2007

A low-cost processor-based logic emulation system using FPGAs

Marwan Kanaan
University of Windsor

A Low-Cost Processor-Based Logic Emulation System Using FPGAs

by

Marwan Kanaan

A Thesis
Submitted to the Faculty of Graduate Studies
through Electrical and Computer Engineering
in Partial Fulfillment of the Requirements for the
Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada
2007

Your file Votre référence
ISBN: 978-0-494-34930-4
Our file Notre référence
ISBN: 978-0-494-34930-4

Abstract

Logic emulation systems are used to verify the functionality of logic designs targeted for integrated circuit implementation. In this thesis, the design and implementation of a low-cost processor-based logic emulation system is presented. It contains multiple processors interconnected together and packaged in one emulation engine. It is capable of emulating combinational and sequential logic at relatively high speeds of 187 KHz or more, in real operating environments and with predictable compile time. The implementation was done on an FPGA to reduce cost. The proposed system is scalable to a multi-FPGA system where several of these identical FPGAs could be connected together to increase the logic capacity of the system.

The architecture and operation of the emulator is first described. Architecture exploration experiments were conducted in order to choose suitable values for different architecture parameters for implementation on the target FPGA. The design was implemented on an Altera Stratix FPGA. A four-bit multiplier was emulated to verify correct operation of the proposed emulation system.

To my family for their unending love and support.

Acknowledgments

I thank God Almighty that this thesis has been completed. I stand here humbly at the end of this accomplishment confident that I would not have been able to do it without His support and help. I ask Him, once and again, to continue to shed light on each and every path I take.

I would like to thank my supervisor, Dr. Mohammed Khalid, for his support, guidance and determination throughout the course of this work. I am deeply and forever grateful for all the invaluable efforts he made. I would also like to thank Dr. Abdel-Raheem and Dr. Zhang for sitting on my committee and reviewing my thesis and Dr. Kar for sitting in as Chair of Defense.

Thanks to my family for all their love, support and advice. To my mom and dad, thanks for all the encouragements, prayers, help and patience. I am what I am today largely because of my parents and for that I am thankful. To my sister and my grandmother, I am thankful for all the encouragements, prayers and care.

Thanks to all my friends and fellow graduate students at the University of Windsor. Jay and Ian, I'll never forget all the times we spent together. It was wonderful to have you as officemates. My thanks also go to the current and former members of our research group: Amir, Kevin, Raymond, Omar, Junsong, Aws, Hongmei and

Thuan. I would also like to thank my friends Harb, Ali, Bahador, Payman, Ashkan, Mahzad, and Amr for their friendship over the past two years.

Funding and technical support for this research was provided by the Natural Sciences and Engineering Research Council (NSERC) of Canada, the University of Windsor, the Canadian Microelectronics Corporation (CMC) and Altera Corporation. Their contributions are gratefully acknowledged.

Contents

Abstract **iv**

Dedication **v**

Acknowledgments **vi**

List of Figures **xii**

List of Tables **xiv**

List of Abbreviations **xv**

List of Symbols **xvi**

1 Introduction **1**

1.1 Thesis Objectives 3

1.2 Thesis Organization 4

2 Background and Previous Work **5**

2.1 Design Verification 5

2.1.1 Formal Verification 6

2.1.2 Software Simulation 7

2.1.3 Hardware-Accelerated Simulation 8

2.1.4 Rapid Prototyping 8
2.1.5 Logic Emulation 9
2.2 Logic Emulation Systems 10
2.2.1 FPGA-Based Emulation Systems 12
2.2.1.1 Field Programmable Gate Arrays 12
2.2.1.2 Architecture and CAD for FBEs 17
2.2.2 Processor-Based Emulation Systems 19
2.2.2.1 Emulation Processors 19
2.2.2.2 Architecture and CAD for PBEs 20
2.2.3 Commercially Available Logic Emulators 23

3 System Architecture and Operation 24
3.1 Introduction and Motivation 24
3.2 Levels of Hierarchy 25
3.3 Logic Emulation Processor 27
3.3.1 Control Store 29
3.3.2 Data Stacks 31
3.3.3 Logic Element 32
3.3.4 Architecture and Operation 33
3.4 Memory Emulation Processor 36
3.4.1 Control Store 37
3.4.2 Memory Store 39
3.4.3 Release Memory Word Unit 39
3.4.4 Capture Memory Word Unit 40
3.4.5 Architecture and Operation 40
3.5 Emulation Module 42
3.5.1 Module Level Routing Switch 43
3.5.2 Sequential Filler 44

3.5.3 Architecture and Operation 44
3.6 Emulation Chip 45
3.6.1 Chip Level Routing Switch 45
3.6.2 Architecture and Operation 46
3.7 Emulation Engine 46
3.7.1 Multi-FPGA System 46
3.7.2 Scalability Issues 48

4 Architecture Exploration and Implementation Results 50
4.1 Implementation Target 50
4.1.1 Altera Stratix FPGA 51
4.2 Architecture Exploration 52
4.2.1 Key Parameters 52
4.2.2 Effect of Changing Parameters 53
4.2.2.1 Effect of Changing Lookup Table Size 53
4.2.2.2 Effect of Changing Number of Emulation Steps . . . 56
4.2.2.3 Effect of Changing Total Number of Outputs 58
4.2.2.4 Effect of Changing Memory Word Size 64
4.2.3 Choice of Parameters 66
4.3 Implementation Results 66
4.3.1 Logic Processor 67
4.3.2 Memory Processor 68
4.3.3 Emulation Module 68
4.3.4 Emulation Chip 69
4.4 Implementation Estimates for Emulation Engine 71
4.5 Emulation Example 72
4.5.1 Four-Bit Multiplier 72
4.5.2 Scheduling and Implementation 73

5 Conclusion and Future Work **77**

5.1 Research Contributions 77

5.2 Comparisons with Other Systems 78

5.3 Future Work 79

References **80**

VITA AUCTORIS **83**

List of Figures

2.1 Logic Emulation System 11

2.2 A Generic FPGA Architecture 14

2.3 Internal Structure of a Logic Element 15

2.4 Internal Structure of a Lookup Table 15

2.5 Multi-FPGA System 17

2.6 CAD Flow for FBEs 18

2.7 Processor-Based Emulation System 21

2.8 CAD Flow for PBEs 22

3.1 Emulation Design Cycle 26

3.2 System Hierarchy 28

3.3 Logic Emulation Processor 29

3.4 Logic Processor Control Word Fields 30

3.5 Logic Element 33

3.6 Operation of the Logic Processor 35

3.7 Memory Emulation Processor 37

3.8 Memory Processor Control Word Fields 38

3.9 Operation of the Memory Processor 41

3.10 Emulation Module 43

3.11 Emulation Chip 45

3.12 8-Way Mesh MFS 47
3.13 Fully Connected MFS 47
3.14 Clock Duty Cycle 48

4.1 LUT Size vs. Area in LP 54
4.2 LUT Size vs. Memory Bits in LP 54
4.3 LUT Size vs. Speed in LP 55
4.4 Number of Emulation Steps vs. Area in LP 57
4.5 Number of Emulation Steps vs. Memory Bits in LP 57
4.6 Number of Emulation Steps vs. Speed in LP 58
4.7 Number of Emulation Steps vs. Area in MP 59
4.8 Number of Emulation Steps vs. Memory Bits in MP 59
4.9 Number of Emulation Steps vs. Speed in MP 60
4.10 Number of Total Outputs vs. Area in LP 61
4.11 Number of Total Outputs vs. Memory Bits in LP 61
4.12 Number of Total Outputs vs. Speed in LP 62
4.13 Number of Total Outputs vs. Area in MP 62
4.14 Number of Total Outputs vs. Memory Bits in MP 63
4.15 Number of Total Outputs vs. Speed in MP 63
4.16 Memory Word Size vs. Area in MP 64
4.17 Memory Word Size vs. Memory Bits in MP 65
4.18 Memory Word Size vs. Speed in MP 65
4.19 Module Level Routing Switch 70
4.20 Operation of the Four-Bit Multiplier 73

List of Tables

3.1 Logic Processor Control Word Fields Description 30

3.2 Memory Processor Control Word Fields Description 38

4.1 Multiplier Scheduling for Processors 0-3 74

4.2 Multiplier Scheduling for Processors 4-7 76

List of Abbreviations

Abbreviation	Definition
ASIC	Application-Specific Integrated Circuit
CAD	Computer Aided Design
DUT	Design Under Test
EDA	Electronic Design Automation
FBE	FPGA-Based Logic Emulation System
FF	Flip-Flop
FPGA	Field Programmable Gate Array
IC	Integrated Circuit
I/O	Input/Output
LE	Logic Element
LP	Logic Emulation Processor
LUT	Lookup Table
MFS	Multi-FPGA System
MP	Memory Emulation Processor
MUX	Multiplexer
PBE	Processor-Based Logic Emulation System
PCB	Printed Circuit Board
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VLSI	Very Large Scale Integration

List of Symbols

Symbol	Definition
M	Lookup table size.
N	Total number of emulation steps in one design cycle.
P	Total number of outputs of all processors in one module.
Q	Memory word size.
R	The number of logic emulation processors in one emulation module.
S	The number of memory emulation processors in one emulation module.
T	The number of emulation modules in one emulation chip.

Chapter 1

Introduction

In this day and age, electronic devices, ranging from cell phones to personal computers, play an essential role in our daily lives. Designing such devices and verifying their functionality could be an excruciating task for engineers if the necessary tools are missing. These tools, known as *Computer-Aided Design* (CAD) tools, have long been a vital part of the research in *chip design* where considerable research efforts have been made and are always being carried out to ensure that designers have the most reliable and efficient of these tools.

One task of these design tools is *design verification*, the process where the functionality of an electronic device is validated. In the past three decades verification has become one of the most crucial parts of the design cycle. Its importance is due to the fact that it is absolutely necessary for designers to make sure that their design is correct prior to fabrication. A simple error discovered after production is very expensive to fix thus potentially costing the manufacturing company millions of dollars in losses [12]. In addition to that, the increase in chip size [26] and the need to reduce time-to-market require more capable and robust design verification tools to cope with the growing industry.

Several design verification tools are available on the market today. The most effective of these tools are *logic emulators.* A logic emulator is a design verification tool where a reprogrammable system imitates the functional behavior of a logic design. The system can be programmed to act exactly as a desired chip and thus gives the user the ability to check the logic design in real time circuit environments and conditions before manufacturing [9, 17]. By doing so the designer could verify the *Design under Test* (DUT) by running tests that the real chip would have to pass. This process could be repeated several times and in this manner most errors could be identified and corrected. Logic emulators give the user the ability to catch almost all functional errors in a logic design, however we should note that timing requirements and constraints cannot be verified using this tool.

Currently there are two types of logic emulators, *FPGA-Based Logic Emulation Systems* (FBEs) and *Processor-Based Logic Emulation Systems* (PBEs). In FBEs, several reprogrammable chips known as *Field Programmable Gate Arrays* (FPGAs) are connected together to emulate the functional behavior of a logic design. While FBEs are considered to be low-cost and efficient emulators they face a major problem when it comes to their CAD tools. The second type of logic emulation systems are processor-based emulators where multiple *emulation processors* are packaged together in an *emulation engine* capable of emulating a logic design of significant size and complexity. PBE systems are considered to be an efficient verification tool and do not suffer from problematic CAD tools, however they are implemented on custom made chips and are therefore very expensive.

The motivation behind this thesis is to design a logic emulation system that would combine the advantages of FBEs and PBEs; a system that would be as efficient as a PBE and as inexpensive as an FBE. To achieve this goal, one solution would be to implement the processor-based emulation system on an FPGA, a reprogrammable chip known for its low cost. In this thesis we explore the architecture of such a system. The proposed emulation system would run at relatively high speeds and be capable of emulating designs of significant logic capacity and complexity.

1.1 Thesis Objectives

The main objective of this thesis is to explore FPGA design and implementation of a low-cost processor-based emulation system. To achieve this goal, an architecture of a logic emulation system is explored and implemented. This thesis has the following objectives:

1. Explore the architecture of a processor-based logic emulation system that can be implemented on an FPGA.
2. Implement the emulator by targeting a specific FPGA.
3. Ensure that the PBE is scalable. Several FPGAs should be able to connect together in a multi-FPGA system to increase logic capacity.
4. Verify the system by emulating a logic design.

To satisfy the first objective, an architecture of a processor-based emulator was explored in terms of cost, functionality, area and speed upon which key design parameters were chosen accordingly. To satisfy the second objective, an Altera Stratix FPGA was targeted. The implementation was tuned specifically for this FPGA. To address the third objective a scalability study was done and results are presented. For the fourth objective, a four-bit multiplier was designed, scheduled and emulated on the designed system to verify its correctness.

1.2 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 discusses the background and previous work done on the subject. Chapter 3 presents the architecture and operation of the proposed system and its various components. Chapter 4 presents the architectural exploration and implementation results. Lastly, chapter 5 concludes with some discussion of possible future work.

Chapter 2

Background and Previous Work

This chapter presents the background for the research done in this thesis and briefly describes related previous work. The first section begins by defining design verification and its significance in today's industry. It then briefly discusses the five major types of verification tools available on the market today, along with their main advantages and disadvantages. The second section of this chapter focuses on logic emulation systems. A brief introduction is given with a discussion of the two main types of logic emulators. The chapter concludes by presenting some examples of commercially available logic emulation systems.

2.1 Design Verification

Design verification is the process whereby a logic design is checked for functional errors. In this part of the design cycle, the functional behavior of a logic design is validated. As chips increased in size and complexity, design verification tools became more complicated and required more efforts. Today the design verification process alone may consume up to 60% of the whole design cycle in terms of time, resources and manpower making it the bottleneck for design development [14, 29, 16].

Over the past few decades design verification has evolved from simple mathematical techniques that were carried out by manual calculations to test the validity of small designs to multi-million dollar machines capable of verifying a design consisting of millions of gates.

There are several types of design verification tools available on the market today, each with advantages and disadvantages, and in general they can be categorized into five major groups:

1. Formal verification
2. Software simulation
3. Hardware-accelerated simulation
4. Rapid prototyping
5. Logic emulation

In the following sections we introduce each type of these tools and we describe their capabilities and weaknesses.

2.1.1 Formal Verification

In *formal verification* the designers prove the validity of a logic design using formal methods; all or part of the design is modeled in a mathematical framework after which the designer would solve the mathematical equations to verify the correctness of the design [20, 23].

The main advantage of formal verification is that it is highly effective in catching design errors. Since it relies on a mathematical approach, formal verification is almost completely guaranteed to find any functional error. The main disadvantage, however, is that it is very time consuming. Proving the validity of a design using formal verification requires extended periods of time from expert designers and hence it is impractical to use in large IC designs [20].

Despite that, formal verification is probably the most comprehensive of all verification tools but due to its heavy cost in terms of time it could be only used in small circuits or in specific parts of large designs.

2.1.2 Software Simulation

Software simulation is without doubt the most popular and widely used verification tool [29]. It is widely available, inexpensive and above all user friendly. In simulation the Design under Test (DUT) is represented in software models which the designer would test for correctness by applying input test vectors to them then reading the outputs to check for errors [27].

Simulation has many advantages over other verification tools. It is generally easy to use; the user's task is only to choose the input vectors after which he or she has to wait for outputs. Simulation is also inexpensive since it requires only a software platform. It provides the user with high visibility and flexible debugging; the user can observe each signal traversing the design to check for errors. But probably the most important advantage is the flexibility that simulation provides. Since it is software-based, changing and modifying parts or all the design is relatively easy to do.

Software simulation also has several disadvantages. The degree of accuracy of the verification process depends heavily on the user's choice of input test vectors. The choice of these vectors should be comprehensive enough to cover all aspects of the design or else some functional behavior of the design might be missed and go unchecked for errors. A second major disadvantage, and perhaps the most important, is that simulation is relatively slow [27]. Because of the sequential nature of software processing, simulating a large design, especially in its real world operating environment, could literarily take days or even weeks.

2.1.3 Hardware-Accelerated Simulation

Hardware-accelerated simulation shares the same basic principles with software simulation. The motivation behind this method was to simply overcome the slow speed problem of software simulation. A logic design is still modeled in software, however this time the simulation is executed on custom made hardware rather on a software platform running on a single processor. The processing power achieved by hardware accelerates the simulation and gives the advantage of faster simulation speed [27, 22, 31].

Despite the speed acceleration that this method provides it still suffers from the same problem that software simulation suffers from: the degree of accuracy of the verification process still depends on the designer's choice of inputs. In addition, the speedup provided by the hardware accelerators is still limited by the communication media between the host computer and the hardware accelerator itself [29]. The time needed for the input vectors to be generated and the output signals to be read is still restricted by the connective devices.

2.1.4 Rapid Prototyping

As the name suggests, in *rapid prototyping*, a custom made prototype of the logic design is built by the designer to verify the functionality of a design [19, 8]. Usually a custom multi-FPGA system is built for each prototype. In such systems, the FPGAs are programmed to imitate the functional behavior of the design and permanent connections are established between them to ensure design connectivity.

The main advantage of prototyping is speed. Since the whole design is implemented in hardware, rapid prototyping achieves the fastest verification speeds of all verification tools. In addition to speed, another advantage of rapid prototyping is that it provides the user with the capability of testing the prototype in its real operating environment. Rather than using input test vectors to test the system, real inputs are supplied from the surrounding target system, thus giving the user higher confidence in the validity of the design.

Nevertheless, rapid prototyping has a major disadvantage when it comes to cost. Once a prototype is built for a specific design it cannot be modified to implement another design; in other words it is a throw-away effort after the user is done with only one design. This basically means that the system is not reusable making its cost very high.

2.1.5 Logic Emulation

The newest type of design verification and the most efficient one is *logic emulation.* A logic emulator is a reprogrammable system that can be programmed and reprogrammed to emulate logic designs at relatively high speeds. Once programmed, an emulator would function exactly as the desired hardware without the need for fabrication. In doing so, an emulator would be combining the advantages of software and hardware together; because it is reprogrammable it is as flexible as software and since it utilizes hardware it achieves very high speeds. However, it is important to note that although an emulator is programmable it is still quite different from software simulation. The hardware here is not being modeled in software; in fact, it is actually implemented on reprogrammable hardware.

Compared to other verification tools logic emulation has many advantages. It is as flexible as software simulation yet much faster and although it is not as fast as rapid prototyping yet it is not as costly because it is reprogrammable. But the main advantages of the logic emulation would have to be in-circuit emulation, the capability to function like an actual IC chip in real world operating environments. After being programmed an emulator could be connected to a target system and tested, giving the user the opportunity to verify the operation before fabrication. This removes the need to generate input test vectors, and like rapid prototyping, gives the user higher confidence in his or her design by relying on real inputs supplied by the target system.

Logic emulation still has some disadvantages, mainly its cost. Logic emulation systems are still very expensive and can only be afforded by big companies. Designing and manufacturing such a system is still a costly process.

Since it is the main focus of this research, in the following sections of this chapter we describe the main types of logic emulators and we discuss their advantages and disadvantages in detail.

