Configurable Computing: Practical Use of Field Programmable Gate Arrays

By
Craig D. Ulmer

School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332-0250

January 5, 1999

Submitted to the Qualifying Examination Committee
In Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy in Electrical Engineering

Qualifying Exam Committee:

Dr. Ian Akyildiz, Chair
Dr. Vijay Madisetti
Dr. David Schimmel
Dr. Sudhakar Yalamanchili, Advisor
Dr. Ellen Zegura

Contents

1. Introduction

Since early conceptualizations of programmable logic, researchers have envisioned a prominent computational role for configurable hardware. In as early as 1960, Estrin proposed supplementing general-purpose CPUs with specialized hardware units that could be configured to emulate logic functions [1,2]. Unfortunately, technology for such programmable logic was insufficient until the invention of the Field Programmable Gate Array (FPGA) in 1985. Initially these devices were limited by a low number of logic gates the array could emulate. To compensate for this deficiency, researchers turned to building large multi-FPGA based systems known as Configurable Computing Machines (CCMs) [3,4]. While working with these machines could be a complicated process, early CCM teams reported significant speedups for their hardware-assisted programs. These systems are interesting because the reconfigurable nature of the FPGAs allows the CCM hardware to be reused as needed by different algorithms. However, the high cost and large size of these machines prohibits them from general use.

Recent advances in FPGA technology provide an opportunity to bring the high performance of CCMs to low cost, general use systems. The two dominant advances making this possible are improvements in FPGA gate density and speed, and the commercial availability of FPGA boards for easy system integration. These boards are a cost-effective means of configurable computing, allowing designs to leverage a middle ground between software-only and dedicated ASIC hardware approaches. Additionally, FPGA-assisted designs may use reconfigurable hardware techniques to enhance performance in ways that are not possible in any other technology.

While configurable computing offers a powerful method of high-performance calculation, the field is relatively new and lacking guarantees of speedup. In order for this technology to be effective, designers must be aware of the underlying hardware features as well as evolving design methodologies. The intent of this report is to summarize work in the configurable computing domain and expose key points of high-performance FPGA-assisted design and use. A primary concern is practicality, with the expectation that configurable hardware should be commercially available and of low cost. This report is organized into seven sections addressing specific topics of FPGA-assisted design. The first three sections describe the potential

use and architecture of FPGAs, specifically highlighting applications that are well suited to the technology. The next two sections provide details of system design and comparison. These details are followed by an examination of three successful applications. Finally, a discussion of obstacles in the technology is provided with suggestions for future enhancement.

2. Configurable Computing

Complex algorithms may be implemented in software, hardware, or a combination of both. Software approaches use general-purpose CPUs, sequencing discrete CPU operations such as multiply or add to realize an algorithm's functionality. While these devices are highly programmable, the overhead for decoding and executing instructions detracts from computational performance. At the other extreme of algorithmic implementation is custom hardware known as ASICs. While these circuits provide optimal computational performance for a given application, the chip cannot be adjusted after fabrication and is therefore only suitable for a single application. An example between the differences in hardware and software implementations can be found with the evaluation of the logic function $F = 4a^2 + 3b$. Figure 1a illustrates a software program for this function, sequencing several basic operations through the processor until the function is evaluated. Figure 1b shows the same operation for a hardware-based design, with dedicated computation units providing an answer in a single iteration [6].

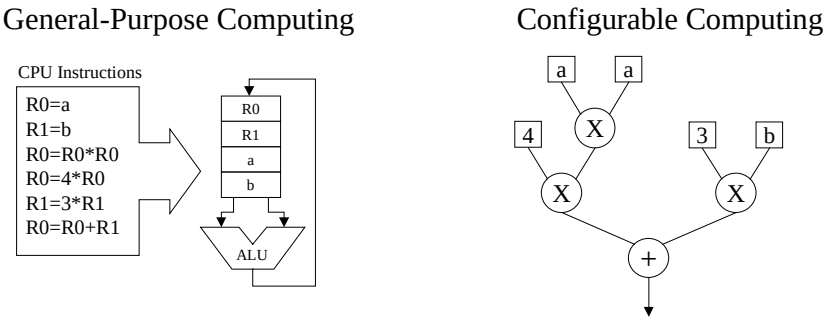


Figure 1: Comparison of general-purpose and configurable computing for $F = 4a^2 + 3b$

Configurable computing is a compromise between hardware and software. Field Programmable Gate Arrays facilitate this tradeoff, allowing hardware configurations or images to be loaded into the device. Once configured, FPGAs behave as though they are custom VLSI

circuits defined by their configuration image. While approximately three times slower than ASICs, performance improvements of FPGAs over CPUs can be significant. In [5,6], Dehon provides both analytical and empirical comparisons of configurable and general-purpose computing. DeHon's analysis shows that FPGAs offers a much higher computational density per unit area compared to general-purpose CPUs. The benefit of this computational substrate is that the hardware may be reused by multiple applications. In this sense the FPGA can ideally provide a "virtual ASIC" for any program that can benefit from hardware-assistance.

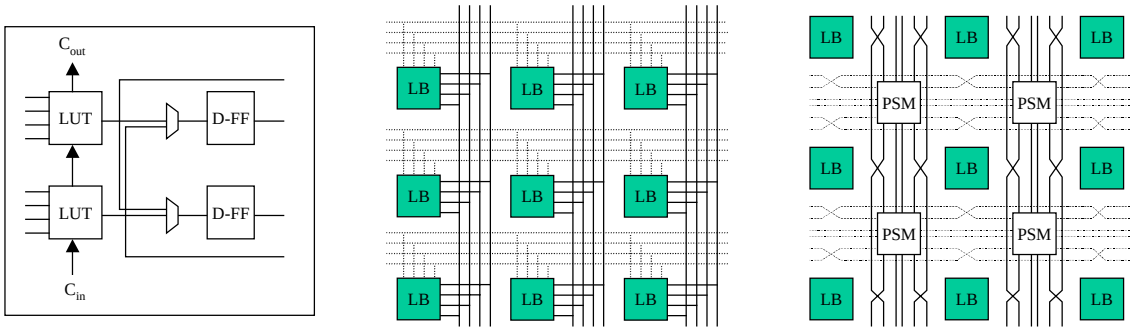
3. FPGA Architectures

Given that configurable computing may be beneficial to modern processing, it is necessary to examine the differences between the theory and the physical implementations. FPGA architectures represent the raw building blocks for which configurable computing works with, and therefore the characteristics of the device must be understood before the benefits can be applied to a system.

