ABSTRACT

H-SIMD MACHINE: CONFIGURABLE PARALLEL COMPUTING FOR DATA-INTENSIVE APPLICATIONS

by
Xizhen Xu

This dissertation presents a hierarchical single-instruction multiple-data (H-SIMD) configurable computing architecture to facilitate the efficient execution of data-intensive applications on field-programmable gate arrays (FPGAs). H-SIMD targets data-intensive applications for FPGA-based system designs. The H-SIMD machine is associated with a hierarchical instruction set architecture (HISA) which is developed for each application. The main objectives of this work are to facilitate ease of program development and high performance through ease of scheduling operations and overlapping communications with computations.

The H-SIMD machine is composed of the host, FPGA and nano-processor layers. They execute host SIMD instructions (HSIs), FPGA SIMD instructions (FSIs) and nano-processor instructions (NPIs), respectively. A distinction between communication and computation instructions is intended for all the HISA layers. The H-SIMD machine also employs a memory switching scheme to bridge the omnipresent large bandwidth gaps in configurable systems. To showcase the proposed high-performance approach, the conditions to fully overlap communications with computations are investigated for important applications. The building blocks in the H-SIMD machine, such as high-performance and area-efficient register files, are presented in detail. The H-SIMD machine hierarchy is implemented on a host Dell workstation and the Annapolis Wildstar II FPGA board. Significant speedups have been achieved for matrix multiplication (MM), 2-dimensional discrete cosine transform (2D DCT) and 2-dimensional fast Fourier transform (2D FFT) which are used widely in science and engineering.

In another FPGA-based programming paradigm, a high-level language (here ANSI C) can be used to program the FPGAs in a mode similar to that of the H-SIMD machine in terms of trying to minimize the effect of overheads. More specifically, a multi-threaded overlapping scheme is proposed to reduce as much as possible, or even completely hide, runtime FPGA reconfiguration overheads. Nevertheless, although the HLL-enabled reconfigurable machine allows software developers to customize FPGA functions easily, special architecture techniques are needed to achieve high-performance without significant penalty on area and clock frequency. Two important high-performance applications, matrix multiplication and image edge detection, are tested on the SRC-6 reconfigurable machine. The implemented algorithms are able to exploit the available data parallelism with independent functional units and application-specific cache support. Relevant performance and design tradeoffs are analyzed.

H-SIMD MACHINE: CONFIGURABLE PARALLEL COMPUTING FOR DATA-INTENSIVE APPLICATIONS

by
Xizhen Xu

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Electrical Engineering

Department of Electrical and Computer Engineering

May 2006

APPROVAL PAGE

H-SIMD MACHINE: CONFIGURABLE PARALLEL COMPUTING FOR DATA-INTENSIVE APPLICATIONS

Xizhen Xu

Dr. Sotirios G. Ziavras, Dissertation Advisor — Date
Professor of Electrical and Computer Engineering, NJIT

Dr. Edwin Hou, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, NJIT

Dr. Jie Hu, Committee Member — Date
Assistant Professor of Electrical and Computer Engineering, NJIT

Dr. Roberto Rojas-Cessa, Committee Member — Date
Assistant Professor of Electrical and Computer Engineering, NJIT

Dr. Alexandros V. Gerbessiotis, Committee Member — Date
Associate Professor of Computer Science, NJIT

BIOGRAPHICAL SKETCH

Author: Xizhen Xu

Degree: Doctor of Philosophy

Date: May 2006

Undergraduate and Graduate Education:

- Doctor of Philosophy in Electrical Engineering
  New Jersey Institute of Technology, Newark, NJ, 2006
- Master of Science in Electrical Engineering
  Northwestern Polytechnic University, Xi'an, China, 1996
- Bachelor of Science in Electrical Engineering
  Northwestern Polytechnic University, Xi'an, China, 1993

Major: Electrical Engineering

Presentations and Publications:

X. Xu and S. G. Ziavras, "A Coarse-Grain Hierarchical Technique for 2-Dimensional FFT on Configurable Parallel Computers," *IEICE Trans. on Information and Systems, Special Issue on Parallel/Distributed Computing and Networking* , vol. E89-D, no. 2, pp. 639-646, Feb. 2006.

X. Xu, S. G. Ziavras, and T.-G. Chang, "An FPGA-Based Parallel Accelerator for Matrix Multiplications in the Newton-Raphson Method," *IFIP International Conference on Embedded and Ubiquitous Computing*, Nagasaki, Japan, pp. 458-468, Dec. 2005.

X. Xu and S. G. Ziavras, "H-SIMD Machine: Configurable Parallel Computing for Matrix Multiplication," *IEEE International Conference on Computer Design*, San Jose, CA, pp. 671-676, Oct. 2005.

X. Xu and S. G. Ziavras, "A Hierarchically-controlled SIMD Machine for 2D DCT on FPGAs," *IEEE International Conference on Systems-on-Chip*, Herndon, VA, pp. 276-279, Sept. 2005.

X. Xu and S. G. Ziavras, "A Configurable and Scalable SIMD Machine for Computation-Intensive Applications," *WSEAS Trans. on Computers*, vol. 2, no. 4, pp. 1021-1029, Oct. 2003 (invited paper).

X. Xu and S. G. Ziavras, "Iterative Methods for Solving Linear Systems of Equations on FPGA-Based Machines," *18th International Conference on Computers and Their Applications*, Honolulu, Hawaii, pp. 472-475, Mar. 2003.

to Yun & Andy
for all the love you gave to me

ACKNOWLEDGMENT

First of all, I own my deepest gratitude to my advisor, Dr. Sotirios G. Ziavras, for his visionary guidance and endless encouragement through my PhD study at NJIT. Greatest appreciation gives to his motivation, creativity, and rich knowledge that always inspire me to move forward in my dissertation research. His strong support and invaluable advisement were always there for me whenever I needed. Greatest thanks give to his enduring patience. From his answer to each of my question to his suggestion to every writing improvement in my dissertation, what I learned is not just a technical concept or an English word; I learned important personal characters as an excellent professor that will continue to guide me in the future.

I also would like to thank Dr. Jie Hu for his technical guidance and insightful comments to this dissertation and for serving on my dissertation committee. My thanks also extend to Dr. Alexandros V. Gerbessiotis, Dr. Edwin Hou and Dr. Roberto Rojas-Cessa for their technical suggestions and for serving on my dissertation committee. I would like to thank the Hashimoto family and fellowship program. Their generosity allowed me to receive a Hashimoto Fellowship for the 2005-2006 academic year. My sincere thanks go to all my friends and colleagues in the CAPPL lab for the help you gave and the friendship we shared. I wish everybody from CAPPL great achievements in the future.

The love, support and encouragement from my family was essential in completing this dissertation. I am extremely grateful for all the sacrifices my parents made to raise me up. My deep thanks to my beloved wife, Yun Teng, for her support, encouragement, tolerance, and patience from anywhere I needed. Without your love and understanding, I would not be here today.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Research Background and Problem Statement

This section presents a research background for this dissertation as it stems from current mainstream computing paradigms. It involves general-purpose microprocessors, application-specific integrated circuits (ASICs) and configurable systems implemented with FPGAs. The motivations for this dissertation are also stated.

1.1.1 Research Background

Conventionally, custom ASICs and general-purpose microprocessors are the two primary hardware technologies used to implement application algorithms [7] [8]. The choice of the specific technology is based on a tradeoff between the contradictory design forces of generalization and specialization. With temporal computing, general-purpose microprocessors can provide adequate performance, reasonable efficiency and reduced costs for many applications but often fail to meet the most demanding performance requirements. ASICs can provide excellent performance by carrying out spatial computing but at the cost of greater non-recurring engineering (NRE) expenses and inflexibility. Obviously, neither ASICs nor general-purpose microprocessors can provide a very good balance among performance, flexibility and cost.

In order to meet these challenges, configurable computing emerged in the past decade as a significant computing paradigm [9]. It attempts to combine the advantages of both ASICs and general-purpose microprocessors. The relationship among ASICs, microprocessors and FPGAs is shown in Table 1.1. Specifically, a configurable computer is characterized by hardware programmability which is not available in other computing paradigms.

The idea of configurable computing was first proposed in 1960 by the computer pioneer Gerald Estrin [10], and yet the technology to realize Estrin's configurable computer did not exist at that time. However, the vision was kept intact and has recently become possible with the development of new-generation FPGA technology. Implemented with programmable logic that can be organized and structured to fit the natural dataflow of an application, configurable computing refers to customizing the system logic functionality and interconnect connectivity through post-fabrication and user-defined configuration once before an application is run. This definition can be further extended to also include run-time configurable computing (run-time reconfiguration, on-line reconfiguration or adaptive computing) by which the functionality of the configurable hardware can be modified during application execution, as needed to fit the latter. The run-time configurable computer can better exploit underutilized hardware and properly partition a large application into the limited set of configurable resources. However, the run-time configuration overhead still remains a big challenge for this computing paradigm to be accepted in mainstream configurable computing.

Configurable computers, such as SPLASH-2 [4], RENCO [11], PAM [12], Chimaera [13], PRISM [14] and DISC [15], have proven to be capable of speeding up a wide variety of applications, such as data encryption [16] [17], system acquisition [18], image processing [19] and artificial intelligence [20] [21]. Typically, a configurable computer consists of the host and the configurable system-on-a-chip (CSOC) components. The host performs the operations that cannot be done efficiently on the CSOC, such as data-dependent control and variable-length loops, while the computation-intensive parts of the application are mapped to the CSOC(s). The FPGAs are the most widely deployed CSOC hardware because of their configurability, ease of application development and recently rich resource provision, making them increasingly popular.

High-performance parallel computers have been the driving force throughout the computing history and have accomplished a great deal of success in solving

Table 1.1 Relationship Summary of ASICs, Microprocessors and FPGAs

		ASICs	Microprocessor	FPGAs
Programmability	**Software**	No	Yes	No
	Hardware	No	No	Yes
Spatial computing		Yes	No	Yes
Temporal computing		No	Yes	No

computation-intensive problems. To distinguish among different types, we can first classify computers as follows according to Flynn's notions of the instruction and data streams [22]: 1) SISD (single instruction stream over a single data stream); 2) SIMD (single instruction stream over multiple data streams); 3) MIMD (multiple instruction streams over multiple data streams); 4) MISD (multiple instruction streams over a single data stream). The SISD computer is the most widely used computer model corresponding to serial machines. The other three classes represent parallel-processing types of computers. The SIMD architecture involves multiple processors simultaneously executing the same instruction on different data. "General-purpose" parallel computers are generally reserved for MIMD machines in which each processor fetches its own instruction and operates on its own data. The MISD architecture is often deemed impractical and may represent systolic arrays. The SIMD computing paradigm for distributed-memory parallel machines is well-known for its scalability, data parallelism and high throughput [23] [24] [25] [26]. It will be employed as the underlying architecture for the proposed configurable computing system. Nevertheless, the further development of supercomputers has been recently affected adversely by high prices, long development cycles, difficulties in programming and high maintenance costs [27]. Thus, system architects have been motivated to consider alternatives. An effective approach to this problem is to implement the high-performance computing architecture by employing one or more configurable components because of

a significant cost reduction in prototyping and rather efficient execution on the CSOCs. Particularly this is true for platform FPGAs that have capacity of up to 10 million system gates with 90nm VLSI technologies [28].

1.1.2 Motivation and Problem Statement

Currently most of the FPGA design tools use a design methodology which fits into the ASIC development model. First, they implement the design using a Hardware Description Language (HDL) and carry out functional simulations; then, logic optimization is employed for logic synthesis and technology mapping; finally, the synthesized design is placed and routed for the vendor's FPGA architecture. Although the ability to program the configurable hardware in such a manner gives the user access to more functionality, it discourages the acceptance of configurable computing platforms due to more arduous design efforts that only a few application designers can deal with. Prior work has taken on this challenge and made some progress. The Garp [29], System C [30], Handle-C [31] and Streams C [32] languages made efforts to extract the hardware realization from a high-level behavioral specification presented in a HLL (such as C/C++). However, the corresponding compilers often require manual hardware/software partitioning which may deteriorate the quality of the results in terms of area, power and system frequency [33]. JBits [34] and JHDL [35] exploit the Java language to facilitate FPGA application development at the hardware level and do not enable high-level synthesis. Additionally, all these tools are difficult to use in exploring the data parallelism inherent in application algorithms.

Another challenge is to bridge bandwidth gaps between the various levels in configurable systems. Generally, there are three types of connections between a host and the CSOCs. First, like the Altera Nios SOPC [36] embedded systems, the CSOCs can serve as functional units tightly coupled with the host processor data path. They could implement an extension of the host processor's ISA (Instruction Set Architecture), such as custom instructions [37] [38]. Second, the CSOCs may be used as coprocessors [29] [39] [40] [41]

which can be larger than a functional unit and less tightly interfaced to the host. They can then provide more resources to take advantage of the existing parallelism in applications. Third, the CSOCs can be used as attached processing units that communicate with the loosely connected host through I/O interfaces [4] [42] [43]. Each scheme has its own pros and cons. The tighter the integration is, the more efficient the communications and yet the fewer the available programmable hardware resources. On the other hand, a more loosely coupled style allows for greater parallelism while it may suffer from higher communication overheads. Extrapolating from Amdahl's Law, the speedup benefits gained from a CSOC implementation can be significantly reduced or even removed if a bandwidth bottleneck exists. Recent work [4] [44] [45] [46] acknowledges the bandwidth bottlenecks in configurable machines, however, without taking them into account in a solution. Usually, either a complete SOC or a reduction in the host access frequency can be employed to solve this problem, but at the expense of lower overall performance. The former solution embeds the host in the CSOC(s), thus increasing tremendously the required configurable resources. The latter can not be applied to applications that require very frequent host accesses.

1.2 Research Objectives

In order to meet the above challenges, the objectives in this dissertation are as follows:

- The notion of a Hierarchical SIMD (H-SIMD) architecture is proposed to facilitate the efficient execution of data-intensive applications on a configurable computer. The target is data-parallel applications. The application is partitioned into different granularities corresponding to the layers in the H-SIMD machine. The enabling SIMD architecture exploits the inherent data parallelism at the application implementation layer. An ISA is to be developed for each application area to facilitate ease of program development and data communication overlapping with

computation operations. This is similar to a technique proposed in [47] for PC clusters.

- The bandwidth bottleneck in the configurable computer is put into perspective. Taking advantage of the hierarchical program coding for H-SIMD, as described above, a memory switching scheme is employed to overlap communications with computations as much as possible. The conditions to achieve a complete overlap are studied for applications running on the H-SIMD machine. This scheme can be applied at run-time, where part of the memory is loaded with new operands while another part is used at that time to run the application.

- An innovative large-sized register file is designed to take advantage of the FPGA on-chip memory with minimal resource consumption. Thus, more functional units can fit in a single CSOC as compared to a conventional approach.

- Typical data-intensive applications are implemented on the target Annapolis Wildstar II FPGA board that resides in our laboratory, in order to evaluate the performance of the H-SIMD design methodology. These applications are used widely in engineering and science. The computation time, the consumed FPGA resources and the performance of the H-SIMD machine are measured to facilitate performance comparisons with other computing platforms.

- A HLL-enabled reconfigurable machine is finally used to exploit the available data parallelism in programs with independent functional units and application-specific cache support. The purpose is to demonstrate that the overlapping technique can also be applied to reduce the effect of runtime reconfiguration. For this purpose, a multi-threaded overlapping scheme is proposed to reduce as much as possible, or even completely hide, runtime FPGA reconfiguration. High-performance can be achieved

without significant penalty on area and clock frequency. Relevant performance and design tradeoffs are analyzed.

1.3 Organization of the Dissertation

Chapter 2 examines state-of-the-art CSOC platforms in detail. Their features and shortcomings for configurable system development are analyzed. Commercial-off-the-Shelf (COTS) platform FPGAs are the chosen research platforms in this dissertation.

Chapter 3 presents the H-SIMD machine and the philosophy for the development of the associated Hierarchical Instruction Set architecture (HISA). Each layer in this hierarchy is studied according to its function and granularity. A memory switching scheme is discussed to overlap communications with computations as much as possible.

Chapter 4 implements three applications on the H-SIMD machine for the purpose of performance evaluation. The three applications are matrix multiplication, 2-dimensional fast Fourier transform and 2-dimensional discrete cosine transform. Test results are presented and analyzed to verify the effectiveness of the proposed H-SIMD machine.

Chapter 5 employs the HLL-enabled SRC-6 reconfigurable machine to exploit the available data parallelism with independent functional units and application-specific cache support. Two important high-performance applications, matrix multiplication and image edge detection, are tested on the SRC-6 machine. The implemented algorithms are able to achieve high-performance without significant penalty on area and clock frequency. Relevant performance and design tradeoffs are analyzed. A multi-threaded overlapping scheme is also proposed to reduce as much as possible, or even completely hide, runtime FPGA reconfiguration.

Chapter 6 draws conclusions and presents future directions.

CHAPTER 2

CSOC PLATFORMS

Configurable-System-on-a-Chip (CSOC) platforms are the workhorse behind configurable computers. Many projects have been launched to study high-performance architectures for CSOCs. The core technology of CSOCs is generally classified into two categories: COTS FPGAs and custom FPGAs. The former are widely deployed because of their rich resources, low cost and standard development process. Mainstream products in this category are made by leading FPGA companies such as Actel, Altera, Lattice and Xilinx. Although a wide variety of COTS FPGA-based computing platforms have been constructed and reported in the literature, two such systems stand out to exemplify what can be achieved: SPLASH-2 [4] and DECPeRLE [12]. Custom FPGA-based configurable systems are mainly developed in research institutions to customize FPGA techniques for specific applications. Prominent projects in this category include MorphoSys [5], PipeRench [48], MATRIX [49], RAW [50], and Garp [29]. Each of these projects proposes a distinctive framework for configurable hardware/software development.

2.1 COTS FPGA Technology

The FPGA technology was invented in the mid 1980's by Xilinx Inc. and developed rapidly into a mainstream technology. It has evolved from logic glue devices to heterogeneous coarse-grain platform FPGAs. The implementation of IEEE single precision floating-point multipliers on a single FPGA was even impratical in 1994 [51] whereas current state-of-the-art FPGAs can easily implement 32 such functional units in parallel [52]. Since FPGAs were initially designed as bit-level fine-grain devices, a large consumption of logic and routing resources may be required for high bandwidth applications. In order to make up for these disadvantages, platform FPGAs have recently become the focus in this field. These

advanced devices feature coarse-grain components such as built-in hardwired processors (e.g., the IBM PowerPC or ARM core), substantial amount of on-chip SRAM, digital clock management (DCM), and support for some of the latest I/O signaling techniques [28]. This section presents the coarse-grain components which are embedded into Xilinx platform FPGAs, the target of our implementations.

2.1.1 On-chip Memory

There are two types of on-chip memory in Xilinx platform FPGAs: distributed SelectRAM and block SelectRAM. According to the Xilinx Virtex-II datasheet [28], there are four slices in each configurable logic block (CLB) and each slice contains two 4-input LUTs (Look-Up Tables), carry logic, arithmetic logic gates, multiplexers, and two flip-flops. LUTs are also referred to as function generators because they can be configured as a 4-inpt LUT, 16 bits of distributed SelectRAM memory, or a 16-bit variable-tap shift register element. The LUT elements within a CLB can be configured to implement memory modules of up to 128 bits. However, the implementation of the distributed SelectRAMs is extremely LUT-consuming because it is done at the fine-grain level. In order to get bigger capacity for on-chip memory without consuming the precious LUT and routing resources, FPGA manufacturers embed coarse-grain block SelectRAMs in platform FPGAs which can be used for large memory storage with the help of a few LUT and routing resources. Xilinx Virtex-II devices, for example, incorporate a large amount of block SelectRAMs containing up to 4704Kbits. These block RAMs supplement the distributed SelectRAMs that consume lots of LUT resources.

Additionally, each Virtex-II block SelectRAM is a true dual-port RAM with two independently clocked and independently controlled synchronous ports that access a common storage area. Both ports are functionally identical. The block SelectRAM diagram is shown in Figure 2.1. The read operation is fully synchronous: the stored data in the given address is loaded into the output register when the rising or falling edge takes

place (depending on the configuration of the clock polarity). A write operation carries out a simultaneous read operation which is performed in one of three configurations [28]: WRITE_FIRST, READ_FIRST, or NO_CHANGE. The WRITE_FIRST option is a transparent mode; the input data is written into the memory and also transferred into the output register on the same clock edge. The READ_FIRST option is used to push the prior content of the memory cell to the output register and write the input data into the memory cell on the same clock edge. The NO_CHANGE option maintains the content of the output register regardless of the write operation. During the NO_CHANGE mode, only the read operation can change the output register.

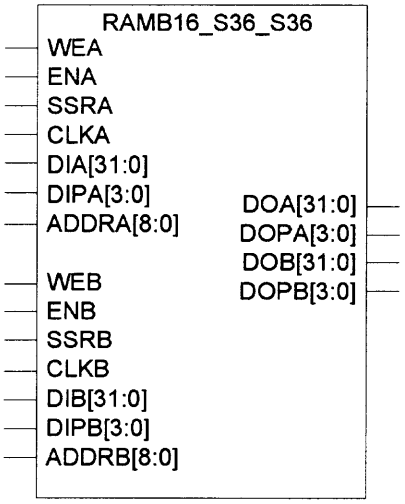


Figure 2.1 Virtex II on-chip block SelectRAM memory.