2.2 Logic Emulation Systems

A typical logic emulation system, shown in Figure 2.1, contains three main elements:

1. Emulation engine (or emulator for short)
2. Emulation support facilities
3. Interface circuitry

An *emulator* is basically a reprogrammable hardware system that can implement any logic design. This reprogrammable system could be a set of FPGAs or emulation processors connected together. Some details about the architecture of the emulator would be discussed in later sections of this chapter.

Emulation support facilities include a host computer along with an emulation compiler [15]. The task of the host computer is to act as an interface between the user and the emulator. The compiler is responsible for converting the DUT supplied by the user into a bit stream to be downloaded to the programmable hardware. The

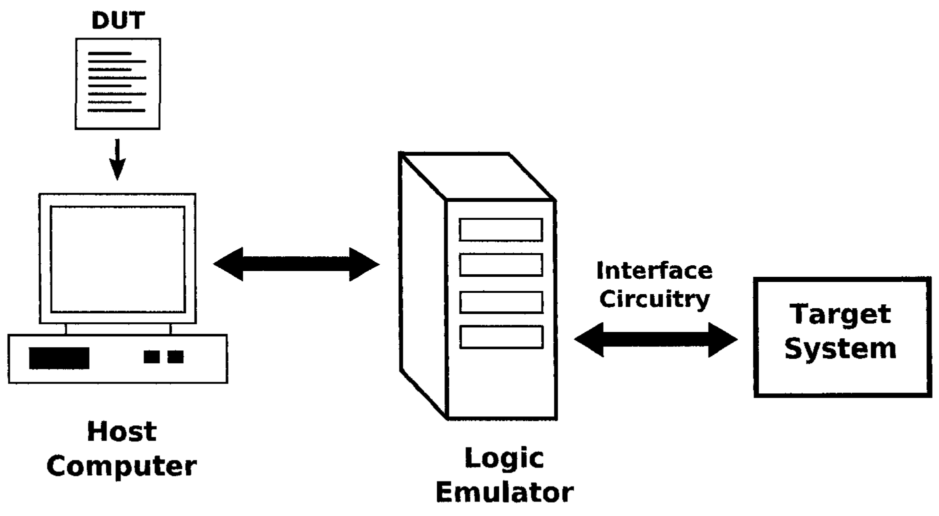

Figure 2.1: Logic Emulation System

emulation support facilities might also include some other components like a Data Capture Unit used to read the outputs from the emulator and relay them to the user.

The *interface circuitry* of the logic emulation system is used to connect the emulator to a target system to perform in-circuit emulation.

To use the system the user supplies the compiler with a logic design (e.g. written in a hardware description language). The compiler compiles the design and generates a bit stream which can then be downloaded onto the emulation engine. At this time an emulator is working exactly as the desired chip would work. Using the interface circuitry the user could connect the emulator to a target system and test the design. This is the main advantage of emulation: the ability to test a design in its typical operating environment with real inputs.

To illustrate this consider the example where the designers are verifying the functionality of a video card for a personal computer. In this case the DUT is the logic design for the video card and the target system is the personal computer. To perform in-circuit emulation, the designers would program the emulator with the design of the video card and connect it to the personal computer using the interface circuitry. The personal computer would then be powered up with the emulator acting as its video card. In this way the emulator could be checked thoroughly for errors.

Logic emulation systems are currently considered to be the most effective and fastest method for design verification. They are used by most top semiconductor vendors to test IC designs before fabrication. The price for such systems varies from tens of thousands to millions of dollars depending on the type of the system, capacity and speed. Currently there are two main types of logic emulation systems available on the market:

1. FPGA-Based Emulators (FBEs)

2. Processor-Based Emulators (PBEs)

In what follows we present each of those types, their architectures, design tools and operation.

2.2.1 FPGA-Based Emulation Systems

The basic building block of an FBE system is an FPGA. In this emulator, several FPGAs are connected together to *emulate* (imitate) the functional behavior of a logic design. Before discussing the architecture of this system we first introduce FPGAs in detail.

2.2.1.1 Field Programmable Gate Arrays

A *field programmable gate array* is a reprogrammable chip that was first introduced in the 1980s [17]. By means of reprogrammable logic embedded inside, an FPGA can virtually implement any logic design. The main advantages of FPGAs can be summed up in two main points. The first advantage is that they are inexpensive; the price for a single FPGA starts from a few dollars. In addition, the reprogrammable capability of the FPGA makes it reusable for many designs which lowers its cost even more. The second main advantage of FPGAs is that they have a fast time-to-market. Unlike custom made chips where every single design has to be handled individually, FPGAs, because they are not custom made, are available off the shelf.

The above mentioned properties or advantages have given great importance to FPGAs in the industry. More and more designs are being implemented on FPGAs to save money and time. Companies could use FPGAs for their designs instead of Application-Specific Integrated Circuits (ASIC) chips to go around the lengthy and costly process of designing and building custom made chips. However, these gains do not come without a price; FPGAs are still bigger and slower than their counterpart ASIC chips. Because of their programmable nature and since they are not built to suit a specific design but rather any design, FPGAs still suffer from a decrease in logic utilization, i.e. bigger area, and slower speed. FPGA manufacturers are addressing this problem now more than ever and with the emergence of modern more sophisticated FPGAs these problems are becoming of lesser importance and the advantages of FPGAs are outweighing any disadvantages they have.

The programmable ability of an FPGA is derived from the use of programmable logic elements able to emulate or imitate the functional behavior of any logic function. Several architectures for FPGAs have been proposed, however, they all share some basic components. Figure 2.2 is a simplified illustration of a typical FPGA architecture [30].

FPGAs are made up of several major components. The two most important components are logic elements and routing resources. A logic element in the FPGA is responsible for emulating the behavior of a logical function. In other words a logic element could imitate the function of any logic gate. A typical logic element, shown in Figure 2.3, contains three main elements: lookup table, flip-flop and a 2-to-1 multiplexer. It is the task of the lookup table to operate as a logic gate. A typical lookup table is shown in Figure 2.4. The lookup table shown in the figure has four

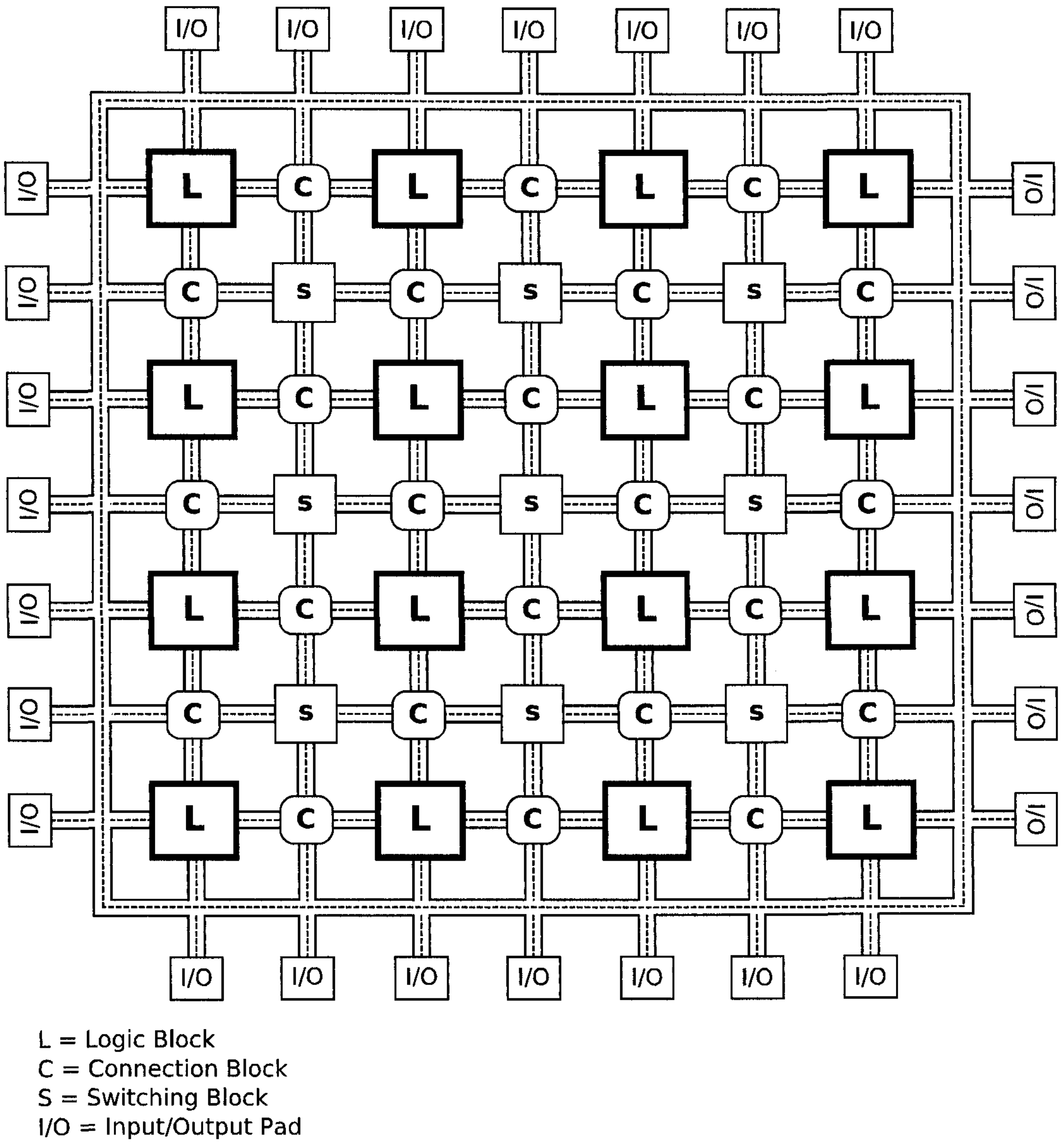

Figure 2.2: A Generic FPGA Architecture

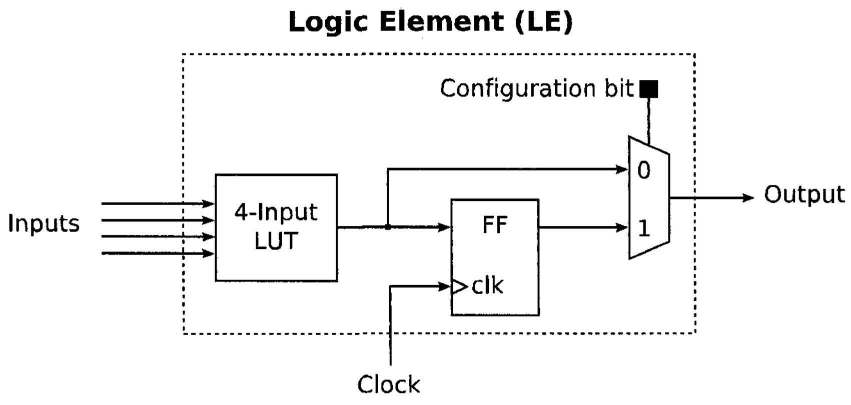

Figure 2.3: Internal Structure of a Logic Element

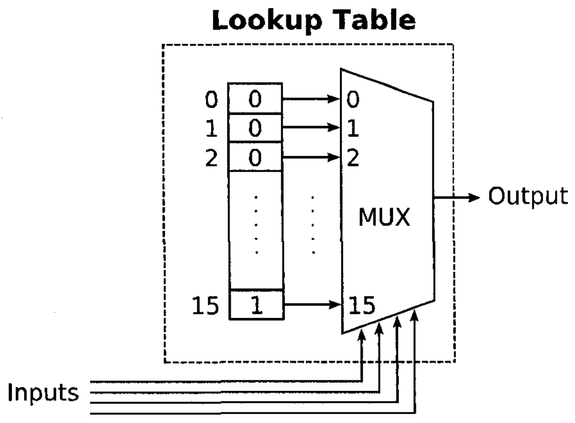

Figure 2.4: Internal Structure of a Lookup Table

inputs. It contains a memory array and each array element is connected to an input of a 16-to-1 multiplexer. The selection bits for this multiplexer are the inputs of the lookup table, i.e. the presumed inputs of the logic gate. To program the table the compiler sets the bits of the memory array. Based on the selection bits (inputs) of the multiplexer one of those array elements is chosen. The lookup table shown in the figure is an example of a four-input AND gate; only when all the inputs are 1's is the last element of the array chosen and the output is 1.

The lookup table only handles the combinational part of the logic element. To accommodate for sequential logic, the logic element contains a flip-flop whose input is the output of the lookup table. The output of this flip-flop is fed into a 2-to-1 multiplexer whose selection bit is reconfigured by the compiler. This selection bit decides the output of the logic element.

Besides the logic elements, FPGAs contain routing resources to connect these elements together. The routing resources are basically made up of connection blocks, switching blocks and a set of wires that run vertically and horizontally across the FPGA. Connection blocks situated between the logic elements can be programmed to connect the outputs of these logic elements to any vertical or horizontal wire. Switching blocks situated between the connection blocks can in turn be programmed to connect the wires together [13].

By programming the logic elements, connection blocks and switching blocks a user can implement any logic design on the FPGA. Nevertheless, mapping a logic design onto an FPGA is not an easy task and is the major challenge in FPGA research.

In addition to the logic elements and routing resources, FPGAs contain two other important components: embedded memory blocks and input/output pads. Typically the FPGA would have several memory blocks of different sizes to store data that would be used to implement memory arrays or registers in a logic design. I/O pads, on the other hand are used to connect the FPGA to the outside world. Both memory blocks and I/O pads are connected to other elements of the FPGA via the routing resources mentioned above.

Currently there are two major FPGA vendors: Altera Corporation and Xilinx Incorporated [4, 34]. The latest FPGAs produced by these companies contains hundreds of thousands of logic elements capable of emulating what is equivalent to millions of ASIC logic gates [6, 35]. The role of FPGAs in the industry is growing and significant research is being carried out to enhance their performance and capabilities.

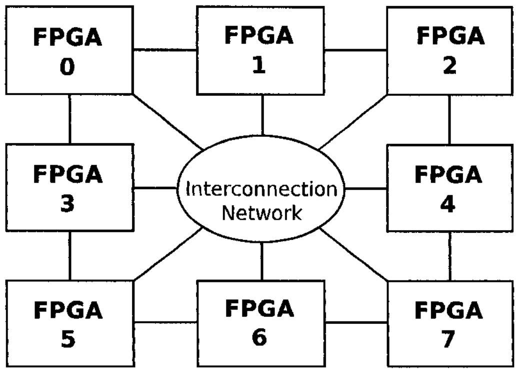

Figure 2.5: Multi-FPGA System

2.2.1.2 Architecture and CAD for FBEs

In an FPGA-based emulation system several FPGAs are connected together to be able to emulate a design of significant size. Several architectures were proposed to create the *Multi-FPGA System* (MFS)[21, 32, 7]. A typical architecture is shown in Figure 2.5. Here eight FPGAs are connected to each other by means of a programmable interconnection network. Such a system is highly flexible since all inter-FPGA connections are programmable.

The CAD flow, shown in Figure 2.6, for a multi-FPGA system is as follows. The user supplies the compiler with a logic design. After performing logic synthesis and technology mapping, the compiler partitions the design into several parts such that each part could fit on one FPGA. Then each part of the design would be assigned to a specific FPGA inside the MFS and the compiler starts routing the signals or connections between all the FPGAs, this is known as inter-FPGA routing. After inter-FPGA routing is done the compiler starts placing and routing each part of the design in its specific FPGA, this is known as intra-FPGA placement and routing. The final step would be to generate a bit stream of the design and download it to the FPGAs [13].

FBEs are an efficient verification tool; they can emulate any design and can ac-

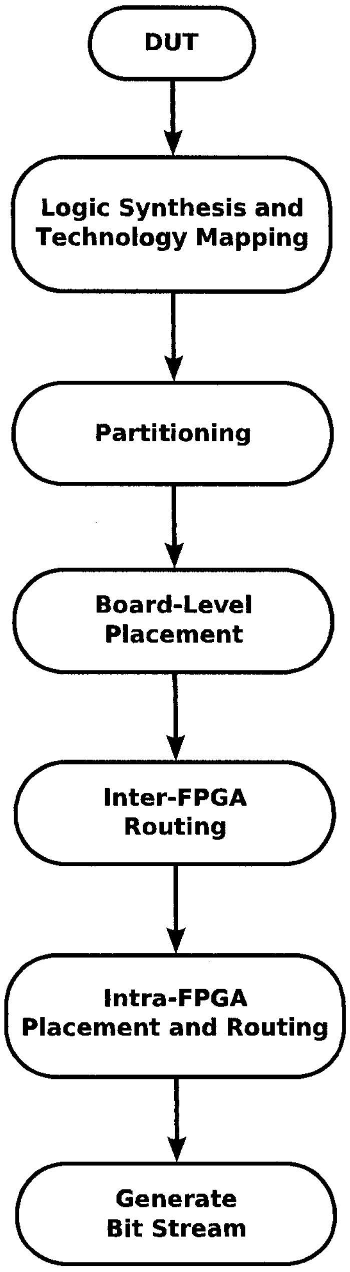

Figure 2.6: CAD Flow for FBEs

commodate any logic capacity by simply increasing the number of FPGAs. FBEs also have a relatively high emulation speed because they exploit parallelism in hardware. Another major advantage is that they have a relatively low price starting from only several thousand dollars.

Nonetheless, FBEs still face a major problem: CAD tools. Mapping a logic design to a multi-FPGA system by programming the FPGAs and the inter-FPGA routing resources is very problematic. Partitioning a design, placing it on FPGAs and then routing the signals have been one of the main focuses of FPGA research. Although many algorithms have been proposed and architectures suggested, CAD tools for FPGAs are still very complex. Compiling a design for a multi-FPGA system has an unpredictable compile time and may never even succeed. In addition, FBEs have very limited visibility and debugging support, which makes it very difficult for the designer to catch errors. Also, if an error was discovered and fixed the change might trigger a chain reaction in the whole system and the design would need to be compiled and downloaded again.

2.2.2 Processor-Based Emulation Systems

The second major type of logic emulation systems is processor-based emulators. The basic building block of a PBE is what is known as an *emulation processor* that can emulate a large number of logic gates and memory functions. Several of these emulation processors are connected together and run in parallel to emulate the functional behavior of a logic design [15]. Before we discuss in detail the architecture of this system it is useful to briefly describe the emulation processors and their operation.

2.2.2.1 Emulation Processors

Similar to a logic element in an FPGA, an emulation processor can perform the logical operation for any given function. Although it is made from custom hardware it is still programmable; that is achieved because embedded inside this processor is a reconfigurable lookup table. The structure of this lookup table is exactly the same as that of the one inside the FPGA. The main difference between this processor and the logic element of the FPGA is that a logic element of the FPGA is programmed only once before emulation starts and therefore can only implement one logic function during the whole emulation cycle. This is in contrast with this processor which can reprogram its lookup table during emulation to emulate different logical functions. The array elements for the lookup table would be stored inside the processor and then loaded into its lookup table during emulation to change the logic function at any time. The ability to change its operation type during emulation is what gives the processor its advantage over the logic element of the FPGA. More on the processor's architecture and operation will be described in later chapters.

It is worth noting that inside a PBE there might be several kinds of emulation processors. A PBE could have all homogeneous processors, in which case each processor would have to be able to perform any function of the logic design. Alternatively, a PBE could have heterogeneous processors, in which case specific processors would perform specific tasks (e.g. several processors would perform logic operations while others would perform memory functions) [15, 10].

