

A NEW VECTOR PROCESSOR ARCHITECTURE FOR HIGH PERFORMANCE SIGNAL PROCESSING

Andreas Bolzer, Gerald Krottendorfer and Manfred Riener

On Demand Microelectronics, Design Center Techgate
Donau City Str. 1, A-1220 Vienna, Austria (Europe)
phone: +43 1 2697985-68, fax: +43 1 2697985-20, email: andreas.bolzer@ondemand.co.at
web: www.ondemand.co.at

ABSTRACT

The continuous and fast development in communication technology demands high flexibility and short product cycles. In addition, demanding signal processing algorithms often exceed the performance of available digital signal processor (DSP) cores. This paper introduces a novel vector signal processor architecture that can achieve these processing and flexibility demands. In contrast to traditional vector processors, our design improves the utilization of scalar algorithms.

1. INTRODUCTION

Signal processing applications, which are utilizing physical resources by converging to the theoretical limit given by Shannon's law are based on algorithms that have an ample necessity for high processing power [1]. These applications often exceed the processing power of existing DSPs which are fulfilling low power and cost constraints [2]. As an example digital subscriber line (DSL) standards are implemented with hardwired cores. Their development is a long process which incurs high costs. The extreme substantial complexity of modern designs increases the probability of design errors and the risks at the market entry of new products while increasing development time and costs [3].

Due to these facts some architecture requirements are proposed in the following:

- Flexible programmability is necessary to enable the product developer to react on diverse and changing customer requirements. Programmability extends the product life cycle and enables to offer products with increased flexibility.
- Scalable architecture is essential as it meets the processing power needs of specific applications.
- Design of product families instead of single products. The presence of a real scalable IP core offers the possibility to design a product family with different processing power needs and therefore decreases development costs dramatically [4]. E.g. a single design of an MPEG decoder can be manufactured to fulfill processing power needs for low resolution palmtop applications as well for high resolution DVD player applications.
- Customized silicon area to meet cost requirements of markets with keen competition.
- Low power design to meet specified power consumption requirements.
- Usage of a tested intellectual property (IP) core to minimize integrated circuit (IC) design risks and to shorten development cycles and costs.

- Availability of software development tools to ease and shorten the development of products.

All these requirements are fulfilled by the *Vector Signal Processor* (VSP). It is a fully-tested IP-core for high-performance low power applications. The VSP combines different processor architecture concepts resulting in a revolutionary new processor.

This paper will introduce the VSP. Section 2 plots several DSP architectures with their inherent drawbacks and presents the VSP design as a suitable solution. The configuration of the VSP architecture according to the requirements is addressed in section 3, where an application implementation is depicted in section 4. Section 5 gives an overview of the VSP programming and a conclusion finishes this paper.

2. VSP ARCHITECTURE

Different methods are used to achieve a higher computing power for DSPs. Since the clock rate is limited by the current technology, we focus on the DSP architecture in this paper [5]:

- Super pipelining architecture: The arithmetic units are connected like a chain. The pipeline has to be filled to provide high efficiency. In case of program jumps the pipeline has to be refilled causing decreased processing performance.
- Super scalar architecture: Several short pipelines are arranged in parallel. The parallel entities must share resources (e.g. data memories), causing bottleneck situations which decrease the performance.
- Multi DSP: Multiple processors are linked with data interfaces. The necessary synchronization effort reduces the computing efficiency. In addition some resources (e.g. DMA and interrupt controller, timer, etc.) are duplicated without being utilized. Multi DSP solutions usually are not appropriate for low power mass market products.

The Vector Signal Processor is a new approach that avoids the drawbacks as described above. The VSP architecture is a fusion of a single instruction multiple data (SIMD) and a multiple instruction multiple data (MIMD) architecture. SIMD uses a single broadcast instruction, while MIMD may use different instructions for each data path [6]. The VSP picks up both architectures concurrently providing a very flexible and very efficient processor core.

