A STUDY OF SIMULATION AND VERIFICATION OF A MANY-CORE ARCHITECTURE ON TWO MODERN RECONFIGURABLE PLATFORMS

by

Dimitrij Krepis

A thesis submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Master of Science in Electrical and Computer Engineering

Summer 2007

A STUDY OF SIMULATION AND VERIFICATION OF A MANY-CORE ARCHITECTURE ON TWO MODERN RECONFIGURABLE PLATFORMS

by

Dimitrij Krepis

Approved: _____

Guang R. Gao, Ph.D.
Professor in charge of thesis on behalf of the Advisory Committee

Approved: _____

Gonzalo R. Arce, Ph.D.
Chair of the Department of Electrical and Computer Engineering

Approved: _____

Eric W. Kaler, Ph.D.
Dean of the College of Engineering

Approved: _____

Carolyn A. Thoroughgood, Ph.D.
Vice Provost for Research and Graduate Studies

ACKNOWLEDGMENTS

I would first thank Prof. Gao for giving me the opportunity to study and work in CAPSL.

I would thank Fei Chen, Yuhei Hayashi, Peiheng Zhang and Joseph Manzano for giving me valuable support and always being great friends.

I would thank XtremeData Inc. and specifically Gary Finley for providing us with the XD1000 Development platform and valuable support.

I would thank DRC Computer Corporation for providing us with the DS1000 Development platform and valuable support.

I want to thank everybody in CAPSL who helped me. You are truly a wonderful group of friends and colleagues. You were my family in the last years.

I want to thank the University of Applied Sciences Esslingen, the DAAD and the University of Delaware for giving me the opportunity to participate in a great exchange program.

DEDICATION

To my parents.

TABLE OF CONTENTS

LIST OF FIGURES . . . . . **viii**
ABSTRACT . . . . . **x**

Chapter

1 INTRODUCTION **1**

1.1 Background 1

1.1.1 Cyclops64 Supercomputer Architecture 1
1.1.2 Cyclops64 Crossbar Switch 4
1.1.3 DIMES Technology 4
1.1.4 MrsClops Emulation Engine 6

1.2 Problem Statement 8

1.2.1 Crossbar Switch Verification 8
1.2.2 SEmulator 9
1.2.3 FPGA Coprocessing Accelerator platform 10

1.3 Contributions 10
1.4 Synopsis 11

2 INTRODUCTION TO FPGA BASED CO-PROCESSING **12**

2.1 Contributions 12
2.2 Introduction 12
2.3 XtremeData XD1000 14

2.3.1 Hardware Programming 14

2.3.2 Software Programming 15

2.4 DRC DS1000 16

2.4.1 Hardware Programming 16
2.4.2 Software Programming 17

2.5 Porting of Cyclops64 Crossbar Switch Emulation 18

2.5.1 General Guidelines on Porting 18
2.5.2 Implementation Issues with FPGA-Coprocessing Accelerators 21

3 SEMULATOR 23

3.1 Contributions 23
3.2 Introduction 23
3.3 Problem Formulation 24
3.4 Solution Methodology and Implementation 25

3.4.1 Integration 25
3.4.2 Performance Improvements 27

3.5 Conclusions and Performance Results 27

4 VERIFICATION UTILITY: DESIGN AND IMPLEMENTATION 29

4.1 Contributions 29
4.2 Overview 29
4.3 Crossbar Switch Details and Principals of operation 30
4.4 C64 Packet Construction Format 33
4.5 Tests to implement 35
4.6 General Implementation 36
4.7 Experimental Results 42
4.8 VHDL Testbench Creation 44

5 RELATED WORK 46
6 FUTURE WORK 47

6.1 Existent Design Improvements 47
6.2 Algorithm Development 47

6.3 MiniMrsClops 48

Appendix

A SEMULATOR AND VERIFICATION UTILITY 50

A.1 SEmulator 50
A.2 Verification Utility 51

B LOADING OF BITFILES ONTO FPGA ACCELERATOR PLATFORMS 52

B.1 XtremeData: Altera FPGA 52
B.2 DRC: Xilinx FPGA 54

BIBLIOGRAPHY 57

LIST OF FIGURES

1.1 Cyclops-64 Chip Architecture 2

1.2 Cyclops-64 System Configuration 3

1.3 Single Logic Module with Multiple States. 5

1.4 Virtual Cycle vs Clock Cycle. 5

1.5 MrsClops Emulation Engine. 6

2.1 Opteron and FPGA Coprocessor. 13

2.2 XD1000 Logic Design. 14

2.3 DRC100-L60 RPU Diagram. 17

2.4 C64 Crossbar - Initial Interface. 19

2.5 C64 Crossbar - Common Interface. 20

2.6 C64 Crossbar - XD1000 Platform Interface. 20

2.7 DRC Manual 1.1.2: Single 32-bit HT DMA Read(1 Cycle Wait, 2 Cycle Latency). 21

4.1 C64 Switch Verification Environment with Visualization 31

4.2 A block diagram of C64 Crossbar Switch 32

4.3 A Logic Channel of The C64 Crossbar Switch 33

4.4 Data Flow Diagram of the Verification Tool 38

4.5 Flow Diagram of Input Pattern Generation 39

4.6 Flow Diagram of I/O with Emulation Hardware 40

4.7 Flow Diagram of Output Analysis 41

B.1 Altera Programmer: Main Screen 53

B.2 Altera Programmer: Device Selection 53

B.3 Altera Programmer: Programming Complete 54

B.4 Xilinx iMPACT: Startup 54

B.5 Xilinx iMPACT: Boundary Scan 55

B.6 Xilinx iMPACT: Device Selection and Programming 56

ABSTRACT

Recent advancements in computer performance have been hindered by the physical limitations of the current state-of-the-art semiconductor manufacturing technology. Steady performance growth, by means of increasing the operational frequency, is not possible any longer.

On the one hand we are "Hitting the Memory Wall"[1]: We need to increase the cache size to reduce the probability of cache misses. With the increased cache size and resulting transistor count on the other hand, we increase static and dynamic current leaks[2]. This results in an exponential growth of power consumption.

To keep up with the steady demand of increased performance, a paradigm shift towards multicore and many-core computer architecture designs has been made by the major microprocessor manufacturers.

This trend is going as far as integrating a very large number of simple processors onto a single die. This type of architecture is excellent for high-performance acceleration of domain-specific tasks. To achieve the best possible results, these accelerator platforms should be coupled with general-purpose microprocessors, which can take over the burden of running the operating system. One should note that the recent advancements in GPGPU technology along with steadily growing FPGA performance present other pathways of creating alternative acceleration platforms.

The IBM Cyclops64 Chip is part of a Petaflop class supercomputer architecture. This chip is a multicore architecture with a very large number of execution cores, memory banks and other components integrated on a single die. Each of these chip components are interconnected via the C64 Crossbar Switch, an efficient

interconnection network. Simulation of such an interconnection network is a very important task throughout the design and implementation process.

This thesis describes the design, implementation, and experimentation with an environment that may be used for acceleration, verification and validation of this interconnection network. In addition to this, a latency accurate Cyclops64 architectural simulator environment has been extended and accelerated.

Under the iterative emulation technology first proposed at CAPSL, named "DIMES"[3], a portion of FPGA resources will be time-shared among several identical modules of the target design and iteratively used to emulate them in multiple steps. The representation of the identical modules in the FPGA consists of (1) a single module copy and (2) a storage block holding all the states of the modules during iterative emulation. With the help of this technology, the Cyclops32[4, 5] chip along with the Cyclops64 Crossbar Switch[6] have been implemented on the AlphaData[7] platform earlier. Additionally, the Cyclops64 chip has been recently fully implemented on the IBM MrsClops[8] Emulation Engine.

Major contributions of this document are:

(i) We have ported the Cyclops64 interconnection network logic onto several state-of-the-art FPGA-Coprocessing Accelerator platforms. The increase in emulation speed as well as new logic designs of the Cyclops64 Architecture were the main driving forces for this work. Platforms such as XtremeData[9] XD1000 and DRC Computer[10] DS1000 were used for this work. Working on those novel platforms was a particularly interesting and challenging experience. We had to work on a range of different FPGA devices; we have faced and solved problems associated with bugs in vendor provided user interface logic, documentation and hardware device implementation. Throughout the process, we have provided valuable feedback to the platform designers. The resulting upgrades for future generations of these platforms will benefit from our

efforts.

(ii) With the use of those FPGA Accelerator platforms and based on the work of Fei Chen on the "LAST"[1][11] simulator, we were able to create a new type of computer architecture simulation. By combining software **S**imulation with hardware **E**mulation, called the **"SEmulator,"**[2] we were able to improve the "LAST" simulator. Using the accelerated "DIMES" emulation of the Cyclops64 interconnection network, we have dramatically increased the performance of this Cyclops64 Architecture simulator.

(iii) The newly developed verification utility for the Cyclops64 Interconnection Network has proven itself as an excellent tool for design verification and evaluation in various stages of development, such as verifying the initial KSM[3] design for AsapSim[4][12] simulation. It can provide VHDL test benches for design debugging during the creation of FPGA based emulation. In addition to that it can be used for the evaluation of these designs on the FPGA Accelerator platforms. The underlying framework also ensures portability over various emulation platforms for the "SEmulator." With this tool we have also demonstrated that the various interconnection network designs work as expected.

---

[1] Latency Accurate Software Testbench - Cyclops64 Architecture Simulator. Created by Fei Chen at CAPSL.

[2] Named "SEmulator" first by Fei Chen.

[3] HDL Language created at IBM. The Cyclops64 logic design is created in this language.

[4] Software simulator of the MrsClops Emulation Engine, created by Fei Chen at CAPSL.

Chapter 1

INTRODUCTION

1.1 Background

It is common knowledge that the number of transistors, integrated on a chip, has doubled every 18 months in the last three decades. Due to this, the power density of the chip becomes a big hindrance to the continuous improvement of its performance. In addition, after the CPU clock rate reaches multiple Gigahertz, the long memory access latency becomes unbearable. Now, a cache miss may cost several hundreds or even nearly a thousand CPU cycles. It is increasingly clear that, following the traditional architecture design methodology, it is not possible to solve, or even alleviate, these problems.

1.1.1 Cyclops64 Supercomputer Architecture

Under this background, "multi-core-on-a-chip" is emerging as a new architecture design methodology with very high potential for the next generation computer processor. It utilizes the huge number of transistors to integrate tens or hundreds of simple processor cores on a chip. Each of these "processors" can independently run a thread. If any thread is stalled because of accessing memory, other threads can still keep running to hide the memory latency. Usually, the chip operates at a moderate clock rate to keep power consumption at a low level. Therefore, multi-core has becomes the main trend in architecture design. Many companies have announced their multi-core plan or have already shipped out their multi-core products. Cyclops-64 (C64) [13, 14] is one of these multi-core architectures designed for

high end computing. It is under development by IBM T.J. Watson Research Center, ET International Inc., and the University of Delaware. A C64 supercomputer will be built out of tens of thousands of C64 processing nodes (chips), which are arranged in 3D-mesh cellular structure. Figure 1.1 shows the internal structure of a

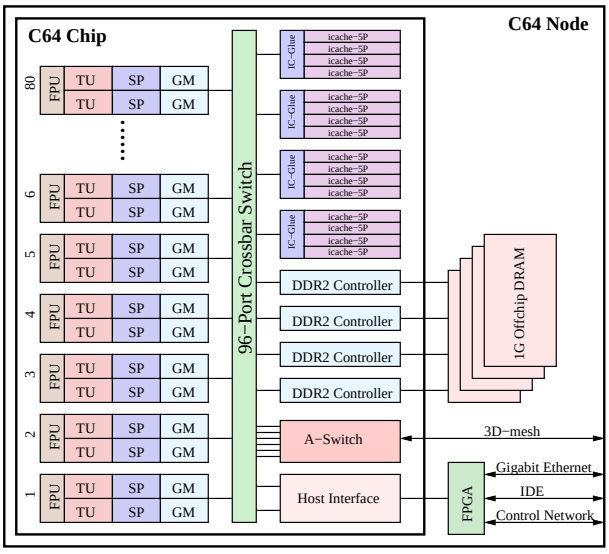

Figure 1.1: Cyclops-64 Chip Architecture

C64 processing node. Each C64 node consists of a C64 chip, 1GB external DRAM, and a small amount of external interface circuitry. The C64 chip has 80 integrated "processors", each processor has two thread units, one floating point unit, and two SRAM memory banks, each 32KB. A thread unit is a 64-bit, single issue, in-order RISC processor core operating at clock rate of 550MHz. A 32KB instruction cache, not shown in the figure, is shared among five processors. There is no data cache in the processor because it is very difficult to maintain cache coherence efficiently across so many processors. For this reason, a portion of each SRAM memory bank can be configured as scratchpad memory (SP), which is a fast temporary storage that can be used to exploit locality under software control. All of the remaining part of the SRAM form the global memory (GM) and is uniformly addressable from all thread units. The A-switch interface in the chip connects the C64 node to its six

neighbors in the 3D-mesh network. This network may scale up to several thousands of nodes, which will form the powerful parallel computational engine of the C64 supercomputer. All of the intra-chip components are connected by a 96-port crossbar network to form a tightly coupled SMP structure. The C64 Crossbar Switch has many unique properties, such as being non-blocking, pipelined, fairly arbitrated and delivering a very high level of data throughput.