2.2.2.2 Architecture and CAD for PBEs

A typical architecture of a processor-based emulator is shown in Figure 2.7. The emulation processors are connected together via a programmable interconnection network to ensure that a signal could traverse from one processor to another. It should be noted that unlike FBEs where the interconnections are fixed during emulation, this interconnection can be reprogrammed during emulation. The reader should keep in mind that reprogramming the processors during emulation is quite different from programming them prior to emulation. The former is done by the emulation support

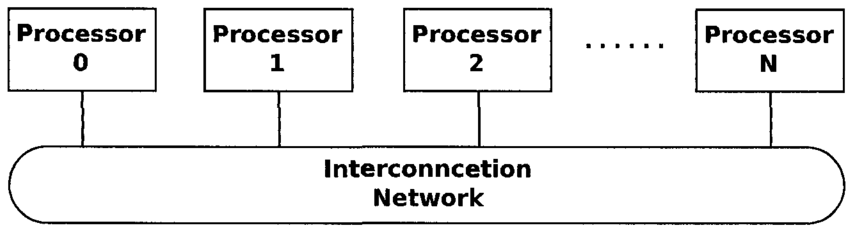

Figure 2.7: Processor-Based Emulation System

facilities while the latter is done by the emulator itself with no connection to the facilities.

The CAD flow for PBE, shown in Figure 2.8, is described as follows. The user supplies the compiler with the logic design. The compiler performs logic synthesis and technology mapping then partitions the design into several parts such that each part would be able to fit in one emulation processor. After each of those parts is assigned to a specific processor, a process called scheduling starts. During scheduling different logic functions which have been assigned to each processor are allotted different time slots throughout the emulation period. For example, an emulation processor would perform a logical AND during a specific time slot and a logical OR during another time slot. After scheduling is done a bit stream is generated and downloaded onto the processors before emulation starts [15].

PBEs have several advantages. They have very efficient and fast CAD tools compared to FBEs. In addition to that, they have much better visibility and debugging support. Finding an error and fixing it in a PBE is a standard procedure and usually does not trigger a chain reaction in the whole emulator. When an error is found the designer would only have to fix the specified processor and not the whole design unlike FBEs. CAD tools in PBEs are much less complicated than FBEs and have a well predictable compile time.

PBEs also have some disadvantages. They are comparatively slower than FBEs. Because processors in PBEs have to reprogram themselves periodically this leaves an

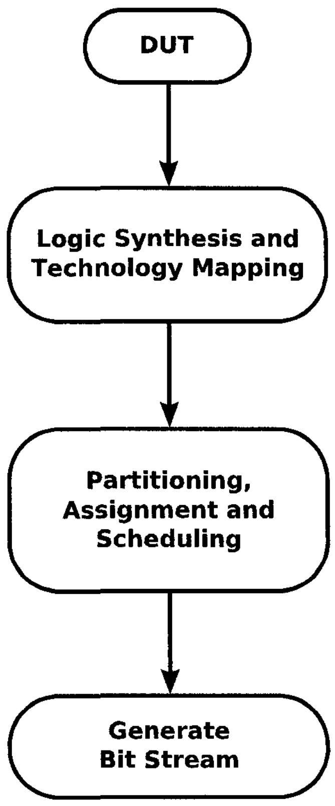
DUT
Logic Synthesis and Technology Mapping
Partitioning, Assignment and Scheduling
Generate Bit Stream

Figure 2.8: CAD Flow for PBEs

effect on the emulation speed. Nonetheless, current PBEs are becoming faster and faster and are able to compete with their FBE counterparts. The major disadvantage of PBEs is their price which is due to the fact that the whole system is built on custom hardware. Their prices are currently in the order of millions of dollars.

2.2.3 Commercially Available Logic Emulators

To give the reader an idea of the emulation technology on the market today, we present two examples of logic emulators manufactured by leading Electronic Design Automation (EDA) companies, Cadence Design Systems and Mentor Graphics [11, 25].

The *Incisive Palladium II* is a PBE system supplied by Cadence Design Systems [18]. This machine is capable of simulation acceleration and in-circuit emulation and can reach a speed up to 1.5 MHz. This emulator can compile up to 30 million gates per hour on a single workstation and has a maximum capacity of 256 million gates.

The *VStationPRO* is an example of an FPGA-based emulation system [33]. This product is manufactured by Mentor Graphics. It has a scalable capacity from 1.6 to 120 million gates and can reach a speed up to 1 MHz. This emulator can compile at a rate of 5 million gates per hour.

Chapter 3

System Architecture and Operation

This chapter presents the architecture and operation of the proposed logic emulation system. The first and second sections include an introduction and a general view of the emulator. Sections 3 and 4 discuss the two basic components, logic and memory emulation processors, in detail. Sections 5, 6 and 7 discuss the emulation module, emulation chip and emulation engine respectively.

3.1 Introduction and Motivation

Chapter 2 introduced the two major types of logic emulation systems, FPGA-based emulators and processor-based emulators, along with the advantages and disadvantages of each one of them. Keeping that in mind, the motivation behind this work is to design an emulator that combines the two most important advantages of both systems: the *low cost* of FBEs and the *high efficiency* of PBEs. To achieve that we have to design a PBE that can be implemented on an FPGA.

It is important to note that this research only deals with the hardware part of this proposed system. The main goal is to design an efficient architecture for a PBE. The CAD tools necessary to operate this emulator are not the focus of this research and are beyond the scope of this work.

Before delving into the details of the system architecture, it is important to highlight one important aspect of a processor-based emulator. The main clock in an emulator, known as the *design clock*, is shown on top of Figure 3.1. It is the frequency of this clock that determines the speed of a PBE. During each clock period of this design clock a number, known as the *emulation step*, increments from zero to a specific number (127 in Figure 3.1). Shown at the bottom of the figure is the *emulation clock* whose clock period corresponds to a single emulation step.

During each emulation step, emulation processors will perform a different operation type which in effect means that a single processor could perform a maximum of 128 different operations given that the number of emulation steps in a single design cycle is 128.

We now discuss the details of the architecture and operation of the logic emulation system starting with the basic components. We should note that the architecture proposed for this design is based on the architectures of [15] and [10] but has substantial differences with them.

3.2 Levels of Hierarchy

To enhance scalability, the design contains three levels of hierarchy connected together using different topologies. These levels are:

1. Emulation module

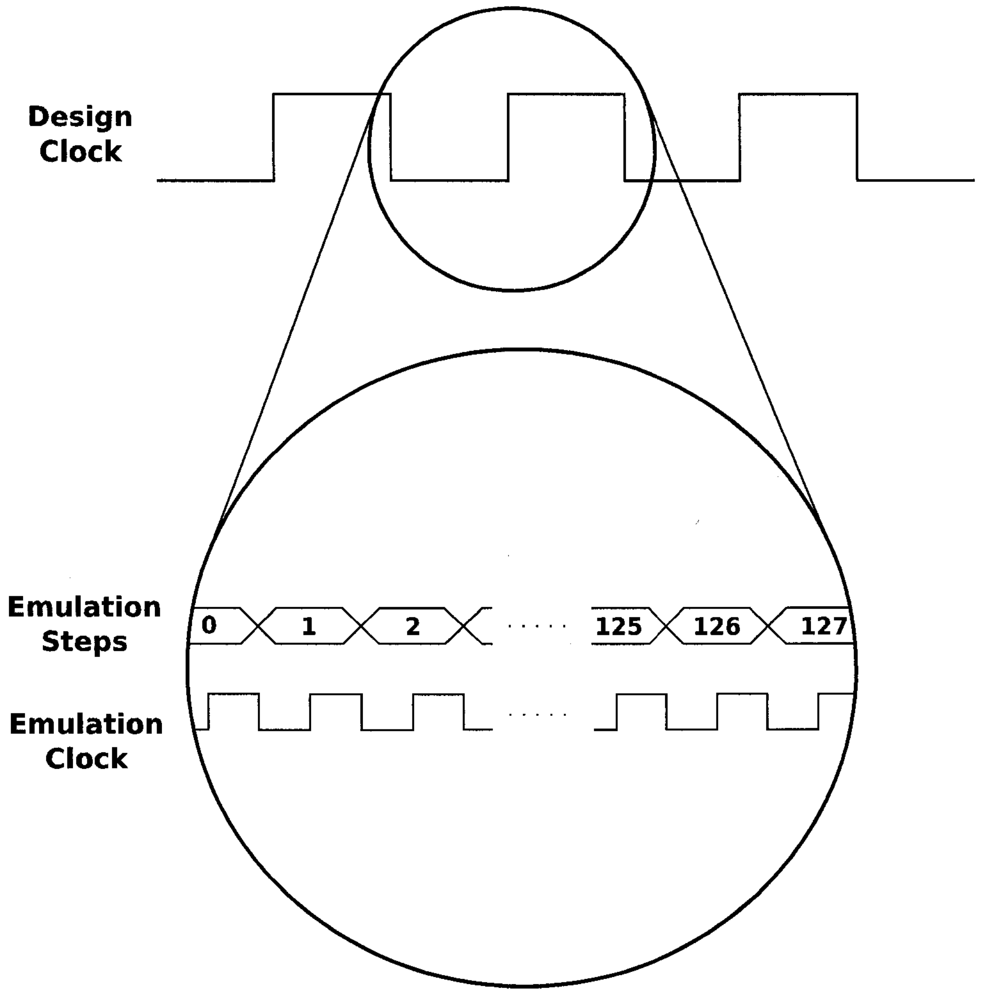

Figure 3.1: Emulation Design Cycle

2. Emulation chip
3. Emulation engine

The building blocks of this emulation system are the logic emulation processor and the memory emulation processor. A specific number of each type of these two processors are connected together by an interconnection network to form an *emulation module* making it the first level of hierarchy.

The second level of hierarchy is the *emulation chip* which contains a certain number of identical emulation modules. All the modules inside one emulation chip are connected by an interconnection network similar to the one inside the emulation module itself. Each emulation chip would fit on one FPGA, hence the name *chip.*

The third level of hierarchy is the *emulation engine.* To increase logic capacity, several emulation chips would be implemented in a specially designed multi-FPGA system. This multi-FPGA system is known as the emulation engine which is capable of emulating a design of significant size.

Figure 3.2 gives an overview of the hierarchy of the system. Here, the emulation engine is made up of 8 emulation chips and each of those chips contains 8 emulation modules. Inside each of those modules is a number of logic and memory processors. Note that the interconnections between the chips and between the modules are not shown.

3.3 Logic Emulation Processor

The most basic component of the system is the *Logic Emulation Processor* (LP). The sole purpose of this processor is to emulate the functional behavior of logic gates. Each gate is represented as a lookup table that can be programmed to imitate any desired logic function. The total number of logic gates that a single processor can

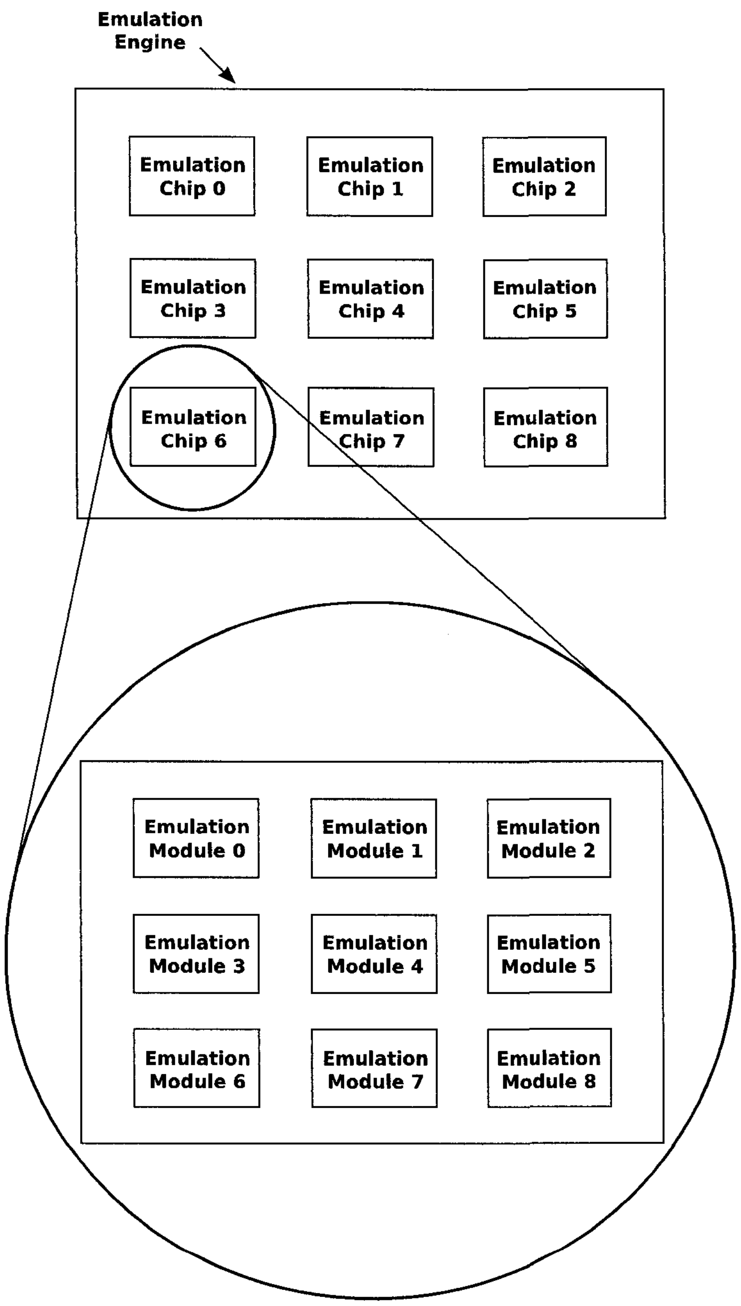

Figure 3.2: System Hierarchy

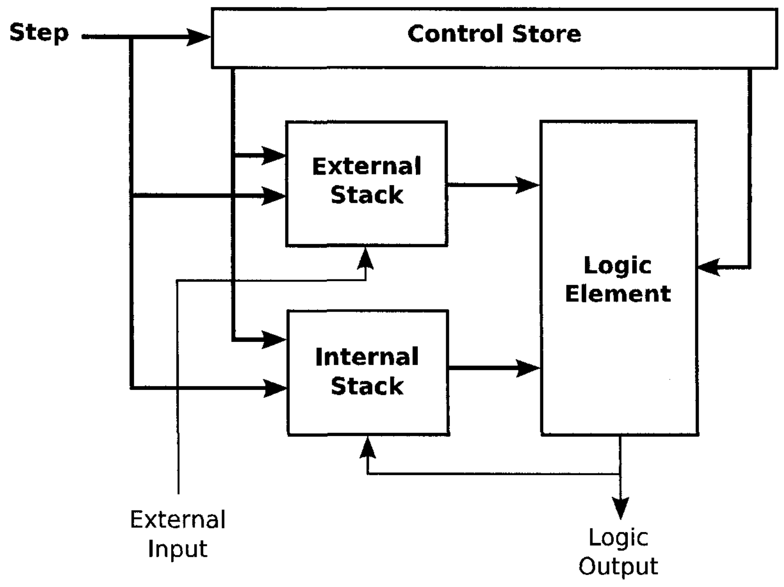

Figure 3.3: Logic Emulation Processor

emulate depends on its lookup table size and the number of emulation steps executed in a single design cycle. The proposed logic processor has three main elements:

1. Control store
2. Data stacks
3. Logic element

An architectural overview of this processor is shown in Figure 3.3.

3.3.1 Control Store

The *control store* is used to store a unique control program for each processor to determine the operation type during each emulation step. The control store contains several instructions of predetermined width that are generated by an emulation compiler whose task is to partition a logic design given by the user into several clusters.

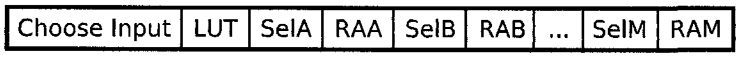

Figure 3.4: Logic Processor Control Word Fields

Table 3.1: Logic Processor Control Word Fields Description

Control Word Field	**Description**
ChooseInput	Picks an external input from the interconnection network.
LUT	The array elements of the lookup table.
SelA	Selects the source of the first input to the lookup table (internal or external stack).
RAA	Read Address A: the address for the first input of the lookup table.
...	...
SelM	Selects the source of the M^{th} input to the lookup table (internal or external stack).
RAM	Read Address M: the address for the M^{th} input of the lookup table.

These clusters are formed such that each one can fit into a single emulation processor. The emulation compiler then converts these clusters into a set of control words. The control store is filled up with these words prior to emulation. During emulation, these control words are read to instruct the processor on what to do during a specific step [15].

The number of these instructions (i.e. the depth of the control store) is equal to the maximum number of emulation steps needed in a single design clock cycle. The fields of these instructions are shown in Figure 3.4 and described in Table 3.1 where M is the size of the lookup table.

The number of bits dedicated for each field of the control word depends on two factors: the size of the internal and external stacks and the lookup table size. The size of the internal and external stacks is equal to the maximum number of emulation steps in a single design cycle as would be discussed later.

Since the inputs of the logic element are located in the stacks, therefore the size of the address of each of these inputs is equal to $Log_2(N)$ where N is the number of emulation steps. The *LUT* size depends on the size of the lookup table inside the logic element. More accurately *LUT* size is equal to 2^M where M is the lookup table size. The size of the *ChooseInput* field depends on the interconnection network, more precisely on the number of processors (both logic and memory) and the number of their outputs in the interconnection network. The size of the *ChooseInput* field is $Log_2(P)$ where P is the total number of outputs of all the processors sharing one interconnection network. The only field that is independent of any external factors is the *Sel* field. The size of this field is only one bit since it is used to choose between only two types of stacks, either external or internal stack.

Therefore, the size of a single control word in bits is $Log_2(P) + M + 2^M + M \times Log_2(N)$.

3.3.2 Data Stacks

The *data stacks* are used to store one bit values provided as inputs to the logic element. The proposed design has two stacks: an internal stack and an external stack. Ideally the two stacks are of the same width (one bit) and same depth. The depth of the stacks is typically equal to the maximum number of emulation steps executed in a single design clock cycle.

The internal stack is used to store values generated internally to the processor, specifically values from previous operations done during different emulation steps. The external stack is used to store values generated externally to the processor, specifically values from other logic or memory emulation processors. Both stacks have one write port and M read ports which are provided as inputs to the lookup table of the logic element.

At each emulation step, the internal stack provides output values on its read ports using the addresses ($RAA...RAM$) supplied to it from the control word. These outputs are used as inputs to the logic element. Also during the same emulation step, the internal stack stores the value of the current operation (i.e. the output of the logic element) in the address derived from the step value.

Equivalently, during each emulation step, the external stack provides output values on its read ports using addresses ($RAA...RAM$) supplied to it from the control words. These outputs are used as inputs to the logic element. Also during the same emulation step, the external stack stores an input value external to the processor in the address derived from the step value.

Note that both internal and external stacks supply the logic element with the same number of inputs (M) at the same time. It is the task of the logic element to choose between these inputs using the select values ($SelM$) in the control word.