2.1.2 Clock Tree Distribution and Management

All platform FPGAs have their own well-designed clock trees, which are used to drive the synchronous components within the chip. Xilinx Virtex-II devices have 16 clock input pins. Eight clock pins are in the upper part of the device while the other eight are located in the lower part. Each Xilinx Virtex-II device is divided into four quadrants: North-West,

South-West, North-East, and South-East. The clock distribution is based on a scheme of eight clock trees per quadrant [28].

Clock skew and current driving become major issues due to the long latencies in the clock tree. Correspondingly, platform FPGAs contain a clock buffer and embedded digital clock management modules (DCMs). The clock buffer, such as BUFG, can generate the global clock signal whenever an input signal drives a clock signal or whenever an internal clock signal reaches a certain fanout.

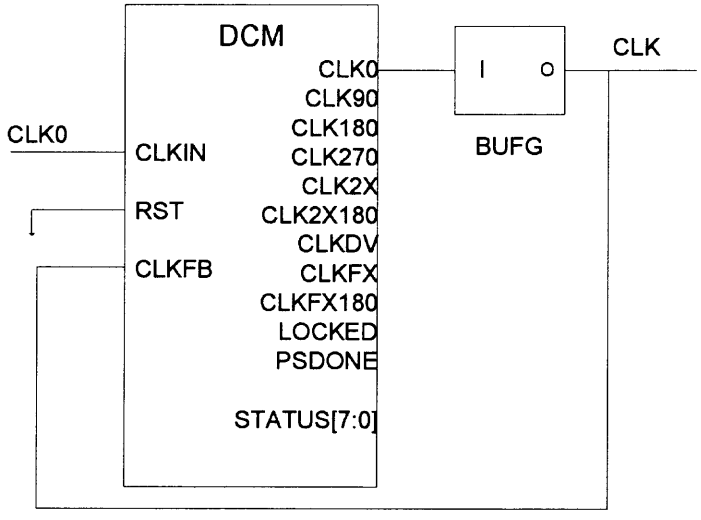


Figure 2.2 Configuration of DCM and BUFG for the clock deskew and current driving.

DCM is another multi-functional clock resource in a platform FPGA [28]. It features de-skew, frequency synthesis and phase shifting functions. The de-skewed clock tree is essential to the functions of all digital circuit designs. Frequency synthesis provides great design flexibility. The phase shifting function is especially useful for multi-chip DDR SRAM burst reading operations. A typical configuration of DCM and BUFG is shown in Figure 2.2. CLK0 is the clock source which is first routed through DCM and then amplified by BUFG. The BUFG output is used to drive the synchronous design inside the FPGAs as well as feed back the DCM component to keep CLK0 and CLK in phase.

2.1.3 I/O Technology

In order to meet a wide variety of I/O standards and speed challenges, each user pin on platform FPGAs is programmable for all the frequently-used single-ended or differential-ended I/O standards. In the Xilinx Virtex-II platform FPGA family, 19 single-ended and 6 differential-ended I/O standards are supported such as LVTTL, LVCMOS33, LVCMOS15, LVDS, SSTL, HSTL, GTL+, etc. All these programmable I/O blocks greatly facilitate interface designs. For example, low voltage differential signaling (LVDS) with a double data rate (DDR) register is capable of delivering 840Mbps performance. Each IOB (input-output block) includes six storage elements, as shown in Figure 2.3 [28]. Each storage element can be configured either as an edge-triggered D-type flip-flop or as a level-sensitive latch. On the input, output, and 3-state path, one or two DDR registers are available. The double data rate is directly accomplished by the two registers on each path, clocked by the rising edges (or falling edges) from two different clock nets. The two clock signals are generated by the DCM and must be 180 degrees out of phase. The DDR register splitter/merger is very useful for high-speed data transmission and conserves FPGA pin usage.

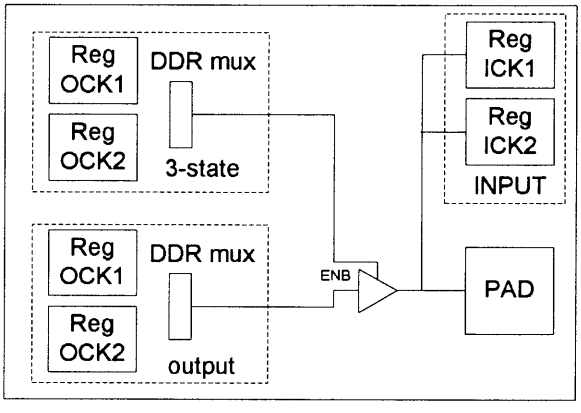


Figure 2.3 Virtex-II IOB diagram.

2.1.4 Multiplier

Multiplication is a common arithmetic operation in scientific applications. If conventionally implemented with fine-grain LUTs, a high bandwidth multiplier will consume too many FPGA resources. To avoid this problem, platform FPGA devices feature a large number of embedded high-bandwidth multipliers. For example, there are 144 two's-complement embedded multipliers in Xilinx Virtex-II FPGA devices [28]. These embedded multipliers offer fast, efficient means to create multiplication products for 18-bit signed inputs. Also, cascading the multipliers with additional logic resources can yield larger multipliers of higher bandwidth.

2.1.5 Configurable Logic Blocks

The backbone of Xilinx FPGA devices is an array of configurable logic blocks (CLBs). They are used to build combinatorial and synchronous logic designs. Each CLB element is tied to a switch matrix to access the general routing framework and contains four similar slices with fast local feedback within the CLB as shown in Figure 2.4. Each slice includes two 4-input LUTs, carry logic, arithmetic logic gates, multiplexers, and two flip-flops. Each 4-input LUT can be programmed as a 4-input LUT, 16 bits of distributed SelectRAM memory, or a 16-bit variable-tap shift register.

From the above analysis of state-of-the-art programmable technology, the clear trend for COTS CSOCs can be observed: from fine-grain PLDs (Programmable Logic Devices) to heterogeneous coarse-grain platform configurable devices.

2.2 COTS FPGA-Based Computing Systems

There is a wide variety of COTS FPGA-based computing systems that have been reported in the literature. TM-2 is based on Altera FPGAs running as an attached functional unit to a SUN workstation [53]. Researchers at Brown University have developed PRISM [54]. Chameleon is based on Algotronix CAL FPGAs [55]. BORG and BORG II are built with

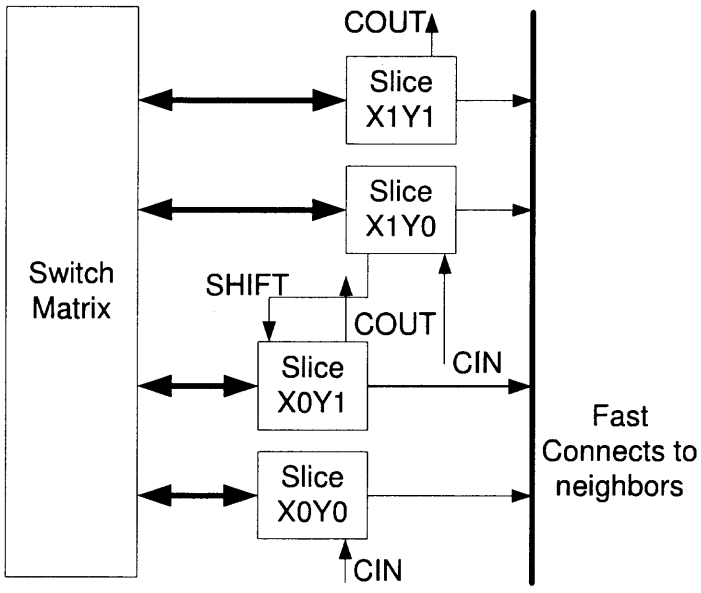


Figure 2.4 Xilinx Virtex-II CLB.

Xilinx FPGAs [56]. SPYDER and RENCO are designed for run-time configuration [11]. Here, the focus is put on SPLASH-2 [4], an exemplary configurable computing system built at the Supercomputer Research Center for defense analysis. The SPLASH-2 system connects Xilinx 4010 FPGAs in a linear systolic array. Figure 2.5 shows a system-level view of the SPLASH-2 architecture. SPLASH-2 is designed as an attached functional unit to the host processor, which is typically a SUN SPARC-II. It is connected to the host through an interface board that extends the address and data buses. The SUN host can access memories and memory-mapped control registers via these buses. Each SPLASH-2 processing board contains 16 Xilinx FPGAs as processing elements (PEs), say X1 - X16. In addition, the 17th Xilinx FPGA (X0) controls the data flow into the processor board. Each PE has 256K 16-bit memory that is also mapped into the address space of the host processor. The PEs are connected through a crossbar that is programmed by X0. The SPLASH-2 system supports several models of computation, including PEs running in the SIMD mode or MIMD mode. SPLASH-2 uses commercial VHDL synthesis tools for

application development. A VHDL model of the complete system is provided that includes all board-level interconnects, memories, and the host interface. Applications are manually partitioned into blocks for each used FPGA. Each block is designed individually, and the integration of the results of all the blocks gives the solution for the application running on SPLASH-2. Significant speedups have been reported for a wide variety of applications, such as image processing [57] [58] and genetic programming [59]. SPLASH-2 has been partially commercialized by Annapolis Micro Systems.

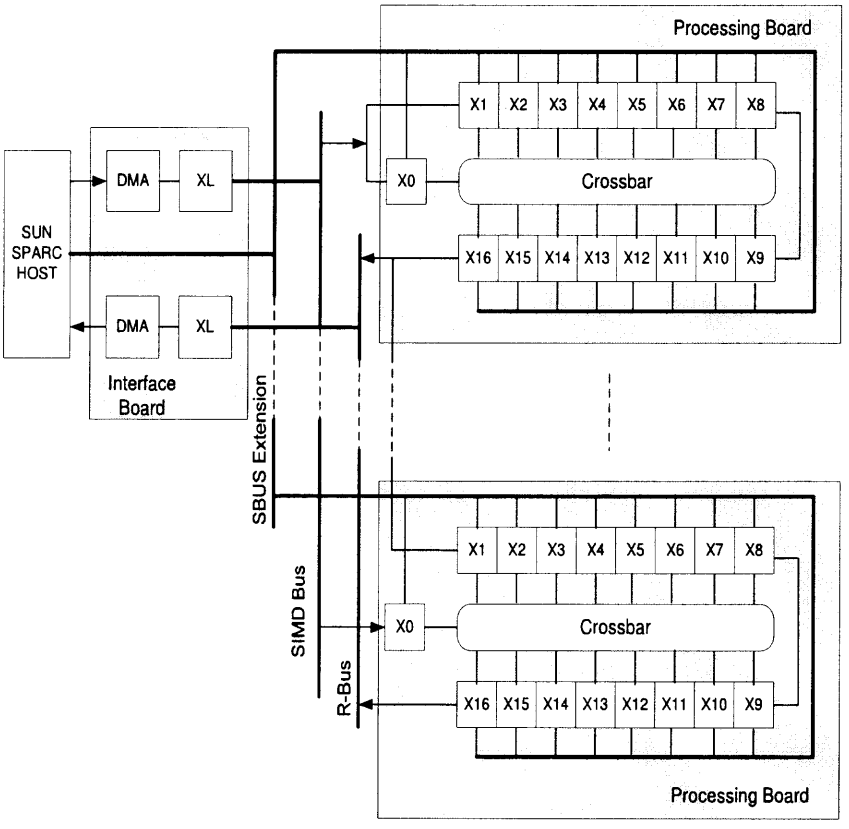


Figure 2.5 SPLASH-2 computing platform (adapted from [4]).

2.3 Custom CSOC Studies

Many research projects have proposed uncommon architectures for FPGAs. Most of them have demonstrated performance success for specific applications and favor run-time configurability. Broadly speaking, custom CSOCs can be classified as either coarse-grain or fine-grain. For example, MATRIX [49], PipeRench [48], MorphoSys [5], and RAW [50] are coarse-grain prototypes of configurable computing systems whereas Garp [29] features fine granularity.

2.3.1 MorphoSys

MorphoSys is a novel model for configurable computing targeting applications with inherent data parallelism and high regularity, and high throughput requirements. Examples of such applications are video compression, graphics and image processing, data encryption, and DSP transforms. The MorphoSys architecture is composed of a CSOC, a general-purpose processor, and a high-bandwidth memory interface, as shown in Figure 2.6. Given the nature of the target applications, the configurable component is organized in the SIMD fashion as an array of reconfigurable cells (RCs). Since most of the target applications possess word-level granularity, the RCs are also coarse-grain. The core (RISC) processor controls the operation of the configurable cell array. The high-bandwidth data interface consists of a specialized streaming buffer to handle data transfers between the external memory and the configurable cell array. The intent of the MorphoSys project is to study the viability of the integrated configurable computing model in satisfying the increasing demand for low cost stream or frame data processing needed for data-intensive applications.

2.3.2 PipeRench

PipeRench has a striped FPGA architecture suitable for run-time configuration. The striped FPGA differs from traditional FPGAs in two ways. First, PipeRench is configured at a granularity that corresponds to the chosen basic unit of configuration, the pipeline stage. An

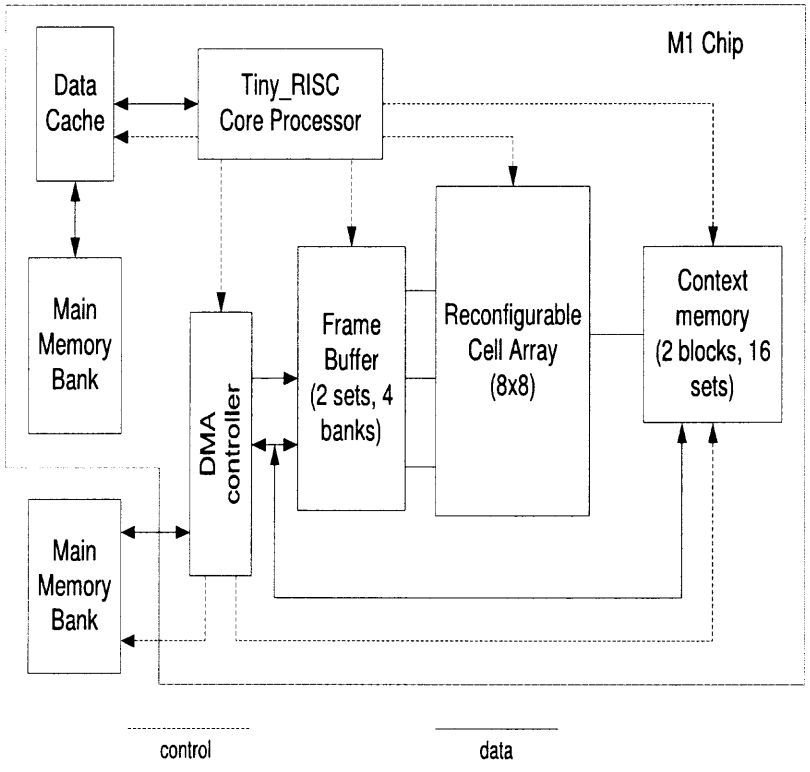


Figure 2.6 Components of MorphoSys implementations (adapted from [5]).

on-chip configuration cache allows pipeline stages to be loaded into FPGA cells at a very high speed. Second, the striped FPGA has special interconnects for the implementation of pipelined applications. The most important idea behind this striped architecture is to virtualize an application with v virtual pipeline stages on a device with a capacity of p physical stages ($p < v$). An FPGA stripe can be configured in one clock cycle from the data stored in a wide, on-chip configuration cache. This provides high speed configuration and allows the unused configuration information to be stored in a single, large on-chip RAM. Modification of the configuration cache can take place concurrently with execution, so there is no hardware constraints to the amount of virtual hardware that can be emulated. This greatly eases compiler development. The compiler begins by reading a description of the architecture from a dataflow intermediate language (DIL) input. PipeRench claims

that its compiler will run two or three orders of magnitude faster than commercial tools because of its deterministic, linear-time, greedy place-and-route algorithm. Yet, designed as an attached coprocessor, PipeRench has limited bandwidth between the main memory and the processor. This places significant limitations on the types of applications that can realize a speedup.

2.3.3 MATRIX

The MATRIX architecture proposes the design of a basic functional unit (BFU) for a configurable system. MATRIX is composed of an array of identical, 8-bit BFUs overlaid with a configurable network. The coarse-grain 8-bit BFUs contain a 256×8 memory, an 8-bit ALU and reduction control logic including a $20 \times 8bit$ NOR plane. The BFUs assume a three-level interconnection network and may be configured for operation in the VLIW (Very Long Instruction Word) or SIMD fashion. The configurable network is a hierarchical collection of 8-bit busses for both data and instruction distribution. Unlike traditional FPGA interconnects, MATRIX has the option to dynamically switch the network connections. The MATRIX port configuration is one of the keys to the architecture's flexibility. Each port in MATRIX can be configured in one of three modes: the static value mode, static source mode, and dynamic source mode. These modes are useful for dataflow or control flow to BFUs. The 100MHz frequency was estimated for the prototype MATRIX chip since a complete system organization based on the BFU is not available yet. The MATRIX approach is too generic and at least one potential problem is the complexity and overhead of the BFU control unit.

2.3.4 RAW

This design implements a highly parallel architecture in the form of a Reconfigurable Architecture Workstation (RAW). The architecture is organized in the MIMD manner with multiple instruction streams. It has multiple RISC processors, each having fine-grain

logic as the configurable component. RAW exposes wire delays at the ISA level. This allows the compiler to explicitly manage gates in a scalable fashion. RAW provides a direct, parallel interface to all of the chip resources: gates, wires, and pins. However, the architecture has a distributed nature and the RISC processors do not exhibit the close coupling present in the RC Array elements. This may have an adverse effect on the performance for high-throughput applications that involve many data exchanges.

2.3.5 Garp

Garp is the architecture of a general-purpose processor tightly coupled with a configurable array. The loading and execution of configurations on the configurable array is always under the control of a program running on the main processor. The main processor is a modified MIPS-II processor whose floating-point unit is replaced with a configurable array. Garp's goal is to execute data-intensive operations on the configurable array and leave general-purpose operations to the processor. Garp's configurable array is composed of entities called blocks. One block on each row is known as a control block. The rest of the blocks in the array are logic blocks that correspond roughly to the CLBs of the Xilinx Virtex series. Software tools have been created that make it possible to write C programs for Garp and then simulate them with approximate clock-cycle accuracy. The major constraint on Garp is the limited amount of configurable resources that can only speedup the application by implementing custom instructions. Table 2.1 summarizes the characteristics of the CSOCs described in this section.

Table 2.1 Comparison of Configurable Systems

	COTS	Granularity	Contexts	Run-time configuration
Commercial FPGA	Yes	Fine	Single	Dynamic/static
MorphoSys	No	Coarse	Multiple	Dynamic
MATRIX	No	Coarse	Multiple	Dynamic
PipeRench	No	Coarse	Multiple	Dynamic
RAW	No	Coarse	Single	Static
Garp	No	Fine	Multiple	Static

CHAPTER 3

THE H-SIMD MACHINE

Several applications that became mainstream in the last decade are characterized by their need for high throughput and data-intensive computations. Such examples are automatic target recognition [5], power-flow solution [60], digital processing [61], and image processing [19]. For this reason, mainstream computer architecture designs have become performance-driven. Many such tasks that are data-intensive can be performed efficiently on SIMD architectures. This trend can be seen even in the Intel Pentium series processors, and Motorola's MPC7400 and Sony's Playstation2 [62]. Additionally, VLSI technology has made such big progress that systems involving dozens of boards a few decades ago can now fit in a single chip. On the other hand, the NRE cost of SOC solutions is so prohibitive that many designs need to hedge the risks [63]. As COTS FPGAs continue to grow in size and complexity, they provide an ideal configurable and cost-efficient solution for prototyping, and even implementing custom parallel computing architectures. This chapter presents the H-SIMD hierarchical architecture and its building blocks for data-intensive applications.

3.1 The H-SIMD Machine

The H-SIMD machine is designed to aid in the task of programming applications targeting configurable systems and attempting to fully overlap communications with computations. Therefore, ease of programming and high performance can be achieved.

H-SIMD consists of two parts: the host PC and the CSOC array. The former is in charge of task partitioning and is programmed with high-level languages such as C/C++ while the latter is designed to speed up the computation-intensive parts in the applications.

The SIMD architecture exploits the inherent data parallelism in the corresponding algorithms.

3.1.1 H-SIMD Architecture

The H-SIMD control hierarchy is composed of three layers, as shown in Figure 3.1. It comprises the host controller (HC), the FPGA controllers (FCs), and the nano-processor controllers (NPCs). The HC lies in the host machine and controls all the FPGA chips in the SIMD mode. Inside each FPGA, an FC is designed to run all the on-chip NPCs in the SIMD mode as well. The NPCs control the execution of machine-level code. Similar to the approach for PC clusters in [47], an effective instruction set architecture (ISA) needs to be developed at each layer for each application domain. Therefore, a hierarchical instruction set architecture (HISA) is created to facilitate the logical interconnection of the three layers. This allows to logically partition the application into the HC, FC, and NPC layers that can be handled efficiently at run time to balance the pipeline running from the host down to the on-chip nano-processors. Task scheduling and the coarse-grain dataflow control of the application are left to the HC.

