Comparative Review of Common Reconfigurable Architectures

Woo Hyong Lee, Arindam Saha, Eun Ji Lee, and Sung Bae Park
SoC R&D Research Center
SYSTEM LSI Division, Samsung Electronics Corp.
Kiheung, Korea
E-mail: {woohyong.lee, s.arindam, eunji.lee, sung.park}@samsung.com

Abstract *- Some may claim that general-purpose computers are reconfigurable in the sense that functional units are reused for different computational tasks at different times, with multiplexers controlling the routing between these units. However, in this paper as well as in the research community at large, the term reconfigurable computing refers to systems where the hardware can be customized and changed periodically to execute different tasks on the same hardware.*

In this paper we survey the reconfigurable computing landscape and make some recommendations. The landscape can be partitioned into two parts – one that is spearheaded by University research and the other that is taking shape in the industry. We describe one example from each, and provide a comparison of some reconfigurable architectures. We conclude by making some recommendations about the architectures, software tools, and applications of reconfigurable computing.

Keywords: *Reconfigurable, Processor, Compiler, SDR.*

1 Introduction

Current computing devices constitute two extremes. On one hand, we have conventional CPUs that rely heavily on one or more complex ALUs, which make frequent calls to large memory resources, but suffer from lack of performance because the architectures do not follow the structure of the task. These CPUs provide flexibility at the cost of performance. On the other end of the spectrum, we have ASICs that provide optimal performance and power for specific tasks but are very inflexible because they are useless for any other task. The quest for *both* ASIC-like performance and CPU-like flexibility leads to *reconfigurable computing*.

Reconfigurable computing, as a concept, dates back almost 40 years [1]. Some may claim that general-purpose computers are reconfigurable in the sense that functional units are reused for different computational tasks at different times, with multiplexers controlling the routing between these units. However, in this report as well as in the research community at large, the term reconfigurable computing refers to systems where the hardware can be customized and changed periodically to execute different tasks on the same hardware. SRAM-programmable Field Programmable Gate Arrays (FPGAs) are the first real implementations of the reconfigurable computing concept. But FPGAs haven't been able to satisfy the needs for dynamically reconfigurable flexible processing for a variety of reasons.

In this paper as well as in the research community at large, the term reconfigurable computing refers to systems where the hardware can be customized and changed periodically to execute different tasks on the same hardware.

The rest of the paper is organized as follows. In Section 2, we describe one representative project being carried out in the academia and list a variety of University projects in reconfigurable computing that are not discussed in this paper. Section 3 deals with one representative reconfigurable computing machine that has been developed in the industry and, then list a group of commercial efforts not discussed in this paper. In Section 4, we compare a number of reconfigurable examples using a variety of metrics, and we list the pros and cons of each. We conclude with some recommendations meant to provide a platform for further discussion.

2 University Research

Universities around the globe have been engaged in reconfigurable computing research for more than a decade. In this paper, we focus on research being carried out at some of the US Universities. These projects are primarily funded by the US Department of Defense. In fact, in 1990's DARPA had funded several Universities research groups to the tune of close to $100M under the auspices of the *Adaptive Computing Systems* program. As a

representative example we discuss the PipeRench architecture from the Carnegie Mellon University.

2.1 Carnegie Mellon PipeRench Project

The PipeRench project at the Carnegie Mellon University has created a programmable datapath for numerically intensive applications.

Architecture

Figure 1 shows the PipeRench architecture [2]. The fabricated chip is organized as 16 *stripes*, each stripe containing 16 processing elements (*PE*s). As shown in Figure 1, the stripes are connected, using their register files, to create an interleaved ring topology [2].