3.3.3 Logic Element

The *logic element* is used to calculate the logic output for the processor. The logic element contains several multiplexers and a lookup table. The number of these multiplexers is equal to the number of inputs to the lookup table, or lookup table size. The Sel fields in the control word are used as selectors in these multiplexers to choose the sources of inputs for the lookup table (either internal or external stack). The logic element also contains an M-input lookup table implemented as a $2^M \times 1$ memory array and M-to-1 multiplexer. The elements of the array are filled by the LUT field in the control word. By doing so we are defining the type of logical function to be emulated during a specific emulation step. Figure 3.5 gives an overview of the logic element. Inputs shown in black are received from the internal stack, while inputs shown in grey

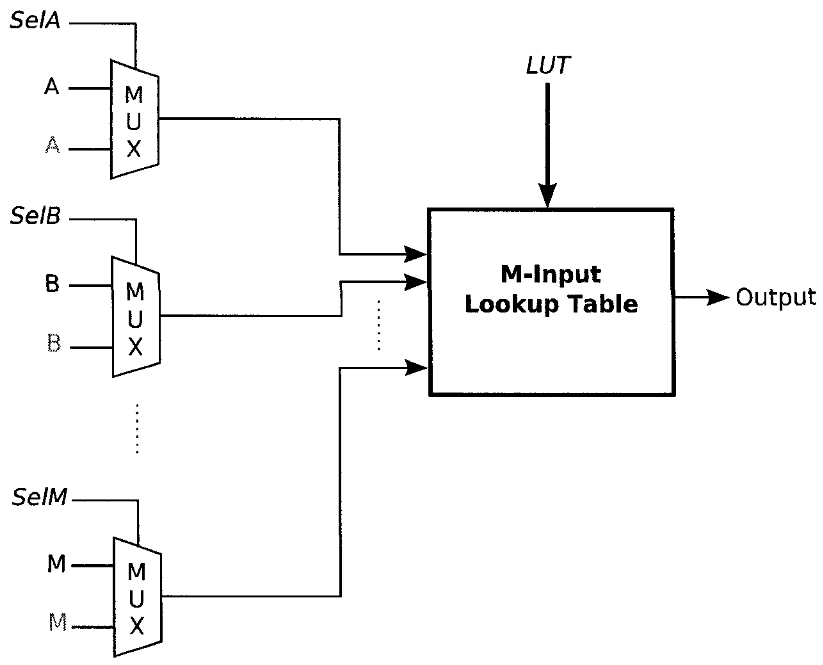

Figure 3.5: Logic Element

are received from the external stack. Select values and *LUT* are supplied from the control word. The lookup table of the logic element is identical to the one described in chapter 2 and shown in Figure 2.4.

3.3.4 Architecture and Operation

The basic architecture of the logic processor is shown in Figure 3.3. The control store is filled up with the control words prior to emulation through dedicated wires (not shown in figure). The processor has two external inputs and two external outputs. The first external input is the step value which is identical for all processors in the emulation engine and the second external input is used as an input for the external stack where it is stored for subsequent operations. The first external output is the *ChooseInput* field of the control word which is supplied to the interconnection network to choose an input for the external stack as would be described later in more detail. The second external output is the output of the logic operation which is supplied directly to the interconnection network to be used by other processors.

Preferably the depths of the control store, internal stack and external stack are the same and equal to the maximum number of emulation steps in a single design clock cycle. This would ensure an entry to every operation output in the internal stack, making this output available to any subsequent operations. As for the external stack, its usage enables the processor to make use of other logical or memory outputs supplied to it through the interconnection network. Another advantage of having the same number of entries in all three embodiments is that the step value is used both as a read address for the control store and a write address for the stacks at the same time.

The operation of the logic processor is as follows. A step value is supplied to the processor. The step value is used as an address to the control store where a control word is read. Address fields, $RAA...RAM$, are sent to the stacks and M bits are read from each stack at the same time then sent to the logic element. Using the Sel fields of the control word the logic element selects the sources of its inputs, either internal stack or external stack. The lookup table, which is filled up using the LUT field from the control word, performs the logic emulation and supplies the output. The output is then written to the internal stack where the step value is used as a write address. Also at the same time, an external input is written in the external stack. This input is chosen among the outputs of the other processors in the interconnection network using the $ChooseInput$ fields of the control words. Again the step value here is used as a write address.

In other words, the logic processor performs three operations in one emulation step. It first executes a logical function using its lookup table then writes the output of this function in the internal stack. In addition to that the processor picks an

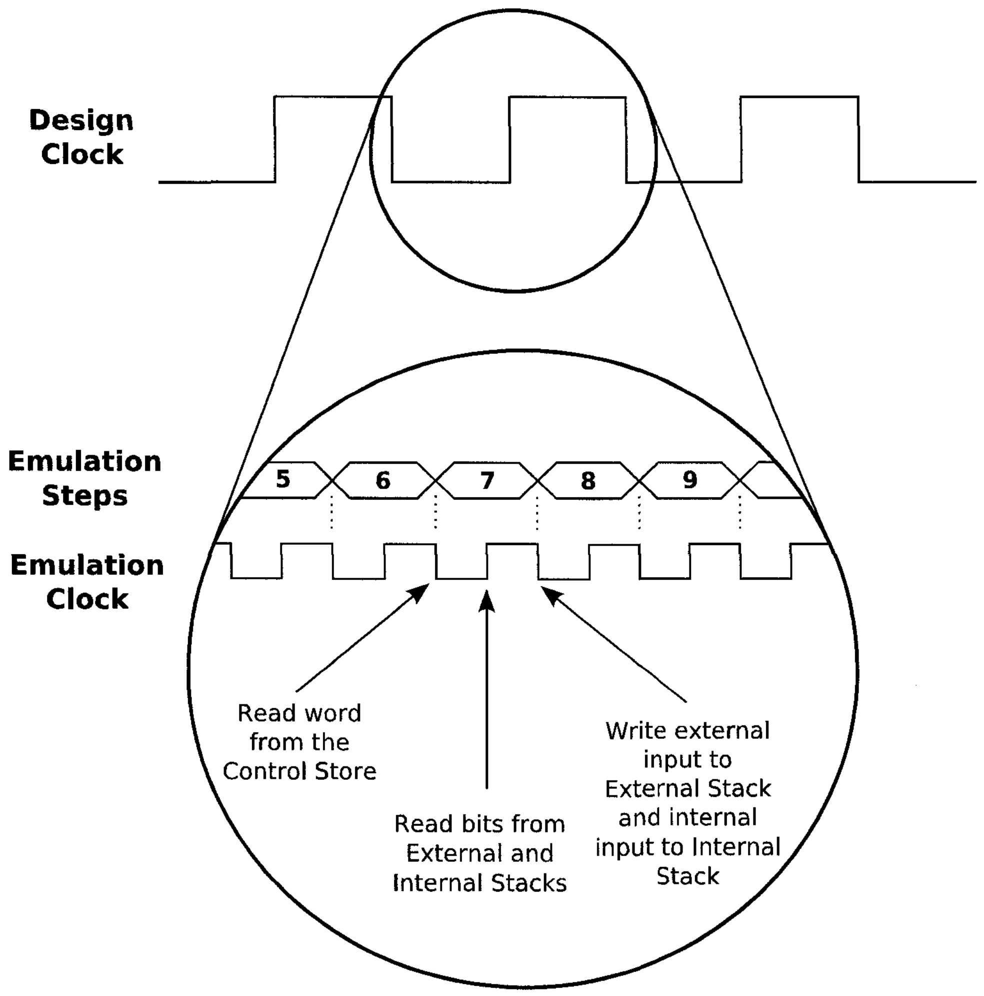
Design Clock
Emulation Steps
5
6
7
8
9
Emulation Clock
Read word from the Control Store
Read bits from External and Internal Stacks
Write external input to External Stack and internal input to Internal Stack

Figure 3.6: Operation of the Logic Processor

external input from the interconnection network and writes it to the external stack.

The operation described above requires three memory accesses which have to occur in a single emulation step but not simultaneously. These accesses are:

1. Reading a control word from the control store.

2. Reading inputs from the data stacks.

3. Writing values to the data stacks.

The reason these accesses cannot be executed at the same time is because a certain delay has to be given between each of them. To read inputs from the data stacks one has to wait for the address fields from the control word and to write values to the data stacks one has to wait for the output of the logical operation to be ready. To accommodate that, both edges of the clock are used since each emulation step corresponds to only one clock period. As shown in Figure 3.6 at the first falling edge of the clock a control word is read using step value n. At the rising edge of the clock and after sufficient time is given to fetch the control word, inputs to the lookup table are read from the stacks using addresses derived from the control word. At the second falling edge of the clock and after sufficient time is given for inputs from the stack to be read and the logic function is executed, the output of this logic function and an external input are written to the stacks at address n. Also at the second falling edge another control word is being read from the control store using address $n + 1$. This scheme ensures that all the memory accesses required for a logical operation are done in a single emulation step at different timings.

3.4 Memory Emulation Processor

In this section we present the *memory processor*, the second basic component of a complete processor-based emulation system implemented on an FPGA. The sole

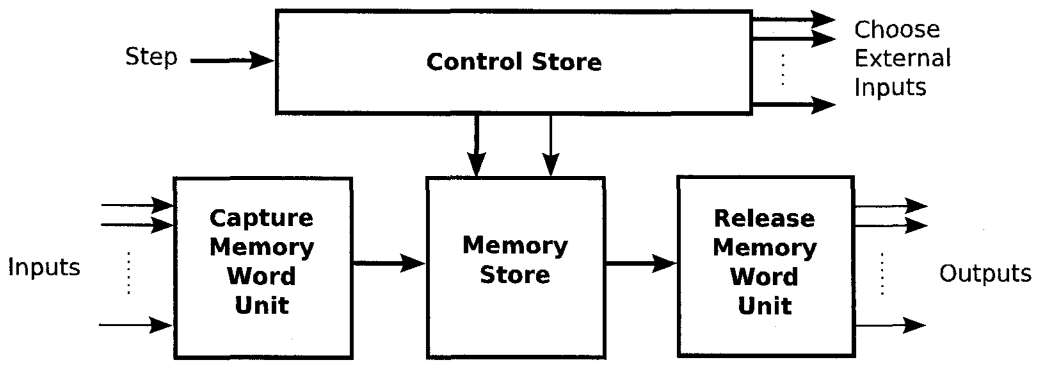

Figure 3.7: Memory Emulation Processor

purpose of the memory processor is to emulate memory registers and their functions. The total number of memory bits that this processor can emulate depends on the size of the embedded memory arrays and the number of emulation steps that are completed during a design cycle. The proposed memory processor has four main elements:

1. Control store
2. Memory store
3. Capture memory word unit
4. Release memory word unit

Figure 3.7 gives an architectural overview of the memory processor.

3.4.1 Control Store

The *control store* is used to store a unique control program for each processor to instruct the processor what to do during each emulation step. The control store contains several instructions of predetermined width. Similar to the instructions of the logic processor, they are generated by an emulation compiler whose task is to

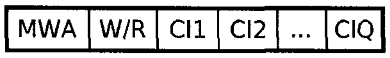

Figure 3.8: Memory Processor Control Word Fields

Table 3.2: Memory Processor Control Word Fields Description

Control Word Field	**Description**
MWA	Memory word address in the memory store.
W/R	Write (1) or read (0) a memory word.
$CI1$	Choose the 1st bit of the memory word from the external inputs.
...	...
CIQ	Choose the Q^{th} bit of the memory word from the external inputs.

partition a logic design given by the user into several clusters. These clusters are formed such that each one can fit in a single emulation processor. The emulation compiler then converts these clusters into a set of control words. The control store is filled up with these words prior to emulation. During emulation these control words are read by the processor to choose an operation to be performed in a specific emulation cycle [15].

The number of these instructions (i.e. the depth of the control store) is equal to the maximum number of emulation steps done in a single design clock cycle. The fields of these instructions are as shown in Figure 3.8 and described in Table 3.2 where Q is the size of the memory word.

The size of the control word and the number of bits dedicated for each field depends on three factors: the word size of the memory store Q, the number of emulation steps in a single design clock cycle N and the total number of outputs of all emulation processors in an interconnection network P.

The word width Q of the memory store is the same as the number of 1 bit inputs to the memory processor. This is because, ideally, we want an entry in the memory store for each input of the processor. The size of the memory word address MWA depends on the size of the memory store. We propose a memory store of size equal to the maximum number of emulation steps done in a single design clock cycle. Therefore the size of MWA in bits is $Log_2(N)$. The size of choose input field CI depends on the total number of outputs of all processors sharing an interconnection network (P). As a result the size of this field in bits is $Log_2(P)$.

Therefore, the size of a single control word in bits is $Log_2(N) + 1 + QLog_2(P)$.

3.4.2 Memory Store

The role of the *memory store* is to emulate real memory functions; more precisely read and write memory operations. It contains several words of predetermined width and number. The memory words can be from either of two sources: emulation support facilities or other emulation processors. Emulation support facilities, such as an emulation compiler, fill up the memory store prior to emulation so that the filled memory words can be read during emulation. Also during emulation the output of other processors in the interconnection network might be written to the memory store.

3.4.3 Release Memory Word Unit

The purpose of this component is to break up the memory word read from the memory store into one bit values. These bits are then supplied as outputs to the interconnection network to be used as inputs to other processors.

3.4.4 Capture Memory Word Unit

The purpose of this component is to concatenate several external inputs into a single memory word. The memory word that is formed after concatenation is entered into the memory store and therefore is of the same size as the memory word.

3.4.5 Architecture and Operation

The basic architecture of the memory processor is shown in Figure 3.7. The control store and the memory store are filled up with the control and memory words prior to emulation through dedicated wires (not shown in figure). The processor has several external inputs and outputs. The first external input is the step value which is identical for all processors in the emulation engine. The rest of the external inputs are values chosen from the interconnection network to be written to the memory store. The external outputs of this processor include the CI fields of the control word which are supplied to the interconnection network to choose inputs for the memory store. The rest of the external outputs are the bits read from the control store then are broken up by the release memory word unit.

Preferably the depths of the control store and the memory store are the same and equal to the maximum number of steps (N) in a single design clock cycle. This would ensure an entry to every memory word read or written in the memory store.

The operation of the memory processor is as follows. A step value is supplied to the processor. The step value is used as an address to the control store where a control word is read. The fields MWA and W/R are sent to the memory store. Using MWA a memory word is read or written and using the W/R field we determine if we are reading ('0') or writing ('1') during this step. In case of a read, a memory word is read and supplied to the release memory word unit where it is broken up into Q bits and output to the interconnection network. In case of a write, a memory word is formed by concatenating Q input bits from the interconnection network. This is the

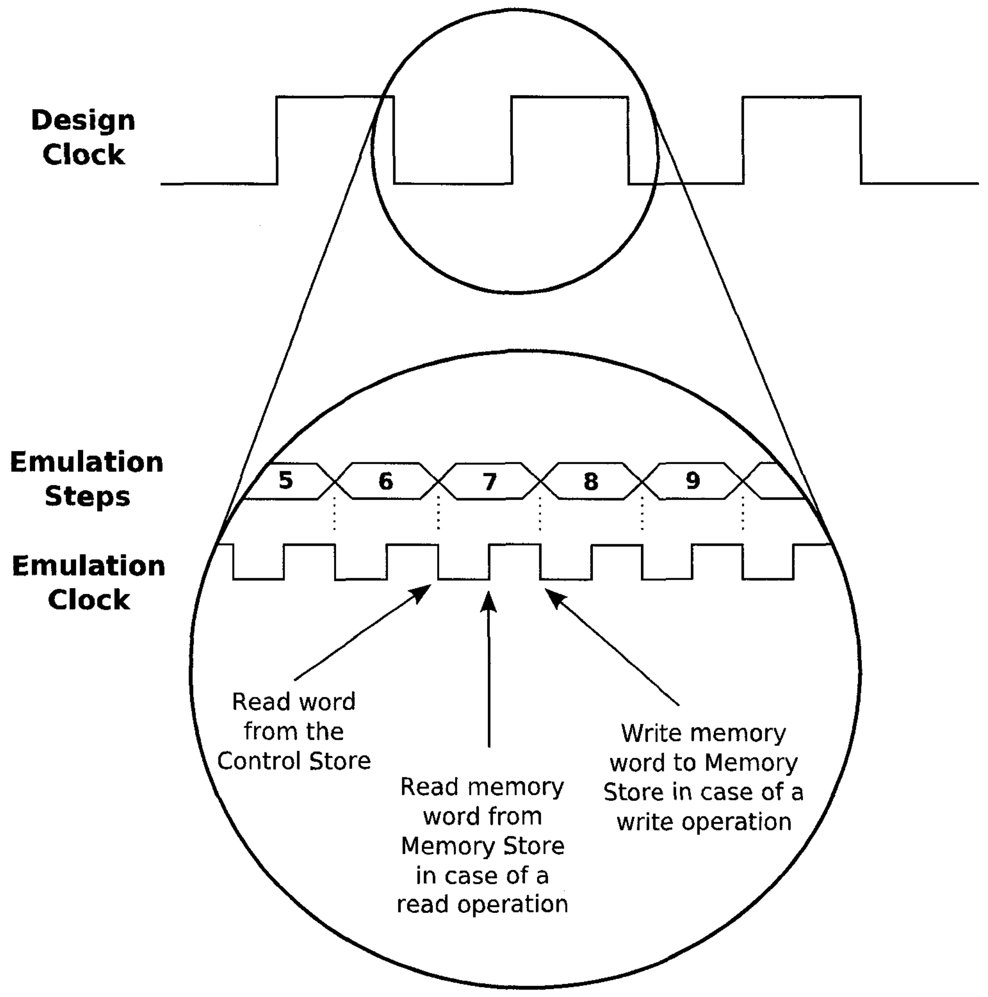
Design Clock
Emulation Steps
5
6
7
8
9
Emulation Clock
Read word from the Control Store
Read memory word from Memory Store in case of a read operation
Write memory word to Memory Store in case of a write operation

Figure 3.9: Operation of the Memory Processor

task of the capture memory word unit which then supplies it to the memory store to be written.

In other words, the memory processor performs either one of two operations during a single emulation step: it can read a memory word from a certain address in the memory store and then supply it to the interconnection network as a set of single bits or it can write a memory word supplied by the interconnection network at a certain address in the memory store.

The above description of the operation indicates that all the stages of the operation described above have to occur during the same emulation step but not simultaneously. A certain delay should be allowed between reading a control word from the control store and reading a memory word from the memory store. Also another delay should be allowed between reading a memory word and writing a memory word to give time for the bits to be read before they are written. To accommodate that, both edges of the clock were used similar to the logic processor. As shown in Figure 3.9, at the first falling edge of the clock a control word is read using step address n. In case of a read, at the rising edge of the clock a memory word is read from the memory store after enough time is given for the address to be derived from the control word. In case of a write, at the second falling edge of the clock several bits from the interconnection network are collected and written to the memory store to ensure that sufficient time was given for these bits to be read. Also at the second falling edge of the clock a new control word is read using step address $n + 1$.