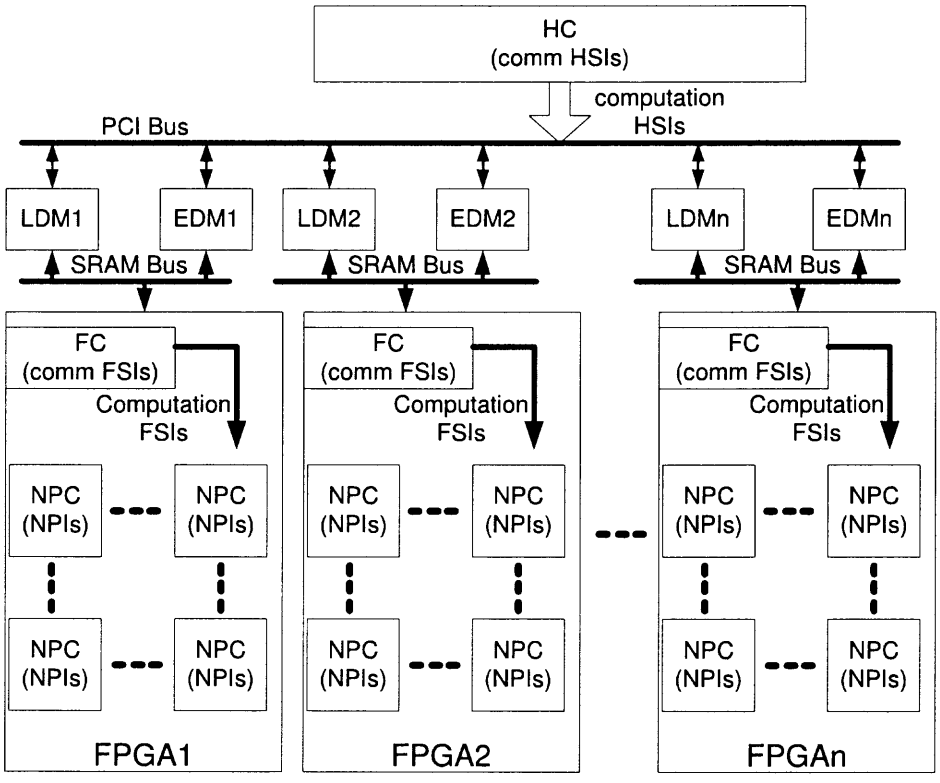
HISA instructions are classified into communication instructions and computation instructions. The former are executed by the local controller while the latter are issued to the lower level. As shown in Figure 3.2, the HC runs the coarse-grain host SIMD instructions (HSIs) which are classified into host-FPGA communication HSIs and time-consuming computation HSIs. The HC executes the communication HSIs only and issues the computation HSIs to the FCs to execute. Inside each FPGA, the FC further decomposes the received computation HSIs into a sequence of medium-grain FPGA SIMD instructions (FSIs). The FC runs them in a manner similar to the HC: executing the communication FSIs and issuing the computation FSIs to the nano-processor array. The NPCs finally decode the received computation FSIs into fine-grain nano-processor instructions (NPIs)

and then sequence them for execution. Additionally, the HC, the FCs, and the NPCs run at different frequencies. The HC is the slowest component while the NPCs are the fastest.

The H-SIMD machine configures one of the FPGA chips as the master FPGA which is responsible for sending an interrupt signal back to the HC once the previously executed HSI has been completed at the FPGA level. Similarly, one NP within each FPGA is configured as the master NP that sends an interrupt signal back to its FC so that a new FSI can be issued. The combination of the HC and FCs, however, is different from the conventional SIMD controller scheme because the former does not actually monitor the execution. Decentralized control in the H-SIMD machine allows the execution of instructions at one layer to be transparent to higher layer(s). On the contrary, centralized control requires the handling of pipeline stalls and exceptions. It is one of the most complex parts in the design and is very hard to be scalable. To summarize, the control flow in the H-SIMD machine is two-way asynchronous instead of the conventional one-way synchronous.

3.1.2 Memory Switching Schemes

The communication overhead between the host and the FPGA cluster is very high due to the nature of the non-preemptive operating system on the host. Based on tests in our laboratory, the one-time interrupt latency for a Windows-XP installed Dell Precision 650 host workstation running the PCI bus at 66MHz is about 5 ms. This penalty is intolerable in high-performance computing because, for example, 60×60 floating-point matrix multiplication takes about 1.3 ms on a single MAC (multiply accumulator) running at 160 MHz (which is within range of current FPGA technology) [64] [65]. If the host frequently intervenes in FPGA operations, the speedup benefits gained from the parallel FPGA implementation can be significantly reduced or even removed. Thus, a design objective of the H-SIMD machine is to hide host communication latencies. Yet, given the system configuration of the host machine, the latency is fixed and cannot be reduced unless a



EDM: execution data memory; LDM: loaded data memory

Figure 3.1 H-SIMD machine architecture.

preemptive operating system or special PCI hardware interface are enlisted. In order to overcome the latency problem, a data prefetching scheme involving memory switching is designed for the H-SIMD machine to overlap communications with computations as much as possible.

The HC-level memory switching scheme is shown in Figure 3.3. The SRAM banks on the FPGA board are organized into two functional memory units: the execution data memory (EDM) and the loaded data memory (LDM). Both the EDMs and LDMs are functionally interchangeable. At one time, the FCs access the EDMs to fetch operands for the execution of received computation HSIs while the LDMs are referenced by the host for the execution of communication HSIs. When the FCs finish their current computation

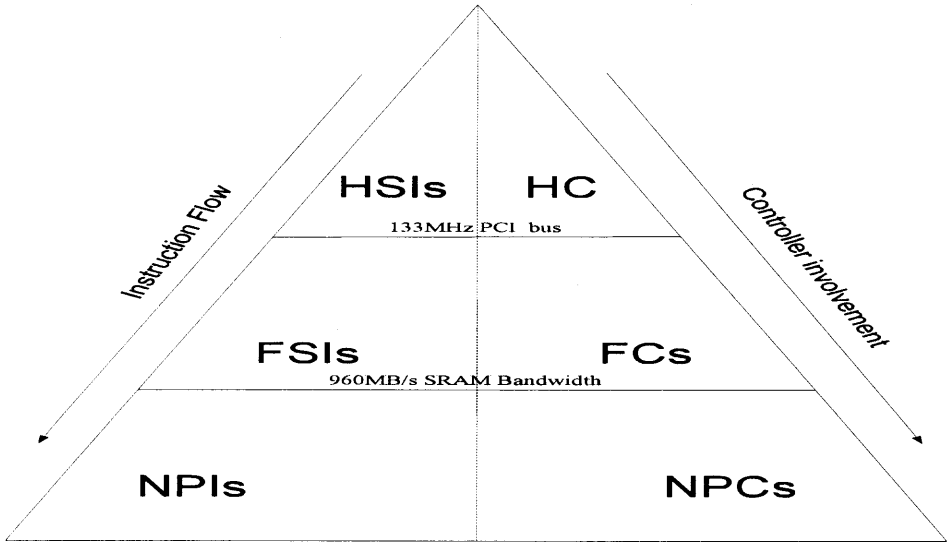


Figure 3.2 The HISA ISA for the H-SIMD architecture.

HSI, they will switch between the EDM and LDM to begin a new iteration. The FC is a finite-state machine responsible for the execution of the computation HSI. The FCs have access to the NP array over a modified LAD (M-LAD) bus. The LAD bus was originally developed by the Annapolis company and used for on-chip memory references [42]. The M-LAD bus controller is changed from the PCI controller to the FCs. The HSI counter is used to calculate the number of finished computation HSIs. The SRAM address generator (SAG) is used to calculate the SRAM load/store addresses for the EDM banks. The FC is pipelined and sequentially traverses the states LL (Load LRFs), IF (Instruction Fetch), ID (Instruction Decode), and EX (Execute). The transition condition from EX to LL is triggered by the master NP's interrupt signal. The interrupt request/response latency is one cycle only as opposed to the tens of thousands of cycles between the host and FPGAs, thus enhancing the H-SIMD's performance.

The nano-processors (NPs) form the execution units of the H-SIMD machine datapath. Their functionality can be customized according to the application. The NPs reside at the lowest layer of the H-SIMD machine hierarchy. Each nano-processor has

two large-sized register files: the load register file (LRF) and the execution register file (ERF) as shown in Figure 3.4. Both register files work in a "memory" switching capacity, similarly to the LDMs and EDMs; i.e., they alternate the execution of FSIs with data loading from the FCs. After the nano-processor finishes data processing with the ERF, its datapath will configure another LRF as an ERF and will then begin a new program flow. At the same time, the just switched-out ERFs will be configured as LRFs to be loaded with new operands from the EDMs controlled by the FCs; these operands will then be ready to be processed. The benefits are obvious here: the "memory" switching scheme involving the register files and the SRAM modules can be used to highly or completely overlap communications with computations. Additionally, the large-sized register file reduces significantly the frequency of nano-processor load/store operations and can get rid of a nano-processor's local data memory. This way, higher performance and the conservation of hardware resources can be achieved.

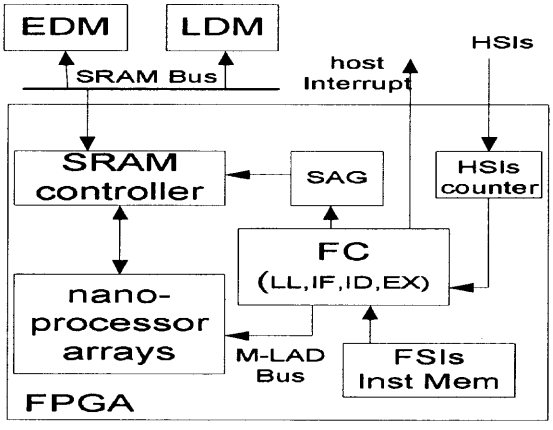


Figure 3.3 HC-level memory switching in H-SIMD.

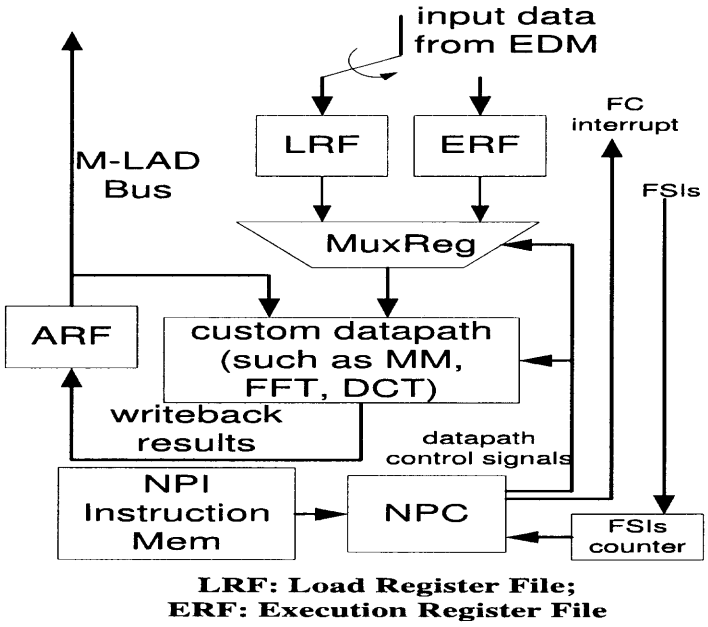


Figure 3.4 Nano-processor datapath and control unit.

3.1.3 H-SIMD Machine Features

The H-SIMD machine's programming capabilities are based on the HISA hierarchy. Much research has shown the big challenges in bridging bandwidth gaps for loosely coupled configurable computing systems [29] [41] [66]. [66] presented a hybrid system architecture model for bridging such gaps in configurable computing systems; it is a heterogeneous system-on-chip platform. [29] demonstrated the usefulness and effectiveness of tightly coupled processing between the microprocessor and the configurable logic array. The H-SIMD machine is designed to meet such challenges in loosely coupled host-CSOC systems.

Based on Amdahl's Law, the overall speedup gained by employing a configurable computing system diminishes if the bandwidth between the host and the configurable computing system becomes a bottleneck. The H-SIMD machine assumes a conventional host and a CSOC array; the latter is implemented with FPGAs. In general, this model

consists of a general-purpose host, a configurable logic array, on-chip memory, off-chip memory, and an interconnection network. At each level, significant speedups are only achieved with configurations where the time to execute the computation instructions is relatively large compared to the overhead required for the communication instructions. To quantify this relative measure, one must consider the following equations:

$$T_{H\_exe} = T_{H\_comp} + (1 - \alpha_H)T_{H\_comm} \tag{3.1}$$

$$T_{F\_exe} = T_{F\_comp} + (1 - \alpha_F)T_{F\_comm} \tag{3.2}$$

$$T_{H\_comp} = T_{F\_exe} \tag{3.3}$$

where T_{H_exe} and T_{F_exe} are the execution times for the host and the FPGAs, respectively; T_{H_comp} and T_{F_comp} are the times spent for computations by the host and the FPGAs, respectively; T_{H_comm} and T_{F_comm} represent the times for communication instructions running on the host and the FPGAs, respectively; α_H and α_F are assumed to represent the overlap factors for such instructions involving the host and the FPGA levels, respectively. The overlap factors indicate the degree of overlap between computations and communications, and their range is $0 \leq \alpha_H \leq 1$ and $0 \leq \alpha_F \leq 1$. The speedup on the H-SIMD machine over a traditional software implementation is expressed as:

$$speedup = \frac{T_{soft}}{T_{H\_exe}} > 1 \tag{3.4}$$

where T_{soft} is the time to execute the function in software running on the host. Equations 3.1, 3.2, 3.3 and 3.4 imply that the following condition must be met for any speedup (> 1) to occur on the H-SIMD machine:

$$\frac{T_{F\_comp}}{T_{soft}} + (1 - \alpha_F)\frac{T_{F\_comm}}{T_{soft}} + (1 - \alpha_H)\frac{T_{H\_comm}}{T_{soft}} < 1 \tag{3.5}$$

The first fraction represents the inverse of the ideal speedup without any overheads; the second and third terms are indicative of the percentage of communication overheads that diminish the speedup. The sum of the three fractions is indicative of the overall speedup: the smaller this sum, the larger the overall speedup for a particular application on the H-SIMD machine. The potential overlaps discussed earlier for program execution on the H-SIMD machine can improve the overall speedup as confirmed by Amdahl's Law: if an enhancement is only usable for a fraction of an application, we cannot expect a speedup by more than the reciprocal of 1 minus that fraction [67].

3.2 Size-Adjustable Register File Design

The H-SIMD machine is implemented on the Xilinx Virtex-II FPGAs in the target Annapolis Wildstar II FPGA board [42]. The number of nano-processors (NPs) in the H-SIMD machine is determined each time by the available FPGA resources for the running application. Each NP's datapath is comprised of the register file, custom datapath, and data memory I/O interface. In a conventional way, the register file is designed with HDL and mapped onto FPGA LUTs. A 32-bit register, however, is inefficiently mapped onto LUTs. Test results are shown in Table 3.1. 0.5% of the LUT resources are used for 32 registers. 29% of the LUTs are used by the 512-register configuration, and thus it is impossible to deploy more than three such register files in one FPGA chip. In the H-SIMD machine, however, a large-sized register file is preferred in order to reduce the number of load/store operations and simultaneously fit many functional units.

Here, a size-adjustable register file is designed with even less consumption of LUT resources. In fact, the number of registers is scalable ranging from 16 to 512. This range meets the requirements of various applications. This design exploits the dual-port block SelectRAMs and the DCM inside the platform FPGA upon which the NP register file is built. In the Xilinx Virtex-II FPGA devices [28], there are 144 18Kbit dual-port BlockRAM (BRAM) cells which are customized as coarse-grain components. The designed register file

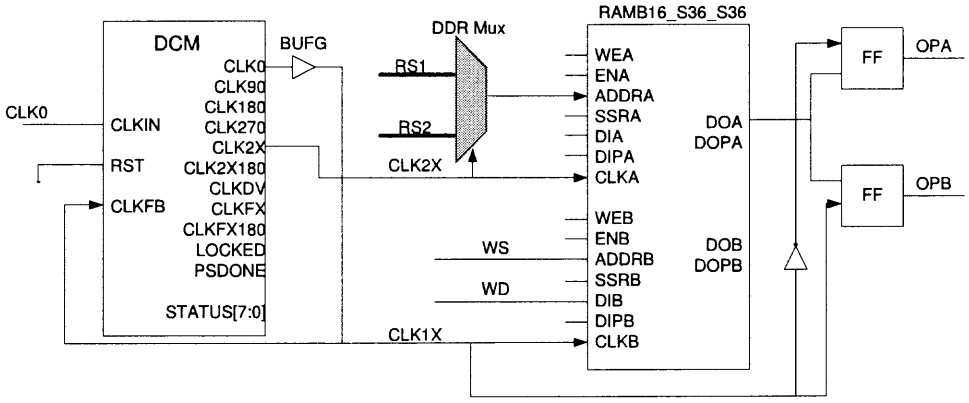


Figure 3.5 Dual-port BRAM-based size-adjustable register file.

diagram is shown in Figure 3.5. One port of the BRAM is used as the writeback port from the NP datapath; the other is used as the output to furnish two operands to the FPU in each cycle. Generally, one port of the BRAM can load one addressed item into the output register in each cycle. But the FPU input requires two operands in each cycle. In order to solve this problem, the register file design takes advantage of the double-speed synchronous clock from the DCM. Two synchronous clocks are used to drive each port. The clock for driving the output operands to the FPU is twice as fast as the one for writing back data which is also identical to the system clock. Once the serialized operands are available on the output port, they are split into two operand streams for datapath execution. With this register file design, one data result from the datapath can be written into the register file and two operands can be read out within one system cycle. Additionally, the write mode of the BRAM cell for the register file is configured as "Write First" (or "Read after Write"), that is, the write data is loaded simultaneously into the output port as well as written into the addressed storage cell. If one port attempts a read of a memory cell while the other one simultaneously writes into that cell, and the clocks of the two ports violate the clock-to-clock setup requirement, then a location conflict occurs and the following policy takes effect [28]:

- The write succeeds.
- The data out on the writing port accurately reflects the data written.
- The data out on the reading port is invalid.
- Conflicts do not cause any physical damage.

The above conflict resolution may result in a Read-After-Write (RAW) data hazard, which has been taken care of by the pipeline forward unit. As shown in Table 3.1, the dual-clock register file design results in a large-sized register file with the smallest LUT consumption.

Table 3.1 FPGA Resource Consumption for Different Types of Register File Designs

	Size	**BlockRAM**	**LUTs***
Case 1 (conventional)	32	0	392(0.5%)
Case 2 (conventional)	512	0	19663(29%)
Case 3 (Dual-Clock)	512	1	80(0.1%)

*Total LUTs: 67584

CHAPTER 4

CASE STUDIES ON THE H-SIMD MACHINE

Case studies are shown in this chapter for the purpose of performance evaluation. The implementation and related experimental results are described for matrix multiplication (MM), 2-dimensional fast Fourier transform (2D FFT), and 2-dimensional discrete cosine transform (2D DCT). These applications are representative of data-intensive computations and are used widely in engineering and science.

4.1 Example 1: Matrix Multiplication

Matrix multiplication (MM) is used widely in scientific and engineering computations where matrix operations are rich [68]. In this section, the development of MM on the H-SIMD machine is presented.

4.1.1 HSIs, FSIs, and NPIs for MM [1]

The HC is programmed using the host API functions for the FPGA board. They are to set up the board, configure the FPGAs, reference the on-board/on-chip memory resources, and handle interrupts [42]. The tailoring of the HSIs for the block-based MM algorithm is presented here. Assume the problem $C = A * B$, where A, B, and C are $N \times N$ square matrices. When N becomes large, block matrix multiplication is used to divide the matrix into smaller blocks to exploit data reusability. Then, the multiplication of these smaller matrix blocks is performed on the FPGA array. In the H-SIMD machine, only a single FPGA or NP is employed to multiply and accumulate the results of one block of the product matrix at the HC and FC levels, respectively. Coarse-grain workloads can keep the NPs busy on MM computations while the HC and FCs load operands into the FPGAs and NPs sequentially. This simplifies the design of the hierarchical architecture and eliminates the need for inter-FPGA and inter-NP communications at the expense of extra memory

reference time. According to the H-SIMD architecture, the HC issues $N_h \times N_h$ sub-matrix blocks to all the FPGAs to multiply. N_h is the block matrix size for the HSIs. Three HSIs are designed:

- $host\_matrix\_load(i, S_{LDM}, N_h)$: Through the PCI bus, this HSI will load an $N_h \times N_h$ matrix block into the LDM of FPGA i with the starting address S_{LDM} in the host memory (host-based DMA control is applied).
- $host\_matrix\_store(i, S_{LDM}, N_h)$: The computation results in the LDM of FPGA i can be retrieved by the host through the PCI bus when the computation is done. $host\_matrix\_load/store$ are communication HSIs executed on the host.
- $host\_matrix\_mul\_accum(H_A, H_B, H_C, N_h)$: For matrix multiplication of size $N_h \times N_h$, H_A, H_B and H_C are the starting addresses of source matrix A, source matrix B and product accumulation matrix C, respectively. This computation HSI is coded in 32 bits, issued by the HC and executed by the FCs.

The FC is in charge of executing the computation HSIs. It will decompose the operation corresponding to $host\_matrix\_mul\_accum$ for size $N_h \times N_h$ into FSIs for size $N_f \times N_f$, where N_f is the sub-block matrix size for the FSIs. Enlisted is the same block matrix multiplication algorithm as the one for the HC. The $host\_matrix\_mul\_accum$ code is pre-programmed in the form of FSIs and is stored into the FC instruction memory. The FSIs are 32-bit instructions with mnemonics as follows:

- $FPGA\_matrix\_load(i, S_{LRF}, N_f)$: the FC will execute this instruction by loading the LRF of NP i with a matrix of size $N_f \times N_f$. S_{LRF} is the starting address in the EDM.
- $FPGA\_matrix\_store(i, S_{ARF}, N_f)$: The NP computation results are stored into the accumulation register file (ARF) and retrieved into the FPGA's EDM at starting address S_{ARF} when the accumulation of the partial products is done. $FPGA\_matrix\_load/store$ are communication FSIs executed by the FCs.
- $FPGA\_matrix\_mul\_accum(F_a, F_b, F_c, N_f)$: For matrix multiplication of size $N_f \times N_f$, F_a, F_b and F_c are the starting addresses of source matrix a, source matrix b and product accumulation matrix c, respectively. This computation FSI is issued by the FCs and executed by the NPCs.