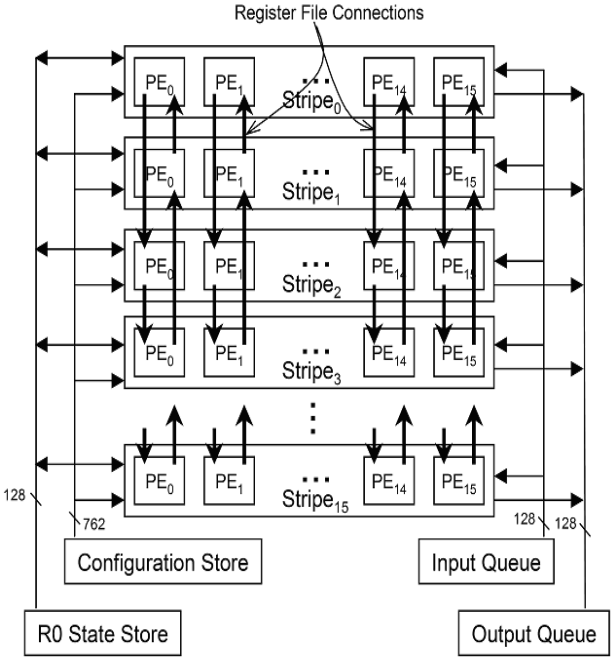

Figure 1: The PipeRench Architecture.

PipeRench uses a technique called *pipeline reconfiguration* to *virtualize* the hardware. Assuming that a stripe is a pipeline stage, one can map an *n*-stage virtual design on to a *m*-stage physical pipe, where $n > m$. This is achieved by storing the configuration bits of the entire virtual hardware on chip, and moving these bits to the physical fabric every cycle. This way, although the chip has 16 physical stripes, it can support up to 256 virtual stripes. For virtual hardware larger than real hardware, physical stripes will eventually be reconfigured with new virtual stripes. The state of over-written virtual stripes are saved in R0 into the R0 state store memory.

The PE block diagram is illustrated in Figure 2 [2]. A PE is 8-bit wide, but adjacent PEs can be connected to perform operations of wider widths. There are eight registers per PE, called *pass register file*. There is one dedicated register per register file that can be used for intra-stripe feedback and therefore must be stored and restored. Output of the ALU can be stored in any one of the eight registers. If the value is not written to a register, then the value from the corresponding register in the previous stripe. This reduces the amount of state because data that travels through the pipeline need not be saved. The functional unit consists of eight 3-input LUTs that are identically configured. 42 configuration bits are required to specify the functionality of a PE.

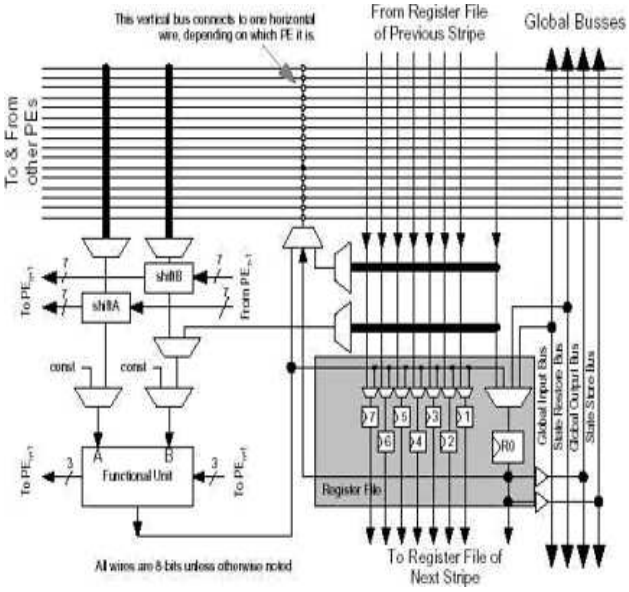

Figure 2: Block Diagram of a PE in PipeRench.

DIL Compiler

A hardware synthesis compiler, called the *DIL compiler*, has been developed at Carnegie Mellon that targets the PipeRench architecture [3]. The source language for the compiler is DIL that can be used as an intermediate language in a HLL compiler. But we do not have any information about such a compiler. So, for now, we assume that the programmer has to program in DIL to use PipeRench. DIL is more like a HLL than a HDL. It is a *single assignment* language and so any variable can be assigned to only once. The DIL compiler follows the following steps: First it reads the PipeRench architecture

description. In evaluation phase, the compiler inlines all modules, unrolls all loops, and generates straight-line, single assignment code. That code is then converted into a hierarchical dataflow graph. Then the compiler goes through a number of passes including a variety of optimizations. The key feature of the DIL compiler is the place and route step that uses a deterministic, linear-time, greedy algorithm [3]. Finally, the code generator produces PipeRench assembly language ready to be executed.

