VHDL Implementation of PPR Systolic Array Architecture for Polynomial GF(2^m) Multiplication

by

Ali Nia
BEng, University of Nortumbria, 2000

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Applied Science

in the Department of Electrical and Computer Engineering

VHDL Implementation of PPR Systolic Array Architecture for Polynomial GF(2^m) Multiplication

by

Ali Nia
BEng, University of Nortumbria, 2000

Supervisory Committee

Dr. Fayez Gebali, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Mihai Sima, Departmental Member
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. Fayez Gebali, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Mihai Sima, Departmental Member
(Department of Electrical and Computer Engineering)

ABSTRACT

This thesis is devoted to efficient VHDL design of Systolic Array Architecture for Polynomial GF(2^m) multiplication. The hardware implements the Processor Elements(PE) and Systolic Array design for Progressive Product Reduction (PPR) method proposed by Gebali and Atef [2]. The experiment first implements a simpler irreducible polynomials GF(2^5) based on the defined algorithms for PPR in order to confirm the functionality of the design and then tries the bigger value of m for GF(2^{133}) and GF(2^{233}), recommended by NIST. The thesis is comparing the three designs based on their power consumption, Maximum Data path delay and device utilization. It also looking in to the different optimization method for the designs and recommends a design optimization based on circuit modification.

Contents

List of Tables

List of Figures

List of Abbreviations

DAG Directed Acyclic Graph

EDIF Electronic Data Interchange Format

FPGA Field Programmable Gate Array

GF Galois Field

NCF Netlist Constraints File

NIST National Institute of Standards and Technology

PE Processing Element

PPR Progressive Product Reduction

RTL Register Transfer Level

UCF User Constraints File

VHDL VHSIC Hardware Description Language

ACKNOWLEDGEMENTS

I would like to thank:

My supervisor Dr. Fayez Gebali for many things,but specially for his vision and positive encouragement, not only through this thesis but all through my Master degree program. I also would like to thank Dr. Mihai Sima and Dr. Haytham El Miligi for their kind guidance through my thesis.

Finally a special thank to my wife Tayebeh and my son Artin, for their patience, love and support.

Chapter 1

Introduction

1.1 Polynomial Galois Field (2^m) Multiplier

Finite or Galois Fields have many important and practical applications. Galois Field can be applied to error correcting coding computer algebra systems, information theory, number theory and public key cryptosystems [5]. The operation of multiplication in Galois Fields is quite different from the usual binary arithmetic operation and is more efficient and more widly used compared with multipliers based on normal and dual basis. Numerous hardware architecture have been proposed for polynomial basis finite field multiplication over GF(2^m) [7] [12] [4] [9] [6] [1] [8] [3]. This thesis is devoted to the efficient VHDL architecture design for implementation of systolic array for polynomial GF(2^m) multiplication which has never been tried before. It concentrate on unpublished work by Dr. Fayez Gebali and Atef Ibrahim [2] on GF(2^m) multiplication. Dr. Gebali and Ibrahim [2] explore systolic architecture for iterative algorithms of finite field multiplication over GF(2^m) based on Irreducible trinomials using systematic linear and non linear techniques that combines affine and non linear Processing Element (PE) scheduling and assignment of computation to processors. The technique iterative multiplication algorithm using Progressive Product Reduction (PPR) is discussed in their paper. Design 1_11 is explained in details in Gebali and Ibrahim work and is the basis of the work for this thesis.

1.2 Contributions

This thesis is introducing the PPR technique and its implementation for Galois Field multiplication which contributes the following

1. Propose systolic architecture for converting GF(2^m) multiplication into an iterative expressions.
2. Apply formal technique for mapping the iteration to obtain proceesing arrays.
3. Design hardware for systolic array obtained.
4. Create the test benches to prove and demonstrate the functionality of the hardware.
5. Verify the hardware design by developing a Matlab code to confirm the correct functionality of the hardware, defined by VHDL implementation and compare with previously published results.

1.3 Organization of the Thesis

The organization of the thesis is as follows:

Chapter 2 presents some mathematical background about finite field. It introduces the concepts of the Field, Prime Field, Extension field, Extension Binary Field and Algorithm for definition of finite field over GF(2^m) using irreducible polynomial.

Chapter 3 describes in details the Progressive Product Reduction (PPR) method and discusses the mathematical theory behind the design. It also explains the Parallelizing technique, Scheduling Function and Projection function techniques for PPR.

Chapter 4 includes the design overview, which contains description of the PE structures and systolic array structure for Systolic Multiplier design $s_1 d_{11}$.

Chapter 5 contains design implementation using VHDL and Matlab Verification. The critical path, On chip device utilization and power consumption for the design are discussed in this chapter.

Chapter 6 discusses the speed optimization techniques to minimizes the delay in data path. This chapter propose two optimization techniques, first using ISE tool for optimization and second proposed design change in order for the design to reduce delays in data path.

Chapter 7 draws the concluding remarks for this thesis and points out some future research and implementation direction.

Chapter 2

Mathematical Background

2.1 Background

2.1.1 Field

A field is an algebraic structure that the operation of Multiplication, addition, subtraction and subtraction can be performed and satisfy the usual rules. More accurately,*field* is a set of F with two binary operations + (addition) and · (Multiplication) that hold the following laws

- $a + (b + c) = (a + b) + c$ (Addition Associative Law)
- $a + b = b + a$ (Addition Commutative Law)
- There is an element 0 such that $a + 0 = 0$ for all a.
- For any a, there is an element -a such that $a + (-a) = 0$.
- $a.(b.c) = (a.b).c$(Multiplication Associative Law).
- There is an element 1 (not equal to 0) such that$a.1 = a$ for all a.
- For any $a \neq 0$, there is an element a^{-1} such that $a.a^{-1} = 1$.
- $a.(b + c) = (a.b) + c$ (Distributive Law).

The above laws cannot be satisfied for all possible field sizes. Finite field or Galois filed is a field that contains finite number of the elements. A Galois field in which the elements can take q different values is referred to as GF(q). Such a field exists, if $q = p^m$ for some prime p and integer m.

2.1.2 Prim Finite Field

The rules for a Galois field with a prime number (p)of elements can be satisfied by carrying out the modulo (p). If two numbers in the range of 0 to $(p)-1$ be considered and either add or multiply them, then it should take the results module (p). For instance the below operations shows modulo 2 multiplication and addition for GF(2). Here $p=2$.

Modulo 2 Addition	Modulo 2 Multiplication
$0+0=0$	$0.0=0$
$0+1=1$	$0.1=0$
$1+0=1$	$1.0=0$
$1+1=0$	$1.1=1$

2.1.3 Extension Field GF(p)

The extension field can be constructed by using polynomial modular arithmetic. In this form, the field elements are represented by the polynomials over GF(p) of degree less than m. The field operations are defined as polynomial addition and multiplication modulo a degree m irreducible polynomial over GF(p). The construction of extension fields using polynomials over prime field is possible due to the fact that the extension field GF(p^m) is a m dimensional vector space over the prime field GF(p). As a result a basis of $\alpha_0, \alpha_1, ..., \alpha_m - 1$ is always exists in GF(p^m) such as each element $a \in$ GF(p^m) can be given by $\{a = a_0\alpha_0, a_1\alpha_1, ..., a_{m-1}\alpha_{m-1}\}$ for a unique set of $a_i \in$ GF(p)

2.1.4 Binary Extension Field GF(2^m)

Binary extension field is a special case of GF(p^m) and elements are represented as polynomials with coefficients in a finite field GF(2^m). The polynomials have maximum degree of *m-1*, so that there are m coefficient in total for every element. for example in the field GF(2^m) each element $A \in$ GF(2^m)is represented as

$$A(x) = a_m x^m + ... + a_1 x + a_0 \quad \text{when } a_i \in GF(2) = \{0, 1\}$$

It is also important to realize that every polynomial can be stored in digital form as an $m$bit vector as shown below.

$$A(x) = (a_m + a_{m-1} + ... + a_1 + a_0)$$

2.2 Algorithm

As was explained earlier in this chapter a finite field over GF(2^m) can be defined as a following equation using irreducible polynomial

$$Q(x) = x^m + q_{m-1}x^{m-1} + \ + q_2x^2 + q_1x + 1 \tag{2.1}$$

where $q_i \in GF(2)$ for $0 < i < m$. The polynomial basis $\{\alpha_0, \alpha_1, ..., \alpha^{m-1}\}$ is used to represent the finite filed where α is a root of $Q(x)$.
Assume two field elements A and B represented by polynomials

$$A = \sum_{i=0}^{m-1} a_i\alpha^i \qquad \text{and} \tag{2.2}$$

$$B = \sum_{i=0}^{m-1} b_j\alpha^j \tag{2.3}$$

where a_i and $b_j \in$ GF(2) for $0 \leq i, j < m$. The product of two elements A and be B over GF(2^m) is given by

$$C = A.B \mod Q(\alpha) \tag{2.4}$$

Equation (2.4) can be expanded and represented by the polynomial sum of

$$C = \sum_{i=0}^{m-1} b_i[\alpha^i A \bmod Q(\alpha)] \tag{2.5}$$

$$C = \sum_{i=0}^{m-1} a_i[\alpha^i B \bmod Q(\alpha)] \tag{2.6}$$

It can be seen that (2.5) and (2.6) are very similar. Hence, it is only needed to investigate the systolic implementation of one the equations. Approach indicated by

equation (2.5) was chosen to be investigated by the author [2] for this purpose. It can be noted from equation (2.5) that each partial product is composed of only m terms and the partial product C^i is given by

$$C^i = b_i \alpha^i A \bmod Q(\alpha) \tag{2.7}$$

The main problem with the equations (2.5) and (2.6) and (2.7) is that the reduction operations are done in one step. Next Chapter will discuss the techniques which overcome this problem by performing iterative reductions such that the degree of the polynomial is reduced by one at each iteration.