3.1 Current Generation Architectures

3.1.1 General Description

Current generation FPGA architectures generally consist of three main components: logic blocks, I/O interface blocks, and a programmable interconnection network. Of these components, the design and relationship between the logic blocks and the interconnection network best characterize an FPGA. I/O interface blocks are peripheral logic built to interface the chip to external circuitry, and therefore are not examined in this report. Figure 2 shows a generalized view of the components found in FPGAs [7].



(a) Logic Block (b) Bus Interconnect (c) Switch Interconnect

Figure 2: Basic FPGA Components

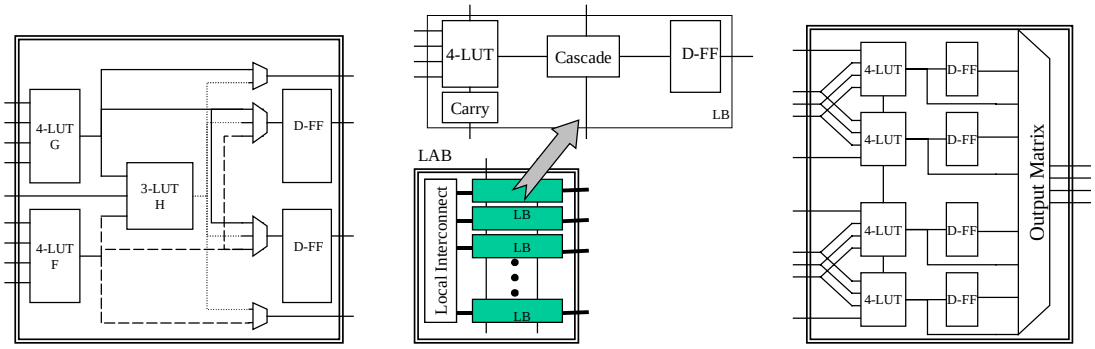
The core building block of an FPGA is the programmable logic block (LB). This block implements the actual logic functions for configurable computing, and a general representation is given in figure 2a. In this model, the logic block uses three stages: function generation, internal routing, and memory. The logic block is loaded at configuration time with information that determines how each of these stages is to behave. The function generation stage is implemented with an array of programmable lookup tables (LUTs). LUTs typically have between three and five inputs, with four inputs frequently being cited as the best compromise between LUT access time and the average desired function delay [8]. The programmable routing stage follows the function generators and allows function results to be supplied to the logic block's final stage with minimal delays. Finally, the logic block uses D-flip-flops to implement bit storage within the logic block. The flip-flops are beneficial for distributed memory in configured designs, including their use as registers between pipeline stages.

Logic blocks in FPGAs are flexible enough to implement at least three distinct modes of operation: combinational logic, arithmetic or ripple modes, and dedicated memory storage. In the combinational logic mode, the LUTs are loaded at configuration time with truth tables necessary to implement a logic function that is dependent on LUT inputs. Early FPGAs with this mode were found to be insufficient for complex operations such as adders, comparators, and multipliers [9]. The arithmetic or ripple mode was thus added to the logic block, using dedicated carry gates to rapidly propagate signals from a logic block to its neighbors. The final mode of memory storage allows the LUTs to be configured as RAM or ROM units. This mode increases the amount of usable internal memory within the device and is particularly useful for embedding memory elements throughout a design.

Interconnection of logic blocks in FPGAs consists of two approaches: bus-based or switch-based. In either case interconnection wiring routes horizontally and vertically, with logic blocks attaching to routing resources as programmed at configuration time. Bus-based routing is the simpler of the two schemes, providing all logic blocks in a row or column equal access to a horizontal or vertical routing resource as shown in figure 2b. As logic block array dimensions increase, the number of logic block taps connected to the wire greatly affects the line's parasitics and thus the wire's speed. To combat this hazard, FPGA architects turn to switched interconnection as illustrated in figure 2c. In these networks, buses are limited to specific distances, typically of length one, two, four, and global. Programmable switch matrices (PSMs) distributed throughout the FPGA allow routes to be established as the combination of multiple length wires. While the switch matrices induce their own delays, the number of taps on a particular wire segment is greatly reduced, compared to bus-based designs [10]. However, this isolation of parasitics comes at a cost of increased complexity for software tools responsible for the placement and routing of logic in the FPGA [11].

3.1.2 Commercial FPGAs: XC4000, FLEX, and ORCA 2

The majority of published work on FPGAs centers around the Xilinx XC4000 series architecture [10]. This family employs large, complex logic blocks combined with a switch based interconnection network. The logic block of the XC4000, as seen in figure 3a, is unique in that it can be configured to use either one or two stages of LUTs per logic block: a 3-input LUT follows the initial twin 4-input LUTs. This arrangement indicates that Xilinx intends for logic blocks to implement complex logic functions, reducing the amount of traffic using the interconnection network. The fast carry-ripple logic between adjacent logic blocks allows for one fast 2-bit full adder per logic block. Non-carry signals route through a programmable switch matrix shown in figure 2c, with single, double, quadruple, or global distance wire lengths.



(a) Xilinx XC4000 CLB (b) Altera FLEX (c) Lucent Orca-2

Figure 3: Simplified Views of Commercial FPGA Logic Blocks

Altera holds the largest share of the programmable logic device market [16] and offers a competitive alternative to the XC4000 architecture. The FLEX family [11] of figure 3b implements a simple logic block with only a 4-input LUT, cascade logic, and a D-flip-flop, but then stacks several logic blocks together to form a Logic Array Block (LAB). Logic blocks within the LAB are tightly coupled with both dedicated routing and fast carry-ripple connections. LABs communicate with other LABs through a bus-based network. This communication strategy works well in the FLEX architecture because local traffic is kept within a LAB, leaving long haul communication to the bus based communication network.

The Lucent Orca 2 FPGA [12] architecture's logic block represents a clever tradeoff between the complexity of the XC4000 and the simplicity of the FLEX. As depicted in figure 3c, the Orca 2 architecture use four 4-input LUTs that may be configured to act as either quad 4-input LUTs, twin 5-input LUTs, or as a single 6-input LUT. While this configuration places some restrictions on the quad 4-input LUT mode, it allows for a high component utilization within each logic block [20]. The Orca 2 implements fast carry-ripple logic, providing a 4-bit full adder per logic block. While the Orca 2 uses bus-based communication, it targets 4-bit data widths, implying high system-level functionality per logic block.

