prototyping purposes. But an increasing number of applications use FPGAs as the final implementation, especially as they move to below 100nm technology. Those architectural solutions include multiple processor cores and dedicated logic, operation accelerators and other approaches to meet system requirements. They allow design of the architecture for the algorithm and exploit the available parallelism.

Energy efficiency remains a problem, but it is very often offset with high solution flexibility. The implementation cost of FPGA-based solutions is reduced by using low feature size technologies that enable integration of huge number of transistors, which are not most efficiently used, but perform their function. In this sense Moore's law works in favour of FPGAs in terms of application volume versus cost.

FPGAs are the commercially available technology that offers customisation of systems that contain both hardware and software and also provide a form of real (dynamic) reconfigurability, which gives a perspective of reducing power consumption by dynamically reusing FPGA resources.

Table 1 DSP implementation technologies

Feature	ASIC	Structured ASIC	FPGA	Reconfigurable Hybrid	DSP	General-purpose µP
Operating freq. (MHz)	> 1000	>1000	100 – 400	50-300	100-600	100-1000
Power consumption	Moderate	Moderate	High	Moderate	Very High	Very High
Parallel execution	Maximum	Maximum	Maximum (flexible)	Moderate	Serial (ILP)	Serial (no spec. FUs)
Complexity of designs	Very high (50-100M gates)	Very high	Very high	Moderate	Very complex - programs	Very complex - programs
Size/area	Large	Large	Very large	Moderate to high	Moderate to High	Moderate to High
Migration/ evolution	None	Low	High	Moderate to High	High (perform. limited)	High (performance limited)
Customization	Difficult	Moderate	Easy	Easy to Moderate	Very Easy	Very Easy
Design verification	Very difficult	Moderate	Moderate	Moderate to difficult	Moderate	Moderate
Design tools	Good, very expensive	Good, moderately expensive	Very good, less expensive	Very poor	Good, inexpensive	Very good, inexpensive

Structured ASIC devices probably most closely resemble gate arrays as they are pre-manufactured and mask-customised by the addition of only a few metal layers. In addition to random logic fabric, structured ASIC devices come with a number of available larger-scale intellectual property (IP) blocks pre-designed on the device, thus reducing design time, testing, and integration time. Performance and power consumption can be comparable to a full ASIC implementation. Compared with an FPGA, a structured ASIC implementation offers potentially lower unit cost, higher performance, lower power consumption, and a smaller footprint.

Structured ASICs, cell-based ASICs and FPGAs, or some combination of these, further emphasise platform-based design. Combination of commonly used components into platform elements that allow design assembly in the same manner as traditional board design, however, appears to be the current as well as the future trend. Technology vendors create more and more competitive differentiation based on collections of IP and application-domain-specific devices tailored to specific markets. Platforms that combine traditional architectures with reconfigurability are emerging mostly from the research projects, giving us a basic idea of the possible future options that will be used in new classes of embedded DSP systems and generally can be viewed as new kind of reconfigurable hybrid architectures [13].

5. ALGORITHM PERSPECTIVE

DSP algorithms are often first described using some formal language (e.g. MatLab, programming language). Recognition of parallelisms in a such an algorithm is usually very difficult if not impossible. Still, most DSP algorithms have inherent parallelisms, which, once recognised, may be described in various ways. The problems appear when trying to express an algorithm, which is parallel, by using programming languages, which are usually sequential. Even if the language has expressiveness for parallel operations, it is usually the responsibility of the designer to recognise and decide how to describe parallelisms [7].

The architectures and platforms onto which we map an algorithm, however, are inherently parallel (except some instruction-set-architectures, which we refer to as "software"). For example, FPGAs and ASICs enable implementation of parallel architectures. If we target an instruction set architecture (ISA), then the problem is how to map the algorithm on the architecture. A compiler and the underlying model of computation are used to perform this mapping.