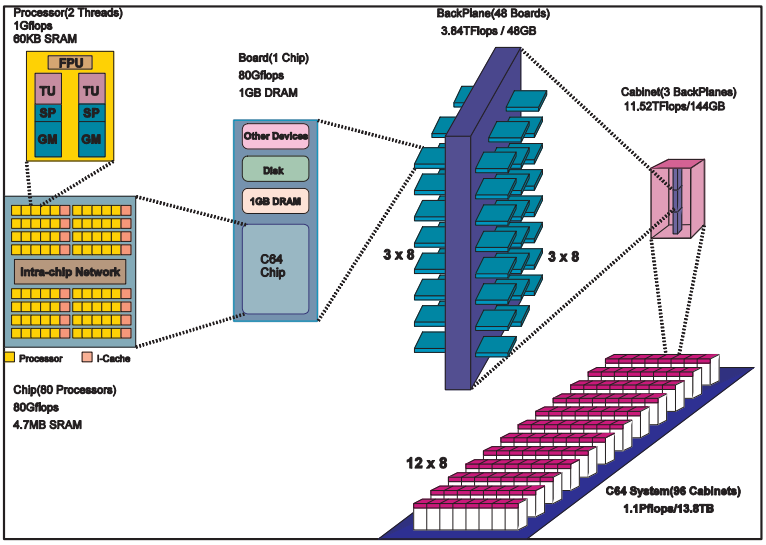

Figure 1.2: Cyclops-64 System Configuration

The proposed system configuration will consist of 96 Cabinets arranged in the pattern shown in Figure 1.2. Each cabinet contains 144 processor boards as well as local I/O Storage and Gigabit Ethernet ports for host communication and diagnostics.

The whole C64 system is designed to provide petaflop computer performance. It is targeted at applications that are highly parallelizable and require enormous amounts of computing power.

1.1.2 Cyclops64 Crossbar Switch

A crossbar switch is a mechanical or electrical switch that has a physical element for every possible connection between users, in which every input has a "cross point" with every output [15, 16, 17].

The Cyclops64 crossbar switch provides communication between the processors, on-chip SDRAM memory banks, off-chip DRAM memories, I-caches, I/O devices, A-switches and host interfaces [3]. The Cyclops64 crossbar switch is a 96 x 96 port buffered and pipelined design with input and output queues. Every port can create a connection to any other port, including itself. It provides 2 virtual channels, for forwarded and returned traffic. It also supports block transfers: a sequence transfer between two channels that cannot be interrupted by any other connection.

The Cyclops64 crossbar is supposed to provide a stable, non-blocking intra-connection network, supporting an efficient communication between components of the C64 chip. The full hardware bandwidth of the crossbar is 96 x 92[1] = 8832 bits /cycle. A performance analysis of the crossbar switch with micro benchmarks has shown an expected throughput of 0.59[17]. The performance of this part will strongly affect the overall performance of the chip. Correct implementation of the C64 Crossbar Switch component is crucial in the process of building the Cyclops64 Chip.

1.1.3 DIMES Technology

The DIMES[2][3] technology is FPGA based hardware emulation for large logic systems, incorporating a number of identical functional modules. Under this technology, a part of the FPGA resources will be time-shared among several identical modules of the target design and iteratively used to emulate them in multiple stages. Figure 1.3 shows a logic design consisting of many instances of identical logic. This

[1] Fixed 92-bit payloads are delivered through each source-destination pair

[2] **D**elaware **I**terative **M**ultiprocessor **E**mulation **S**ystem

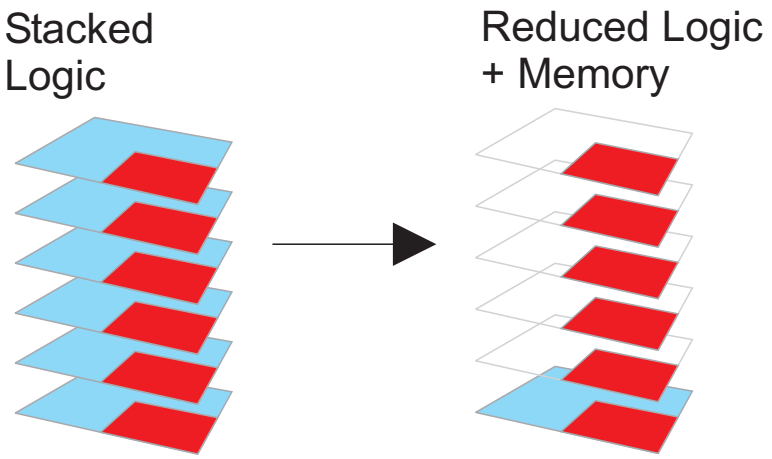

Figure 1.3: Single Logic Module with Multiple States.

design can be reduced to a single instance of logic and a memory block containing the states of every instance. By swapping the states, we can iteratively emulate the entire design. The implementation of the DIMES technology requires that all registers within the source logic must consist of the same logic primitive and all state bits must be load/storable via the top-level of the module. An automatic parser[3] is usually required to reach this condition. A state machine, for iterative emulation

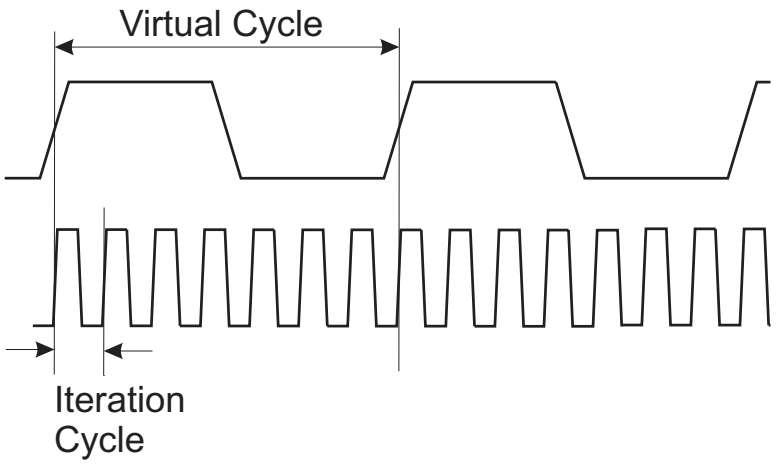

Figure 1.4: Virtual Cycle vs Clock Cycle.

[3] Such as moveRRReg. Created by Fei Chen

control, must be implemented to control the transition of logic state bits to and from the the memory. As we can see in Figure 1.4, the iterative emulation operates at a given clock speed on the FPGA. It takes a number of iteration cycles to finish one virtual cycle, hence to emulate the target logic for one cycle.

1.1.4 MrsClops Emulation Engine

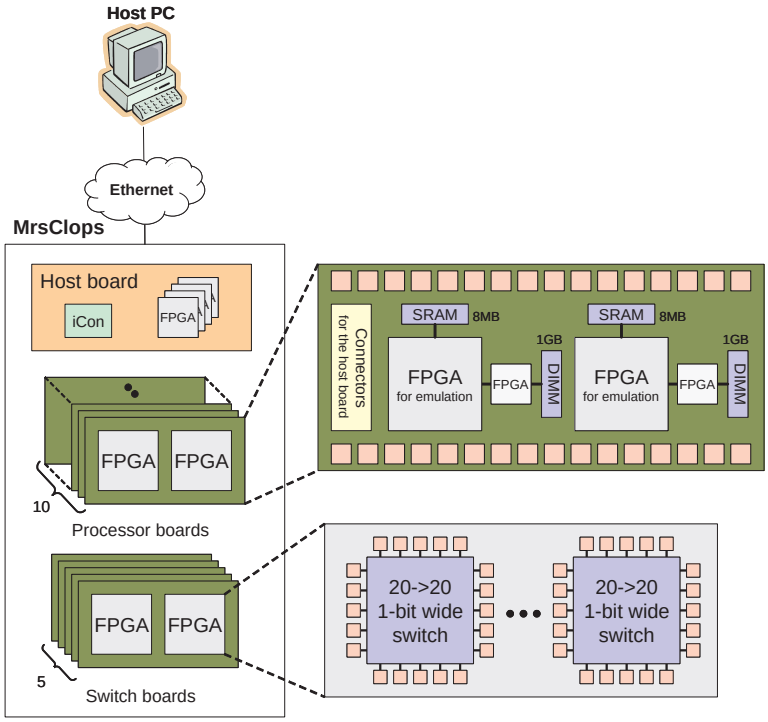

Figure 1.5: MrsClops Emulation Engine.

MrsClops is an FPGA-based high performance logic emulation and data processing engine. Developed in collaboration between IBM T.J. Watson Research Center and CAPSL[4], it is specifically designed for an efficient emulation of the Cyclops64 architecture. IBM has developed the hardware, the firmware and supporting software was developed entirely at CAPSL.

[4] by Yuhei Hayashi and Fei Chen

Figure 1.5 shows the basic internal structure of the MrsClops unit. The main part of the system consists of 30x Altera 2S90 FPGAs, 20 of them are used for emulation purposes. The remaining 10 provide the interconnection network between the emulation FPGAs. The MrsClops emulation engine is controlled by a host PC over an Ethernet connection.

The MrsClops system can emulate a large-scale logic design by distributing it over multiple FPGAs. The communication between those design partitions is controlled by the ECL[5] logic, which is present in every emulation FPGA. The ECL implements a NIOS II softcore processor to control this task.

To fit a large-scale design onto the MrsClops emulation engine, it must be distributed by submodules. Every submodule will be assigned to an emulation FPGA. We should not forget that we are trying to emulate the Cyclops64 chip architecture; it consists of a large number of identical modules. Depending on the size of these modules, we have a hybrid approach of emulation: "DIMES" and "ASAP" modes.

In DIMES mode, we will try to emulate multiple instances of the same module in a time-shared fashion. This can be done by swapping out the inputs, outputs and the internal state of the module iteratively. This approach is the preferred solution in terms of emulation speed; however the implementation of DIMES technology requires a considerable amount of work and may not be possible for some submodules.

ASAP: Any logic design can be represented as a data-flow graph for functional simulation. This data-flow graph can be translated into program code and executed on a processor. A number of simple execution units can be implemented onto an FPGA to execute the program code representation of a logic design in SIMD fashion. The ASAP mode comes into play in case that the DIMES approach is not possible

[5] Emulation Control Logic, created by Yuhei Hayashi.

due to fitting constraints.

AsapSCG[6]: AsapSCG creates the stack code representation of a logic design for emulation and ECL instructions for communication, according to the "shell" file[7]. This code can be natively executed on MrsClops and AsapSIM[8].

1.2 Problem Statement

1.2.1 Crossbar Switch Verification

The development of a high performance interconnection network for a multicore-chip is a very critical task in the system design workflow. This becomes especially important when handling a one-of-its-kind design which has never seen the light of day before. It is crucial, in the design phase, to verify the performance expectations and wisely choose the right design decision path.[6]

Based on those results, as well as, on implementation constraints such as die space usage, operation frequency requirements and the actual layout, the design will undergo multiple revisions. Each of these revisions must be verified to make sure that the components will perform to the specifications.