The NPIs are designed for the execution of the computation FSI $FPGA\_matrix\_mul\_accum$. The code for $FPGA\_matrix\_mul\_accum$ is pre-programmed by the NPIs and stored into the NPC instruction memory. There is only one NPI to be implemented for MM, corresponding to floating-point multiply accumulation: $NP\_MAC(R_{s1}, R_{s2}, R_d)$, where R_{s1}, R_{s2}, and R_d are registers for the function $R_d = R_{s1} * R_{s2} + R_d$. The NPI code for the computation FSIs needs to be scheduled carefully to avoid data hazards. They occur when operands are delayed in the addition pipeline with latency L_{adder}. Thus, the condition to avoid data hazards is $N_f^2 > L_{adder}$.

4.1.2 General-purpose Nano-processor ISA

In order to exploit the configurability of the FPGAs, additional general-purpose NPIs are developed to augment the general-purpose scientific computations on the H-SIMD machine. Their implementation is based on a tradeoff between application demands and FPGA resource consumption. All these instructions are listed in Table 4.1.

Arithmetic Instructions The arithmetic instructions $NP\_MAC/ADD/SUB/PSUB/MUL/DIV$ include MAC, addition, subtraction, absolute value subtraction, multiplication, and division. They are executed on the SIMD NP array. Division is very expensive not only in execution cycles but also in FPGA resource consumption. If the division operation is very rare in the application algorithms, only one NP is configured with a divider circuit.

Routing Instructions The SIMD nano-processor array needs to communicate values between the NPs. Based on the NEWS grid interconnection [46], the data needed by an NP can be routed from its north/east/west/south neighbors by using an $NR/ER/WR/SR$ instruction.

Mask Instruction The same operation is normally applied to all the NPs simultaneously in the SIMD architecture. There are ways to nullify the effects of an instruction on the

selected NPs. The mask instruction can be used to load a masking pattern into each NP's mask bit in order to enable/disable the NP.

Branch Instruction There are two branch instructions: conditional and unconditional. BR_P is a conditional branch instruction which puts a new instruction address into the Program Counter (PC) if the content of the source register is greater than zero. JMP is an unconditional branch instruction that puts a new value directly into the PC.

Compare Instruction It is a common operation to compare the values of two operands for magnitude. Conventionally, this operation will be done with three other instructions. The $COMP$ instruction is designed as a general-purpose NPI to speed up the comparison between the two operands.

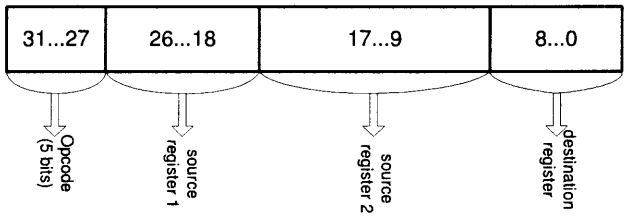
General-Purpose NP Instruction Format There are 32 bits in the instruction register which is exactly the same number as the length of the 32-bit data output from the Xilinx Virtex-II dual-port RAM. The fields in the instruction encoding are shown in Figure 4.1. The instruction opcode uses bits 31 to 27. In Figure 4.1(a), bits 26 to 18, 17 to 9 and 8 to 0 are used to represent the source register r_{s1}, source register r_{s2}, and destination register r_d, respectively, for MAC/arithmetic/compare/routing instructions. In Figure 4.1(b), bits 26 to 18 and bits 17 to 5 are used for the base register and the new PC instruction address, respectively. The masking pattern in Figure 4.1(c) is included in bits 17 to 0, corresponding to NP17 to NP0. If the mask bit is '1', the corresponding NP is enabled; otherwise, it is disabled. Bits 26 to 18 are reserved for future extension.

4.1.3 Assembler Design and Data Initialization

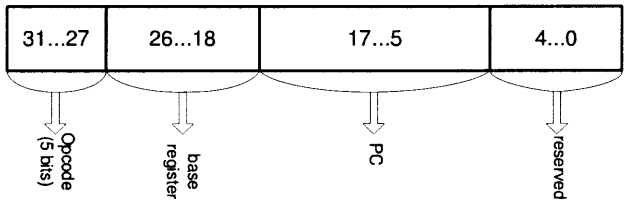
The assembler and data initialization software are developed using the C programming language. The assembler is responsible for the conversion of mnemonics into machine code for SIMD machine execution. The data initialization software can greatly facilitate

Table 4.1 General-Purpose NPIs

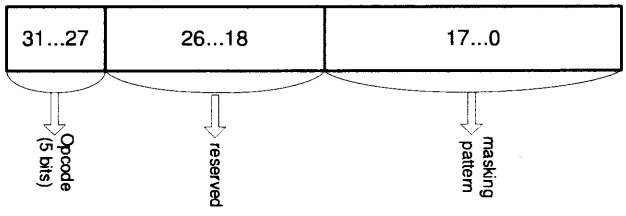
Mnemonic	Description of Operations
$NP\_MAC\ r_d,\ r_{s1},\ r_{s2}$	Accumulate the product of r_{s1} and r_{s2} into r_d
$NP\_ADD\ r_d,\ r_{s1},\ r_{s2}$	Add r_{s1} and r_{s2} and put result into r_d
$NP\_SUB\ r_d,\ r_{s1},\ r_{s2}$	Subtract r_{s2} from r_{s1} and put result into r_d
$NP\_MUL\ r_d,\ r_{s1},\ r_{s2}$	Multiply r_{s1} with r_{s2} and put result into r_d
$NP\_DIV\ r_d,\ r_{s1},\ r_{s2}$	Divide r_{s1} by r_{s2} and put result into r_d
$NP\_PSUB\ r_d,\ r_{s1},\ r_{s2}$	Subtract the absolute value of r_{s2} from the absolute value of r_{s1} and put the absolute value of the result into r_d
$NR/ER/WR/SR\ r_d,\ r_s$	Route r_s to the north/east/west/south and store into r_d of the receiving PE
$LM\ mask\_pattern$	Load $mask\_pattern$ to disable or enable the corresponding PEs for the coming instructions
$BR\_P\ r_s,\ mem(Addr)$	Branch to a new PC specified by the instruction memory address if the content in r_s is greater than zero
$JMP\ mem(Addr)$	Unconditionally jump to a new PC specified by the instruction memory address
$COMP\ r_d,\ r_{s1},\ r_{s2}$	Compare the absolute value of r_{s1} and r_{s2} and store the bigger value into r_d



(a) MAC/Arithmetic/Compare/Routing instruction format



(b) Branch/Jump instruction format



(c) Mask instruction format

Figure 4.1 General-purpose NP instruction format.

application development and output the data initialization file which is consistent with the Xilinx Core Generator format. Some other utilities for register file initialization are also developed to bridge the gap between Xilinx EDA tools and the application mapping file in the H-SIMD machine. All of these form the preliminary software development platform for the H-SIMD machine.

4.1.4 Task Partitioning Analysis for Matrix Multiplication

The H-SIMD machine targets applications of high data parallelism. The input matrices for MM are pre-partitioned into blocks of size $N_h \times N_h$, where N_h is the matrix size at the HC layer. The FCs further decompose $N_h \times N_h$ MMs into $N_f \times N_f$ MMs which can be

run on the nano-processor arrays in parallel. The block-based MM algorithm for dense matrices is employed. After issuing $\lceil N_h/N_f \rceil$ FSIs for $N_f \times N_f$ MMs, the FCs execute $FPGA\_matrix\_store$ to retrieve the final products from the LRFs and store them back into the SRAM EDM blocks. Besides issuing the FSIs, the FC also keeps loading data from the EDMs into the LRFs. Once the $N_f \times N_f$ MMs are done on each nano-processor, the FCs respond to interrupt requests by first initiating another $N_f \times N_f$ MM for the nano-processors and then loading new data into the LRFs. All these steps are done within the FPGAs without host intervention. This greatly speeds up nano-processor execution, and the interaction between the FC and the nano-processor arrays.

The bandwidth of the communication channels in the H-SIMD machine varies greatly. Basically, there are two interfaces in the H-SIMD machine: a PCI bus of bandwidth B_{pci} between the host and the FPGAs; the SRAM bus of bandwidth B_{sram} between the off-chip memory and the on-chip nano-processor array. The HSI parameter N_h is chosen in such a manner that the execution time $T_{host\_compute}$ of the HSI computation instruction $host\_matrix\_mul\_accum$ is greater than $T_{host\_i/o}$ which is the sum of the execution times T_{HSI_comm} of all the communication HSIs $(host\_matrix\_load/store)$ and the master FPGA interrupt overhead T_{fpga_int}. If so, the communication and interrupt overheads can be hidden. Assume that there are q FPGAs of p nano-processors each. Specifically, the following lower/upper bounds should hold for matrix multiplication:

$$T_{host\_compute} > \tau * N_h^3/p \tag{4.1}$$

$$T_{host\_i/o} < T_{HSI\_comm} * q + T_{fpga\_int} = 4 * b * N_h^2/B_{pci} * q + T_{fpga\_int} \tag{4.2}$$

where τ is the nano-processor cycle time and b is the width in bits of each I/O reference. Simulation results in Figure 4.2 show that the HSI computation and I/O communication times vary with N_h, p, and q, for $b = 64$ and $\tau = 7ns$. With increases in the block size for the HSIs, the computation time grows in a cubic manner and yet the I/O communication time grows only quadratically, which is exploited by the H-SIMD machine. This means

that the host may load several LDMs sequentially while all the FPGAs run the issued HSI in parallel.

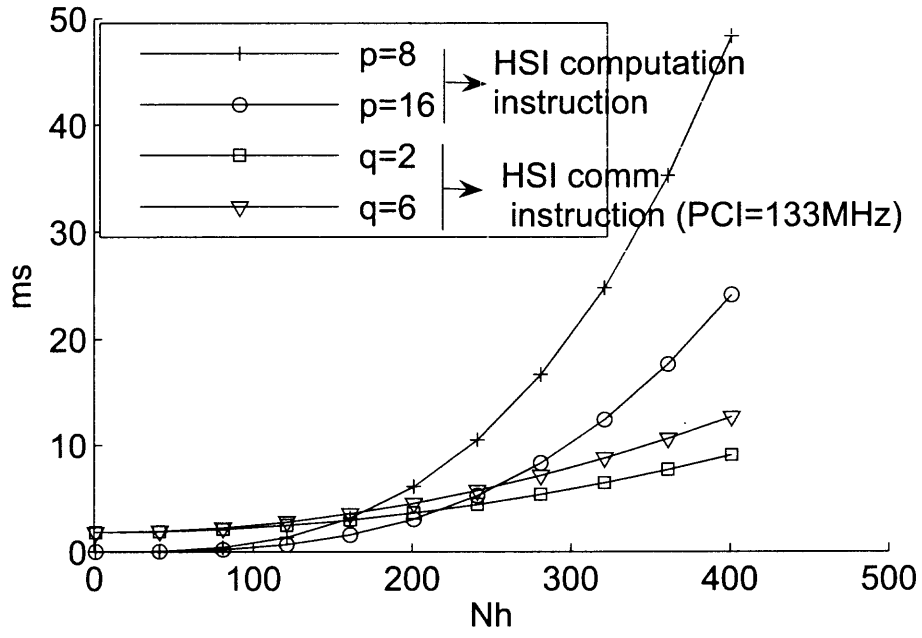


Figure 4.2 Execution times of the computation and communication HSIs as a function of N_h, p and q.

For FC-level $N_f \times N_f$ block MM, tweaking N_f can overlap the execution time $T_{FPGA\_compute}$ of the FSI computation instruction $FPGA\_matrix\_mul\_accum$ with the sum $T_{FPGA\_i/o}$ of the execution times $T_{NP\_i/o}$ of all the communication FSIs and the NP interrupt overheads T_{NP_int}. The following upper/lower bounds should hold:

$$T_{FPGA\_compute} > \tau * N_f^3 \tag{4.3}$$

$$\begin{aligned} T_{FPGA\_i/o} &< T_{NP\_i/o} * p + T_{NP\_int} \\ &= 4 * b * N_f^2/(B_{sram} * N_{bank}) * p + T_{NP\_int} \end{aligned} \tag{4.4}$$

N_{bank} is the number of available SRAM banks in each FPGA. Simulation results in Figure 4.3 show that the computation FSI takes more execution time than the

communication FSIs with an increase in N_f. More SRAM banks can provide a higher aggregate bandwidth to reduce the execution times of the communication FSIs. Using the above analysis of the execution time, the design space is explored for a lower bound on N_h and N_f, respectively. On the other hand, the capacity of the off-chip and on-chip memories defines the upper bounds on N_h and N_f. For each FPGA for MM operations: $4 * r * N_h^2 * b < C_{sram} * N_{bank}$ and $4 * r * N_f^2 * b < C_{on-chip}$, where C_{sram} represents the capacity of one on-board SRAM bank; $C_{on-chip}$ represents the on-chip memory capacity of one FPGA; r stands for the redundancy of the memory system, so $r = 2$ for the memory switching scheme. In summary, the upper bounds of N_h and N_f are $\sqrt{C_{SRAM} * N_{bank}/(8 * b)}$ and $\sqrt{C_{on-chip}/(8 * b)}$, respectively.

4.1.5 Matrix Multiplication: Implementation and Test Results

The H-SIMD machine was implemented on the Annapolis Wildstar II PCI board containing two Xilinx Virtex-II 6000 FPGAs [27]. The Quixilica FPU [69] is employed to build up the NP's floating-point MAC. Table 4.2 gives the characteristics of the Quixilica FPU and MAC for the 64-bit IEEE double-precision format. In the design environment, ModelSim5.8 and ISE6.2 are enlisted as development tools. The Virtex-II 6000 can hold up to 16 NPs running at 148MHz. Broadcasting the FSIs to the nano-processor array is pipelined so that the critical path lies in the MAC datapath. The 1024×1024 MM operation is tested. The block size N_f of the FSIs is set to 8. The test results break down into computation HSIs, host interrupt overhead, PCI reference time, and initialization and NP interrupt overhead, as shown in Figure 4.4. The performance of the H-SIMD machine depends on the block size N_h. When N_h is set to 64, the frequent interrupt requests to the host contribute to the performance penalty. When N_h is set to 128, the computation time of the coarse-grain HSI does not increase long enough to overlap the sum of the host interrupt overhead and the PCI sequential reference overhead. If N_h is set to 512, there is a long enough computation time to overlap the host interrupt. However, the memory switching

Table 4.2 Characteristics of the Quixilica FPU and H-SIMD MAC

	fpAdder	fpMultiplier	MAC
Pipeline Stages	12	11	24
Slice Usages	815	923	1802
Clock Speed(MHz)	153	150	148

scheme between the EDMs and LDMs does not work effectively because of the limited capacity of the SRAM banks, which results in penalties from both host interrupts and PCI references. If N_h is set to 256, the H-SIMD pipeline is balanced along the hierarchy such that the total execution time is very close to the peak performance $2 * p * q * freq$, where all the nano-processors work in parallel. The H-SIMD machine can sustain 9.1 GFLOPS, which is 95% of the peak performance. The execution overhead on the H-SIMD machine comes from LDM and LRF initialization, and nano-processor interrupts to the FCs.

For an arbitrary size of square MM operations, a padding technique is employed to align the size of the input matrices to multiples of N_f because $FPGA\_matrix\_mul\_accum$ works on $N_f \times N_f$ matrices. N_f is set to 8 during the test. Let A and B be square matrices of size $N \times N$. If N is not a multiple of eight, then both the A and B input matrices are padded up to the nearest multiples of eight by the ceiling function. The padded zeros will definitely increase the H-SIMD's computation overhead and lower its performance. Table 4.3 presents test results for different cases. For matrices of size less than 512, the H-SIMD machine is not fully exploited and does not sustain high performance. For the large matrix ($N > 512$), the H-SIMD machine with two FPGAs can achieve about 8.9 GFLOPS on average. In fact, the H-SIMD machine can be built with multiple FPGAs because no inter-FPGA communications are needed. Figure 4.5 shows the relationship between the execution time of 2048×2048 MM and the number q of FPGAs. There exists a saturation point, beyond which the number of FPGAs does not affect the performance

Table 4.3 Execution Time of MM for Various Test Cases

Matrix size	H-SIMD machine(ms)	GFLOPS
200	7	2.28
397	18	6.952
601	47	8.683
999	225	8.849
2001	1720	9.039
3999	13882	9.027

significantly. For the case study of block matrix multiplication, seven Virtex II 6000 FPGAs can be enlisted to achieve 31.85 GFLOPS for MM based on the 64-bit IEEE floating-point format.

Table 4.4 compares the performance of the H-SIMD machine with that of previous works on FPGA-based floating-point matrix multiplication [70] [71]. Their designs were implemented on Virtex II Pro125 FPGAS (55,616 slices) as opposed to the Virtex II 6000 (33,792 slices). The H-SIMD performance is scaled to match the Virtex II Pro125. It is estimated that 26 NPs can fit into one Virtex II Pro125 running at 180MHz and can achieve a peak performance of 9.36GFLOPS. The H-SIMD running frequency can be further increased if optimized MACs are used. [70] [71] presented systolic arrays to achieve 8.3 GFLOPS and 15.6 GFLOPS on a single Xilinx Virtex II Pro XC2VP125, respectively. However, the H-SIMD machine can be used as a computing accelerator for the workstation, thus providing tremendous flexibility [68]. The systolic array approach does not fit well into this paradigm because of the specialized approach, significant interrupt overhead, FPGA configuration overheads, and large size of configuration files.

Table 4.4 Performance Comparison between H-SIMD and Other Works

	H-SIMD	[70]	[71]
Frequency	180	500	200
Number of PEs	26	24	39
GFLOPS	9.36	8.3	15.6
Hide interrupt overhead	Yes	No	No
configuration file size(MB/100 cases)	5	500	500

4.2 Example 2: 2D Fast Fourier Transform

The two-dimensional fast Fourier transform (2D FFT) is a popular algorithm used widely in signal processing. It is taken as an example to show the effectiveness of the H-SIMD machine.

4.2.1 HISA for 2D FFT [2]

FFT is an efficient algorithm to compute the discrete Fourier transform (DFT). FFT can reduce the computation complexity of the N-point DFT from $O(N^2)$ to $O(NlogN)$. For a discrete-time sequence $x(n)$, where $n = 0, , N-1$, its 1D DFT is defined as [72]:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-jnk2\pi/N} k = 0, \ldots, N-1 \tag{4.5}$$

where the twiddle factors $e^{-jnk2\pi/N}$ can be pre-computed and stored in the on-chip memory of the FPGAs. The 2D FFT can be carried out by applying 1D FFT in parallel to all the rows and then in parallel to all the columns, or vice versa. As already mentioned, the H-SIMD machine can fit a wide variety of applications characterized by data parallelism. Three HSIs at the topmost layer are needed for 2D FFT as follows:

- $host\_fft2\_load(LDM_i, N_h)$: This HSI will load each FPGA's LDM_i memory sequentially via the host PCI bus (host-based DMA control is used);
- $host\_fft2\_retrieve(LDM_i, N_h)$: The final results are stored in the switched-out LDM_i and can be retrieved by the host sequentially using DMA through the PCI bus when the last issued HSI is done. $host\_fft2\_load/retrieve$ are communication HSIs for 2D FFT;
- $host\_fft2(H_A, H_B, N_h)$: H_A is the input matrix of size $N_h \times N_h$ and $H_B = fft2(H_A)$. This is the computation HSI issued for the FPGA executions.

The FSIs in the middle of the HISA hierarchy run on the FPGA hardware. Assume that there are q FPGAs with p nano-processors each. There are four FSIs:

- $FPGA\_fft\_load(F_a, LRF_i, N_h)$: There are two register files in nano-processor i, LRF_i and ERF_i. FC will execute this instruction by loading each nano-processor's LRF with a row vector of size N_h from matrix F_a of size $N_h/q \times N_h$. F_a is the starting addresses of the input;
- $FPGA\_fft(F_a, F_b, N_h)$: matrix F_b of size $N_h/q \times N_h$ is produced by applying 1D FFT to each row of the input matrix F_a;
- $FPGA\_fft\_transpose(F_a, F_b, N_h)$: matrix F_a of size $N_h/q \times N_h$ is transposed and stored into the FPGA's communication data memory (CDM) via the high-speed SRAM interface with source starting address F_a and destination starting address F_b. CDM stores the nano-processor computation results which can be transmitted via the 8Gbytes/s LVDS connection to other FPGA chips;
- $FPGA\_lvds\_comm(F_a, N_h)$: matrix F_a of size $N_h/q \times N_h$ within one FPGA is transmitted by a LVDS connection to neighbor FPGAs.

The NPIs constitute three instructions: $NP\_load(R_a, N_h)$, $NP\_retrieve(R_a, N_h)$, and 1D N_h-point FFT, $NP\_fft(R_a, N_h)$. The last instruction can be produced by the Annapolis CoreFire libraries [73]. The customized FFT nano-processor is shown in Figure 4.6. The input data in the experiments is a vector R_a of N_h = 64, 256, or 1024-point values represented as 16-bit complex or IEEE754 single-precision floating-point numbers. R_a is the starting address of the vector residing in the ERF memory of each nano-processor.