3.2 Emerging Architectures

The first implementations of these current generation architectures were fabricated as early as 1994. The FPGA industry is currently on the verge of an architecture family transition, with all three major FPGA companies announcing the release of their next generation architecture chips. This section briefly describes features found in commercial chips released after 1994.

The most influential commercial architecture since the release of the current generation of FPGAs is undoubtedly the Xilinx XC6200 RPU [19]. Released in 1995 as an experimental chip, the XC6200 addressed many of the requests from the configurable computing community. The primary benefit of this device is that it allows users to configure portions of the device at a time rather than forcing an entire unit reconfiguration. Partial reconfiguration is essential to efficient configurable computing since it allows better real-time interaction with the device than in previous architectures. A direct effect of the partial reconfiguration architecture in the XC6200 is an increased amount of routing structure. This routing structure is collectively known as the FastMap interface and allows an external processor to directly read or write any register or logic block in the device. The FastMap communication network reduces the complexity of moving data into or out of the design and results in an overall tighter coupling with the host processor.

The upcoming generation of FPGAs expands upon the previous generation's architectures, and exhibits gains in foundry technology. While none of the upcoming architectures are as dramatic as that of the XC6200, they all provide significant architecture enhancements. These features include an increased logic block complexity, higher gate densities, improved high-speed clock handling, dedicated multiply propagation support, and integration of dedicated RAM into the chip. In addition to these properties, Xilinx's Virtex [13] supports a fast partial reconfiguration mode similar to the XC6200. Altera's APEX [14] architecture offers a special Content Addressable Memory (CAM) [17,18] mode for its RAM. Lucent's Orca 3 [15] technology focuses on a complex logic block, designed to implement higher level units without causing low gate utilization for simple logic functions. While the preliminary datasheets for these architectures show an increased awareness of system level functionality in course-grain FPGAs, there is little experimental data available to impartially compare the architectures. Therefore, this upcoming generation of FPGAs is not the dominant focus in this report.

3.3 Consequences of FPGA Architectures

A common misconception about FPGAs is that configuration images are implemented as a 'sea-of-gates'. An examination of the underlying hardware reveals that this conceptualization is not the case. While FPGAs do serve as a regularly arranged gate array, there are architectural factors that clearly separate FPGAs from custom 'sea-of-gates' designed VLSI hardware [21]. First, FPGAs implement functional logic with n-input lookup tables. Because of this

implementation, the delay of a 1-bit NOT function is the same as the delay for an n-input complex logic function. Therefore, the benefits of traditional VLSI logic equation reduction may be lost to the granularity of the FPGA's lookup tables. Second, the interconnection networks in FPGAs induce costly delays for routing between logic blocks. This results in both nonlinear delays between wired elements and a contradiction of the sea-of-gates assumption that wiring nearly "comes for free." Finally, architectural enhancements such as fast carry-ripple logic generally outperform complex gate arrangements that inevitably must be mapped to LUTs. These features make it difficult to assume that FPGA designs are best served with traditional VLSI approaches.

One of the best examples of the nonlinearities involved in FPGA-based designs is found in Xing and Yu's analysis of adder implementations for the XC4000 [21]. This study compares several binary integer addition techniques for a wide range of data widths. In particular the authors examine carry-skip, carry-select, carry-look-ahead, and the XC4000 native carry-ripple style adders. It should be noted that the XC4000 design specification [10] warns that "the dedicated carry circuitry is so fast and efficient that conventional speed-up methods like carry generate/propagate are meaningless at the 16-bit level." The result of Xing and Yu's study confirms this warning: the most efficient adder in both speed and size for up to 48 bits is the native carry-ripple adder. This conclusion yields two important insights about FPGAs. First, VLSI architectures may not translate to efficient FPGA implementations. Second, minor enhancements to an FPGA's logic block can provide significant performance and density improvement for designs utilizing such features.

Another common building block in configurable computing that is dependent on FPGA architecture is integer multiplication. While the current generation of FPGA architectures do not provide direct support for hardware multiplication, it is possible to make use of the fast carry-ripple logic for speed improvements. Peterson and Hutchings present a comparison of multiplication strategies for different FPGA architectures in [22]. This study examines bit-serial, parallel-array, and constant parallel-array multipliers for the FLEX, XC4000, and CLAy FPGA families. While occupying more chip resources, the parallel designs provide a 2-3 factor of improvement over bit-serial or iterative implementations. Typical speeds for an 8-bit multiplication range from 4-5MHz for bit-serial to 8-15MHz for parallel array designs. Efficient

carry-ripple addition is specifically cited as a doubling of performance in their final implementations. Do et al. extend this work in [23] by pipelining multipliers to achieve high peak performance. While a single multiplication is achieved at a rate of roughly 5MHz, the overall 15-stage pipeline supplies results at 75MHz. This approach exemplifies how data streaming greatly improves the practicality of an FPGA, as well as how designs may capitalize on an FPGAs bulk resource size to compensate for a lack of specific functional support.

Floating point operation on FPGAs is a continuing problem for which there is no immediate architectural solution. The main problem with floating point arithmetic in FPGAs is that there is no direct hardware support for such operations. As a result, floating point units in configurable computing must be constructed from other building blocks such as integer multipliers and adders. While building floating point designs for FPGAs is certainly possible [9, 25], the implementations exhibit a high resource cost. Ligon et al. [24] present multiple styles of floating point units to examine the costs in terms of area and speed. While their final pipelined design produces a 40 MFLOPS floating point adder, it occupies 40% of the total logic block resources for an XC4020. Interestingly, the authors observe that pipelining the design accounted for only a small increase in resource cost, with only a slight growth from the 36% logic block utilization of the iterative approach. However, the authors conclude that the performance of the FPGA in floating point arithmetic is far worse than commercial processors. Additionally, the high resource cost for implementing these designs makes FPGA-based floating-point operations impractical. Based on the preliminary data sheets for the upcoming generation of FPGAs, it is expected that floating point operation in FPGAs will not be viable for some time.

4. Strengths and Weaknesses of FPGA-assisted Designs

For a fair evaluation of the role of FPGAs in practical computational scenarios, the strengths and weaknesses of both the FPGAs and the manner in which they are utilized in a system must be considered.