2.2 Other University Projects

Other notable University projects in the reconfigurable computing area that are not covered in this paper are: MorphoSys from the University of California Irvine [4], BRASS/Garp from the University of California Berkeley [5], U.C. Berkeley Pleiades [6], Northwestern Chimaera [7], MIT MATRIX [8], and University of Washington RaPiD [9].

3 Commercial efforts

There is a tremendous amount of activity within the industry as far as reconfigurable computing is concerned. Almost every major company, including Intel, NEC, Toshiba, Sun, etc., have active programs in this area. A number of startups have been funded in the last few years to push the envelope of reconfigurable computing. As a representative example, we discuss the NEC DRP architecture.

3.1 NEC DRP

NEC is one of the few established companies that announced a Dynamically Reconfigurable Processor (*DRP*) architecture [10].

Architecture

The NEC DRP is organized as *tiles*, each tile, as shown in Figure 3 [11], includes, among other things, 64 byte-oriented Processing Elements (PEs) organized as an 8x8 array, Configurable Horizontal Memory (HM) blocks that are 8bx8kW with 1 R/W port, Configurable Vertical Memory (VM) blocks that are 8bx256w with 1 R and 1 R/W ports, A State Transition Controller (STC) which is a simple sequencer controls tile reconfiguration, and External ports to which one can attach complex operation units like multipliers, external memory controllers and peripheral bus controllers like PCI.

The DRP-1 prototype silicon announced [10] has 8 tiles for a total of 512 PEs, 160kb and 2Mb of VM and HM respectively, 8 STCs, 8 32b multipliers, and 1 memory controller and 1 PCI controller.

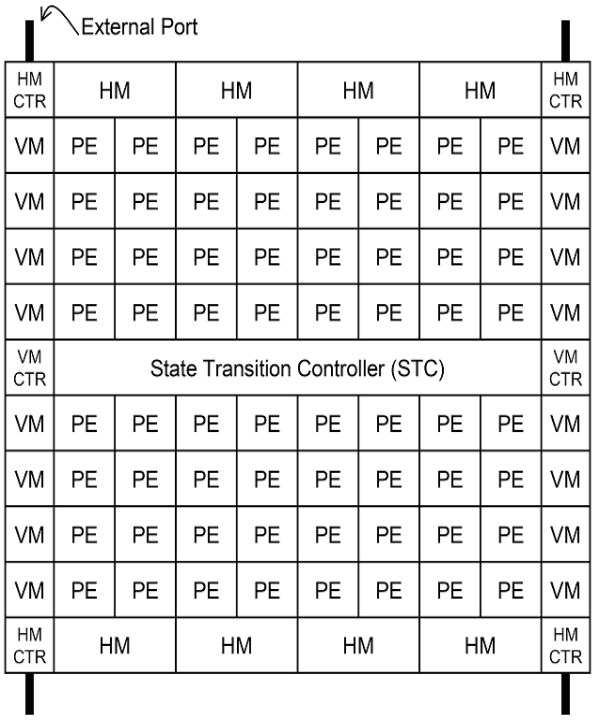

Figure 3: One NEC DRP Tile.

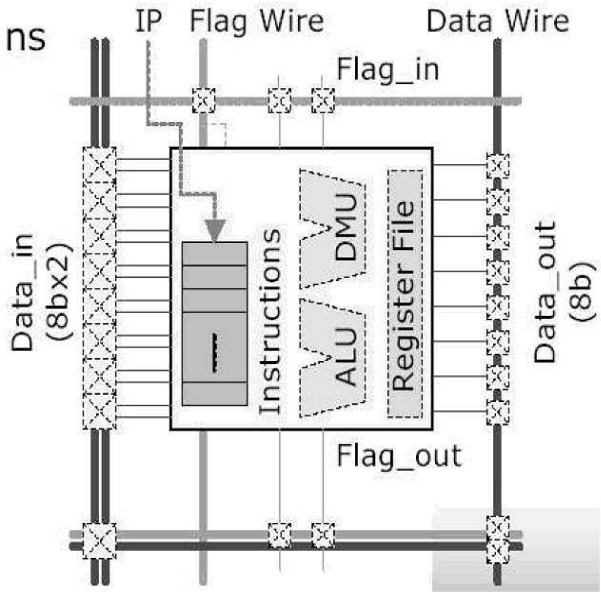

Figure 4: A Processing Element.

