


2004

## Cryptarray A Scalable And Reconfigurable Architecture For Cryptographic Applications

Michael John Lomonaco  
*University of Central Florida*

 Part of the [Electrical and Computer Engineering Commons](#)  
Find similar works at: <https://stars.library.ucf.edu/etd>  
University of Central Florida Libraries <http://library.ucf.edu>

This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations, 2004-2019 by an authorized administrator of STARS. For more information, please contact [STARS@ucf.edu](mailto:STARS@ucf.edu).

---

### STARS Citation

Lomonaco, Michael John, "Cryptarray A Scalable And Reconfigurable Architecture For Cryptographic Applications" (2004). *Electronic Theses and Dissertations, 2004-2019*. 141.  
<https://stars.library.ucf.edu/etd/141>

**CRYPTARRAY  
A SCALABLE AND RECONFIGURABLE ARCHITECTURE  
FOR  
CRYPTOGRAPHIC APPLICATIONS**

by

**MICHAEL LOMONACO**  
B.S. University of Central Florida, 1999

A thesis submitted in partial fulfillment of the requirements  
for the degree of Master of Science  
in the Department of Electrical and Computer Engineering  
in the College of Engineering and Computer Science  
at the University of Central Florida  
Orlando, Florida

Summer Term  
2004

© 2004 Michael Lomonaco

## **ABSTRACT**

Cryptography is increasingly viewed as a critical technology to fulfill the requirements of security and authentication for information exchange between Internet applications. However, software implementations of cryptographic applications are unable to support the quality of service from a bandwidth perspective required by most Internet applications. As a result, various hardware implementations, from Application-Specific Integrated Circuits (ASICs), Field-Programmable Gate Arrays (FPGAs), to programmable processors, were proposed to improve this inadequate quality of service. Although these implementations provide performances that are considered better than those produced by software implementations, they still fall short of addressing the bandwidth requirements of most cryptographic applications in the context of the Internet for two major reasons:

- (i) The majority of these architectures sacrifice flexibility for performance in order to reach the performance level needed for cryptographic applications. This lack of flexibility can be detrimental considering that cryptographic standards and algorithms are still evolving.
- (ii) These architectures do not consider the consequences of technology scaling in general, and particularly interconnect related problems.

As a result, this thesis proposes an architecture that attempts to address the requirements of cryptographic applications by overcoming the obstacles described in (i) and (ii).

To this end, we propose a new reconfigurable, two-dimensional, scalable architecture, called *CRYPTARRAY*, in which bus-based communication is replaced by distributed shared memory communication. At the physical level, the length of the wires will be kept to a minimum. *CRYPTARRAY* is organized as a chessboard in which the dark and light squares represent Processing Elements (PE) and memory blocks respectively. The granularity and resource composition of the PEs is specifically designed to support the computing operations encountered in cryptographic algorithms in general, and symmetric algorithms in particular. Communication can occur only between neighboring PEs through locally shared memory blocks. Because of the chessboard layout, the architecture can be reconfigured to allow computation to proceed as a pipelined wave in any direction. This organization offers a high computational density in terms of datapath resources and a large number of distributed storage resources that easily support a high degree of parallelism and pipelining. Experimental prototyping a small array on FPGA chips shows that this architecture can run at 80.9 MHz producing 26,968,716 outputs every second in static reconfiguration mode and 20,226,537 outputs every second in dynamic reconfiguration mode.

## **ACKNOWLEDGMENTS**

Thanks to Dr. Ejnoui for all of your help and patience with this project and with my busy schedule. It has been an honor and a pleasure getting to know you.

Thanks to the committee members Dr. Taskin Kocak and Dr. Ronald DeMara.

Thanks to my wife Amanda for supporting me emotionally and sometimes financially, as well as providing laughter and encouragement throughout the graduate school process.

Thanks to God for guiding me to and through graduate school, an undertaking that I would not have pursued or completed if not for His guidance and blessings.

## TABLE OF CONTENTS

<b>TABLE OF CONTENTS .....</b>	<b>vi</b>
<b>LIST OF FIGURES .....</b>	<b>viii</b>
<b>LIST OF TABLES .....</b>	<b>x</b>
<b>CHAPTER ONE: INTRODUCTION .....</b>	<b>1</b>
1.1 Cryptographic Applications .....	1
1.1.1 Symmetric Algorithms .....	2
1.1.2 Asymmetric Algorithms .....	3
1.2 Cryptographic Hardware Systems .....	5
1.3 CMOS Technology Scaling .....	8
1.3.1 Gate Delay Scaling .....	8
1.3.2 Wire Delay Scaling .....	9
1.4 Architectural Implications .....	11
1.5 Thesis Contribution .....	12
1.6 Thesis Outline .....	13
<b>CHAPTER TWO: RELATED WORK .....</b>	<b>14</b>
2.1 Cryptographic Systolic and VLSI Architectures .....	14
2.2 Cryptographic Programmable Processors .....	15
2.3 Cryptographic FPGA Designs .....	17
2.4 Summary .....	18
<b>CHAPTER THREE: CRYPTARRAY .....</b>	<b>19</b>
3.1 Layout of CRYPTARRAY .....	19
3.2 Shared Memory Blocks .....	20

3.3 PE Organization .....	25
3.4 PE Instructions .....	27
3.5 PE Reconfiguration .....	31
<b>CHAPTER FOUR: PE ARCHITECTURE .....</b>	<b>35</b>
4.1 Architectural Components of the PE .....	35
4.2 Architecture of Block 1 .....	37
4.3 Architecture of Block 2 .....	39
4.4 Architecture of Block 3 .....	41
4.5 Architecture of Block 4 .....	42
4.6 Architecture of Block 5 .....	43
4.7 PE State Control .....	44
<b>CHAPTER FIVE: MODELING AND IMPLEMENTATION OF</b>	
<b>CRYPTARRAY .....</b>	<b>47</b>
5.1 Modeling of CRYPTARRAY .....	47
5.2 Verification of the VHDL Entities .....	49
5.3 Synthesis of the PE Model .....	56
5.4 Synthesis of the SMB Model .....	64
5.5 Performance of CRYPTARRAY .....	65
<b>5.5.1 Clock Frequency .....</b>	<b>65</b>
<b>5.5.2 Area Cost .....</b>	<b>66</b>
<b>5.5.3 Possible Bandwidth .....</b>	<b>67</b>
<b>5.5.4 Summary of CRYPTARRAY's Performance .....</b>	<b>67</b>
<b>CHAPTER SIX: CONCLUSION .....</b>	<b>69</b>
<b>REFERENCES .....</b>	<b>72</b>



## LIST OF FIGURES

Figure 1: Encryption and decryption [2].....	2
Figure 2: Symmetric cryptography [4]. ....	3
Figure 3: Asymmetric cryptography [4]. ....	4
Figure 4: SSL characterizations by session length. ....	6
Figure 5: FO4 delay scaling.....	9
Figure 6: An FO4 delay. ....	9
Figure 7: Short and global wires.....	10
Figure 8: Wire delay in FO4 for scaled-length wires spanning 50K gates.....	10
Figure 9: Wire delay in FO4 for fixed-length wires 1 cm long. ....	11
Figure 10: Checkerboard layout of CRYPTARRAY. ....	20
Figure 11: Connectivity of the SMB to its four surrounding PEs.....	22
Figure 12: Structural Organization of the PE. ....	26
Figure 13: Formats of the block instructions in a PE.....	27
Figure 14: Instruction dispatch and capture by a PE. ....	32
Figure 15: Hierarchical encoding of the instructions.....	33
Figure 16: Architectural components of a single PE. ....	37
Figure 17: Architecture of block 1.....	38
Figure 18: Architecture of block 2.....	40
Figure 19: Architecture of block 3.....	41

Figure 20: Architecture of block 4.....	42
Figure 21: Architecture of block 5.....	43
Figure 22: State diagram of the PE controller. ....	45
Figure 23: State cycling in the blocks of a PE. ....	46
Figure 24: Instruction path through the PE simulation. ....	50
Figure 25: Simulation of the block 1 module. ....	51
Figure 26: Simulation of the block 3 module. ....	52
Figure 27: Simulation of the block 4 module. ....	53
Figure 28: Simulation of the block 5 module. ....	54
Figure 29: Simulation of the state machine. ....	55
Figure 30: PE implementation of a timing-driven low-effort mapping.....	58
Figure 31: PE implementation of a timing-driven fast-effort mapping. ....	59
Figure 32: PE implementation of a timing-driven high-effort mapping.....	59
Figure 33: PE implementation of an area-driven low-effort mapping.....	60
Figure 34: PE implementation of an area-driven fast-effort mapping.....	61
Figure 35: PE implementation of an area-driven high-effort mapping.....	61
Figure 36: Summary of PE implementations.....	62
Figure 37: Partial view of the floorplan of a synthesized PE onto a Virtex-II Pro XC2VP125. ..	63
Figure 38: Partial view of the floorplan of a synthesized SMB onto a Virtex-II Pro XC2VP125. .....	64

## LIST OF TABLES

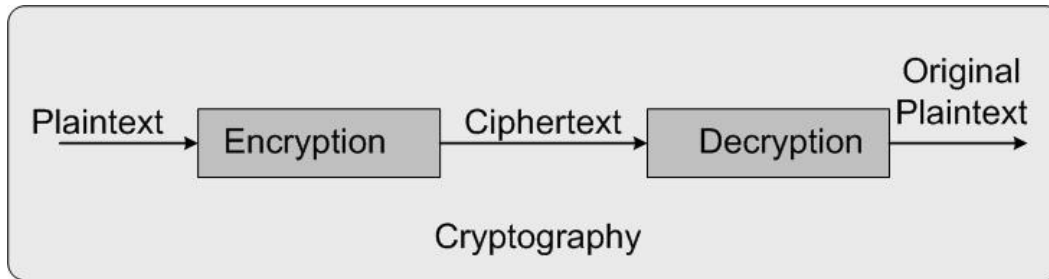
Table 1: Summary of secret and public key attributes [4]. .....	5
Table 2: Possible configurations of read/write ports between an SMB and its four surrounding PEs. ....	23
Table 3: Comparison of the five SMB-PE interface configurations. ....	24
Table 4: Block-level and PE-level encoding of operations. ....	29
Table 5: Breakdown of VHDL lines of code based on the modeled entities. ....	47
Table 6: Area and Speed optimization of the PE under three mapping effort levels. ....	56
Table 7: Synthesis results of a 512 x 4-bit SMB. ....	65
Table 8: Summary of the performance characteristics of CRYPTARRAY's components. ....	68

## CHAPTER ONE: INTRODUCTION

The fast pace of advancement in semiconductor integration and fabrication spurred the development of computing applications that began to shift from client-server based computing confined inside private networks to the world-wide open connectivity of the Internet. This shift to an Internet-based computing mandated that the Internet becomes a secure vehicle for communication and electronic commerce. As a result, cryptography and its various applications became an essential component of modern information systems. Semantically, cryptography is the art of writing secrets [1]. In practice, cryptography encodes information using an encryption process, into a form that is incomprehensible to anyone except to the intended recipient, who can then decode the original information using a secret key, a process called decryption [2].

### 1.1 Cryptographic Applications

The science of *cryptography* refers to the study of methods for sending messages in *secret*, namely in *enciphered* or *disguised* form, so that only the intended recipient can remove the disguise and read the message or *decipher* it. The original message is called the *plaintext* while the disguised message is called the *ciphertext*. The final sent message is called a *cryptogram*. The process of transforming plaintext into ciphertext is called *encryption* or *enciphering*. The reverse process of turning the ciphertext into plaintext is called *decryption* or *deciphering* [3]. In general, cryptosystems can be broadly classified into symmetric and asymmetric algorithms.



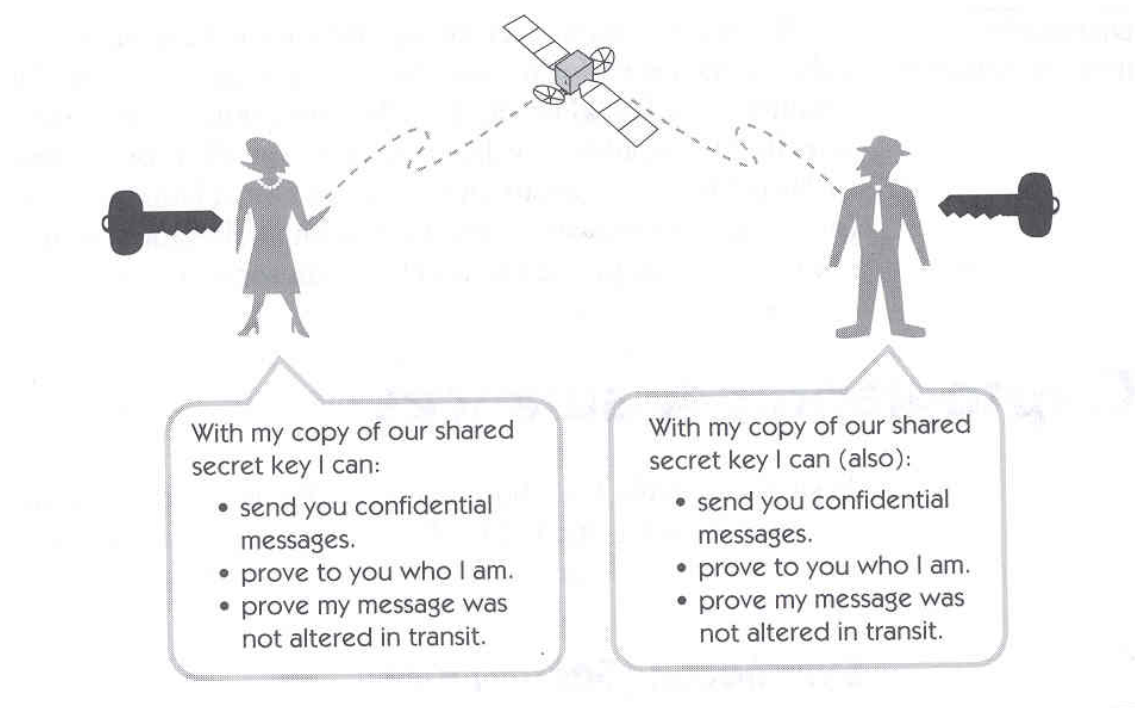
**Figure 1: Encryption and decryption [2].**

### 1.1.1 Symmetric Algorithms

The symmetric or secret-key algorithms, such as DES, IDEA, and SAFER require that the sender and receiver share the same secret key that is used to encrypt and decrypt the messages exchanged between both.

*Definition 1: A cryptosystem is called symmetric-key if for each key pair  $(e, d)$ , the key is “computationally easy” to determine knowing only  $e$  and to similarly determine  $d$  knowing only  $e$  [3].*

It is meant by a *computationally easy problem* a problem that can be solved in expected polynomial time and can be attacked using available resources. Symmetric algorithms can be subdivided into stream ciphers or block ciphers. Stream ciphers are algorithms that operate on the plaintext a single bit at a time, and block ciphers are algorithms that operate on the plaintext in groups of bits or blocks. In general, secret key cryptography implements confidentiality, authentication, and integrity for both holders of the secret key.



**Figure 2: Symmetric cryptography [4].**

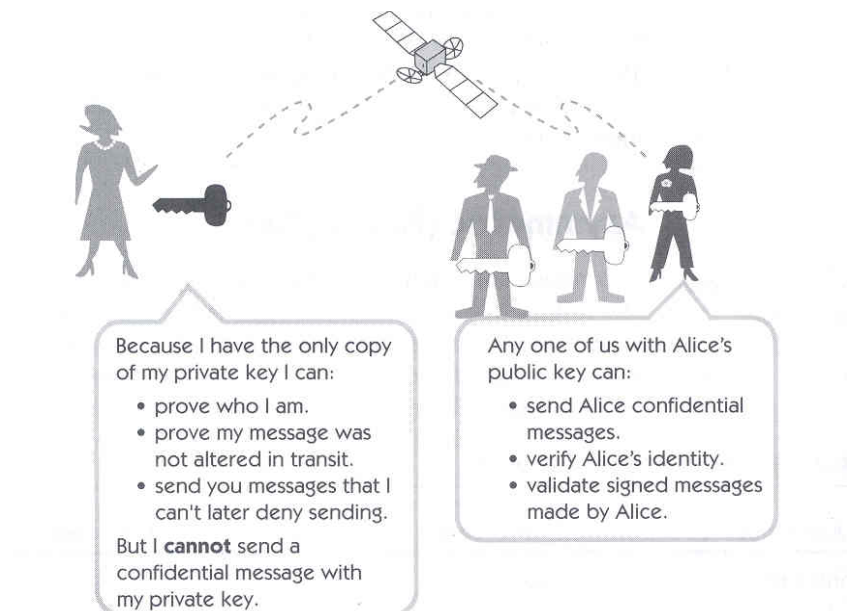
### 1.1.2 Asymmetric Algorithms

On the other hand, asymmetric or public-key algorithms, such as RSA, rely on a public key that is stored in the open and can be used by anyone to encrypt a message. A private key is generated from the public key and then used by the recipient to decrypt the message.

*Definition 2: A cryptosystem consisting of a set of enciphering transformations  $\{C_e\}$  and a set of deciphering transformations  $\{D_d\}$  is called an asymmetric or public-key if, for each key pair  $(e, d)$ , the enciphering key  $e$ , called the public key, is made publicly available, while the deciphering*

key  $d$ , called the private key, is kept secret. The cryptosystem must satisfy the property that it is computationally infeasible to compute  $d$  from  $e$  [3].

It is meant by a *computationally infeasible problem* a problem that, given the enormous amount of computer time that would be required to solve the problem, this problem cannot be solved in realistic computational time. Thus, *computationally infeasible* means that, although there theoretically exist a unique solution to the problem, this solution cannot be found even if all the available time and resources are devoted to its discovery. In contrast to symmetric algorithms, asymmetric algorithms allow confidentiality, authentication, integrity, and nonrepudiation to be asymmetrically shared among key holders. Table 1 shows a summary of the attributes of symmetric and asymmetric algorithms.



**Figure 3: Asymmetric cryptography [4].**

**Table 1: Summary of secret and public key attributes [4].**

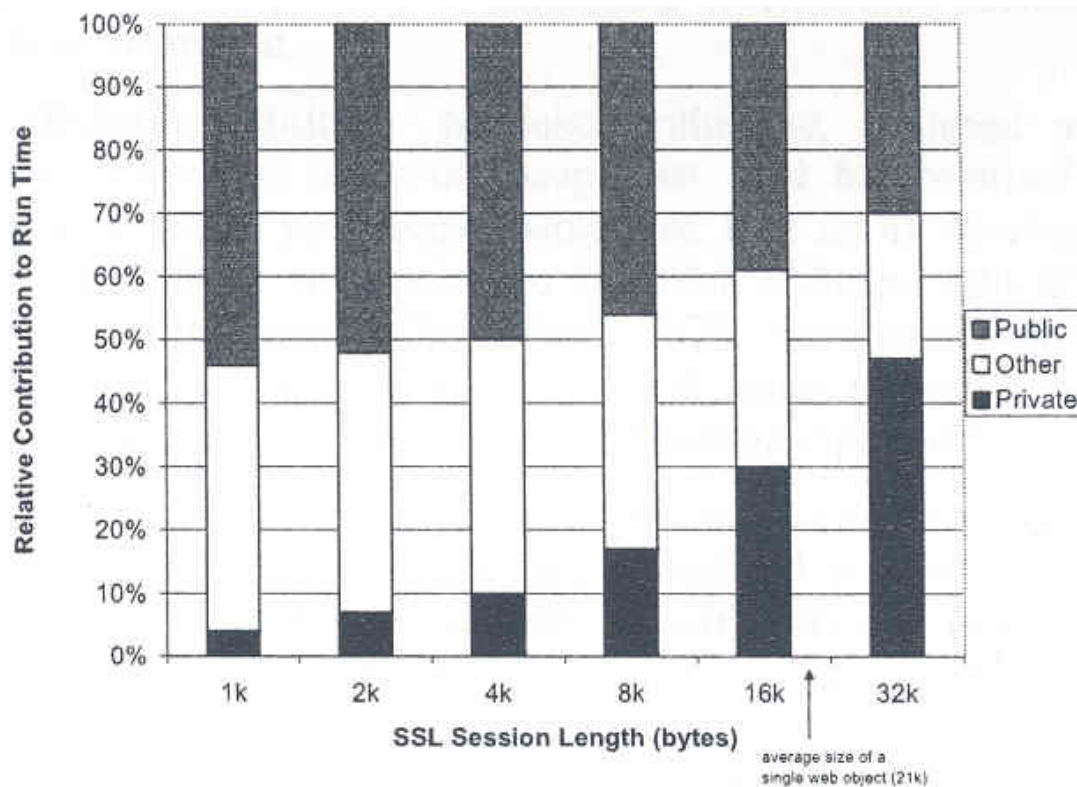
Attribute	Symmetric Cryptosystem	Asymmetric Cryptosystem
Years in use	Thousands	Less than 50
Current main use	Bulk data encryption	Key exchange, digital signatures
Current standard	DES, Triple DES, and Rijndael	RSA, Diffie-Hellman, DSA (Elliptic curve)
Encryption/decryption speed	Fast	Slow
Keys	Shared secret between at least two persons	<u>Private:</u> Key concealed by one person  <u>Public:</u> Key widely distributed
Key exchange	Difficult and risky to transfer a secret key	Easy and less risky to deliver a public key  Private key never shared
Key length	56-bit obsolete  128-bit considered safe	1024 suggested (RSA)  Some users demand 2048 bits
Confidentiality, authentication, message integrity	Yes	Yes
Nonrepudiation	No  Need trusted third party to act as witness	Yes  <u>Digital signatures:</u> No need for a trusted third party
Attacks	Yes	Yes

## **1.2 Cryptographic Hardware Systems**

Early efforts of integrating cryptography into current information systems were software implementations. Although some implementations can deliver satisfactory performance, most cannot address the bandwidth requirements of many applications that rely on cryptography to secure data integrity. In some instances, security-related processing can consume as much as 95% of a server's processing capacity [5]. Today, most secure information systems establish communication sessions during which information is exchanged. These sessions are usually initialized by exchanging keys which are used for encrypting and decrypting exchanged



information. For instance, the Secure Socket Layer (SSL) protocol extends TCP/IP protocol by supporting secure encrypted connections with authentication of senders and receivers. Web servers and browsers use this protocol to establish secure HTTP connections. At the start of a session, a public key is exchanged to authenticate the identity of the sender and receiver. In the remainder of the session, only private key encryption/decryption will be used to exchange content. Figure 4 shows the relative costs of symmetric and asymmetric cryptography in a web server [6]. The numbers shown in the figure were obtained for a heavily loaded web server running on an Itanium iA32 platform.