4.1 Strengths

System level benefits of FPGAs are largely captured in the reasoning behind configurable logic, as described in section 2. However, designers must achieve high performance at the FPGA level before the advantages of configurable computing can be utilized. Noting that gates emulated

by FPGAs are slower than custom VLSI circuits, designers must use latency hiding techniques to achieve high performance. Fortunately, the high densities of FPGAs make these techniques possible, as well as competitive with dedicated hardware. There are three dominant strengths that exploit the FPGAs characteristics: pipelining, parallelism, and partial evaluation

Pipelining is the hardware technique of segmenting a complex operation into distinct stages so that multiple data values are computationally in-flight at the same time. Pipelining is therefore a natural choice for FPGAs due to the discrete and regular qualities of FPGA logic blocks. Memory found at the tail end of the logic blocks completes this image, utilizing the storage as pipeline stage registers. Several groups [22-24] observe that pipelining an iterative design in FPGAs generally comes with only minimal resource costs. As a result, pipelines allow FPGAs to transform mediocre iterative designs into high-throughput realizations competitive with alternate hardware. However, the most interesting feature for FPGA pipelining is the ability to combine complex operations into a single deep pipeline. This approach is similar to systolic or bit-serial strategies: data flows out of one pipeline into the next, without stalling for the collection of an entire data value. While DSP architectures exhibit some aggregate operations such as the Multiply-Accumulate (MAC), FPGAs have the ability to chain together any sequence of operations as needed by an algorithm. These deep pipelines are difficult to abstract for general-purpose CPUs, and therefore FPGAs offer computational potential found elsewhere only in ASICs.

The second strength of FPGA design is the ability to implement a large degree of computational parallelism. The number of independent computations that may be implemented in an FPGA is limited only by the size of the chip and the ability to find such computations. Two types of parallel exploitations are common: data parallelism and algorithmic parallelism. Data parallelism occurs when regular processing may be performed over a large data set concurrently, such as in image processing. Algorithmic or control parallelism is the act of allowing multiple independent algorithmic tasks to operate concurrently. An example of algorithmic parallelism is found in a network interface chip: send and receive threads are independent tasks and therefore may be implemented as concurrent state machines in an FPGA. Finally, a combination of data and algorithmic parallelism can yield high throughput devices with low speed parts as depicted in figure 4. This hardware technique streams high-speed data values into and out of an array of low

speed computational units. This method demonstrates how an FPGA can compete with other dedicated hardware devices by trading device area for speed.

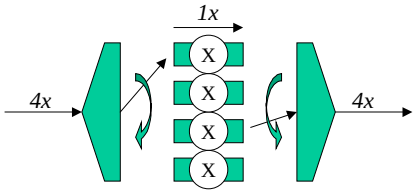


Figure 4: High Performance with Low Speed Components

The third primary strength of FPGAs is the ability to use partial evaluation techniques to minimize computation logic and delay. Partial evaluation is a technique used by compilers and FPGAs to reduce a multivariable function to a less complicated expression based on information known at compile time. Consider for example a multiplication unit. A general-purpose multiplier must logically produce answers for all possible sets of inputs. However, if one of the multiplier's inputs is a constant then the unit produces only multiples of that constant. Using this information, all paths leading to non-multiples of the constant may be eliminated and the logic equations for the multiplier thus reduced. This technique is not feasible in general-purpose CPUs, but the reconfigurable nature of the FPGA makes such optimizations possible. In Peterson and Hutchings' study of multiplication implementations for FPGAs [22], partially evaluated circuits for constant multiplication are considered, and result in a factor of 2-3 improvement over the best case general-form multiplier.

A key dependency of the above strengths is the ability for an FPGA to be configured to perform as hardware relevant to a given application. The middle ground offered by configurable computing between general-purpose CPUs and dedicated ASICs represents an emerging design style dealing with "disposable hardware."[1,2,5,6] In the simplest form, common applications may utilize an FPGA as if the application is worthy of its own dedicated ASIC. A more interesting application of configuration arises in the field of adaptive hardware. Researchers such as Mangione-Smith see a great opportunity for the configurable nature of FPGAs not only to provide high-speed computation, but also to give a means for reacting to a problem's computational progress [1]. An adaptive system potentially would provide multiple hardware images optimized for specific cases of a given problem. System hardware would thus be

responsible for swapping in different hardware configurations as a problem's nature changes. Applications such as target recognition currently implement crude versions of this strategy [1, 38, 39]. In [26], Rashid sees a more featured system, capable of generating its own hardware images dynamically. Configuration manipulations therefore present a computational opportunity for FPGAs unavailable in any other technology.

4.2 Weaknesses

The physical constraints of the FPGAs are a primary source of weakness for FPGA-assisted calculations. In addition to the computational building block limitations described section 3.3, designs implemented in configurable logic are subject to factor of 3 slowdown in speed and a factor of 10 degradation in density when compared to similar logic implementation in ASICs [8]. This degradation is a result of gates and interconnection being implemented through configurable SRAM-based devices. Gate density is perhaps the most critical limitation in FPGA devices since the size of the gate array determines how much logic can be implemented, and therefore restricts the degree of achievable parallelism. A number of works specifically note the inability of FPGAs to store large designs as a significant performance limit [9, 23, 24]. Realizing the importance of this problem, the FPGA industry is continuously increasing gate capacity with each chip revision. Along similar lines, it is well known that the limited routing resources of interconnection networks prevent 100% utilization of an FPGA's logic blocks. As a consequence the FPGA industry is continually exploring the routing algorithms for synthesis tools as well as offering additional routing resources for high-density devices.

Beyond the physical constraints of the FPGA architecture, there are a number of barriers in current systems that prevent efficient use of FPGA-assisted processing. Without question, the ultimate weakness of current systems is the lack of proximity of the FPGA to data. A number of FPGA-based computation boards are available, but these boards all reside at the end of the PCI bus or serial port. The computational flow in this arrangement is a costly path: computation starts at the host CPU, migrates with data through the memory system, and finally moves through the I/O bus into the FPGA card. This process is reversed once the FPGA finishes computation and needs to move results back to the host CPU. Clearly these overheads make the practicality of assisted computation questionable. Alternate proposals to create a more efficient computational environment are discussed in section 8.

Another challenge for FPGAs is the relatively long delay required for re-programming the device. For the current generation chips entire configurations must be loaded into the device at a time, creating an offline time on the order of a few milliseconds. This dead time can be significant for applications where the cost of reconfiguring the device is comparable to the amount of time to perform the operation. Therefore, various research groups propose operating system management of the FPGA to perform better scheduling of FPGA utilizing applications [27]. This work creates a "Virtual FPGA" and leverages existing OS research in managing slow multi-user devices such as memory or disk. A number of hardware architectures to reduce configuration time have been proposed and implemented, such as the XC6200 partially reconfigurable RPU [19] and the Sanders corporation's CSRC multi-context FPGA [48]. These are discussed in section 8.