The Cyclops64 Crossbar Switch is a crucial part of the Cyclops64 Chip. It consists of 96 in/output ports with 102 Bits each and is a stateful logic. A direct verification, by applying every possible input vector and comparing the output vectors to expected results, is impractical, if not impossible, to apply within a reasonable amount of time.

The functionality of the C64 Crossbar Switch can be obviously verified "in-vitro": running the Cyclops64 Crossbar design against an architecture simulator. Such a simulator is the latency accurate system simulator for the Cyclops64 Architecture, called "LAST". Running programs on this simulator creates real traffic,

[6] Stack Code Generator for ASAP mode. Created by Fei Chen.

[7] Top file of a logic design. Describes how the submodules are interconnected.

[8] Simulator for MrsClops Emulation Engine, created by Fei Chen.

thus it can test the functionality of the C64 Switch design. However, this approach has advantages as well as flaws. Its obvious advantage is the fact that running a program and receiving correct calculation output from the run may seem like a good result, but a program execution does not cover enough corner cases and may not necessarily reveal errors and run successfully to completion.

The only usable approach for verification, in this case, is functional verification. By feeding the inputs with valid packets and checking the outputs for correct packet delivery, we can determine that the switch logic is performing as required. The input packets must be created according to the specifications, thus only legal input is provided. At the same time, the broadest range of input combinations should be covered to reveal weak points in the design.

To verify that the Cyclops64 Crossbar Switch is a reliable interconnection network, the following properties that define a reliable network must be confirmed:

- no data corruption
- no duplication
- in order delivery
- no lost data

These properties can be confirmed by the creation of an automatic testing utility that will create legitimate traffic on the C64 Crossbar Switch. Furthermore, the testing utility should be able to handle various design versions, support multiple emulation platforms and provide visualization and statistics.

1.2.2 SEmulator

It is common knowledge that a new computer architecture should have a short time-to-market in the HPC field. This also means that system software for this architecture should be on the market before the system is delivered. This will ensure

that the end-user base already has enough experience with the system environment and can productively use the system upon delivery, thus reducing the initial down-time. A successful creation of system software requires more then just following the set of system specifications; it requires creation of architectural simulators to verify the system software functionality as well as performance.

In case of the Cyclops64 Architecture there are 2 architectural level simulators existent: "FAST"[18] - a functionally accurate simulator and "LAST"[11] - a latency accurate counterpart. As the names may already suggest, the latency accurate simulator reflects the performance of the architecture very well due to the cycle accurate simulation of the interconnection network, however at a very low speed. This issue was addressed by improving the performance of the simulator in terms of emulating the interconnection network logic in FPGA hardware.

1.2.3 FPGA Coprocessing Accelerator platform

The use of SEmulator with PCI-Bus based FPGA platforms was limited due to the relatively high latency associated with such a shared interconnection bus. With the availability of state-of-the-art FPGA Coprocessing Accelerator platforms, directly coupled with the CPU over the HyperTransport Bus, an opportunity has arisen for the improvement of the SEmulator.

1.3 Contributions

This document introduces our design, implementation, and experiments with the following projects:

(i) The porting of the Cyclops64 interconnection network emulation using various state-of-the-art FPGA-Coprocessing Accelerator platforms. Platforms such as XtremeData[9] XD1000 and DRC Computer[10] DS1000 were used for this work.

(ii) Creation of a new type of computer architecture simulator, combining software **S**imulation with hardware **E**mulation, named the **"SEmulator,"** and thus improving the **L**atency **A**ccurate **S**imulation **T**estbench - "LAST"[11].

(iii) Creation of a verification utility for the Cyclops64 Interconnection Network as well as the underlying framework which ensures portability over various emulation platforms.

1.4 Synopsis

The paper is organized as follows. Chapter 2 introduces state-of-the-art FPGA Coprocessing Platforms and their programming. Chapter 3 introduces the SEmulator project, the acceleration of the LAST simulator with a FPGA emulation of the Crossbar Switch. Chapter 4 introduces the Verification of the Cyclops64 Crossbar Switch. Chapter 5 shows the related work. Chapter 6 gives a brief description of future work. Appendix A will teach users how to use the SEmulator, the Verification Utility Appendix B will show how to load bit-files onto XtremeData and DRC platforms.

Chapter 2

INTRODUCTION TO FPGA BASED CO-PROCESSING

2.1 Contributions

We have ported the Cyclops64 interconnection network logic onto several state-of-the-art FPGA-Coprocessing Accelerator platforms. The increase in emulation speed as well as new logic designs of the Cyclops64 Architecture were the main driving forces for this work. Platforms such as XtremeData[9] XD1000 and DRC Computer[10] DS1000 were used for this work. Working on those novel platforms was a particularly interesting and challenging experience. We had to work on a range of different FPGA devices; we have faced and solved problems associated with bugs in vendor provided user interface logic, documentation and hardware device implementation. Throughout the process, we have provided valuable feedback to the platform designers. The resulting upgrades for future generations of these platforms will benefit from our efforts.

2.2 Introduction

The recent advances in FPGA acceleration technology have made it possible to use FPGA as coprocessors. Platform solutions from DRC Computers and XtremeData Inc. make it possible to have a very low latency and high bandwidth connection between the FPGA and the main processor.

Both platforms are quite similar: Using a Dual-Processor computer mainboard, replace one of the processors with an FPGA. The communication with the

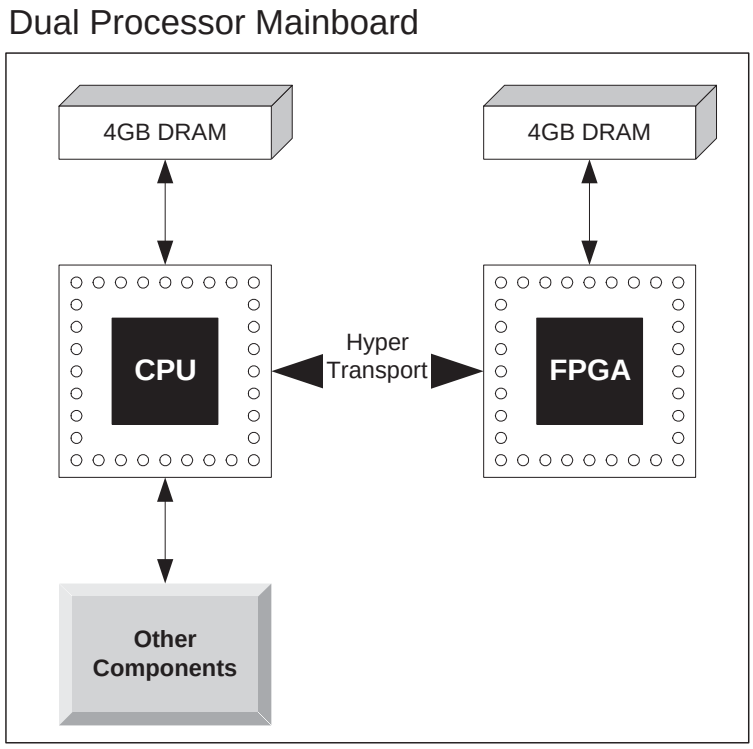

Figure 2.1: Opteron and FPGA Coprocessor.

processor runs over the HyperTransport interface. Using off-the-shelf computer parts, most of the hardware infrastructure is already present on the system. As we can see in Figure 2.1, the main processor and the FPGA have their own, logically and physically independent DRAM memory plugged into the mainboard directly. The platform vendors have to supply the PCB to electrically connect the FPGA into the mainboard socket. Both platforms also contain additional SRAM, a JTAG connection and FLASH memory for FPGA reprogramming.

The main difference between those two platforms is the vendor-supplied API for the user logic. We should also not forget that XtremeData is using Altera's FPGA products while DRC is using Xilinx. This is only important if the designer is taking advantage of FPGA vendor's exclusive functionality.

2.3 XtremeData XD1000

2.3.1 Hardware Programming

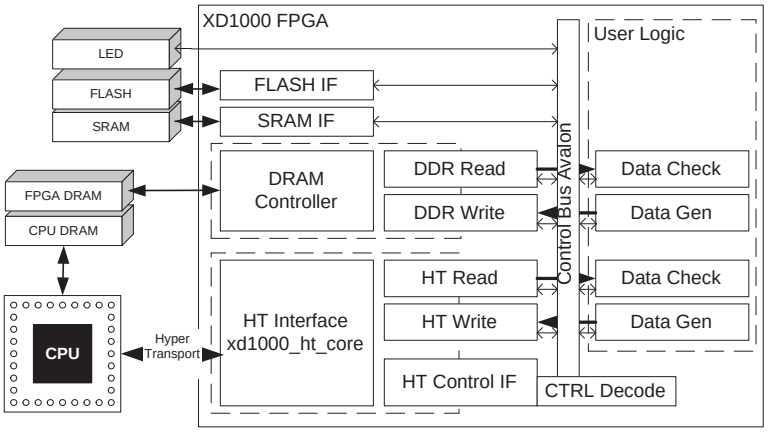

Figure 2.2: XD1000 Logic Design.

Figure 2.2 shows the reference design of the XD1000 FPGA Logic. As we can see, we have several important components for the hardware programming of this platform: We have the HyperTransport IP Core that controls the traffic between the host processor and the FPGA. We have a DDR controller for the off-board DRAM. In the XD1000 platform, the FPGA has access to 4GB of external memory exclusively. The CPU does not have direct access to this memory unless this functionality is implemented by the user within the user logic. We have the SRAM interface to communicate to the off-chip SRAM, in case the logic designer should run out of on-chip memory blocks. We also have FLASH memory to load FPGA images upon system bootup.

The communication between the FPGA and the Opteron processor is performed over the HyperTransport bus. The reference design utilizes a HyperTransport Controller Interface, **xd1000_ht_core**, to translate packets from HT Traffic into internal bus signals on the FPGA logic side. Altera Avalon[19] is used as the internal control bus with a 16 Bit address space. On the software side, this creates a 64kByte BAR window in shared memory, which is used to access the user logic. The

ht_ctrl_if module is responsible for bus access of each logic submodule in the user logic. It enables the select signal for the appropriate submodule and multiplexes the outputs.

While it is possible to use the control bus for data communication, one should use DMA[1] transfer capabilities of the logic design. It can be used by programming the **ht_write** and **ht_read** modules over the control bus. Programming of the DMA controllers, to initiate a transfer, can be accomplished from the software side. In the Reference Design, the modules **data_gen** and **data_check** are providing the data for the transfer. In the actual design, the user logic modules can generate and accept this data.

The modules **ddr_write** and **ddr_read** work identically to the DMA controller modules. Once specified the address and word count, they can transfer data to and from the FPGA attached DRAM memory modules.

Not shown in this picture, the PCB contains a small LED array, which can be accessed by the user logic to display status information.

2.3.2 Software Programming

Software programming of the XtremeData machine is quite straight forward. As mentioned in the hardware programming section, the device driver will create a shared memory window within the user application. This shared memory window is 64kByte big and directly maps into the control bus of the user logic design. All the writes and reads to this memory location will be redirected to the FPGA logic.

Upon the examination of the reference test utility, one will notice, that no address offsets are calculated directly to access each submodule within the reference design. Instead, a struct construct is created that directly maps its variables into the appropriate addresses. This is done by padding unused address space with dummy

[1] Direct Memory Access

arrays. The resulting construct is 64kBytes big and maps over the entire shared memory space. The following example is taken from the included test utility.

```
struct fpga_t {
        struct led_t {
                hw_uint32_t reserved1[3];
                hw_uint32_t data;
                hw_uint32_t reserved2[60];
        };
        struct dma_t {
                hw_uint32_t addrL;
                dummy_t<uint32_t> addrH;
                hw_uint32_t size;
                hw_uint32_t reserved1;
                hw_uint32_t control;
                hw_uint32_t reserved2[59];
        }
........
};
```

As we can see, each submodule has a 256byte address space within the user logic. Writing into the variables of those substructures, led_t and dma_t, will result in direct access to the submodule's registers.

2.4 DRC DS1000

2.4.1 Hardware Programming

Figure 2.3 shows the reference logic design by DRC, called "RPWare". It consists of "RPSysCore", a communication API, and User Logic. Considering the similarity of the two platforms, it contains the same basic building blocks. We have the HyperTransport Interface, the DMA controller for data transfers, SRAM and DRAM controllers for off-chip storage.

