A Quick Safari Through the Reconfiguration Jungle

Patrick Schaumont
IMEC
Leuven, Belgium
schaum@imec.be

Ingrid Verbauwhede
UCLA Dept of EE
Los Angeles, CA, USA
ingrid@ee.ucla.edu

Kurt Keutzer
UCB Dept of EECS
Berkeley, CA, USA
keutzer@eecs.berkeley.edu

Majid Sarrafzadeh
UCLA Dept of CS
Los Angeles, CA, USA
majid@cs.ucla.edu

ABSTRACT

Cost effective systems use specialization to optimize factors such as power consumption, processing throughput, flexibility or combinations thereof. Reconfigurable systems obtain this specialization at run-time. System reconfiguration has a vertical, a horizontal and a time dimension. We organize this design space as the reconfiguration hierarchy, and discuss the design methods that deal with it. Finally, we survey existing commercial platforms that support reconfiguration and situate them in the reconfiguration jungle.

1. INTRODUCTION

Programmability and reconfigurability are considered to be a key ingredient for future silicon platforms [1]. The support of this flexibility requires a dedicated and specialized toolset [2, 3]. Despite that, reconfiguration is not yet generally recognized as a separate axis of design. A generic definition of reconfigurable computing is available [4], but a system level view on the reconfiguration process has been missing. Also, the design data models and the computational models needed to represent reconfiguration effectively are still being researched [5].

In this paper we look at reconfiguration from the system-design point of view. This includes a generic definition of the reconfiguration process (Section 2) and an enumeration of the system design technologies used to support it (Section 3). In Section 4, an overview of several existing platforms is given.

2. RECONFIGURATION HIERARCHY

2.1 Design Space

To motivate the origin of the reconfiguration hierarchy, consider an FPGA that runs a soft Intellectual-Property (IP) core running a protocol stack. Clearly, the protocol stack is a program, as it consists of soft-core instructions. The FPGA however merely treats this program as data that is being processed on the reconfigurable fabric. The meaning of this data varies with the abstraction level. A further complication originates when we would change the protocol stack program dynamically depending on the type of protocol being processed. While this can be called reconfiguration, it is not of the same kind as the bitstream configuration on the FPGA.

Reconfigurable systems introduce multiple levels of programming and design. As a consequence, also the configuration data itself occupies multiple levels. Besides hierarchy in configuration, there is also a hierarchy of processing as well as in data representation. For instance, the routing table in a packet routing engine (Figure 1) is a system configuration, while an FPGA bitstream containing a queuing algorithm inside of the same routing engine is an architecture configuration. The router itself is logically organized as a (logical or physical) hierarchy of machines, each layer controlling a lower level and processing finer data granularity.

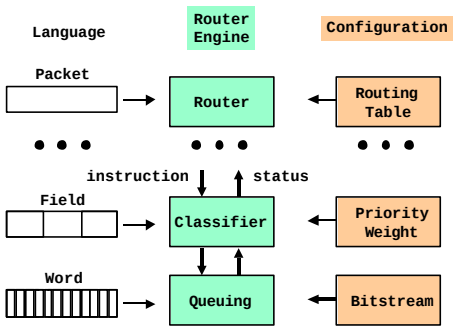

Figure 1: Reconfiguration Levels in Router

We define a design space of three orthogonal axes to describe (re)configurable systems: A *vertical axis* that expresses the level of abstraction, a *horizontal axis* that expresses the reconfigurable feature diversity, and a *time axis* that expresses the timing relationship of configuration to processing.

The *vertical axis* is related to the level of abstraction. For an application developer this is closely coupled to the idea of a virtual machine or interpreter that fetches instructions and executes them in terms of primitives at lower abstraction level. For an IC designer the vertical axis is associated with the hierarchy of the granularity of the computing primitives of the device. At the lowest level we naturally recognize logic primitives (gates), simple storage (registers) and routing. At higher levels, the micro-architecture, instruction-set architecture and process architecture (or systems architecture) represent additional layers of computation abstraction. Reconfiguration is applicable to each new abstraction layer that is introduced.

While the vertical axis describes a hierarchy, the *horizontal axis* describes slices of the hierarchy. Each level of the hierarchy is made up of a combination of communication, storage and computation. Reconfiguration can affect each of those individually. In table 1, an enumeration is given of different such design elements (horizontal) characterized at different design levels (granularities). The term *coarse grain* and *fine grain* reconfigurability are usually associated with the variation of horizontal features at the architectural level [6].