Designing configurations that efficiently utilize the FPGA can be accompanied by a complicated development cost. Clearly, development for hardware-assisted devices is more costly than software-only approaches due to the complexity of hardware-software co-design. Design environments such as hardware description languages (HDLs) and hardware compilers create abstractions of the hardware to provide simpler design flows and portability between target architectures. Unfortunately a target architecture's strengths may be masked by these abstractions. The balance between high performance and ease of design for configurable computing is a complicated issue, and is further discussed in section 5.

Strengths	Weaknesses
Extreme parallelism potentials	Poor floating point performance
Deep customized pipelining	Limited resources
Partially evaluated circuits	Poor data proximity
Disposable circuits	Overall design complexity

Table 1: Summary of FPGA Strengths and Weaknesses

4.3 Suitable Applications

The strengths and weaknesses for FPGA-assisted computation are summarized in table 1. From these points it is apparent that applications with the following qualities are more suitable for FPGA assistance:

- **Highly parallel applications**: The FPGAs capacity for parallel hardware is a great strength. Therefore the most successful applications are the ones with large degrees of data or algorithmic parallelism.
- **Streamlined Data:** To overcome the burdens of slow internal units, FPGAs often must pipeline data to achieve satisfactory speeds. Applications performing regular operations on a stream of data are a good fit for FPGA use.
- **Prior knowledge circuits**: Algorithms with constant data values or algorithm computation reductions perform well in configurable devices.
- **Complex Custom Logic**: Applications with regular logic operations generally transfer well to state machines in FPGAs. FPGA based evaluation of these operations is beneficial since the FPGA can provide better custom logic implementations than a CPU, and can perform these operations at the same time as other algorithm functions.

4.4 Unsuitable Applications

Conversely, the following forms of applications typically have poor implementations in FPGA-based configurable computing:

- **Sequential programs:** Programs with tight loops are difficult to parallelize for any target architecture. Therefore, general-purpose CPUs perform significantly better than FPGAs in sequential applications.
- **Large floating point calculations:** The high resource cost for FPGA based floating point computation makes applications with floating point operations difficult to justify in FPGA consideration.
- **Non-localizable data:** FPGA performance is significantly reduced when data must flow up and down the memory subsystem. Applications that frequently pass large data sets between the host and FPGA are challenging in current FPGA-assisted systems.

5. Design Methodologies

One of the dominant factors inhibiting the widespread use of configurable computing comes from the complexity of hardware-software co-design. The process of analyzing an algorithm and developing an FPGA-assisted design is substantially more complicated than software-only approaches, and has no guarantee of performance enhancement [28]. Software

engineering practices such as code reuse, iterative design, and automated tool assistance are important for successful design, and captured in the field of rapid prototyping. Due to the depth of the rapid prototyping domain, this report is limited to two primary design methodologies: tool-assisted manual design and custom compilation.

The majority of configurable computing is currently implemented by custom manual design. This technique centers on an engineer examining an algorithm, determining the most complicated operations that could benefit from hardware, and hand designing an FPGA image to execute the computations. While this is admittedly a very slow and complicated design procedure, it often produces the highest performance results, which in the end may be reused by other designs. Typical hardware/software co-design practices use hardware description languages (HDLs) such as VHDL and Verilog to write a software simulation of the device. The HDL source code is then run through complex synthesis tools to generate gate-level descriptions for the circuits. After running the gate-level descriptions through an FPGA specific place and route tool, designs are ready for use in FPGAs. The final step involves the interfacing of host level programs to the loaded FPGA [28, 41]. With hardware and software tasks designed and built, the overall process iterates until a design is fully tested and meets timing specifications.

While the manual design methodology presents high-performance custom hardware, it suffers from a high overhead of analysis, device design, and system testing. A number of research efforts [29,30] realize that the complexity of this design process makes it impractical for general use. These groups propose an alternative approach of placing the burden of analysis and design on the compiler. Specifically these groups use an analysis tool that takes a given C program and compiles it into a hardware gate description. This work capitalizes on compiler technology such as loop unrolling and maps data operations into predefined logic block macros. While appealing in the sense of automated hardware generation, these approaches suffer from the problems inherent in all parallelizing compilers: C is a sequential language and as such it is difficult to extract high-performance parallelism. Typical speedup for these programs in FPGA implementations is reported at roughly 2x [2].

6. Custom Computing Performance Measurements

Evaluating the performance of FPGA-based designs can be difficult. To better understand a design's weaknesses, performance is ideally measured at both the FPGA and system levels. In terms of actual FPGA performance, it is desirable to apply the design to multiple FPGA target architectures as illustrated in [22]. To facilitate a more generic comparison of FPGA architectures, the MIT Reconfigurable Architecture Workstation (RAW) project provides a benchmarking suite of traditional FPGA applications in [31]. However, the authors of [2] suggest that generic benchmarking suites for FPGAs expose little practical information. Since these suites use generic designs, special features of individual FPGA architectures cannot be leveraged in the tests. With the assumption that performance of FPGA designs overrides ease of implementation, the best method of FPGA level comparison is with designs hand crafted to take advantage of a target architecture's features. Therefore, benchmark comparisons can only give approximations of the hardware's performance.

The ultimate performance evaluation of an FPGA-based design is in system level speedup over software-only approaches. Should the overall system speedup fail to be substantial enough to justify the added system complexity, then clearly the FPGA-assisted design should be avoided. Additional factors may play a role in the assessment of an FPGA-assisted design's worth. First, a design with only a low performance enhancement may be more meaningful if the application is frequently used. Second, a design's hardware resource requirements such as a gate count or additional memory dictate the dimensions and expense for the FPGA board's hardware. FPGA literature can be misleading in that hardware requirements necessary for peak performance are often downplayed. Finally the side effects of FPGA hardware on the overall system must be considered in a design's evaluation. For example, while an FPGA board may offload computation from the CPU, it may also increase bus traffic in the region where the FPGA resides.