In contrast to XtremeData, the RPSysCore is only provided as a precompiled netlist, we do not have access to the source code. This means that the interface will be consistent, unless major improvements like bandwidth may change it. The

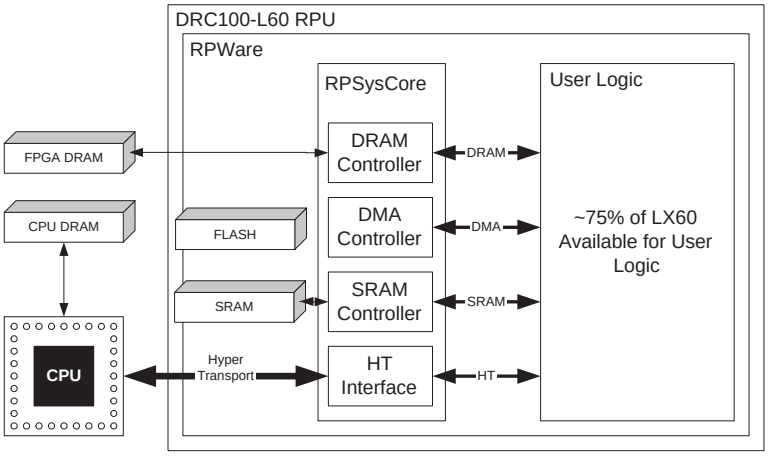

Figure 2.3: DRC100-L60 RPU Diagram.

communication between the user logic and RPWare is specified in the DRC User's Guide[10]. In case of this API, we only have DMA transfers to and from the logic. Every transfer provides a 23 bit address to the user logic. Using this information, the logic designer can direct the data to the proper location within the design. This transferred data is stored inside a FIFO within the RPSysCore, until it is ready to be consumed by the user logic.

2.4.2 Software Programming

Programming on the DS1000 platform is very straight forward. We have a set of the following interesting functions:

```
Rpu()

WriteUser(offset,void*,,size)
ReadUser()

WriteRam()
ReadRam()
```

The function "Rpu" will instantiate the driver connection to the FPGA logic. To write data to the FPGA, we can simply invoke the "WriteUser" and "ReadUser"

functions. In contrast to the XtremeData machine, we don't have to program the DMA controllers ourselves, because this task is taken care of by the drivers. The same method works for the off-chip DRAM, using "WriteRam" and "ReadRam" functions, we can access data directly from the off-chip memory of the FPGA. We can also check for the correctness of the transfers with the "GetErrorNum/Message functions", they will tell us if a write or read operation has completed correctly.

One should take a closer look at the DRC User's guide for the complete reference on the API.

2.5 Porting of Cyclops64 Crossbar Switch Emulation

2.5.1 General Guidelines on Porting

The iterative crossbar design was originally present on the AlphaData platform equipped with a Xilinx Virtex 2 FPGA. Porting this design onto the XtremeData platform has consisted of the following tasks: (1)Porting of the Xilinx based design to Altera and (2)integration of the new design into the vendor-provided reference design of XtremeData. Currently, the C64 crossbar switch design is provided for the MrsClops Emulation Engine in Altera IP format.

Porting of a design between different FPGA vendors usually consists of replacing the vendor-specific IP cores with their equivalents of the other vendor. In case of the iterative C64 Crossbar Switch design, we need to replace the memory-blocks containing the crossbar port data and state bits for iterative emulation.

Porting of IP Cores between Xilinx and Altera platforms requires an analysis of the original IP core with the core generator to determine the correct functionality and timing properties of the IP core. A simulation should verify those results, especially the timing. One must be very careful about the number of output pipeline stages and the registered ports.

After a new IP core is created according to the original specifications, it should be instantiated together with the original module, sharing the same inputs.

In this case, output timing differences can be detected during simulation. It is often necessary to rewrite wrappers for large memory blocks, such as those used to store the logic states during iterative emulation, since different FPGAs have different basic memory block sizes. The designer should not be wasteful with the resource usage on an FPGA, as difficulties with design fitting will arise.

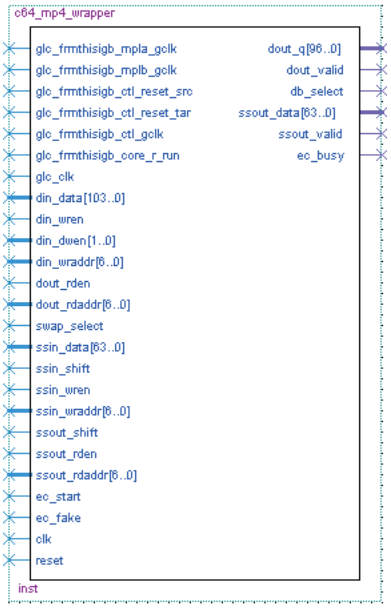

Figure 2.4: C64 Crossbar - Initial Interface.

While porting a design to a new platform, one should consider future work to make the new design easily portable to other platforms. Figure 2.4 shows the original interface of the Crossbar64 Switch, as used within the MrsClops Unit. As we can see, this interface provides a lot of signals not interesting for our integration. Editing this top file directly should be avoided, because the C64 logic is under constant development. New versions will be provided to us and they should be easy to integrate into our emulation platform. Creating a wrapper, which is a good representation of the crossbar interface, will make it easy to access the crossbar switch design, while being independent of the actual target platform. New versions of the crossbar switch design will be easier to integrate.

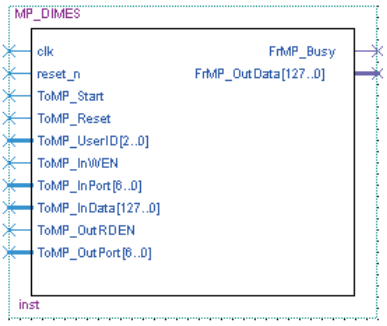

Figure 2.5: C64 Crossbar - Common Interface.

Figure 2.5 shows the new common interface wrapper. It only contains signals that are useful for the emulation of the C64 Crossbar Switch. We can write and read the data of the crossbar ports; we can initialize the emulation logic, we can start the emulation process and we have a notion of when the emulation cycle is finished. This logic interface wrapper is independent of the actual emulation platform and can be used on both, the XtremeData and the DRC machines.

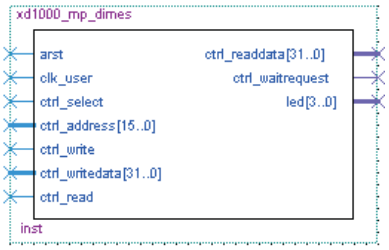

Figure 2.6: C64 Crossbar - XD1000 Platform Interface.

Finally, Figure 2.5 shows the actual wrapper used for the XD1000 platform. This is very platform specific. The wrapper logic provides access to the crossbar ports and status registers. As we can see, the data width is only 32 bits; the internal logic must make this wrapper compliant to the control-bus specifications of the reference design. This design can be easily integrated into the emulation platform.

2.5.2 Implementation Issues with FPGA-Coprocessing Accelerators

DRC and XtremeData have taken an entirely different approach with the releases of their FPGA programming API. XtremeData has released their entire communication logic to the customers, apart from the HyperTransport Core. This makes the inner workings of the system transparent to the customer.

For example, we[2] were able to fix the issue of not being able to freely set the operational frequency of the control bus to any value. This problem was actually caused by a copy-paste bug in the asynchronous FIFO control.

DRC has chosen a different path, their communication logic is provided as a precompiled netlist only and we have a black-box system. This means, we can only follow the documentation and hope that it is correct. At one point, we have noticed that the read-channel throttling seems to not work correctly. We will illustrate this problem with the following example:

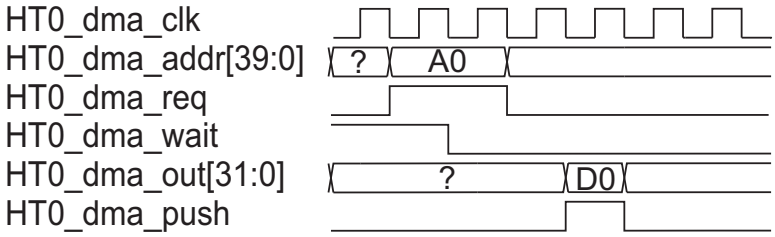

Figure 2.7: DRC Manual 1.1.2: Single 32-bit HT DMA Read(1 Cycle Wait, 2 Cycle Latency).

As we can see in Figure 2.7, reading from the user logic is throttled by the "dma_wait" signal. The communication API will assert a read request "dma_req" and keep the signal high and the address unchanged until the user logic accepts the read request. The user logic does this by deasserting the wait signal, or silently when the wait signal is always low. Then the user logic must assert the data to the "dma_out" signal and signalize it by the "dma_push" signal. We can see that a high

[2] Peiheng Zhang and Dimitrij Krepis

"dma_wait" signal is a valid initial condition according to the manual. However if we keep the "dma_wait" signal high as a general condition, so we don't have to buffer the address of the read request ourselves, we will never receive a read request delivered to the user logic. The reason therefore is the following: The read request is generated by a FIFO that queues the read requests and their respective addresses. Actually, the read request is the "data valid" signal from the FIFO. The "dma_wait" signal is attached to the output enable signal of the FIFO. If we never enable the output in the first place, we will never receive data from this FIFO.

The DRC and XtremeData platforms are both novel FPGA accelerators and, as any new product, they have their "children's diseases". It takes some engineering effort and time until those problems are fixed by the platform manufacturer. We can only try to avoid those problems and report them to the vendor.

Chapter 3

SEMULATOR

3.1 Contributions

With the use of state-of-the-art FPGA Coprocessing Accelerator platforms and based on the work of Fei Chen on the "LAST"[1][11] simulator, we were able to create a new type of computer architecture simulation. By combining software **S**imulation with hardware **E**mulation, called the **"SEmulator,"**[2] we were able to improve the "LAST" simulator. Using the accelerated "DIMES" emulation of the Cyclops64 interconnection network, we have dramatically increased the performance of this Cyclops64 Architecture simulator.

3.2 Introduction

It is common knowledge that a new computer architecture should have a short time-to-market in the HPC field. This also means that system software for this architecture should be on the market before the system is delivered. This will ensure that the end-user base already has enough experience with the system environment and can productively use the system upon delivery, thus reducing the initial down-time. A successful creation of system software requires more then just following the set of system specifications; it requires the creation of architectural simulators to verify the system software functionality as well as performance.

[1] Latency Accurate Software Testbench - Cyclops64 Architecture Simulator. Created by Fei Chen at CAPSL.

[2] Named "SEmulator" first by Fei Chen.

In the case of the Cyclops64 architecture, there are 2 architectural level simulators existent: "FAST" - a functionally accurate simulator and "LAST" - a latency accurate counterpart. As the names may already suggest, the latency accurate simulator reflects the performance of the architecture very well due to the cycle accurate simulation of the interconnection network, however at a very low speed. This issue was addressed by improving the performance of the simulator in terms of emulating the interconnection network logic in hardware.

3.3 Problem Formulation

The "LAST" simulator is a latency accurate simulator for the Cyclops64 architecture. It does reflect the performance of the Cyclops64 computer architecture precisely because of the accurate simulation of the on-chip interconnection network. The performance of the LAST simulator reaches about 500 cycles/second on a typical desktop machine. While it is possible to execute simple kernel benchmarks on this simulator, they take a very long time to complete. For example, a simple matrix multiplication of 64x64 elements takes about 24 hours to complete.[6]

A runtime analysis of the "LAST" simulator has revealed that more then 90% of processing resources are spent on the software simulation of the C64 Crossbar Switch. In fact, this figure is independent of the amount of data processed by the crossbar switch. This provides a great point of attack to improve this performance bottleneck.

Typically, a DIMES mode emulation of the C64 crossbar switch can be synthesized to run at 80MHz. The execution of one virtual cycle requires about 400 cycles for the iterative emulation. The reason for this length is the 4 iterations of 96 ports each, required for a correct execution of the crossbar logic. Running the DIMES mode crossbar logic at 80MHz, we can expect a peak performance of 200,000 cycles/second for the emulation. Due to communication overhead however, the real emulation speed will be significantly slower.