The *binding time* expresses when configuration data is send to the processing part. Each level of the hierarchy can be bound

38th DAC, June 18-21, 2001, Las Vegas, Nevada, USA.

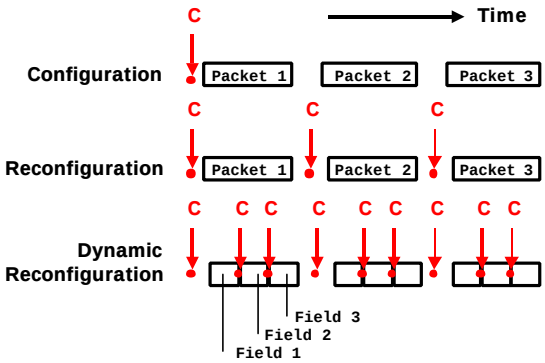

Figure 2 : Binding Time

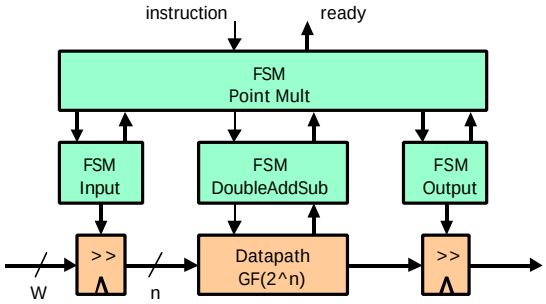

Figure 3: Elliptic Curve Encryption Processor

individually. We distinguish implementation-time binding and design-time binding. With implementation-time binding, configuration is postponed until actual execution of the processing part is required. With design-time binding, configuration is done at the moment that the processing part is conceived. This is equivalent to hard-coding. These terms are preferred over the more traditional *run-time* and *compile-time* since the latter ones are not unique for hierarchical systems. In order to have a physical implementation, the lowest processing level of a system is always design-time bound. The top level of programmable systems is always implementation-time bounded. In between, there is a smooth transition called the *binding time continuum* [4].

It is useful to relate processing activity to binding time. For the packet routing engine example shown earlier we can consider how often reconfiguration is done for each processed packet. It allows distinction of *configurability* from *reconfigurability* and *dynamic reconfigurability*. Refer to Figure 2. The *C* arrows indicate points of configuration. Configurability indicates the possibility to fix configuration data once and for all subsequent received packets (typical use would be configuration once per silicon implementation). Reconfigurability allows changing the implementation in between any received packet (typical use would be configuration once per computational task). Finally, dynamic reconfigurability applies new configuration data at a faster rate then the reception of packets, for instance per each processed packet field (typical use would be configuring every 100 – 10,000 cycles of execution).

2.2 Design Example

We now give an example of a hierarchical system and point out how reconfiguration is useful at different levels of the hierarchy. Elliptic-curve public key cryptography is based on the operations on points of a specific curve in a finite field, the so-called underlying field. The point multiplication is the fundamental operation for the key agreement protocol. The Diffie-Hellman key agreement protocol works as follows [7]: given a point P on the curve, Alice will compute *a.P*, and Bob will compute *b.P*. Alice receives *b.P* and computes *a.b.P*. Bob receives *a.P* and computes *a.b.P*. They now share a secret key *a.b.P*. The assumption is that an eavesdropper, who has access to *a.P* and *b.P* cannot compute *a.b.P* because the discrete logarithm problem is a hard problem in the elliptic curve group. The algorithm can be implemented across different abstraction levels. At the highest level, the point multiplication k.P is executed, where k is an integer and P is a point on the elliptic curve. The point multiplication can be decomposed into doublings, additions and subtractions of points on the elliptic curve. These primitive operations on points of the elliptic curve can again be decomposed in operations on elements of the underlying field. These operations are the addition, the division, the multiplication and the squaring of elements of the underlying field.

The architecture of this elliptic curve crypto-processor is shown in Figure 3. This processor has been designed with a hierarchical controller [8]. At the top layer, an FSM *PointMult* accepts an instruction stream that allows programming of algorithm parameters as well as initiating the point multiplication k.P. The next layer arranges three FSM. *Input and Output* match the word-length characteristics of the encryption algorithm (n bits) to that of the system (W bits), while FSM *DoubleAddSub* implements primitive elliptic curve operations. The data-path implements those in operations of the underlying field.