**Figure 4: SSL characterizations by session length.**

It is clear that for short sessions, fast asymmetric cipher processing is needed to insure high throughput while symmetric cipher processing is important for longer sessions. As secure communication requires increasingly larger bandwidths, the performance of cryptographic applications becomes critical to overall system performance. Recently, several efforts went into overcoming the shortcomings of software implementations by mapping cryptographic algorithms directly into hardware. These efforts evolved in three different directions:

- (i) Extension of the instruction sets of general purpose processors to support specific operations that are frequent in cryptographic algorithms, but execute inefficiently in these processors [7, 8].
- (ii) Implementation of specific algorithms or complex arithmetic functions as hardware cores that can be incorporated into an ASIC or mapped onto an FPGA [9-11].
- (iii) Design of programmable processors optimized for cryptography [12-14].

Although the approach in (i) can enhance the performance of general-purpose processors, it is doubtful that it can accommodate the bandwidth requirements of new communication systems. The approach in (ii) can deliver superior performance, but it does not offer any flexibility if future modifications to the initial cryptographic algorithm need to be added. This is quite restrictive given the fact that most cryptographic algorithms are still evolving at a faster rate in order to withstand the rigors of cryptanalysis [14]. The approach in (iii) is attractive since it offers a great degree of flexibility and performance. Although their performance can be quite significant, programmable processors fall short of the great potential that can be achieved should

their design take into consideration the physical realities imposed by the scaling of CMOS technology [15].

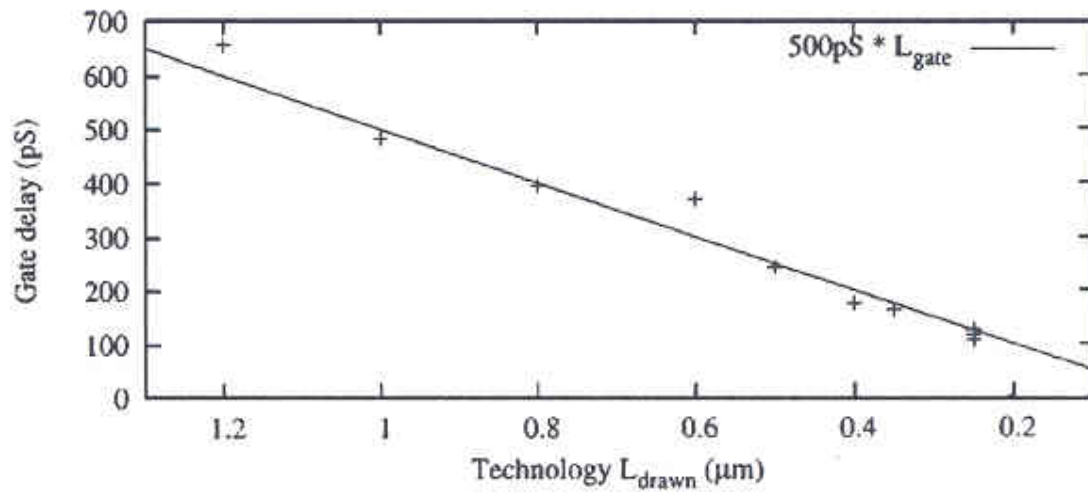
### **1.3 CMOS Technology Scaling**

The continuous scaling of CMOS technology shifted the focus of computer architecture from gate performance to wire performance. In general, wires delay kept increasing as transistors kept shrinking.

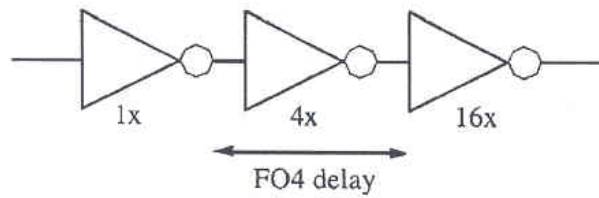
#### **1.3.1 Gate Delay Scaling**

Historical records of the characterizations of various CMOS processes show that gate delay has scaled linearly with technology. Figure 5 shows the gate delay in different process technologies running under the worst environmental conditions (125°C, 90%  $V_{dd}$ ). In the figure, the gate delay is expressed in FO4, a “fanout-of-four inverter delay” [15].

An FO4 delay is the delay through an inverter that is driving four copies of itself as shown in Figure 6 [15]. Designers use this simple metric to overcome the complexity of characterizing delay in transistor devices. For example, an FO4 is about 90 picoseconds in a 0.18  $\mu\text{m}$  process under worst environmental conditions characterized by a high temperature and low  $V_{dd}$ .



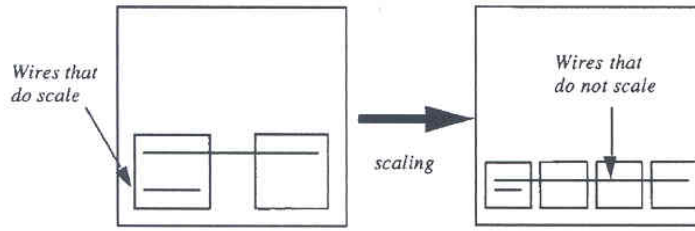
**Figure 5: FO4 delay scaling.**



**Figure 6: An FO4 delay.**

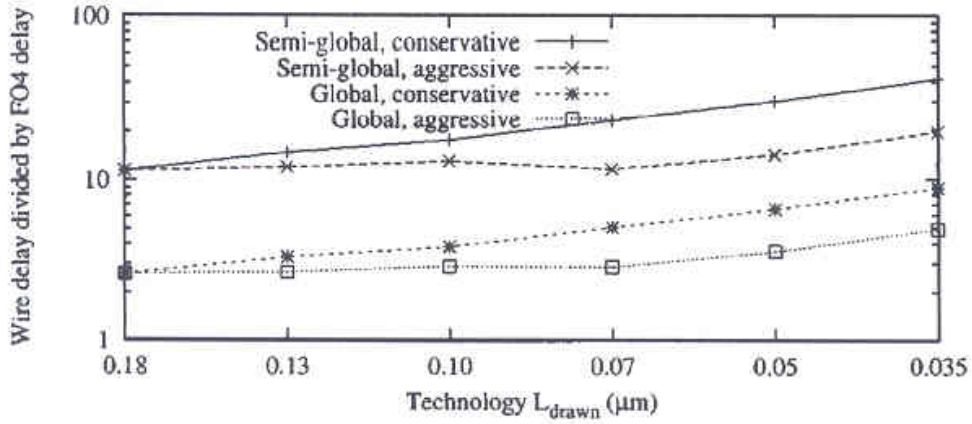
### 1.3.2 Wire Delay Scaling

Most technology studies show that chip architectures tend to use two types of wires as shown in Figure 7, where the first type connects gates locally inside the blocks while the second type connects blocks together [15]. The first type consists of short wires while the second type consists of global wires.



**Figure 7: Short and global wires.**

These studies show that short wires exhibit a constant wire resistance and a falling wire capacitance with regard to length scaling factors as shown in Figure 8. The figure shows the delay of a wire that spans at most a block of 50,000 gates [15]. However, the same studies show that the delay of global wires displays a large disparity with the delay in gates. Figure 9 shows the delay of 1-cm long wire relative to gate delay on a log scale [15].



**Figure 8: Wire delay in FO4 for scaled-length wires spanning 50K gates.**

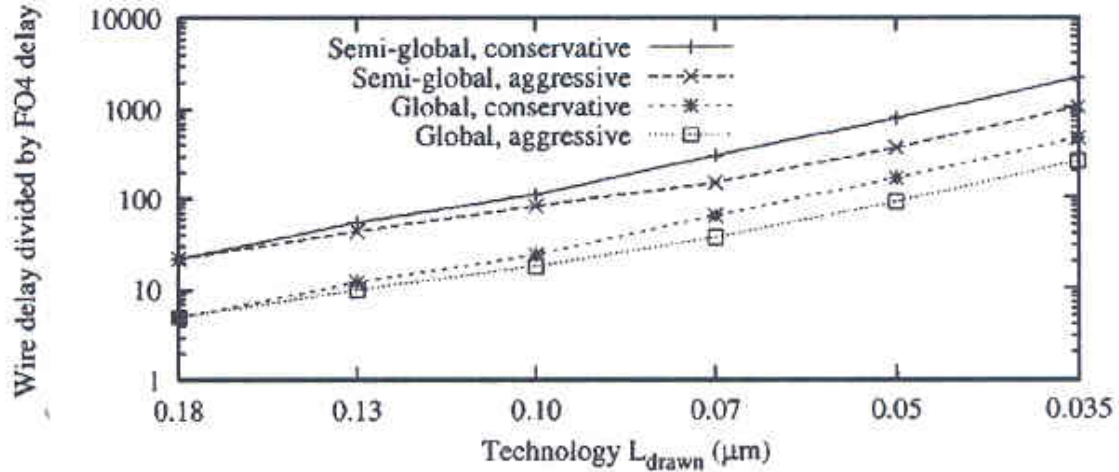


Figure 9: Wire delay in FO4 for fixed-length wires 1 cm long.

#### 1.4 Architectural Implications

Technology scaling studies show that global wires ought to be avoided as much as possible in most architectures since new processes offer new possibilities for designers to pack a large number of gates in a given area of silicon. This exponential increase in the number of gates makes it very difficult for many signals to reach their destination gates in one clock cycle.

As a result, the distance that signals can travel on the wires per clock cycle has been decreasing exponentially for some years. While in the past global communication on global wires was sufficiently cheap, it encouraged architects to focus highly on functionality and less on communication. What ensued is a plethora of function-centric architectures in which the overall architecture is conceived as a monolithic entity without any regard to the costs of global

communication and where the primary objective is to fit the design on the chip. As the complexity of on-chip architectures continues to increase, there seems to be an urgent need to give priority to communication over functionality in architectural considerations. Architects are increasingly interested in breaking architectures into modular sub-architectures in which communication in the basic blocks tend to grow sub-linearly as technology is scaled down. These highly scalable architectures consist usually of identical processing nodes connected by short wires and tailored specifically to a class of applications. One approach advocates the duplication of functional units to consume the growing number of available transistors, thus increasing the explicit degree of parallelism and hence throughput [16]. This approach can be realized by architectures that rely on local communication between low-complexity nodes [17]. Such architectures tend to scale effectively to the problems imposed by the interconnect [16, 18]. Because of the severity of the wiring effects and bandwidth requirements for security applications, these modular architectures are good candidates for addressing the computational requirements of cryptographic applications.

## **1.5 Thesis Contribution**

In this thesis, we propose a new reconfigurable, scalable, two-dimensional architecture, called *CRYPTARRAY*, in which bus-based communication is replaced by distributed shared memory communication. At the physical level, the length of the wires will be kept to a minimum. *CRYPTARRAY* is organized as a chessboard in which the dark and light squares represent Processing Elements (PE) and memory blocks respectively. The granularity and resource

composition of the PEs is specifically designed to support the computing operations encountered in cryptographic algorithms in general, and symmetric algorithms in particular. Communication can occur only between neighboring PEs through locally shared memory blocks (SMBs). Because of the chessboard layout, the architecture can be reconfigured to allow computation to proceed as a pipelined wave in any direction. This organization offers a high computational density in terms of datapath resources and a large number of distributed storage resources that easily support a high degree of parallelism and pipelining.

## **1.6 Thesis Outline**

Chapter 2 reviews previous work related to hardware implementation of cryptographic algorithms and computations while chapter 3 describes the overall architecture of CRYPTARRAY. Chapter 4 describes the architecture and state control of the PE while chapter 5 presents the modeling and implementation of the PEs and SMBs. Chapter 6 presents the conclusion of this thesis.



## **CHAPTER TWO: RELATED WORK**

In this chapter, a brief overview of the various architectures for cryptographic applications is presented. The chapter presents previously proposed systolic and VLSI architectures that support compute-intensive arithmetic operations encountered in cryptographic algorithms. Later, the chapter describes new programmable architectures optimized towards cryptographic operations. These architectures are motivated by the need for a greater flexibility to address the various requirements of cryptographic applications as security standards keep changing. Finally, the chapter concludes by presenting recent attempts at implementing cryptographic algorithms on Field-Programmable Gate Arrays (FPGAs). FPGA technology has matured to the point where high throughputs are easily obtainable in many applications, including cryptography.

### **2.1 Cryptographic Systolic and VLSI Architectures**

Early hardware implementations of cryptography focused on complex arithmetic operations encountered in public cryptography such as modular multiplication involving operands of more than 1024 bits. This multiplication is based on the Montgomery method [19]. In general two distinct approaches were used to support performance with wide-operand multiplication: redundant representation [20-22] and systolic arrays [23-25]. Both approaches are used in conjunction with Montgomery reduction. The implementations based on the first approach suffer from excessive storage area or inadequate performance to complete the multiplication while those based on the second approach deliver good performance although they tend to consume

large logic resources. However due to their high flexibility, systolic approaches based on clever algorithm and architecture design can overcome these difficulties as was previously done in various other applications [26-28]. Other efforts to remedy these limitations were undertaken by either using improved redundant representations or digit-serial architectural approaches [29, 30]. Recently, secret-key algorithms became the subject of various VLSI implementations as well [31-33].

## **2.2 Cryptographic Programmable Processors**

Beside systolic and VLSI implementations, some authors suggested extending the instruction sets of existing processors with special instructions to handle specific operations encountered in cryptography. One of the earliest attempts in this direction proposed a special instruction to support efficient software implementations of general bit permutations [8]. Later in [6], the authors recommended adding instructions to rotate bits left and right, S-box operations, X-box operations, and modular multiplication since they are heavily used in secret-key algorithms. As cryptographic standards and algorithms kept evolving through cryptanalytic studies, other authors emphasized the need for agile architectures in order to offer high flexibility and acceptable performance. Until now, only a handful of attempts pursued this direction. In [14], the authors describe the architecture of a programmable processor that can handle cryptographic algorithms in general. The architecture supports exponentiation by embedding an optimized multiplier with the exponentiation unit. Through careful loop unrolling of the Montgomery algorithm, the processor is able to deliver relatively high encryption rates even though the

architecture was synthesized in 2- $\mu\text{m}$  CMOS technology considered somewhat outdated. In [12], the authors propose an energy-efficient reconfigurable processor for public-key cryptography. The processor architecture is designed to support only the subset functions required for asymmetric cryptography [34]. As a result, the processor's instruction set contains operations related to conventional arithmetic, modular integer arithmetic,  $\text{GF}(2^n)$  arithmetic, and elliptic curve field arithmetic over  $\text{GF}(2^n)$ . The processor is relatively energy efficient when compared to software and FPGA implementations of typical operations. Another proposal for a programmable processor has been described in [13]. The processor architecture targets the primary bottlenecks in private cryptography by matching the instruction set and functional resources to support the compute intensive operations in secret-key algorithms. The processor is a four-issue VLIW processor consisting of four pipeline stages: Instruction Decode (ID), Instruction Decode/Register Fetch (ID/RF), Execute/Memory Access (EX/MEM), and Write-Back (WB). The EX/MEM stage contains four functional units where each unit consists of two logical units, a 32-bit pipelined multiplier, a 1KB cache for S-box operations, a 32-bit adder, and a 32-bit rotator. Although the architecture was automatically synthesized, it delivers a performance that is 32% to 290% better than that of a 600 MHz Alpha processor for the Blowfish, 3DES, MARS, and Rijndael kernels. While the programmable processor approach is highly agile, it still falls short of the potential performance gains that can be achieved if the architecture adopts a low-latency communication scheme between medium granularity functional units, instead of the costly global communication approach used in monolithic processors.

### **2.3 Cryptographic FPGA Designs**

Using FPGA technology, several implementations have been proposed whereby multi gigabits per second performances were obtained [35-37]. The flexibility provided by FPGA implementations can be quite attractive since most cryptographic algorithms are still evolving. In addition, the capacity of many FPGA chips has reached a level that is suitable to support the mapping of the numerous rounds present in most cryptographic algorithms. In [35], 11 rounds of the AES selected Rijndael algorithm are unrolled and pipelined onto a high-capacity Virtex FPGA in such a way that a new 128-bit data-key pair can be input at every clock cycle. The result of this pipelined approach is a design that can run at 139.1 MHz with a throughput of 17.8 Gbps. While most FPGA chips are general-purpose reprogrammable devices, they are mostly used for specific application domains. The analysis of cryptographic applications reveals that these applications are usually dominated by varying width arithmetic operations and bit computations. Arithmetic operations can benefit from the use of coarse-grain reconfigurable components instead of the Look-Up Tables (LUT) used in FPGAs. If reconfiguration bit quantity is used to measure LUT complexity, it becomes clear that when LUTs are used for arithmetic operations, this complexity increases with operand width [38]. As for bit computations such S-box operations, they can be supported efficiently through tables or memories. Although most recent FPGAs contain memory blocks that can be used for bit operations, they can be located quite apart from where arithmetic operations are occurring in the chip depending on the mapping and placement. Transferring data between these memory blocks and arithmetic operations can be detrimental to performance, considering the penalty associated with FPGA interconnects.

## **2.4 Summary**

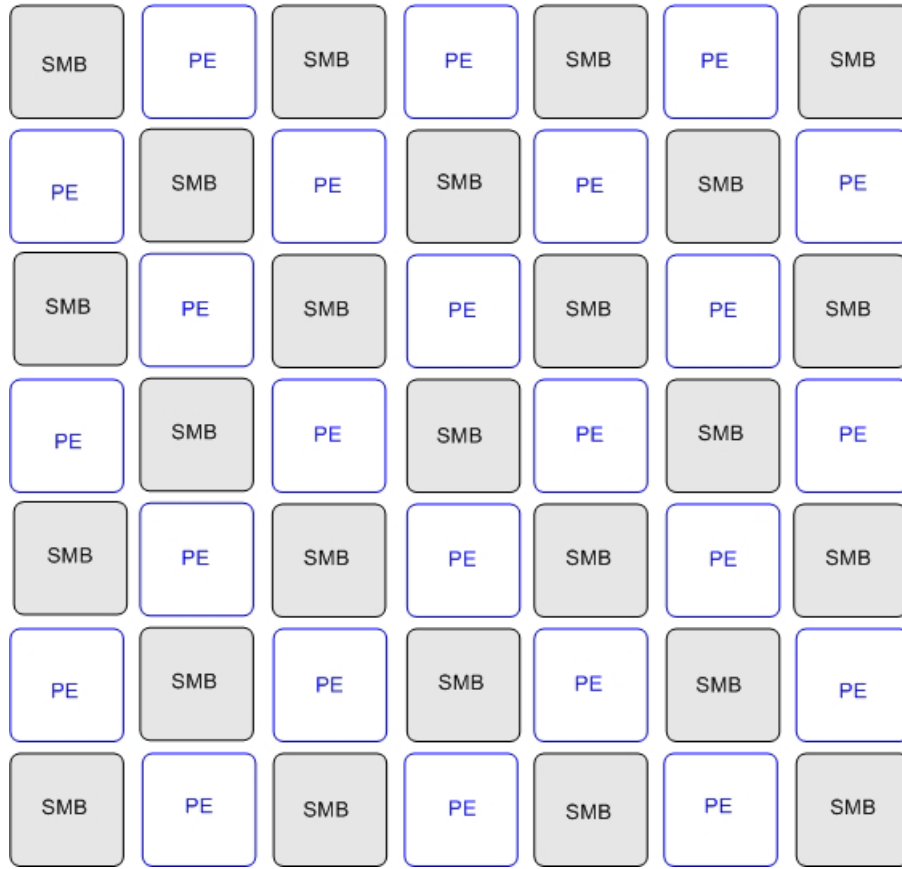
Given the urgency needed to address the interconnect problem, modular scalable reconfigurable architectures are good candidates to address the computational requirements of cryptographic applications. As opposed to the three hardware approaches described above, reconfigurable architectures can provide the following significant advantages: (i) the bit-width of the operations can be tailored to a given computation, (ii) multiple PEs can operate in parallel to take advantage of data dependencies inherent to the application, (iii) PEs can be pipelined through reconfiguration to increase the application throughput, (iv) PEs can be reconfigured in groups to support complex operations if need be, (v) input and output values are recycled several times within a computation, thus avoiding slow and repetitive accesses to monolithic RAMs associated with general purpose processors [39]. All these advantages can be readily realized in CRYPTARRAY.