7. Design Examples

7.1 Data Streaming and Partial Evaluation: Cryptography

Data encryption and decryption is an application for which FPGAs have recently received a large amount of public attention. Recent estimates predict that previously secure cryptography

may be vulnerable to a dedicated network of FPGAs or custom ASICs. With dedicated hardware, brute force key search attacks on encrypted messages may crack messages in time periods of weeks or months instead of years or centuries. The strength of encryption generally relies on two algorithmic details. First the keyspace from which a particular key is chosen to encrypt a message must be large enough that incrementally guessing all possible keys takes a considerable length of time. Second, the encryption process must be complicated enough that it cannot be trivially performed by general-purpose CPUs. The combination of these two factors leads to lengthy searches in the case of brute force attacks. Unfortunately this complexity also limits the speed at which data streams may be encrypted. Therefore, the interest in FPGAs or ASICs is to improve the speed in which encryption and decryption of a data stream can occur.

The encryption and decryption processes are typically not considered as targets for common use of FPGAs because commercially available chips can rapidly perform the calculation. However, in [32] Leonard and Mangione-Smith present a stream oriented cryptography implementation offering improvements in the raw encoding/decoding speeds for the DES algorithm. The basis for this work is the assumption that a session key is used to encode/decode data changes infrequently for a given data stream, and is therefore ripe for partial evaluation techniques in an FPGA. The authors therefore build and compare an FPGA circuit with particular session keys hard-wired into the design. This key-specific circuit is able to achieve a 45% reduction in FPGA logic blocks and a 35% improvement in bandwidth. This work is important because it demonstrates how partial evaluation can provide substantial speed improvements for complex operations that are generally not possible in other forms of CPU or ASIC implementation.

While speed benefits for stream-based encryption/decryption are important, the primary interest in FPGAs for cryptography is high-speed key breaking. Due to the complexity of the encryption process, hardware support to pipeline the operation and minimize key evaluation times is critical. An ASIC implementation of a key breaking circuit would be ideal, but is of little common use to non-cryptographic breaking agencies. Therefore the FPGA is the architecture of choice for ad-hoc evaluation of cryptographic strength. A general key breaking system consists of three main components: a key generator, an encryption unit, and a comparison unit. Although a key generator can be a simple counter, alternate approaches choose less complex but unique key

generators for high speed [33]. The encryption unit may be pipelined depending on the complexity of the algorithm, and generally provides the critical delay for the system. Performance of key breaking circuits for FPGAs is impressive: studies show that a single DES key-breaking circuit can test 1.02Mkeys per second with a system clock of 17MHz [34]. By comparison, software implementations can check at least 50Kkeys per second. Accepting the parallel nature of the operation, multiple key breaking circuits may be placed in a single FPGA, as well as in multiple FPGA based systems.

7.2 Data Parallelism: Digital Signal Processing

FPGAs are a natural choice for digital signal processing (DSP) due to the large degree of parallelism commonly found in this domain's algorithms. However it is not necessarily true that all DSP operations are well suited for FPGAs. For example, the fast Fourier transform (FFT) algorithm is a common DSP building block employing a regular and parallel computation structure. Unfortunately the FFT requires complex value multiplications that are best served with floating point precision. FPGA implementations generally resort to iterative approaches (reducing parallelism), fixed-point precision (reducing resolution), or require large multi-FPGA systems (reducing practicality)[22,35]. It is therefore difficult for low-cost FPGA systems to be competitive with specialized DSP processors in FFT computation. There are many other DSP applications that do transfer well to FPGA-assisted processing. Such computations include the discrete cosine transform (DCT) for JPEG and MPEG image compression [36,37], target detection in image processing [38,39], and signal filtering [5,22].

While examples of FPGA-based DSP operations are abundant, Mangione-Smith observes in [40] that "no companies are known to use reconfigurable computing for a competitive advantage." Singh and Slous accept this challenge in [41] and explore how a commercially available Xilinx XC6200 RPU board can be used to assist real world DSP applications. This work demonstrates how an FPGA-based PCI board can supplement computations for the popular Adobe Photoshop image processing software. Adobe provides software extensions to Photoshop that allow users to write custom filter operations in standard C. Singh and Slous use this software interface to allow Photoshop to directly interact with the FPGA for hardware based filter operations. The software driver extracts image data from Photoshop, transfers the data to memory on the FPGA card, and then triggers the FPGA to begin computations such as colorspace

conversion or 1D/2D convolution. Once computation is complete, the results are transferred back to Photoshop's environment. The conclusions drawn by this work strongly reflect the strengths and weaknesses of current FPGA-assisted systems. While data is processed at a high rate of up to 20Mpixels per second at the FPGA, system data transfers to the card slow the perceived operation to 0.22Mpixels per second. For comparison the authors cite that the on-card FPGA performance is approximately ten times greater than that of a dedicated 4-processor Power PC Genesis MP600 graphics workstation.

7.3 Custom Logic: Packet Scheduling

A potential environment suitable for FPGAs is emerging in high-speed data communication networks. With the increased bandwidth of Gigabit ATM and Ethernet, switch and network interface cards face increased throughput requirements as well as more demanding Quality of Service (QoS) needs. Some developers see the FPGA as a tool for implementing custom network processing logic that directly interacts with high-speed link transceivers [42]. These implementations can be expanded to provide hardware support for QoS oriented packet scheduling. Current schedulers analyze a list of queued packets' statistics and recompute priorities to make intelligent scheduling decisions. In a real-time context this work is non-trivial for CPUs, and may be better suited for FPGAs that can compute priorities in parallel. Additionally, the reconfigurable aspect of FPGAs allows researchers to experiment with complex scheduling algorithms at previously infeasible speeds.

The Illinois Pulsar-based Optical INTerconnect (iPOINT) project seeks to enhance QoS features of the Washington University Gigabit ATM switch through an FPGA card inserted at each switch port [43]. At the lowest levels, these FPGA cards implement an ATM port's standard responsibilities such as VPI/VCI translation, packet header CRC checks, and physical interface management. Recent work in the iPOINT project extends these duties to include multicast support and an elaborate input queue management algorithm known as 3-Dimensional Queuing (3DQ) [44]. The 3DQ design is significant for at least two reasons. First, it improves the service performance of the switch by prioritizing input queue packets based on a combination of virtual circuit ID, destination port, and a packet's global priority level. Second, this hardware implementation built with 1997 technology performs these QoS decisions at a speed sufficient to satisfy OC-12 data rates (622Mbps). Observing that other scheduling algorithms constantly

update queued packets' priorities, FPGAs may prove to be ideal companions for an upcoming generation of high-performance queuing systems.