Table 1 : Examples of Configurable Design Elements

	Communication	**Storage**	**Processing**
Implementation	Switches Muxes	RAM Organization	CLB Parametrizable IP-block
Micro-Architecture	Crossbar Busses	Register File Size Cache Architecture	Execution Unit Type Interpreter Levels
Instruction Set Architecture	Size of address/data bus	Register Set Memory Architecture	Custom Instructions Interrupt Architecture
Process Architecture/ Systems Architecture	Interconnection network	Buffer Size	Number and type of asynchronous processes and tasks

The advantages of a hierarchical control specification have been advocated before [9]. They include better control on complexity and increased reuse opportunities. In addition, we note here that reconfiguration is useful at each level of abstraction.

- The architecture can support different field sizes (*n* values) by making the data-path reconfigurable. Typically *n* is 200 bits. Fine-grain reconfigurability allows supporting the optimal architecture for each polynomial length.
- A reprogrammable FSM DoubleAddSub receives micro-instructions from the FSM PointMult and translates this in a sequence of micro-instructions for the data path.
- A reprogrammable FSM PointMult allows changing of the elliptic curve multiplication algorithm itself. The instruction set allows reprogramming of the elliptic curve as well as the irreducible field polynome.

3. DESIGN TECHNOLOGIES

This section briefly overviews how design technologies support the types of reconfigurability discussed above. The aim of these technologies is (a) to identify the required reconfigurability, and (b) to effectively exploit it. Figure 4 demonstrates the general approach to achieve this. Starting with an application *domain*, we profile a set of applications to identify common and computationally intense kernels (domain primitives). The parametric implementation of these kernels forms the building blocks of the reconfigurable platform.

In the second step a single application out of the application domain is mapped, manually or automatically, onto the reconfigurable portions of the platform. The granularity of these computational kernels varies significantly as do the techniques for doing the mapping. The remainder of the section considers design technology support for these two steps at different levels of abstraction.

3.1 Instruction Set Architecture

First we consider configurations of the instruction set architecture. Specialized operations have to be included in a system through a careful trade-off of area, throughput and or power consumption. The following two examples illustrate this point.

Example 1: The instruction set of general-purpose microprocessors has the widest coverage. The power consumption is however several orders of magnitude higher than programmable DSP processors such as the ones used in cellular phones. The reason is that DSP processors have the right amount of programmability for the application domain of wireless communications [10]. This includes specialized instructions to accelerated Viterbi decoding and even turbo decoding. Example systems that use instruction-set configuration are given in section 4.1.

Example 2: The FPGA supports the most generic type of reconfiguration: implementation-level reconfiguration. A power breakdown of the FPGA shows that 65% of the power consumption is associated with the interconnect [11]. This is a high price for general reconfigurability, especially when considering applications that have high operation regularity (like DSP) and thus only have low routing requirements.

Consequently there is a need for domain specific processors that provide sufficient reconfiguration and/or reprogrammability within the application domain, yet at the same time don't need and cannot afford the general reprogrammability. Examples are wireless communications, multimedia, packet processing in routers and switches, signal processing, encryption, and so on.

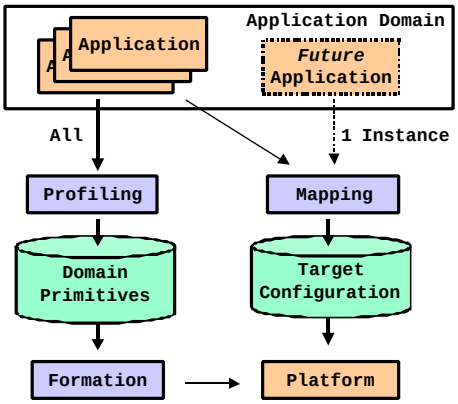

Figure 4: Design of Reconfigurable Platforms

There are four areas of possible specialization in a processor: the *data paths* that perform the actual operations, the *memory* architecture, the *interconnect* architecture and the *control* architecture.