2.1 Slice Architecture

A so called *slice* contains a calculation entity. Each slice, as depicted in Fig. 1, consists of two independent *data mem-*

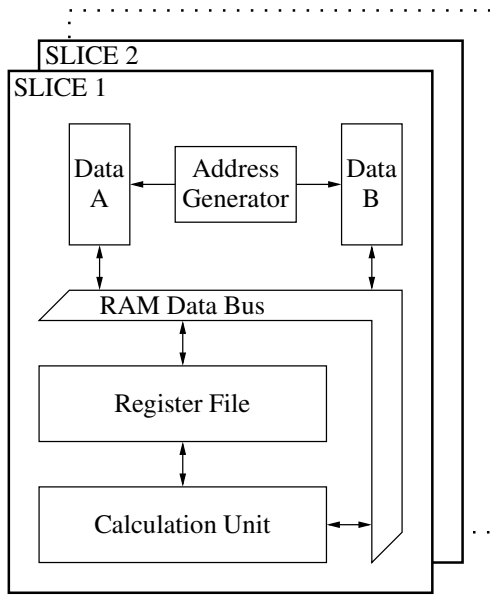


Figure 1: Structure of a Single Slice

ories and an address generator, which enables modulo and bit-reversed addressing. Both memories can be accessed simultaneously, enabling read and write transfers in the same cycle. The data bus connects the memories with the register file and the fixed-point calculation unit. A set of registers provides input data for the calculating instructions. Some registers are directly connected to the corresponding registers of the neighbor slices to perform fast data transfers. The calculation unit contains of adders, multipliers, shifters, logical entities, alignment and rounding units. Results can either be stored in the data memories, transmitted or processed in the global arithmetic unit. The section below will describe this issue.

2.2 Global Architecture

A block diagram of the VSP is shown in Fig. 2. Several slices described in section 2.1 are arranged in parallel. The global arithmetic unit enables a synchronous processing of the partial results from several slices. E.g. a dot product operation is implemented by using a global adder. The global data bus links all slices, the global arithmetic unit, and input/output ports together.

The program sequencer provides the program counter that points to a very long instruction word (VLIW) in the program memory. Each slice is controlled by its own instruction word that is located inside the VLIW. Global instructions, which are also provided by the VLIW, manage global data transfers, jumps and loops. Conditional jumps are feasible and depend on the arithmetic unit state of a chosen slice. In addition conditional executions of each slice can be performed. Since each slice contains its individual program, different algorithms can be executed in parallel. Attention should be drawn to the fact that the slices don't share resources, since each slice consists of entities for calculation and data storage.

Due to the common control unit all slices are tightly coupled resulting in an inherent synchronization. This leads to a

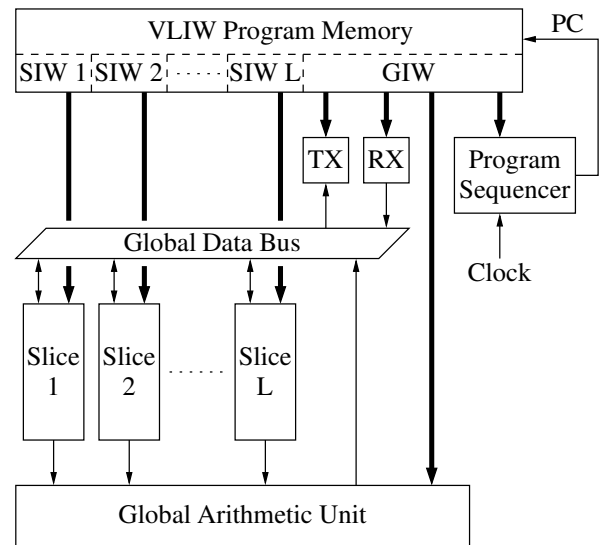


Figure 2: VSP Architecture and Program Control; VLIW...very long instruction word, SIW...slice instruction word, GIW...global instruction word, PC...program counter, TX...transmit channels, RX...receive channels; The thick lines represent control paths

saving of clock cycles and enables easier code generation of parallel algorithms. The imposition of a unique program flow to all slices can be considered as a drawback. This fact is no constraint for actual applications, since the flexible method of partitioning code to the parallel slices guarantees maximum processor utilization.