Figure 4, taken from [10], shows a block diagram of a PE. The ALU can handle 8-bit arithmetic and logic functions. The Data Management Unit (DMU) handles data manipulation functions like byte select, shift, mask, constant generation, etc. as well as bit manipulation functions. Each PE can store up to 16 instructions in the local instruction store. The specific ALU/DMU operation as well as inter-PE connections are dictated by an instruction. The PE allows the operands to come from either its own register file or from some other PEs (flow-through case), but not directly from memory. Each PE has an *instruction pointer* (IP), provided by the STC, that identifies a datapath plane. Instantaneous dynamic reconfiguration occurs as one sequences through instructions and IP changes. The collection of PE instructions behaves like an extreme VLIW machine.

Software Tools

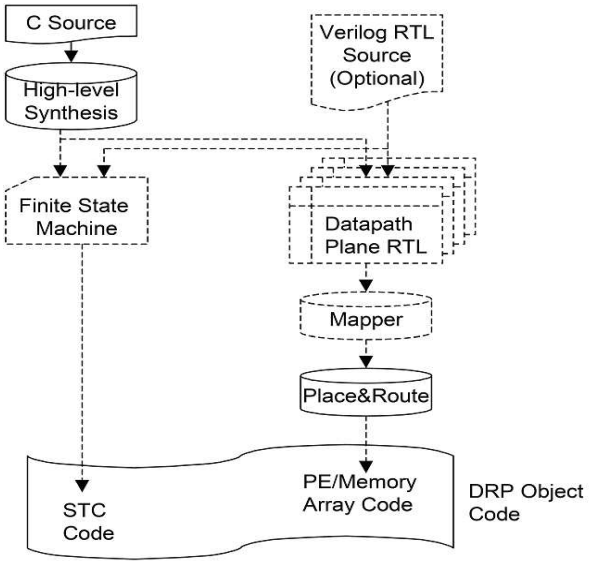

Figure 5: The DRP tool flow.

NEC has paid special attention to creating a good software development platform for the DRP. As shown in Figure 5, taken from [10], the key component of this toolchain is a C compiler. NEC has modified their in-house ASIC high-level synthesis tool called *Cyber* to exploit some of the DRP architectural features. This compiler accepts C source and generates FSM code *and* associated datapath planes. One can optionally input Verilog RTL code directly as well as mix it with the compiler generated RTL. The *mapper* then maps this RTL for each datapath plane to individual PEs and memories. Finally, a place and route tool physically locates the PEs and memories and mutually connects them. The STC control code and the PE/memory array code are linked to form the DRP object code. The NEC toolchain has rich GUI providing both a high-level synthesis view (combining both a scheduled data flow graph for the array code and a scheduled state transition diagram for the FSM code) as well as a place and route view (that can be used for critical path delay analysis).

3.2 IPFlex

IPFlex is a Japanese startup in the reconfigurable computing domain and is very focused on network processing as a target application. IP Flex announced in September 2002 that the working sample chips of their first generation, DAP/DNA-HP (Digital Application Processor based on Distributed Network Architecture) had been already fabricated by Fujitsu (Japan) and available since August 2002.

3.2.1 Architecture

Figure 6. illustrates a block diagram of the DAP/DNA architecture [11]. It is comprised of their proprietary 32-bit RISC CPU core (DAP) and the matrix of 148 processing elements (DNA). They claim the CPU core (DAP) can be replaced with other popular CPU core, such as ARM or MIPS compliant core. Networking processors are supposed to be primary application. The interconnections between elements in DNA are dynamically reconfigured by software and realize pipeline or parallel structure. Each element itself can be reconfigurable among 8 types of arithmetic/logic unit. Those configurations can be changed in 1 clock cycle. DAP/DNA-HP works on 120MHz and power consumption is about 5W.