2.3 Chapter Summary

This chapter provides some mathematical background on Field, Prime field, Extension Field and the Galois field. It latter discusses the algorithm for Progressive Product Reduction (PPR) which is explained completely in the next chapter.

Chapter 3

Progressive Product Reduction (PPR)

Since the addition operation is associative,the Progressive Product Reduction can evaluate equation (2.6) which will be discussed in this chapter. This method iteratively calculates $b_i\alpha^i A \bmod Q(\alpha)$ for decreasing values of the index i.

3.1 Progressive Product Reduction (PPR) Method

This method converts equation equation (2.6) into an iteration using decreasing powers of the summation index i. The iterations are described by

$$c^m = 0 \tag{3.1}$$

$$c^i = b_i\alpha^i A + [c^{i+1} \mod \alpha^i Q(\alpha)] \quad 0 \le i < m \tag{3.2}$$

$$C = c^0 \tag{3.3}$$

It can be seen from (3.2) that the polynomial c^{i+1} of degree $i+1$ is reduced to another polynomial of degree of i. The following theorem shows how this can be done.
Theorem: Assume an m-term polynomial with degree $l+m-1$ of a form

$$p(\alpha) = \alpha^l \sum_{i=0}^{m-1} q_i\alpha^i \tag{3.4}$$

The order of this polynomial can be reduced by one using the reduction polynomial $\alpha^{l-1}Q(\alpha)$, where $Q(\alpha)$ is defined in equation $Q(x) = x^m + q_{m-1}x^{m-1}, ..., q_2x^2 + q_1x + 1$.

Proof: $P(\alpha)$ in (3.4) can be written in the form of

$$p(\alpha) = \alpha^{l-1} \sum_{i=0}^{m-1} q_i \alpha^{i+1} \tag{3.5}$$

α^m is used in equation (3.6) to get the modulo operation

$$p(\alpha) \mod \alpha^{l-1} Q(\alpha) = q_{m-1}(\alpha^k + 1) + \alpha^{l-1} \sum_{i=1}^{m-1} q_{i-1} \alpha^i \tag{3.6}$$

Thus the input polynomial $P(\alpha)$ has now been reduced by one as required.
From(3.1)-(3.3) and (3.6) the bit level for iterations for PPR algorithm can be obtained as

$$\begin{aligned} c_j^m &= 0 && (3.7)\\ c_j^i &= c_{j-1}^{i+1} + b_i a_j \quad j \neq 0, k && (3.8)\\ c_j^i &= c_{j-1}^{i+1} + b_i a_j \quad j = 0, k && (3.9)\\ C_j &= c_j^0 && (3.10) \end{aligned}$$

3.1.1 Parallelizing The PPR Technique

The algorithm has two input variables (A and B), one intermediate variable c_i^j , and one output variable C. Figure 3.1 shows the dependence graph for PPR algorithm.In the Graph, input bits A_j are shown as the vertical lines and input bits b_i are shown as the horizontal lines. The intermediate variables c_j^i are shown by the diagonal lines and the output variable C is obtained at the top row as shown.

3.1.2 Scheduling Function Design for PPR Technique

The analysis starts with a simple affine scheduling function such that point $\mathbf{p}= [i \;\; j]^t$ is associated with the time value at point $[i\ j]$

$$t(p) = sp - \gamma = ia + jB - \gamma \tag{3.11}$$

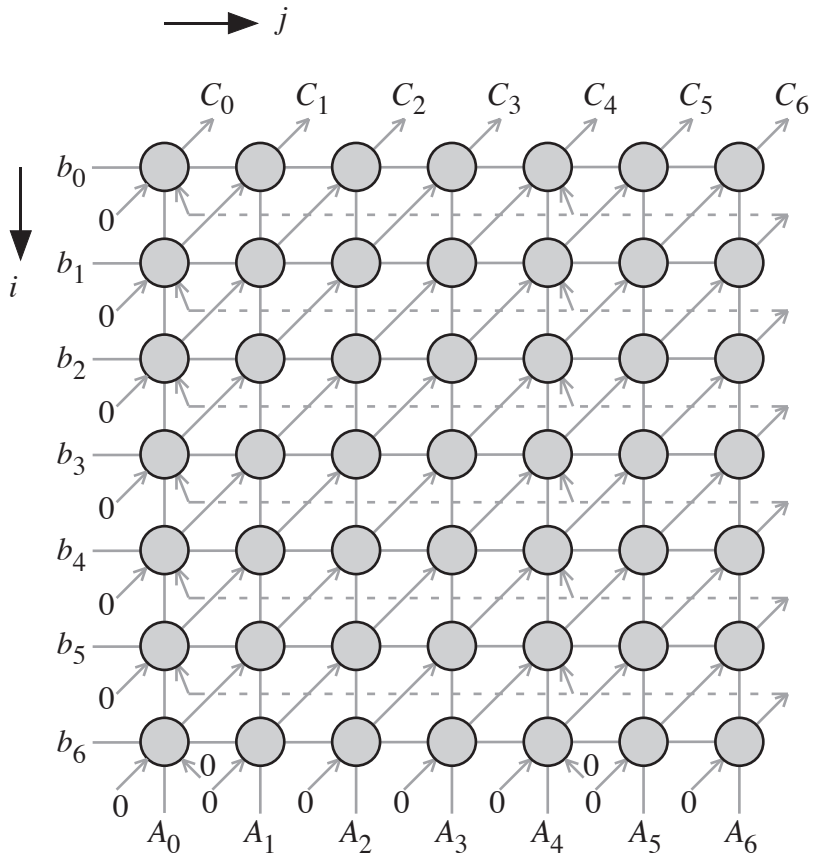

Figure 3.1: Dependence graph for PPR algorithm

There are two restrictions in choice of s based on the data flow in Figure 3.1.

First, the iterative calculation of c_j^i in (3.8) at point $[i, j]$ must be executed after the task at point $[i+1, j-1]$ is completed. this can be written as

$$[\alpha \quad \beta][i \;\; j]^t > [\alpha \;\; \beta][i+1 \quad j-1]^t \tag{3.12}$$

which results in

$$\alpha < \beta \tag{3.13}$$

Second restriction on timing stem from (3.9) is that point $[i, 0]$ can only proceed after point $[i+1,\ m-1]$ has been evaluated

$$[\alpha \quad \beta][i \;\; 0]^t > [\alpha \;\; \beta][i+1 \quad m-1]^t \tag{3.14}$$

which results in

$$\alpha < -(m-1)\beta \tag{3.15}$$

Based on (3.1) and (3.2) there is a simple scheduling vectors that is listed in first column of Table 3.1.

Figure 3.2 shows node timing for PPR algorithm using the scheduling functions s_1 for $m = 7$ and $k = 4$. The Grey boxes indicated the nodes that execute at the same time. The numbers on the right of the figure indicate the times. The PPR technique will require $m$time steps to complete.

Table 3.1: **Scheduling and associated projection vectors for PPR**

Scheduling Vectors	**Projection Directions**
$s_1 = [\,-1 \;\; 0\,], \quad \gamma_1 = -m+1$	$d_{11} = [\,-1 \;\; 0\,]^t, \quad \delta_{11} = 0$

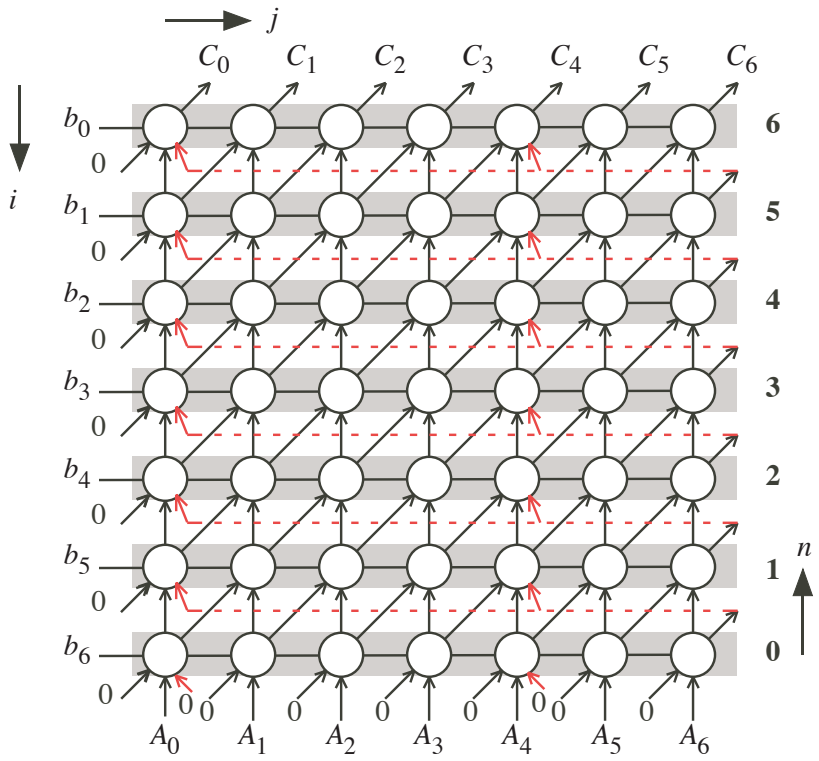

Figure 3.2: Node Timing for the PPR algorithm using the scheduling functions s_1 for $m = 7$ and $k = 4$