## CHAPTER THREE: CRYPTARRAY

In this chapter, section 3.1 presents the overall organization of CRYPTARRAY while section 3.2 explains the organization of the shared memory blocks. In addition, section 3.3 presents an overview of the functionality of the processing element in CRYPTARRAY while section 3.4 describes the format and hierarchical encodings of the instructions used to program the array. Finally, section 3.5 presents the instruction-dispatching based reconfiguration mechanism and its two modes for CRYPTARRAY.

### **3.1 Layout of CRYPTARRAY**

CRYPTARRAY is a two-dimensional array of tiles organized in a checkerboard-type pattern. Each tile can be either : (i) a datapath tile containing a single processing element (PE), or (ii) a storage tile containing a shared memory block (SMB). Tiles are connected on their perimeter by direct short wires, and can subsequently communicate only with their immediate neighbors. Figure 10 shows the layout of an array architecture using 24 PEs and 25 SMBs. The tiles in the array can be reconfigured by dispatching wide instructions to the PEs. This mechanism of programming the array can lead to a high degree of parallelism and pipelining.



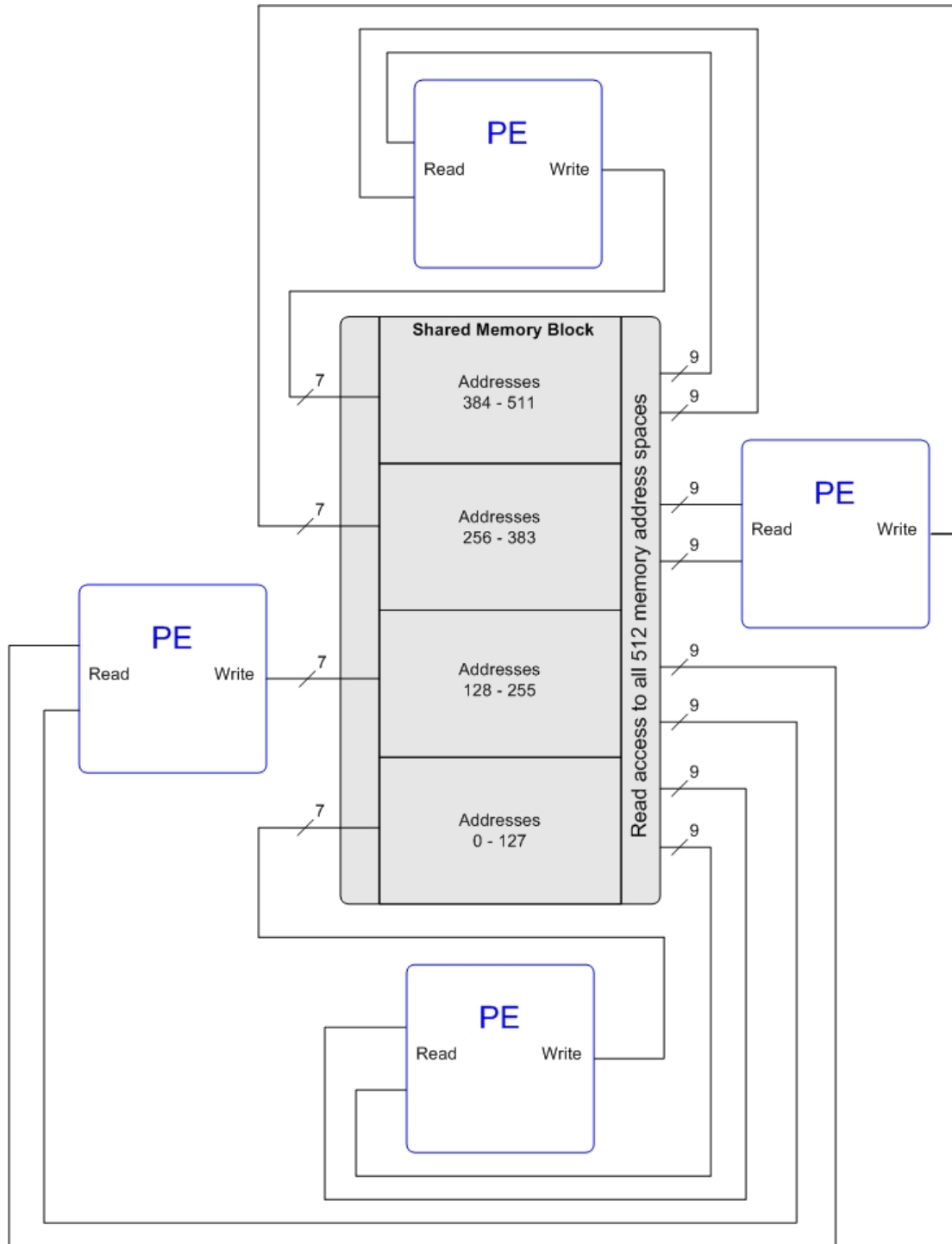
**Figure 10: Checkerboard layout of CRYPTARRAY.**

### **3.2 Shared Memory Blocks**

A storage tile consists of a 512 x 4-bit multi-port memory block. This memory stores the operands and results of arithmetic operations and can be used for substitution operations. These substitutions use table lookups to support any key-parameterized function such as S-BOX operations, which are common in cryptographic algorithms. S-BOX operations consist mainly of searching entries in 512 x 32-bit tables.

An SMB is shared between its four surrounding PEs as it can be accessed for reading and writing by any adjacent PE. It is connected to each single PE using three four-bit data lines where the two first data lines are used for reading the operands while the remaining data line is used for writing the resulting data. In addition, the SMB has three address lines where each address line serves a single data line. Since there are four PEs connected to each SMB, a total of eight read and four write ports are available for each SMB. To avoid write conflicts, each of the four surrounding PEs can write to only a fourth of the 512 available memory addresses, or 128 possible locations of an SMB. This introduces some asymmetry in the addressing busses by making them nine and seven-bit wide for reading and writing respectively. Figure 11 shows the connectivity of an SMB to its four surrounding PEs. As shown in the figure, the PE located to the bottom of the SMB can write to the first set of 128 addresses (0-127) while the PE located to the left of the SMB can write to the second set of 128 addresses (128-255). In addition, the PE located to the right of the SMB can write to the third set of 128 addresses (256-383) while the PE located on the top of the SMB can write to the final set of 128 addresses (384-511). For reading, all four PEs can access the entire 512MB of memory space in the SMB. For performance purposes, it was decided that this configuration is more efficient since it provides a realistic number of read/write memory ports and hence is low in area cost. The consequence of this configuration, when compared with the next best alternative, is that two additional clock cycles are needed to complete the input and output of data for all five of the operation blocks within the PE.





**Figure 11: Connectivity of the SMB to its four surrounding PEs.**

For each of the PEs, there are 11 data inputs and 5 data outputs required. Table 2 displays the possible configurations that were considered for selecting a final read/write SMB-PE interface configuration, where the leftmost column is simply an alphabetic label assigned to the particular setup allowing for identification of that configuration in further discussion.

**Table 2: Possible configurations of read/write ports between an SMB and its four surrounding PEs.**

Configuration	Memory Ports			Time (cycles)	Cycle Access	
	Reads (per PE)	Writes (per PE)	Total		Blocks (per PE)	PEs
A	11	5	44 reads + 20 writes = 64	1	All	All
B	6	3	24 reads + 12 writes = 36	2	3 reads, 3 writes	All
C	4	2	16 reads + 8 writes = 24	3	2 reads, 2 writes	All
D	11	5	11 reads + 5 writes = 16	4	All	1
E	2	1	8 reads + 4 writes = 12	6	1 read, 1 write	All

The next three columns in the table refer to the read ports, write ports, and total number of ports respectively. The fifth column represents the number of clock cycles that are required for all of the inputs and outputs of the four neighboring PEs to access an SMB. The sixth column shows the number of operation blocks in each PE that can access an SMB per clock cycle while the last column indicates how many of the neighboring PEs can access the SMB in each clock cycle. Table 3 shows the advantages and disadvantages of each SMB-to-PE interface configuration.

**Table 3: Comparison of the five SMB-PE interface configurations.**

Configuration	Comments, Advantages, and disadvantages	
A	Comment	All the I/O ports of each PE are able to access the SMB in each cycle.
	Advantage	With one required cycle for all five PE blocks and all surrounding PEs, it is obvious that this is the fastest configuration.
	Disadvantage	Too many read/write ports to efficiently implement in an SMB.
B	Comment	Half of the read and write ports of all surrounding PEs access the SMB simultaneously.
	Advantage	With two cycles for all five PE blocks and all surrounding PEs, this is the second fastest configuration.
	Disadvantage	The required 24 read and 12 write ports will produce an inefficient implementation.
C	Comment	A third of the read and write ports of all surrounding PEs can access the SMB simultaneously.
	Advantage	With three cycles for all five PE blocks in the four surrounding PEs, it is the third fastest configuration.
	Disadvantage	The implementation can be costly since this configuration requires 16 read and 8 write ports.
D	Comment	All I/O ports of a single PE amongst the surrounding PEs can access the SMB.
	Advantage	This configuration has a significantly reduced number of read/write ports in contrast to the previous configurations.
	Disadvantage	This configuration requires a complex control to determine which PE is ready to write to the SMB for any given cycle.
E	Comment	One PE block writes while another reads in each of the four surrounding PEs.
	Advantage	An eight-read four-write port memory is not a costly implementation.
	Disadvantage	This is the slowest configuration since it requires six cycles to complete.

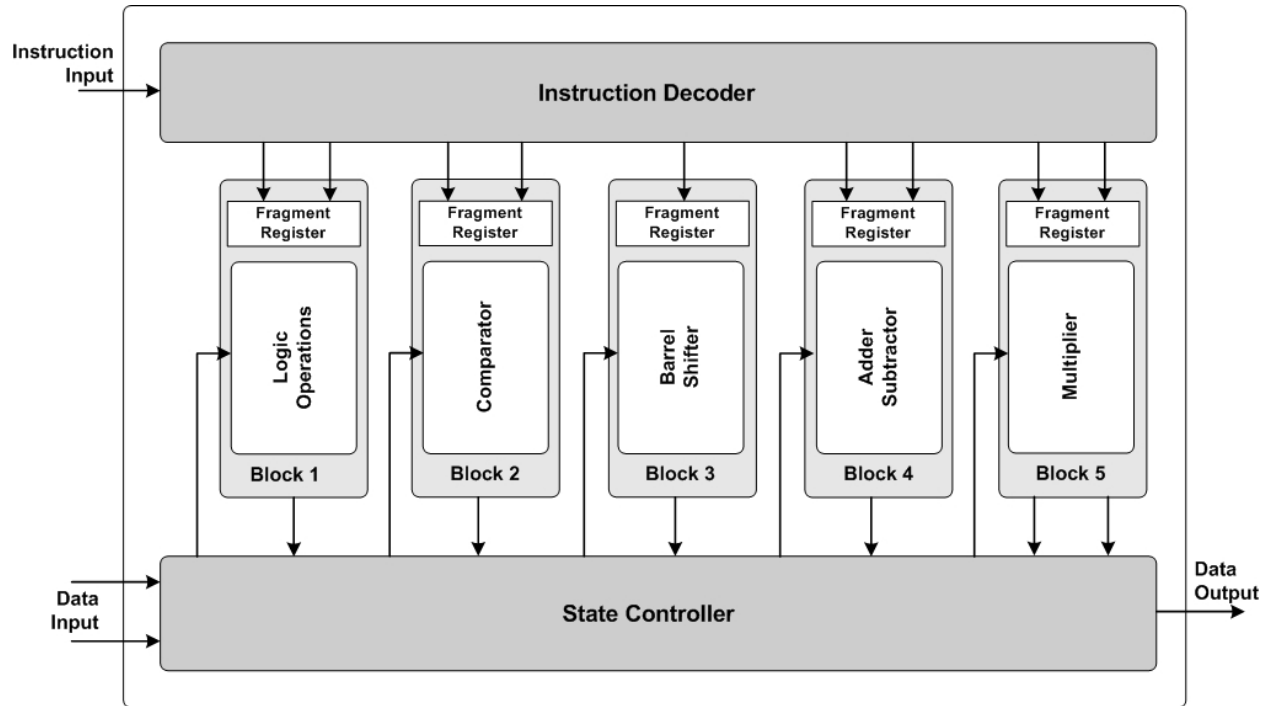
Although configuration E has the slowest access as shown in Tables 2 and 3, it was chosen for this design since its implementation of the SMB requires a reasonable number of read and write ports. For configuration D, which is the next best option, significant control complexity could arise with regard to determining which of the four PEs that surround a particular SMB would be

allowed to access it next. This is further complicated by the necessity for each PE to be able to fully access any of its four surrounding SMBs.

### **3.3 PE Organization**

Depending on its configuration, a PE can read from and write to any one of its four neighboring SMBs. The choice of datapath resources in the PE is heavily based on the primary operations required in cryptographic applications. These include modular addition, modular multiplication, substitutions such as SBOX operations, and general permutations [6]. As a result, the following arithmetic and logic operations are supported by the PE: (i) addition, (ii) subtraction, (iii) multiplication, (iv) rotation, (v) comparison, and (vi) logical operations. These operations are stored in the configuration of the PE and are sent as static instructions. As shown in Figure 12, the structure of a PE consists of five blocks for arithmetic/logic operations supported by additional logic for controlling the read/write memory access.

The operation blocks are organized as follows: (i) the first block supports four-bit logic operations, (ii) the second block supports four-bit comparisons, (iii) the third block supports four-bit shift rotations, (iv) the fourth block supports four-bit addition and subtraction, and (v) the fifth block supports four-bit multiplication. The control logic consists of two blocks, (i) an instruction decoder, and (ii) a state machine that enables and disables the operating blocks. Each block can independently access a neighboring SMB to retrieve its operands and write its result.



**Figure 12: Structural Organization of the PE.**

Consequently, while all four neighboring PEs can access an SMB simultaneously, only one of the five blocks within each PE can access the SMB per cycle. These five blocks cyclically rotate in turn for memory access. If block one is the first to receive input data, block two will be next, followed by block three, and so forth. The logic operations of the first block are AND, OR, NAND, NOR, XOR, XNOR and NOT. The comparator of the second block can perform comparisons of two numbers, which can be used to support branching and looping control operations. The barrel shifter in the third block can rotate by one, two, or three positions to the left or to the right depending on its configuration. The adder in the fourth block can be configured to perform addition or subtraction. Each of the five blocks in the PE are enabled and

configured by the bit contents of an instruction fragment register. Each fragment register can contain between one and four bits to control the functional unit in that particular block, 11 bits to address the first source operand, 11 bits to address the second source operand, and nine bits to address the destination operand. The number of bits needed to configure a single PE block ranges from 24 to 48 thus providing 173 bits as the total number of bits in the five fragment registers within each individual PE.

### 3.4 PE Instructions

Each PE block is configured by the contents of its fragment register, shown in Figure 12. These contents make up a specific instruction tailored to that block. In all, there are three distinct formats of block instructions as shown in Figure 13.

49	44	43		33	32		22	21		11	10		0		
6 bits for operation		11 address bits				11 address bits				11 address bits				11 address bits	
		Operands for Comparison								Alternate Operands					

**(a) Instruction format for block 2.**

49	44	43	42	41								20	19					9	8					0
6 bits for operation		2 bits for data as address		22 unused bits										11 address bits				9 address bits						
														Input Operand				Output data						

**(b) Instruction format for block 3.**

49	44	43	42	41	40	39					31	30					20	19					9	8					0
6 bits for operation		2 bits for data as address		location of carry-in (2 bits)		9 unused bits			11 address bits				11 address bits				9 address bits												
Input Operands									Output data																				

**(c) Instruction format for block 1, 4, and 5.**

**Figure 13: Formats of the block instructions in a PE.**

Figure 13(a) shows the instruction used in a comparison operation of block 2 where bits 0 through 10 and 11 through 21 represent the memory addresses of both operands that will be used in another operation block if a comparison is found to be true. For example in the following comparison:

```
if A > B then  
    C + D;  
endif
```

If A is greater than B, an add operation is performed on operands C and D. Bits 0 through 21 in block 2 instruction format represent the addresses of operands C and D while the addresses of operands A and B are located in bits 22 through 43.

Figure 13(b) shows the instruction format used in block 3 for shifting operations. In this format, bit 0 through 8 represent the memory addresses used to store the resulting output data from the barrel shifter block while bits 9 through 19 represent the address of the input operand to be shifted. However, bits 20 through 41 are unused for this operation. The two bits 43 and 42 are used in some cases where the output data is to be used as an input address. A '00' in bits 43 and 42 indicates that these two bits are ignored while a '01', '10', and '11' indicate that the output data of blocks 1, 3, and 4 respectively, are used as the lowest four bits of the input operand's address. For example, if these two bits are '10' and the output of block 3 is '0110', the seven most significant bits (bit 10 through 4) of the 11-bit address of the input operand would remain unchanged while the four least significant bits (bits 3 through 0) will be set to '0110'.

Figure 13(c) shows the instruction format used in the logic operations of block 1, the addition and subtraction of block 4, and the multiplication of block 5. In this format, bits 0 through 8 represent the memory address of the output operand while bits 9 through 30 represent the addresses of the two input operands. Since corresponding blocks in neighboring PEs can be linked together through carry chains to handle wide operand operations, bits 40 and 41 are used to determine which of the neighboring PEs feed the carry bit to block 4 for addition operations on wide operands. Bits 42 and 43 are used for data-to-address functions as described in the preceding paragraph.

In each of the instructions shown in Figure 13, the six most-significant bits (44 through 49) are used to indicate the specific operation that needs to be executed. The encoding of these operations is shown in Table 4 in which the leftmost column represents the block within a PE while the operation labeled column represents the operations performed in the corresponding block.

**Table 4: Block-level and PE-level encoding of operations.**

<b>Block</b>	<b>Operation</b>	<b>Block-Level Encoding</b>	<b>PE-Level Encoding</b>
1	Disabled	0XXX	000001
	Disabled	1000	-----
	AND	1001	100001
	NAND	1010	100010
	OR	1011	100011
	NOR	1100	100100
	XOR	1101	100101
	XNOR	1110	100110
	NOT	1111	100111



2	Disabled	00XX	000010
	If A > B then send C and D to block 1	0100	101000
	then send C and D to block 3	0101	101001
	then send C and D to block 4	0110	101010
	then send C and D to block 5	0111	101011
	If A < B then send C and D to block 1	1000	101100
	then send C and D to block 3	1001	101101
	then send C and D to block 4	1010	101110
	then send C and D to block 5	1011	101111
	If A = B then send C and D to block 1	1100	110000
	then send C and D to block 3	1101	110001
	then send C and D to block 4	1110	110010
	then send C and D to block 5	1111	110011
3	Disabled	0XXX	000011
	Disabled	1000	-----
	1-bit Right Rotate	1001	110100
	2-bit Right Rotate	1010	110101
	3-bit right Rotate	1011	110110
	Pass through	1100	110111
	1-bit Left Rotate	1101	111000
	2-bit Left Rotate	1110	111001
	3-bit Left Rotate	1111	111010
4	Disabled	0XX	000100
	4-bit Add	100	111011
	Extended Width Add	101	111100
	4-bit Subtract	110	111101
	Extended Width Subtract	111	111110
5	Disabled	0	000101
	Multiply	1	111111
PE	Disabled	0	000000
	Enabled	1	1XXXXX

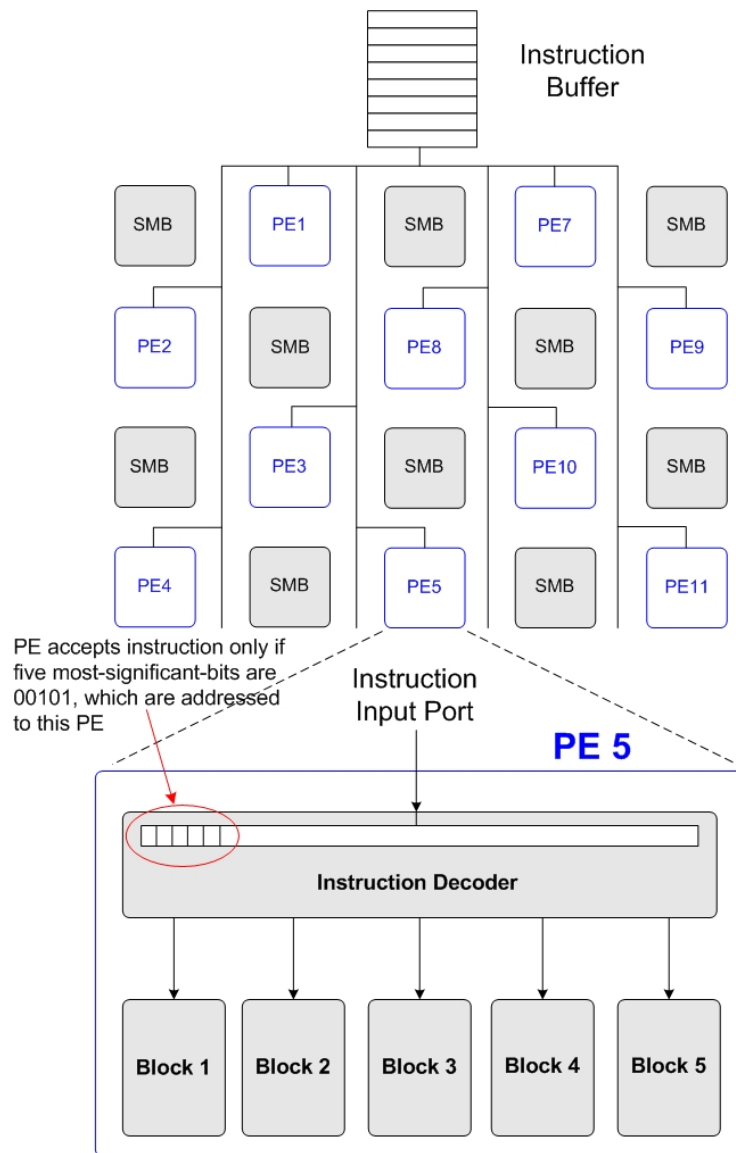
For block 1, the logic operations of AND, NAND, OR, NOR, XOR, XNOR, and NOT can be performed using the encoding shown in the table. In the case of block 2, the operation column indicates which comparison is to be performed, and which operation is performed on the alternate operands if the comparison is true. Alternate operands are used in the instruction format

of block 2 as shown in Figure 13(a). In row 2, 3, 4, and 5 of block 2, if operand A is greater than B, both operands C and D can be sent to block 1, 3, 4, or 5 depending on the instruction following the If statement. After the C and D operands are sent to a given block, they are used as input operands for the operation of that block. The rows of block 3 show the encoding of the 1, 2, and 3-left or right shift instructions, while the rows of block 4 show the encoding of the addition or subtraction using either 4-bits or wider through the carry-bits. Finally, the rows of block 5 show the encoding used for multiplication. The column labeled Block-level encoding shows the bit encoding of each operation as it is sent to the fragment register of its block while the PE-level encoding labeled column shows the bit encoding of the operation as it is seen in the array before passing through the Instruction Decoder module. This hierarchical encoding of the instruction is explained in section 3.5.