3. ARCHITECTURE CONFIGURATION

The VSP provides a processor architecture for efficient implementations of scalar and vector algorithms. Flexible partitioning of the algorithm to a variable number of slices offers many degrees of freedom to optimize the processing efficiency.

3.1 Partitioning

Imagine a least mean square (LMS) update of an finite impulse response (FIR) filter [7]:

$$\mathbf{h} = \mathbf{h}_{old} + e\beta\mathbf{x}, \quad (1)$$

where e is the error signal and β is a measure for the adaption speed. The vectors \mathbf{h} and \mathbf{x} are filter coefficients and input signal of length N , respectively:

$$\mathbf{h} = [h(0), h(1), \dots, h(N-1)]^T,$$

$$\mathbf{x} = [x(n), x(n-1), \dots, x(n-N+1)]^T.$$

With the updated coefficients \mathbf{h} the output of the filter is determined by

$$y(n) = \mathbf{x}^T \mathbf{h}. \quad (2)$$

This algorithm should run on the VSP. Since every slice consists of two data memories, one contains the history of \mathbf{x} and

the other the coefficients \mathbf{h} . To split the processing power to two slices we rewrite Eq. 2

$$y(n) = \mathbf{x}_1^T \mathbf{h}_1 + \mathbf{x}_2^T \mathbf{h}_2 \quad (3)$$

with

$$\mathbf{h}_1 = [h(0), h(1), \dots, h(N/2 - 1)]^T,$$

$$\mathbf{h}_2 = [h(N/2), h(N/2 + 1), \dots, h(N - 1)]^T,$$

$$\mathbf{x}_1 = [x(n), x(n - 1), \dots, x(n - N/2 + 1)]^T,$$

$$\mathbf{x}_2 = [x(n - N/2), x(n - N/2 - 1), \dots, x(n - N + 1)]^T.$$

The update of \mathbf{h}_1 and the calculation of $\mathbf{x}_1^T \mathbf{h}_1$ is executed in slice 1, where the update of \mathbf{h}_2 and the calculation of $\mathbf{x}_2^T \mathbf{h}_2$ is done in slice 2. The global arithmetic unit adds the results of both slices to gain $y(n)$. For the history update the last value $x(n - N/2 + 1)$ of slice 1 must be stored as first value $x(n - N/2)$ of slice 2. This is achieved through the slice interconnections without additional synchronization demand. Notice that the number of calculations per slice is now reduced by half. By further splitting the processing power to L slices

$$y(n) = \sum_{k=1}^L \mathbf{x}_k^T \mathbf{h}_k, \quad (4)$$

the number of calculation cycles can be reduced significantly. Many sub-tasks run in parallel in order to shorten the time-to-solution for the main task to be executed.

3.2 Scalability

The previous section shows the principle of algorithm partitioning to several slices. The number of slices L can be configured arbitrarily to meet the required processing power of a specific application. This scalability enables modular tailoring of the VSP, which is considered by an example.