3.1.3 Projection Function Design for PPR method

In Section 3.1.2 we discussed how we can associate a time index to each point (task) in the dependence graph. In this section we discuss how we can assign a processor to each node in the dependence graph. We can use the techniques proposed in [14] to derive linear affine task projection. Assume two points in DAG lie along the projection direction **d** such that

$$\mathbf{p}_2 = \mathbf{p}_1 + e\mathbf{d} \tag{3.16}$$

Where e is some constant. These two points will be mapped to the same processor if we make that projection direction the nullvector of a projection matrix **p**[38]. Typically we should ensure that $\mathbf{ds} \neq 0$. So a choice for a scheduling vector has implications for the choice of projection vectors A point $\mathbf{p} \in DAG$ space will be projected to point **p** in the processor array space using the affine projection operation

$$\overline{\mathbf{p}} = \mathbf{P}\mathbf{p} - \delta \tag{3.17}$$

where **p** is a rank-deficient projection matrix and δ is a scalar constant to adjust the processor indices to start at 0 value. Most of the time, we will be seeking one-dimensional processor arrays to implement an algorithm. In that case **p** will reduce to a row vector such that the product in (3.17) will result in scalar value for the processor index that the point maps to.
Generalization to processor arrays of higher dimensions is beyond the scope of this work. The following subsections illustrate the design space exploration for the PMR technique through the different choices of scheduling functions and projection directions. Reference [14] places one restriction on the projection directions; namely:

$$\mathbf{s} \quad \mathbf{d} \neq 0 \tag{3.18}$$

which ensures that each processor does not do all the workload per clock cycle and also that all processors are well utilized by working at each time step. Table 3.1 shows the simple projection directions that could be associated with each scheduling function. The following section discusses the different designs associated with each choice of **s** and **d**. We note that we used simple linear affine projection directions

and also used complex-looking nonlinear projection directions. This latter choice will result in simple hardware for the processor array where the feedback signal is easily extracted from the processor with the highest index as will be explained in sequel.

3.1.4 Design Space Exploration For PPR Technique [Design 1_11 Using $\mathbf{s}_1, \mathbf{d}_{11}$]

In this case, a point $\mathbf{p} = [i \quad j]^t \in DAG$ will be mapped by the projection matrix $\mathbf{p}_{11} = [0 \quad 1]$ on to the point

$$\overline{\mathbf{p}} = \mathbf{P}_{11}\mathbf{p} - \delta_{11} = j \tag{3.19}$$

Thus all nodes in a column will map to a single PE such that the nodes in a column will execute at a different time steps due to our choice of the scheduling function. Figure 3.3 shows the hardware details for Design number 1_11.

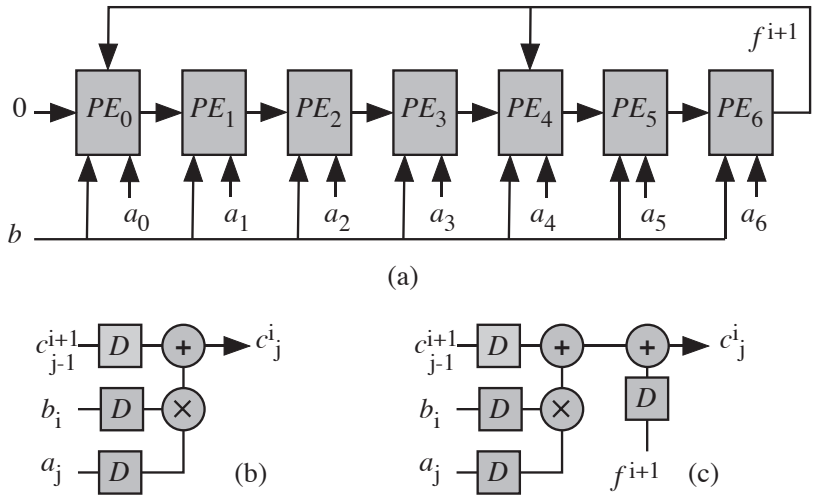

Figure 3.3: Design 1_11

3.2 Chapter Summary

This chapter explained in details the Progressive Product Reduction (PPR) method which the rest of this thesis is focuses on its design and implementation. This section went through the PPR Parallelizing and scheduling function design techniques and explored the PPR technique for $s_1 d_{11}$

Chapter 4

Design overview

General description of the system is defined by processor elements. Figure 3.3(a) shows the general Register Transfer Level (RTL) view of the entire design. This is a coarse view of the system emphasizing the major components of the design. In addition, the diagram shows the I/O requirements as well as the data path through the design. The processing elements perform GF Multiplication and in GF(2^m). The polynomial basis multiplication based on irreducible polynomial over GF(2^m) achieves by pipe-lining the $m-1$ number of the processor elements and routing the signals correctly as explained in previous chapter.

4.1 Processing Elements(PE)

The main components of the design are processing elements. Processor Elements are combination of XOR gates, AND gates and Flip Flops which are combined, based on the functionality of each individual PE. Figures 4.1 and 4.2 show the schematic view of two different PE_j designs. Figure 4.1 shows the case when $j \neq [0 \text{ or } k]$ and Figure 4.2 shows the case when $j = [0 \text{ or } k]$. Note that an extra XOR gate and one Flip Flop is needed to process the feedback signal f^i.

4.2 Systolic Multiplier for GF(2^m)

A systolic systems consists of an array of PE(Processing Elements). Each cell is connected to a small number of nearest neighbors in a mesh like topology. Each cell performs a sequence of operations on data that flows between them. Generally the

operations are the same in each cell but could differ based on the design. Each cell performs an operation or small number of operations on a data item and then pass that to its neighboring cell.

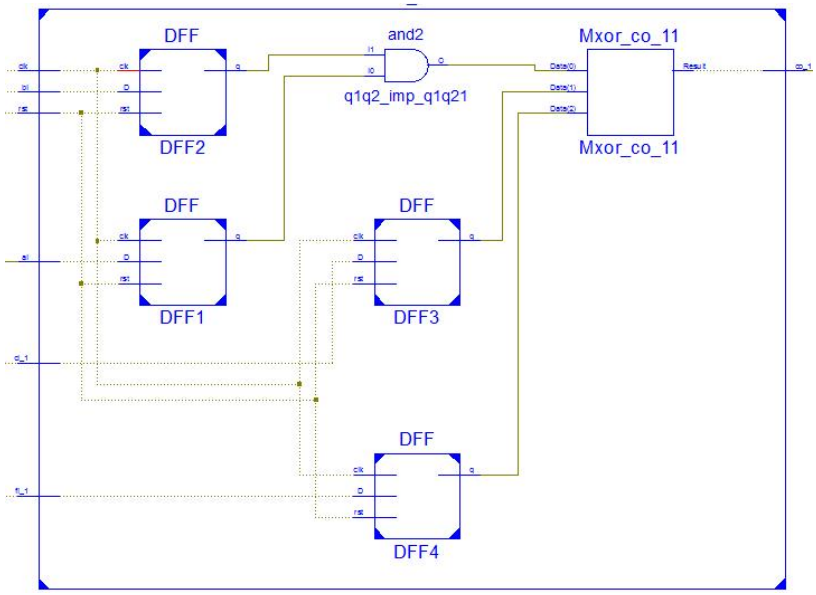

Figure 4.1: Schematic view of Processing Element with Feedback Input

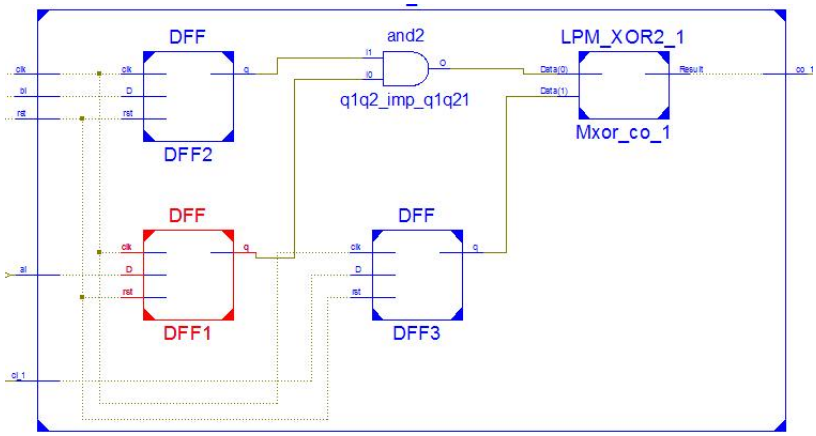

Figure 4.2: Schematic view of Processing Element with no Feedback Input

4.2.1 Systolic Multiplier for 1_11 design

Figure 4.3 depics an example of pipelined parallel systolic architecture for implementation of 1_11 multiplier over GF(2^m), where one of the operands is fed to the structure

in parallel, while the other is fed digit-by-digit to the next PE of the structure. Communication between adjacent PEs requires only a one bit line for transmitting bits c_j^i. The feedback signal $f^i = c_{m-1}^{i+1}$ is obtained from the output of PE_{m-1} at each clock. This signal is fed back to the two processing elements for digit PE_0 or PE_k. The operation of each $\mathrm{PE}_j (0 \leq j < m)$ can be summarized as follow,

- Input multiplicand bit b_i is broadcast to all PEs at iteration i.
- The DFF_2 in Figure 4.1 and Figure 4.2 is responsible for accumulating output bits $c_j (0 \leq j < m)$. the accumulator is cleared at time $n = 0$ which accomplished using reset signal.
- The feedback signal $f^i = c_{m-1}^{i+1}$ is obtained from the output of PE_{m-1} at time n.This signal is fed back to the two processors PE_0 and PE_k.

Clock and Reset are common among all the Processing Elements and the last Processing Element produces the result. The combination of the processing Elements performs Galois Field Multiplication base on the Progressive Product Reduction technique. The calculation steps for each processing elements is explained in section 3.1.