### **3.5 PE Reconfiguration**

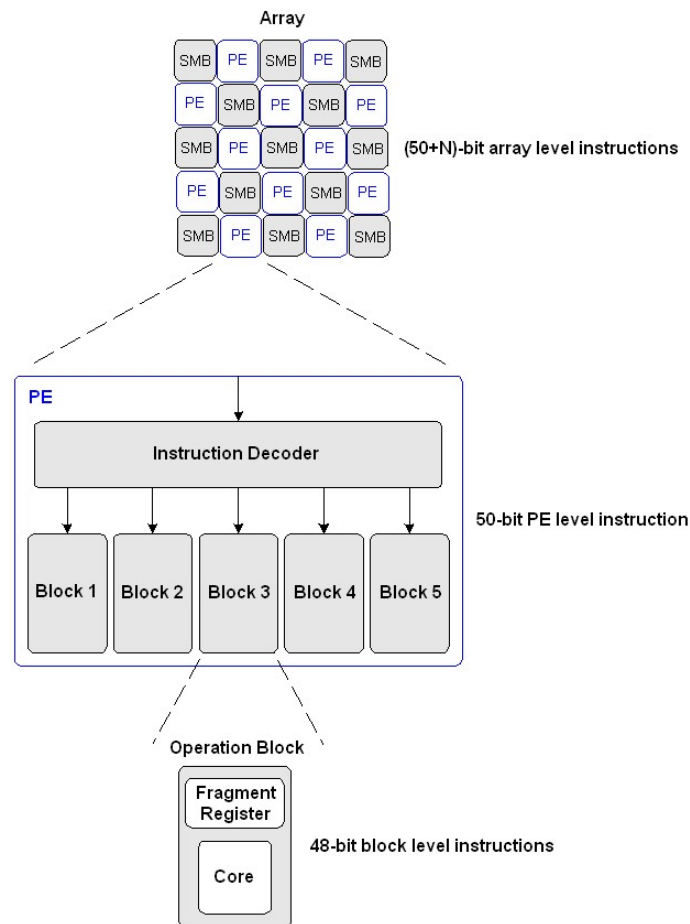
The reconfigurability of the array is based on dispatching instructions which are stored in the fragment registers of a PE. To dispatch an instruction to a specific PE, a mechanism is needed to address that particular PE. Several architectures are possible to support instruction dispatching the simplest of which is to hardwire the address of each PE. In that case, dispatching consists of sending instructions on a clock-like bus distribution scheme as shown in Figure 14 that reaches each PE in the array. All of the PEs will see each instruction, but only if the address indicates that it is intended for that particular PE will it capture the instruction, then store it in one of the PE's fragment registers. To accomplish this, an additional  $N$  bits are added to the most-

significant end of the instruction. Figure 15 shows  $(50+N)$  address bits are used to encode an instruction at array level. These address bits will be used for allowing the runtime environment through the instruction buffer to intentionally target the instruction to a specific PE in the array.



**Figure 14: Instruction dispatch and capture by a PE.**

After capture by the PE, the instruction is stripped of its  $N$  bits leaving 50 bits for decoding. Figure 15 shows the 50 bits making up an instruction after being captured by a PE. This 50-bit instruction is decoded based on the six operation bits in the Instruction Decoder module, and distributed to the appropriate fragment register of one of the five operation blocks by stripping two more bits from it leaving only 48 bits in the instruction. Figure 15 shows the 48-bits that make up an instruction after reaching a specific block in the target PE.



**Figure 15: Hierarchical encoding of the instructions.**

This bus-based dispatching scheme allows the dispatch of a single instruction per cycle. Although this scheme is quite simplistic, it allows the array to operate in two distinct reconfiguration modes:

- (i) *Static reconfiguration*: In this mode, the instructions of all the blocks in all the PEs are loaded ahead of time to reconfigure the entire array before it starts running. No other instruction can be sent to the array during run time. In this mode, each block reads operands from an SMB, executes an operation, and writes the result to an SMB in three clock cycles.
- (ii) *Dynamic reconfiguration*: In this mode, a single instruction is dispatched at the start of a clock cycle. Blocks can be reprogrammed as the array is running. In this mode, a block receives an instruction that gets stored in its fragment register, reads operands from an SMB, executes an operation, and writes the result to an SMB in four clock cycles. This mode support also fine-grain partial reconfiguration.

## CHAPTER FOUR: PE ARCHITECTURE

Since the PE is the primary computation component in CRYPTARRAY, its architecture and its implementation are described in detail in this chapter. Section 4.1 presents the architecture of the PE at the highest level of abstraction while Section 4.2 through 4.6 presents the architecture of each block within the PE. Section 4.7 describes the states and their control within a PE.

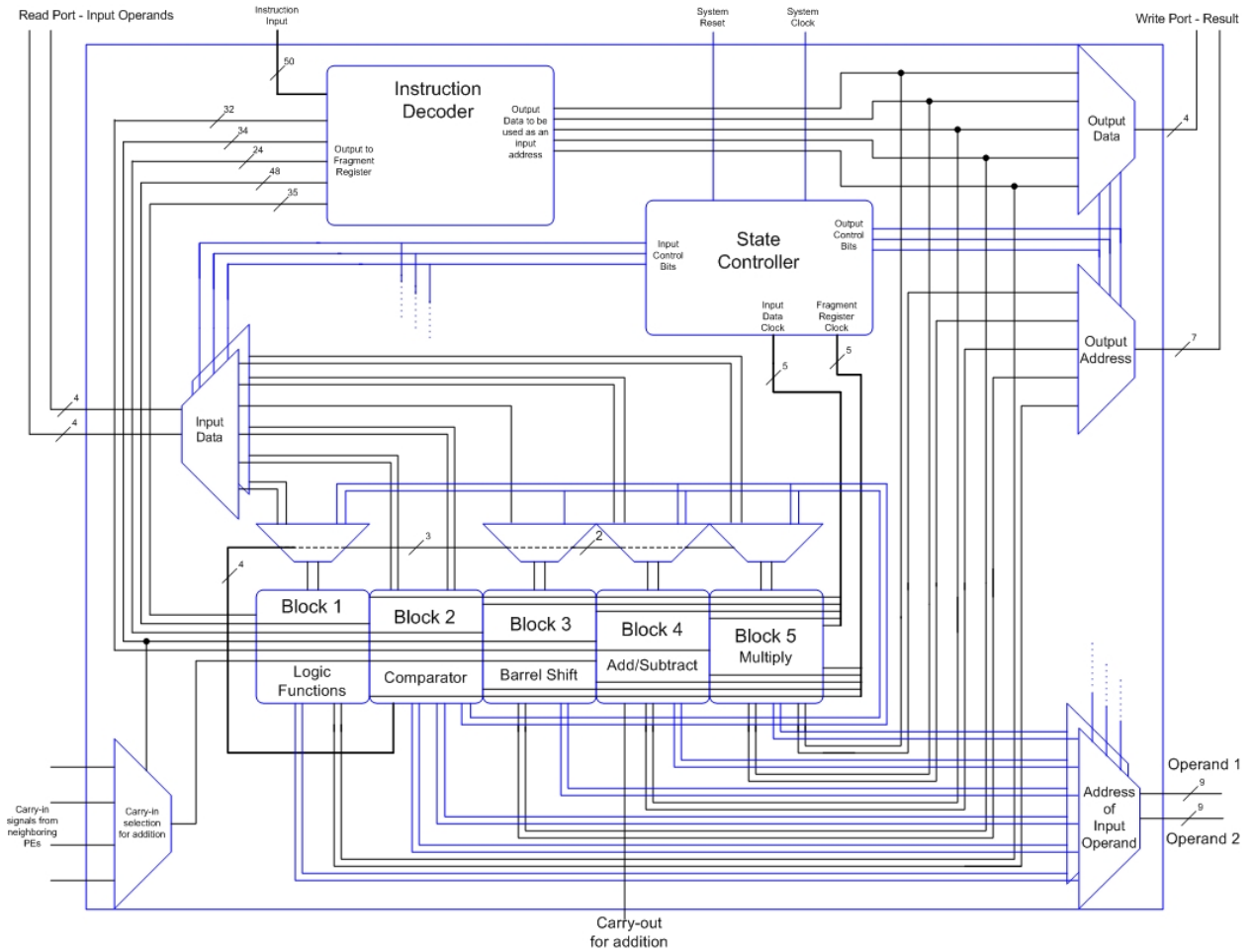
### 4.1 Architectural Components of the PE

Figure 16 shows the primary components within a single PE at the highest level of organizational hierarchy. A PE consists of:

- *Instruction Decoder*: This module decodes the incoming instruction, reformats it, and routes it to the proper block of the PE for execution. In addition, it can set the output data as input addresses for some blocks.
- *State Controller*: This controller is a state machine that determines which of the five operation blocks will be sending and receiving data in any given cycle by controlling all the multiplexers and demultiplexers, with the exception of the multiplexer used for carry-in selection. The demultiplexer labeled Input Data determines which of the five operation blocks will receive the incoming data from the memory. Accordingly, the two multiplexers labeled Address of Input Operands specify to the memory which address location to retrieve the input data from. The multiplexers labeled Output Data and Output Address are used to select which operation block will send its output to memory. The actual implementation

requires four of each of the multiplexers and demultiplexers in order to allow data transfer to each of the four neighboring SMBs. Each operation block controls individually from/to which of the four SMBs accept or send data based on the addresses in their corresponding fragment registers.

- *Operation Blocks*: There are five of these blocks in each PE where each block consists of three hierarchical modules:
  - *Memory Interface*: Each individual block communicates through an interface of read and write ports to its four surrounding SMBs. Each address in the instruction specifies to/from which SMB (top, right, bottom, left) to send and receive data. Because it must interface with the memories which are exterior to the PE, this is the outmost layer of the operation block.
  - *Fragment Register*: This register stores the incoming instruction, which controls the operation of the block and is sent from the Instruction Decoder module.
  - *Core*: The core of a block represents the datapath used to execute the operations of the block. For example, the core of block 1 contains the gates to support the logic operations (AND, NAND, OR, NOR, etc...) while the core of block 2 contains the comparator used to determine the outcome of branching and loop operations. This module is the innermost layer of each operation block.



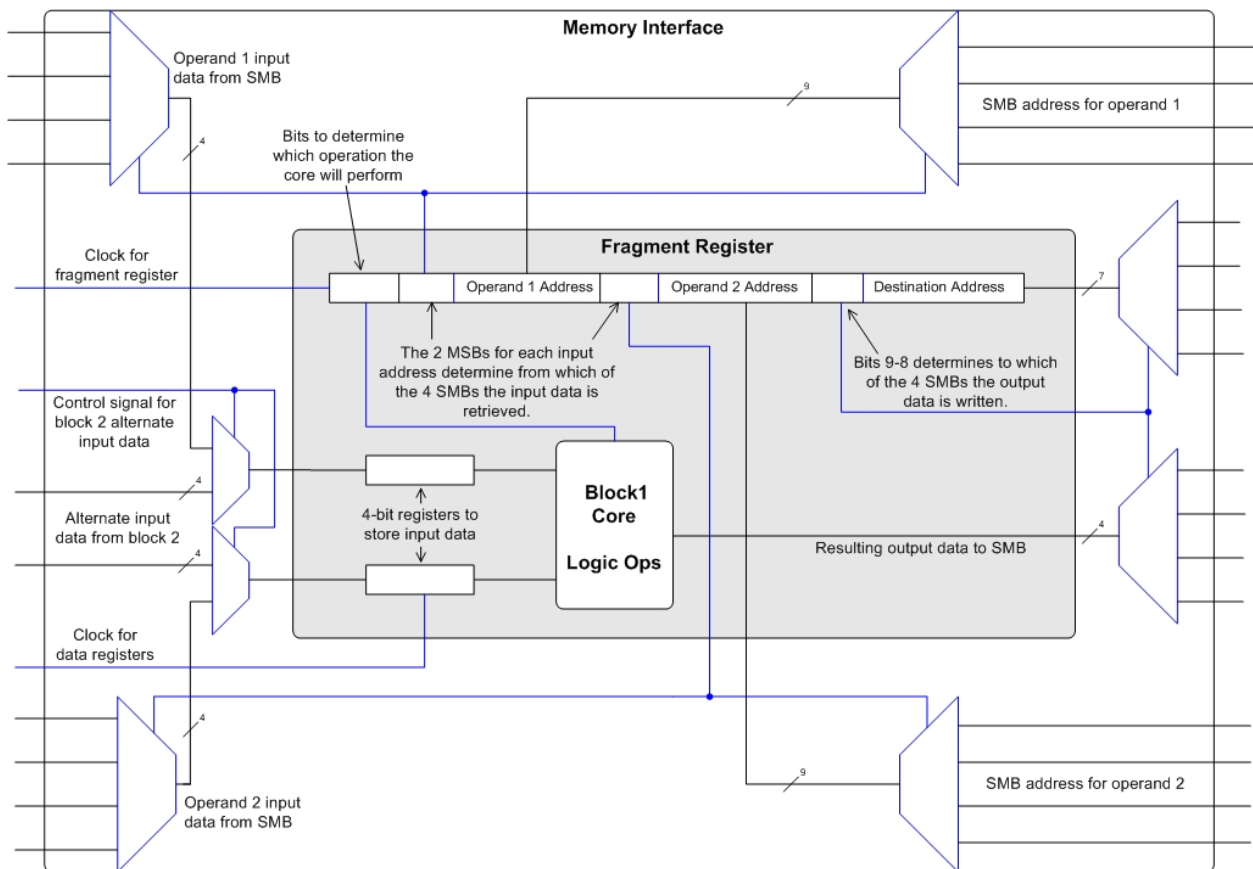
**Figure 16: Architectural components of a single PE.**

## **4.2 Architecture of Block 1**

Figure 17 shows the architecture of block 1 where each 4-to-1 multiplexer and 1-to-4 demultiplexer is used to determine which of the four neighboring SMBs will be accessed for reading or writing respectively. Three of the demultiplexers are used for addressing an SMB while the fourth is used for writing the output data. The two multiplexers on the left side of the



figure are used to read in the 4-bit operand data from the neighboring SMBs. In addition, the smaller 2-to-1 multiplexers on the left side of the figure are controlled by block 2 where they are used to read in alternate data if (i) the comparison in block 2 is found to be true, and (ii) a logic operation is to be performed on the alternate data. These multiplexers are also shown at the bottom of Figure 18.



**Figure 17: Architecture of block 1.**

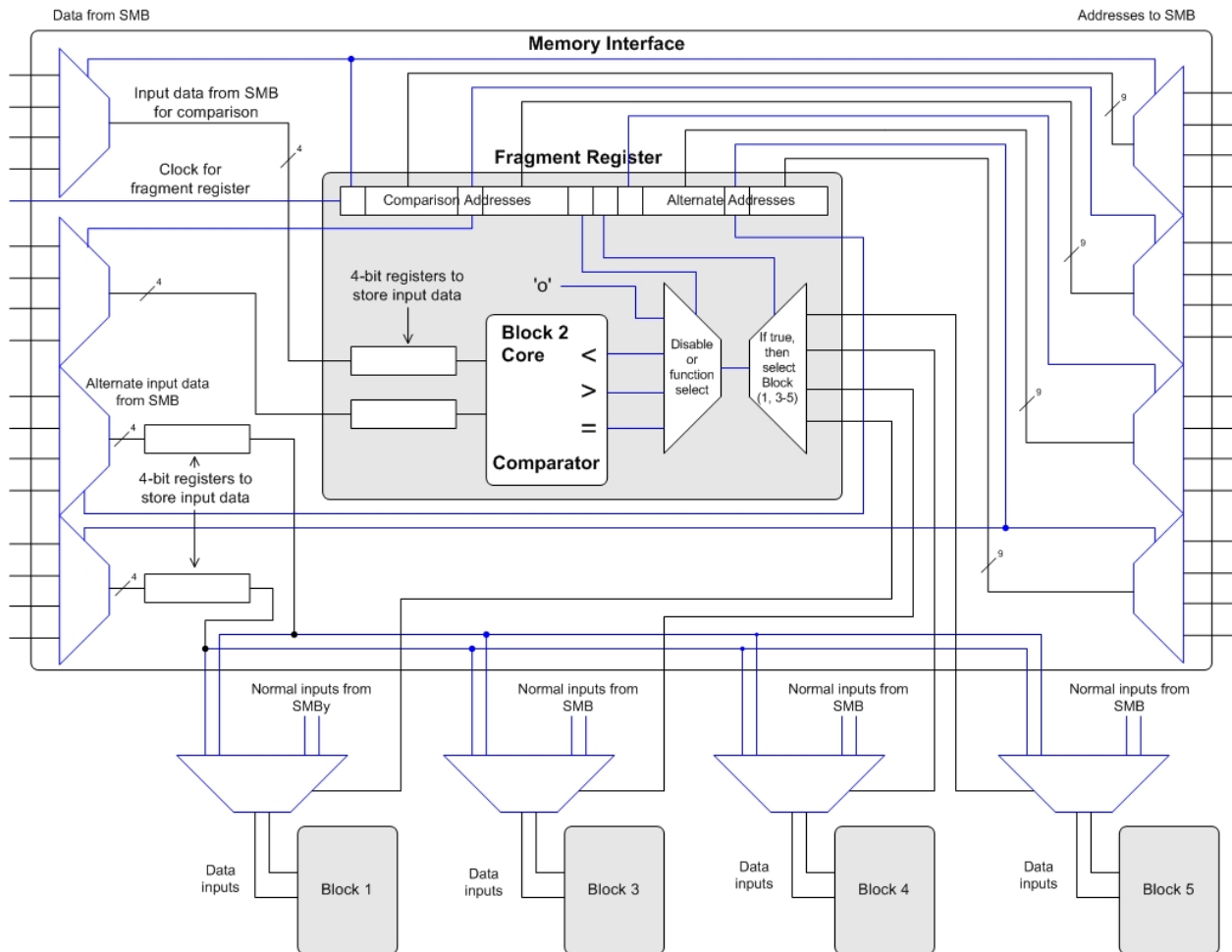
### **4.3 Architecture of Block 2**

Figure 18 shows the architecture of block 2. This block has four data inputs and no data outputs to the SMBs. Two of these inputs are used to compare two operands while the other two are used to send alternate inputs should the comparison be true. The four 4-to-1 multiplexers shown on the left side of the figure are used to read in operands while the four 1-to-4 demultiplexers shown on the right side of the figure are used to address the SMBs. Immediately after the core, the multiplexer closer to the comparator's core determines which comparison operation to perform (less than, greater than, or equal to). If the control lines are '00', which are the two most significant bits of the operation field in the instruction, the output of this multiplexer is always 0 thus disabling the block altogether. The demultiplexer located on the right side of the output side of the comparator determines which of the other four operation blocks (block 1, 3, 4, and 5) will read in the alternate data if the performed comparison by the comparator is found to be true. For example, consider the following comparison:

```
if A > B then  
    C + D;  
endif
```