3.5 Emulation Module

The first level of hierarchy in our system is the *emulation module*, shown in Figure 3.10. It consists of R logic processors and S memory processors. Each processor in one emulation module is connected to every other processor in the same module to ensure that the output of any processor is readily available as an input to every other

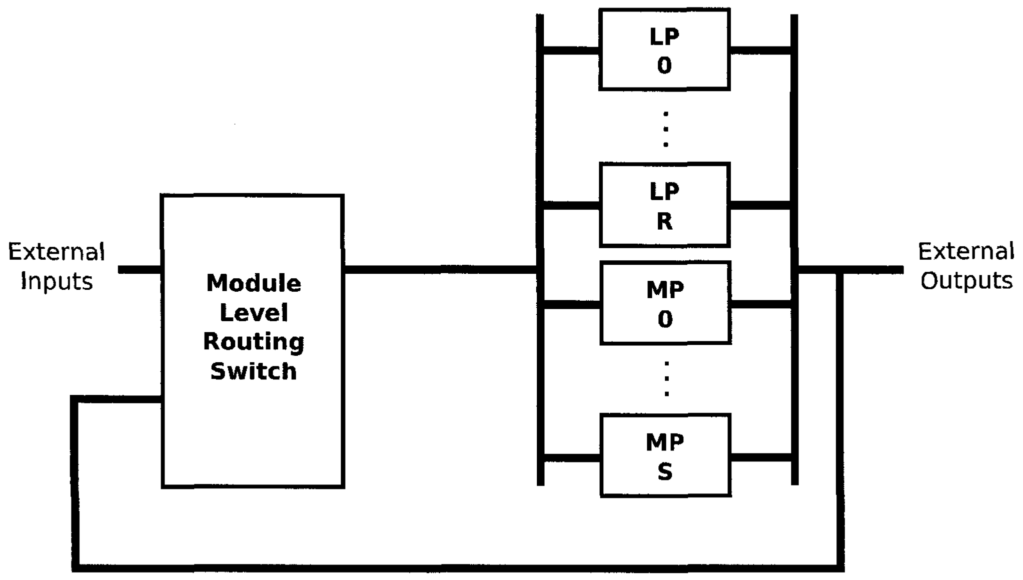

Figure 3.10: Emulation Module

processor. Moreover, each processor has an external input which could be connected to other processors in other modules.

In addition to the processors, the emulation module contains two other components: the interconnection network and the sequential filler. The interconnection network is made up of the set of wires connecting all the processors together and the module level routing switch.

3.5.1 Module Level Routing Switch

Connecting all the processors together in an emulation module is an interconnection network controlled by the *module level routing switch*. This switch is basically made up of (R+$Q \times S$) (R+$Q \times S$)-to-1 multiplexers; one multiplexer for each logic processor and one for every input of each memory processor. The purpose of this switch is to make the output of each processor readily available to every other processor to use. Moreover, the switch can supply the processors inside the module with external inputs that are derived from other modules.

The switch uses the *ChooseInput* field supplied by each processor as the selection bits of the multiplexers to route the signals between processors.

3.5.2 Sequential Filler

The limited number of pins on an FPGA makes it impossible for the user to fill up the control and memory stores of all processors at the same time. For this reason, a *sequential filler* is created to fill up the stores in a sequential manner. To choose which processor to fill up, the sequential filler has two input signals: one to choose which logic processor and the other to choose which memory processor. The usage of the filler should not affect performance in any way since it is only used once prior to emulation and at high speed.

3.5.3 Architecture and Operation

The basic architecture of the emulation module is shown in Figure 3.10. Each processor has one external input and one external output. The external input could be chosen from among all the outputs of all the processors in the same module or from a different source outside the module. The choice of this source will be described later. Each processor also supplies the interconnection network with an output. The wires used to fill up the control and memory store along with the sequential filler are not shown in the Figure 3.10.

All the processors in one emulation module receive an identical step value during an emulation step. The processors use this value to determine the operation as described earlier.

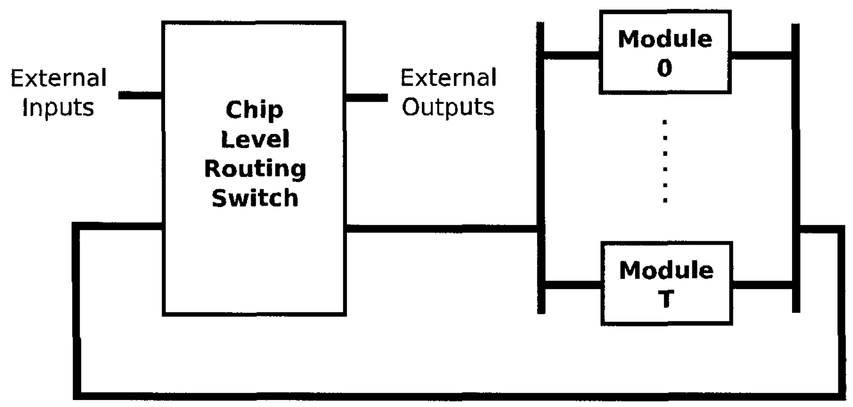

Figure 3.11: Emulation Chip

3.6 Emulation Chip

The second level of hierarchy in this system is the *emulation chip*. It consists of T identical emulation modules and fits on one FPGA. Specific processors inside the emulation module are connected to specific processors in other emulation modules to ensure that their outputs are readily available as inputs. Moreover, an emulation chip has several external inputs and external outputs whose numbers are to be chosen depending on the availability of pins on the FPGA. In addition to the modules, the emulation chip has one other component, the chip level routing switch.

3.6.1 Chip Level Routing Switch

The *chip level routing switch* connects all the module pins, external inputs and external outputs together. This switch is made up of several multiplexers; one multiplexer for each input of every module and one for each external output. The number of these multiplexers depends on the number of modules inside the emulation chip. As for their selection capacity, it depends on the resources available on the FPGA to store their selection bits. More about these multiplexers will be described in the next chapter.

The switch allows inputs of processors inside one module to choose among certain outputs of other modules. The switch also allows external outputs to choose among the outputs of certain processors or external inputs.

3.6.2 Architecture and Operation

The basic architecture of the emulation chip is shown in Figure 3.11. Each module would be able to choose among several outputs from different modules or external inputs. Similar to the module, the step value is identical for all processors in the emulation chip.

3.7 Emulation Engine

An *emulation engine* is the third and last level of hierarchy. The emulation engine contains a number of emulation chips connected together in a multi-FPGA system.

3.7.1 Multi-FPGA System

Several FPGA connection schemes are available today. The system shown in Figure 3.12 is an example of an 8-way mesh multi-FPGA system architecture while the one shown in Figure 3.13 is an example of a fully connected multi-FPGA system architecture.

Each FPGA contains an emulation chip. The chip would have a certain number of inputs and outputs through which it would communicate with other chips. Each processor inside the chip can choose among several of the external inputs and each of the external outputs can choose among several outputs of the processors. This gives each processor in the chip the capability to communicate with other processors in other chips implemented on other FPGAs.

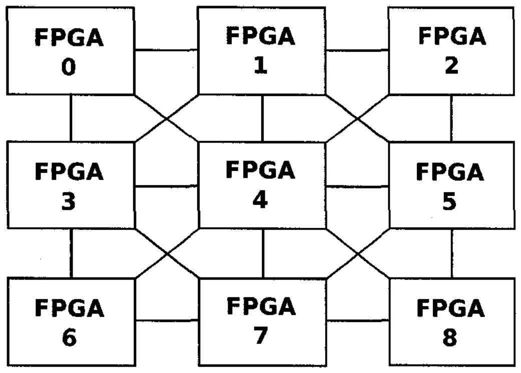

Figure 3.12: 8-Way Mesh MFS

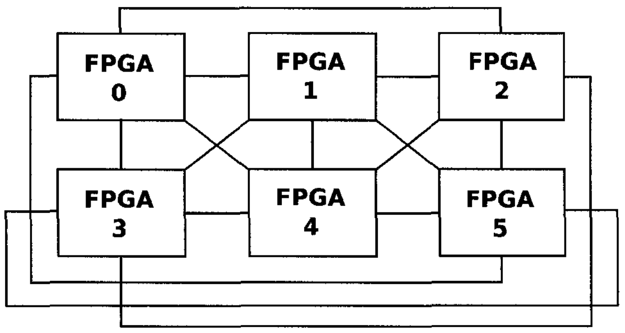

Figure 3.13: Fully Connected MFS

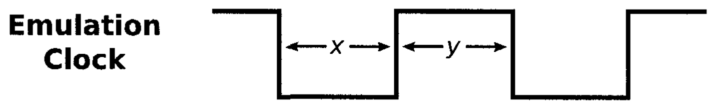

Figure 3.14: Clock Duty Cycle

In addition to that, some of these external outputs can choose among several of the external inputs enabling the FPGA to act as a routing switch. This would become useful in the case where two emulation chips are implemented on two FPGAs that do not share a direct connection. Here, intermediate FPGAs would serve as routers of the signal from its source and until it reaches its destination. Note that in the case where full connectivity is ensured for the multi-FPGA system these outputs are not necessarily useful.

We should note that the selection bits for the signal routings occurs at the rising edge of the clock in order to precede the writing operations that occur at the falling edge of the clock in all logic and memory processors.

3.7.2 Scalability Issues

The basic challenge that we have to solve in the multi-FPGA system is the one that deals with speed. The difference of time between the read and write operations is the crucial factor when dealing with the speed. We have to make sure that the difference between the rising and falling edges of the clock is long enough for the signal to traverse throughout the system. As mentioned above the first falling edge of the clock is when we read the control word, the rising edge is when we read from the stacks or the memory store and the second falling edge is when we write to the stacks or the memory stores. The challenge is to connect the FPGAs in such a way that if a certain processor in one FPGA needs to write a signal (or output) from another processor in a second FPGA the time between the rising edge and the second falling edge is long enough for the signal to traverse from the first FPGA to the second.

Assuming a non 50% duty cycle, the falling time of the clock is x and the rising time of the clock is y, as shown in Figure 3.14. This means that the critical time is y not x. The falling edge of the clock x deals with intra-FPGA connections; the rising edge of the clock y may have to deal with inter-FPGA connections. More on the scalability issues will be discussed in the next chapter.

Chapter 4

Architecture Exploration and Implementation Results

This chapter discusses the architecture exploration carried out and the implementation results for the proposed logic emulation system. The first section describes the implementation target used in this research. Section 2 presents the architecture exploration and the effect of changing key parameters on the area and performance of the emulator. Section 3 presents the implementation results.

4.1 Implementation Target

The FPGA used for implementation in this research is the Altera Stratix EP1S40F780C5 FPGA. We now present a detailed description of this FPGA.

4.1.1 Altera Stratix FPGA

The Altera Stratix FPGA Family [3] contains the following resources:

1. Logic Array Blocks (LABs)
2. M512 Blocks
3. M4K Blocks
4. M-RAM Blocks
5. DSP Blocks
6. I/O Elements

Each LAB block contains ten logic elements similar to the ones discussed in chapter 2 and shown in Figure 2.3. These logic elements are capable of emulating virtually any logic function. M512 is a memory block which contains 512 programmable bits plus parity bits. It can be configured as single-port or simple dual-port mode. The M4K is another memory block which contains 4,096 programmable bits plus parity bits and can be configured as single-port, simple dual-port or true dual-port mode. The third, and largest, memory block is the M-RAM which contains 512 kilobits of programmable memory plus parity bits. This memory block can be configured as single-port, simple dual-port or true dual-port mode. The DSP blocks of the Stratix FPGA are used to implement several forms of multipliers while the I/O elements are connected to the FPGA pins and support different I/O standards.

The FPGA used in this research, the Altera Stratix EP1S40F780C5 FPGA, contains 4,125 LABs or 41,250 LEs. It also contains 384 M512, 183 M4K and 4 M-RAM blocks making the total number of memory bits 3,423,744. In addition to that, it contains 14 DSP blocks and 616 I/O pins [3].

4.2 Architecture Exploration

In chapter 3 we described the architecture and operation of the emulation system without specifying certain values for important parameters such as the lookup table size or the number of emulation steps. In this section we describe architecture experiments that were performed to determine the effects of varying different architectural parameters on the area and delay of the proposed emulator.

4.2.1 Key Parameters

The key parameters that were presented in chapter 3 and explored in this design are:

1. M: lookup table size.
2. N: number of emulation steps.
3. P: total number of outputs of all processors in one emulation module.
4. Q: memory word size.

The main goal of the exploration is to choose a value for each of the above parameters. The best way to accomplish this goal is to vary each of the parameters and fix the others while checking for effect on area and performance. To do that the logic and memory processors were both implemented after each change and the results were recorded. It is important to note that the effects of the change of the parameters were only considered for individual processors. The routing between these processors, and in effect the hierarchy of the emulator, were not taken into consideration due to the complexity of the process. Instead the effect of each parameter on single processors was assumed to be proportional to its effect on the whole system.

4.2.2 Effect of Changing Parameters

In this section we aim to monitor the effect of each of the parameters on the area and performance of the logic and memory processors. For that reason, each parameter under consideration was changed and its effect observed while the other parameters were given fixed values. This process was repeated for each parameter on both processors. In what follows we show in graphs the effect of the change of each of the parameters. The effect on area is measured by the number of logic elements and memory bits each processor consumes when implemented on the FPGA while the performance is measured by emulation clock speed.

It is important to note that the results here were obtained after implementation and not from mathematical equations. More about the implementation of each processor will be discussed in later sections of this chapter.

4.2.2.1 Effect of Changing Lookup Table Size

The size of the lookup table of the logic processor determines the logic capacity of the processor. In other words, it determines how many logic gates each processor can emulate. To determine how this parameter might affect the area and performance of the processor, the size of the lookup table was increased by one starting with 2 and ending with 8. To ensure that we are reading the effect of the lookup table size only, the other parameters were never changed. The number of emulation steps and the total number of outputs were fixed at 128 and 64 respectively. Note that varying the lookup table size has no effect on the memory processor but on the logic processor alone. The results are shown in Figures 4.1, 4.2 and 4.3.

It is clear from Figure 4.1 and Figure 4.2 that as the size of the lookup table increased the area consumed by the logic processor increased exponentially. The reason behind that could be mainly attributed to the effect of the lookup table size on the control store and the logic elements. The size of the control word of the control

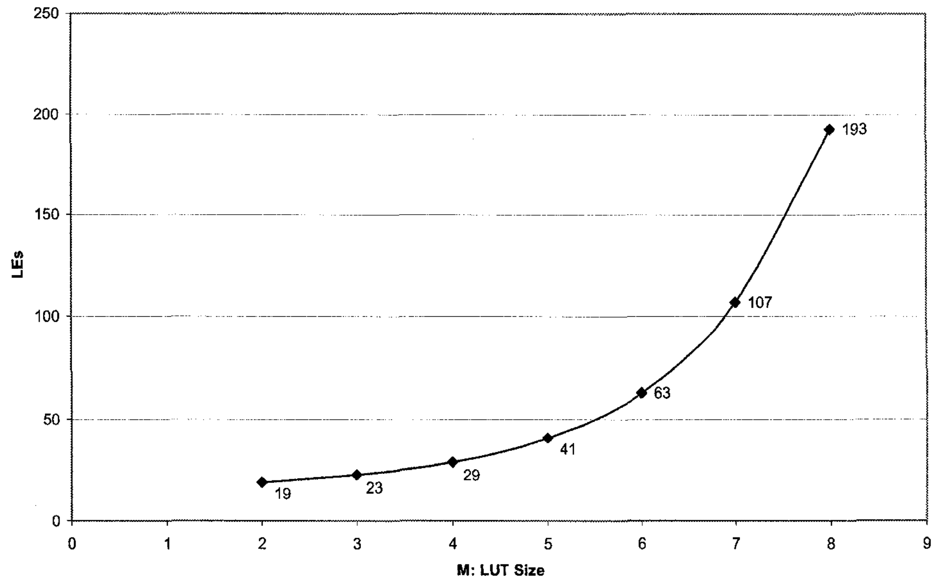

Figure 4.1: LUT Size vs. Area in LP

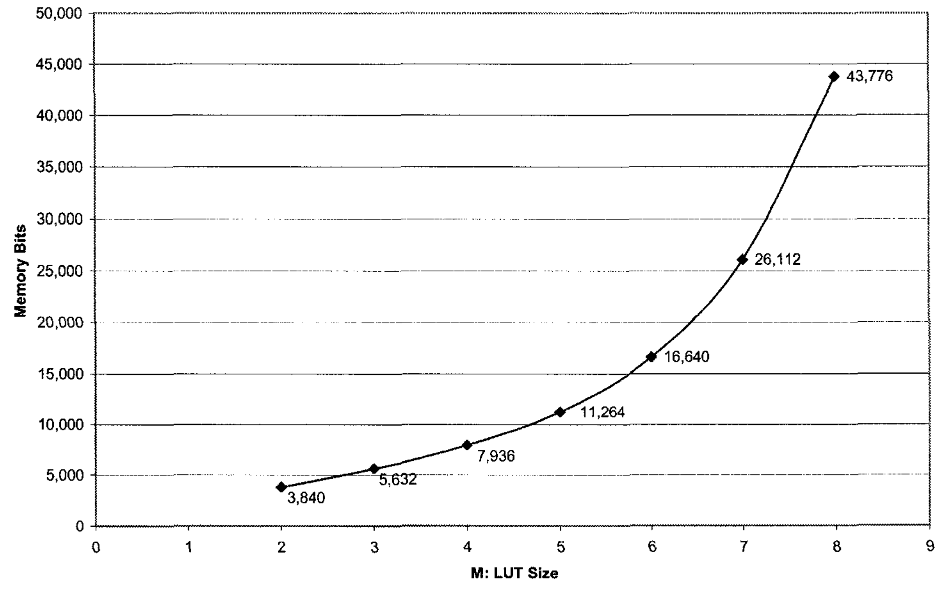

Figure 4.2: LUT Size vs. Memory Bits in LP

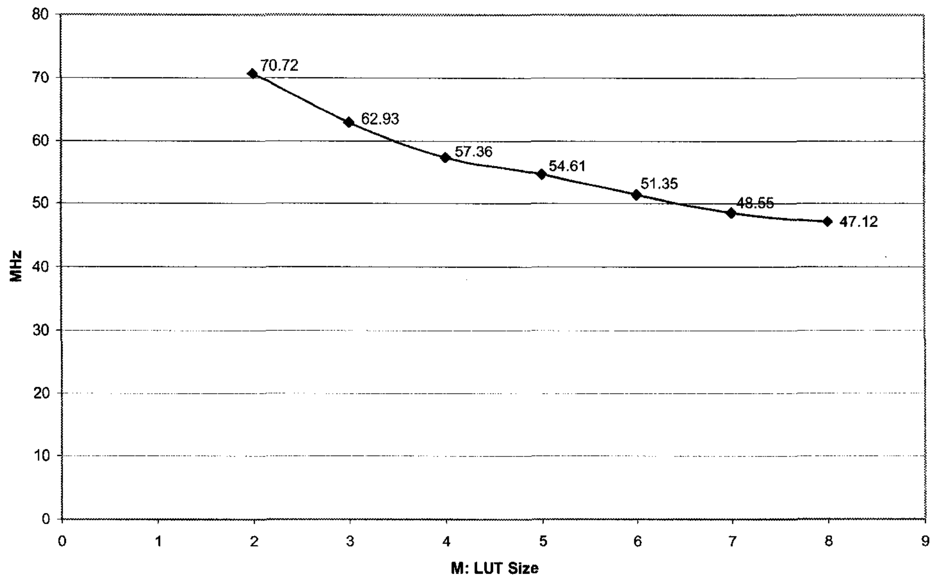

Figure 4.3: LUT Size vs. Speed in LP

store is exponentially proportional to the lookup table size, the size of a single control word in bits is $Log_2(P)+M+2^M+M\times Log_2(N)$, and thus increasing the lookup table size will result in an exponential increase in the control store size. The exponential increase in the number of logic elements could be explained in a similar way. It is due to the fact that the lookup table is implemented in the FPGA's logic elements and its size increase meant an increase in the number of logic elements consumed.