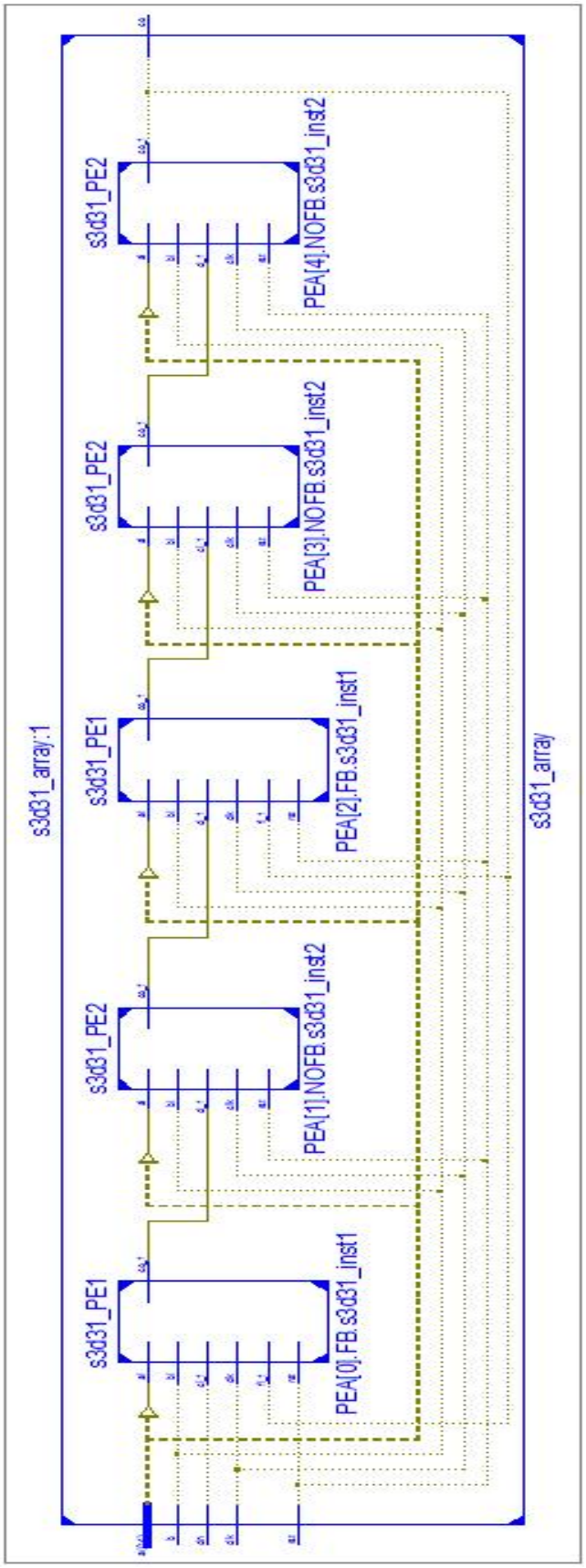

Figure 4.3: Schematic view of Processing Element Array.

Chapter 5

Implementation, Results and Analysis

The purpose of this chapter is to prove the PPR design in previous chapter will work base on algorithm discussed in chapter 3 .Throughout the implementation phase a respectable amount of testing and calculation were done to ensure proper function and performance of the system. This chapter documents some of the tests and its results which were carried out to ensure the proper functionality of the design. Initially a relatively small version over GF(2^m) was constructed in order to prove that the design would create the correct result and later a prototype of the architecture has been implemented for the field GF(2^{233}) and GF(2^{133}) based on NIST recommended field polynomial. The three designs get compared based on their Critical path, Power consumption and Device Utilization.

5.1 Verification Using MATLAB

To verify the hardware, a software implementation was made using Matlab.The purpose was to verify the result sin Galois field (2^m) multiplications prior to any practical test and compare the with hardware result which then can confirm the correct functionality of the hardware.
Matab *gfconv* function multiplies polynomials over Galois Filed. Algebraically, multiplying polynomials over a Galois Field is equivalent to convolving vectors containing the polynomials coefficients, where the convolution operation uses arithmetic over the same Galois field. For instance *gfconv*= $(a, b, 2)$ multiplies two GF(2) polynomials of

a and b.
Matlab *gfdeconv* function divides polynomials over a Galois Field. Algebraically, dividing polynomials over a Galois field is equivalent to deconvolving vectors containing the polynomials coefficients, where the deconvolution operation uses arithmetic over the same Galois field. For instance $[quot, remd] = gfdeconv(b, a, 2)$ divides the polynomial b by the polynomial a over GF(2) and returns the quotient in *quot* and the remainder in *remd*.
All the Matlab commands line entered in an M-file which called *main*, which can be executed by typing the file name in the Matlab command line. It's worth mentioning that the operands must entered in order of MSB to LSB for Matlab calculation. In order to create less confusion for the user *fliplr* command was used to enter the operands in order of LSB to MSB. The full Matlab code can be found in appendixA.

5.2 VHDL Implementation

The design is implemented in structural VHDL model. The code synthesized using Xilinx ISE FPGA Compiler 12.3 [11]. Figure 5.3 shows the general structure of VHDL code from which the description of the models was derived. The code is block oriented and in that way uses hierarchy. It instantiates the design block which have sub blocks and the each block has data ports. The ports of the top level design block implemented the inputs and outputs of the algorithm. The Figure shows that the architecture consists of a declaration part and an implementation part. The entity covers the port declarations whereas the architecture part of the code implements the computation part. The latter part contain, sub block instantiations and concurrent statements. They are executed in parallel, indicated by grayed boxes in the Figure 5.3. The code uses generics in order to make the code reusable in term of values of m and k. Generic also helps in optimizing unnecessary logic or modifying useful logic when it comes to synthesis. Generate Statements used in the code to instantiate processor elements in the PE array design. the code instantiates $m - 1$ processor elements for different value of m.

As indicated at the beginning of this chapter, initially a small version of GF(2^m) was constructor. For the purpose of the test, two polynomials $a = (0x^4 + 0x^3 + 1x^2 + 1x^1 + 1x^0)$ and $b = (0x^4 + 1x^3 + 0x^2 + 0x^1 + 0x^0)$ were considered as operands.
The operand values were entered in the test bench in order to evaluate the functionality of the design. Both VHDL and Matlab result in the same output which confirms

the correct functionality of the hardware.

Xilinx Plan Ahead [10] tool is used to observe the output C . Both systolic array structure and the resultant waveform can be seen in Figure 5.1 and Figure 5.2.

The proposed Multiplier circuit operates as follows. At clock zero all the registers initialized to zero, During every clock cycle the input ports should read $ab_j + c_j$ for $j = \{1, ..., m_1\}$ for $(j \neq 0, k)$ with exception of clock cycle zero and k which the inputs read $ab_j + c_j + f$ for $j = (0, k)$. The output $C = AB$ mod (P) will be generated at $m - 1$ clock cycle.