When using standard programming languages, a parallel algorithm is first mapped to a sequential program and then mapped to a parallel architecture. Obvious goal would be to avoid two translations and perform direct mapping of the algorithm to the parallel architecture. A further (ideal) goal would be to partition a parallel algorithm and map it onto a combination of sequential program(s) and parallel HDL and then on a hybrid architecture automatically.

6. SPECIFICATION LANGUAGE PERSPECTIVE

Various languages are used to specify embedded systems behaviours and structures:

- Programming languages, such as assembler, C/C++, Java, are used to describe primarily algorithmic behaviours and are aimed for execution on "software" platforms (DSPs, GPPs).
- Hardware description languages enable RTL level descriptions, but also are suitable to describe behaviours, and hierarchies in embedded systems (VHDL, Verilog).
- Data-flow languages more naturally describe data-flow oriented systems (SDF), but lack facilities to describe irregular, primarily control-driven, behaviours.
- Hybrid languages (such as Esterel, SystemC) are aimed at the mixed heterogeneous systems which combine data-driven and control-driven features.
- Graphical languages are used to express reactive systems (StateCharts, SyncCharts, Statemates) or data-driven applications (Simulink).
- MatLab, the common engineering modelling language, is used for direct synthesis of both software targeting microprocessors and hardware [12].

Languages are designed with the aim to cover all or nearly-all classes of applications, but do not cover sufficient breath of models of computation [11]. No single specification/capture tool is suitable to describe all types of applications. Heterogeneous embedded applications require the use of multiple tools/languages. Also, languages are architecture limited and inefficient, as they do not map (synthesise) automatically to all target architectures. This leads to a need for the application-specific (or domain-specific) languages and specification entry mechanisms, which are easy to learn – easy to use and enable efficient synthesis.

7. DESIGN VALIDATION

Design validation must be repeated at several abstraction levels. An ideal case would be correctness by construction, which is possible only for relatively simple designs. Some tools, such as Esterel Studio [6], offer formal verification of behaviours. Still, they are restricted to a specific and limited class and size of systems. Simulation and co-simulation are standard approaches based on exhaustive testing of the system behaviour upon presenting test vectors to the system inputs. Due to the increased complexity of designs, exhaustive simulation becomes impractical and non-feasible. Various co-simulation techniques, which enable efficient co-simulation of software and hardware, are desirable.

Functional simulation is usually performed on high-level language model (MatLab or programming language), but the mappings performed to implement an embedded solution introduce many approximations, which also require validation. As RTL level models are usually those which are synthesised into final embedded solutions, verification of these models is crucial to show validity of the implementation.

When prototyping or implementing a system, final verification, which requires variables, states and behaviour monitoring, becomes crucial. The built-in test facilities typically used in ASICs are not sufficient for FPGA-based or complete So(P)C solutions. Interaction of hardware and software becomes difficult to verify and debug. Embedded instrumenta-

tion (sometimes called virtual instrumentation) increasingly becomes important in monitoring and checking timing characteristics of the application [1, 21].

Although new high-level languages and tools produce executable models that can be simulated and checked against functional and some other system requirements, there is still a need for prototyping the most critical operations to validate algorithms and behaviours. Purpose of the prototyping is typically twofold: (1) it provides more realistic synthesised models which are either identical or very close to the final implementation and (2) it speeds up execution bringing it to near real-time and thus may replace simulation and co-simulation (prototyped models may perform the orders of magnitude faster than simulated models and speed up the verification and design cycle).

Prototyping platforms that utilise combinations of general-purpose microprocessors, DSP processors and hardware-implemented functions have become common. Some of the FPGA-based prototyping systems enable complete system prototyping including software and hardware-implemented functions on a single programmable chip (SoPC). They also provide good facilities for platform-based designs. As they are based on the use of own and third-party IP, they further stress the importance of on-chip instrumentation. They allow checking the operation of processor cores implemented in actual hardware and their interaction with third-party and own IP used to implement systems. Because software is executed on the real hardware and interacts with real hardware-implemented functions, the designer is able to verify that the design does work, rather than simulating it to see if it should work.