The input matrices for 2D FFT have size $N_h \times N_h$, where $N_h \leq 1024$ in the current implementation due to the recent CoreFire6.2.03 release by Annapolis Micro Systems and

the capacity constraints of the on-board SRAMs. Each FPGA is assigned N_h/q vectors of size N_h, where q is the number of FPGAs in the H-SIMD machine. For each FPGA chip, the FC issues $FPGA\_fft(F_a, F_b, N_h)$ to carry out N_h/q N_h-point 1D FFTs which can be run on the nano-processor arrays in parallel. After finishing the FFTs of the first dimension vectors, the FCs execute $FPGA\_fft\_transpose(F_a, F_b, N_h)$ to transpose the results and write back into the CDMs. Then, $FPGA\_lvds\_comm(F_b, N_h)$ is invoked to broadcast the results from CDM to the neighbor FPGAs' EDMs in the ring network. The inter-FPGA communication cost is negligible due to the high-speed 8Gbytes/s LVDS connections between FPGAs. After that, FC issues another $FPGA\_fft(F_a, F_b, N_h)$ to transform the second dimension vectors. The FCs also respond to nano-processor interrupt requests by first initiating a new FFT execution for the nano-processors and then loading new data into the LRFs. All these steps are done within the FPGAs without host intervention.

4.2.2 Task Partitioning in 2D FFT: Performance Analysis

The bandwidth of the communication channels in the H-SIMD machine varies greatly. Basically, there are three interfaces in H-SIMD: the PCI bus with a bandwidth B_{pci} of $133MHz \times 64$ bits, the SRAM bus with a bandwidth B_{sram} of 960Mbytes/s, and the LVDS inter-FPGA connections with a bandwidth B_{lvds} of 8Gbytes/s. In the case of 2D FFT on an $N_h \times N_h$ matrix, assume that $N_h/q \times N_h$ sub-matrices of an $N_h \times N_h$ matrix are uniformly distributed among the q FPGAs, with p nano-processors each. The $host\_fft2(H_A, H_B, N_h)$ HSI consists of three operations on the input matrix H_A, i.e., the F_a sub-matrix of size $N_h/q \times N_h$ from matrix H_A will go through 1D FFTs on its rows and transposed columns, sub-matrix transpose and inter-FPGA communications. The execution time of the three operations is denoted as T_{1D_fft}, T_{trans}, and $T_{fpga-fpga}$, respectively. The total computation time for $host\_fft2(H_A, H_B, N_h)$ is $T_{compute}(host\_fft2) = 2 * N_h/(q * p) * T_{1D\_fft} + T_{trans} + T_{fpga-fpga} + T_{initialize\_fft}$. On the other hand, the communication time T_{i/o_pci} of $host\_fft2\_load/retrieve$

depends on the available PCI I/O bandwidth and the interrupt latency T_{host_int}, i.e., $T_{i/o\_pci} = 2*b*N_h^2/B_{pci}+T_{host\_int}$, where b is the width in bits of each transaction. Pipeline balancing is guaranteed if $T_{compute}$ is greater than or equal to T_{i/o_pci}, and the computations fully overlap I/O communications. Specifically, the following approximations hold for 2D FFT:

$$T_{initialize\_fft} > 3 * N_h * \tau \tag{4.6}$$

$$T_{1D\_fft} > N_h * \tau \tag{4.7}$$

$$T_{trans} > 2 * N_h * N_h * \tau/q \tag{4.8}$$

$$T_{fpga-fpga} > N_h * N_h * (q-1) * b/(q * B_{lvds}) \tag{4.9}$$

Thus,

$$T_{i/o\_pci} = 2 * b * N_h^2/B_{pci} + T_{host\_int}, \tag{4.10}$$

$$\begin{aligned} T_{compute}(host\_fft2) &> 2 * N_h * T_{1D\_fft}/(q * p) + T_{trans} + T_{fpga-fpga} \\ &\quad + T_{initialize\_fft} \\ &= (2 * N_h * N_h/(p * q) + 2N_h * N_h/q + 3 * N_h) * \tau \\ &\quad + N_h * N_h * (q-1) * b/(q * B_{lvds}) \end{aligned} \tag{4.11}$$

where τ is the nano-processor cycle time. For the configuration $p = 6$, $B_{pci} = 133MHz \times 64bits, T_{host\_int} = 3.5ms, b = 32$ and $\tau = 11ns$ on 16-bit complex numbers, the simulation results in Figure 4.7(a) show that the computation time varies with the size N_h of the matrix and the number q of FPGAs. The communication time is independent of the number of FPGAs because the data traffic on the PCI bus, given a problem size N_h, is fixed and the input vectors from the host are uniformly distributed among all the FPGAs. With increases in the matrix size for 2D FFT, the computation time grows faster than the I/O communication time, which is exploited by H-SIMD to implement host-level memory switching (the FPGAs waste no time to wait for data loads/retrievals). This

condition is easier met for applications with high computation load (e.g., floating-point matrix multiplication) rather than with low computation load (e.g., integer FFT). In fact, a full overlap is achieved when the input matrix has size greater than 896 and there are two FPGAs. If more FPGAs are enlisted, H-SIMD is difficult to fully overlap communications with computations for 2D FFT on 16-bit complex numbers.

At the FC level, $FPGA\_fft(F_a, F_b, N_h)$ is carried out on all the nano-processors while vectors of size N_h are loaded into LRF at the same time. For effective nano-processor-level memory switching, the execution time T_{NPI_fft} of the 1D N_h-point FFT $NPI\_fft(R_a, N_h)$ must be greater than p times the register file reference time T_{NP_load}, i.e., $T_{NPI\_fft} > p * T_{NP\_load}$, where $T_{NPI\_fft} = N_h * \tau$ and $T_{NP\_load} = N_h/(B_{sram} * N_{bank})$. N_{bank} is the number of the SRAM banks available to each FPGA. In fact, this condition can be easily met when $B_{sram} = 960Mbytes/s, p = 6, \tau = 11ns$ and $N_{bank} = 6$, as shown in Figure 4.7(b). If computations do not overlap fully communications at the FC level, more SRAM banks can be employed to provide higher aggregate bandwidth.

4.2.3 Implementation Results for 2D FFT

The 2D FFT is also implemented on a host PC workstation and an Annapolis FPGA board containing two Xilinx Virtex-II 6000 FPGAs. The host is given a program of 2D transformation $H_{fft2}(N_h, N_h) = fft2(H_A)$, where H_A and H_{fft2} are matrices of size $N_h \times N_h$. N_h/q vectors of size N_h are assigned to each FPGA for 1D N_h-point FFT, where N_h is the size of the HC-level matrix. Two kinds of nano-processors are implemented on the H-SIMD machine with 16-bit complex numbers and IEEE754 single-precision floating-point numbers, respectively. The CoreFire Design Suite is a dataflow-based application package that provides end-users with a large collection of cores, including host access cores. A 32-bit counter is embedded inside the FPGAs to count the elapsed cycles between the initiation of the first HSI and the completion of the last HSI. After Xilinx ISE6.2 Place &

Route, six 89MHz nano-processors can fit in one FPGA for 16-bit complex numbers while only one 80MHz nano-processor for single-precision floating-point numbers. 2D FFTs on 64×64, 256×256 and 1024×1024 matrices were tested. The results were compared with results produced by Matlab. The H-SIMD machine has precision 10^{-5} if the Matlab results are assumed as the benchmark. The timing results break down into inter-FPGA communication, interrupt overhead, PCI reference time and 1D FFT/transpose computation time, as shown in Figure 4.8. The performance depends heavily on the interrupt overhead, 1D FFT/transpose computation time and host-PCI reference time. All the other factors contribute little to the total execution time, which is desirable for the H-SIMD design. When N_h is set to 64, the frequent interrupt requests to the host contribute excessively to the performance penalty. When N_h is set to 256, the computation time does not increase long enough to overlap fully the sum of the costs for host interrupts and PCI-SRAM memory sequential references. If N_h is set to 1024, the designed overlap is so good that the interrupt and communication overheads are hidden, and all the nano-processors work in parallel.

Only arithmetic operations are counted for the input 16-bit complex numbers. The million-operations-per-second (MOPS) metric is used for the purpose of benchmarking. The operation count is consistent with the one in [74]. For complex-data and real-data FFTs, the number of operations is $5 * N * log_2 N$ and $2.5 * N * log_2 N$, respectively. Based on the same 2D FFT problem, results are compared in Table 4.5 for H-SIMD MOPS with 16-bit complex numbers and MFLOPS for IEEE 754 single-precision numbers and the performance of a 2.8GHz Xeon processor as presented in [75]. The H-SIMD machine suffers a great deal of performance loss due to frequent host interventions when the application does not have enough parallelism to exploit. However, it can outperform the powerful Xeon processor when applications show enough parallelism. This proves that H-SIMD fits well the data-intensive applications.

A cost-performance analysis of the H-SIMD machine and a Xeon processor is in order now. The ten million system gates in the Virtex II FPGA consume about 250 million

Table 4.5 Performance Comparison of the H-SIMD Machine and a 2.8GHz Xeon Workstation for 2D FFT

Matrix size	H-SIMD on 16-bit complex numbers (MOPS)	H-SIMD on IEEE754 single precision real numbers (MFLOPS)	2.8 GHz Xeon on IEEE754 single precisions (MFLOPS)
64	68.8	33	2700
256	1796	557	3100
1024	7643	2630	2300

transistors [76]. The H-SIMD machine built on the Annapolis board contains two Virtex II FPGAs. The current implementation employs roughly 500 million transistors. On the other hand, a 2.8GHz Xeon processor contains about 300 million transistors [77]. For 1024×1024 FFT on IEEE-754 single-precision numbers, it takes 23 ms on a Xeon processor as opposed to 20 ms on the H-SIMD machine. According to a widely used VLSI complexity model, the cost C of implementing an algorithm is defined as $C = AT^N$, where A is the chip area, T is the execution time and N is the exponential weight. The chip area is directly proportional to the number of transistors, so the latter can be substituted for the former in the cost equation. Here, N is chosen as 3 to showcase the performance gain in the computation cost because of the slogan "silicon is free". The VLSI cost and speedup results in Table 4.6 are normalized with respect to the Xeon processor. The H-SIMD machine provides a speedup of 15% while its VLSI cost increase is only 9%. The cost-efficiency of the H-SIMD machine can be further increased by employing faster FPGAs than the Virtex II 6000 (90MHz). [71] reported 210MHz for floating-point operations on the Virtex2Pro. For recent FPGAs, like Virtex 4 and Stratix II, both Xilinx and Altera claim clock frequencies of 500MHz [64] [65]. Unlike gigahertz microprocessors

Table 4.6 Cost-Performance Comparison of the H-SIMD Machine and the Xeon Processor

System	Transis-tors (millions)	Execution Time(ms)	VLSI Cost (normalized)	Speedup (normalized)
2.8GHz Xeon	286	23	1	1
H-SIMD (Virtex-II)	700	20	1.85	1.15

with power problems, state-of-the-art FPGAs still have much room to grow their clock speed. It is expected to achieve a dramatic cost reduction in the near future with steady advances in FPGA technologies.

4.3 Example 3: 2D Discrete Cosine Transform

The two-dimensional discrete cosine transform (2D DCT or DCT2) is a popular algorithm widely used in image compression. Here, it is mapped onto the H-SIMD machine for performance evaluation.

4.3.1 HISA ISA for DCT2 [3]

DCT2 is a technique widely used in image processing and adopted by several compression standards such as H.261, H.263, and MPEG-4. For an input image s of size $N \times N$, the 2D DCT is computed in a simple way: the 1D DCT is applied to each row of s and then to each column of the result or vice versa. Thus the transform is given by:

$$Y(u,v) = \frac{2}{N} * C_u * C_v * \sum_{x=0}^{N-1} * \sum_{y=0}^{N-1} \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N} s(x,y) \qquad (4.12)$$

where $C_u = C_v = \sqrt{1/2}$ for $u = v = 0$ and $C_u = C_v = 1$ otherwise; $u, v = 0, \ldots, N-1$. The basis vectors of $\cos \frac{(2x+1)u\pi}{2N}$ can be pre-computed and stored in the on-chip memory of the FPGA for better performance. Assume large images are divided into blocks of size 8×8. These blocks are transformed via an 8×8 forward DCT2. Here, the HISA tailoring is shown for DCT2 with the H-SIMD design methodology.

Three HSIs are needed for DCT2. They are programmed with the software API library for the target Annapolis FPGA Wildstar II board.

- $host\_dct2\_load(LDM, N_h)$: This HSI loads the FPGA's LDM memory with an input matrix of size $N_h \times N_h$ via the host PCI bus;
- $host\_dct2(H_A, H_B, N_h)$: H_A is the input image of size $N_h \times N_h$ and $H_B = dct2(H_A)$;
- $host\_dct2\_retrieve(LDM, N_h)$: The final results are stored in the switched-out LDM and can be retrieved by the host through the PCI bus.

The FSIs in the middle of the HISA hierarchy run on the FPGA hardware. Assume that there are p nano-processors in each FPGA chip. There are two FSIs:

- $FPGA\_dct2\_load(F_a, LRF_i, N_f)$: FC will execute this instruction by loading each nano-processor's LRF with a block matrix of size $N_f \times N_f$ from matrix F_a of size $N_f^2 \times p$. F_a is the starting address of the input;
- $FPGA\_dct2(F_a, F_b, N_f)$: matrix F_b of size $N_f^2 \times p$ is stored into EDM and produced by applying DCT2 to the input matrix F_a.

The NPIs constitute only one customized instruction: 8×8 point DCT2, $NP\_dct2(R_a, N_f)$. Its datapath is produced by the Xilinx CoreGenerator [76] and targets image blocks of size 8×8, i.e., $N_f = 8$. The input data in our experiments is stored in a vector R_a of size 64 represented as 8-bit signed integer numbers. R_a is the starting address of the vector residing in the ERF memory of each nano-processor.

4.3.2 Task Partitioning for DCT2: Performance Analysis

DCT2 also uses the two interfaces in H-SIMD: the PCI bus with a bandwidth $B_{pci} = 133MHz \times 64$ bits and the SRAM bus with a bandwidth $B_{sram} = 960Mbytes/s$. In the case of DCT2 on an $N_h \times N_h$ matrix, assume that $N_h^2/64$ block matrices of size 8×8 are uniformly distributed among p nano-processors. The HDL simulation results in Figure 4.9 show that the CoreGenerator-produced 8×8-point DCT2 has a latency $L_{dct2} = 117cycles$. For the total computation time $T_{comp\_hosts\_dct2}$ of $host\_dct2(H_A, H_B, N_h)$:

$$T_{comp\_hosts\_dct2} > N_h^2/(N_f^2 * p) * T_{NP\_dct2\_8\times8} * N_{frame}, \tag{4.13}$$

where $T_{NP\_dct2\_8\times8}$ is the nano-processor execution time of DCT2 on a block matrix of size 8×8 and N_{frame} is the number of input image frames. On the other hand, the communication time T_{i/o_pci} of $host\_dct2\_load/retrieve$ depends on the available PCI I/O bandwidth and the interrupt latency:

$$T_{i/o\_pci} = 2 * b * N_h^2/B_{pci} * N_{frame} + T_{host\_int} \tag{4.14}$$

where b is the width in bits of each transaction. Pipeline balancing is guaranteed if $T_{comp\_hosts\_dct2}$ is greater than or equal to T_{i/o_pci}, and the computations fully overlap I/O communications. For $p = 2,\ 4\ or\ 8$, $B_{pci} = 133MHz \times 64bits$, $T_{host\_int} = 1.8ms, b = 8, N_{frame} = 6$ and $\tau = 8ns$ for 8-bit signed integer numbers, the simulation results in Figure 4.10 show that the computation time varies with two parameters: N_h and p. By tweaking these parameters, the computation time grows faster than the I/O communication time.

For effective nano-processor-level memory switching, the execution time $T_{NP\_dct2\_8\times8}$ of an 8×8-point DCT2 $NP\_dct2(R_a, 8)$ must be greater than p times the register file reference time T_{NP_load}. In fact, this condition can be easily met when $B_{sram} = 960Mbytes/s, p = 8, \tau = 8ns$.

4.3.3 DCT2: Implementation and Test Results

DCT2 is also implemented on a host PC workstation and an Annapolis FPGA board. The host is assigned the DCT2 transform $H_{dct2}(N_h, N_h) = dct2(H_A)$, where H_A and H_{dct2} are matrices of size $N_h \times N_h$.

The DCT2 cores were generated by the Xilinx CoreGenerator [76] and were configured for 8-bit input data, 16-bit coefficients, internal data and output results. After Xilinx ISE6.2 Place&Route, eight 126MHz nano-processors fit in each FPGA. DCT2 was tested on matrices of size 128×128, 256×256, 512×512 and 1024×1024. The timing results break down into the interrupt overhead, PCI reference time and DCT2 computation time, as shown in Figure 4.11 when N_h is set to 128, the frequent interrupts to the host contribute excessively to the performance penalty. When N_h is 256 or 512, the computation time does not increase long enough to fully overlap the sum of the host interrupt and PCI-SRAM sequential reference overheads. If $N_h = 1024$, the designed overlap scheme is so good that the interrupt and communication overheads are hidden and all the nano-processors work in parallel.

Denote by t_B the DCT2 time for one HSI-level $N_h \times N_h$ matrix. If the frame size is $N \times N$, then the time for DCT2 on a frame is $t_{frame} = N^2/N_h^2 * t_B$ and the frame rate is $R = 1/t_{frame}$. The frame rates are shown for various frames and HSI matrix blocks in Table 4.7. $N_h = 1024$ scores the highest frame rate. This is due to the combination of HISA and the memory overlap scheme.

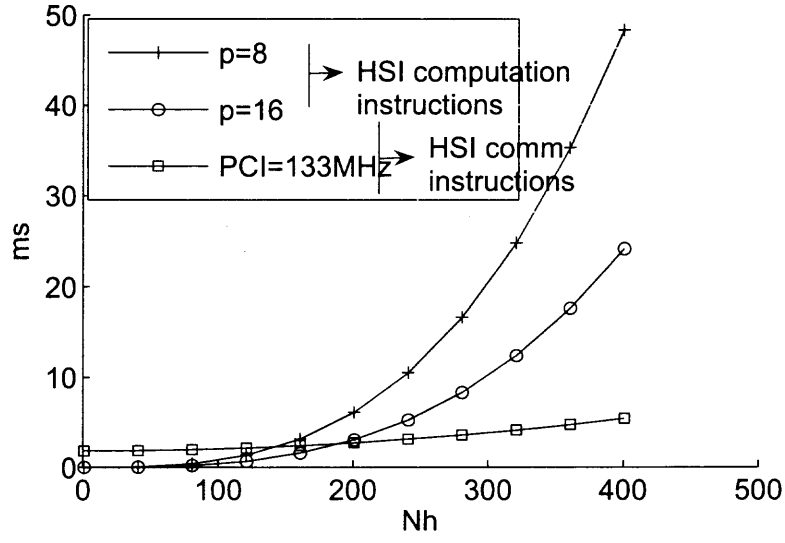
Table 4.8 compares the performance of the implementation on one Virtex II 6000 FPGA to the performance of a 2GHz Pentium processor as presented in [78]. If MMX and streaming SIMD instructions are enabled on the latter, H-SIMD yields a speedup of about $6\% \sim 18\%$; otherwise, the speedup is about four. This shows the effectiveness of the H-SIMD architecture on data-intensive applications.

Table 4.7 Frame Rates for Various Frame and Matrix Block Sizes

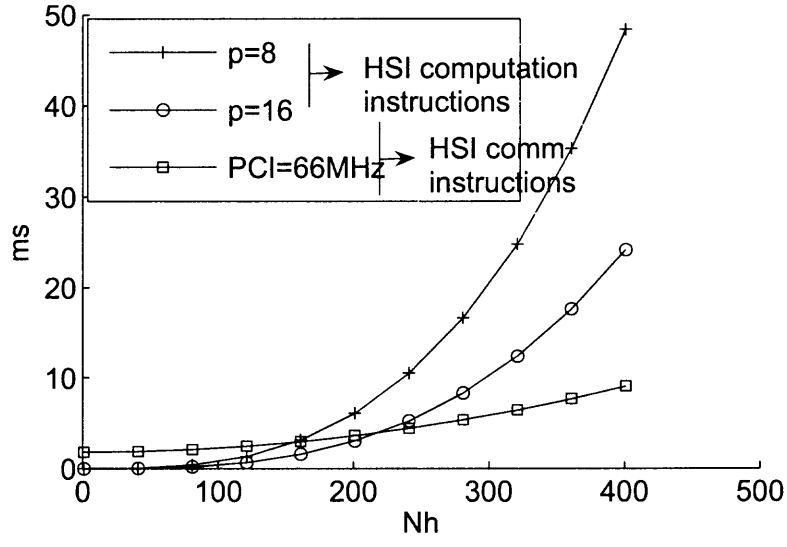
HSI Matrix Size / Frame Size	128	256	512	1024
1024 × 1024	45	129	243	516
2048 × 2048	11	32	60	129
4096 × 4096	3	8	15	32
8192 × 8192	0.7	2	3	8

Table 4.8 Performance Comparison for the 1024 × 1024-point DCT2

		Execution time (ms)
2GHz Pentium	**Integration implementation**	7.9
	MMX instructions	2.29
	MMX and streaming SIMD	2.05
H-SIMD machine		1.93



(a) $N_{bank} = 2$.



(b) $N_{bank} = 6$.

Figure 4.3 Execution times of the computation and communication FSIs as a function of N_f, p, and N_{bank}.