As for the effect of the lookup table size on the speed of the processor, it can be seen in Figure 4.3 that as the lookup table size increased the performance of the processor decreased gradually. This is predictable since the time for processing a certain number of inputs inside a processor is likely to increase as the number of these inputs increase.

4.2.2.2 Effect of Changing Number of Emulation Steps

The second key parameter to be tested is the number of emulation steps. The number of emulation steps, N, is a critical parameter in both the logic and the memory processor. Here we study its effect in both processors.

- *Effect on the Logic Processor*: The number of emulation steps was varied from 64 to 512 in steps of power of 2. The size of the lookup table, M, was fixed at 4 while the number of total outputs, P, was fixed at 64. The results are shown in Figures 4.4, 4.5 and 4.6.

  As shown in Figure 4.4 the change in the number of emulation steps barely had any effect on the number of logic elements used to implement the logic processor. In contrast, as shown in Figure 4.5, the change in the number of emulation steps had a linear effect on the number of memory bits. This could be explained by the fact that the number of emulation steps does not affect the combinational part of the processor but rather the size of the memory blocks, control store and data stacks.

  As for the speed, it is clear from Figure 4.6 that changing the number of emulation steps had little effect on the speed of the processor.

- *Effect on the Memory Processor*: The number of emulation steps also affects the implementation of the memory processor. Here, the number of steps was also varied from 64 to 512 in steps of power of 2. The size of the memory word, Q, was fixed at 8 and the total number of outputs, P, was fixed at 64. The results of the implementation are shown in Figures 4.7, 4.8 and 4.9.

  Similar to the logic processor, the effect of the number of steps was only limited to the number of memory bits as shown in Figures 4.7, 4.8 and 4.9. This is expected because the number of emulation steps determines the size of the

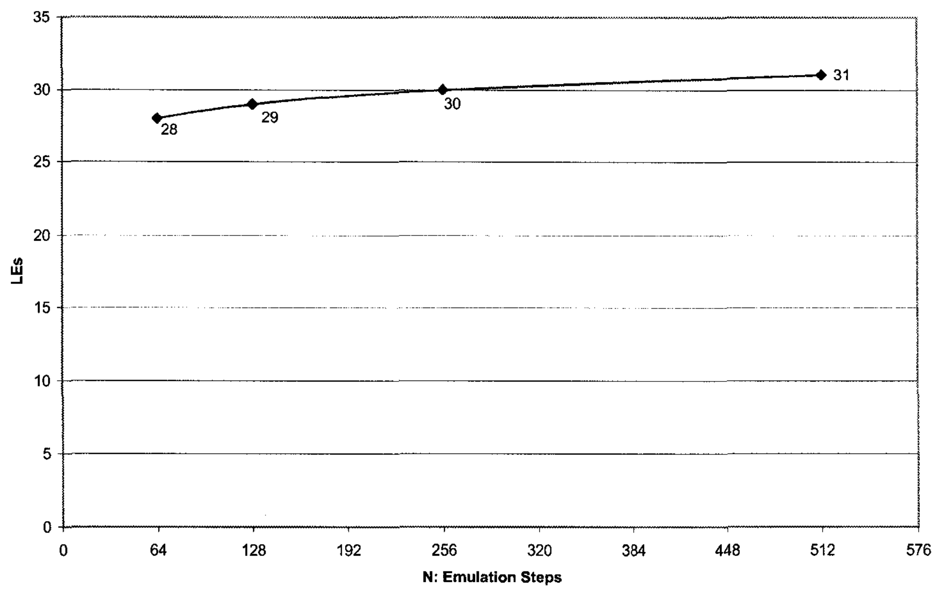

Figure 4.4: Number of Emulation Steps vs. Area in LP

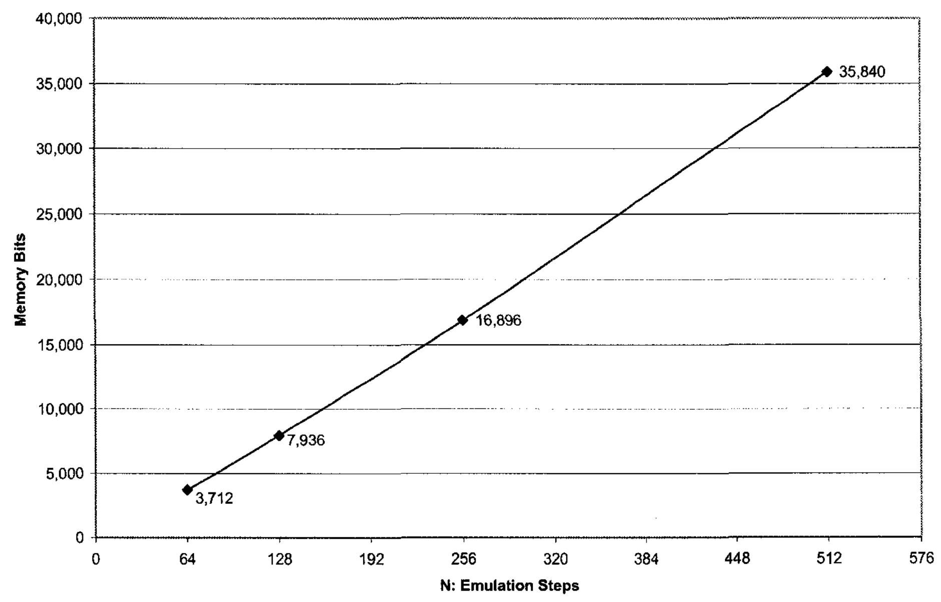

Figure 4.5: Number of Emulation Steps vs. Memory Bits in LP

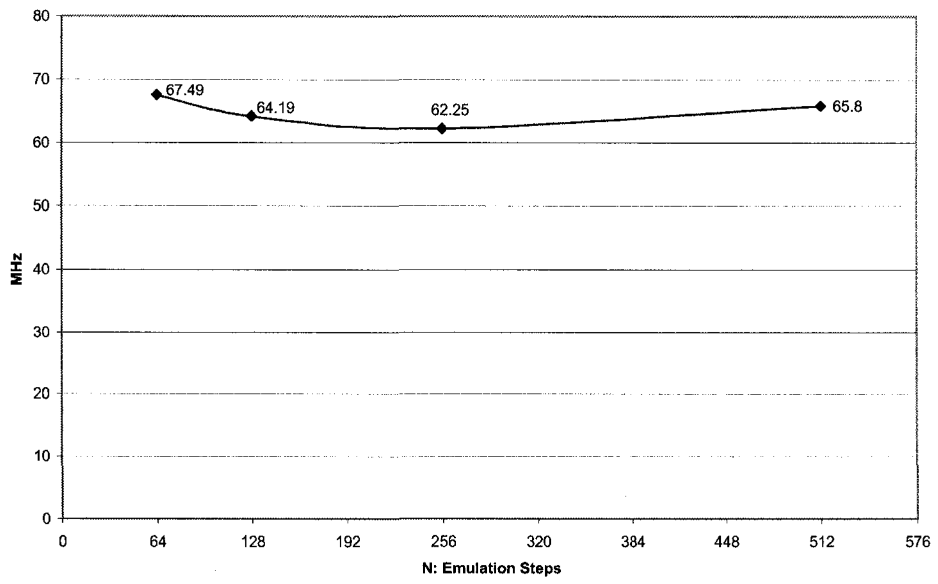

Figure 4.6: Number of Emulation Steps vs. Speed in LP

control store and the memory store and does not affect the combinational part of the processor.

4.2.2.3 Effect of Changing Total Number of Outputs

The third parameter to be checked for its effect is the total number of outputs, P, which will help determine the numbers of logic and memory processors packed in one emulation module.

- *Effect on the Logic Processor*: P was varied from 32 to 256 in steps of power of 2 while the lookup table size, M, was fixed at 4 and the number of emulation steps, N, fixed at 128. The results are shown in Figures 4.10, 4.11 and 4.12.

  As can be observed in Figures 4.10, 4.11 and 4.12 the parameter had virtually no effect on the area and performance of the processor. This can be explained by the fact that the only effect this parameter has is on the size of the *ChooseInput*

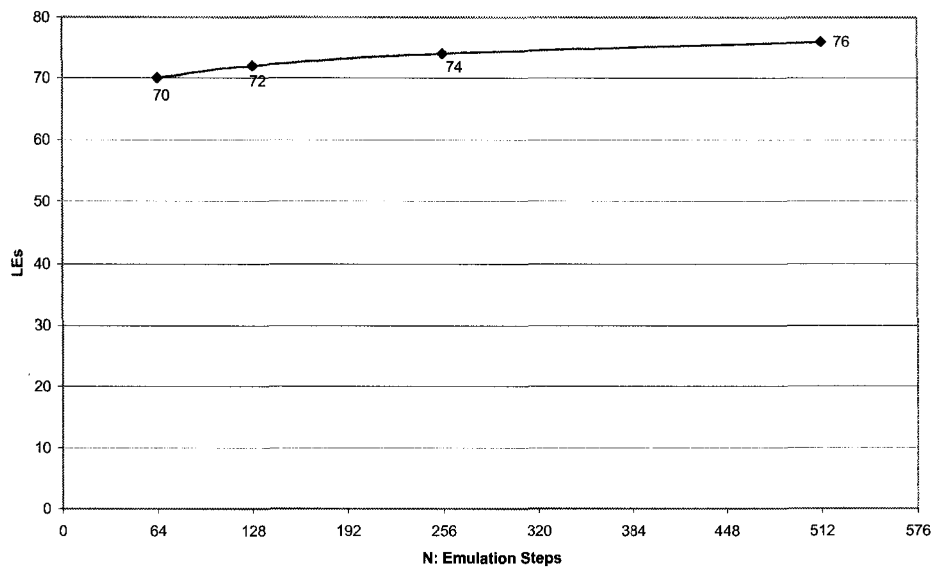

Figure 4.7: Number of Emulation Steps vs. Area in MP

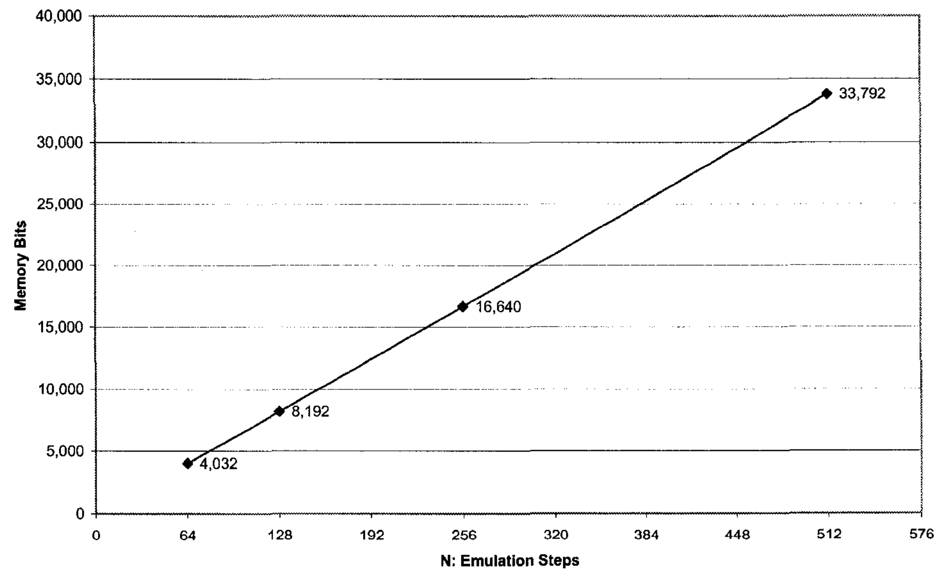

Figure 4.8: Number of Emulation Steps vs. Memory Bits in MP

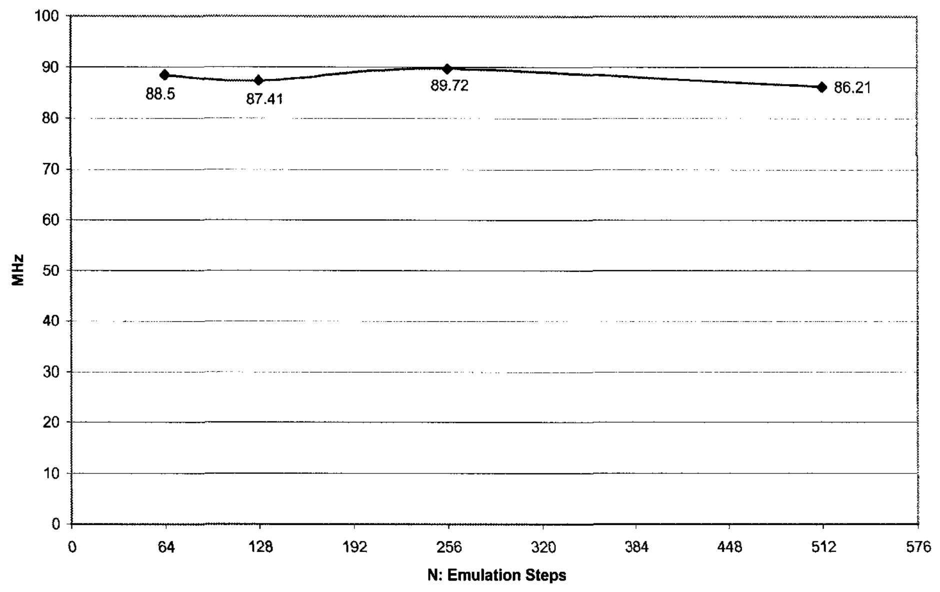

Figure 4.9: Number of Emulation Steps vs. Speed in MP

field of the control word. Changing the number of outputs only increases this field by one bit at a time.

- *Effect on the Memory Processor*: P was varied from 32 to 256 in steps of power of 2 while the size of the memory word, Q, was fixed at 8 and the number of emulation steps, N, fixed at 128. The results are shown in Figures 4.13, 4.14 and 4.15.

  As shown in Figure 4.13 and Figure 4.14, the area consumed by the processor increased as the number of outputs increased. This can be explained by the fact that the CI field in the control word for each input bit increases as P increases. As for the speed, it is shown in Figure 4.15 that changing the number of outputs had no major effect on speed.