3.4 Solution Methodology and Implementation

The combination of the Cyclops64 simulator with an FPGA emulated version of the C64 Crossbar Switch is the main idea behind the SEmulator. The base for this project was the latency accurate C64 simulator "LAST". An iterative DIMES implementation of the C64 crossbar switch has been originally created by Fei Chen for the AlphaData [7] platform.

This project can be divided into the following phases:

- Initial Integration
- Performance Improvements
- Portability

3.4.1 Integration

The initial phase of the SEmulator project is the integration of the software simulator and the crossbar switch emulation.

The basic idea of integrating an emulation of the crossbar switch is quite simple. We must identify where input port data is inserted into the crossbar switch and write this data into the emulation. Furthermore, we must identify the execution of every simulation cycle to start the iterative emulation and stop the simulator until the emulation is complete. Then the data must be read from the emulation logic and inserted back into the simulation.

There are further steps involved, like the initialization phase to load drivers and bit-files into the FPGA logic, perform a logic reset and terminate the emulation logic connection upon finishing of the simulator. It is also of interest to collect performance data to see the improvement over the purely software simulation speed.

Packet data must be conditioned prior to being written to the hardware. The general format for the data is described in Table 4.1. Within the simulator, this data

is stored inside a word-aligned structure. To be processed by the emulation logic, it must be compacted to 102 bits. Same goes to the decoding stage for received traffic. Currently this data conversion is handled by an abstraction layer. The layer will also encode acknowledgement packets for the Token protocol.

The following data structure is sufficient to describe the FPGA emulation of the crossbar switch; it will not contain platform-specific data:

```
typedef struct FPGA_EMULATION
{
uint32 toCrossbar[4*PORT_NUM];
uint32 fromCrossbar[4*PORT_NUM];
uint32 cycle_cnt;
struct timeval    t_start;

}fpga_emulation;
```

We have two data arrays here: toCrossbar and fromCrossbar. They will contain a copy of the data that is currently inside the registers of the emulated crossbar. The reasons being are the read and write latency associated with the real hardware. Writing or reading data, especially with a DMA transfer, requires a large amount of data to perform efficiently. The field "cycle_cnt" contains the current number of executed emulation cycles; "t_start" has the timestamp of the initialization. Those fields are interesting for performance analysis.

The abstraction layer provides the following essential functions:

```
fpga_init()
fpga_write_data()
fpga_run()
fpga_read_data()
fpga_term()
```

Those functions provide the driver and hardware initialization, data communication between the shadow copies, as specified above, and the actual emulation hardware execution of the iterative emulation and statistical analysis. The interface of those

functions is independent of the platform. However, they are platform-specific internally.

3.4.2 Performance Improvements

In a project requiring a large amount of live communication with hardware, a low latency and high bandwidth connection with the emulation hardware is essential. This issue has been attacked from both flanks. On the one hand, we can use low-latency emulation hardware such as the FPGA accelerator platform from XtremeData. On the other hand, we can reduce the amount of communication by transmitting only new data and receiving where new data is expected. The algorithm therefore was implemented within the hardware abstraction layer. This algorithm is using the "guest-list" as mentioned in the previous chapter for crossbar switch verification.

One must consider a typical program execution pattern on the software simulator: Initially, code is loaded from DRAM into local SRAM. The only involved crossbar ports are the DRAM and the Thread units. When the code is loaded, there is virtually no communication happening. This fact can be used to our advantage to improve the simulation time.

We can switch between a programmed I/O communication pattern when a low number of ports is utilized, and switch to DMA transfers otherwise. With this technique, we can typically increase performance by a factor of 3.

3.5 Conclusions and Performance Results

The SEmulator project has proven itself successful in merging two technologies, software simulation and hardware emulation. By doing so, we have significantly accelerated the LAST simulator.

While the DIMES acceleration reaches a factor of 10 to 30 compared to a regular desktop machine, it only reaches a factor of 2 to 6 versus the software simulation

running on the accelerator platforms. The reason therefore is the improved microarchitecture of the AMD Opteron CPUs, thus alone providing a 5-fold acceleration over a Intel Pentium 4.

This may lead to the question: Why do we need costly hardware acceleration platforms? We should note that the latency accurate simulation of the Cyclops64 Crossbar Switch has been created manually, by transforming VHDL code into C, with a considerable effort. Each new version of this logic design would result in a repetition of this work, which we want to avoid.

We can use AsapSIM to easily emulate new versions of the crossbar switch in software. However, we should note that the simulation speed of the AsapSIM crossbar is only 1.5 cycles/second. At this simulation speed, even micro benchmarking would be out of reach.

Chapter 4

VERIFICATION UTILITY: DESIGN AND IMPLEMENTATION

4.1 Contributions

The newly developed verification utility for the Cyclops64 Interconnection Network has proven itself as an excellent tool for design verification and evaluation in various stages of development, such as verifying the initial KSM[1] design for AsapSim[2][12] simulation. It can provide VHDL test benches for design debugging during the creation of FPGA based emulation. In addition, it can be used for the evaluation of these designs on the FPGA Accelerator platforms. The underlying framework also ensures portability over various emulation platforms for the "SEmulator." With this tool we have also demonstrated that the various interconnection network designs work as expected.

4.2 Overview

The idea of implementing a generic verification utility has emerged from the project of improving the LAST Simulator. This simulator is cycle-accurate as it is entirely emulating the C64 Crossbar Switch design. The original performance of

[1] HDL Language created at IBM. The Cyclops64 logic design is created in this language.

[2] Software simulator of the MrsClops Emulation Engine, created by Fei Chen at CAPSL.

the simulator is quite slow on a typical desktop machine[3]: reaching around 500-800 cycles/second.While this is sufficient for micro benchmarking, running larger applications is impossible. Instead, the "FAST"[18, 13] simulator was developed for real application development. It is a functionally accurate simulator for the C64 Architecture, providing a very high simulation speed. On the other hand, it lacks a timing accurate implementation of the crossbar, thus limiting its use for benchmarking. To overcome this performance gap, the LAST simulator has been improved by using an FPGA accelerated version of the crossbar design. A performance analysis of the LAST simulator has revealed that over 90% of CPU-time is spent inside the crossbar code; thus proving the effectiveness of the approach.

The first version was implemented on an AlphaData ADM-XRC-II Board[7] using iterative emulation methodology[3]. As the nature of the application requires large amounts of communication, the PCI-Bus latency has shown itself as the bottleneck for achieving further speedups. Searching for a high-bandwidth and low-latency connection application platforms using the HyperTransport interface [20] XtremeData's XD1000 [9] and DRC Computing's DS1000 [10] were found. The crossbar design has been implemented on those platforms with a significant performance increase. Figure 4.1 shows a typical execution run of the verification utility with the crossbar visualization enabled.

4.3 Crossbar Switch Details and Principals of operation

The Cyclops64 Crossbar Switch is a stateful interconnection network. It provides a physical element for every possible connection between users, where every input has a corresponding output[6] [21]. The Cyclops64 Crossbar is a 7-stage pipelined 96x96 crossbar switch with input/output queues. It is used to provide on-chip communication between the execution units, SRAM memory banks, I-Caches,

[3] Intel P4-3GHz.

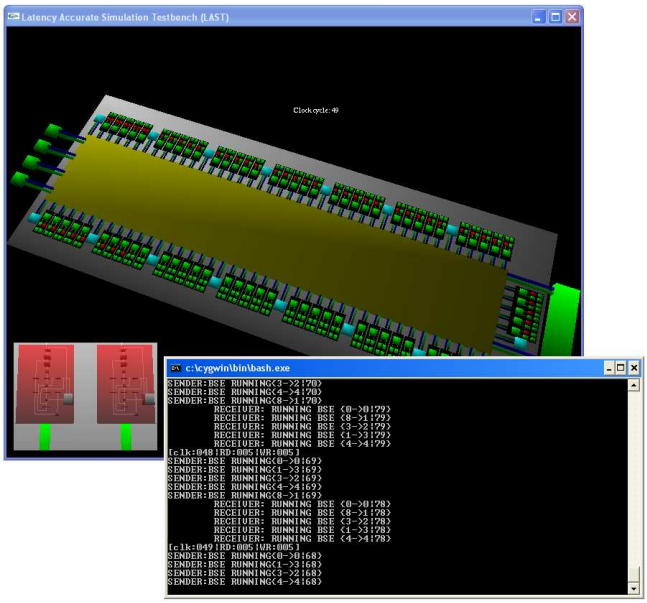

Figure 4.1: C64 Switch Verification Environment with Visualization

as well as off-chip DRAM memories, I/O devices, host interface and the A-Switch[4].

Figure 4.2 shows the basic internal structure of the crossbar switch. The crossbar switch consist of 96 physically identical ports which are interconnected. Each port of the crossbar consists of several components: a user interface providing input and output data ports, a source control unit, a target control unit, a 96x1 multiplexer, data FIFO and extra registers. The data FIFOs and their multiplexers are combined into the C64 crossbar core.

Packets are routed through the crossbar switch by the source control unit of the source port and the target control unit of the destination port. Flow control is handled by the token protocol. Virtual channels are supported for forwarded and returned traffic (from the standpoint of view of the same port), as well as non-interruptible block transfers. Arbitration is handled by a segmented LRU algorithm to reduce the floorspace of the switch.

[4] Inter-Chip communication within the 3D-Mesh

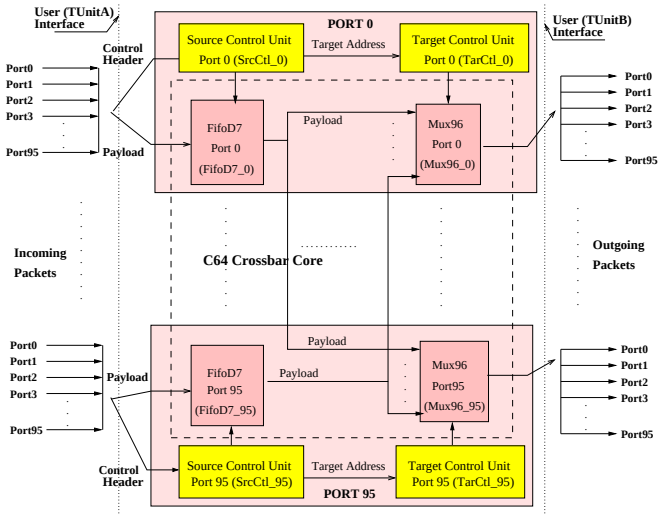

Figure 4.2: A block diagram of C64 Crossbar Switch

Data is being transmitted through the crossbar switch by means of datagram packets which contain the actual payload and routing tokens such as destination information, class and various other tags. Figure 4.3 shows a detailed operation of the crossbar design, representing a logic channel between ports i and j. This logic channel is composed of the source control unit of the port i and the target control unit of port j.

The source control unit handles data buffering in the FIFO, sending requests and forwarding the control header to the target control unit of the destination port and delivering the payload to the multiplexer.

The target control unit has to arbitrate the winner from all requesting source control units, send back the confirmation token to the requester and forward the respective flags to the destination port.

Flow control is achieved by the token protocol which is implemented by a 2-bit token and a counter inside each port. With each packet being injected into the port, the counter is decremented. Whenever the packet is delivered, an acknowledgment token is sent back to the source port and the counter is incremented.