The control line for the left multiplexer located immediately after the comparator's output will be '10' while the control line for the demultiplexer to its right will be '00'. If the result of the comparison is true, a '1' will be output from the multiplexer, routed by the demultiplexer to the control signal of the 2-to-1 multiplexer which controls block 1. This multiplexer, shown at the bottom of the figure in front of block 1 (See also Figure 17) requires that the alternate input

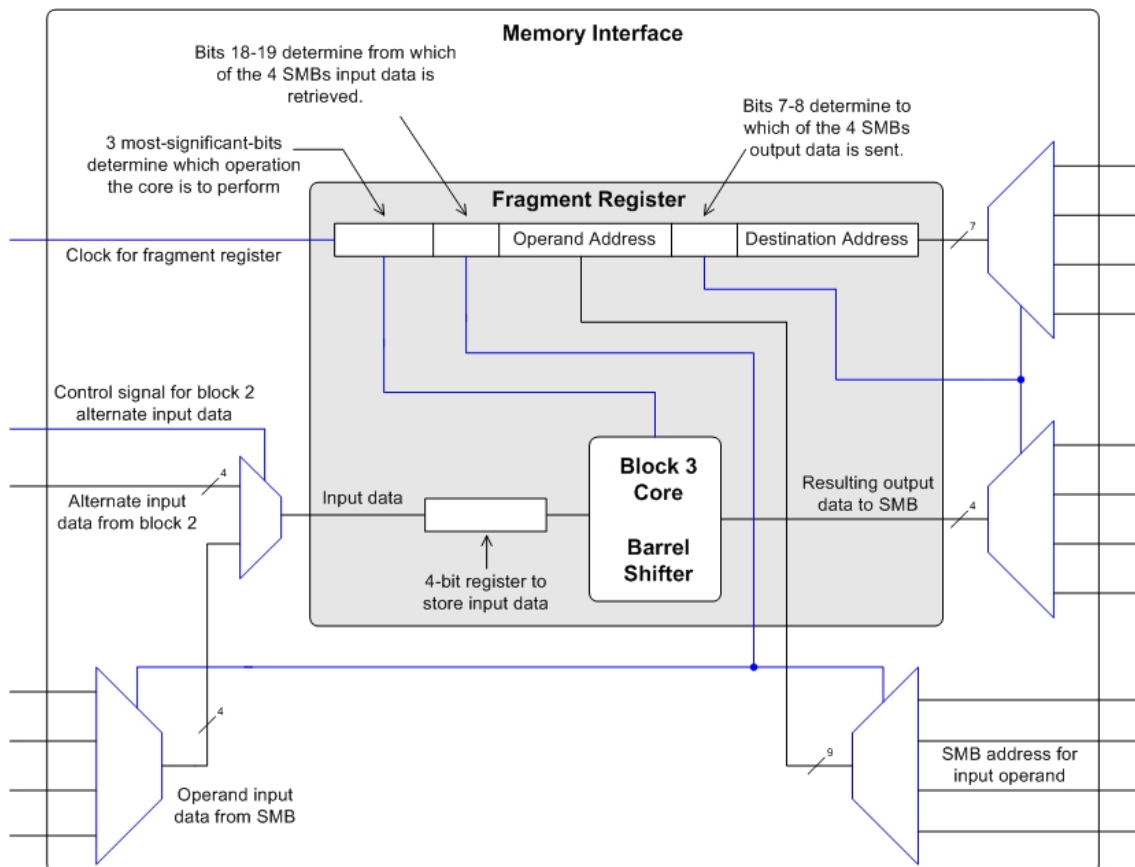
operands C and D be fed to block 1 instead of the original input operands specified in the fragment register of block 1.



**Figure 18: Architecture of block 2.**

#### 4.4 Architecture of Block 3

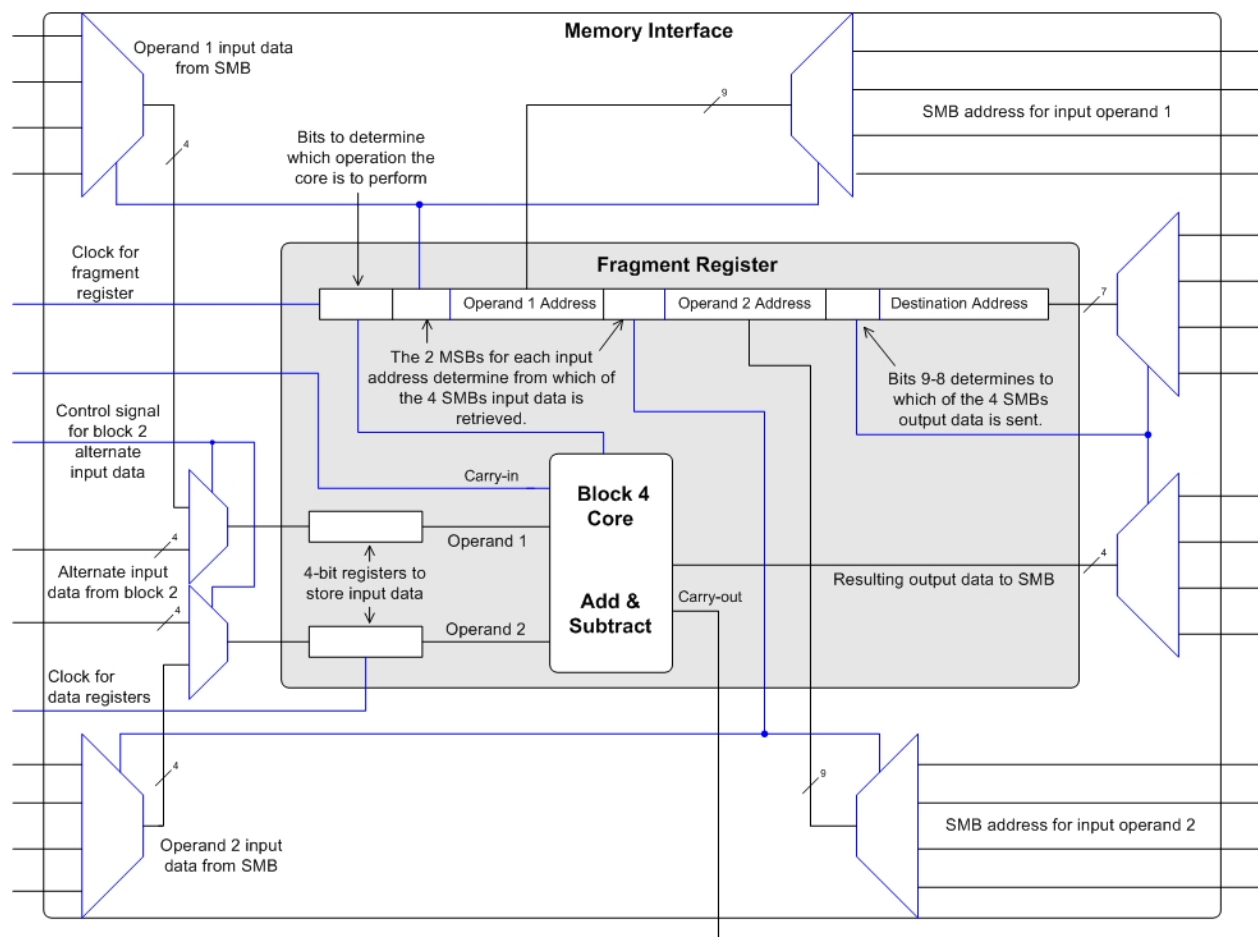
Figure 19 shows the architecture of block 3. Since block 3 reads only a single operand as shown in the figure, only one 4-to-1 multiplexer shown on the left side of the figure is used to read in data from the SMBs. On the right side of the figure, two of the three 4-to-1 demultiplexers are used to output data and address while the third demultiplexer is used to read in data. This demultiplexer is controlled by block 2 and is used to read alternate data if (i) the comparison in block 2 is true, and (ii) a barrel shift operation is to be performed on the alternate data.



**Figure 19: Architecture of block 3.**

## 4.5 Architecture of Block 4

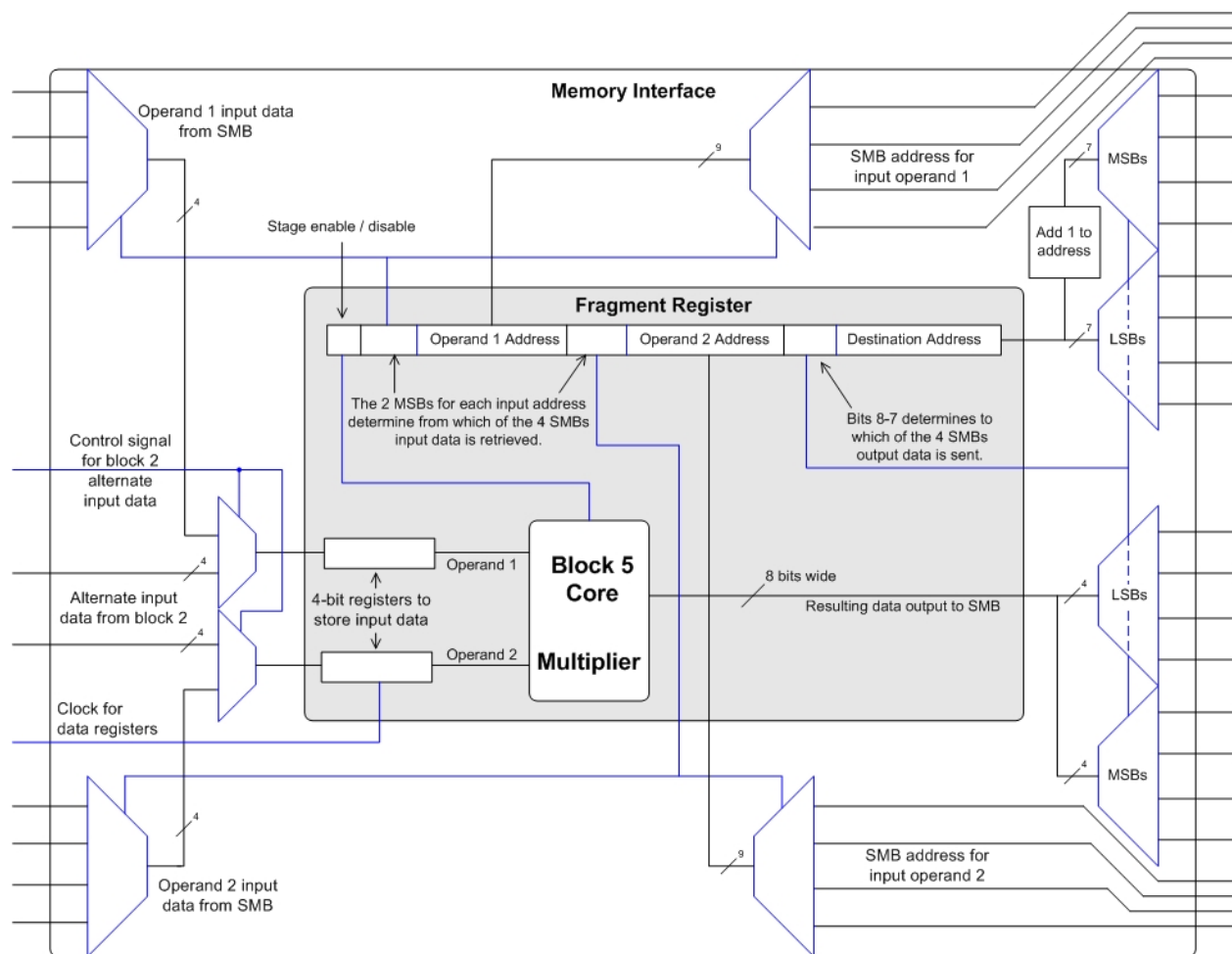
Figure 20 shows the architecture of block 4. This architecture resembles to some extent the architecture of block 1 shown in Figure 17 in the sense that it has the same multiplexers and demultiplexers for reading in and writing out data. However, block 4 is equipped with a carry-in and a carry-out chain that allows it to add or subtract operands wider than 4 bits.



**Figure 20: Architecture of block 4.**

## 4.6 Architecture of Block 5

Figure 21 shows the architecture of block 5. Since the product of a multiplication can be as wide as the sum of the bit width of the multiplicand and the multiplier, block 5 has two sets of output data and addresses to the neighboring SMBs as shown on the right side of the figure.



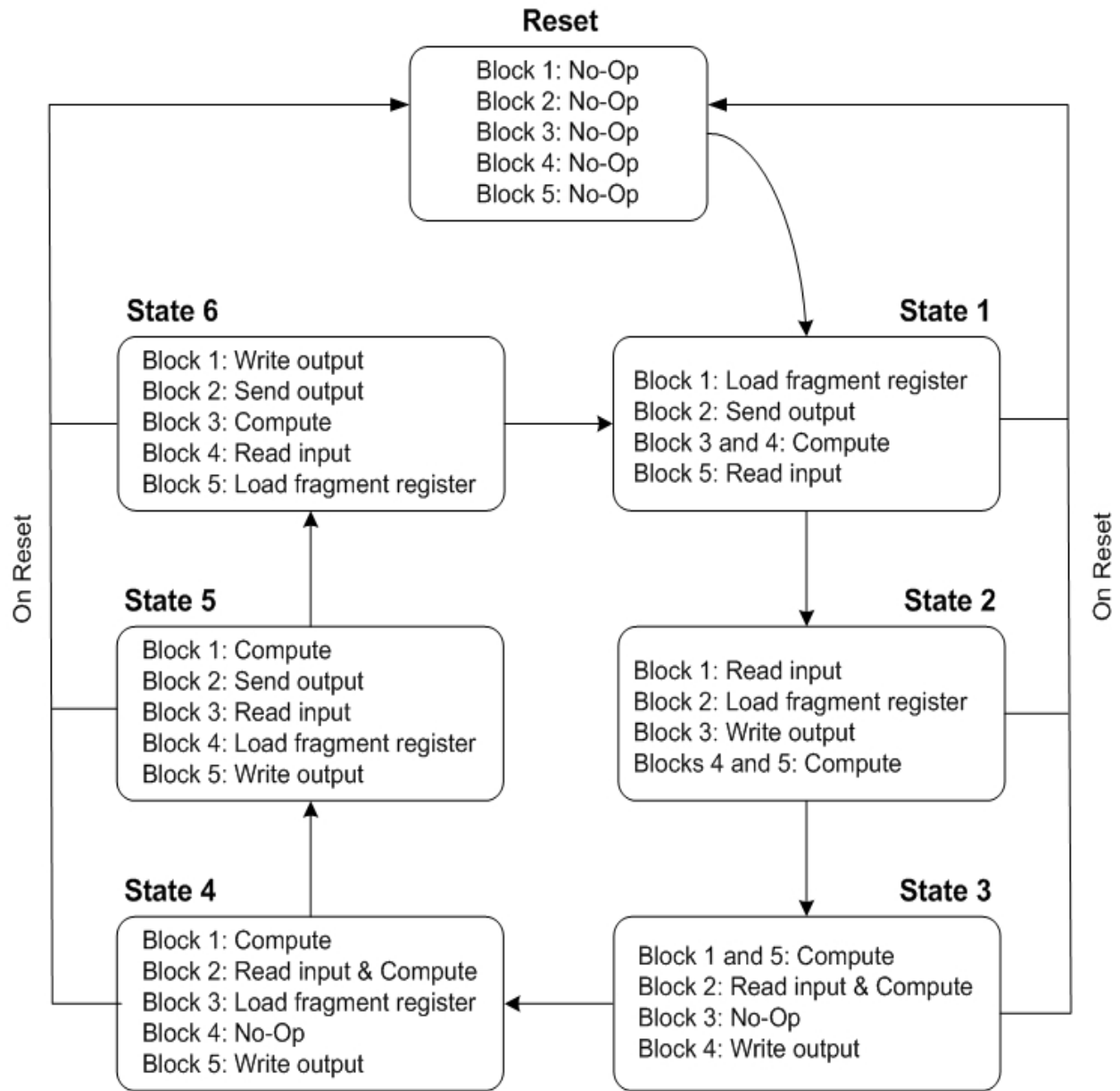
**Figure 21: Architecture of block 5.**

To accommodate the wider output bit expansion, the 8-bit output is stored in two sequential SMB locations. A '1' is therefore added to the destination address for specifying the storage location of the four most significant bits of the output data while the four least significant bits are stored in the address location specified in the fragment register. Beside this feature, the remaining architectural features are similar to the ones found in blocks 1 and 4.

#### **4.7 PE State Control**

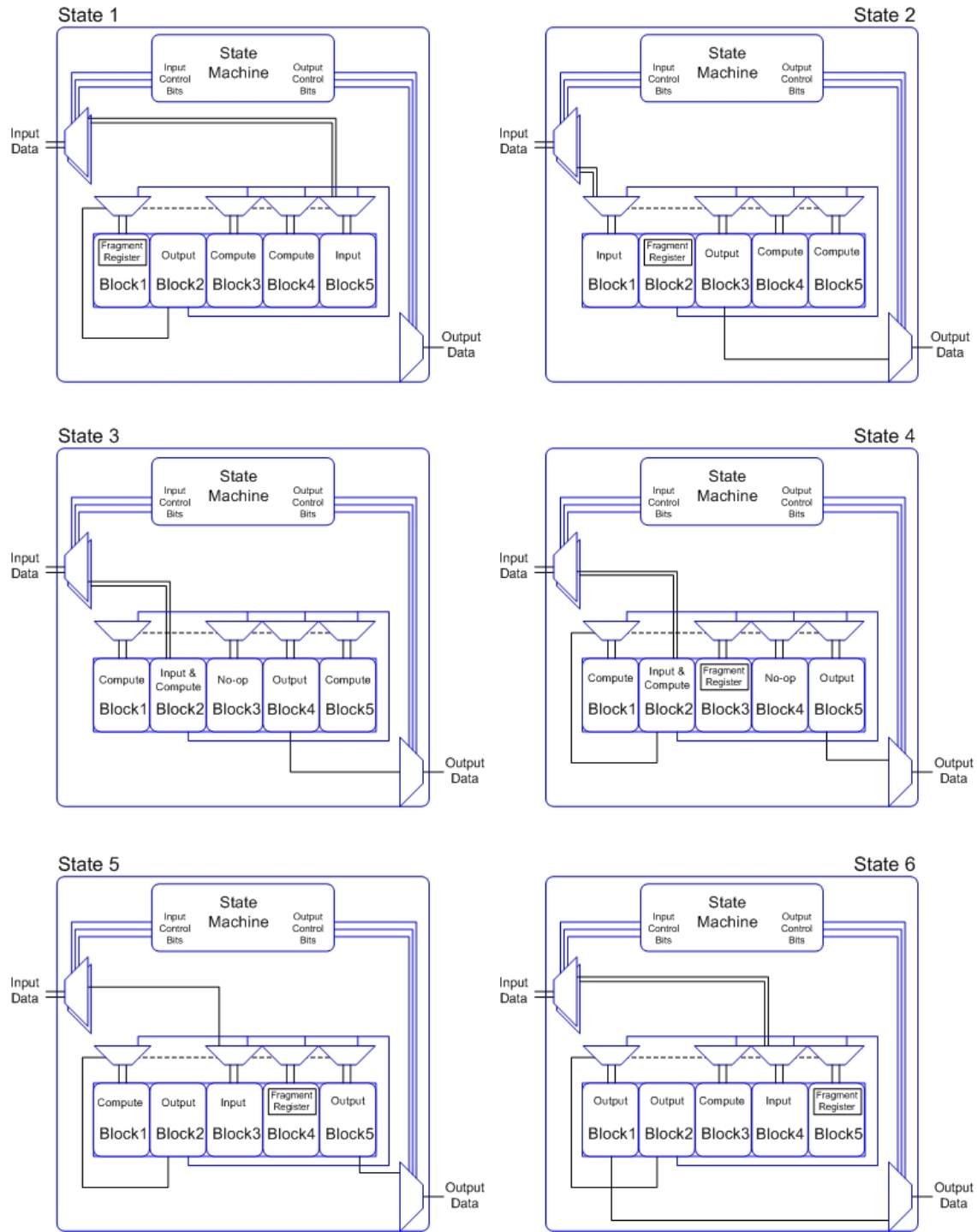
A single PE can be reconfigured by loading the decoded instructions into the fragment registers of the five operation blocks. However, the operation of the blocks inside a PE follows a pre-defined sequence that is controlled by a seven-state machine as shown in Figures 22 and 23. Each state is annotated with what each of the blocks will be doing at that state.