Figure 5.1: Resultant wave from for systolic array, $m = 5$ and $k = 2$

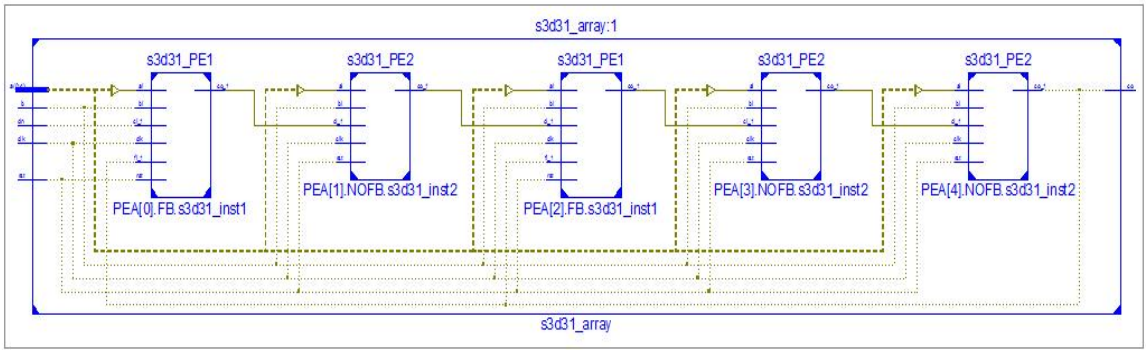

Figure 5.2: Systolic array structure for $m = 5$ and $k = 2$

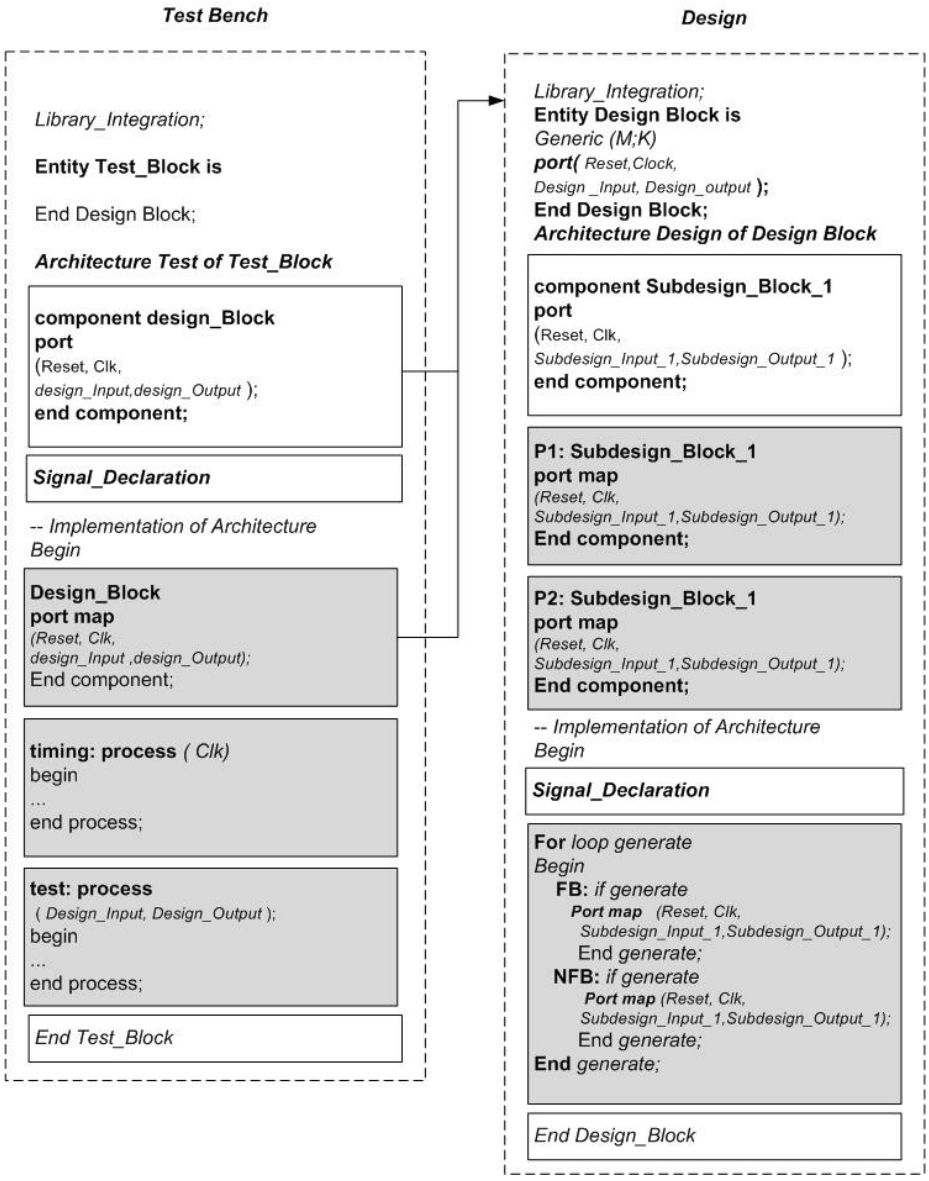

Figure 5.3: General VHDL program structure

5.3 VHDL Implementation and analysis of GF(2^5), GF(2^{133}) and GF(2^{233})

This section studied the implementation of three different trinomial and compare their results based on their Power consumption, Device utilization and Data path delay. The trinomials used for the purpose of these tests are $Q(z) = z^5 + z^2 + 1$, $Q(z) = z^{133} + z^9 + 1$ and $Q(z) = z^{233} + z^{74} + 1$.
The value of m and k can be easily be altered by changing the generic values in the VHDL code, as code is written in away to be reusable in terms of m and k. The hardware device choose for the implementation is Xilinx Virtex 4-8F363. Virtex4 contains 6144 slices, 12,288 LUTs and 12,288 Flip Flops. There are 4 Slices, 8 LUTs and 8 Flip Flops in each slices. Besides their use logic components, slices are also used for routing signals within the device.
The following tests study and compare the three irreducible trinomials in term of Power consumption, data Path Dealy and Device utilization using Xilins ISE tool.

5.3.1 Data Path Delay (Critical Path) Analysis

In this test the Data Path Delay of three mentioned trinomials studied and compared based on different value of the m. Based on Xilinx recommendation the Hierarchy option in the ISE synthesis tool should be keep in order to achieve the best result for the data path delay or the critical path. A $5ns$ timing constraint is also introduced to the clock frequency using User Constraints File (UCF). The Data Path Delay results for three different design can be seen in the Table 5.1. Figure 5.4 shows the schematic view of the Data Path Delay for $m = 5$ and Figure 5.5 shows the schematic view of the Data Path Delay for $m = 113$, while Figure 5.6 shows the schematic view of the Data Path Delay for $m = 233$.

Table 5.1: **Maximum Data Path Delay**

m	5	113	233
Max Data Path Delay (ns)	1.44	2.64	3.26

The expectation was the same Data path delay routing for all three different values of m but it can be observed from the schematic views that the Data Paths are different depending on the value of the m. The Data Path Delay for $m = 5$ is

identified between the two processing elements, PE_1 and PE_2 while the Data Path Delay for $m = 233$ is between the two processing elements PE_{232} and PE_{74} and finally the Data Path Delay for $m = 113$ is identified between two processing element PE_{15} and PE_{16}.

5.3.2 Power Analysis

The source of power consumption can be categories into two, dynamic and static.The dynamic power consists of power consumed to switch the signals, where as static power is the power consumed to hold by the signal. Majority of the power consumption are dynamic which means unless there is no activity in the design, then the static power consumption can be ignored. Dynamic power consumption can be divided in few categories. One is the clock, that will consume power every time that clock signal switches and as a result the higher clock activity will result in increase of the power consumption.

In this section the three design implementations for different value of m are compare in terms of power consumption. The power results obtained from power data generator in Xilinx ISE tool. The tool reports all the on chip power consumption by clocks, gates, IOs, signals and logic devices. The total power consumption for all three designs are depicted in Figure 5.7. The power consumption for each design is also shown on the graph when the clock speed is increased for Critical path analysis. The Figure indicates that power consumption increase as the number of processing elements increase. It means that there is small power consumption for a small amount of logic used. It is also shown that the amount power consumption increases with as clock frequency increase.The Xilinx report shows that the clock consumes the majority amount of power on the chip and the much lesser amount is consumed by the logics, signals or the IOs. Tables 5.2 and Tables 5.3 shows the on chip power consumptions for different value of m. Table 5.2 shows the results when the design is flatten and Table 5.3 shows the results when the Hierarchy is kept for the design.