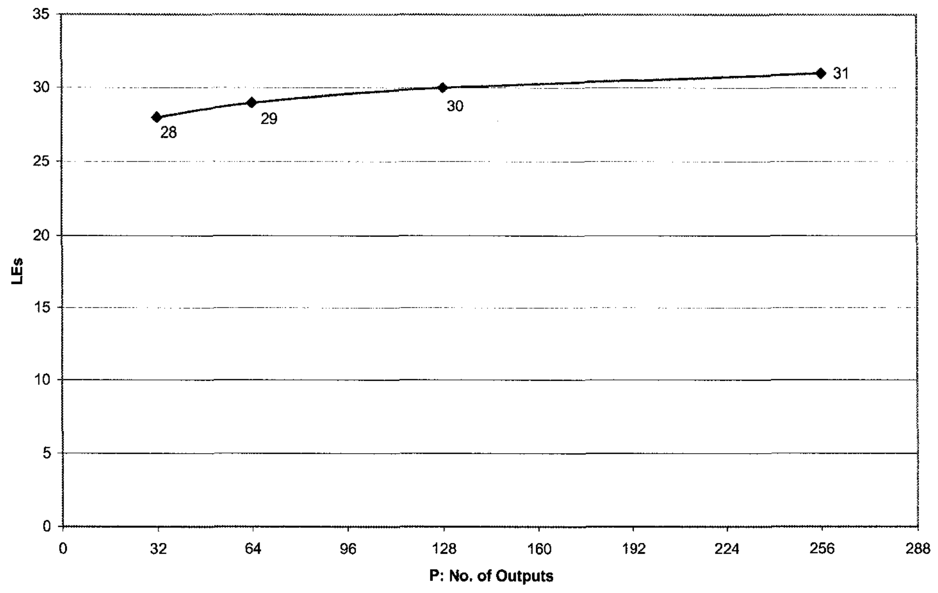

Figure 4.10: Number of Total Outputs vs. Area in LP

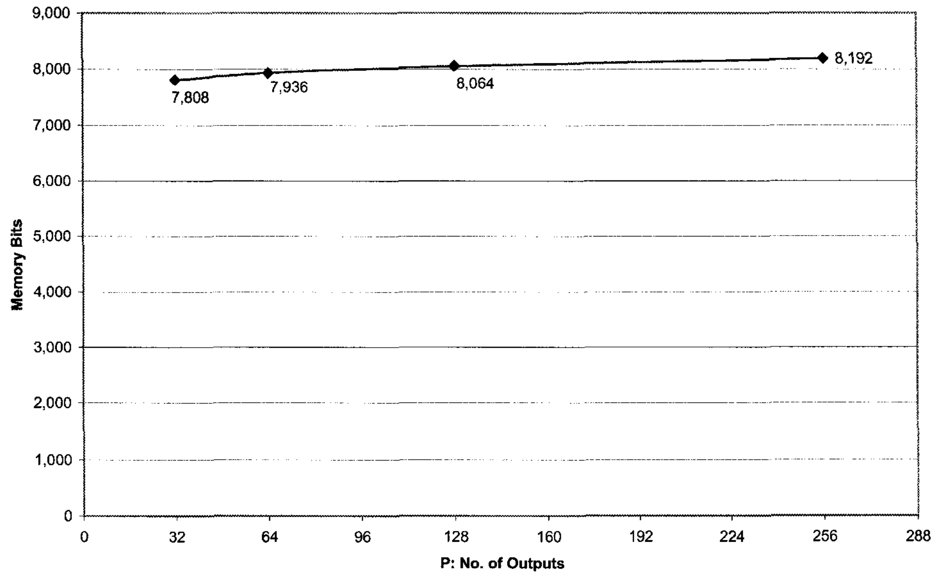

Figure 4.11: Number of Total Outputs vs. Memory Bits in LP

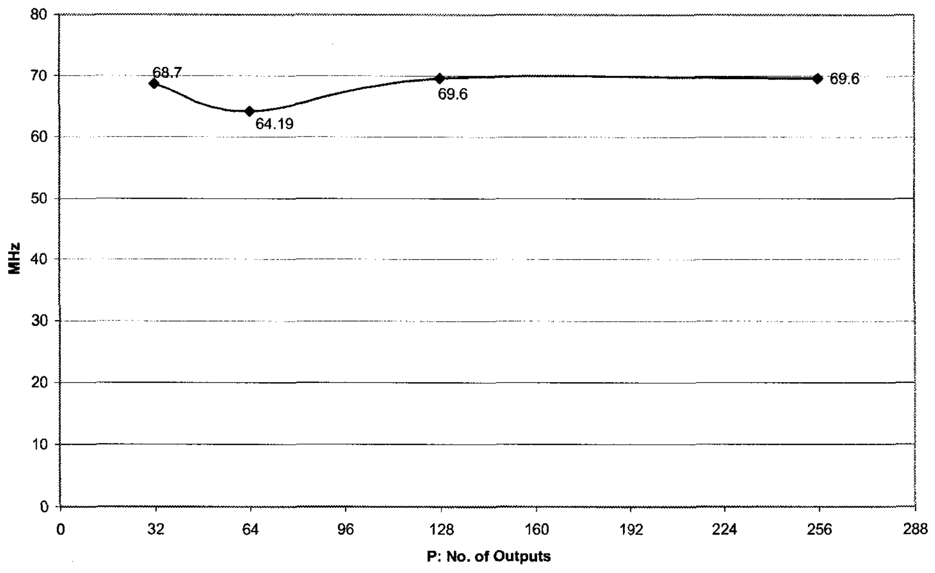

Figure 4.12: Number of Total Outputs vs. Speed in LP

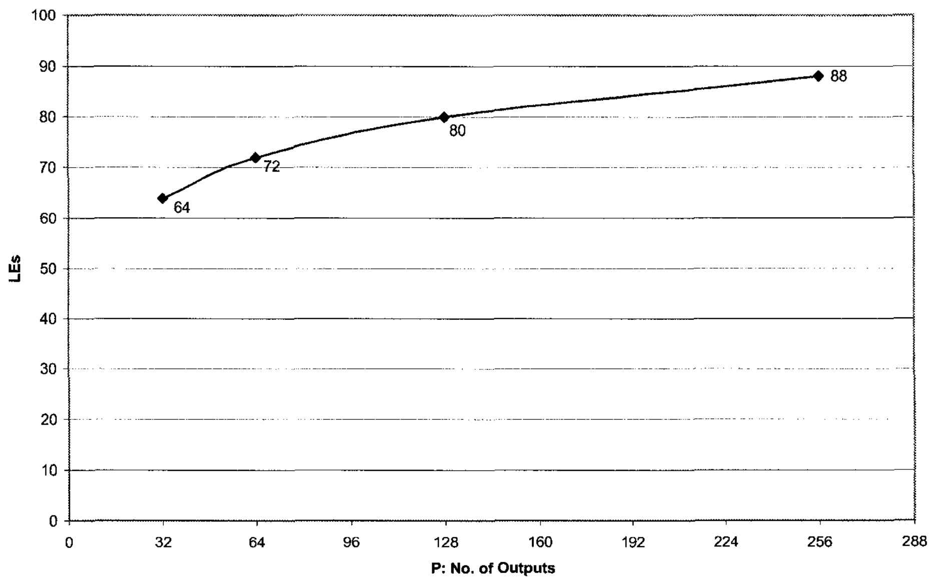

Figure 4.13: Number of Total Outputs vs. Area in MP

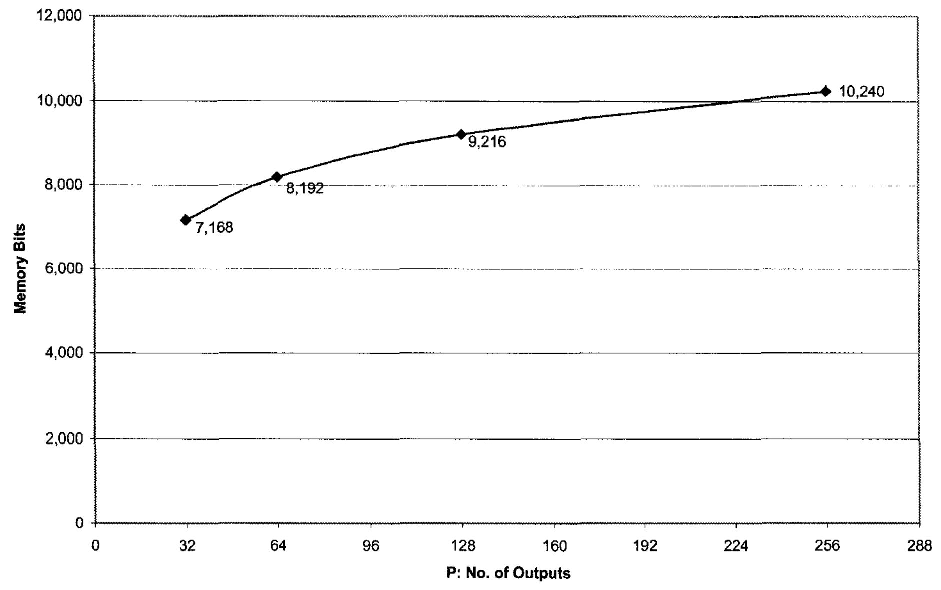

Figure 4.14: Number of Total Outputs vs. Memory Bits in MP

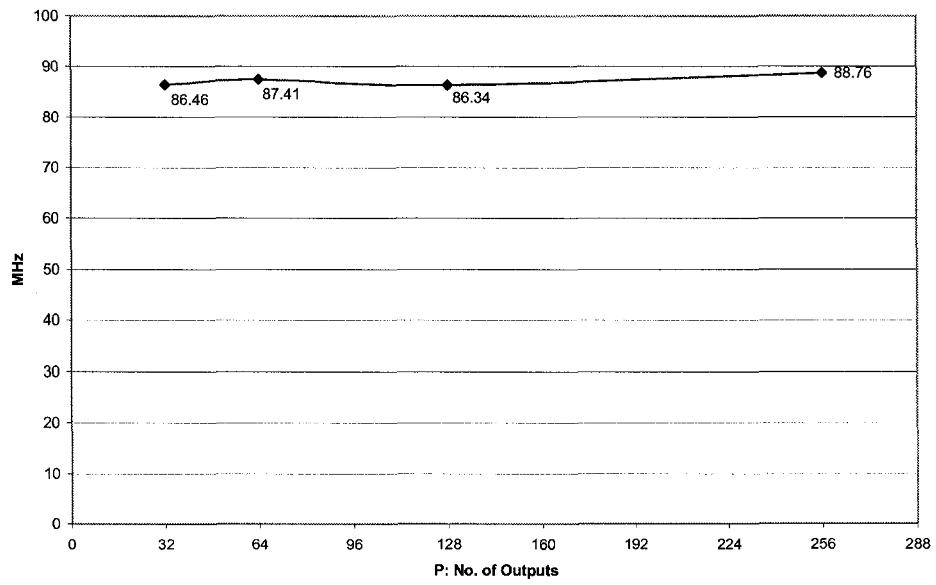

Figure 4.15: Number of Total Outputs vs. Speed in MP

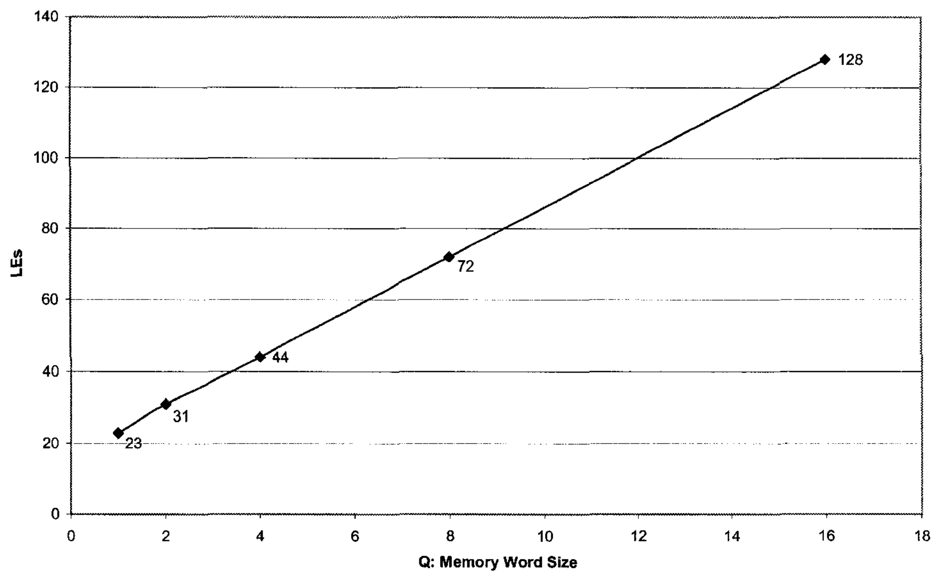

Figure 4.16: Memory Word Size vs. Area in MP

4.2.2.4 Effect of Changing Memory Word Size

The last key parameter to be checked for its effect is the memory word size, Q, which only affects the memory processor. Here the number of emulation steps, N, was fixed at 128 and the total number of outputs, P, was fixed at 64. The size of the memory word was varied from 1 to 16 in steps of power of 2. The results are shown in Figures 4.16, 4.17 and 4.18.

As can be seen in Figure 4.16 and Figure 4.17 increasing the size of the memory word had a linear effect on the implementation of the memory processor. This is expected since as the size of the memory word increases the combinational logic required for the capture and release memory word units increases. In addition, the size of the memory store where the memory word is stored increases as the size of the word increases.

As shown in Figure 4.18, the effect of the memory word size on the speed of the processor was limited; however, it was observed that there was a small decrease in

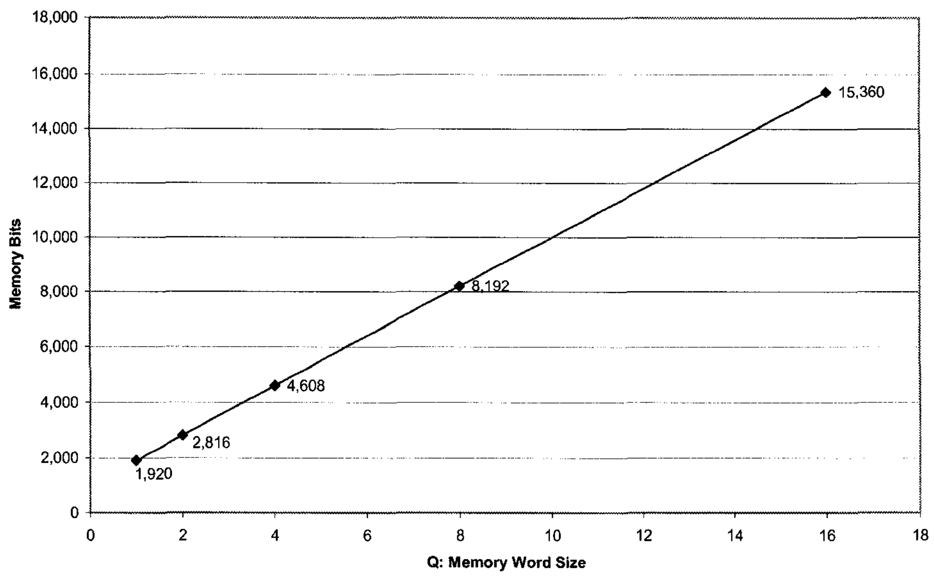

Figure 4.17: Memory Word Size vs. Speed in MP

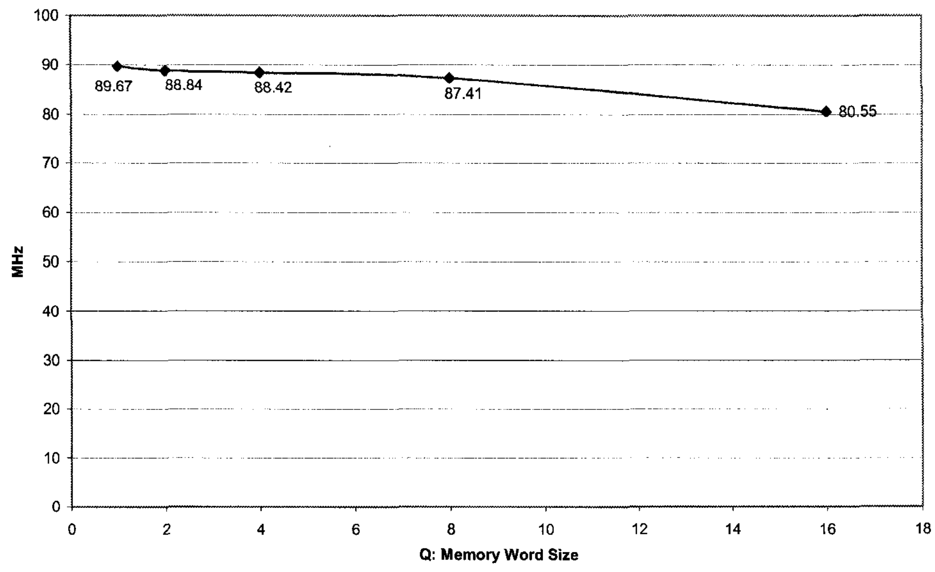

Figure 4.18: Memory Word Size vs. Speed in MP

processor speed when the memory word size was increased from 8 to 16. This decrease in speed could be attributed to the fact that the processing time for the memory word will take longer as its size increases.

4.2.3 Choice of Parameters

The choice of the parameters used in our implementation is based on the results obtained above. The following values were chosen:

- $M = 4$. The lookup table was chosen to be 4 because it provides good emulation speed and a low area cost.
- $N = 128$. The number of emulation steps was chosen to be 128. The reason behind this decision was the fact that FPGA resources are limited. M4K and M512 blocks both are of limited size and 128 words stored in each of them seems a reasonable size.
- $Q = 8$. The size of the memory word was chosen to be 8 mainly because of the emulation speed. As noted earlier the emulation speed was relatively stable until the memory word size was increased from 8 to 16 where the speed comparatively fell more.
- $P = 64$. The total number of outputs was chosen to be 64 which in effect meant 32 logic processors and 4 memory processors were packaged together in one module. The exploration did not show that any specific value of this parameter had a major effect on the area and performance of any of the processors.

4.3 Implementation Results

In this section we discuss the implementation results of the system. Note that the values for the parameters used here were the ones chosen above. The design tool that was used was Quartus II which is supplied by Altera [2]. The hardware description language that was used was VHDL [28].

4.3.1 Logic Processor

The elements of the logic processor were implemented as follows:

- *Control Store*: using $M = 4$, $N = 128$ and $P = 64$ the size of the control word would be 54 bits. This means that the size of the control store is 128×54. The control store has no combinational logic and only needs to be implemented as a memory block. The memory block chosen to implement the store was the M4K. To save on memory bits the control stores of two processors were combined together and implemented in 3 M4K blocks (each M4K is of size 128×36). A decoder was created to later separate the two words from each other.

- *Data Stacks*: since the number of emulation steps is 128 then the size of each stack is 128×1. Similar to the control store, both the internal and external stacks need no combinational logic and are implemented as memory blocks inside the FPGA. The memory blocks chosen to implement the stacks were the M512 blocks. Since each M512 block can supply at most two outputs at a time, each M512 was duplicated to ensure that 4 outputs can be supplied at the same time.

- *Logic Element*: the logic element is made up of purely combinational logic and requires no memory blocks. The 2-to-1 multiplexers and the lookup table, which is in turn a 4-to-1 multiplexer, were implemented in standard VHDL code used typically to describe multiplexers. The memory array of the lookup table was also implemented in logic elements and no memory blocks were used.

The implementation of each logic processor requires 1.5 M4K blocks, 4 M512 blocks and 29 FPGA logic elements.

4.3.2 Memory Processor

The elements of the memory processor were implemented as follows:

- *Control Store*: using $Q = 8$, $N = 128$ and $P = 64$ the size of the control word would be 56 bits. This makes the control store of size 128×56. The control store needs no combinational logic and is implemented in 2 M4K blocks.
- *Memory Store*: since the size of the memory word is 8 and the number of emulation steps is 128 then the size of the memory store is 128×8. Similar to the control store, the memory store needs no combinational logic and is implemented in one M4K block.
- *Capture and Release Memory Word Units*: the two units only require combinational logic. Their functional behavior was described using standard VHDL statements used in typical concatenation and breakup instructions.

The implementation of each memory processor requires 3 M4K blocks and 72 FPGA logic elements.

4.3.3 Emulation Module

Each emulation module contains 32 logic processors and 4 memory processors together having a total number of 64 outputs and sharing one interconnection network. Aside from these processors the module contains two other elements: the sequential filler and the module level routing switch. Both of these elements were implemented in VHDL and require no memory blocks.

Since the routing switch is made up of multiplexers, the VHDL code used to describe its functional behavior is that which is typically used to describe the functionality of multiplexers. As for the sequential filler its functional behavior was described in a series of conditional statements which determine which processor is being filled up before emulation starts.

4.3.4 Emulation Chip

Because of the limited resources of the FPGA each emulation chip in our design contains three modules. It requires all 384 M512 blocks, 180 M4K blocks (98% of all M4K blocks) and all 4 M-RAM blocks. The M-RAM blocks were used to store the selection bits for the multiplexers of the chip level routing switch.

The chip level routing switch, shown in Figure 4.19, connects all the module pins, external inputs and external outputs together. This switch is made up of 256 4-to-1 multiplexers and 64 2-to-1 multiplexers; one multiplexer for each input of the three modules and one for each external output. The switch allows inputs of processors inside one module to choose among certain outputs of other modules. For example, the input of processor 0 of module 0 can choose between: output of processor 0 in module 1, output of processor 0 in module 2, external input 0 or external input 1. The switch also allows external outputs to choose among outputs of certain processors or external inputs. For example, external output 0 can choose between: output of processor 0 in module 0, output of processor 0 in module 1, output of processor 0 in module 2, or external input 0.