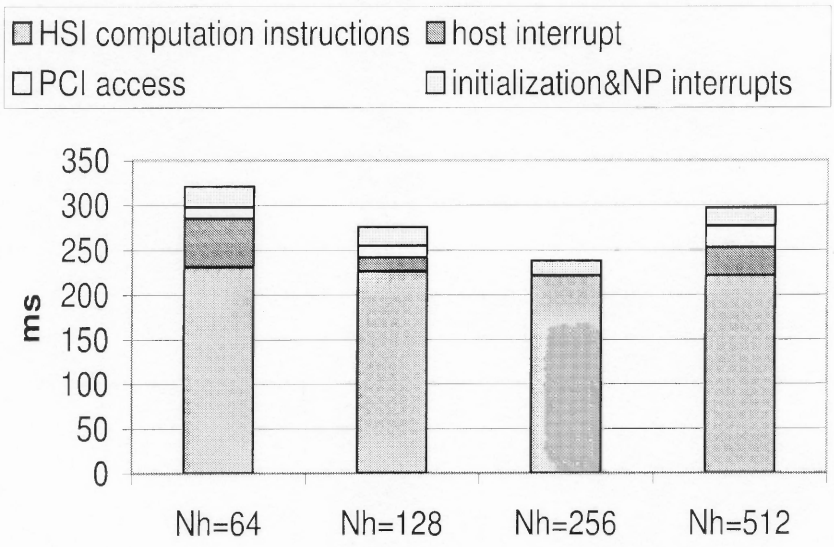


Figure 4.4 1024×1024 MM execution time as a function of N_h.

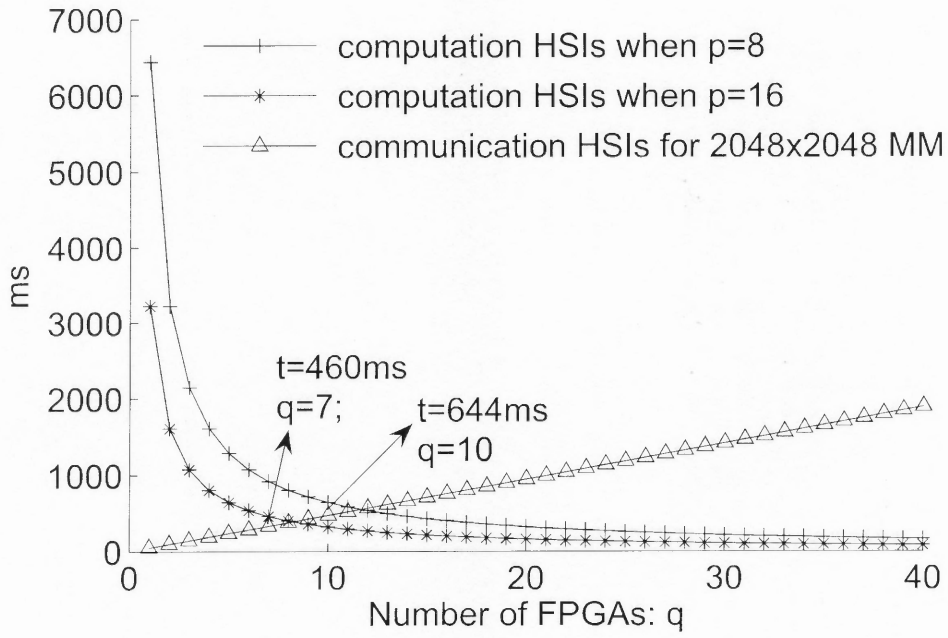


Figure 4.5 Execution time vs. number of FPGAs (2048×2048 MM).

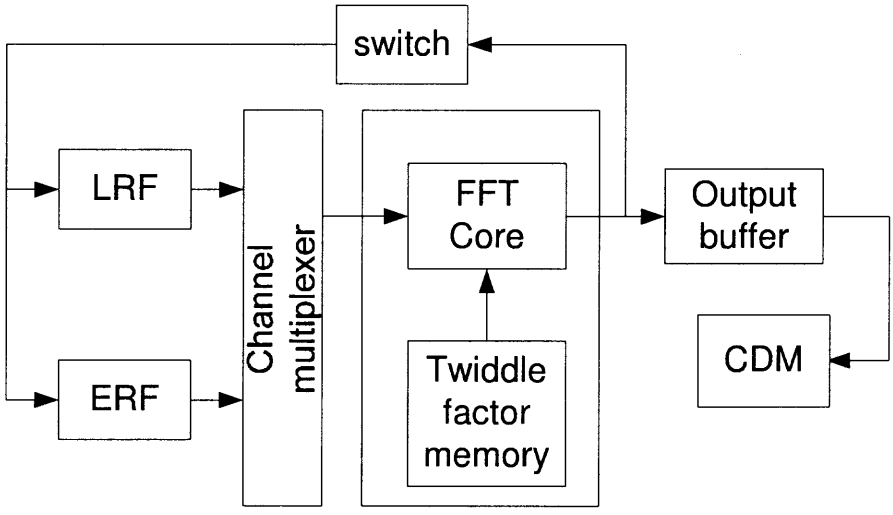
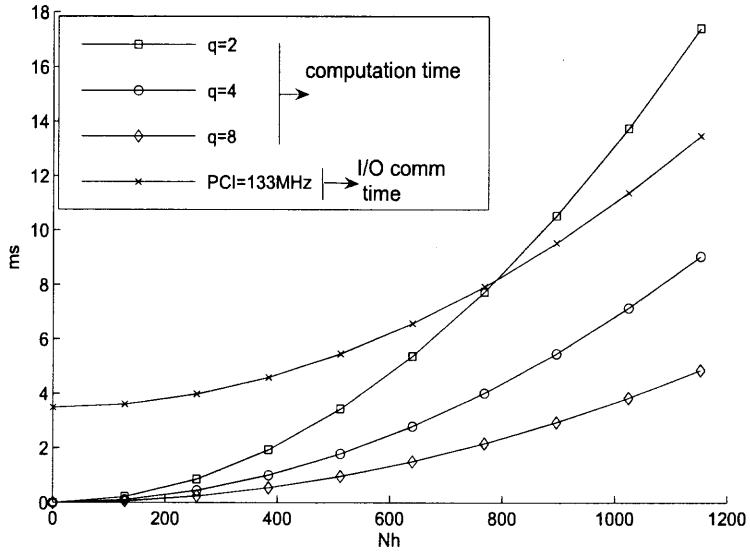
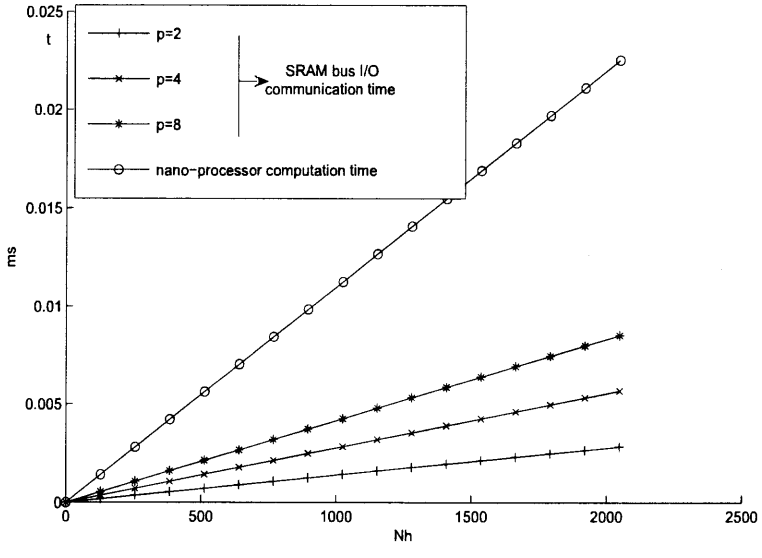


Figure 4.6 Nano-processor FFT datapath.

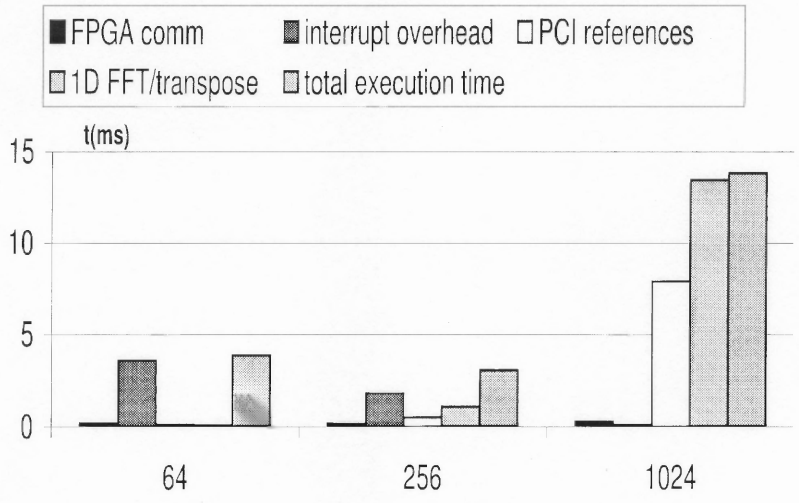


(a)

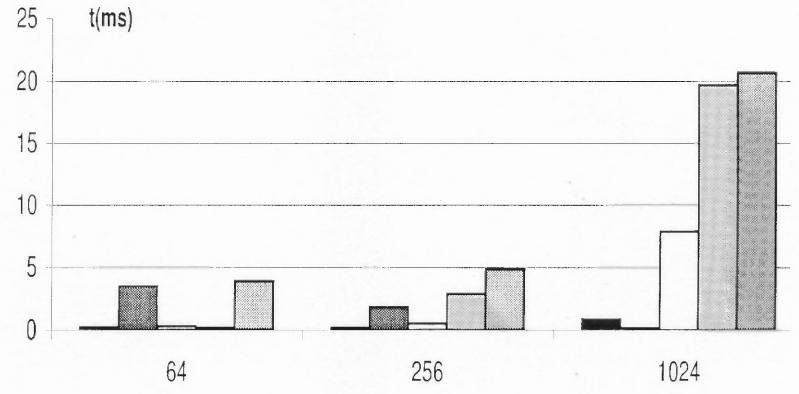


(b)

Figure 4.7 Computation and I/O communication times with (a) host PCI bandwidth and (b) SRAM bandwidth.



(a)



(b)

Figure 4.8 Execution time breakdown of 2D FFT on (a) 16-bit complex numbers and (b) IEEE754 single-precision floating-point numbers.

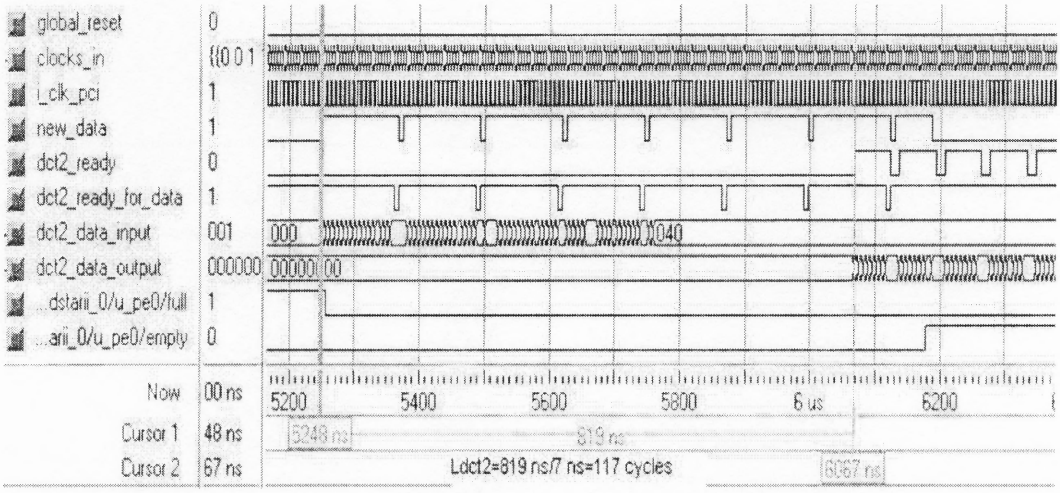


Figure 4.9 8×8-point 2D DCT engine's simulation result and its latency.

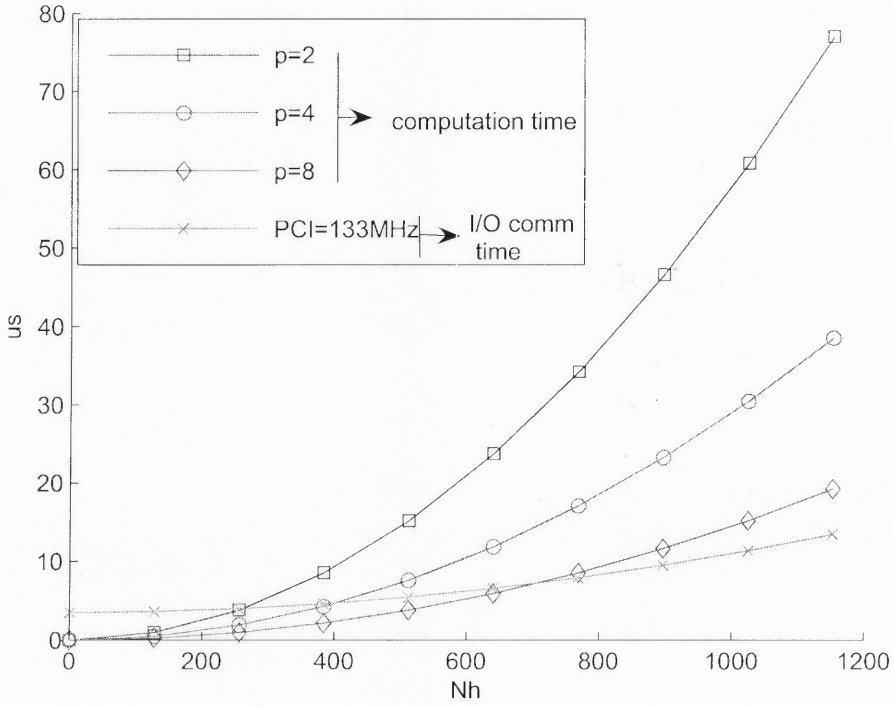


Figure 4.10 Computation vs I/O communication times as a function of N_h and p.

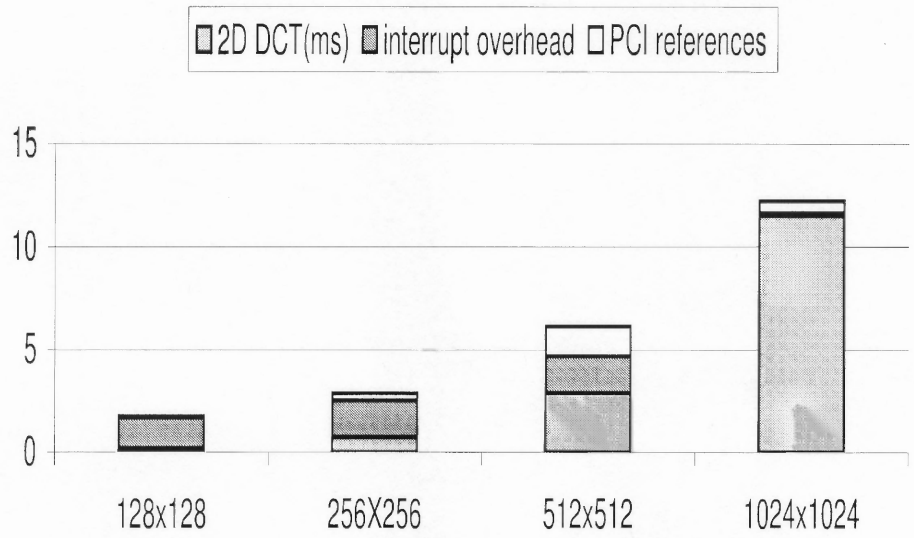


Figure 4.11 Execution time breakdown of DCT2 for six input frames.

CHAPTER 5

HLL-SUPPORTED RECONFIGURABLE COMPUTING

In previous chapters, a high-performance H-SIMD machine is investigated. Although it is designed to facilitate ease of application development, the hardware description language is employed to customize the logic functions at the FPGA and nanoprocessor layers. The HDL programming is very different from the algorithmic programming languages that are typically used by software developers. This chapter studies the SRC-6 reconfigurable machine that can use a high level algorithmic language, like the ANSI C language, to program the FPGA devices. Two important high-performance applications, matrix multiplication and image edge detection, are tested on the SRC-6 machine. The implemented algorithms are able to exploit the exposed data parallelism with independent functional units and application-specific cache support. Relevant performance and design tradeoffs are analyzed. A multi-threaded overlapping scheme is also proposed to reduce as much as possible, or even completely hide, runtime FPGA reconfiguration overheads.

5.1 SRC-6 General Purpose Reconfigurable Computer

5.1.1 Hardware Architecture

The SRC-6 architecture is a scalable, hybrid workstation-FPGA platform capable of supporting a combination of up to 512 Intel microprocessors and 256 MAP processors with common shared memory. The MAP processor consists of two user-programmable Xilinx FPGA devices, six 4MB banks of DMA-enabled On-Board Memory (OBM), and a control FPGA [6]. Figure 5.1 shows the architecture of the SRC MAP processor. It is designed to expose the data parallelism in applications. The SRC-6 machine is also known for its high bandwidth interconnections. A system-level SRC-6 machine can be configured as a cluster of MAP processors interconnected via patented high-bandwidth interconnections: SNAP

and the Hi-Bar switch, as shown in Figure 5.2. A host microprocessor is connected to a MAP processor via a SNAP interface. The SNAP interface can be plugged directly into the microprocessor's DIMM slot, allowing SRC systems to sustain higher interconnection bandwidths than other PCI-based reconfigurable machines [42]. SNAP uses separate input and output ports, with each port currently sustaining a data payload bandwidth of 1.4GB/s. The SRC-patented Hi-Bar interconnection is characterized as a scalable, high-bandwidth and low-latency switch. Microprocessors, MAPs and common memory nodes can all be connected to Hi-Bar. Each I/O port can sustain a data payload of 1.4 GB/s for an aggregate bisection data bandwidth of 22.4 GB/s per 16 ports.

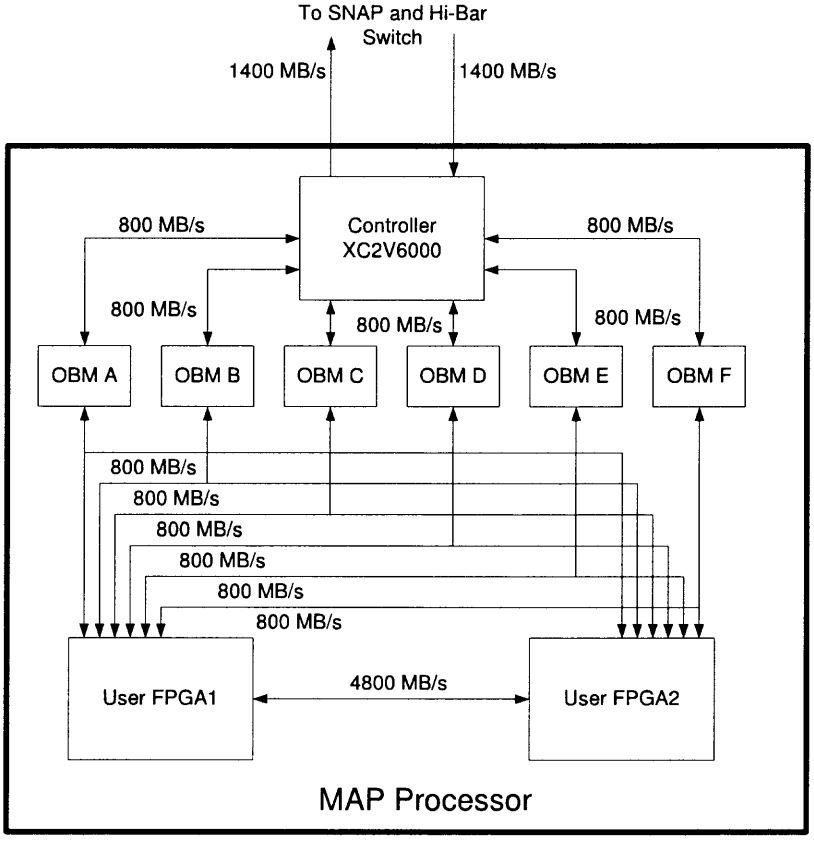


Figure 5.1 The architecture of the MAP processor in the SRC-6 machine [6].

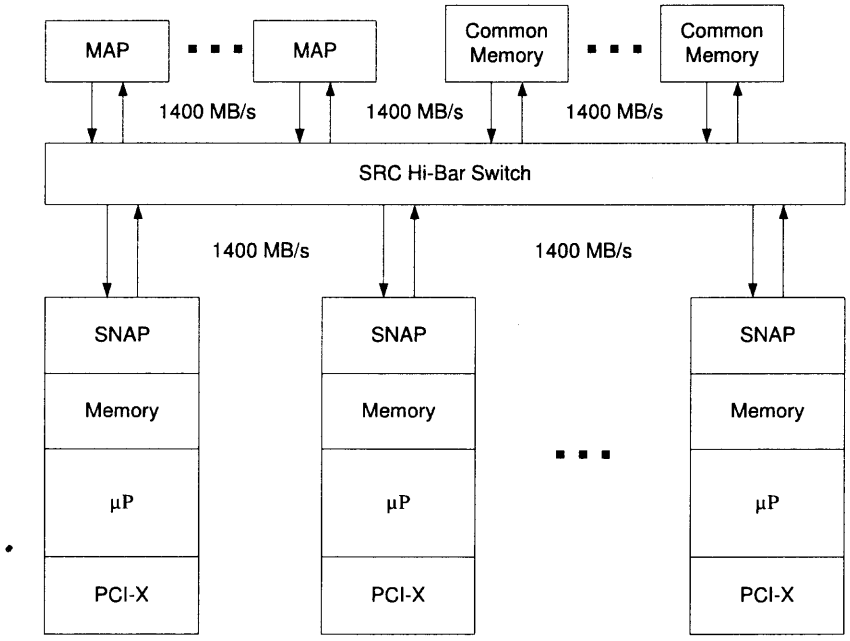


Figure 5.2 SRC-6 high end configuration with SNAP and a Hi-Bar switch.