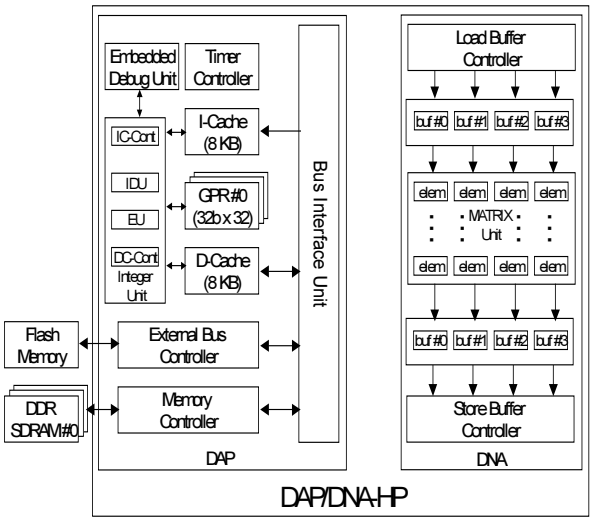

Figure 6: DAP/DNA block diagram.

There are 148 compute elements (denoted as *elem* in Figure 6) in the DNA array. The breakup of the 148 elements is as follows:

- 66 for arithmetic operations
- 44 for control of data delay
- 12 for arithmetic operations with multipliers
- 8 for address calculation for data transfer between buffers within the matrix
- 8 for address calculation for data transfer from external memory to buffers
- 6 for data storage among the matrix (SRAM), and
- 4 for data input.

Figure 7, taken from [12], shows a detailed diagram of such an element. There are eight different types of compute elements. In that sense we can classify the IPFlex DNA as a heterogenous architecture. Each arithmetic element has two inputs and one output. Most ALUs are simple arithmetic and logic functions without any multiplication capability.

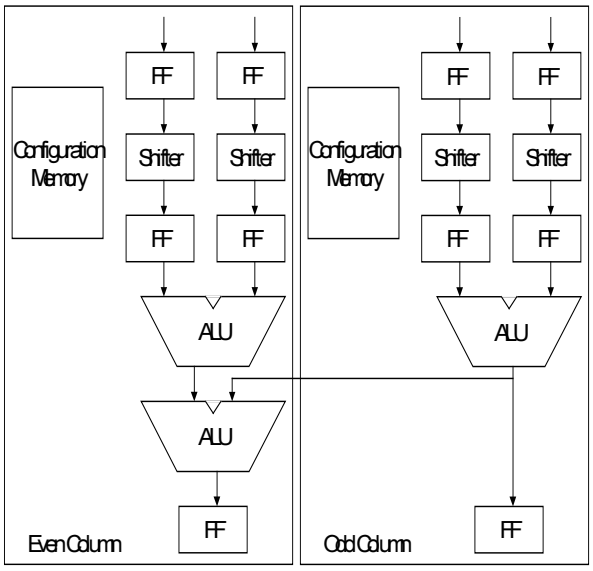

Figure 7. Block diagram of a DNA element.

3.2.2 Software Tools

Development tool chains are created by their own, which includes a compiler, a assembler, a debugger, and a simulator. Figure 8 shows its overview flow. At first algorithm in C language is analyzed on "DNA analyzer" which includes ISS (Instruction Set Simulator) for DAP. Then some portions for DNA are manually extracted and modified as DNA configuration files for input of "DNA Compiler." And "DNA Compiler" converts them to description based on C language. Finally both DAP portion and DNA portion are compiled into object files and uploaded into its hardware or the simulator. It can accept not only C language but also MATLAB input.