Although there are only five operation blocks, the two data inputs for Block 2 and two data outputs of Block 5 necessitate all six non-reset states. In association with this requirement, two compute states exist for each of the blocks for a complete cycle. This can be advantageous in that while the maximum clock frequency for the PE will be determined by the longest computation path through the operation blocks, two cycles are available to complete this computation. Therefore the PE can be clocked at twice the frequency that the longest path would otherwise require. It should also be noted that Block 2 does not require a separate compute state. This is because its outputs are used only within the PE, so as soon as it receives its input operands and the data propagates through the comparison logic, the result is then immediately utilized.



**Figure 22: State diagram of the PE controller.**





**Figure 23: State cycling in the blocks of a PE.**

## CHAPTER FIVE: MODELING AND IMPLEMENTATION OF CRYPTARRAY

This chapter describes the implementation of CRYPTARRAY. Section 5.1 presents an overview of the modeling of the array while section 5.2 describes the simulation of the VHDL entities used in modeling the components of the array. Section 5.3 explains how the PE model was synthesized while section 5.4 presents the synthesis of an SMB. Finally, section 5.5 extends the obtained synthesis results to characterize the timing and area performance of CRYPTARRAY.

### 5.1 Modeling of CRYPTARRAY

To model CRYPTARRAY, an array of 12 PEs and 21 SMBs has been coded in VHDL using a mixed level modeling. The SMBs have been modeled using a dataflow approach while the PEs and their inner components have been modeled mostly structurally. Table 5 shows the modules, their entities, and the lines of VHDL written to model each entity.

**Table 5: Breakdown of VHDL lines of code based on the modeled entities.**

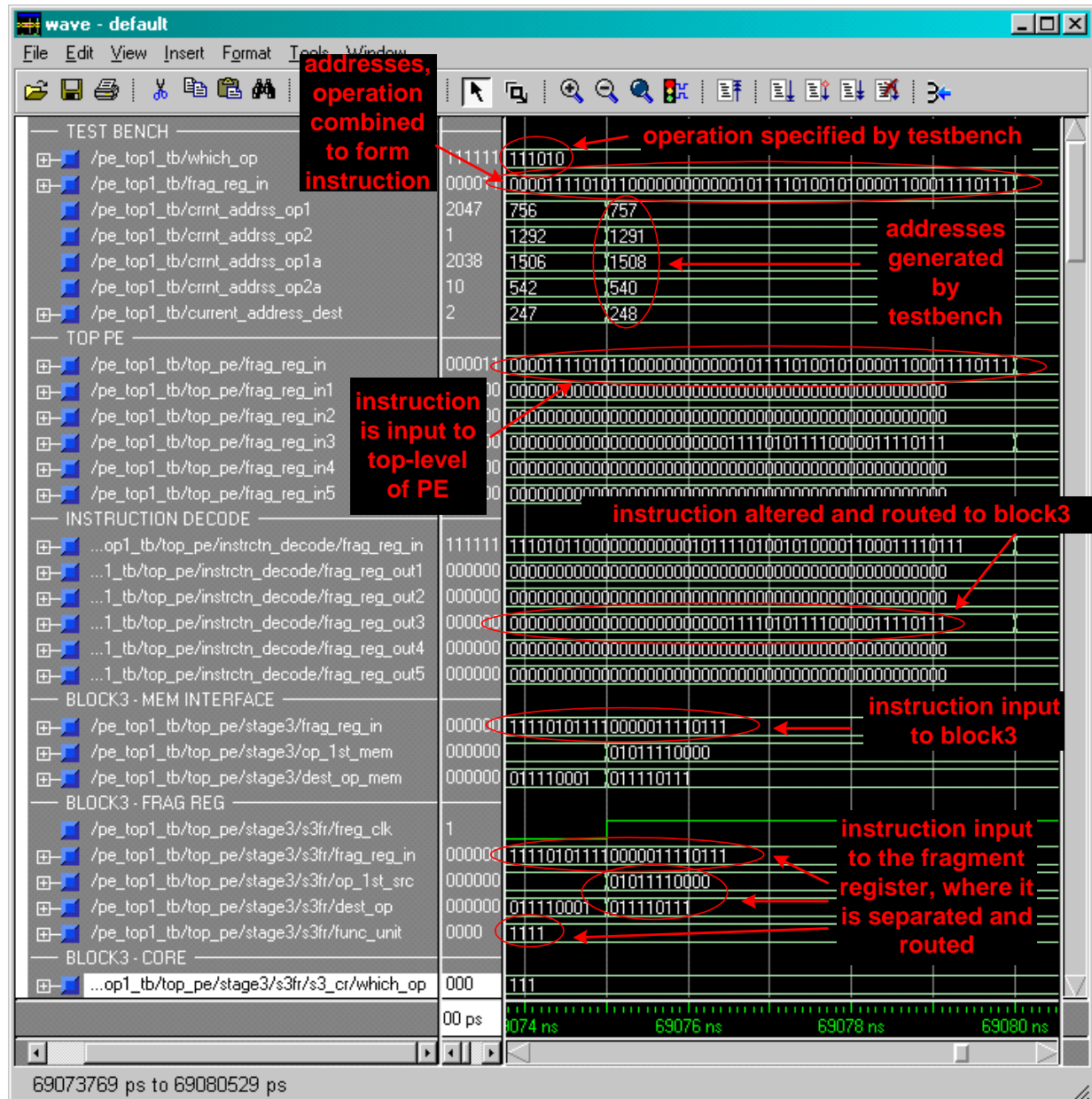
Module	Entity	VHDL Lines of Code
Array's Testbench	CRYPTARRAY_TB	172
12-PE 21-SMB Array	PE12_MEM21	1,075
	Subtotal	1,247
Shared Memory Block	MEM_512X4BIT	140
PE's Testbench	PE_TOP_TB	1,388
PE's Top Level	PE_TOP_FULL	444
	INSTRUCTION_DECODER	128
	STATE_CONTROLLER	128

	Subtotal	2,088
Block 1	MEM_INTRFCE1	111
	STG1_FRAG_REG	89
	STG1_CORE	40
	Subtotal	240
Block 2	MEM_INTRFCE2	126
	STG2_FRAG_REG	100
	STG2_CORE	44
	Subtotal	270
Block 3	MEM_INTRFCE3	101
	STG3_FRAG_REG	80
	STG3_CORE	58
	Subtotal	239
Block 4	MEM_INTRFCE4	114
	STG4_FRAG_REG	105
	STG4_CORE	57
	FULL_ADDR_4B	32
	Subtotal	308
Block 5	MEM_INTRFCE5	139
	FULL_ADDR_7BIT	41
	STG5_FRAG_REG	74
	STG5_CORE	72
	FULL_ADDR_1B	21
	Subtotal	347
Multiplexers	MUX_1BIT_2TO1	26
	MUX_1BIT_4TO1	27
	MUX_1BIT_8TO1	33
	MUX_4BIT_2TO1	29
	MUX_4BIT_4TO1	33
	MUX_4BIT_8TO1	35
	MUX_7BIT_8TO1	41
	MUX_8BIT_2TO1	33
	MUX_9BIT_4TO1	43
	MUX_9BIT_8TO1	45

	Subtotal	345
Demultiplexers	DEMUX_1BIT_1TO4	42
	DEMUX_1BIT_1TO8	95
	DEMUX_4BIT_1TO4	33
	DEMUX_4BIT_1TO8	39
	DEMUX_7BIT_1TO4	40
	DEMUX_8BIT_1TO4	42
	DEMUX_9BIT_1TO4	44
	Subtotal	335
D Flip-Flops	DFF_1BIT	30
	DFF_4BIT	31
	DFF_7BIT	34
	DFF_9BIT	36
	DFF_11BIT	38
	Subtotal	169
Total		5,728

## 5.2 Verification of the VHDL Entities

The VHDL model of each entity has been verified through extensive simulation using ModelSim XE II simulator version 5.6e. Separate simulations were performed on each individual entity. In addition, functional simulation of a prototype array consisting of 12 PEs and 21 SMBs has been performed. For the PE blocks and the top level entity of the PE, input stimuli were generated using three embedded *for* loops whereby an outermost loop feeds all possible operation combinations while two inner loops cycle through all possible combinations of instructions for those operations. Figure 24 shows a simulation snapshot of the top level module of a PE receiving an instruction into a fragment register and processing it.



**Figure 24: Instruction path through the PE simulation.**

Figure 25 shows a simulation snapshot of block 1 of the PE performing a logical NAND operation followed by a logical OR operation.

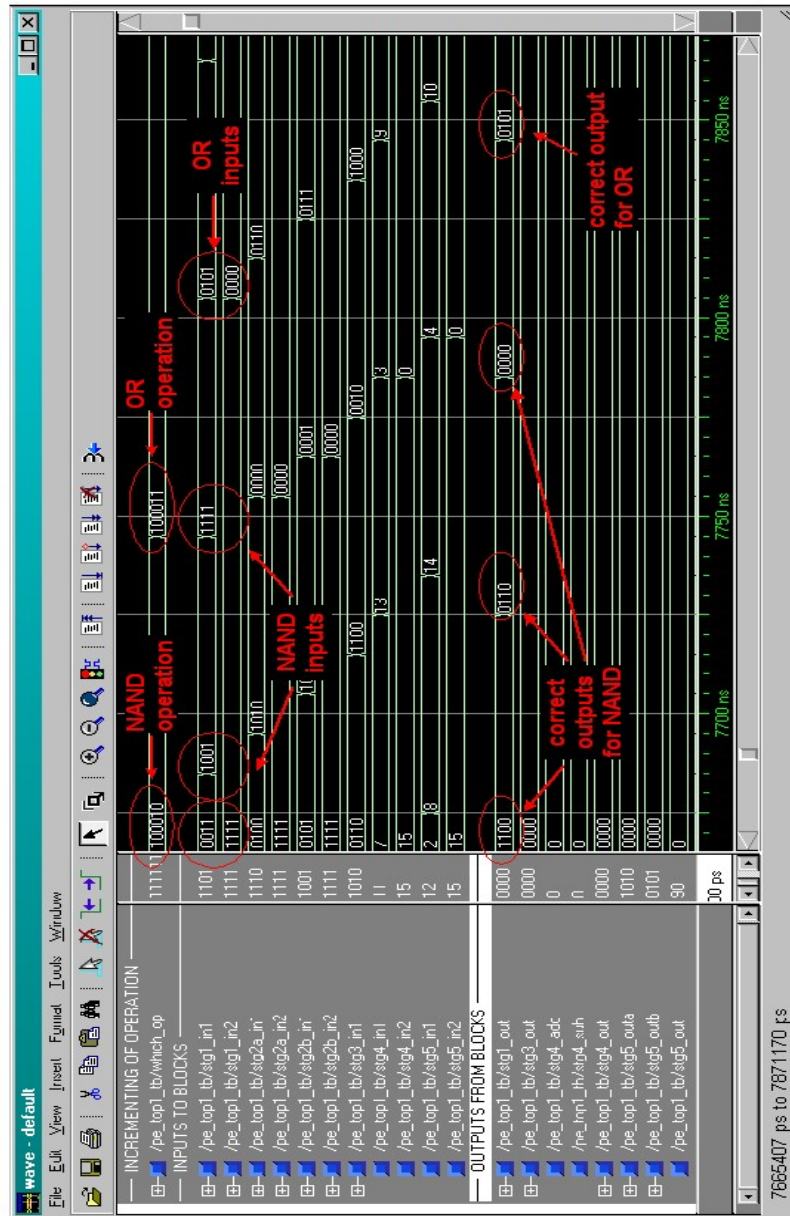


Figure 25: Simulation of the block 1 module.

Figure 26 shows a simulation snapshot of a shift to the right by one and two positions as seen from the top level of the PE, performed by the barrel shifter core of block 3.

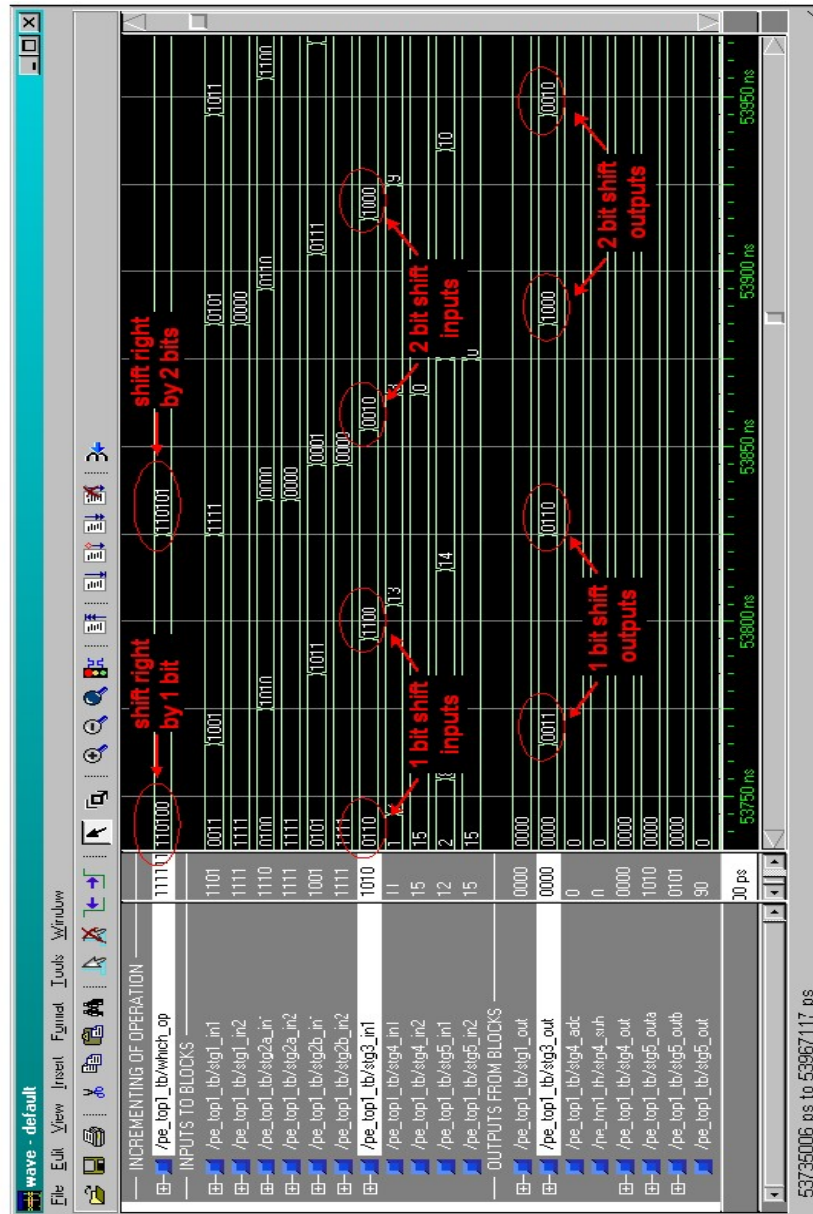


Figure 26: Simulation of the block 3 module.

Figure 27 shows a simulation snapshot of an addition with carry and a subtraction as seen from the top level of the PE performed by the adder/subtractor core of block 4 of the PE.

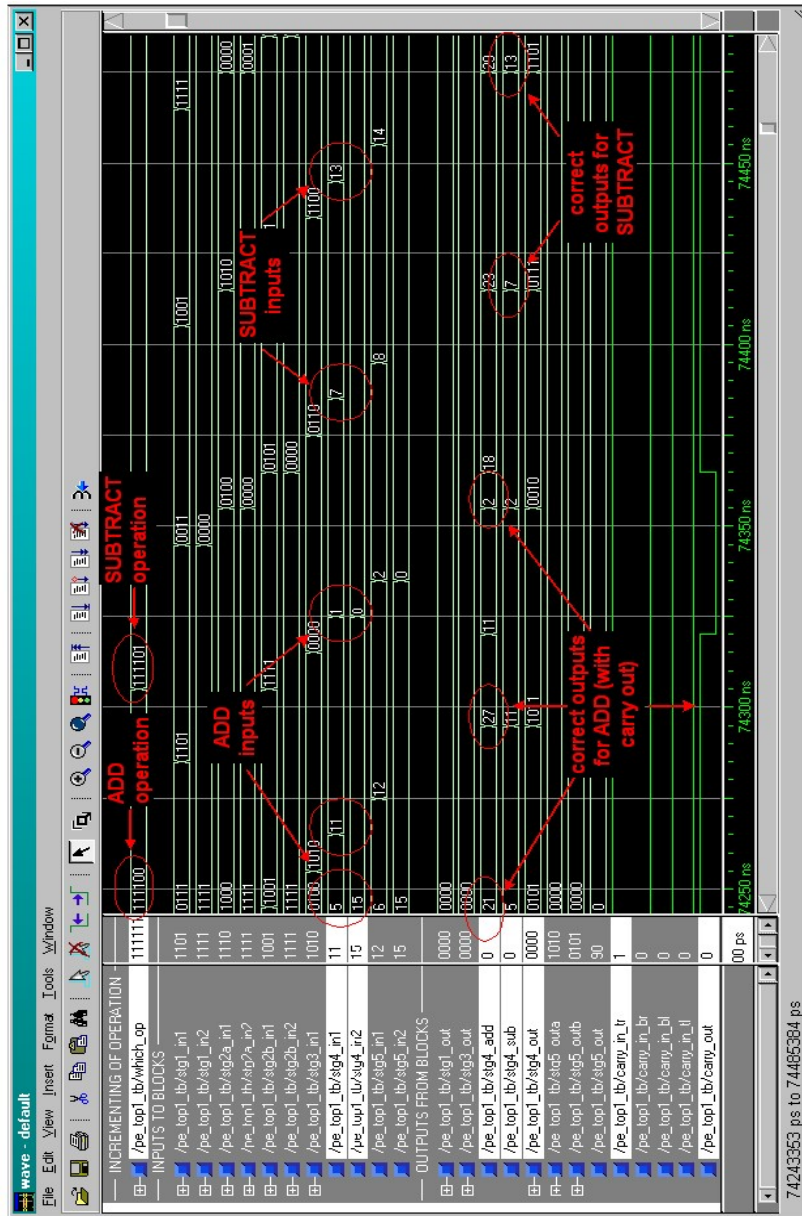


Figure 27: Simulation of the block 4 module.



The screenshot displays a logic simulator interface with two main windows: a Verilog code editor and a timing diagram.

**Verilog Code:**

```

1 //INCREMENTING OF OPERATION
2 /pe_top1_tb/which_op
3
4 //INPUTS TO BLOCKS
5 /pe_top1_tb/sig1_in1
6 /pe_top1_tb/sig1_in2
7 /pe_top1_tb/sig2a_in1
8 /pe_top1_tb/sig2a_in2
9 /pe_top1_tb/sig2b_in1
10 /pe_top1_tb/sig2b_in2
11 /pe_top1_tb/sig3_in1
12 /pe_top1_tb/sig4_in1
13 /pe_top1_tb/sig4_in2
14 /pe_top1_tb/sig5_in1
15 /pe_top1_tb/sig5_in2
16
17 //OUTPUTS FROM BLOCKS
18 /pe_top1_tb/sig1_out
19 /pe_top1_tb/sig2_out
20 /pe_top1_tb/sig4_add
21 /pe_top1_tb/sig4_sub
22 /pe_top1_tb/sig4_out
23 /pe_top1_tb/sig5_outa
24 /pe_top1_tb/sig5_outb
25 /pe_top1_tb/sig5_out

```

**Timing Diagram:**

The timing diagram shows the signals over time, with red annotations highlighting the 'Multiply Operation' and 'Correct outputs from multiply'.

**Multiply Operation:**

The diagram shows the inputs to the multiplier blocks (sig1\_in1, sig1\_in2, sig2a\_in1, sig2a\_in2, sig2b\_in1, sig2b\_in2, sig3\_in1, sig4\_in1, sig4\_in2, sig5\_in1, sig5\_in2) and the outputs (sig1\_out, sig2\_out, sig4\_add, sig4\_sub, sig4\_out, sig5\_outa, sig5\_outb, sig5\_out) over time. The operation is annotated with a red arrow pointing to the 'Multiply' block.

**Correct outputs from multiply:**

The diagram shows the outputs of the multiplier blocks (sig1\_out, sig2\_out, sig4\_add, sig4\_sub, sig4\_out, sig5\_outa, sig5\_outb, sig5\_out) over time. The outputs are annotated with red circles and arrows pointing to the 'Correct outputs from multiply' block.

**Time Scale:**

The time scale ranges from 00 ps to 80000 ns, with major ticks every 1000 ns.

54

Timing diagram for the STATE MACHINE. The diagram shows the waveforms for various signals over time. The signals are: /pe\_top1\_tb/top\_pe/st\_cntn1/gbl\_st, /pe\_top1\_tb/top\_pe/st\_cntn1/gbl\_clk, /pe\_top1\_tb/top\_pe/st\_cntn1/rnw\_clk, /pe\_top1\_tb/top\_pe/st\_cntn1/data\_in\_cntn1, /pe\_top1\_tb/top\_pe/st\_cntn1/data\_out\_cntn1, /pe\_top1\_tb/top\_pe/st\_cntn1/rneg\_clk, /pe\_top1\_tb/top\_pe/st\_cntn1/data\_clk, /pe\_top1\_tb/top\_pe/st\_cntn1/state, /pe\_top1\_tb/top\_pe/st\_cntn1/next\_state, /pe\_top1\_tb/top\_pe/st\_cntn1/data\_st, and /pe\_top1\_tb/top\_pe/st\_cntn1/rnx\_data\_st. The time scale is 40 ns. The diagram shows the state transitions and data flow of the machine.