5.1.2 SRC Programming Model

The SRC programming model, named Carte, uses a similar compilation process with conventional microprocessor-based computing systems. The C syntax can be applied to program MAP processors as well as host microprocessors [79]. Two types of application source files must be compiled for the Intel processor and the MAP processor, respectively. This is shown in Figure 5.3. Source files of the first type intended for execution on the Intel host are compiled using the Intel C Compiler (ICC). Source files of the second type that invoke FPGA macros are compiled with the MAP compiler to be executed on the MAP processor. Here, a macro is defined as a piece of digital logic designed to implement certain functions. SRC provides a standard built-in macro library for the implementation of DMA controllers, accumulators, counters, floating-point units and so on. Since users often wish to extend the existing set of macro operations, the Carte compiler allows users to integrate

their own custom macros into the compilation process for higher speedups achieved via custom hardware.

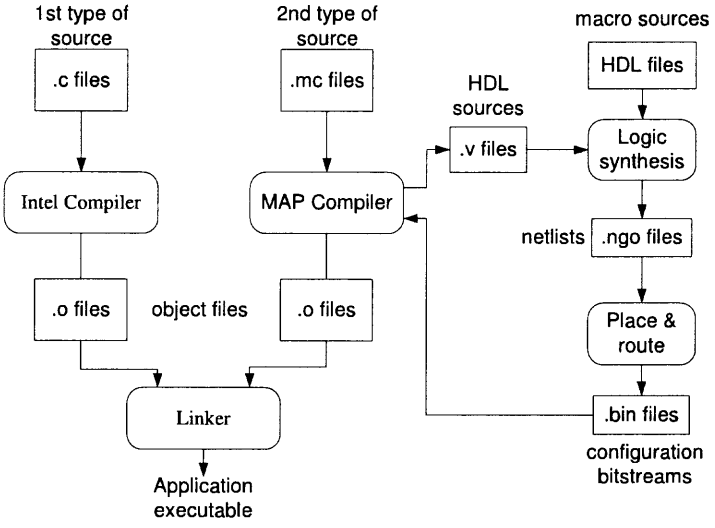


Figure 5.3 Carte compilation process.

5.2 Case Studies

5.2.1 Matrix Multiplication

The matrix multiplication kernel is a widely used computation engine for many scientific applications. The implemented algorithm employs an array of independent MACs and an array-based cache to exploit the exposed data parallelism. The data independence of the MACs allows the MAP processor to maximize the available data parallelism without data/structural hazards. The array-based cache, built with on-chip memory, is used to store rows of the input matrix. In order to efficiently provide computations with communicated input data, the number of cached rows is equal to the number of independent MACs. We instantiate P independent MACs. The block-based MM algorithm is described as follows: 1) Use DMA to transfer the input matrix blocks A and B from the host memory to the OBM banks A and B, respectively.

2) Cache P rows of matrix A and one column of matrix B from the OBM banks to the on-chip array-based cache.

3) Initiate P independent MAC computations in parallel; the results are the corresponding column parts of the product matrix C and are stored into the OBM bank C.

4) Cache the next column of matrix B and repeat Step 3 until all the columns of matrix B are processed.

5) Go to Step 2 until all the rows of matrix A have been processed.

An illustration of this algorithm is demonstrated in Figure 5.4, where the product of $A(4 \times N)$ multiplied by $B(N \times 4)$ is $C(4 \times 4)$; i.e., $C = A \times B$. N represents the number of elements in a row of matrix A or in a column of matrix B. The parameters P and N depend on the available FPGA resources and the specific implementation. According to our tests on the SRC-6 machine, up to 30 and 60 independent MACs can be implemented on a single MAP processor for IEEE 754 double-precision and single-precision floating-point operations, respectively. The timing reports show that the system can run at 100 MHz. This algorithm has a peak performance of $2 * P * f$ FLOPS (f is the running clock frequency) since each MAC performs two operations per cycle. Our scalable approach that can always take advantage of additional FPGA resources ensures that the data parallelism in MM can be fully exploited by the independent MACs.

Execution of an algorithm on the MAP processor generally goes through four steps: 1) the included FPGA devices are first configured with custom hardware logic; 2) input data are loaded from the host memory into the OBM banks; 3) data manipulation; 4) after data processing, the results are transferred from the OBM banks back to the host memory. The standard timing schedule is shown in Figure 5.5. Thus, the overall execution time for any application on the SRC-6 machine consists of three parts: the computation time T_{MAP} defined as the time spent within the MAP processor; the input and output times $T_{i/o}$ and $T'_{i/o}$ defined as the communication times (via DMA) to/from the MAP processor; and the

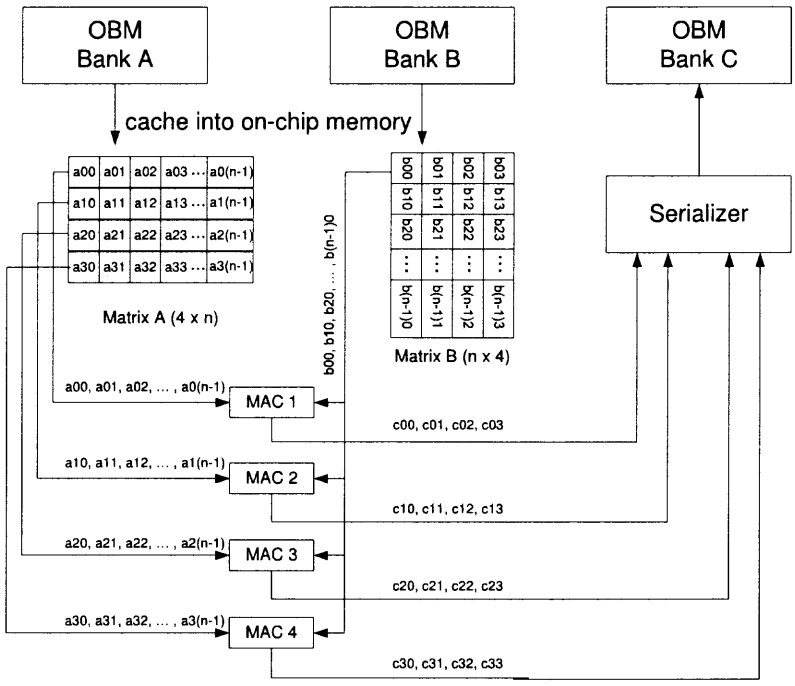


Figure 5.4 Illustration of MM computations and data movements.

FPGA (re)configuration overhead T_{config}. We define the actual execution time T_{exe} as the sum of T_{MAP}, $T_{i/o}$, and $T'_{i/o}$ only, i.e.,

$$T_{exe} = T_{MAP} + T_{i/o} + T'_{i/o} \tag{5.1}$$

The overall execution time of an algorithm is defined as:

$$T_{all} = T_{exe} + T_{config} \tag{5.2}$$

Table 5.1 presents MM experimental results for single-precision and double-precision floating-point implementations, respectively. All the timing results reported here are measured by the SRC-provided macro *read_timer ()* and the Linux system call *gettimeofday ()*. The matrices were dense and square, and were produced by ANSI C random functions.

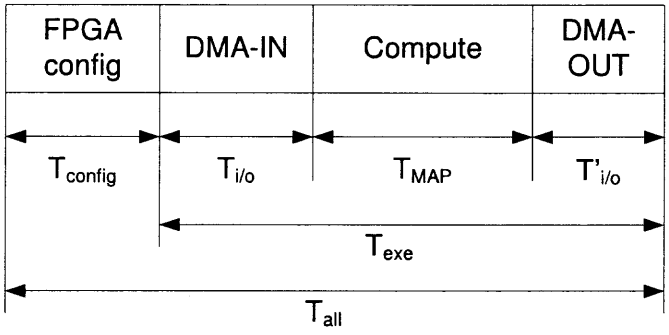


Figure 5.5 Run time schedule.

For matrices of size less than 800 × 800 (for simplicity, $N = 800$, from now on), the available data parallelism in the SRC-6 machine is not fully exploited, so it does not sustain very high performance. For a large matrix (i.e., $N > 800$), we can sustain 9.86 and 5.03 GFLOPS for the IEEE 754 single-precision and double-precision floating-point implementations, respectively. This is about 82.5% of the peak performance. Due to the exposed data parallelism, the sophisticated SRC interconnections and the data independence of the MACs, this performance can scale up easily with an increase in the number of MAP processors. In the SRC-6 machine, each MAP contains two XC2VP100 FPGA devices for user logic functions. The FPGA resource utilization results are shown in Table 5.2.

Although the implementation is restricted in performance by the Carte HLL environment, Table 5.3 compares the performance of our design with previous works on FPGA-based platforms for IEEE double-precision floating-point MM [70] [71] [1]. All of these implementations employ MACs as their underlying building blocks to gain hardware speedups. [70] [71] [1] build their architectures with processing elements (PEs) that also contain control logic. The specialized design in [70] is exclusive to MM and uses a MAC block optimized for the highest throughput per unit area. In [71], each PE comprises data registers, FIFOs, one MAC unit and control logics. All the PEs are connected as a linear

Table 5.1 Experimental MM Results on a Dual-FPGA MAP Processor for Single-precision and Double-precision Floating-point Implementations (Square Matrices)

	Matrix Size	$T_{exe}(ms)$	**Sustained GFLOPS**	**Peak Performance**
Single-precision ($P = 60$)	200	2.72	5.89	49%
	400	16.2	7.9	65%
	600	48.43	8.92	74%
	800	107.44	9.53	79%
	1000	202.78	9.86	82%
Double-precision ($P = 30$)	200	5.21	3.07	51%
	400	31.89	4.01	66%
	600	94.45	4.57	76%
	800	210.15	4.87	81%
	1000	397.39	5.03	83%

array to run in the SPMD (Single-Program Multiple-Data) mode. [1] designed the PE with the concepts of primitive nanoprocessors and register-level data switching. These designs were evaluated on XC2VP125 FPGAs (containing 55,616 slices) as opposed to the XC2VP100 devices in MAP (with 44,096 slices). With extrapolation, we can estimate that 20 independent double-precision MACs can fit into a single XC2VP125 FPGA running at 150 MHz and thus one MAP processor can achieve a peak performance of 12 GFLOPS. However, the MAP processor doubles the FPGAs in other solutions. Hence, we use "peak GFLOPS/10000 slices" as one of the benchmarking metrics, which indicates that the HLL-programmed reconfigurable machine may speed up the application development and yet incur more penalties on area and system frequency than other solutions.

Table 5.2 The MAP FPGA Resource Utilization Results on XC2VP100

IEEE floating-point format	BRAMs	Slices	Mults
single-precision	13%	91%	27%
double-precision	12%	93%	50%

5.2.2 Image Edge Detection

The Prewitt algorithm is a well-known edge-detection algorithm used in many image processing applications [80]. The algorithm employs the convolution of an image with two 3×3 constant masks shown in Figure 5.6; one is for detecting the X image gradient and the other for detecting the Y image gradient. The strength of the edge at any given image location is then the square root of the sum of the squares of these two gradients. This fundamental operation is performed over the entire image, with the result being another two-dimensional array of the same size called the gradient array.

$$\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

(a) (b)

Figure 5.6 Convolution masks for Prewitt edge detection. (a) X gradient; (b) Y gradient.

The computation with mask-based operations involves a high degree of operand reuse. On a cache-based general-purpose processor, a typical 9-point mask will most often have at least 8 cache hits, thus needing only one true memory load operation; in contrast, the FPGA can load a single operand per cycle from one memory bank. In order

Table 5.3 Comparison with Other MM Approaches on FPGAs

	Our approach	**[1]**	**[70]**	**[71]**
Frequency	150	180	200	200
Number of MACs	40	26	24	39
Peak GFLOPS	12	9.36	8.3	15.6
peak GFLOPS/10000 slices	1.0788	1.6830	1.4924	2.8049
HDL Support	Yes	Yes	Yes	Yes
HLL Support	Yes	No	No	No
Hide interrupt overhead	No	Yes	No	No

to overcome the I/O-bound bottleneck, we can employ on SRC-6 streaming data and delay-queue techniques to build up an FIFO-based cache architecture suitable for image convolution [81]. Streaming data enable the MAP processor to take two 64-bit words in each clock cycle. This avoids the need to buffer the input data before processing and eliminates the delay associated with data buffering. The data movement in the convolution cache system is illustrated with the sliding window in Figure 5.7. After the initial latency of queuing the first two rows of the input image, the edge detector can approach the theoretical limit of one clock cycle per pixel per sliding window.

Multiple sliding windows can enhance the performance. However, there is a tradeoff for the number of sliding windows because more sliding windows do not always guarantee higher performance. Given the input image of size $M \times N$ (M and N represent the image width and length, respectively), let us denote by W the number sliding windows of size $S \times S$. The startup latency $T_{startup}$ is required to fill up the FIFO-based cache using the

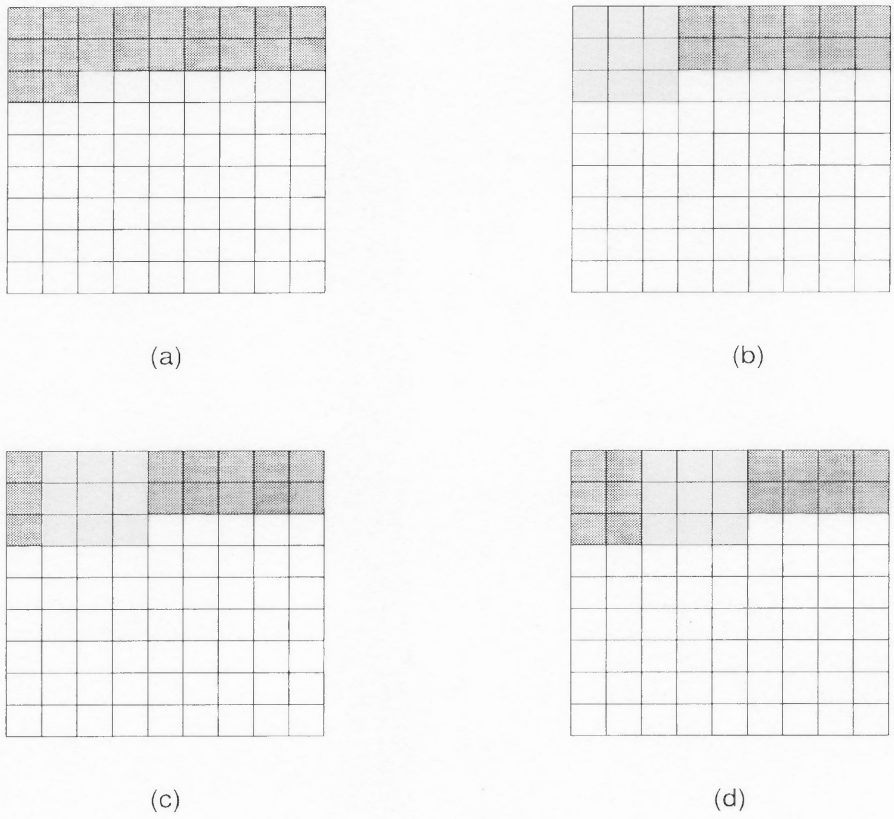


Figure 5.7 Applying 3×3 sliding window on SRC with delay queue support. (a) the startup latency; (b) the first, (c) the second, and (d) the third processing cycles.

interconnection bandwidth B. Then, the running time T_{slide} for the application of a sliding window on the assigned image partition contains two factors: the computation time T_{comp} and the streaming data I/O time T_{stream}. If T_{comp} is greater than T_{stream}, then the application is computation-bound. Otherwise, it is communication bound. The total execution time T_{exe} is the sum of T_{slide} and $T_{startup}$. Specifically, the following equations hold:

$$T_{startup} = ((S-1) * N + S - 1) * W/B \tag{5.3}$$

$$T_{comp} = (M - S + 1) * N/W \tag{5.4}$$

$$T_{stream} = W/B \tag{5.5}$$

$$T_{slide} = max\{T_{comp}, T_{stream}\} \tag{5.6}$$

Hence,

$$\begin{aligned} T_{exe} &= T_{slide} + T_{startup} \\ &= max\{(M-S+1)*N/W, W/B\} \\ &\quad +((S-1)*N+S-1)*W/B \end{aligned} \tag{5.7}$$

The objective here is to minimize the execution time T_{exe}. From the above analysis, we can tell that W will make two opposing contributions to T_{exe}. A bigger W can reduce the computation time and yet increase the I/O complexity. More sliding windows may consume up the available bandwidth in either delay queue initialization or pixel transfers. Hence, W is constrained not only by the available FPGA resources but also by the image size and the data movement bandwidth. Based on the current FPGA technology and standard image sizes, $T_{startup}$ can be ignored since the image size can be considered relatively large to the available FPGA resources. W is chosen such that the FPGAs run the computation at the rate that data can be delivered. The streaming input on the MAP processor is for two 64-bit words with each clock cycle. Therefore, the optimal W is 16 for 8-bit pixels. Based upon our tests, this number is not constrained by the FPGA resources. However, we implemented 8 sliding windows in our final experimentation to reduce the design complexity of both the implementation and the pixel layout in the memory. FPGA resource utilization results are shown in Table 5.4 for 8 and 16 sliding windows, respectively.

Table 5.4 FPGA Resource Utilization for Prewitt Edge Detection

Number of sliding windows	**BRAMs**	**Slices**	**Multiplier Blocks**
8	3%	52%	16%
16	7%	98%	33%

Table 5.5 shows the experimental results for 8 sliding windows and various image sizes. The system frequency is 100MHz. The execution time is counted similar to MM; i.e., it is the sum of the MAP computation and communication times. Ideally, each sliding window can process one pixel per cycle. The peak performance represents the processing of W pixels per cycle. We can see that the MAP can achieve on the average about 93% of the peak performance.

Table 5.5 MAP Performance for Prewitt Edge Detection

Input Image	MAP Execution (ms)	Peak Execution (ms)	Peak Percentage
256 × 256	0.0879	0.0819	93.18%
512 × 512	0.3530	0.3277	92.82%
640 × 480	0.4045	0.3840	94.94%
1024 × 1024	1.3946	1.3107	93.98%

We also compared the performance of SRC-6 with other computing platforms, namely SA-C [80] and the Pentium Processor [82]. [80] reported an implementation of the 512 × 512 Prewitt algorithm in 1.9 ms at 42 MHz. Its performance will double if the running clock frequency is doubled. Comparison results are shown in Table 5.6. Both of the FPGA implementations easily outperform the general-purpose processor because of their capability to exploit the data parallelism in the algorithm. The SRC implementation further gains a factor of 2 speedup over the SA-C solution.

5.2.3 Operation Overlapping via Multithreading

The FPGA configuration overhead may be a major contributing factor to the overall execution time of an application since more configuration bits are needed for high-density platform FPGAs. Based on tests in our laboratory, the overhead to reconfigure a user

Table 5.6 Performance Comparison for 512 × 512 Prewitt Edge Detection

	Pentium	**SA-C**	**SRC**
Frequency (MHz)	2000	100	100
Execution Time (ms)	1.5	0.8	0.35

FPGA in MAP, shown in Figure 5.1, is about 105ms; this was obtained by averaging the timing results from the SRC-provided macro *read_timer ()* and the Linux system call *gettimeofday ()*. This penalty is intolerable in high-performance computing because, for example, 600 × 600 single-precision floating-point matrix multiplication and edge detection for 50 1024 × 1024 frames take about 48 ms and 70 ms on a dual-FPGA MAP processor; the configuration overhead is about 210 ms for the two FPGAs. Thus, a design objective here is to reduce the configuration overhead. Our multithreading-based scheme is designed to overlap an application's computation time with another application's configuration time as much as possible, assuming multiple FPGAs. This scheme is shown in Figure 5.8. The master thread detects available FPGAs and loads the configuration bits for new applications while other FPGAs are busy on their computations. It enables FPGA configuration-data preloading to overlap computations, thus reducing or eliminating the impact of configuration overheads.

When multiple threads of control are initiated, Reader/Writer (R/W) locks are used for the purpose of thread synchronization. R/W locks are similar to mutexes (i.e., mutual exclusion objects), except that they allow for higher degree of parallelism. Three states are possible with a R/W lock: locked in read mode, locked in write mode, and unlocked. Only one thread at a time can hold a given lock in the write mode to run an application on the FPGAs while many threads can concurrently hold a lock in the read mode. In order to showcase the effect of the proposed overlapping scheme, the two user FPGAs in our SRC-6 machine were configured with two different tasks T1 and T2. They are designed for IEEE

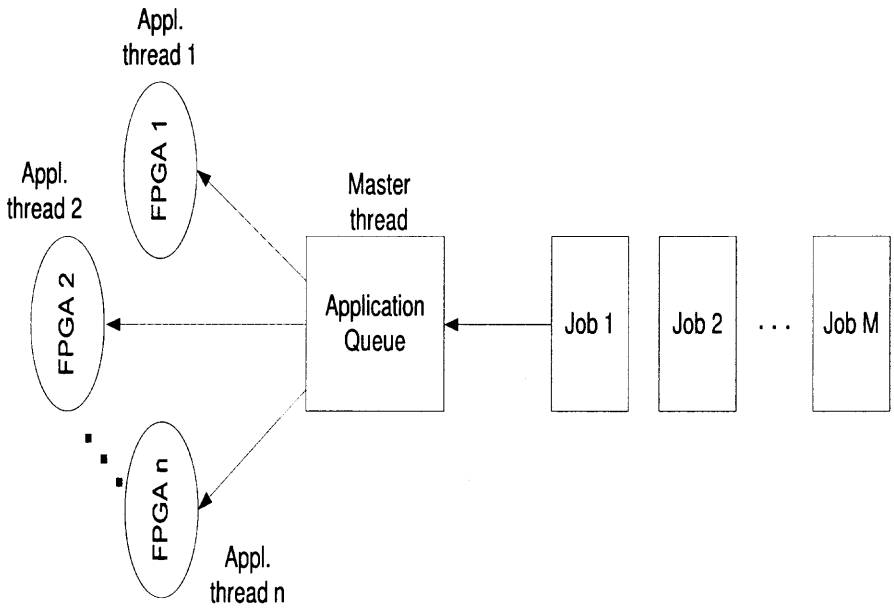


Figure 5.8 Multithreading-based operation overlapping scheme.