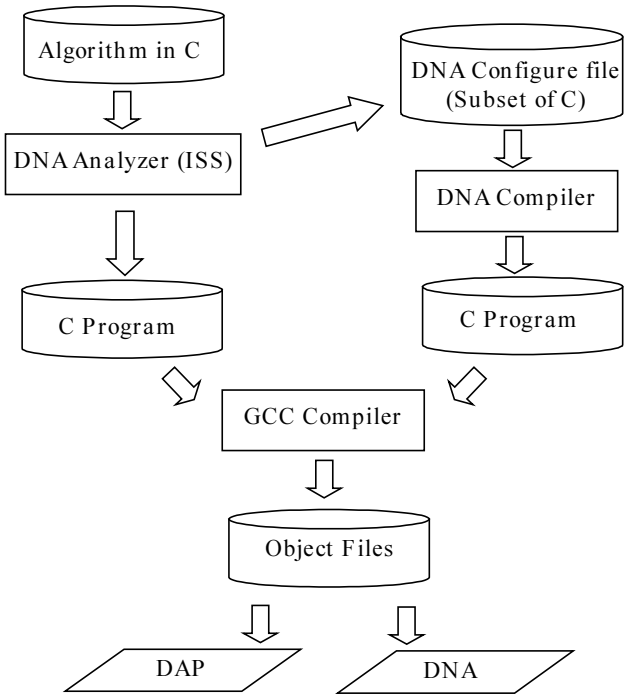

Figure 8: IPFlex software development tool flow.

3.2.3 Applications

PFlex has positioned its DAP/DNA architecture as an alternative to conventional network processors. Like the NEC DRP, IPFlex is also targeting networking applications like packet processing, network security, etc.

	Granularity	Topology	Functionality	µcontr.	Reconfig.	Software Tools	Applications
Berkeley Garp	Fine-grain, 2bit wide	2D mesh, 24x32	4-in LUT, carry chain, multiplexer	MIPS-II plus 20 instrs.	64b per PE, cache, encoded	C entry, `garpcc` configurator, simulator	DES/MD5/SHA, image dithering, median filter
CMU PipeRench	Medium, 8bit wide	Interleaved ring, 16x16	Eight 3-in LUTs, Pass Reg	None	42 config. bits per PE	DIL language and compiler	SAR ATR, FIR, IDEA encryption
Irvine MorPhoSys	Coarse, 16bit wide	2d mesh, full conn. quadrant, 8x8	16bit ALU, 16x12 mult, MAC, absdiff	TinyRISC plus instrs.	16 context planes	Allows C, `tcc` compiler, mLoad, simulator, mView	MPEG video, ATR, IDEA encryption
PACT XPP	Coarse, 32bit wide	Array, 64 per cluster, up to 4 clusters	Four 32bit integer ops, BREG/FREG	MIPS 5Kc	Ring mem., differential reconfig.	C, XPP-VC, NML, mapper, XSIM, GUI	Wireless base station, WLAN OFDM, imaging
Elixent RAP	Fine, 4bit wide	Alternating logic/switch array, 64x64	Simple ALU, 4bit instr., 4-in LUT	None	Dynamic instructions	Allows Matlab & Handel-C, RTL tools, AccelFPGA	Consumer electronics, imaging, comm.
NEC DRP	Medium, 8bit wide	8x8 array per tile, 8 tiles	ALU + DMU, VLIW	Own simple STC	Instr. pointer, dynamic	C entry, compiler, mapper, P&R, GUI	IPsec, IPv4, Packet processing
QuickSilver ACM	Coarse, variable width	Fractal with 4-node clusters, MIN	ALU, Bit manipulation, FSM, Scalar	KARC, MARC	Data & config info mixed	SilverC, compiler, Silverware, mapper	Wireless comm., WLAN, Vocoder
Chameleon RCP	Coarse, 32bit wide	(7+2)per tile, 3 tiles per slice, 4 slices	32bit DPU plus 16x24 multipliers	125 MHz ARC	Two config planes, config stack	C, compiler, eBIOS, fabric function optimize	Wireless base station, cdma2000
MorphICS WSP	Coarse, variable width	Hierarchical, array, slice, bit-slice	RFU with LUT, FFU	External GP CPU	Hierarchicalconfigurable interconnect	C++, compiler, extensible data types, VMI	Multi-channel 2.5G/3G base station
IPFlex DAP/DNA	Coarse, 32bit wide	Array, 148 PEs	8 types of ALU	Own DAP	Not known	C, DNA compiler, analyzer, DAP ISS	Network processing
picoChip	Coarse, 16bit wide	Array, 430 AEs	control, memory, MAC, standard	Control AE	None in particular	Assembler, P&R tool, debugger, no godd C compiler	Wireless comm., now multimedia

Table 1: Comparison of different reconfigurable architectures.