Profiling can be used to identify the right processor specializations. In *first-order* profiling, a set of representative algorithms from an application domain is executed and the type and quantity of operations per algorithm needed is collected. This information then is used in the instruction set design process. But more advanced strategies are possible. Future generations of instruction sets will have explicit communication and dependence information embedded in the instruction. This is because of the dominance of interconnect and memory bandwidth in deep sub-micron technologies, and because many of the application domains are heavily throughput driven. Such dependence information can be derived from a *higher-order* profiling step that also considers operation data dependencies next to operation types, and is a topic of current research.

3.2 Micro-Architecture Level

Flexibility in reconfigurable systems comes at the expense of the reconfiguration time (including synthesis, place and route). To circumvent that, we can move the system hierarchically as described in Section 2. At any given level of hierarchy, one can think of the basic building blocks of a reconfigurable architecture as parameterized functional blocks that are pre-placed within a fully reconfigurable fabric. We call these *Versatile Parameterizable Blocks (VPBs)*. VPBs are programmable with minimum amount of input. At the lowest level of hierarchy, VPBs are equivalent to CLBs and the highest-level VPBs are capable of performing complex functions (as described in Section 2). When implementing an application, the VPBs will perform operations that can be mapped onto these fixed blocks; computational blocks that will be instantiated on the fully reconfigurable portion of the chip will perform the remaining operations. (For a more detailed description of VPBs see [12].)

We demonstrate this concept with an example in image processing. Repetitive arithmetic operations on a large amount of data can be very efficiently implemented using hardware. First, algorithms that

have common properties and operations are grouped together. Such algorithms can use a common set of VPBs for their implementation. The algorithms and the classes that they belong to are summarized in Table 2.

An example VPB that is useful for image processing is the Filter Operations Block shown in Figure 5. This block takes five parameters that define its operation. It can be used for an iterative image restoration algorithm, and several other filtering operations such as mean computation, noise reduction, high pass sharpening, Laplace operator, and edge detection operators (e.g. Prewitt, Sobel). The mask-coefficients array holds the values of the coefficients. Parameters B, A, and c take the value 0 for all the functions except the iterative image restoration algorithm.

Given a fabric with a collection of VPBs, one can use compiler techniques to map a given algorithm/specification to this heterogeneous architecture. An initial mapping can be extracted based on profiling techniques. In present-day design, a final manual intervention is needed to obtain an optimized design.

We performed a preliminary study into application profiling. The starting point is high level code, e.g. C/C++/Fortran, of a set of applications that is compiled into control dataflow graphs (CDFGs). This can be done using the SUIF compiler [13] as a front-end and Machine-SUIF [14] to take the SUIF Intermediate Format (IR) to control flow graphs (CFG). From there, one can implement a pass to translate the CFGs to CDFGs and perform some simple profiling.

Multimedia applications are known to have a high level of parallelism, which can be exploited for performance on hardware. Therefore, we examined the MediaBench application suite [15]. We profiled files from the suite which performed the actual multimedia operations e.g. FFT, motion detection, etc. Table 3 presents the results for simple operation sequences. The notation OP1-OP2 denotes that OP2 directly uses output from OP1 i.e. there is an edge {OP1, OP2} in the CDFG.

The percentage listed for the potential VPB operations is $\frac{num\_seqeunce(OP1, OP2)}{total\_num\_ops}$; therefore the sum of the percentage across a single application will exceed 100%. We can gather a lot of information using this simple profiling. For example, the sequence of operations deviates from probability theory as the sequence MUL-ADD is found with much greater frequency than ADD-MUL. Probabilistically, these sequences should occur in the same proportion. Additionally, the profiling shows that the MUL-ADD sequence should be implemented as a VPB as it is widely used across all the applications.

Table 2: Classification of Image Processing Algorithms

Algorithms	Operations	Class
Image Restoration, Mean Computation, Noise Reduction, Sharpening/Smoothing Filter	Weighted Sum, addition, subtraction, multiplication	Filter Operations
Image Halftoning, Edge Detection	Comparison	Thresholding
Image Darkening, Image Lightening	Addition, Subtraction	Pixel Modification