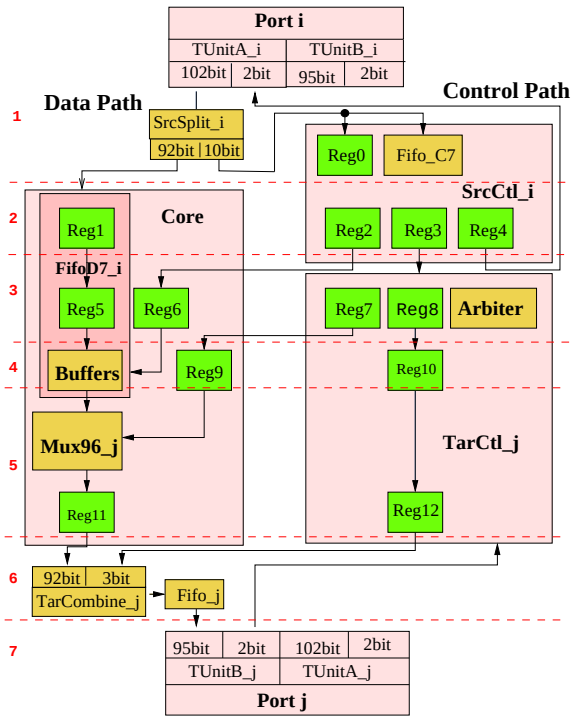

Figure 4.3: A Logic Channel of The C64 Crossbar Switch

4.4 C64 Packet Construction Format

Table 4.1 describes how a general C64 crossbar packet is constructed[22]. It consists of Data and Tag, as well as other fields. The 27-Bit **Tag** field contains addressing information for SRAM memory and can be treated as payload along the 64-Bit **Data** fields. Of interest is the 7-Bit **Tar** field that will address 96 possible destination ports. The **BSE** bit will be set at the beginning and end of a Block transfer. Such a series of packets can not be interrupted at the destination by packets coming from other source ports. The **C** bit will indicate the Class of the packet. A memory load request will have the Class set to 0, while the returned result will have the Class set to 1. The last bit indicates that the data at the port is valid and should be processed. In this table we omit the 2-bits used for the acknowledgment tokens. After a packet is delivered to the destination port, it's data-width is reduced to 95 bits by having the destination field removed.

Position	Field	Full Name	Notes
101 (94 after Tar stripped)	V	Valid	1 for a valid packet
100 (93 after Tar stripped)	C	Class	0 for forward, 1 for reverse
99 (92 after Tar stripped)	BSE	Block Start/End	1 during the first and last packets of a block transfer
98:92	Tar[6:0]	Target	Identification of one of 96 targets
91:64	T[27:0]	Tag	Tag
63:0	D[63:0]	Data	Data or other info

Table 4.1: C64 Crossbar Packet Format

Position	Field	Full Name	Notes
61	BSE Start	BSE Start	1 for start of Block Transfer
60	BSE Stop	BSE Stop	1 for end of Block Transfer
59:48	BSE TTL[11:0]	BSE TTL	Number of Packets until the end of Block Transfer
47:40	SrcID[7:0]	SourceID	Source Port, packet's origin.
39:32	TarID[7:0]	TargetID	Target Port, packet's destination
31:0	Cnt[31:0]	Cycle Cnt	Timestamp of packet injection

Table 4.2: Payload Packet Format

Table 4.2 describes fields that are necessary to verify the operational characteristics of the C64 crossbar switch. The fields **SrcID** and **TarID** store packets origin and destination ports and the field **Cnt** contains the emulation clock cycle number of the packet injection. This represents the packet's timestamp. With this information at hand the following characteristics can be assured:

Every packet injected into the crossbar can be identified by those 3 fields(SrcID, TarID, Cnt) and no two packets will have the same payload within a reasonable emulation time. By including the target information we can verify that the port which has received the packet was indeed the destination of the packet. Storing the timestamps of the most recently received packet from each source port at the destination port, we can verify the in-order-delivery of packets. The block transfer fields provide information about block transfers being performed. Knowing the number of remaining packets ,as stored in the **BSE TTL** field, permits block transfers lengths of up to 2^{12}.

4.5 Tests to implement

The verification framework, as mentioned in the earlier section, supports a number of tests to evaluate the reliability of the C64 Crossbar Switch. The hardware under testing must perform to specifications in all testing scenarios, which include regular traffic that would be expected in the operational environment of the C64 Chip, as well as, corner cases that are not likely to occur. Regular traffic scenarios consist of the following traffic patterns:

- One to One - single source port and single destination port
- One to Many - single source port and many destination ports
- Many to Many Set - many source and destination ports with set port selection

- Many to Many Random - many source and destination ports with random port selection
- Block Transfers - all of the above employing block transfers

Traffic patterns, such as one to many, may represent interleaved memory access across on-chip local memory. Other patterns, like many to many, would represent concurrent access by multiple thread units to local memories. Block transfers may represent loads and stores of data structures and program code into local memory. Within the above mentioned traffic patterns, there are some that represent corner cases that would normally not occur. Since these cases are within the specifications of the C64 Crossbar Switch, they should be accounted for.

- Block Transfer of 2 packets, meaning that the BSE bit will be asserted continuously
- Block Transfer of 3 packets, meaning that the BSE bit will be asserted with every second packet
- Block Transfers of more then 7 packets, exceeding the input FIFO's depth of 7 Packets

One of those cases is the short 2-packet Block transfer. A block transfer is specified by the BSE-Bit set in the starting and ending packet of the block. The intermediate packets have the BSE-Bit unset, as any other packet. In case of a 2-packet block transfer, the BSE bit is always set. In the case of a 3-packet transfer, the state of BSE Bit will alternate in every packet sent.

4.6 General Implementation

The packet generation and verification tool is designed to perform various operational and verificational tests on the various design stages of the C64 Crossbar

Switch. It is used with various hardware emulation platforms, such as the software XtremeData's XD1000, DRC's DS1000, IBM's MrsClops FPGA Based Machines and the software based AsapSim logic emulator.

This tool shares the same hardware abstraction layer interface as the accelerated LAST simulator, meaning that all emulation platforms can also run C64 Architecture based programs.

The Packet generation and verification tool's functionality consists of the following tasks:

- C64 Crossbar Packet and Payload generation, processing and analysis
- I/O with Emulated C64 Crossbar Logic
- Graphical Visualization and Analysis

Those tasks are accomplished in the way as described in the Data Flow Diagram in Figure 4.4. Each of those tasks are tightly coupled in the following manner: The verification tool is driven by the OpenGL visualizer which executes the entire application in a cyclic fashion, executing one emulation cycle at a time. For the execution of every clock cycle a new set of Crossbar input packets is generated. Such input packets are injected into the Crossbar emulation hardware. Then output packets are received and analyzed and the visualization is updated according to the new state of the Crossbar.

Figure 4.5 describes the generation of new packets. New packets can only be injected into the Crossbar if the input port's FIFO is not full. Each time a packet is removed from the input FIFO, a Token is generated by the Crossbar logic at the respective input port. The number of outstanding Tokens cannot exceed the depth of the input FIFO, otherwise newly injected packets will be lost. In that case, packets cannot be injected into this port and we must proceed with the next input port.

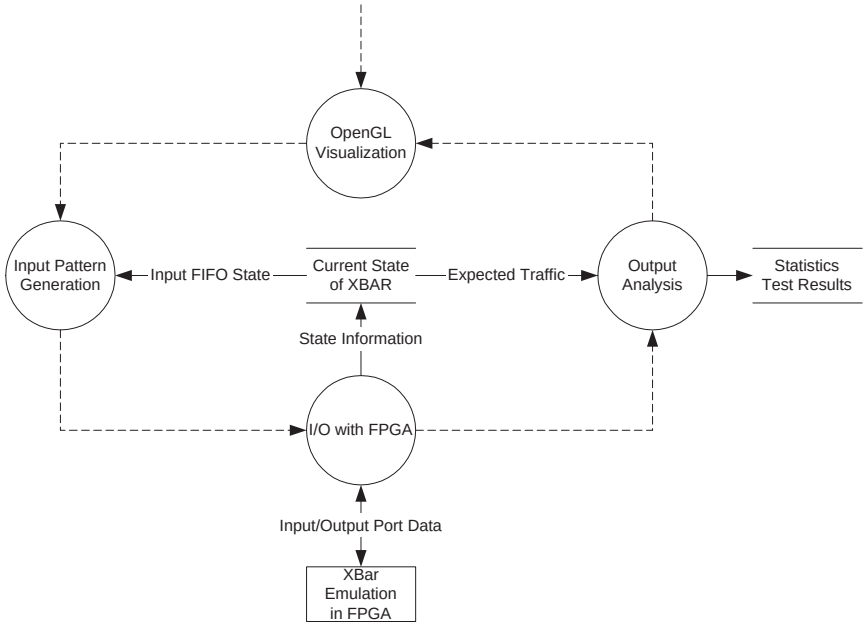

Figure 4.4: Data Flow Diagram of the Verification Tool

According to the actual test set being performed on the Crossbar, there may be no need to create a packet at this particular input port at the current cycle.

The packet must be constructed according to the Table 4.1 so that it can work correctly on the C64 Crossbar switch. Fields that define basic properties of the packet must be set, such as the destination port, virtual channel selection and the block transfer tag. The payload of this packet must be constructed according to the Table 4.2. Fields logically defined within the packet payload specify such properties as Source and Destination Ports of this specific packet, description about a possible block transfer and a timestamp specified by the current emulation cycle. The packet payload contents will obviously carry more useful data in the actual working environment of the Crossbar switch.

Finally this packet is added to the "Guest list" at its respective destination, the number of expected packets from the current input port is increased by one. Each destination port maintains a Guest list, which records the number of expected

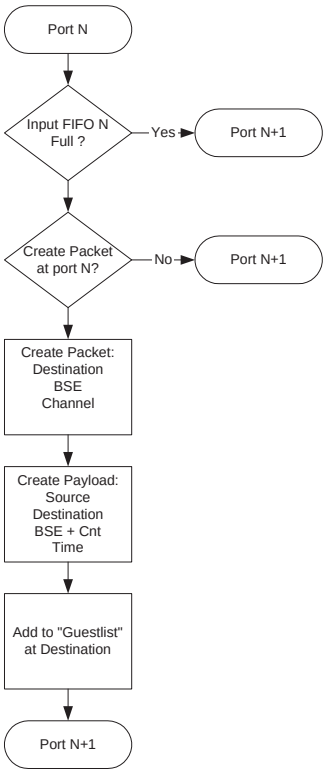

Figure 4.5: Flow Diagram of Input Pattern Generation

packets from each source port that were addressed to it and the timestamp of the most recently received packet from every destination. The goal of this list is to detect lost and duplicate packets. In case of lost packets, the number of expected packets will not match the number of received packets and in case of duplication, we will have a mismatch in the timestamp, each newly received packet must have a timestamp which is larger then the timestamp of the last packet coming from the same source port.

This newly created packet is stored until being injected into actual emulation hardware and execution is continued on with the next input port.

The Figure 4.6 describes the I/O Communication details with the actual emulation of the C64 Crossbar running on dedicated hardware. Prior to the use, the emulation hardware must be initialized and reset to a defined state. Depending

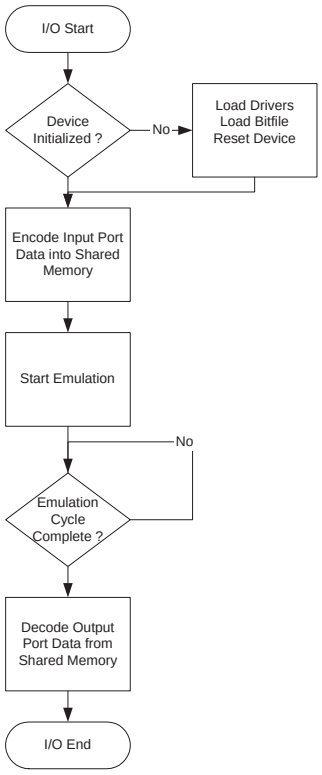

Figure 4.6: Flow Diagram of I/O with Emulation Hardware

on the emulation environment, such things as communication driver initialization and loading of bit files into the FPGA have to be performed. In any case, the Crossbar must be run for a certain length of time with the reset signals set to flush the internal FIFOs and arbitration. Typically this reset takes around 25 emulated cycles.

Communication with the Crossbar running on dedicated hardware usually works via shared memory. Access to a certain memory address results in data movement to and from hardware, thus writing data into a certain memory region will write this data into the input ports of the crossbar switch. The prestored input packets, as created in the Figure 4.5, must be converted into bit-compatible data format prior to being written into the hardware. This data usually includes acknowledgment tokens for received packets.

With these steps completed, the emulation of the Crossbar Switch can be performed for one virtual cycle.

The Processing of the output ports works in a similar fashion as the input ports. Data is usually read from shared memory and decoded into structures describing the output packets. They are almost identical to input packets, but they don't contain the destination port fieled.