An implementation of an LMS adapted FIR filter on the VSP showed that

$$N_C = \frac{2N}{L} + 6 \quad (5)$$

clock cycles per sample are needed, where L is the number of slices. The constant 6 represents an amount of clock cycles for initialization and loop unrolling. Additional clock cycles will be saved if algorithms are merged. Assume a processor clock rate f_{CLK} . The maximum number of processed data symbols per second is now

$$f_{SYM,max} = \frac{f_{CLK}}{N_C}. \quad (6)$$

Fig. 3 depicts several curves representing the maximum symbol rate that can be achieved for a VSP running at $f_{CLK} = 100$ MHz. This rate is determined by the number of filter coefficients N and the number of slices L . If a specific application requires a filter with 128 coefficients and a symbol rate of 4 megasamples per second, Fig. 3 shows that 16 slices

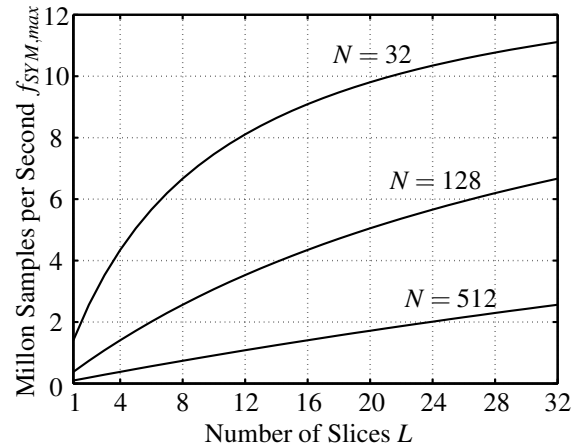


Figure 3: LMS FIR Filter: Max. data symbol rate as a function of filter length N and number of slices L . VSP clock rate $f_{CLK} = 100$ MHz

are sufficient. Each of the slices process $N/L = 128/16 = 8$ coefficients and input history data words. Only 19 words of VLIW lines have to be used for the program. In combination with the efficient data memory usage, silicon die size can be optimized. The required data word width (8...32 bit) may also be considered before manufacturing. All these considerations result in a shortened development of cost optimized systems.

Another dimension of scalability is the implementation of several VSP cores on one IC, which are linked together through a bus system. Each VSP executes one part of a complex system.

4. APPLICATION IMPLEMENTATION

Fig. 4 depicts a block diagram of a simplified part of an *Single-pair High-bit-rate Digital Subscriber Line* (SHDSL) transceiver [8]. Two input signals are treated by one vector and four scalar algorithms. A possible partitioning for a 4-slice VSP is depicted in Fig. 5. The echo canceller algorithm (LMS adapted FIR filter) is processed in all slices simultaneously. Each scalar algorithm is treated in a separate slice. The arrows inside the box represent data transfers.

Due to the common global unit, which tightly connects all VSP units, no clock cycles are wasted for slice synchronization and data transfers. For scalar algorithms with similar complexity, all slices run with maximum load. To optimize the utilization of scalar algorithms to the parallel slices, it is possible to process several simple scalar algorithms subsequently in one slice or to split one complex scalar algorithm to different slices.

5. PROGRAMMING

The parallel slices of the VSP can be programmed with a C-like assembly language. It is easy to learn and enables the generation of optimized code for time-critical signal processing applications. The instruction set is optimized for typical applications that need high processing power, e.g. filter calculations, Fast Fourier Transform (FFT) [9], Viterbi algorithm [7].

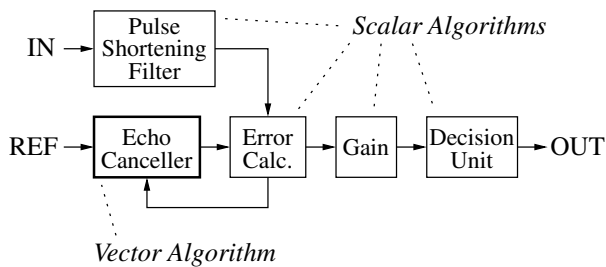


Figure 4: Block Diagram of an Application Example

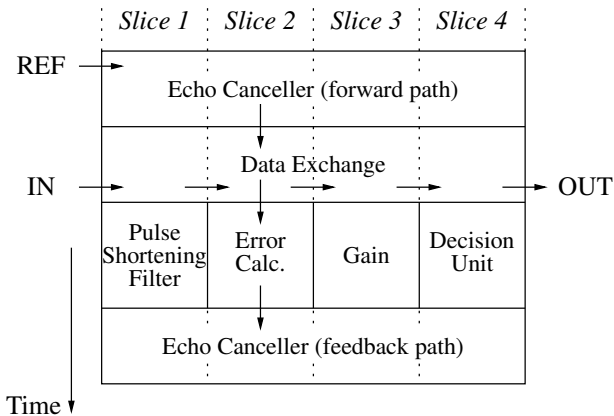


Figure 5: Algorithm Partitioning of an Application Example.