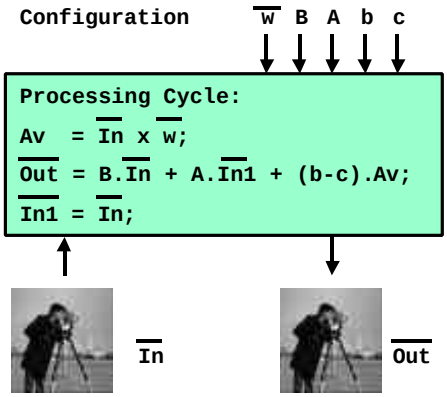

Figure 5: Filtering Block

4. PLATFORM SURVEY

Having presented a taxonomy of reconfigurable devices and given some indication of design support for these devices, we now survey reconfigurable platforms and (do our best to) place them in our taxonomy. As several good surveys of academic reconfigurable platforms are available [16,17,18,6], we concentrate on the commercial ones. Table 4 lists both academic and commercial platforms, along with pointers to more detailed information. This set is not exhaustive. Due to the rapid evolution in this area we can only claim to have a representative selection of devices.

4.1 Commercial Configurable Platforms

The architectural elements that are most naturally configurable are processor blocks. These blocks are not intended to be fully autonomous devices, but are intended to be included as Intellectual Property (IP) blocks on a silicon die. In the basic parameters of the processor, we can define three types of vertical configurability: architectural, micro-architectural, and implementation-level configurability. Architectural configurability means that the actual programmer's view of the processor is configured in different ways. For example the number of registers or even the instruction set may change. Micro-architectural configurability means that the functional unit organization can change. For example the number of arithmetic-logic units to implement an instruction may change. Finally, implementation configurability means that the physical

Table 3 Profile of Operations and Simple Combinations

	MediaBench file name				
	motion	getblk	adpcm	convol	jctrans
ADD	50.3%	44.5%	45.3%	64.8%	84.6%
MUL	36.3%	24.0%	9.4%	22.2%	13. 8%
Potential VPB Operations					
MUL-MUL	0.0%	1.3%	0.0%	0.0%	0.0%
ADD-ADD	14.5%	3.2%	0.0%	0.0%	9.1%
ADD-MUL	0.0%	0.6%	0.0%	0.5%	0.4%
MUL-ADD	36.3%	21.5%	9.4%	20.1%	13.0%

implementation may change. For example one physical implementation of the same architecture may be more power-efficient, while another may be more performance (speed) oriented.

The ARC Tangent A4-RISC core and the Tensilica Xtensa micprocessor each illustrate all three types of configurability. All IP blocks delivered in soft (i.e. in a hardware description language) format may have their implementation configured by the RTL synthesis tools and the place and route tools that create their physical implementation. Thus these processors have configurable implementations.

Each of the Tangent and Xtensa processors are based on simple RISC cores. These processors also offer per-instance horizontal micro-architectural configuration options such as size and organization of caches or the number and priority of interrupts.

The most significant configurability of these processors comes from instruction-set architectural configurations. Starting with a base instruction set of 29 instructions the ARC Tangent user can add application specific instructions such as multiply-accumulate (MAC) or normalize without writing any HDL. The Tensilica Xtensa and ARC Tangent both offer set instruction extension packages for specific domains such as DSP. ARC offers two types of use-level instruction extensibility. In the first type the user chooses from a set of pre-defined instruction set extensions. For these instructions the software environment including simulator, assembler, and compiler are automatically regenerated. The designer may also augment the ARC with an arbitrarily complex co-processor. For such augmentations more user intervention is required to generate the software environment.

The Xtensa also supports user-defined instruction extensions. Tensilica places limits on the extensibility of instructions but fully generates the software environment for these instructions. Using either type of instruction extensions users are able to gain from 6X (JPEG) to 100X (Viterbi decoding) speed improvement over a standard RISC microprocessor.

VLIW processors are another class of processors that are well suited for configurable applications. In such processors there are a significant number of architecture parameters that can be varied while still adhering to the same VLIW system concept and design environment. Examples are the number and type of data-paths, the interconnect-architecture, the controller architecture and so on. AR/T Designer from Frontier Design is an environment that creates (configures) VLIW processors starting from a C++ description.

The Improv Programmable Systems Architecture® offers configurability at both the instruction-set architecture and the process architecture(or systems architecture) level. Designers may configure the individual VLIW Jazz processor by adding designer-defined computational units. These units are scheduled and controlled in an integrated fashion with the other execution units. Multiple Jazz processors may be then integrated together with integration blocks to form a multi-processor Integration Systems Architecture®.