55

### 5.3 Synthesis of the PE Model

Since the PE is the primary computation component in CRYPTARRAY, its synthesis can reveal significant insights on the performance of CRYPTARRAY. The model of a PE has been synthesized to a LUT netlist using FPGA Compiler II, version 3.5.1, from Synopsys and mapped onto a Virtex-II Pro FPGA chip using Xilinx ISE place-and-route tool, version 4.1i. Synthesis has been performed to optimize both area and speed. For each synthesis objective, three mapping effort levels were tried as shown in Table 6:

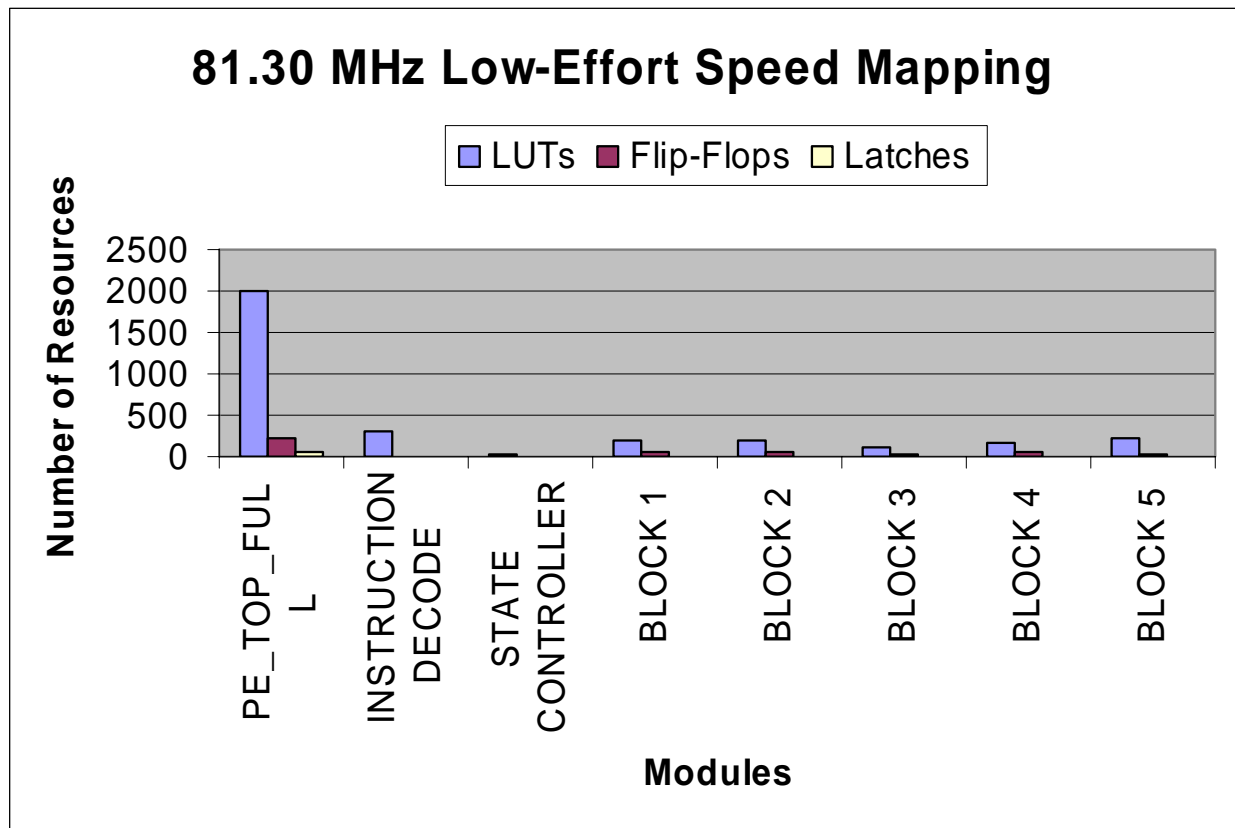
- (i) *Low* mapping effort takes the least time to compile. It is recommended if compilation time is a premium and at the same time the design timing has slack to spare.
- (ii) *Fast* mapping effort reduces the number of iterations the optimization process goes through. This effort level attempts to balance between quality of results and compilation time.
- (iii) *High* mapping effort takes longer to compile but should produce better designs. With this effort, the optimization process proceeds until it has tried all strategies.

**Table 6: Area and Speed optimization of the PE under three mapping effort levels.**

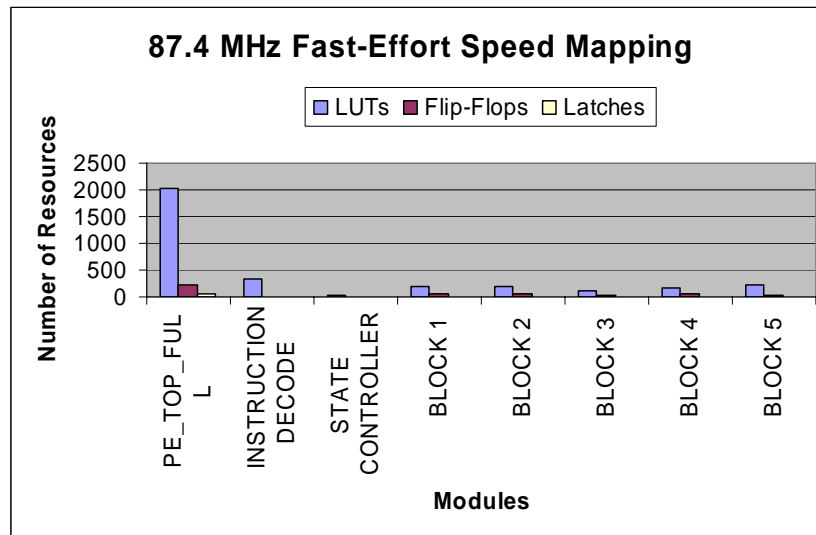
Objective	Effort Level	Obtained Frequency (MHz)	Critical Path Delay (ns)	Module	Resources Used		
					LUTs	Flip-Flops	Latches
Speed	Low	81.30	12.30	PE	2011	223	50
				Instruction Decoder	311	0	0
				State Controller	17	6	0
				Block 1	189	43	0
				Block 2	195	64	0
				Block 3	117	28	0
				Block 4	160	42	0
				Block 5	234	40	0

	Fast	87.40	11.44				
				PE	2038	223	50
				Instruction Decoder	334	0	0
				State Controller	17	6	0
				Block 1	189	43	0
				Block 2	193	64	0
				Block 3	117	28	0
				Block 4	166	42	0
				Block 5	234	40	0
	High	81.30	12.30				
				PE	2007	223	50
				Instruction Decoder	299	0	0
				State Controller	17	6	0
				Block 1	189	43	0
				Block 2	203	64	0
				Block 3	117	28	0
				Block 4	160	42	0
				Block 5	234	40	0
Area	Low	75.99	13.15				
				PE	1881	223	50
				Instruction Decoder	302	0	0
				State Controller	17	6	0
				Block 1	185	43	0
				Block 2	191	64	0
				Block 3	113	28	0
				Block 4	160	42	0
				Block 5	233	40	0
	Fast	78.55	12.73				
				PE	1897	223	50
				Instruction Decoder	318	0	0
				State Controller	17	6	0
				Block 1	185	43	0
				Block 2	191	64	0
				Block 3	113	28	0
				Block 4	160	42	0
				Block 5	233	40	0
	High	78.55	12.73				
				PE	1869	223	50
				Instruction Decoder	290	0	0
				State Controller	17	6	0
				Block 1	185	43	0
				Block 2	191	64	0
				Block 3	113	28	0
				Block 4	160	42	0
				Block 5	233	40	0

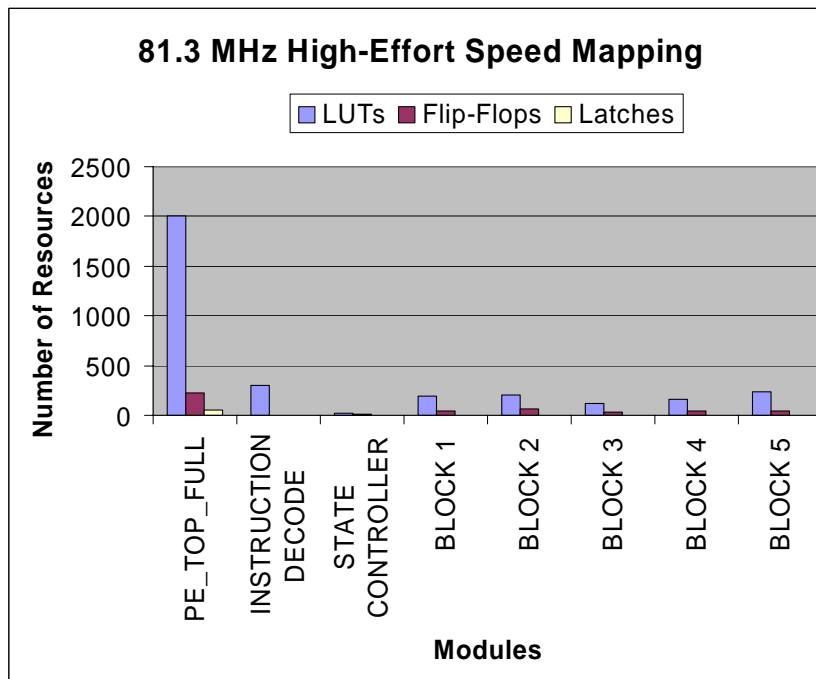
The results shown in Table 6 are plotted in Figures 30 through 35. In these tables, the x-axis represents the various primary modules that were synthesized, while the y-axis represents the number of resources required by the FPGA, categorized by look-up-tables, flip-flops, and latches.



**Figure 30: PE implementation of a timing-driven low-effort mapping.**



**Figure 31: PE implementation of a timing-driven fast-effort mapping.**



**Figure 32: PE implementation of a timing-driven high-effort mapping.**

It is clear from Figures 30 through 32 that the best clock frequency is synthesized by the fast effort level mapping. This mapping produces  $\frac{87.40 - 81.30}{81.30} \times 100 = 7.5\%$  improvement in clock frequency over the worst clock frequency obtained by any mapping with only  $\frac{2038 - 2007}{2007} \times 100 = 1.54\%$  area penalty in the implementation of the top level module of a PE.

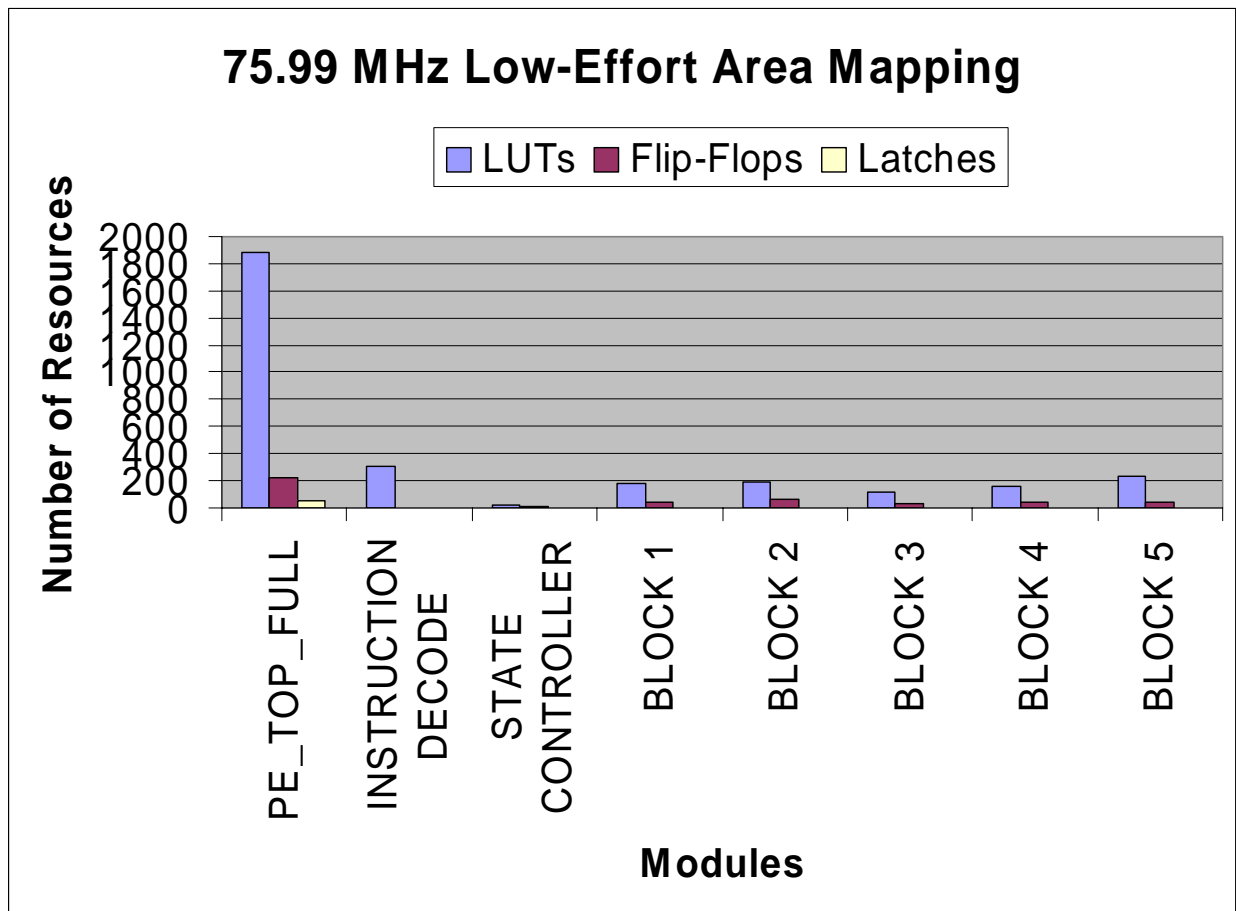
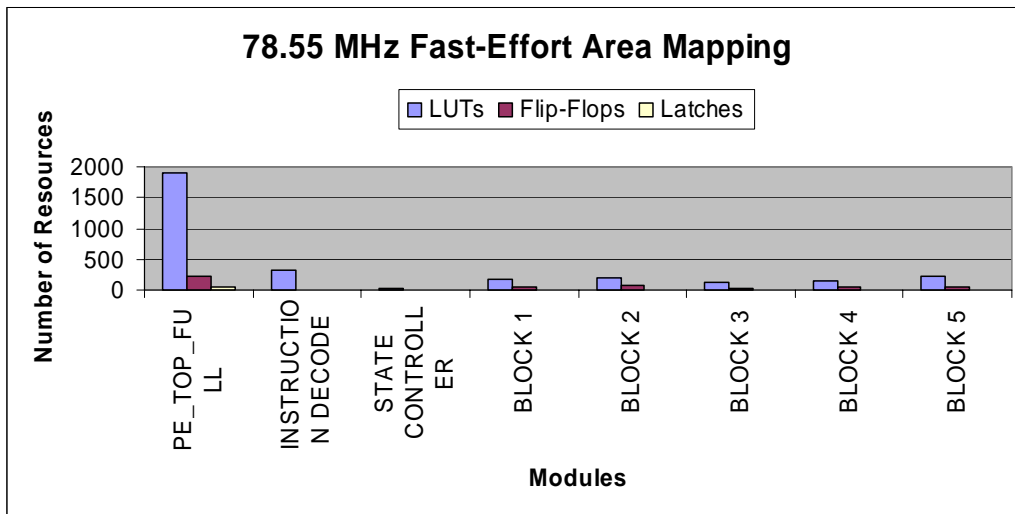
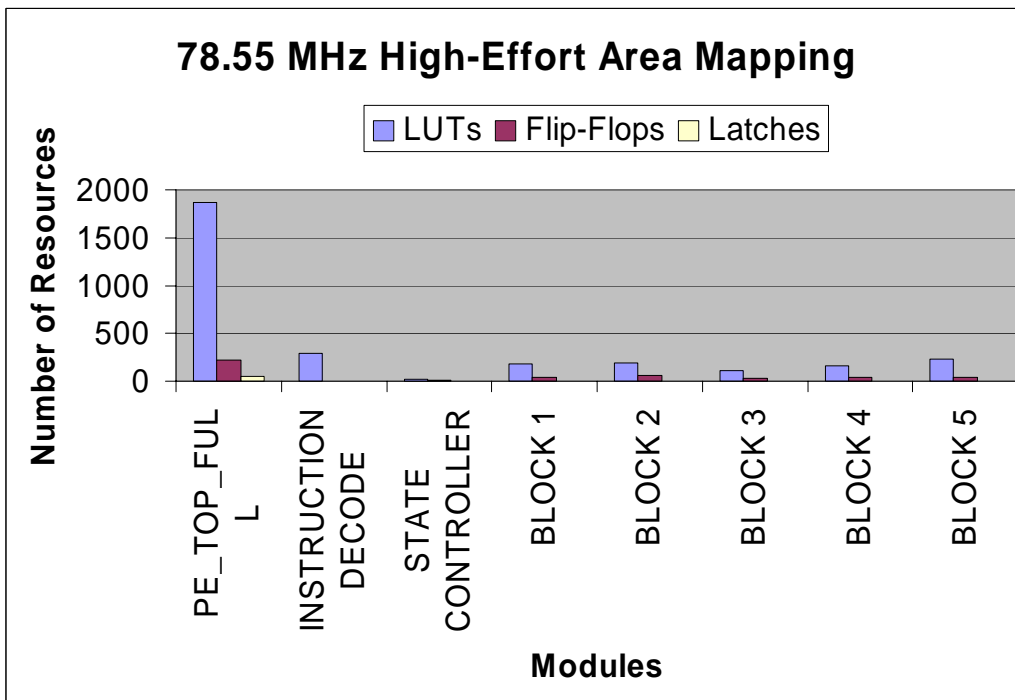


Figure 33: PE implementation of an area-driven low-effort mapping.



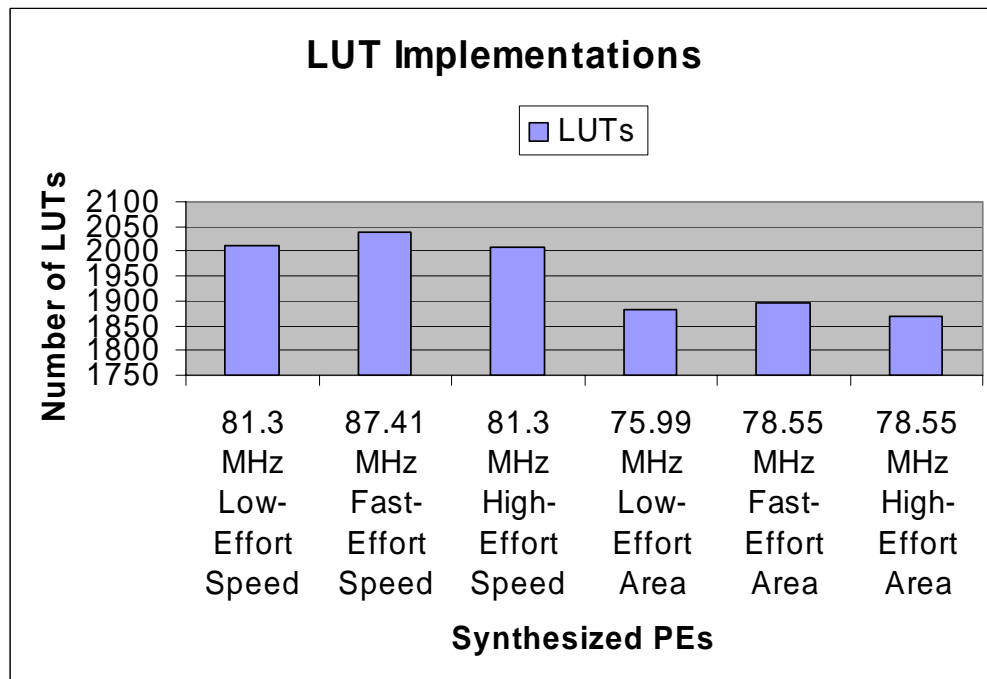
**Figure 34: PE implementation of an area-driven fast-effort mapping.**



**Figure 35: PE implementation of an area-driven high-effort mapping.**

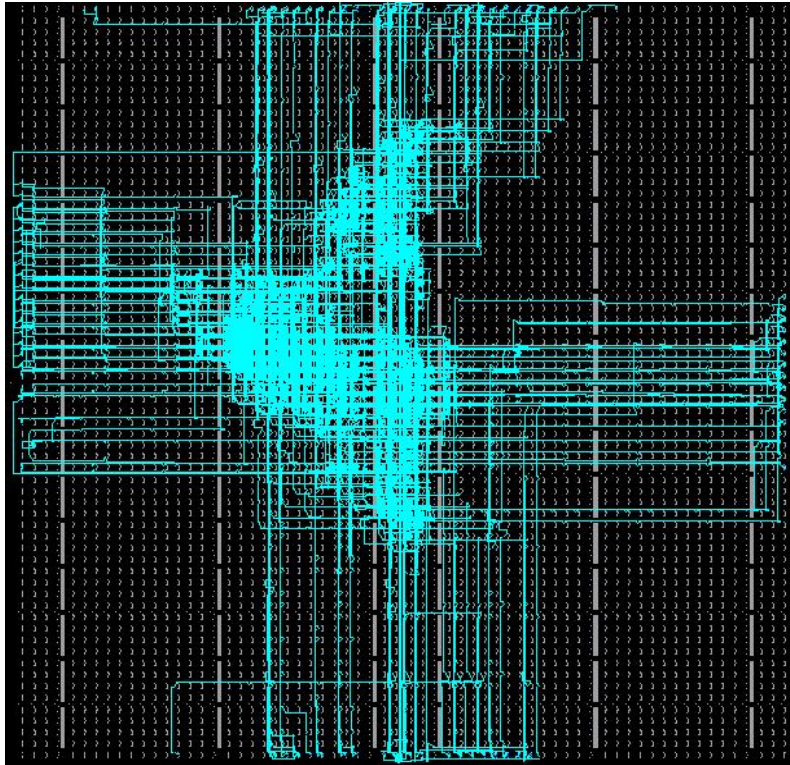


It is clear from Figures 33 through 35 that the best area implementation is obtained with the high level effort mapping. This mapping produces only  $\frac{1897-1869}{1869} \times 100 = 1.49\%$  marginal improvement in area cost over the worst area implementation of the top level of a PE. This improvement comes with a slight improvement of  $\frac{78.55-75.99}{75.99} \times 100 = 3.36\%$  in clock frequency. Figure 36 contrasts the area cost in terms of LUTs for the top level module of a PE across the various timing and area-driven mapping effort levels where the leftmost three bars in the figure represent timing-driven mappings while the rightmost three bars represent area-driven mappings.