8. Obstacles and Future Enhancements

As seen in sections 4.2 and 7.2, the primary obstacle for practical use of FPGA-assisted computation is the positioning of the FPGA in the overall system. Performance of the FPGA significantly degrades as the distance of the FPGA from the host CPU and memory subsystem increases. Various proposals suggest methods of decreasing this distance to provide a tighter coupling with the host processor. The first and most practical method is to move FPGA processing boards off the PCI bus and into the high-speed AGP slot in PCs [41]. This upgrade gives the boards a high-speed bus as well as better access to the host's memory subsystem without radical system modifications. The second and more elaborate method addresses the I/O limitations of the actual FPGA chip. In [45] the authors propose adding a high-speed VLSI communication core such as RAMBUS to an FPGA chip. Such a core allows FPGAs to overcome physical pin limitations by multiplexing an FPGA design's data lines onto a high-speed link. This modification is drastic since it calls for FPGA architecture changes as well as an integration of the FPGA directly into the host memory subsystem. The third and most radical approach to improved coupling is to embed the FPGA as a unit inside the CPU [46]. Observing the inclusion of MMX in the x86 architecture, researchers are hopeful that industry will see the embedding of FPGAs in processors as useful in the times of Gigascale design. Such implementations represent a fulfillment of Estrin's 1960 proposal for variable structure processing in CPUs.

The ability to rapidly reconfigure an FPGA is essential to configurable computing and a limitation of current-generation architectures. Two methods of improving this problem include partial reconfiguration and context switching FPGAs. Partial reconfiguration devices such as the XC6200 and Virtex families allow portions of the device to be read or written at a time without disturbing the entire device. This style is conducive to "Virtual FPGA" management operating systems that swap hardware images into and out of the FPGA as needed by application. Partial reconfiguration is a natural extension of previous FPGA research and is expected to give the upcoming Virtex family an edge in the configurable computing community.

A more sophisticated style of dealing with rapid FPGA reconfiguration is though multi-context FPGAs. In these devices additional memory is placed in each logic block to store a fixed number of configurations. These configuration arrays allow multiple hardware image planes, or contexts, to be loaded into the FPGA at once. Contexts are given time slices for which they can perform computation with the FPGA hardware, giving the appearance of several "virtual FPGAs" operating at once. Sanders of Lockheed Martin has fabricated a multi-context FPGA known as the Context Switching Reconfigurable Computing (CSRC) chip to perform high-speed signal processing [48]. Xilinx has also designed a multi-context implementation of the XC4000, though it has not been fabricated [47]. Aggressive views of multi-context FPGAs would suggest allowing context switching at a finer granularity than at the entire plane level. These implementations could better use hardware, realizing that delay through units in a configuration plane is not equal. For example, if a fast unit of a plane completes before a slower unit, the area of the chip that completes calculations early could context switch to another image plane. Thus, hardware is reused in an extremely efficient manner. Unfortunately the overwhelming complexity of multi-context designs is a limitation. Likewise, the device is restricted to a fixed number of contexts. However, these devices provide a number of research opportunities in configurable computing, as well as self-modifying hardware.

9. Conclusions

Due to the limitations of the previous generation of hardware, configurable computing is largely still in its infancy. Recent work and enhancements in upcoming FPGA hardware provides an opportunity for configurable computing to mature into a technology suitable for practical use in the near future. For widespread acceptance, FPGA users must understand a number of issues associated with configurable computing. First and foremost, designers must know the characteristics of both the target FPGA architectures and their system-level integration. Knowledge of the hardware is critical since performance can be masked by system deficiencies. Second, users must understand the types of applications suitable for configurable computing. Architecture features of current FPGA based systems indicate that successful applications use data streams, parallelism, regular computation, constant data values, or custom logic. Conversely, failed FPGA designs often have tight algorithmic loops, floating point precision, or non-

localizable data, and are better suited for general-purpose CPUs and ASICs. Third, while FPGA designs have demonstrated high performance, users must be aware that performance often comes at a cost of a lengthy design process. Design tools being developed today may simplify this process, but the reported performance of automated designs is worse than manual implementations.

The release of the next generation of hardware offers a great deal of promise for the configurable computing community. These devices provide higher gate densities, improved arithmetic support, and methods of fast re-programming ideal for configurable computing. While system integration still remains challenging, the number of proposals addressing the problem suggests that adding FPGAs to general use systems is both worthwhile and practical. Ultimately, such innovations will lead to the computational role for programmable logic envisioned by pioneers such as Estrin.

10. References

[1] J. Villasenor and W Mangione-Smith, "Configurable Computing", in *Scientific American*, June 1997 Available at http://www.sciam.com/0697issue/0697/villasenor.html

[2] W. Mangione-Smith and B. Hutchings, "Configurable Computing: The Road Ahead," in *Reconfigurable Architectures Workshop*, 1997

[3] M. Gokhale et al., "SPLASH: A Reconfigurable Linear Logic Array," in *Proceedings of the International Conference on Parallel Processing*, August 1990

[4] W. Culbertson, R. Amerson, R. Carter, P. Kuekes, and G. Snider, "Defect Tolerance on Teramac Custom Computer," in *Proceedings of the 1997 IEEE Symposium on FPGA's for Custom Computing Machines*

[5] A. DeHon, "Comparing Computing Machines," in *Proceedings of SPIE Vol. 3526*, 1998

[6] A. DeHon, "Why Configurable Computing? The Computational Density Advantage of Configurable Architectures," at http://www.cs.berkeley.edu/~amd/CS294F98/papers/whycc.ps.Z

[7] S. Brown and J. Rose, "Architecture of FPGAs and CPLDs: A Tutorial," *IEEE Design and Test of Computers*, Vol. 13, No. 2, pp. 42-57, 1996

[8] V. Betz and J. Rose, "How Much Logic Should Go in an FPGA Logic Block?" *in IEEE Design &Test of Computers*, January-March 1998

[9] B. Fagin and C. Renard, "Field Programmable Gate Arrays and Floating Point Arithmetic," in *IEEE Transactions on VLSI Systems*, September 1994, pp. 365-367

[10] Xilinx Corporation, "XC4000E and XC4000X Series Field Programmable Gate Array Product Specification", November 1997

[11] Altera Corporation, "FLEX 8000 Programmable Logic Device Family Data Sheet," 1998

[12] Lucent Technologies Corporation, "ORCA OR2CxxA (5.0 V) and OR2TxxA (3.3 V) Series Field-Programmable Gate Arrays Product Brief", December 1997

[13] Xilinx Corporation, "Virtex 2.5 V Field Programmable Gate Arrays Product Specification," November 1998