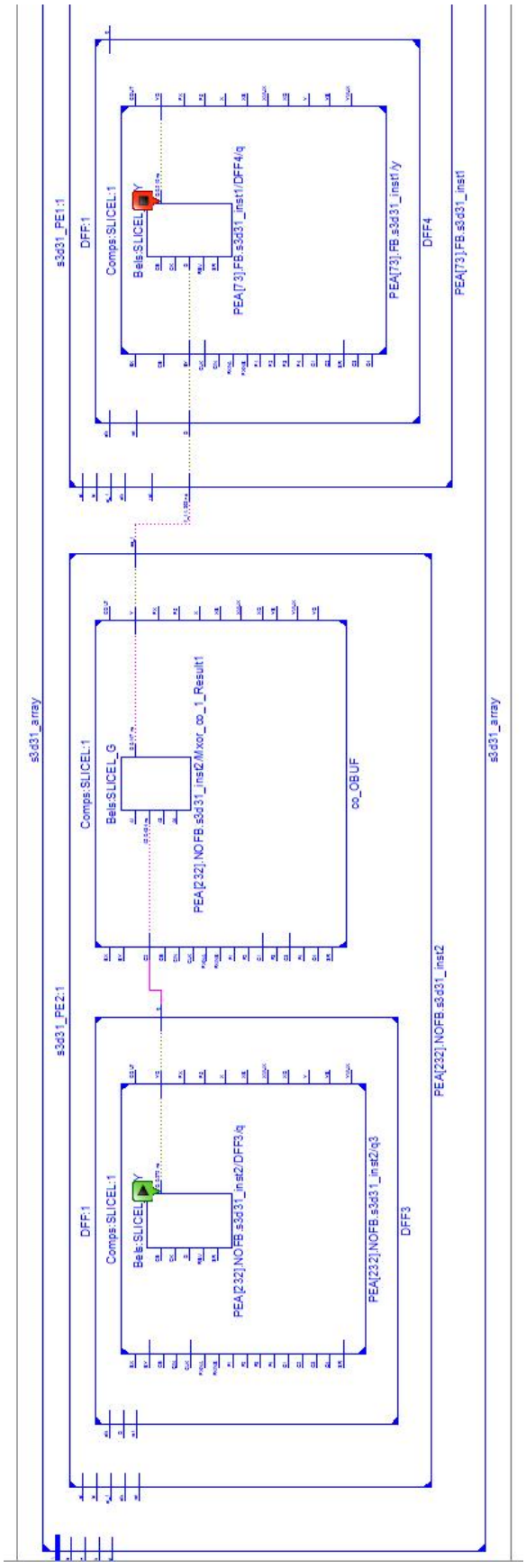

Figure 5.4: Maximum Data Path Delay $m = 5, k = 2$

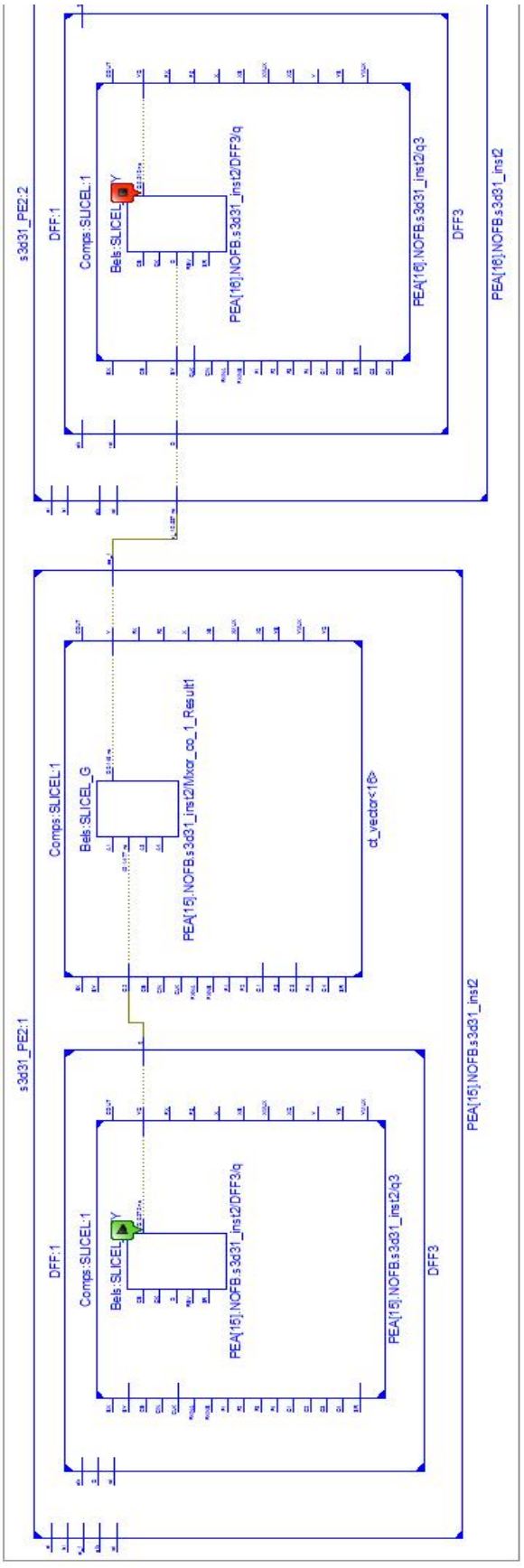

Figure 5.5: Maximum Data Path Delay $m = 113, k = 9$

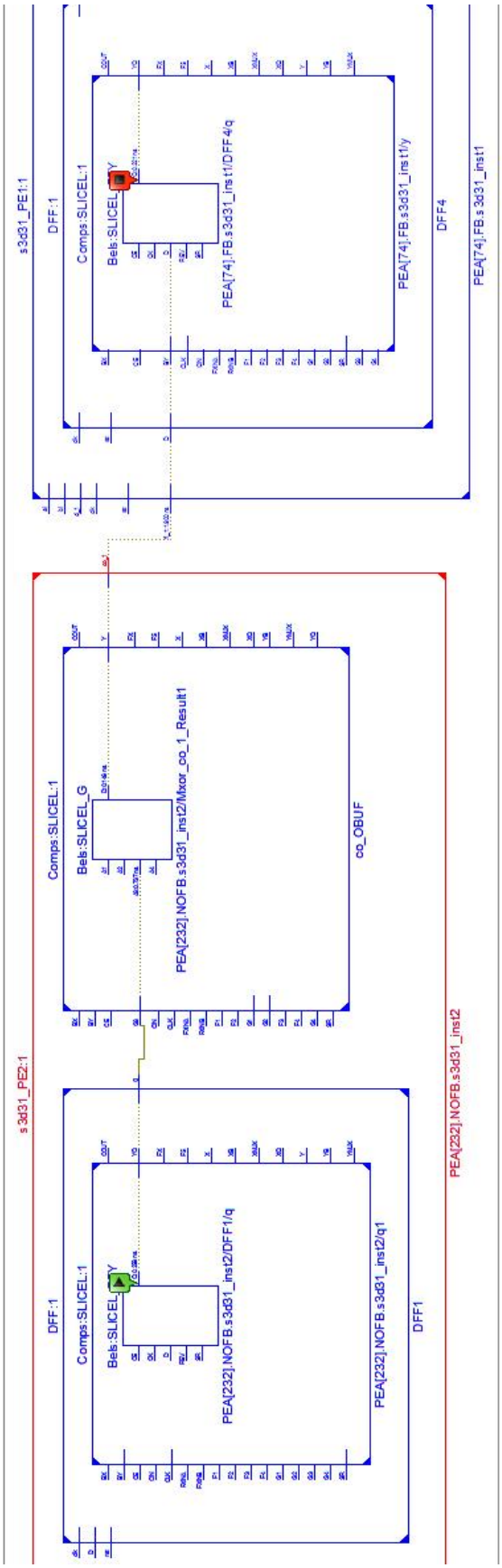

Figure 5.6: Maximum Data Path Delay $m = 233, k = 74$

Table 5.2: **On chip power consumption**

m	5	113	233
Clocks (mW)	5.04	17.41	34.22
Logic (mW)	0.03	0.63	1.29
Signals (mW)	0.07	1.19	2.66
IOs (mW)	0.82	1.39	2.08

Table 5.3: **On chip power consumption with clock constraint**

m	5	113	233
Clocks (mW)	7.81	46.16	66.94
Logic (mW)	0.13	2.41	4.73
Signals (mW)	0.40	7.35	13.67
IOs (mW)	3.29	5.57	8.31

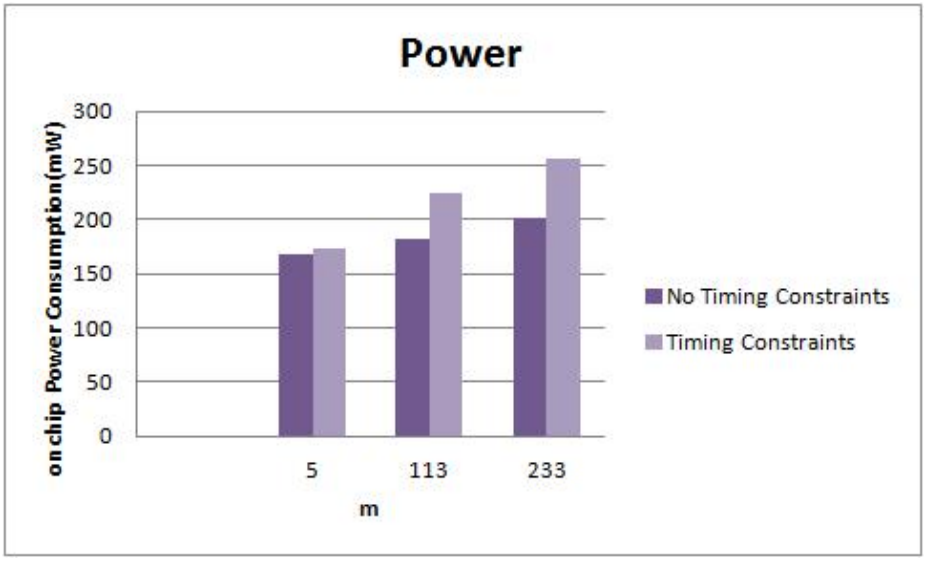

Figure 5.7: On chip power consumption for different value of m

5.3.3 Device Utilization

This section studies the device utilization on Virtex4 FPGA for different designs in terms of value of m. After the designs were compiled , the report option of Xilinx synthesis tool determines preliminary device utilization and performance. The device utilization results for all three designs are depicted in Figures 5.8- 5.10. The device utilization for each design is also shown on the graph when the clock speed is increased for Critical path analysis. The Figures are indicating that number of devices used on the chip increases as the number of processing elements increase. Tables 5.4 and Tables 5.5 shows the device utilization result for different value of m. Table 5.4 shows the results with no timing constraint and Table 5.5 shows the results when the the timing constraint applied to the design. It can also result from the obtained information that only small area of device is used for the largest value of m.

Table 5.4: **On chip Utilized devices**

m	5	113	233
Slices	7(0%)	127(2%)	260(4%)
Flip Flops	12(0%)	234(1%)	480(3%)
LUT-4	5(0%)	113(0%)	233(1%)
IOs	10	118	236

Table 5.5: **On chip utilized with clock constraint**

m	5	113	233
Slices	7(0%)	127(2%)	260(4%)
FlipFlops	12(0%)	234(1%)	480(3%)
LUT-4	5(0%)	113(0%)	233(1%)
IOs	10	118	236