8. SYNTHESIS PERSPECTIVE

Algorithms described using either sequential or concurrent languages must be transformed into executable models for the underlying architecture (processor for software and the rest to hardware). The new hybrid languages (Esterel, SystemC) are flexible supporting mapping on either hardware or software architecture, but still leave open the issue of hardware/software partitioning and co-design, which are usually performed in an interaction with the designer. A key element of new design methodologies is to decompose the problem and conquer by:

- separating functional level (what the system is supposed to do) from the architecture level (how it is done) and
- separating computation tasks (and behaviours) from communication tasks (between behaviours)

Many issues appear when mapping a functional description on an architecture such as accuracy of computation induced by a type of arithmetic (fixed-point, floating-point, number of bits used to represent data), power consumption, etc. Synthesis from hybrid high-level languages is not yet sufficiently efficient, although algorithm and system validation is much faster. Simulink to synthesis [1, 21] solution is similar to RTL design, although on a higher structural level. The design flows are based on libraries of pre-designed parameterised DSP functions, which can be mapped directly onto optimised FPGA implementations for the particular FPGA technology. These functions generate high-performance implementations of the critical parts of the datapath while hiding the low-level details from the designer. If the datapath represents a serial combination of those predefined functions, then there is almost straightforward translation from programming language representation of the algorithm onto Simulink representation and automatic synthesis. However, the approach fails if the system requires more decision-making based on its own dynamics and/or interaction with its environment based on random requests and events.

On the other hand, traditional processor architectures are suitable only for sequential execution, as they do not support real parallelisms. Also they support only interrupt-based or polling-based reactivity. We proposed an approach to system synthesis based on combination of reactive processor core [16, 17] and dedicated functional units for intensive data-driven portions of an application. The approach is suitable for systems that can be represented as hierarchy of a "master" (configurable processor) module (MP) and a number of functional units (FU), where the role of the MP is to implement main system thread and activate FUs as necessary.

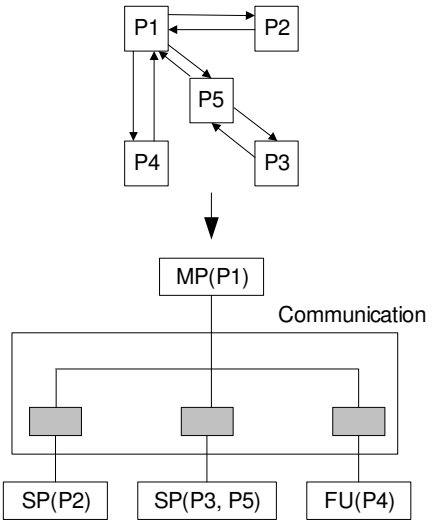

Figure 6 Mapping specification onto architecture

Now, we extend the approach to the architecture that may contain processor cores instead of functional units as it is illustrated in Figure 6, where SPs on the lower hierarchical level can be configurable processors of the same kind as the MP. By adding a communication layer, implemented as shared memory blocks (or just registers), the system becomes a multiprocessor suitable for FPGA implementation. The role

of the master processor is executing the main thread of an application and coordination and communication between multiple threads that run on SPs and FUs. In this way communication function is completely separated from computation functions that are implemented in concurrent threads implemented on SPs and FUs.

Figure 6 illustrates a possible mapping of a hypothetical system specification, described by five concurrent behaviours (P), onto our architecture that consists of three processor cores and one hardware functional unit. For system specification we defined a low-level language with primitives that directly support most of the features of Esterel [2] including reactivity and concurrency. Specification language implies manual partitioning of functionality and allocation of behaviours on modules, but subsequently enables automatic synthesis of individual behaviours and their communication. It is especially suitable for applications in which computation functions map on software that runs on processor cores.