The emulation chip consumes 10,579 logic elements and 933,888 memory bits. This puts the FPGA logic utilization at 25% and memory utilization at 27%. The key limitation in resources was due to the memory blocks, mainly the M512 and M4K blocks which were almost fully utilized by our design. The reason that the total memory utilization shows only 27% is due to the fact that the majority of the memory

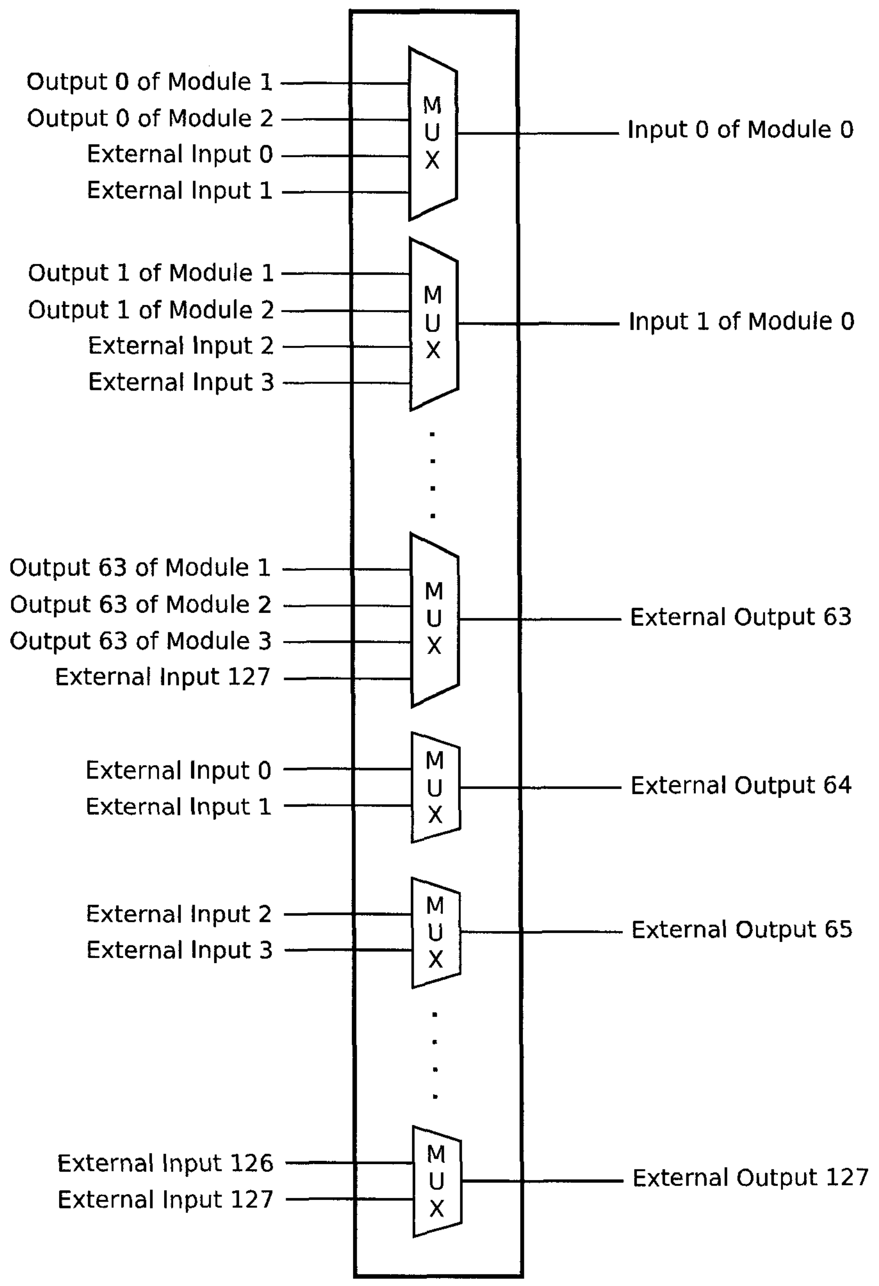

Figure 4.19: Module Level Routing Switch

bits available are stored in the M-RAM blocks which were only partially utilized. In addition to the logic elements and memory blocks, each emulation chip requires 531 pins, making the pin utilization 86%.

The whole design, one emulation chip, was made up of almost 6,100 VHDL lines describing the functional behavior of the combinational logic. The memory blocks were designed and implemented by means of megafunctions, a design tool supplied by Quartus II to save the time required to write the code in VHDL. The design contains 7 megafunctions [24].

Each emulation chip is capable of emulating what is equivalent to 98,304 ASIC gates per design cycle. This was calculated by assuming that each logic processor with a 4-input lookup table can implement 8 ASIC gates per emulation step and 1,024 ASIC gates per design cycle [10]. The emulation chip can also emulate 12,288 memory bits, which is the sum of all the bits stored in all the memory stores of the memory processors.

Lastly the emulation clock frequency of a single emulation chip is 24.04 MHz. The emulated design can run at 187.8 KHz or more depending on the number of emulation steps used in the design cycle.

4.4 Implementation Estimates for Emulation Engine

The implementation of this design only involved the second level of hierarchy, the emulation chip. The highest level of hierarchy, the emulation engine, was not implemented. In this section we give some estimates of the implementation of this engine.

A typical emulation engine would be made up of a fully connected multi-FPGA system, as the one shown in Figure 3.13. Here six FPGAs are connected together to act as an emulation engine. The logic capacity of this engine is equivalent to 589,824

ASIC gates and 73,728 memory bits.

As mentioned in chapter 3 the main issue we would need to deal with in such a system is the speed. The high pulse of the clock, symbolized by y in Figure 3.14, need to be long enough for the signal to traverse the longest path delay of the multi-FPGA system. On one Printed Circuit Board (PCB) we can assume that this delay is on average the same for all the connections.

To calculate this delay we assume that the dielectric used for the PCB is FR-4, the most widely used dielectric for PCBs [5]. This means that the propagation speed on the PCB is $1.48 \times 10^8\ m/s$ [1]. Therefore, the time needed by a signal to traverse half a meter, a typical size of a PCB, is approximately 3.4 ns. Following this logic y should be at least 3.4 ns to ensure that the signal has enough time to reach its destination.

If we choose to have a 50% duty cycle then the period of one clock cycle should be around 7 ns for the signal to traverse the longest path delay. However, it was mentioned before that the emulation clock frequency is 24.04 MHz and its period is 41.6 ns. It is clear that emulation clock period is much longer than the longest path delay and therefore the PCB connections would not add any extra delay and should not decrease the speed of the clock if the board remained of reasonable size.

4.5 Emulation Example

To illustrate and verify the operation of the emulator we chose to emulate a four bit multiplier on a single emulation chip.

4.5.1 Four-Bit Multiplier

The multiplier has two inputs each of size 4 bits and has one output of size 8 bits. Figure 4.20 shows the multiplication process. The goal is to give each operation of this

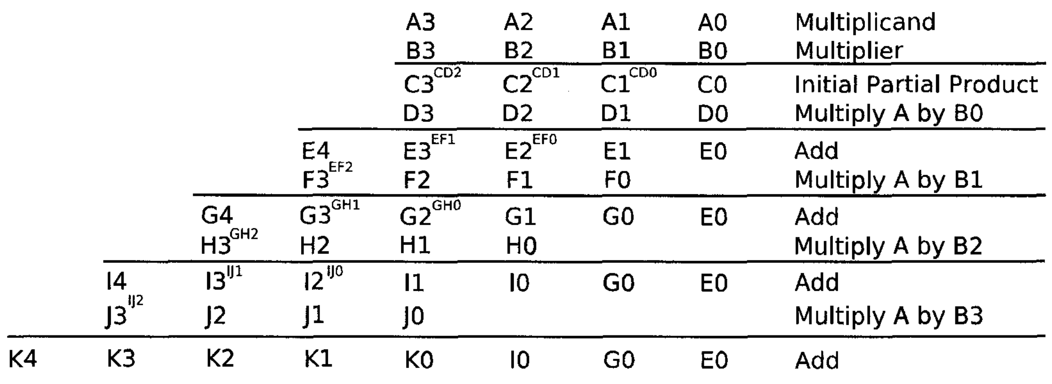
A3 A2 A1 A0 Multiplicand
B3 B2 B1 B0 Multiplier
C3^CD2 C2^CD1 C1^CD0 C0 Initial Partial Product
D3 D2 D1 D0 Multiply A by B0
E4 E3^EF1 E2^EF0 E1 E0 Add
F3^EF2 F2 F1 F0 Multiply A by B1
G4 G3^GH1 G2^GH0 G1 G0 E0 Add
H3^GH2 H2 H1 H0 Multiply A by B2
I4 I3^IJ1 I2^IJ0 I1 I0 G0 E0 Add
J3^IJ2 J2 J1 J0 Multiply A by B3
K4 K3 K2 K1 K0 I0 G0 E0 Add

Figure 4.20: Operation of the Four-Bit Multiplier

multiplication to one logic processor in a process known as scheduling. The symbols shown as superscripts are the overflow from the previous operations.

4.5.2 Scheduling and Implementation

Tables 4.1 and 4.2 shows the scheduling of the eight processors used for emulating the four bit multiplier. Normally the scheduling process would be automated but since our design lacks the CAD tools associated with it, the scheduling was done manually. It is important to note that this schedule might not be the most efficient one since the aim here is only to verify the functionality of the emulator. *Cap* and *Cal* in the tables stand for capture value and calculate value respectively.

Step	**LP0**	**LP1**	**LP2**	**LP3**
0	Cap(A0)		Cap(A1)	
1	Cap(B0)		Cap(B0)	
2	Cal(D0)		Cal(D1)	
3	Cal(E0), Cap(B1)		Cal(E1), Cap(B1)	Cap(E1)
4	Cal(F0),	Cap(F3)	Cal(F1),	Cap(F0)

	Cap(F3)		Cap(F0)	
5	Cap(B2)		Cal(G0), Cap(B2)	Cal(EF0)
6				
7	Cap(EF2)	Cap(EF2)		
8	Cal(G3)	Cal(G4), Cap(G3)	Cap(G4)	Cap(G4)
9	Cal(H0), Cap(H2)	Cap(H2)	Cal(H1), Cap(H3)	Cap(H3)
10				
11	Cap(GH1)	Cap(GH1)		
12	Cal(I2), Cap(B3)	Cal(GH2), Cap(I2)	Cap(GH2)	Cap(GH2)
13			Cal(I3), Cap(B3)	Cal(I4), Cap(I3)
14	Cal(J0), Cap(J1)	Cap(J1)	Cal(J1), Cap(J2)	Cap(J2)
15	Cap(IJ0)	Cap(IJ0)		
16	Cal(K1)	Cal(IJ1)	Cal(IJ1)	Cal(IJ1)
17			Cal(K2)	Cal(IJ2)
18				

Table 4.1: Multiplier Scheduling for Processors 0-3

Step	**LP4**	**LP5**	**LP6**	**LP7**
0	Cap(A2)		Cap(A3)	
1	Cap(B0)		Cap(B0)	
2	Cal(D2)		Cal(D3)	
3	Cal(E2), Cap(B1)	Cap(E2)	Cal(E3), Cap(B1)	Cap(E3)
4	Cal(F2), Cap(F1)	Cap(F1)	Cal(F3), Cap(F2)	Cap(F2)
5	Cap(EF0)	Cap(EF0)		
6	Cal(G1), Cap(B2)	Cal(EF1), Cap(G1)	Cap(EF1)	Cap(EF1)
7			Cal(G2), Cap(B2)	Cal(EF2), Cap(G2)
8				
9	Cal(H2), Cap(H0)	Cap(H0)	Cal(H3), Cap(H1)	Cap(H1)
10	Cal(I0), Cap(B3)	Cap(GH0)	Cap(GH0)	Cap(GH0)
11			Cal(I1), Cap(B3)	Cal(GH1), Cap(I1)
12				
13	Cap(I4)	Cap(I4)		

14	Cal(J2), Cap(J3)	Cap(J3)	Cal(J3), Cap(J0)	Cap(J0)
15			Cal(K0)	Cal(IJ0)
16				
17	Cap(IJ2)	Cap(IJ2)		
18	Cal(K3)	Cal(K4)		

Table 4.2: Multiplier Scheduling for Processors 4-7

After scheduling, the control words for each processor were generated and downloaded onto the processors. Several values were tested and the multiplier gave the correct results proving the validity of our design.

We should note the emulation of the multiplier was simulated and not downloaded on the FPGA. The reason behind that is that we lack the connection circuitry with the FPGA pins and building such a circuitry would be very time consuming. A more important reason is that we do not have a Data Capture Unit to read the outputs of the processors and therefore we cannot verify the operation.

Chapter 5

Conclusion and Future Work

The first section of this chapter summarizes the contributions made by this research. In section 2 we present a brief comparison between our design and previous processor-based emulator designs. We conclude in section 3 with some remarks on possible future work.

5.1 Research Contributions

The main contribution made by this research is the design and implementation of a low-cost processor-based logic emulation system. To reduce cost, the design was implemented using FPGA technology. Before implementation, architecture exploration experiments were conducted in order to choose suitable values for key architecture parameters. The proposed emulator can verify the functionality of logic designs at relatively high speeds and in real operating environments.

To increase logic capacity a fully connected multi-FPGA system can be used.

Each FPGA is programmed to act as an emulation chip. The full (multi-FPGA) design of the emulator was not implemented in this research. Only one emulation chip was implemented using a single FPGA. Each of these emulation chips is capable of emulating around 98 thousand ASIC gates and 12 thousand memory bits. It can run at a speed of almost 187 KHz per design cycle or more, depending upon the number of instruction cycles needed in one design cycle.

A four-bit multiplier was emulated to verify the correctness of the proposed emulator design. Because we lack the CAD tools for the bit stream generation, all the tasks of the CAD flow were carried out manually. The multiplier was emulated and verified the correct operation of the emulator.

5.2 Comparisons with Other Systems

To provide a context regarding the contributions made by this research, we present a brief comparison with previously proposed processor-based logic emulation systems.

In [15], a processor-based emulator was implemented using custom made chips. The building block of the system is a processor which can emulate both logic and memory functions. The two main differences between this design and ours are in architecture and implementation. In terms of architecture, the processor in this system performs both logic and memory emulation. In our design, two different processors are used one to emulate logic functions and the other to emulate memory functions. As for implementation, this design was implemented on custom made chips which makes it very expensive. In contrast, our processor-based emulator was implemented on FPGAs which would effectively make it a much lower cost system. We should note that there are several other similarities and differences in terms of operation and hierarchy.

In [10], a processor-based emulator was implemented on FPGAs. The emulator contains several kinds of processors. The main differences between this design and ours are in the design architecture and hierarchy. This design contains different kinds of processors in addition to logic and memory processors; our design only contains those two kinds. Another difference in architecture is in the interconnection network. The designer in this case chose to use buffers in the interconnections between the processors to give more flexibility to the CAD tools in terms of routing. We thought that using such buffers would consume the limited resources of the FPGA and decided to leave a tighter constraint on the CAD tools.

In comparison with commercially available emulators like the Incisive Palladium II [18] our emulator reached almost one eighth Palladium's top speed. Although speed might be a drawback, our design has a lower cost because of FPGA implementation.

5.3 Future Work

In our research we focused on the hardware architecture of a processor-based logic emulation system. The other main part is the mapping CAD tools that are required for a real world emulation system. The next step would be to design and develop the mapping CAD tools for this system. The mapping CAD tools would compile the logic design of the DUT and generate the bit stream which could be downloaded to the programmable hardware.

Another future work in the hardware part of the project might involve designing a data capture unit which would help the designer in finding errors and automate the checking process.

References

[1] Altera Corporation. *High Speed Board Designs*, November 2001.

[2] Altera Corporation. *Introduction to Quartus II Version 5.0*, April 2005.

[3] Altera Corporation. *Stratix Device Handbook*, July 2005.

[4] Altera Corporation. http://www.altera.com/, Accessed August 2007.

[5] Altera Corporation. *Stratix II Device Handbook*, May 2007.

[6] Altera Corporation. *Stratix III Device Handbook*, May 2007.

[7] J. Babb, T. Russell, M. Dahl, S. Z. Hanono, D. M. Hoki, and A. Agrawal. Logic emulation with virtual wires. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(6):609–626, June 1997.

[8] K. Banovic, M. A. S. Khalid, and E. Abdel-Raheem. FPGA-based rapid prototyping of digital signal processing systems. In *Proc. of the 48th Mid-West Symposium on Circuits and Systems*, pages 647–650, August 2005.

[9] M. Butts. Future directions of dynamically reprogrammable systems. In *Proc. of IEEE Custom Integrated Circuits Conference*, pages 487–494, 1995.

[10] M. R. Butts. Logic multiprocessor for FPGA implementation. U.S. Patent Application 2004/0123258 A1, June 2004.

[11] Cadence Design Systems Incorporated. http://www.cadence.com/, Accessed August 2007.

[12] E. M. Clarke and R. P. Kurshan. Computer-aided verification. *IEEE Spectrum*, 33(6):61–67, June 1996.

[13] K. Compton and S. Hauck. Reconfigurable computing: A survey of systems and software. *ACM Computing Surveys*, 34(2):171–210, June 2002.

[14] C. Edwards. Tracking down the chip killers. *IEE Review*, 50(12):44–46, December 2004.

[15] Beausoleil et al. Multiprocessor for hardware emulation. U.S. Patent 5551013, August 1996.

[16] H. Goldstein. Checking the play in plug-and-play. *IEEE Spectrum*, 39(6):50–55, June 2002.

[17] S. Hauck. The roles of FPGA's in reprogrammable systems. *Proceedings of the IEEE*, 86(4):615–638, April 1998.

[18] Incisive Palladium II. http://www.cadence.com/products/functional_ver/acel_emul/pseries.aspx, Accessed August 2007.

[19] P. H. Kelly, K. J. Page, and P. M. Chau. Rapid prototyping of ASIC based system. In *Proc. of the 31st ACM/IEEE Design Automation Conference*, pages 460–465, June 1994.

[20] C. Kern and M. R. Greenstreet. Formal verification in hardware design: A survey. *ACM Transactions on Design Automation of Electronic Systems*, 4(2):123–193, April 1999.

[21] M. A. S. Khalid and J. Rose. A novel and efficient routing architecture for multi-FPGA systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(1):30–39, February 2000.

[22] D. MacMillen, M. Butts, R. Camposano, D. Hill, and T.W. Williams. An industrial view of electronic design automation. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 19(12):1428–1448, December 2000.

[23] K. L. McMillan. Fitting formal methods into the design cycle. In *Proc. of the 31st ACM/IEEE Design Automation Conference*, pages 314–319, June 1994.

[24] Altera Megafunctions. http://www.altera.com/products/ip/altera/mega.html, Accessed August 2007.

[25] Mentor Graphics Corporation. http://www.mentor.com/, Accessed August 2007.

[26] G. E. Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, January 1998.

[27] R. Murgai and M. Fujita. Some recent advances in software and hardware logic simulation. In *Proc. of the 10th IEEE International Conference on VLSI Design*, pages 232–238, January 1997.

[28] Institute of Electrical and Electronics Engineers. IEEE standard VHDL language reference manual, ANSI/IEEE Std 1076-1993, 1993.

[29] C. Pixley, A. Chittor, F. Meyer, S. McMaster, and D. Benua. Functional verification 2003: Technology, tools and methodology. In *Proc. of the 5th IEEE International Conference on ASIC*, pages 1–5, October 2003.

[30] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli. Architecture of field-programmable gate arrays. *Proceedings of the IEEE*, 81(7):1013–1029, July 1993.

[31] L. Soule and T. Blank. Parallel logic simulation on general purpose machines. In *Proc. of the 25th ACM/IEEE Design Automation Conference*, pages 166–171, June 1988.

[32] J. Varghese, M. Butts, and J. Batcheller. An efficient logic emulation system. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 1(2):171–174, June 1993.

[33] VStationPRO. http://www.mentor.com/products/fv/emulation/vstation_pro/, Accessed August 2007.

[34] Xilinx Incorporated. http://www.xilinx.com/, Accessed August 2007.

[35] Xilinx Incorporated. *Vertex-5 Family Overview - LX, LXT, and SXT Platforms*, May 2007.

VITA AUCTORIS

Marwan Kanaan was born in Haret Hreik, Lebanon, in 1983. He received his B.E. in computer and communications engineering in 2005 from the American University of Beirut in Beirut, Lebanon. He is currently a candidate in the electrical and computer engineering M.A.Sc. program at the University of Windsor. His research interests include logic emulation systems, field-programmable technologies and digital design.