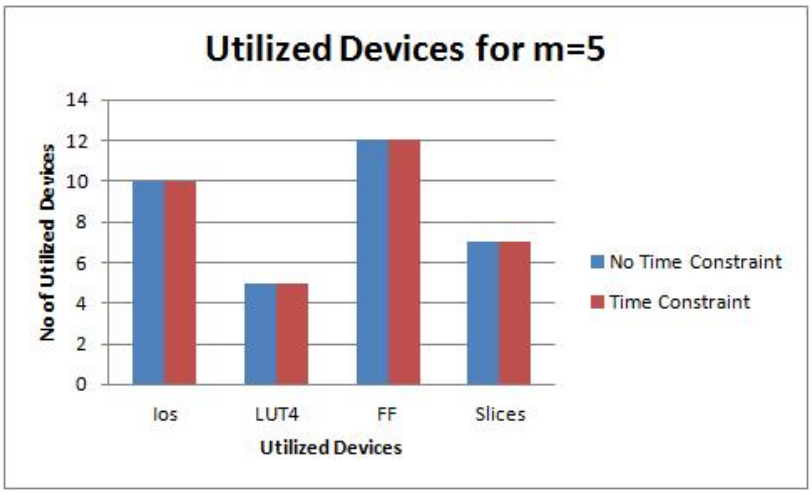

Figure 5.8: Number of on chip utilized devices for $m = 5$

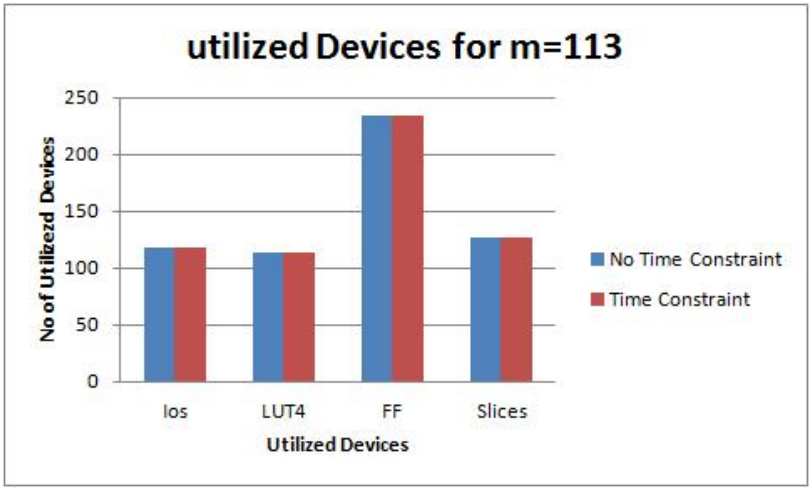

Figure 5.9: Number of on chip utilized devices for $m = 113$

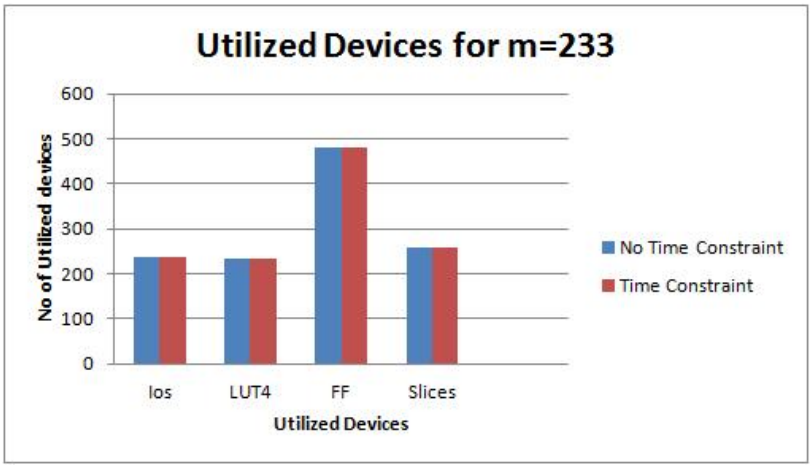

Figure 5.10: Number of on chip utilized devices for $m = 233$

5.4 Chapter Summary

In this chapter Initially a Matlab code was developed for a small Galois field Binary multiplication GF(2^5) in order to verify its result with the result obtained from the VHDL code. The result can be then used to confirm the correct functionality of VHDL design. Latter three different designs were implemented based on different value of the m and their results were compared in terms of power consumption, number of on chip utilized devices and Maximum Data path delay. It was thought that feedback path would be the maximum data path delay in all three designs. But it was found that is only true in case of GF(2^{233}) and is not the case for the other two designs. It assumes that the tool determines the maximum data path delay based on best optimized performance for the device and not just the longest path for the smaller values of m. Table 5.3 shows that the amount of power consumption has increased with the clock frequency increase. The study of the power consumption by the on chip resources also shows that the clock consumes more power compare to the others. This higher power consumption by the clock is due to the higher switching circuit by the clock and is also due to the maximum routing connection which the clock holds on the FPGA device.

Chapter 6

Optimization

This chapter observe the impact of the speed optimization and study the results. There are two optimization methods are proposed in this chapter, Tool optimization and new design optimization methods.

6.1 Speed optimization using Xilinx XST Tool

This section using Xilinx speed strategies in order to observe any speed improvements in the design.These optimization's apply globally to the entire design. Prior to optimization it was ensured that the Hierarchy was flatten and the Xilinx XST reads the EDIF and NCF files to ensure the better results will be obtained.

- Optimization Effort: This is set to Normal by default but the optimization effort can set to high in synthesis process properties. This could improve the speed by only 5 percent.
- Register Balancing: Register Balancing improve the speed at cost of the increasing the area. The option register balancing was set to Yes in the process properties for this test. Register Balancing Moves registers through combinational logic to evenly distribute the path's delay between registers. This is also referred to as flip-flop retiming.
- Optimize Instantiated Primitives: This option will instantiate premitives as necessary.

Although there were some speed optimization was expected from tool but the results from the above test showed no improvements in term of speed when compared with Normal speed (default setting) analysis.

6.2 Proposed optimization method

the maximum clock frequency is determine with the PE which requires feedback signal and architecture of the PE would affect maximum speed.The processing elements with the feedback was initially designed as shown in Figure 6.1.

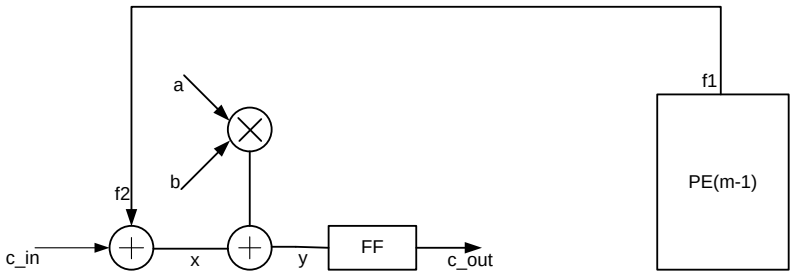

Figure 6.1: Critical path slow design

It can be seen from the design an the resultant waveform (Figure 6.2) that

$$\tau_x = \tau_{XOr} + \tau_{f_2} \tag{6.1}$$

$$\tau_y = 2\tau_{XOR} + \tau_{f_2}. \tag{6.2}$$

From the design it can be seen that the data has to wait $2\tau_{XOR} + \tau_{f_2}$ before producing the output. This design can change and optimize to Figure 6.3 and can result from the design that

$$\tau_{a,b} = \tau_d \qquad (d = \text{delay}) \tag{6.3}$$

$$\tau_x = \tau_{a,b} + \tau_{XOR} + \tau_{AND} \tag{6.4}$$

$$\tau_y = \tau_{XOR} + \tau_{f_2} \tag{6.5}$$

The proposed optimization calculates the ($a\ AND\ b$) prior to the arrival of the

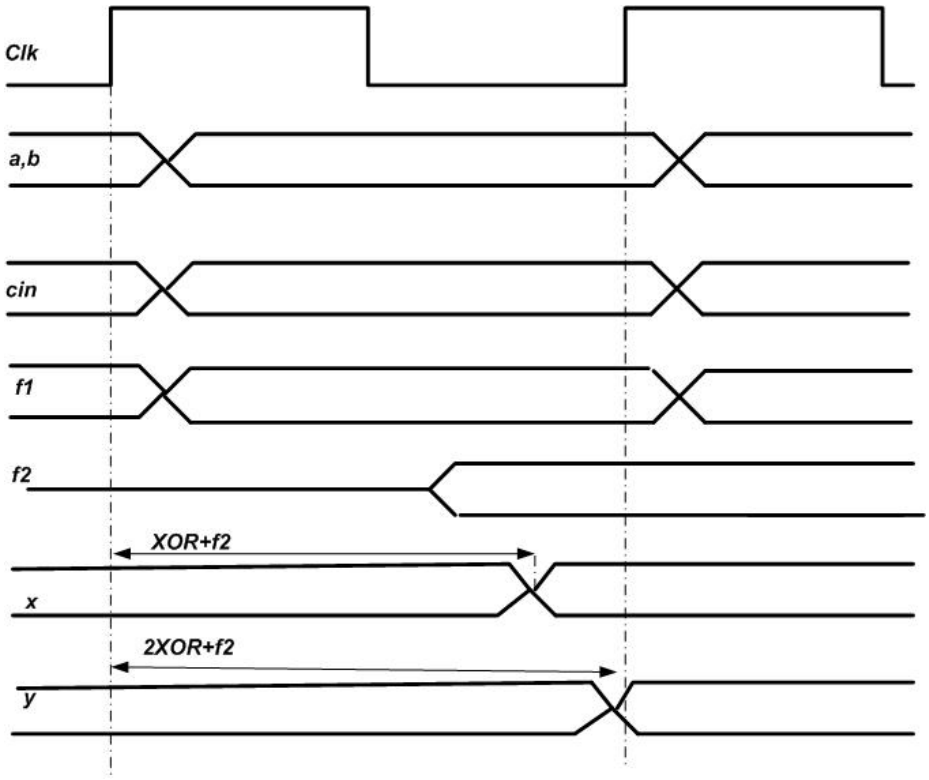

Figure 6.2: Waveform based on Figure 6.1

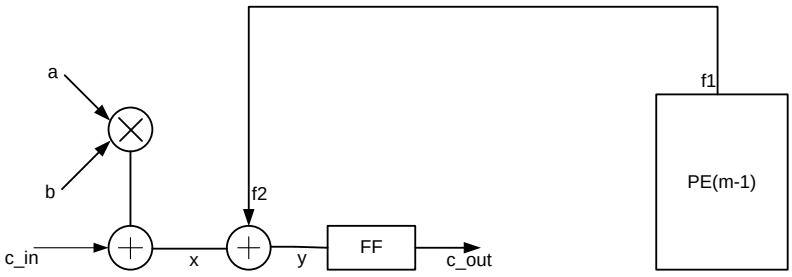

Figure 6.3: Optimized design for Figure 6.1

(f_2) which will reduce the output delay by XOR. The output waveform for the optimize design can be seen in Figure 6.4.