9. CONCLUSIONS

Embedded DSP systems represent a huge class of applications that require multiple languages, tools and technologies to describe and implement. Traditional ways of implementing embedded DSP applications using DSP processors and ASICs are used in majority of applications, but have limitations which can be overcome only by using different implementation and architectural solutions (FPGAs, combinations of other technologies etc).

New methodologies are emerging for more efficient system specification, simulation, modelling and synthesis. Still, integrated approaches are necessary to "glue" design phases and enable more consistency between those phases. Prototyping remains one of the key steps as it provides an insight into system operation. Design flow is becoming more complex and difficult to automate. It also introduces many trade-offs. We believe that a better link between high-level specification languages and underlying execution architectures represents a key in making the design flow simpler and fully automated and the resulting embedded systems more reliable from the very beginning.

REFERENCES

1. Altera, Available at www.altera.com
2. G. Berry and G.Gonthier, "The ESTEREL synchronous programming language", *Sc. Comput. Prog.*, 19; 87-152, 1992
3. J. Cao, Z. Salcic and S.K. Nguang: "A Floating-point All Hardware Self-Tuning Regulator for Second Order Systems", *Proceedings of IEEE Region 10 Technical Conference on Computers, Communications, Control and Power Engineering*, vol 3, pp. 1733 -1736
4. D. Gajski, and L. Ramacahndran,: "Introduction to high level synthesis", IEEE Design and Test Computer, October 1994.
5. Elliott, J.P.: *Understanding Behavioral Synthesis: A Practical Guide to High-Level Design*. Kluwer Academic Publishers, 1999.
6. Esterel Technologies, Available at www.estereltechnologies.com
7. Handel-C, Available at www.celoxica.com
8. J. Hurk, and E. Dilling: "System Level Design, a VHDL Based Approach".,Proceedings of Euro-DAC 1995.
9. Jerraya. A.A, et al.: *Behavioral Synthesis and Component Reuse with VHDL.* Kluwer Academic Publishers, 1996.
10. L. Jin, Z. Salcic, C. F.Mecklenbräuker: "Simulation Model of the LS-DRMTA Adaptive Algorithm for Multiple Antenna System for DS-CDMA", International Conference on Communication Systems, ICCS 2002, Singapore, 2002
11. E.A. Lee and A. Sangiovanni-Vincentelli, "A framework for comparing models of computation," *IEEE Trans. CAD*, vol. 17, pp. 1217–1229, Dec. 1998.
12. MathWorks Inc., Matlab, Simulink and Stateflow. Available at www.mathworks.com
13. Quicksilver Technologies, Available at www.qstech.com
14. Z. Salcic and C.R. Lee: "FPLD-Based Automatic Tracking Estimation Computer", *IEEE Trans. On Aerospace and Electronic Systems*, Vol. 37, No 2, 2001, pp. 699 – 706
15. Z. Salcic and R. Mikhael, "A new method for instantaneous power system frequency measurement using reference points detection", *Elsevier Science Electric Power Systems Research* Journal 55 (2) (2000) pp. 97 - 102.
16. Z. Salcic, P. Roop, M. Biglari-Abhari and.A. Bigdeli, "REFLIX: A Processor Core for Reactive Embedded Applications", *Lecture Notes in Computer Science,* Volume 2438, pp 945-954, Springer 2002
17. Z. Salcic, P. Roop, M. Biglari-Abhari and.A. Bigdeli, "REFLIX: A Framework of a Novel Processor Core for Reactive Embedded Applications", accepted for publication, *Elsevier Journal of Microprocessors and Microsystems*, 28, 2004, pp. 13-25
18. SpecC. Available at www.ics.uci.edu/~specc/
19. SystemC v2.0.1 Functional Spec, www.systemc.org
20. W. To: "Neuro-adaptive smart antenna for 3G cellular communication systems", ME Thesis, University of Auckland, 2004
21. Xilinx, Available at www.xilinx.com