**Figure 36: Summary of PE implementations.**

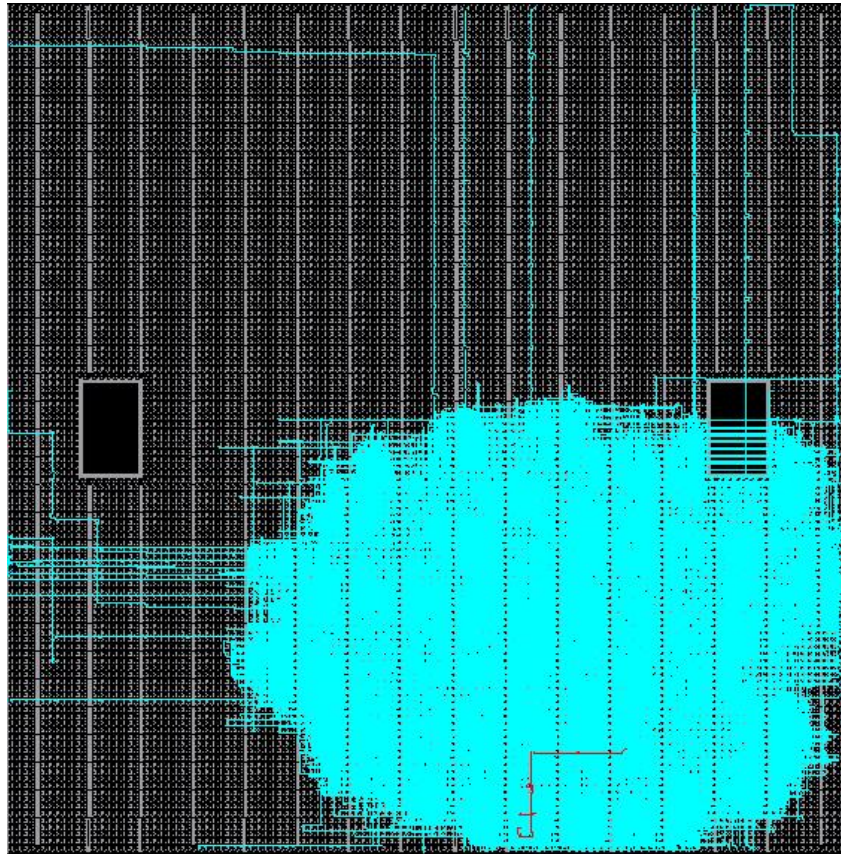
As the figure shows, the best timing-driven implementation (the second bar from the left in the figure) is obtained with a timing-driven fast effort level mapping while the worst timing-driven implementation (the fourth bar from the left of the figure) is obtained with an area-driven low effort level mapping. The best implementation produces  $\frac{87.40 - 75.99}{75.99} \times 100 = 15.01\%$  improvement in clock frequency with  $\frac{2038 - 1881}{1881} \times 100 = 8.34\%$  area penalty in terms of LUTs used in the implementation of the top level module of a PE. Figure 37 shows a partial view of the floorplan of synthesized PE onto a Xilinx Virtex-II Pro XC2VP125.



**Figure 37: Partial view of the floorplan of a synthesized PE onto a Virtex-II Pro XC2VP125.**

#### **5.4 Synthesis of the SMB Model**

The SMB model has been synthesized to a LUT netlist using FPGA Compiler II, version 3.5.1, from Synopsys and mapped onto a Virtex-II Pro FPGA chip using Xilinx ISE place-and-route tool, version 4.1i. Synthesis has been performed to optimize speed. Figure 38 shows a partial view of the layout of an SMB onto a Xilinx Virtex-II Pro XC2VP125, while Table 7 shows the synthesis results of a 512 x 4-bit timing-optimized SMB.



**Figure 38: Partial view of the floorplan of a synthesized SMB onto a Virtex-II Pro XC2VP125.**

**Table 7: Synthesis results of a 512 x 4-bit SMB.**

Parameter	Value
Frequency (MHz)	80.9
Critical Path Delay (ns)	12.36
LUTs	14899

As the table shows, a read or write from the SMB can be completed in 12.36 ns. In addition, an SMB consumes a relatively large number of LUTs compared to a PE. In effect, an SMB occupies

$\frac{14899}{2038} = 7.31$  more area than the largest PE implementation in terms of LUTs.

## **5.5 Performance of CRYPTARRAY**

In this section, a brief analysis of the performance of a prototype array will be presented in terms of clock frequency, area cost, and bandwidth.

### **5.5.1 Clock Frequency**

By integrating SMBs with PEs to form an array of a given size, the array's clock frequency will be limited by the slowest among SMBs and PEs. Since the clock frequency of the SMBs is lower than the best frequency of the PEs, the array's frequency will subsequently be determined by that of the SMBs. As a result, a prototype array running at the frequency of the SMBs, which is 80.9 MHz, can be obtained by assembling a number of SMBs and PEs. In this case, if the array is running in static reconfiguration mode, a PE block of the array can read the operands from an SMB, perform its operation, and write the result to an SMB in  $3T$  where  $T$  is the clock cycle of

the array. Since  $T = \frac{1}{80.9 \text{ MHz}} \times 10^3 = 12.36 \text{ ns}$ , a PE block can complete an instruction in  $3T =$

$3 \times 12.36 \text{ ns} = 37.08 \text{ ns}$ . With this PE block performance, the array can have a throughput

$$\rho_{static} = \frac{1 \text{ sec}}{37.08 \text{ ns}} = 26,968,716.28 \text{ outputs/sec.}$$

However, if the array is running in dynamic reconfiguration mode, a PE block has to load its fragment register before proceeding with the steps of reading the operands from an SMB, computing, and writing the result to an SMB. For a

lack of an accurate estimate for the time to dispatch and load an instruction into a fragment register, one can assume for simplicity that this time can be equal to  $T$ . In this case, a PE block

can perform the four steps in  $4T = 4 \times 12.36 \text{ ns} = 49.44 \text{ ns}$ . With such a block performance, the

$$\text{array can have a throughput } \rho_{dynamic} = \frac{1 \text{ sec}}{49.44 \text{ ns}} = 20,226,537.28 \text{ outputs/sec.}$$

### 5.5.2 Area Cost

Based on the results shown in Table 6, a PE can consume 2038 LUTs while an SMB can occupy 14899 LUTs as shown in Table 7. A tile consisting of an SMB and a PE can occupy  $3288 + 14899 = 18187$  LUTs. It is clear that to prototype CRYPTARRAY on FPGAs, large capacity FPGA chips are needed. For example, the Virtex-II Pro XC2VP125 which contains 125136 LUTs can pack an array consisting of only 12 PEs and 12 SMBs. To implement an array of reasonable size, it is necessary to use a multi-chip configuration.

### 5.5.3 Possible Bandwidth

If a single chip configuration is considered, an array consisting of 12 PE and 12 SMBs can be packed within a Virtex-II Pro XC2VP125. Assume that the chessboard layout of the array is preserved in its placement and layout onto the chip. It is reasonable to view the layout as a set of three rows where a row can either have one or two SMBs connected to the IO pins of the chip. Since each SMB is a 512 x 4-bit memory block, an SMB can output 4 bits each 37.08 ns if the array is running in static reconfiguration mode. This means that an SMB can output

$\frac{1 \text{ sec}}{37.08 \text{ ns} \times 10^{-9}} \times 4 \text{ bits} = 102.87 \text{ Mbps}$ . If a multi-chip configuration is used, it would take only

10 SMBs be connected to the IO pins of the chips to produce up to 1.02 Gbps. Note that an SMB consumes only 4 IO pins on a chip when an FPGA chip such as the Virtex-II Pro XC2VP125 has 1200 IO pins. It is clear that such a bandwidth can easily support the processing requirements of many cryptographic algorithms running on Internet servers.

### 5.5.4 Summary of CRYPTARRAY's Performance

Table 8 summarizes the performance characteristics of the components of CRYPTARRAY where the leftmost column shows the array's components while the second column shows the components area in LUTs. The third column shows the ratio of the component area to the area of the Virtex II Pro XC2VP125 chip. The fourth column shows the best clock frequency of the component obtained through synthesis while the last column shows the bandwidth produced by the component based on the obtained clock frequency.

**Table 8: Summary of the performance characteristics of CRYPTARRAY's components.**

Component	Area (LUTs)	Area Ratio $\frac{\text{Component Area}}{\text{Chip Area}}$	Clock Frequency (MHz)	Bandwidth (Mbps)
PE	2038	$\frac{2038}{125136} = 0.0162$	87.4	
SMB	14899	$\frac{14899}{125136} = 0.119$	80.9	102.87
$N \times N$ Array $= \frac{N^2}{2} \text{SMBs} + \frac{N^2}{2} \text{PEs}$	$\left( \frac{14899N^2}{2} \right) + \left( \frac{2038N^2}{2} \right)$ $= 7449.5N^2$	$\frac{7449.5N^2}{125136}$ $= 0.0595N^2$	80.9	102.87M

$N$  = number of array tiles (a tile can be a PE or a SMB);  $M$  = number of array SMBs connected to the chip output pins; Chip Area = area of a Virtex-II Pro XC2VP125 chip = 125136 LUTs;

An  $N \times N$  array contains  $N$  tiles in each plane dimension where a tile can be either an SMB or a PE. For simplification, it is assumed that an  $N \times N$  array will have all its SMBs placed on the periphery of the array and subsequently directly connected to the IO pins of the chip. In such a placement, there are  $\frac{N}{2} \text{SMBs} + \frac{N}{2} \text{PEs}$  in each row if  $N$  is an even natural number. In this case, an array of  $N$  rows consists of  $\frac{N^2}{2} \text{SMBs} + \frac{N^2}{2} \text{PEs}$ . Since only a subset of the SMBs in the array of size  $M$  is connected to the output pins of the chip, the bandwidth of the array depends primarily on the cardinality of this subset. Note that since the PEs are not connected directly to the chip IO pins, they cannot support any bandwidth at all.

## CHAPTER SIX: CONCLUSION

This thesis proposes CRYPTARRAY, a two-dimensional, scalable architecture in which bus-based communication is replaced by distributed shared memory communication. At the physical level, the length of the wires is kept to a minimum. The array is organized as a chessboard in which the dark and light squares represent PEs and SMBs respectively. The granularity and resource composition of the PEs is specifically designed to support the computing operations encountered in cryptographic algorithms in general, and symmetric algorithms in particular. Communication can occur only between neighboring PEs through local SMBs. Because of the chessboard layout, the architecture can be reconfigured to allow computation to proceed as a pipelined wave in any direction. This organization offers a high computational density in terms of datapath resources and a large number of distributed storage resources that easily support a high degree of parallelism and pipelining. In addition, this architecture provides a high degree of flexibility supported by its reconfigurability. Based on the obtained experimental results, this architecture can deliver a performance that can easily address the bandwidth requirements of many cryptographic applications if sufficient resources are available.

While this thesis shows how CRYPTARRAY can address the performance requirements of most cryptographic applications, future work can improve further the proposed architecture if the following issues are considered:



- (i) *What would be the optimal size of the SMBs considering the variety of cryptographic algorithms?* The answer to this question can minimize SMB waste when mapping cryptographic applications. This answer can be obtained by mapping representative cryptographic algorithms on CYRPTARRAY and evaluate memory usage for each algorithm in order to derive an optimal size of the SMBs.
- (ii) *How many ports can an SMB have in order to simplify the access of the PE to the SMB?* By increasing the number of ports of an SMB, more than one PE can access the SMB at the same time. This capability can increase the degree of parallelism in the array by simplifying the state controller used to control the access of the PE blocks to an SMB. However, implementing multi-ports SMBs onto FPGA chips is not area efficient. Such SMBs can be built in an area-economic fashion if implemented as custom circuits on ASICs. Examples of such implementations can be found in the multi-port memories offered by IDT in which each memory cell consists of four CMOS transistors and each port addressing the cell consists of two transistors [40].
- (iii) *How can the bit width of the array be improved to handle asymmetric algorithms?* Since asymmetric algorithms use keys that are thousands of bits wide, it is not clear if a 4-bit architecture is suitable for executing these algorithms. Mapping and profiling these algorithms on CRYPTARRAY can reveal valuable insights on how the bit width can be changed to handle efficiently secret-key cryptography.

- (iv) *How can CRYPTARRAY's throughput be measured accurately?* It seems that an accurate measure of the array's bandwidth can be obtained by mapping representative applications and tallying the number of encrypted packets per seconds.

The answers to these questions can increase the flexibility of the array and improve its performance further in supporting cryptographic processing on Internet-based applications.

## REFERENCES

- [1] C. Kaufman, R. Perlman, and S. M., *Network security: Private communication in a public world*: Prentice Hall, 1995.
- [2] B. Schneier, *Applied Cryptography*, Second Edition ed: John Wiley & Sons, 1994.
- [3] R. A. Mollin, *An introduction to cryptography*. Boca Raton, FL: Chapman & Hall/CRC, 2001.
- [4] H. X. Mel and D. Baker, *Cryptography decrypted*. Upper Saddle River, NJ, 2001.
- [5] M. S. Merkow and J. Breithaupt, *The complete guide to Internet security*: AMACOM, 2000.
- [6] J. Burke, J. McDonald, and T. Austin, "Architectural support for fast symmetric-key cryptography," *Architectural Support for Programming Languages and Operating Systems*, 2000, pp. 178-189.
- [7] S. Moore, "Enhancing security performance through IA-64 architecture," Intel Coporation, 1999.
- [8] Z. Shi and R. B. Lee, "Bit permutation instructions for accelerating software cryptography," *International Conference on Application-Specific Systems*, 2000, pp. 138-148.
- [9] X. Lai, *On the design and security of block ciphers*: Hartung-Gorre Veerlag, 1992.
- [10] B. Schneier, J. Kelsey, D. Whiting, C. Wagner, and N. Ferguston, "Twofish: A 128-bit block cipher," Counterpane Labs 1998.
- [11] Counterpane Labs, "The blowfish encryption algorithm," Counterpane Labs, 2002.
- [12] J. Goodman and A. P. Chandrakasan, "An energy-efficient reconfigurable public-key cryptography processor," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 11, pp. 1808-1820, Nov. 2001.
- [13] L. Wu, C. Weaver, and T. Austin, "CryptoManiac: A fast flexible architecture for secure communication," *Interantional Symposium on Computer Architecture*, 2001, pp. 110-119.

- [14] S. S. Raghuran and C. Chakrabarti, "A programmable processor for cryptography," *IEEE International Symposium on Circuits and Systems*, 2000, pp. V/685-V/688.
- [15] R. Ho, K. W. Mai, and M. A. Horowitz, "The future of wires," *Proceedings of the IEEE*, vol. 89, no. 4, pp. 490-504, Apr. 2001.
- [16] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz, "Smart memories: A modular reconfigurable architecture," *Annual International Symposium on Computer Architecture*, 2000, pp. 161-71.
- [17] W. J. Dally and S. Lacy, "VLSI architectures: Past, present, and future," *Conference on Advanced Research in VLSI*, 1999, pp. 232-241.
- [18] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpfen, M. Frank, S. Amarasinghe, and A. Agarwal, "The raw microprocessor: A computational fabric for software circuits and general-purpose programs," *IEEE Micro*, vol. 22, no. 2, pp. 25-35, Mar./Apr. 2002.
- [19] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519-521, 1985.
- [20] N. Takagi, "A radix-4 modular multiplication hardware algorithm for modular exponentiation," *IEEE Transactions on Computers*, vol. 41, no. 8, pp. 949-956, Aug. 1992.
- [21] M. Shand and J. Vuillemin, "Fast implementations of RSA cryptography," *IEEE Symposium on Computer Arithmetic*, 1993, pp. 252-259.
- [22] S. E. Eldridge and C. D. Walter, "Hardware implementation of Montgomery's modular multiplication algorithm," *IEEE Transactions on Computers*, vol. 42, no. 6, pp. 693-699, June 1993.
- [23] C. D. Walter, "Systolic modular multiplication," *IEEE Transactions on Computers*, vol. 42, no. 3, pp. 376-378, Mar. 1993.
- [24] J.-H. Guo, C.-L. Wang, and H.-C. Hu, "Design and implementation of an RSA public-key cryptosystem," *IEEE International Symposium on Circuits and Systems*, 1999, pp. 504-507.
- [25] A. A. Tiountchik, "Systolic modular exponentiation via Montgomery algorithm," *Electronic Letters*, vol. 34, no. 9, pp. 874-875, 1998.

- [26] A. Ejnioui and N. Ranganathan, "Systolic algorithms for tree pattern matching," *International Conference on Computer Design*, 1995, pp. 650-655.
- [27] V. Krishna, N. Ranganathan, and A. Ejnioui, "A tree matching chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 2, pp. 277-280, June 1999.
- [28] S. Krishna, N. Ranganathan, and A. Ejnioui, "A VLSI architecture for object recognition using tree matching," *International Conference on Application-Specific Systems, Architectures and Processors*, 2002, pp. 325-334.
- [29] S. Ishii, K. Ohyama, and K. Yamanaka, "A single-chip RSA processor implemented in a 0.5-mm rule gate array," *IEEE International ASIC Conference and Exhibit*, 1994, pp. 433-436.
- [30] J.-Y. Leu and C.-L. Wu, "A scalable low-complexity digit-serial VLSI architecture for RSA cryptosystems," *IEEE Workshop on Signal Processing Systems*, 1999, pp. 586-595.
- [31] A. Curiger, H. Bonnenberg, R. Zimmermann, N. Felber, H. Kaeslin, and W. Fichtner, "VINCI: VLSI implementation of the new secret-key block cipher IDEA," *IEEE Custom Integrated Circuits Conference*, 1993, pp. 15.5.1-15.5.4.
- [32] S. Wolter, H. Matz, A. Schubert, and R. Laur, "On the VLSI implementation of the International Data Encryption Algorithm IDEA," *IEEE International Symposium on Circuits and Systems*, 1995, pp. 397-400.
- [33] Y.-K. Lai and Y.-C. Shu, "VLSI architecture design and implementation for BLOWFISH block cipher with secure modes of operation," *IEEE International Symposium on Circuits and Systems*, 2001, pp. 57-60.
- [34] IEEE Standards Board, "IEEE standard specifications for public-key cryptography," 2000.
- [35] K. U. Jarvinen, M. T. Tommiska, and J. O. Skytta, "A fully pipelined memoryless 17.8 Gbps AES-128 encryptor," *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, 2003, pp. 207-215.
- [36] A. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 4, pp. 545-557, Aug. 2001.
- [37] P. Chodowiec, P. Khuon, and K. Gaj, "Fast implementations of secret-key block ciphers using mixed inner- and outer-round pipelining," *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, 2001, pp. 94-102.

- [38] K. Leitjen-Nowak and J. L. Van Meerbergen, "An FPGA architecture with enhanced datapath functionality," *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, 2003, pp. 195-204.
- [39] R. R. Taylor and S. C. Goldstein, "A high-performance flexible architecture for cryptography," *Workshop on Cryptographic Hardware and Embedded Systems*, 1999
- [40] J. R. Mick, "Introduction to IDT's four-port SRAMs", Application Note AN-45, IDT Corporation, Aug. 1999, available at [http://www1.idt.com/pcms/tempDocs/7052\\_AN\\_52995.pdf](http://www1.idt.com/pcms/tempDocs/7052_AN_52995.pdf).