3.3 Other Commercial Work

Other reconfigurable computing efforts in the industry that we have not discussed here are PACT [13], Elixent [14], and QuickSilver [15].

4 Comparative Analysis

Table 1 is a self-explanatory comparison of a number of reconfigurable architectures including the two discussed in this paper, analyzed with seven metrics.

5 Conclusions

The purpose of this paper has been to present the state-of-the-art out there vis-à-vis reconfigurable computing. As one can surmise reading the paper, this field, though not commercially successful yet, is rather mature and varied. After analyzing the reconfigurable computing landscape, we make the some recommendations.

Architecture: One has to consider the following factors before making choices for the reconfigurable architecture:

- Power consumption – most existing reconfigurable systems ignore the important power dissipation issue. Depending on the target application, one has to make architectural choices to reduce power consumption at the outset.

- Reconfiguration overhead – one should avoid the problems associated with FPGAs. In order to be truly dynamically reconfigurable, one should pay special attention to reducing the configuration overhead. This should be a key area of research. Techniques like wave configuration and differential configuration are good starters.

- Topology – The commonly used two-dimensional mesh interconnection is rather constrained for most applications. On the other hand, the fully connected crossbar is prohibitively expensive. One must find a middle ground. Topologies like a mesh of trees or MOTs where one can combine the advantages of meshes and complete binary trees should be explored. The topology choice should consider features like large bisection width, low critical path, good VLSI layout and most importantly good *mapability* (we define mapability as the ease of mapping a wide variety of computational task graphs onto the reconfigurable accelerator topology).

- Number of processor elements and data width – this will be dictated by the die size and power consumption constraints. But keeping the advanced process technology and future competition in mind, the number of processors may have to be in thousands rather than hundreds. Also, our analysis indicates that medium granularity is a prudent choice.

- Functionality – one of the key decisions to be made in terms of the architecture is what kind of functionality to provide within a compute element. Besides basic ALU and bit manipulation capabilities, we think that multiplication (MAC) and LUT capabilities are also necessary for most applications.

- Heterogeneity – we think that instead of having all compute elements the same, it will be a better idea to provide a heterogeneous architecture with the capability of extending via extension processors.

- Microcontroller – this may not be a big issue. Nevertheless, one should carefully analyze the pros and cons of using *decentralized* microcontrollers tightly coupled with the reconfigurable fabric, before deciding whether to use an ARM microprocessor as a central controller or not.

- Staging data in and out – this may turn out to be a bottleneck in most data-bound applications running on the reconfigurable platform. Keeping the massively parallel reconfigurable engine busy with quick data movement will be essential.

- System on a chip (SoC) – we should keep in mind that the reconfigurable accelerator is part of a bigger SoC and not a standalone device. Accordingly, one needs to carefully design the entire system architecture including SoC integration bus (like the AXI or SONICS bus), interface with the microcontroller, caches and local memories, embedded DRAM, DMA, memory controller, and other peripherals.

- Differentiation – Since there are a plethora of reconfigurable architecture companies, the success will depend on clear differentiation. Making an architecture domain-specific rather than general purpose is a good differentiation. Moreover, reducing cost and power consumption will be crucial.

Software Tools: Software tools can make or break a reconfigurable computing machine. In other words, the success of such a system will largely be dictated by the quality of the application development environment. This has always been true, even for conventional computers. But this assumes far greater importance in the case of reconfigurable systems. We feel that the software tools should have the following features:

- First and foremost, one should allow the design entry in a high-level language like C, C++, Java, etc. This necessitates a high quality compiler. Such a compiler faces challenges similar to those faced by compilers of conventional architectures. Additionally, one has to go beyond the traditional compiler optimization techniques, and use state-of-the-art high-level synthesis know-how to create a high quality compiler for reconfigurable accelerators.