Integration of processors together with IP blocks on a single die led to the predominance of the system-on-a-chip IC. With configurable processors such as the ARC Tangent, Tensilica Xtensa, and the Improv Jazz, a new type of platform is introduced that supports integration at the instruction level. The resulting platforms rival ASIC speeds in an IC that may be entirely programmed in a traditional C-based software environment.

Table 4 : (Re)configurable Platforms

	Platform	Vertical[1]	Reference
Commercial	Excalibur, Altera	I, M	http://www.altera.com
	Virtex, Xilinx	I	http://www.xilinx.com
	Xtensa, Tensilica	I,M,ISA	http://www.tensilica.com/
	Tangent, ARC Compiler	I,M,ISA,P	http://www.arccores.com/
	Frontier Design	P	http://www.frontierd.com/
	Chess, Target	ISA	http://www.retarget.com/
	Jazz, Improv Systems	I,ISA,M,P	http://www.improvsys.com
	MECA 4I, PMC-Sierra	P	http://www.pmc-sierra.com/
	Morphics	P	http://www.morphics.com/
	E7/A5, Triscend	ISA,P	http://www.triscend.com/
	FPSLIC, Atmel	I,P	http://www.atmel.com/
	CS2112, Chameleon Syst.	M,P	www.chameleonsystems.com
Academic	Garp, UCB	I, ISA, P	http://brass.cs.berkeley.edu/
	PipeRench, CMU	M	http://www.ece.cmu.edu/research/piperench
	MorphoSys, UCI	I,ISA,P	http://www.eng.uci.edu/morphosys
	SPS, UCLA	I,M	http://www.cs.ucla.edu/~elib/reconfigurable
	C-RISP, KULeuven	ISA	http://www.acca.be/

([1]) I = Implementation, ISA= instruction set architecture, M = Micro-Architecture , P = Process architecture

4.2 Commercial Reconfigurable Platforms

Configurability is powerful but in configurable devices the architecture and implementation are fixed during semiconductor processing. Many designers will want the flexibilility of deferring configuration decisions until actual field deployment or even runtime.

FPGA's deal with reconfigurability at implementation level. Xilinx introduced these devices over 15 years ago and now offers differentiated product families optimized for high performance, low power consumption or low cost.

The Chameleon Reconfigurable Communications Processor is targeted towards wireless communication applications such as cdma2000, W-CDMA and UMTS. Reconfiguration is possible at the micro architecture level, although an on chip ARC core also supports process-level reconfiguration. The reconfigurable fabric of the CS2112 is organized as four processing slices, each consisting of three tiles and each tile consists of seven 32-bit datapath elements and two-multipliers. This offers 84 32-bit datapath elements and 24 multpliers. Two planes of configuration memory are present and accessible from the ARC core. Compiler techniques have been presented [3] to map a single C program thread on a combination of the fabric and the core, and to optimize reconfiguration overhead. Product literature indicates this device is able to handle cdma2000 Chip Rate Processing of 50 channels. The device also allows for

relatively inexpensive dynamic reconfiguration and algorithms exploiting reconfiguration every 1000 to 10,000 cycles are in development.

By driving specialization further across the micro-architecture level and into the task-level, platforms for specific application areas are obtained. For example the PMC-Sierra MECA 4I™ is targeted for Voice-over-IP applications and comes with its own firmware for reconfiguring the device. The advantage of reconfigurability is realized when the user downloads firmware upgrades in the field. Compared to Chameleons' CS2112, the MEC 4I™ offers relatively fewer but more powerful coarse-grain datapath-oriented reconfigurable elements. Another example of coarser-grain reconfigurability is provided by Morphics, which offers both packaged stand-alone reconfigurable devices targeted for the wireless base-station market as well as reconfigurable IP blocks. Architecturally the standalone Morphics device is organized as a series of reconfigurable kernel processing elements controlled by a general-purpose processor. Each reconfigurable element is also coarse-grained and datapath-oriented.

Another recent trend is the combination of a fine-grain reconfigurable fabric with a general-purpose processing core. The resulting system provides implementation level reconfiguration of the fabric, combined with task-level reconfiguration of the core. The fabric is memory mapped to the core. One example here is the Triscend E5 and A7 CSOC (Configurable System-on-a-Chip) families. Each platform supports different processor cores (8032 MCU/E5;ARM 7/A7) and application types. Atmel offers the FPSlic based on the AVR micro-controller. Altera combines embedded microprocessors (ARM, MIPS) and peripherals with reconfigurable logic in the Excalibur™ family.