Assuming the application example of section 4, the following coding scheme is executed in one clock cycle. Each slice declaration part contains MIMD instructions that calculate the four scalar algorithms as depicted in Fig. 5:

```
{slice[1]
  instructions pulse shortening filter
slice[2]
  instructions error calculation
slice[3]
  instructions gain
slice[4]
  instructions decision unit
global
  global instructions}
```

Vector algorithms often contain equal slice instructions that can be bundled to SIMD instructions:

```
{slice[1:4]
  instructions echo canceller
global
  global instructions}
```

The global declaration part defines SIMD instructions of the global architecture units as well as instructions for the global program sequencer.

6. CONCLUSION

This paper describes a novel signal processor architecture. Each parallel calculation entity (slice) comprises all resources, which are needed to enable self-containing signal processing. By configuring the number of slices, the VSP

can be easily tailored to the actual processing power needs of a certain application. Therefore this architecture does not suffer from architectural processing power limits as of standard DSPs. A global arithmetic unit enables post calculation of a selected number of slice results. E.g. dot products are processed in the global arithmetic unit using a multiple input adder. All slices are tightly coupled by a global control unit. As a result internal data transfers does not need any synchronization, which saves clock cycles. This processor is therefore suitable for high performance real time applications with high data throughput, e.g. for broadband communication or multimedia applications.

There are certain limitations due to the fact that a single global control unit controls the program flow of all arithmetical units. In this paper we could show that this is not constraining in practical applications, because algorithms can be flexibly assigned to the slices. Therefore, the VSP architecture can guarantee, that all slices are always fully utilized. The VSP instruction set is optimized for communication algorithms with the highest demand of processing power like FFT or Viterbi decoder. Minimized chip area and low power design of the VSP emphasizes this processor to substitute hardware non programmable signal processing applications, which suffer from high development costs and risks. A C-like assembly language has been developed, ensuring an optimal load balancing of all architectural units of the VSP.

REFERENCES

- [1] M. Holzer, P. Belanovic, B. Knerr, and M. Rupp, "Design methodology for signal processing in wireless systems," *Informationstagung Mikroelektronik 2003, Vienna*, Oct. 2003.
- [2] Berkeley Design Technology, "Buyer's guide to DSP processors." Available from: <http://www.bdti.com>, 2003.
- [3] B. R. Wiese and J. S. Chow, "Programmable implementations of xDSL transceiver systems," *IEEE Comm. Mag.*, pp. 114–119, May 2000.
- [4] G. Sivard, *A Generic Information Platform For Product Families*. PhD thesis, Royal Institute of Technology, Stockholm, Sweden, 2000.
- [5] V. K. Madiseti, *VLSI Digital Signal Processors*. IEEE Press, Butterworth-Heinemann, 1995.
- [6] M. J. Flynn, "Very high-speed computing systems," *Proc. IEEE*, vol. 54, pp. 1901–1909, Dec. 1966.
- [7] E. A. Lee and D. G. Messerschmitt, *Digital Communication*. Kluwer Academic Publishers, second ed., 1994.
- [8] M. Riener and A. Bolzer, "Programmable implementations of xDSL transceiver systems," *Proc. European Signal Processing Conf.*, 2004.
- [9] A. V. Oppenheim and R. W. Schaffer, *Discrete-Time Signal Processing*. Prentice Hall, second ed., 1999.