- A good mapping software that maps the task to the reconfigurable fabric is essential. This mapper can be integrated with the compiler.

- In addition to HLL design entry, one has to make extra effort to reach the many signal processing researchers to make the reconfigurable accelerator an effective target. To accomplish that, one has to allow design entry in languages like Matlab and Simulink from MathWorks.

- An authentic, fast, cycle-accurate simulator of the reconfigurable accelerator will be crucial for applications development. SystemC may be the language of choice to build such a simulator.

- Similarly, a visualizing debugger is essential for any flexible accelerator.

- Recognizing the fact that eventually we are dealing with a complex SoC that combines a reconfigurable accelerator with a microprocessor core leads us to the path of effective co-simulation and co-design. Integrating with

existing tools like AXYS MaxSim, Cadence VCC, CoWare N2C, Mentor Graphics Seamless, and other similar commercial products will be necessary.

Applications: The reconfigurable computing concept is domain-specific, not for general-purpose applications. Therefore, right at the outset, one should know the target application(s). There is a wide range of current and future applications that can benefit from the acceleration provided by reconfigurable computing. These applications share the requirement for lots of numeric computations as well as the presence of inherent massive parallelism. In our estimation, the following applications are some that seem promising:

- Software Radios for Mobile Terminals, wireless infrastructure, as well as in future automobiles.
- Multi-format multimedia processing
- Image post processing and enhancement in DTV and STBs
- Mobile applications for 2.5G/3G wireless in smartphones and PDAs
- Human Computer Interface including image processing, speech recognition, and gesture recognition.

References

[1] G. Estrin, et al "Parallel processing in a restructarable computer system," IEEE Transactions on Computers, pp. 747-755, 1963.

[2] H. Schmit, et al, "PipeRench: A virtualized programmable datapath in 0.18m technology," IEEE Custom Integrated Circuits Conference, 2002.

[3] M. Budiu and S. C. Goldstein, "Fast compilation for pipelined reconfigurable fabrics," ACM SIGDA 7th International Symposium on FPGAs, February 1999.

[4] H. Singh, et al, "MorPhoSys: An integrated reconfigurable system for data-parallel computation-intensive applications," submitted to IEEE Transactions on Computers, 2000.

[5] J. Hauser, "Augmenting a microprocessor with reconfigurable hardware," Ph.D. Dissertation, University of California, Berkeley, December 2000.

[6] H. Zhang, et al, "A 1V heterogenous reconfigurable processor IC for baseband wireless applications," ISSCC, pp. 68-69, February 2000.

[7] S. Hauck, et al, "The Chimaera reconfigurable functional unit," FCCM, pp. 87-96, April 1997.

[8] E. Mirsky and A. DeHon, "MATRIX: A reconfigurable computing architecture with configurable instruction distribution and deployable resources," FCCM, 1996.

[9] C. Ebeling, et al, "RaPiD: Reconfigurable Pipelined Datapath," 6th Annual Workshop on Field-Programmable Logic and Applications, 1996.

[10] M. Motomura, "A dynamically reconfigurable processor architecture," Microprocessor Forum, October 2002.

[11] Nikkei Electronics Online article, http://ne.nikkeibp.co.jp/DSP/2002/09/1000014482.html, September 2002.

[12] Nikkei Electronics, "Idea of software generated virtual circuits," (in Japanese) pp. 59-63, November 18, 2002.

[13] P. Glaskowsky, "PACT debuts extreme processor," Microprocessor Report, October 2000.

[14] Elixent Corporation, "Changing the electronic landscape – the reconfigurable algorithm processor," Technical White Paper.

[15] P. Master, "A look into QuickSilver's ACM architecture," EETimes Supplement called The Art of Change: Technologies for Designing our Future*, pp. 124-125, September 2002.*