754 single-precision and double-precision floating-point MM, respectively. Their workload characteristics are shown in Table 5.7. The last column shows the task execution time T_{exe} with a single FPGA. T_{exe} is defined in Equation 5.1. Without a loss of generality, we chose the task workloads to be around the FPGA configuration overhead with $T_{exe}(T1) < T_{config}$ and $T_{exe}(T2) > T_{config}$ for the purpose of enhancing the effect of overlapping. The thread relationship is shown in Figure 5.9. Each thread needs the R/W lock to access the FPGAs. Two test cases were studied: T1 is set to run prior to T2, denoted by T1 $\rightarrow$ T2; in the other case T2 runs prior to T1 (i.e., T2 $\rightarrow$ T1). The test results in Table 5.8 shows the achieved speedup for the overlapped task in each test case. The sequential mode means that a task is prepared and runs in a single thread sequentially, starting with FPGA configuration and continuing with execution. In both cases, we can gain a performance boost of 83% on the average.

Table 5.7 Workload Characteristics of the T1 and T2 tasks

	Type	**Matrix Size**	**Execution Time** $T_{exe}(ms)$
T1	IEEE single-precision floating-point MM	600 × 600	98.63
T2	IEEE double-precision floating-point MM	510 × 510	128.16

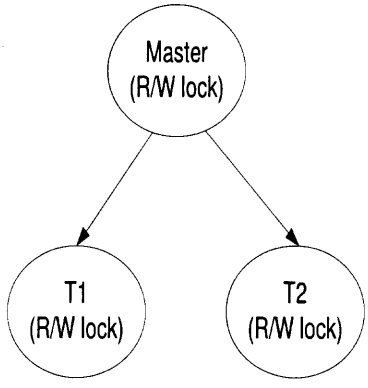


Figure 5.9 Thread relationship with R/W lock.

Table 5.8 Overlapped Task Execution Time for Multithreading-based MM

Exe Mode / **Test Case**	**Sequential (ms)**	**Multithreading (ms)**	**Speedup**
T1 (T2→T1)	203.87	104.4	1.95
T2 (T1→T2)	234.40	137.04	1.71

CHAPTER 6

CONCLUSIONS AND FUTURE RESEARCH

In this dissertation, a hierarchical SIMD computing architecture was presented. The H-SIMD machine is designed to facilitate ease of application development and reduction in channel bandwidth gaps associated with configurable computers controlled by a host. The H-SIMD machine is paired each time with an appropriate HISA instruction set to realize these objectives. This hierarchy makes the low-level HDL design flow transparent to the application designers and enables the reuse of pre-defined IP cores at the FPGA layer. A main idea behind the H-SIMD machine is to gain speedups by separating communication and computation instructions at each level, and then attempting to highly overlap the implementation of the former with the latter. This hierarchical classification of instructions is the enabling factor to combine the flexibility of general-purpose microprocessors with the efficiency of customized hardware to achieve significant speedups at reasonable cost compared to other computing paradigms. An innovative register file design based on the dual-clocked BlockRAMs of Xilinx FPGAs is employed to overcome the native shortcomings of conventional register files that may consume too many FPGA resources.

A memory switching scheme is applied at run time to address the bandwidth bottlenecks that may otherwise reduce or even remove speedups possible with configurable computing. It is designed to overlap communications with computations as much as possible. The conditions to achieve full overlaps of communications are extensively studied and applied to applications running on the H-SIMD machine. The exploration of the H-SIMD design space can be used to choose the proper task granularity at each layer to meet these conditions.

The H-SIMD machine sufficiently explores the parallelism in data-intensive applications. Its effectiveness was proved on matrix multiplication, 2D DCT, and 2D FFT.

Significant performance speedups are obtained compared to other computing paradigms. Aspects concerning area-efficiency, expandability, and generalization of the H-SIMD machine were also studied.

The HLL-enabled SRC-6 reconfigurable machine was used as well to exploit the available data parallelism by implementing independent functional units and application-specific cache support. Relevant performance and design tradeoffs were analyzed. Two important high-performance applications, matrix multiplication and image edge detection, were tested on the SRC-6 machine. The implemented algorithms are able to achieve high-performance without significant penalty on area and clock frequency. A multi-threaded overlapping scheme was proposed as well to reduce as much as possible, or even completely hide, runtime FPGA reconfiguration overheads.

Future research includes porting more applications to the H-SIMD machine to support a library of functional units for the FPGAs. This can further relieve application developers from the need for low-level hardware descriptions. More investigation is needed on resource utilization of the H-SIMD machine. Compared with a traditional FPGA tool chain, the H-SIMD machine tends to consume more configurable resources. Developing high-density functional units is one way that could improve the H-SIMD's resource efficiency in the future. Additionally, a performance estimation for the H-SIMD machine should be taken into account as early as possible in the design process, for example, prior to logic synthesis or technology layout. In order to do so, an accurate computation model for the H-SIMD machine is necessary. For an HLL-enabled reconfigurable machine, more exposed data parallelism can be exploited by efficiently scheduling data computations and communications between the FPGA and high bandwidth interconnections. System-level workload partitioning can also be used to take advantage of any available data parallelism at the algorithmic level, thus improving the application performance.

REFERENCES

[1] X. Xu and S. G. Ziavras, "H-SIMD machine: configurable parallel computing for matrix multiplication," in *Proc. IEEE Int. Conf. on Computer Design*, Oct. 2005, pp. 671-676.

[2] X. Xu and S. G. Ziavras, "A coarse-grain hierarchical technique for 2-Dimensional FFT on configurable parallel computers," *IEICE Trans. on Information and Systems, Special Issue on Parallel/Distributed Computing and Networking*, vol. E89-D, no. 2, Feb. 2006.

[3] X. Xu and S. G. Ziavras, "A hierarchically-controlled SIMD machine for 2D DCT on FPGAs," in *Proc. IEEE Int. Conf. on Systems-on-Chip*, Sept. 2005, pp. 276-279.

[4] J. Arnold, D. Buell, and E. Davis, "SPLASH 2," in *Proc. The Fourth ACM Symp. of Parallel Algorithms and Architectures*, June 1992, pp. 316-322.

[5] H. Singh, M. H. Lee, G. M. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. C. Filho, "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Trans. on Computers*, vol. 49, no. 5, pp. 465-481, May 2000.

[6] D. Poznanovic, "Application defined processors," *Linux Journal*, vol. 2005, no. 129, Jan. 2005.

[7] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Computing Surveys*, vol. 34, no. 2, pp. 171-210, June 2002.

[8] R. Tessier and W. Burleson, "Reconfigurable computing for digital signal processing: a survey," *Journal of VLSI Signal Processing*, vol. 28, no. 1-2, pp. 7-27, May 2001.

[9] J. Villasenor and W. H. Mangione-Smith, "Configurable computing," *Scientific American*, vol. 276, no. 6, June 1997.

[10] G. Estrin, "Organization of computer systems-the fixed plus variable structure computer," in *Proc. Western Joint Computer Conf.*, New York, 1960, pp. 33-40.

[11] E. Sanchez, M. Sipper, J.-O. Haenni, J.-L. Beuchat, A. Stauffer, and A. Perez-Uribe, "Static and dynamic configurable systems," *IEEE Trans. on Computers*, vol. 48, no. 6, pp. 556-564, June 1999.

[12] J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, , and P. Boucard, "Programmable active memories: reconfigurable systems come of age," *IEEE Trans. on VLSI Systems*, vol. 4, no. 1, pp. 56-69, Mar. 1996.

[13] S. Hauck, T. W. Fry, M. M. Hosler, and J. Kao, "The Chimaera reconfigurable functional unit," in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, Apr. 1997, pp. 87-96.

[14] P. M. Athanas and H. F. Silverman, "Processor reconfiguration through instruction-set metamorphosis," *Computer*, vol. 26, no. 3, pp. 11-18, Mar. 1993.

[15] M. J. Wirthlin and B. L. Hutchings, "A dynamic instruction set computer," in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, Apr. 1995, pp. 99-107.

[16] A. J. Elbirt and C. Paar, "An FPGA implementation and performance evaluation of the Serpent block cipher," in *Proc. ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays*, Feb. 2000, pp. 33-40.

[17] H. J. Kim and W. H. Mangione-Smith, "Factoring large numbers with programmable hardware," in *Proc. ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays*, Feb. 2000, pp. 41-48.

[18] G. Farquharson, W. Junek, A. Ramanathan, S. Frasier, R. Tessier, D. McLaughlin, M. Sletten, , and J. Toporkov, "A pod-based dual-beam InSAR," *IEEE Trans. on Geoscience and Remote Sensing Letters*, vol. 1, no. 2, pp. 62-65, Apr. 2004.

[19] N. K. Ratha and A. K. Jain, "Computer vision algorithms on reconfigurable logic arrays," *IEEE Trans. on Parallel and Distributed Systems*, vol. 10, no. 1, pp. 29-43, Jan. 1999.

[20] P. Zhong, M. Martonosi, P. Ashar, and S. Malik, "Accelerating boolean satisfiability with configurable hardware," in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, Apr. 1998, pp. 186-195.

[21] K. H. Leung, K. W. Ma, W. K. Wong, P. H. W. Leong, and N. T. Shatin, "FPGA implementation of a microcoded elliptic curve cryptographic processor," in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, Apr. 2000, pp. 68-76.

[22] M. J. Flynn, "Some computer organizations and their effectiveness," *IEEE Trans. on Computers*, vol. 21, no. 9, pp. 948-960, Sept. 1972.

[23] M. C. Herbordt, "Array control for high-performance SIMD systems," *Journal of Parallel and Distributed Computing*, vol. 64, no. 3, pp. 400-413, Mar. 2004.

[24] K. Suzuki, K. Suzuki, M. Daito, T. Inoue, K. Nadehara, M. Nomura, M. Mizuno, T. Iima, S. Sato, T. Fukuda, T. Arai, I. Kuroda, , and M. Yamashina, "A 2000-MOPS embedded RISC processor with a Rambus DRAM controller," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 7, pp. 1010-1021, July 1999.

[25] A. Kowalczyk, V. Adler, C. Amir, F. Chiu, C. P. Chang, W. J. D. Lange, Y. Ge, S. Ghosh, T. C. Hoang, B. Huang, S. Kant, Y. S. Kao, C. Khieu, S. Kumar, L. Lee, A. Liebermensch, X. Liu, N. G. Malur, A. A. Martin, H. Ngo, S. Oh, I. Orginos, L. Shih, B. Sur, M. Tremblay, A. Tzeng, D. Vo, S. Zambare, and J. Zong, "The first MAJC microprocessor: a dual CPU system-on-a-chip," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 11, pp. 1609-1616, Nov. 2001.

[26] C. Chakrabarti and M. Vishwanath, "Efficient realizations of the discrete and continuous wavelet transforms: from single chip implementations to mappings on SIMD array computers," *IEEE Trans. on Signal Processing*, vol. 43, no. 3, pp. 759-772, Mar. 1995.

[27] R. Duncan, "A survey of parallel computer architectures," *IEEE Trans. on Signal Processing*, vol. 23, no. 2, pp. 5-16, Feb. 1990.

[28] Xilinx Inc. (2006, Mar.) Xilinx Virtex II Platform FPGA User Guide. [Online]. Available: http://www.xilinx.com

[29] J. R. Hauser and J. Wawrzynek, "Garp: A MIPS processor with a reconfigurable coprocessor," in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, Apr. 1997, pp. 12-21.

[30] System C Group. (2006, Mar.). [Online]. Available: http://www.systemc.org

[31] I. Damaj, J. Hawkins, and A. Abdallah, "Mapping high level algorithms onto massively parallel reconfigurable hardware," in *Proc. ACS/IEEE Int. Conf. on Computer Systems and Applications*, July 2003.

[32] M. Gokhale, J. Stone, J. Arnold, and M. Kalinowski, "Stream-oriented FPGA computing in the Streams-C high level language," in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, Apr. 2000, pp. 49-56.

[33] D. Andrews, D. Niehaus, and P. Ashenden, "Programming models for hybrid CPU/FPGA chips," *Computer*, vol. 37, no. 1, pp. 118-120, Jan. 2004.

[34] C. Patterson and S. Guccione, "JBits design abstractions," in *Proc. IEEE Symp. on Field-programmable Custom-Computing Machines*, 2001, pp. 251-252.

[35] P. Bellows and B. Hutchings, "JHDL: an HDL for reconfigurable systems," in *Proc. IEEE Symp. on Field-programmable Custom-Computing Machines*, 1998, pp. 175-184.

[36] Altera Inc. (2006, Mar.) SOPC Builder User Guide. [Online]. Available: http://www.altera.com

[37] X. Wang and S. G. Ziavras, "Parallel LU factorization of sparse matrices on FPGA-based configurable computing engines," *Concurrency and Computation: Practice and Experience*, vol. 16, no. 4, pp. 319-343, Apr. 2004.

[38] Altera Inc. (2006, Mar.) Custom Instructions for the Nios Embedded Processor. [Online]. Available: http://www.altera.com

[39] T. Miyamori and U. Olukotun, "A quantitative analysis of reconfigurable coprocessors for multimedia applications," in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, Apr. 1998, pp. 2-11.

[40] C. R. Rupp, M. Landguth, T. Garverick, E. Gomersall, H. Holt, J. M. Arnold, and M. Gokhale, "The NAPA adaptive processing architecture," in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, Apr. 1998, pp. 28-37.

[41] R. D. Wittig and P. Chow, "OneChip: an FPGA processor with reconfigurable logic," in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, Apr. 1996, pp. 126-135.

[42] Annapolis Micro Systems Inc. (2004) Wildstar II Hardware Reference Manual.

[43] R. Laufer, R. Taylor, and H. Schmit, "PCI-PipeRench and the SwordAPI: a system for stream-based reconfigurable computing," in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, Apr. 1999, pp. 200-208.

[44] B. Mei, A. Lambrechts, J. Mignolet, D. Verkest, and R. Lauwereins, "Architecture exploration for a reconfigurable architecture template," *IEEE Design & Test of Computers*, pp. 90-101, Mar. 2005.

[45] C. Chang, J. Wawrzynek, and R. Brodersen, "BEE2: a high-end reconfigurable computing system," *IEEE Design & Test of Computers*, pp. 90-101, Mar. 2005.

[46] P. H. W. Leong, C. W. Sham, W. C. Wong, H. Y. Wong, W. S. Yuen, and M. P. Leong, "A bitstream reconfigurable FPGA implementation of the WSAT algorithm," *IEEE Trans. on VLSI Systems*, vol. 9, no. 1, Feb. 2001.

[47] D. Jin and S. G. Ziavras, "A super-programming approach for mining association rules in parallel on PC clusters," *IEEE Trans. on Parallel and Distributed Systems*, vol. 15, no. 9, pp. 783-794, Sept. 2004.

[48] S. Goldstein, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R. Taylor, and R. Laufer, "PipeRench: a coprocessor for streaming multimedia acceleration," in *Proc. Int. Symp. on Computer Architecture*, May 1999, pp. 28-39.

[49] E. Mirsky and A. DeHon, "MATRIX: a reconfigurable computing architecture with configurable instruction distribution and deployable resources," in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, Apr. 1996, pp. 157-166.

[50] J. Babb, M. Frank, V. Lee, E. Waingold, R. Barua, M. Taylor, J. Kim, and A. A. S. Devabhaktuni, "The RAW benchmark suite: Computation structures for general purpose computing," in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, Apr. 1997, pp. 134-143.

[51] B. Fagin and C. Renard, "Field programmable gate arrays and floating point arithmetic," *IEEE Trans. on VLSI Systems*, vol. 2, no. 3, pp. 365-367, Sept. 1994.

[52] K. Underwood, "FPGA vs. CPUs: trends in peak floating-point performance," in *ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays*, Feb. 2002, pp. 171-180.

[53] D. Galloway, "The Transmogrifier C hardware description language and compiler for FPGAs," in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, Apr. 1995, pp. 136-144.

[54] P. M. Athanas and H. Silverman, "Processor reconfiguration through instruction-set metamorphosis," *Computer*, vol. 26, no. 3, pp. 11-18, Mar. 1993.

[55] Chameleon Systems. (2006, Mar.). [Online]. Available: http://www.chameleonsystems.com

[56] P. K. Chan, *A Field-Programmable Prototyping Board: XC4000 User Guide*, 1st ed. University of California, Santa Cruz, 1994.

[57] A. Abbott, P. Athanas, L. Chen, and R. Elliott, "Finding lines and building pyramids with SPLASH 2," in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, Apr. 1994, pp. 155-161.

[58] P. Athanas and A. Abbott, "Real-time image processing on a custom computing platform," *Computer*, vol. 28, no. 2, pp. 16-24, Feb. 1995.

[59] Graham and B. Nelson, "Genetic algorithms in software and in hardware: a performance analysis of workstation and custom computing machine implementations," in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, Apr. 1996, pp. 216-225.

[60] X. Xu and S. G. Ziavras, "Iterative methods for solving linear systems of equations on FPGA-based machines," in *Proc. 18th Int. Conf. on Computers and Their Applications*, Mar. 2003, pp. 26-28.

[61] C. Kozyrakis, "Scalable vector media-processors for embedded systems," PhD dissertation, Univ. of California, Berkeley, May 2002.

[62] M. Mittal, A. Peleg, and U. Weiser, "MMX^{TM} technology architecture overview," *Intel Technology Journal*, no. 3, 1997.

[63] D. C. Chen and J. M. Rbaey, "A reconfigurable multiprocessor IC for rapid prototyping of algorithmic-specific high-speed DSP data paths," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 12, pp. 1895-1904, Dec. 1992.

[64] Xilinx Inc. (2006, Mar.). [Online]. Available: http://www.xilinx.com

[65] Altera Inc. (2006, Mar.). [Online]. Available: http://www.altera.com

[66] K. Bondalapati and V. K. Prasanna, "Reconfigurable computing systems," *Proceedings of IEEE*, vol. 90, no. 7, July 2002.

[67] J. L. Hennessy and D. A. Patterson, *Computer Architecture: a Quantitative Approach*, 3rd ed. Morgan Kaufmann, 2003.

[68] X. Xu, S. G. Ziavras, and T.-G. Chang, "An FPGA-based parallel accelerator for matrix multiplications in the Newton-Raphson method," in *Proc. IFIP Int. Conf. on Embedded and Ubiquitous Computing*, Dec. 2005, pp. 458-468.

[69] QinetiQ Ltd. (2004) Quixilica floating point FPGA cores datasheet.

[70] L. Zhuo and V. K. Prasanna, "Scalable and modular algorithms for floating-point matrix multiplication on FPGAs," in *Proc. 18th Int. Parallel and Distributed Processing Symp. (IPDPS'04)*, Apr. 2004.

[71] Y. Dou, S. Vassiliadis, G. K. Kuzmanov, and G. N. Gaydadjiev, "64-bit floating-point fpga matrix multiplication," in *Proc. ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays*, Feb. 2005.

[72] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*, 1st ed. Prentice-Hall Inc., 1975.

[73] Annapolis Microsystems, Inc. (2004) Corefire design suite.

[74] M. Frigo and S. Johnson, "The design and implementation of FFTW3," *Proceedings of IEEE*, vol. 93, no. 2, pp. 216-231, 2005.

[75] FFTW Group. (2006, Mar.). [Online]. Available: http://www.fftw.org

[76] Xilinx Inc. (2006, Mar.) Xilinx IP Center. [Online]. Available: http://www.xilinx.com/ipcenter

[77] Intel Inc. (2006, Mar.) Microprocessor Quick Reference. [Online]. Available: http://www.intel.com/pressroom/kits/quickreffam.htm#Xeon

[78] Intel Corp., "A fast precise implementation of 8×8 discrete cosine transform using the streaming SIMD extensions and MMX^{TM} instructions, Tech. Rep. AP-528.

[79] SRC Computers, Inc. (2003) SRC-6E C-programming environment guide.

[80] W. A. Najjar, W. Bohm, B. A. Draper, J. Hammes, R. Rinker, J. R. Beveridge, M. Chawathe, and C. Ross, "High-level language abstraction for reconfigurable computing," *Computer*, vol. 36, no. 8, pp. 63-69, Aug. 2003.

[81] D. S. Poznanovic, "Application development on the SRC computers," in *Proc. 19th Int. Parallel and Distributed Processing Symp. (IPDPS'05)*, Apr. 2005.

[82] S. J. Dillen and B. Cockburn, "Parallel filtering and thresholding of images on the SIMD DSP-RAM architecture," in *Proc. IEEE Canadian Conf. on Electrical and Computer Engineering*, May 2002, pp. 995-1000.