Finally, it should also be mentioned that reconfigurable blocks are also being offered as IP blocks that can be embedded by an end-customer IC developer. Besides Morphics, reconfigurable logic vendors Actel with its new VariCore™ program and Adaptive Silicon with its new Programmable Logic Core™ are taking this direction. This will allow one more mechanism by which the reconfiguration hierarchy can proliferate in the ASIC landscape.

5. CONCLUSIONS

Reconfigurable systems introduce an exciting new dimension of programming, spanning all abstraction levels of a system. New design technology is needed to offer an effective use model for these systems. On the commercial side, it is encouraging to see that many of the product offerings are application focused and rely on reconfigurabilty as simply an efficient mechanism to provide application performance. As a result reconfigurable technology is moving from academic research to a solutions oriented commercial technology.

6. ACKNOWLEDGEMENTS

The authors would like to thank the companies mentioned in this paper for providing us with product details and useful insights. The authors are also grateful for the organizational contributions of Diederik Verkest at IMEC.

7. REFERENCES

[1] *ASIPs: Get ready for reconfigurable silicon*, M. Santarini, http://www.eetimes.com/story/OEG20001120S0028

[2] *Xtensa Application Specific Microprocessor Solutions*, Overview Handbook pp. 59-64, http://www.tensilica.com

[3] A Compiler Directed Approach to Hiding Configuration Latency in Chameleon Processors, X. Tang, M. Aalsma, R. Jou, FPL 2000, pp. 29-38, 2000.

[4] Reconfigurable Computing: What, Why, and Implications for Design Automation, A. Dehon, J. Wawrzynek, DAC99, 1999.

[5] *Stream Computations Organized for Reconfigurable Execution (SCORE)*, E. Caspi, M. Chu, R. Huang, J. Yeh, Y. Markovskiy, J. Wawrzynek, and A. DeHon, FPL 2000, pp. 605-614, 2000.

[6] A Decade of Reconfigurable Computing: A Visionary Perspective, R. Hartenstein, Proc DATE '01, Munchen, March 13-16, 2001.

[7] E. Dewin, B. Preneel, "Elliptic Curve Public-Key Cryptosystems: An Introduction," LNSC 1528, Springer-Verlag, June 1997, pg. 131-141

[8] A Hardware Implementation of Elliptic Curve Public-Key Cryptosystems, W. Borremans, P. Gijsels, MS thesis, KULeuven, July 2000.

[9] *SpecCharts: A VHDL front-end for embedded systems*, F. Vahid, S. Narayan, and D. Gajski, IEEE Trans CAD, 14(6):694-706, June 1995.

[10] *Low Power DSP's for Wireless Communications*, Ingrid Verbauwhede, Dave Garrett, ISLPED 2000, pp. 303-310, August 2000.

[11] *The design of a low energy FPGA*, V. George, Hui Zhang, Jan Rabaey, ISPLED 1999, pp. 188-193, San Diego, 1999.

[12] *SPS: A Strategically Programmable System*, S. Ogrenci, E. Bozorgzadeh, R. Kastner and M. Sarrafzadeh, Proc. RAW 2001, April 2001, San Francisco.

[13] *The SUIF 2 Compiler System*, Stanford University Compiler Group, http://suif.stanford.edu/suif/suif2/.

[14] *The Machine SUIF SUIFvm Library*, Glenn Holloway and Michael D. Smith, Technical Report, Division of Engineering and Applied Systems, Harvard University, 2000.

[15] *MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems*, Chunho Lee, Miodrag Potkonjak and William H. Maggione-Smith. Technical Report, Computer Science Department, University of California, Los Angeles, 2001

[16] *The flexibility of configurable computing*, J. Villasenor, B. Hutchings, IEEE Signal Processing Magazine, pp 67-84, September 1998.

[17] The Roles of FPGA's in Reprogrammable Systems, S. Hauck, IEEE Proceedings, pp. 615-638, April 1998.

[18] *Reconfigurable Instruction Set Processors: A Survey*, F. Barat and R. Lauwereins, pp. 168-173, Proc. RSP 2000.