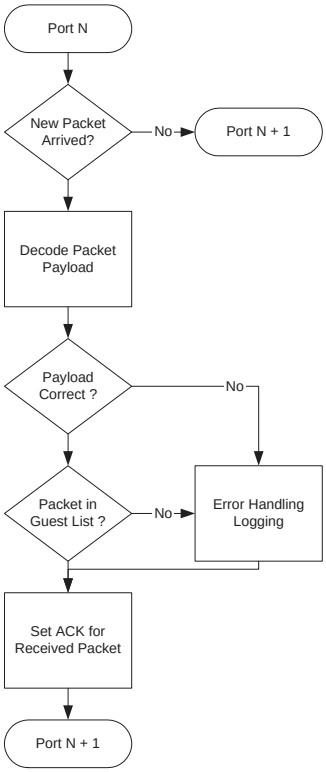

Figure 4.7: Flow Diagram of Output Analysis

Figure 4.7 describes the analysis of the newly received packets. If a packet is received on an output port of the Crossbar, its payload must be decoded to reveal the routing information contained inside the payload. Such data contains the source and destination ports, information about block transfers and a timestamp of the packet injection.

With this information, the packet can be matched against the Guest list to ensure that the packet has arrived correctly. The source port information is used to choose the correct guest list at the destination port. Within the guest list, the destination information must match the destination port where the packet is received, the timestamp must be larger then the one already recorded and the overall number of received messages on this port must be smaller or equal as of messages sent.

Each received packet must be acknowledged by the receiving port, otherwise the Crossbar will not deliver packets. This mechanism is identical to the input port token protocol, the C64 Chip will have FIFOs at the output ports, so each time a packet is removed from the output FIFO, an acknowledgment signal will be sent. The Crossbar logic compares the number of acknowledgments received with the number of packets delivered. The difference can not exceed the depth of the output FIFO.

4.7 Experimental Results

The following tests have been performed on the latest C64 Crossbar Design (Version 9). Each test has been performed over at least 50,000 cycles, meaning that new inputs have been generated for this period of time. The tests were running until completion which means that all packets have been delivered and all BSE Transfers have completed. The following tests have been run on the following platforms:

- XtremeData XD1000 running the MP_V9 version of the crossbar switch in DIMES mode.
- AsapSim Software Simulator running MP_V9 version of the crossbar switch in DIMES mode.

Input Ports	Output Ports	BSE Length	Notes
1	48	1	One to Many without Blocktransfers
1	96	1	One to Many
48	1	1	Many to One
96	1	1	Many to One
48	48	1	Many to Many
48	96	1	Many to Many
96	48	1	Many to Many
96	96	1	Many to Many
48	48	2	Many to Many with BSE Bit continuously set
48	96	2	Many to Many with BSE Bit continuously set
96	48	2	Many to Many with BSE Bit continuously set
96	96	2	Many to Many with BSE Bit continuously set
48	48	3	Many to Many with BSE Bit alternatively set
48	96	3	Many to Many with BSE Bit alternatively set
96	48	3	Many to Many with BSE Bit alternatively set
96	96	3	Many to Many with BSE Bit alternatively set
48	48	32	Many to Many with Blocktransfers exceeding the Input FIFO Depth
48	96	32	Many to Many with Blocktransfers exceeding the Input FIFO Depth
96	48	32	Many to Many with Blocktransfers exceeding the Input FIFO Depth
96	96	32	Many to Many with Blocktransfers exceeding the Input FIFO Depth

Table 4.3: Verification Tests performed on Crossbar Design.

All tests have finished without packets lost, duplication or misrouting. Therefore, we have proven that the DIMES mode design works to the specifications as of Cyclops64 Principles of Operation Manual[22].

4.8 VHDL Testbench Creation

The packet generation and verification tool has proven itself as a great support during the logic debugging phase. It is specifically useful whenever a new logic design is released and needs to be reimplemented as a design logic for iterative emulation. Each of these new designs is usually simulated using the ModelSim[23] software simulator, before it is synthesized to run on the FPGA accelerator platforms.

The reason therefore is pretty obvious: A synthesis run usually takes hours, depending on the actual accelerator platform and settings selected. Once the logic is loaded onto the accelerator, the developer has hardly any debugging capabilities. In fact he can be considered lucky if the system crashes without corrupting the entire file system.

Therefore, a gate-level software simulator is a great tool for logic debugging and validation. However, this approach requires the use of testbenches - they must be created by the user of the simulator. The manual creation of VHDL testbenches is a tedious task, in the case of the C64 Crossbar switch, a large amount of input signals has to be set and reset for each simulation step. In addition, the testbench requires logic module instantiation and initialization.

A simple testbench can be created by hand, however even in that case the amount of VHDL code is quite large. This shows a great possibility for automation: the testbenches can be created on the fly running the verification tool on already existent crossbar logic and simply recording the traffic that is occuring between the emulation and the tool into the testbench. This automated approach proves itself

especially practical, because it uses feedback from the crossbar logic emulation. It can detect timing differences, found between the testbench and the simulated logic.

The implementation is quite simple, given the underlying framework. During device initialization phase, the tool writes instantiation and initialization code into the testbench. In the I/O stage, the tool writes register input and output port contents. The testbench is finalized when the execution of the tool is finished.

Chapter 5

RELATED WORK

At the time of creation of this document, work is being performed at CAPSL and ETInternational on the verification of the IBM Cyclops64 Chip. Emulation of this architecture is performed with the AsapSim Software Simulator and the MrsClops Emulation Engine[8].

Currently, the entire Cyclops64 Chip can be emulated by the MrsClops Emulation Engine. At 120,000 virtual cycles/second, the performance is equivalent with the functionally accurate "FAST" simulator. However this emulation is latency accurate like the "LAST" simulator and the "SEmulator". The emulation speed of MrsClops, running the Cyclops64 Chip, surpasses all other latency accurate simulators. This is true due to the enormous amount of available resources for emulation of all chip sub modules, also for being self-contained and highly optimized. In fact, the C64 emulation on MrsClops uses the same iterative emulation logic for the crossbar switch as the "SEmulator" does.

Chapter 6

FUTURE WORK

A large quantity of valuable experience and knowledge has been gained from work on the FPGA-based Accelerator platforms. This experience can be employed towards other uses of those plaforms, such as: financial applications, bioinformatics, algorithm development and implementation, image processing and finally other types of logic emulation.

6.1 Existent Design Improvements

So far, the C64 Crossbar design has been implemented on various FPGA-based coprocessing platforms successfully. However, accelerator vendors are constantly releasing new and improved versions of their devices, software and hardware libraries. Those improvements usually reflect increased performance, but certainly require more work then a simple resynthesis of the design to benefit. New software drivers are sometimes not compatible with the old designs, so work always needs to be done to keep up with the pace.

6.2 Algorithm Development

Algorithm Development in the field of Reconfigurable Computing has long been dominated by complicated Custom logic designs for specific applications, such as those used for logic emulation in this document. With the recent advances of products which can generate code for FPGA accelerator platforms automatically,

such as ImpulseC[24] and Mitrion[25], FPGA-based Acceleration became much more accessible to end users.

Those products can take existent applications and algorithms, written in higher programming languages such as C, and accelerate them by the means of calculating performance-intensive program parts in hardware. This work certainly does require user input to choose the correct application of the tool, however the user is often not required to have in-depth knowledge of logic design. The tool can even completely generate and synthesize logic designs and software counterparts for certain platforms automatically.

It must be noted however, that the resources of the FPGAs are usually limited. Even the largest chips will have problems with a large amount of floating point arithmetic operations. It even goes so far, that alot of mathematical functions may not be available to the user automatically.

Despite those limitations, algorithm development can greatly benefit from FPGA-acceleration due to the massively parallel and pipelined data manipulation capabilities. This is specifically useful for streaming data processing, improving the performance by multiple levels of magnitude versus a software implementation.

6.3 MiniMrsClops

The Cylcops64 Architecture is currently under emulation by the MrsClops Emulation Engine[8]. This machine consists of 30 FPGAs, Altera Stratix 2S90, and 20GByte of Memory. This machine is currently capable of emulating an entire C64 Chip at around 120,000 cycles/second.

In contrast, the XtremeData XD1000[9] is equipped with one Altera Statix 2S180 FPGA and 4 GBytes of memory. Considering that MrsClops is using only 20 FPGA Chips for data processing and the chips are smaller by a factor of 2, we do have a very high performance platform for logic emulation with XD1000. It would be perfectly suitable for building an efficient emulation platform of smaller scale.

We should also note, that we could use the existing stack code generators such as AsapSCG to emulate any kind of logic on this platform.

Appendix A

SEMULATOR AND VERIFICATION UTILITY

A.1 SEmulator

The SEmulator is an extension to the latency accurate simulator LAST. Therefore, the use is identical. We should consult the C64 Toolchain Manual for further instructions on how to use the simulator.

```
./last hello_world.bin
               Load @ address: 0000000000000000, length: 0000000000008ec0
               ...
               Initialize @ address: 000000000000ae90, length: 0000000000000000
Cyclops says... Hello world!
SEMULATOR used for: 11.004125 seconds at a rate of 1164.654164 cycles/second
```

With the option "-c", the emulation hardware is employed for the acceleration of the simulator.

```
./last -c hello_world.bin
crossbar reset done!
FPGA Initialized after: 0.874599 seconds
               Load @ address: 0000000000000000, length: 0000000000008ec0
               ...
               Initialize @ address: 000000000000ae90, length: 0000000000000000
Cyclops says... Hello world!
Crossbar used for: 1.163554 seconds at a rate of 9905.857399 cycles/second
SEMULATOR used for: 2.121563 seconds at a rate of 5432.787054 cycles/second
```

A.2 Verification Utility

The Verification Utility is designed to provide functional verification of the Cyclops64 Crossbar Switch. This task is accomplished by creating traffic on the crossbar switch and verifying the outputs.

The execution of this utility consists of three phases:

- Phase 1: The program is creating new packets and block transfers according to the rules specified in the input.
- Phase 2: The program is finishing existing block transfers, no new transfers are created.
- Phase 3: The program is receiving remaining packets.

The programs can create new packets according to the rules specified in the program parameters.

```
./stress_test "Length" "BSE Length" "Input Ports" "Output Ports"
./stress_test 100 5 10 10
```

The first parameter specifies how long the program will be creating new transfers. The second parameters specifies the length of the block transfers. The third parameter specifies the number of input ports. The last parameter specifies the number of output ports.

Appendix B

LOADING OF BITFILES ONTO FPGA ACCELERATOR PLATFORMS

This section will introduce the loading of Bit files onto the XD1000 and DS1000 Accelerator Platforms: Upon being powered on, a regular FPGA chip does not contain a logic design in a predictable state. In the case of an FPGA Coprocessing platform, the FPGA is directly connected to the HyperTransport link and therefore must contain a valid logic design. Otherwise, the system can not initialize the HT link upon boot up.

The task of loading the image into the FPGA is performed by additional components. The image is either stored within a FLASH memory chip, or can be loaded via the JTAG interface from an additional machine.

This task is accomplished with the software "Quartus Programmer" for Altera FPGA or "iMPACT" for Xilinx chips.

During the process of loading the FPGA, we will have unknown signal states on the pins, also those pins connected to the HyperTransport link. To avoid an unknown bus state, which will certainly result in a system lockup, we need to halt the machine prior to loading the image onto the FPGA.

```
$sudo halt
```

B.1 XtremeData: Altera FPGA

After we have verified that the platform is halted, we do not turn off the main power, but proceed to the next step.

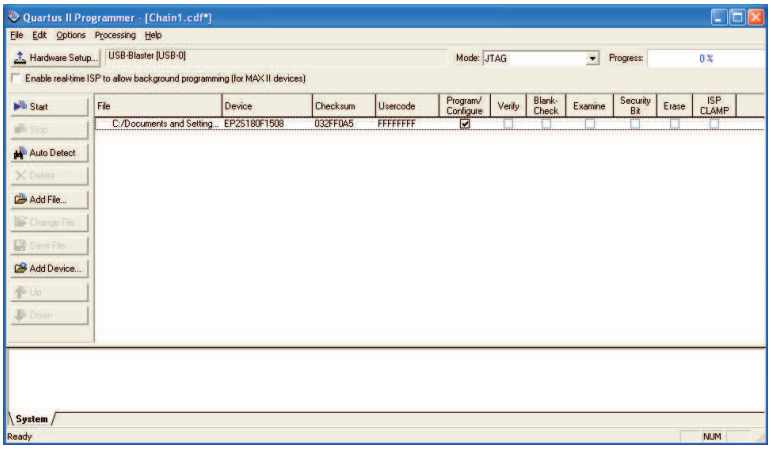

Figure B.1: Altera Programmer: Main Screen

With the program started as shown in Figure B.1, we must make sure that the correct programming interface is selected: Hardware Setup.