[14] Altera Corporation, "APEX 20K Programmable Logic Device Family Advance Information Brief", October 1998

[15] Lucent Technologies Corporation, "ORCA Series 3 Field-Programmable Gate Arrays Preliminary Data Sheet", August 1998

[16] Electronic Buyer's News, 30-March-98, at http://www.optimagic.com/market.html

[17] A. Stansfield and I. Page, "The Design of a New FPGA Architecture" available at ftp://ftp.comlab.ox.ac.uk/pub/Documents/techpapers/Ian.Page/cam95.ps

[18] MUSIC Semiconductors, "MUSIC Semiconductors CAM Tutorial," available at http://www.music-ic.com/cam.html

[19] Xilinx Corporation, “XC6200 Field Programmable Gate Arrays”, Data Book, April 1997

[20] J. Cong and S. Xu, “Delay-Optimal Technology Mapping for FPGAs with Heterogeneous LUTs,” in *Proceedings. of 35th Design Automation Conference*, Jun. 1998, pp. 704-707

[21] S. Xing and W. Yu, “FPGA Adders: Performance Evaluation and Optimal Design,” in *IEEE Design & Test of Computers*, January-March 1998, pp. 24-29

[22] R. Petersen and B. Hutchings, “An Assessment of the Suitability of FPGA-Based Systems for use in Digital Signal Processing,” in *Proceedings of the 5th Annual Workshop on Field-Programmable Logic and Applications* (FPL ’95), August 1995

[23] R. Do, H. Kropp, M. Schwiegershausen, and P. Pirsch, “Implementations of Pipelined Multipliers on Xilinx FPGAs," in *Proceedings of the 7th Annual Workshop on Field-Programmable Logic and Applications* (FPL ’97)

[24] W. Ligon III, S. McMillan, G. Monn, K. Schoonover, F. Stivers, and K. Underwood, “A Re-Evaluation of the Practicality of Floating Point Operations on FPGAs,” in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM 98)*, April 1998

[25] Y. Li and W. Chu, “Implementation of Single Precision Floating Point Square Root on FPGAs,” in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM 97)*, 1997

[26] A. Rashid, J. Leonard, and W. Mangione-Smith, “Dynamic Circuit Generation for Solving Specific Problem Instances of Boolean Satisfiability,” in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM 98)*, 1998

[27] W. Fornaciari and V. Piuri, “Virtual FPGAs: some steps behind the physical barriers,” in *Parallel and Distributed Processing, Reconfigurable Architectures Workshop* (RAW 98), 1998

[28] N. Narasimhan, V. Srinivasan, M. Vootukuru, J. Walrath, S. Govindarajan, and R. Vemuri, “Rapid Prototyping of Reconfigurable Coprocessors,” in *International Conference on Application-specific Systems, Architectures and Processors*, August 1996

[29] I. Page, “Parameterised Processor Generation,” *International Workshop on FPGAs at Oxford*, September 1993

[30] A. Agarwal, S. Amarasinghe, R. Barua, M. Frank, W. Lee, V. Sarkar, D. Srikrishna, and M. Taylor, “The RAW Compiler Project,” in *Proceedings of the Second SUIF Compiler* Workshop, 1997

[31] J. Babb et al., “The RAW Benchmark Suite: Computational Structures for General Purpose Computing,” in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM 97)*, 1997

[32] J. Leonard and W. Mangione-Smith, “A Case Study Partially Evaluated Hardware Circuits: Key Specific DES,” in *Proceedings of the 7th Annual Workshop on Field-Programmable Logic and Applications* (FPL ’97)

[33] I. Goldberg and D. Wagner, “Architectural considerations for cryptanaltic hardware.” CS252 Report, May 1996, Available at http://www.cs.berkeley.edu/~iang/isaac/hardware/

[34] T. Kean and Ann Duncan, “DES Key Breaking, Encryption and Decryption on the XC6216,” in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machine*, 1998

[35] Altera Corporation, “Implementing fft with On-Chip RAM in FLEX 10K Devices,” Application Note 84, February 1998

[36] G. Aggarwal and D. Gajski, “Exploring DCT Implementations,” Technical Report UCI-ICS-98-10, March 1998, Available at http://www.ics.uci.edu/pub/gaurav/tech_reports/

[37] B. Schoner, J. Villasenor, S. Molloy, and R. Jain, “Techniques for FPGA Implementation of Video Compression Systems"” in *FPGA’95 ACM/SIGDA International Symposium on Field Programmable Gate Arrays* 1995

[38] Myricom Corporation, “Scalable, Network-Connected, Reconfigurable, Hardware Accelerators for an Automatic-Target-Recognition” Technical Report

[39] J. Leonard and A. Rashid, “Evaluating the Benefits of Hardware Context Switching for Automated Target Recognition,” UCLA EE Dept. Technical Report TR98-1

[40] W. Mangione-Smith et al., “Seeking Solutions in Configurable Computing,” *IEEE Computer, December, Vol. 30, No. 12*. December 1997

[41] S. Singh and R. Slous, “Accelerating Adobe Photoshop with Reconfigurable Logic,” in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, 1998

[42] G. Glykopoulos, “Design and Implementation of a 1.2 Gbit/s ATM Cell Buffer using a Synchronous DRAM chip,” University of Crete Technical Report FORTH-ICS/TR-221, July 1998, Available at ftp://ftp.ics.forth.gr/tech-reports/1998/

[43] H. Duan, J. Lockwood, and S. Kang, “FPGA Prototype Queuing Module for High Performance ATM Switching,” in *Proceedings of the Seventh Annual IEEE International ASIC Conference*, September, 1994

[44] H. Duan, J. Lockwood, S. Kang, and J. Will, “A High-performance OC-12/OC-48 Queue Design Prototype for Input-Buffered ATM Switches,” in *IEEE Infocom '97*, April 1997

[45] N. Margolus, “An FPGA architecture for DRAM-based systolic computations,” in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM 97)*, 1997

[46] J. Hauser and J. Wawrzynek, “Garp: A MIPS Processor with a Reconfigurable Coprocessor,” in *Proceedings of the IEEE Symposium on Field Programmable Gate Arrays for Custom Computing Machines (FCCM 97)*, April 1997

[47] S. Trimberger, D. Carberry, A. Johnson, J. Wong, “A Time-Multiplexed FPGA,” in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM 97)*, April 1997

[48] S. Scalera and J. Vazquez, “The Design and Implementation of a Context Switching FPGA,” in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM 98)*, 1998