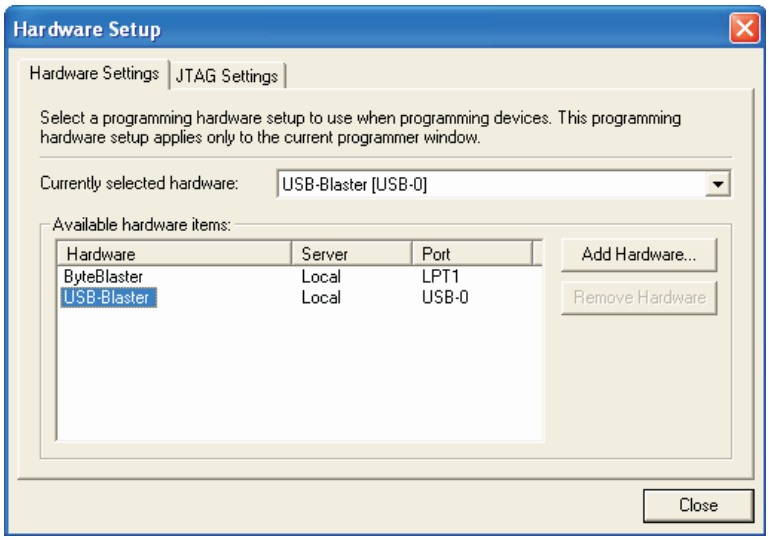

Figure B.2: Altera Programmer: Device Selection

We must select the USB-Blaster programming interface in Hardware Setup, as shown on Figure B.2.

With "Add File..." we must select a bit-file that will be programmed onto the FPGA and also select the "Program/Configure" checkbox. By pressing the "Start"

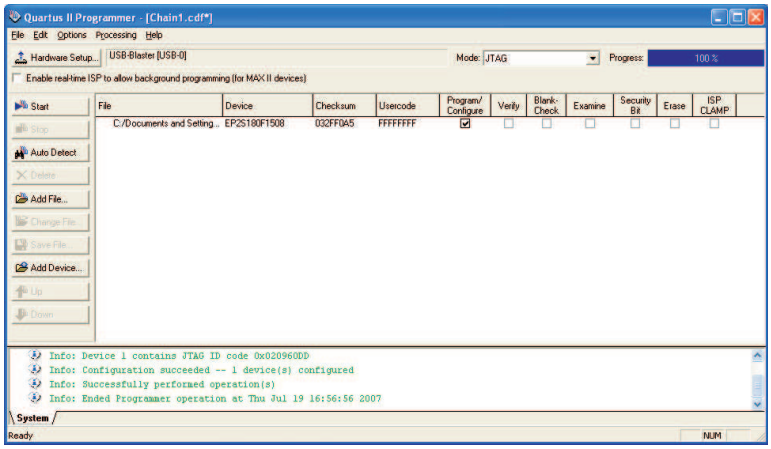

Figure B.3: Altera Programmer: Programming Complete

button we can program the FPGA as shown on Figure B.3

The FPGA Accelerator will reboot and start beeping. It must be manually reset until the machine starts a regular booting procedure.

B.2 DRC: Xilinx FPGA

The loading of bit-files onto the DRC platform is essentially identical to the XtremeData. The machine must be halted as well, but not turned off. The software "iMPACT" from the Xilinx package is used for this task. Upon startup of the

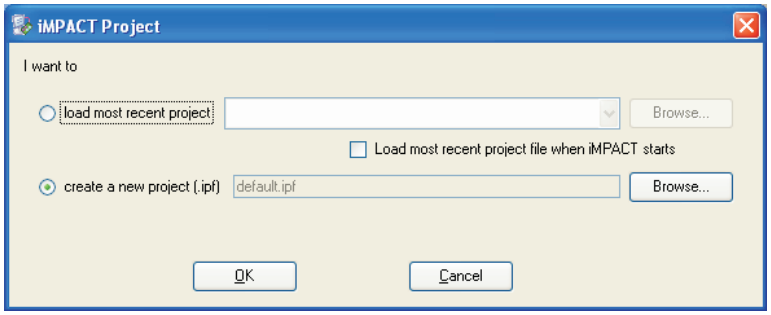

Figure B.4: Xilinx iMPACT: Startup

program, a new project will be created as seen in Figure B.4.

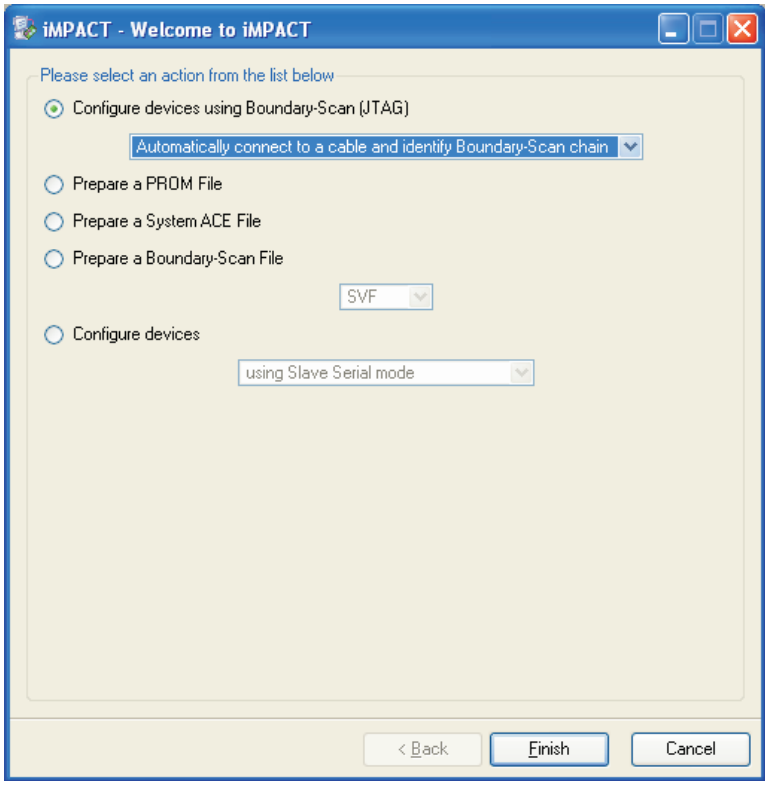

Figure B.5: Xilinx iMPACT: Boundary Scan

The software will perform a Boundary Scan of the JTAG chain, it will detect all devices connected to the programmer. Figure B.5 shows this screen. We must perform this scan to continue.

As shown on Figure B.6, we have to select the device that we want to program, Xilinx "xc4vlx60", right click it and select the appropriate bitfile. The programming software is now ready to load the bitfile. We just have to right click the device again, choose "Program Device" and the FPGA platform will automatically reboot after it is programmed.

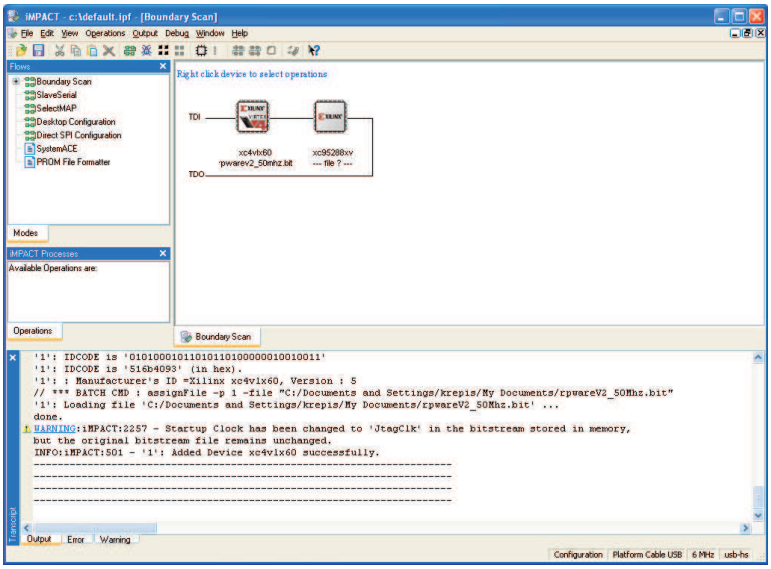

Figure B.6: Xilinx iMPACT: Device Selection and Programming

BIBLIOGRAPHY

[1] W. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," *ACM SIGARCH Computer Architecture News*, pp. 20–24, March 1995.

[2] C. A and M. S. R, "The power of functional scaling: beyond the power consumption challenge and the scaling roadmap," *Circuits and Devices Magazine*, pp. 27–35, February 2005.

[3] H. Sakane, L. Yakay, V. Karna, C. Leung, and G. R. Gao, "Dimes: An iterative emulation platform for multiprocessor-system-on-chip designs," in *IEEE International Conference on Field-Programmable Technology (FPT'03)*, Tokyo, Japan, Dec. 2003.

[4] H. Sakane, L. Yakay, and V. Karna, "DIMES/P hardware technical manual," University of Delaware, CAPSL Tech. Note 12, 2003.

[5] M. Denneau, private communication, 2002.

[6] Y. P. Zhang, T. Jeong, F. Chen, H. Wu, R. Nitzsche, and G. R. Gao, "A study of the on-chip interconnection network for the ibm cyclops64 multi-core architecture," in *20th International Parallel and Distributed Processing Symposium (IPDPS2006)*, 2006.

[7] http://www.alpha-data.com/, "Alpha data: Fpga based solutions for high end applications."

[8] International Business Machines Corporation and ET International, "Mrsclops emulation engine."

[9] http://www.xtremedatainc.com/, "Xtremedata inc.: Computing redefined."

[10] http://www.drccomputer.com/, "Drc computer: Acceleration technologies for high-performance computing."

[11] Fei Chen, "Latency accurate simulation testbench, 2005."

[12] ——, "Asapsim,2006."

[13] Juan del Cuvillo, Weirong Zhu and Ziang Hu and Guang R. Gao, "FAST: A Functionally Accurate Simulation Toolset for the Cyclops-64 Cellular Architecture," in *Workshop on Modeling, Benchmarking and Simulation (MoBS'05) of ISCA'05*, Madison, Wisconsin, June 2005.

[14] Juan del Cuvillo, Weirong Zhu, Ziang Hu and Guang R. Gao, "Towards a Software Infrastructure for Cyclops-64 Cellular Architecture," in *HPCS 2006*, Labroda, Canada, June 2005.

[15] W. J. Dally and B. Towels, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[16] M. K. Chen, X.-F. Li, R. Lian, J. H. Lin, L. Liu, T. Liu, and R. Ju, "Shangri-la: Achieving high performance from compiled network applications while enabling ease of programming," in *Proceedings of ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation (PLDI05)*, Chicago, Illinoi, June 2005.

[17] Y. P. Zhang, "A study of architecture and performance of ibm cyclops64 interconnection network," Master's thesis, Univ. of Delaware, Newark, DE, Summer 2005.

[18] J. D. Cuvillo, W. Zhu, Z. Hu, and G. R. Gao, "Fast: A functionally accurate simulation toolset for the cyclops64 cellular architecture," in *Procedings of the 32nd Annual International Symposium on Computer Architecture (ISCA'05)*, Madison, Wisconsin, June 4 2005.

[19] http://www.altera.com/literature/manual/mnl_avalon_spec.pdf, "Memory-mapped interface specification, 2007."

[20] http://www.hypertransport.org/, "Hypertransport consortium."

[21] Y. P. Zhang, T. Jeong, F. Chen, H. Wu, R. Nitzsche, and G. R. Gao, "Performance analysis of interconnection network of cyclops-64 chip architecture," in *CAPSL Technical Memo 60*, 2005.

[22] I. B. M. Corporation, *64-Bit Cyclops Principles of Operation*. IBM Internal Documentation, 2004.

[23] http://www.model.com, "Modelsim - simulation and debug environment for complex asic and fpga designs."

[24] http://www.impulsec.com, "C programming tools for fpga platforms."

[25] http://www.mitrion.com/, "The mitrion platform by mitrionics."