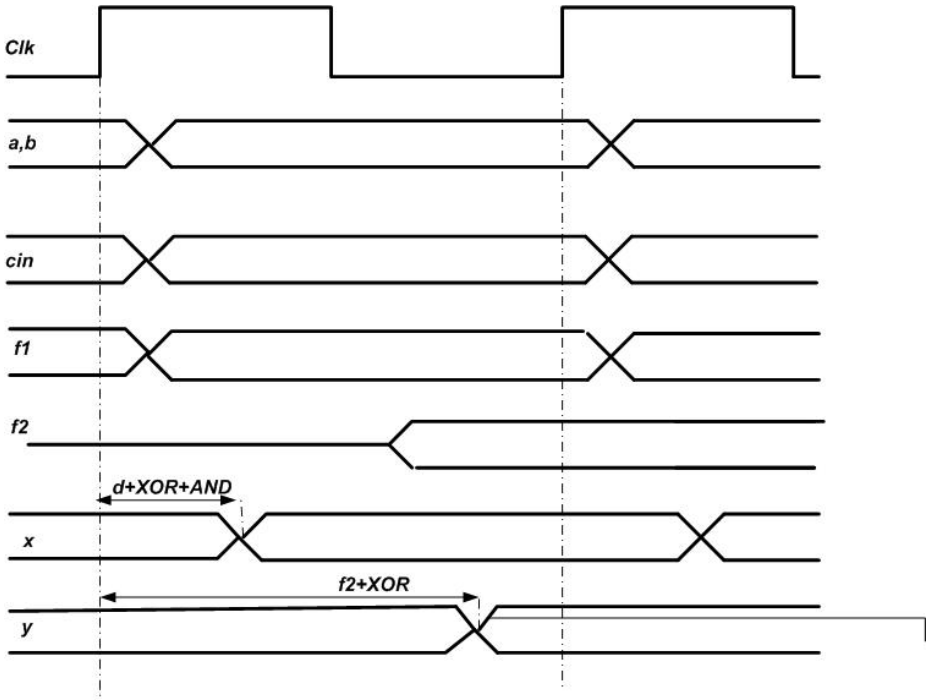

Figure 6.4: Waveform for design in Figure 6.3

6.3 Chapter summary

This section discusses the optimization methods which applied to the design with the highest amount of processing elements or when ($m = 232$). The optimization was only considered for speed optimization. The results obtained from the tool optimization method showed no difference at all compare with when no optimized option was set for the design. There is also a proposed optimization method in this chapter which discusses the changes in the design. This change can result with the better performance of the design in terms of speed.

Chapter 7

Conclusions and Contributions

This thesis considered structural VHDL coding approach for implementation of Systolic Array Progressive Product Reduction(PPR) Method with a goal to reduce the hardware complexity and achieve high speed implementations.The VHDL design proved the correct functionality of the 1_11 design with confirmed verification in Matlab.

Three different analysis (Critical Path delay, device Utilization and Power consumption) were studied on this thesis base on different value of m for GF(2^m). It was shown, there is increase in term of power consumption, Data path delay and Device utilization with increase number of processing elements. the experiment shown that majority of power consumed on the chip was by the clocks and only negligible amount of power used by the logics and IOs. The critical path was the longest path of the design (Feedback path) for $m = 233$ but for the smaller value of m, it seems that the compiler decide on the path base best optimized performance for the device. Routing is an important step of the process as most of the FPGAs area is devoted to the interconnect, and the interconnection delays are greater than the logic delays of the designed circuit. Therefore an efficient routing algorithm can reduce the total wiring area and the lengths of critical path nets to improve the performance of the circuit. The future design should also focus on routing in order to reduce the critical path delay. It is worth mentioning that further designs based on PPR is going to be introduced in Gebali, Ibrahim's paper which can also be implemented and compared with the design in this thesis for any future work.

The following contributions are made based on this thesis

- Propose systolic architecture for converting GF(2^m) multiplication into an iter-

ative expressions.

- Apply formal technique for mapping the iteration to obtain processing arrays.
- Design hardware for systolic array obtained.
- Create the test benches to prove and demonstrate the functionality of the hardware.
- Verify the hardware design by developing a Matlab code to confirm the correct functionality of the hardware, defined by VHDL implementation and compare with previously published results.

Bibliography

[1] E. Abdel-Raheem. Design and VLSI implementation of multirate filter banks. *Ph.D. dissertation, Dept. of Electrical and Computer Eng., Univ. of Victoria*, 1995.

[2] Fayez Gebali and Atef Ibrahim. Systolic array architectures for polynomial GF(2^m) multiplicand. *not Published*.

[3] Y. T. Hwang and Y. H. Hu. MSSM a design aid for multi-stage systolic mapping. *Journal of VLSI Signal Processing*, 4:125 – 145, 1992.

[4] J. M. S'anchez J. L. Imana and F. Tirado. Bit-parallel finite field multipliers for irreducible trinomials. *IEEE Trans. Comput*, 55(5):520–533, May 2006.

[5] N. Kolbitz. Elliptic curve cryptosystem. *Mathematics of Computation*, 48:203–209, 1987.

[6] S. Kung. *VLSI Array Processors.* Englewood Cliffs, N.J.: Prentice-Hall, 1988.

[7] P. Meher. Systolic and super systolic multipliers for finite field GF(2^m) based on irreducible trinomials. *IEEE Transactions on Circuits and Systems*, 55(4):1031–1040, 2008.

[8] K. Parhi. *VLSI Digital Signal Processing Systems Design and Implementation.* John Wiley, 1999.

[9] S. Rao and T. Kailath. Regular iterative algorithms and their implementation on processor arrays. *Proc. IEEE*, 76(3):259–269, Mar 1988.

[10] xilinx. Plan Ahead. `http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/PlanAhead_UserGuide.pdf/`, 2009. [Online; accessed-Jan-2013].

[11] Xilinx. ISE Tool. `http://www.xilinx.com/products/design-tools/ise-design-suite/`, 2013. [Online; accessed-Jan-2013].

[12] T. Zhang and K. K. Parhi. Systematic design of original and modified mastrovito multipliers for general irreducible polynomials. *IEEE Transactions on Computers*, 50(7):734–749, July 2001.

Appendix A

Matlab code

```
1) a= [ 0 0 1 1 1 ]; - - First Operand in binary, the left bit is MSB
2) arev=fliplr(a); - - Flip the digits from LSB to MSB
3) b= [ 0 0 1 1 1 ]; - - Second Operand in binary, the left bit is MSB
4) brev=fliplr(b); - - Flip the digits from LSB to MSB
5) trinom= [ 1 0 1 0 0 1]; - -Trinomial mod in binary
6) trinomrev=fliplr( trinom); - - Flip the digits from LSB to MSB
7) c=gfconv (arev,brev,2); - - multiplications of two operands
8) [quot,remd] = gfdeconv(c,trinomrev,2); - - Division of multiplication
   result by trinomial
```

The above code takes the two binary numbers and arrange them from LSB to MSB. It then multiply the two numbers together using Galois Field multiplication. the result of the multiplication is then divided by a trinomail which is also in binary form and returns the remainder and the quotient.

Appendix B

VHDL code

```
- - File D-FF library ieee;
use IEEE.STD_LOGIC_1164.ALL;
entity DFF is port(D, clk,rst:  in std_logic; q:Out std_logic);
end DFF;
architecture DFF_1 of DFF is Begin
   Process(clk,rst,D) begin
         if rst='1' then q <='0';
elsif rising_edge(clk) then
q <= (d) ;
end if;
end process;
end DFF_1;

library ieee;
use IEEE.STD_LOGIC_1164.ALL;
Entity s3d31_PE1 is
port(ai,bi,clk,rst,ci_1,fi_1:  in std_logic; - - ci_1= ci+1, fi_1=fi+1
co_1:  out std_logic);
end s3d31_PE1;

architecture structural of s3d31_PE1 is
   component DFF is
      port( d,clk,rst:  in std_logic;
```

```
      q:  out std_logic);
   end component;

   Signal q1, q2, q3, q1q2, q1q2q3, y:  std_logic;
begin
   DFF1:  DFF port map (ai, clk, rst, q1);
   DFF2:  DFF port map (bi, clk, rst, q2);
   DFF3:  DFF port map ( ci_1, clk, rst, q3);
   DFF4:  DFF port map ( fi_1, clk, rst, y);
   q1q2< = q1 and q2;
   q1q2q3 < = q3 xor q1q2;
   co_1 < = y xor q1q2q3;
end structural;
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
library ieee;
use IEEE.STD_LOGIC_1164.ALL;
   Entity s3d31_PE2 is
   port(ai,bi,clk,rst,ci_1:  in std_logic;
         co_1:  out std_logic);
   end s3d31_PE2;
architecture structural of s3d31_PE2 is
component DFF is
      port( d,clk,rst:  in std_logic;
         q:  out std_logic);
end component;
   Signal q1:  std_logic;
   signal q2:  std_logic;
   signal q3:  std_logic;
   Signal q1q2:  std_logic;
begin
   DFF1:  DFF port map (ai,clk,rst,q1);
   DFF2:  DFF port map (bi,clk,rst,q2);
   DFF3:  DFF port map ( ci_1,clk,rst,q3);
   q1q2<= q1 and q2;
   co_1<= q3 xor q1q2;
```

```
end structural;
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
library ieee;
use IEEE.STD_LOGIC_1164.ALL;
Entity s3d31_array2 is
Generic (M: INTEGER:=233  INTEGER:=73 );
   port(a:  in std_logic_vector(0 to M-1);
      b,clk,rst :  in std_logic;
      cin:  in std_logic;
      co:  out std_logic);
   end s3d31_array2;
architecture structural of s3d31_array2 is
   component s3d31_PE1 is
      port(ai,bi,clk,rst,ci_1,fi_1 :  in std_logic;
      co_1:  out std_logic);
   end component;
   component s3d31_PE2 is
      port(ai,bi,clk,rst,ci_1 :  in std_logic;
      co_1:  out std_logic);
   end component;
   signal ct_vector:  std_logic_vector(0 to M);
   signal ct_vector_out:  std_logic_vector(0 to M);
begin
ct_vector(0)<= cin;
   PEA : --label name
      for i in 0 to M-1 generate
   begin --"begin" statement for "generate"
   FB: if (i=0 or i=k) generate
      s3d31_inst1 :  s3d31_PE1 port map(a(i),b,clk,rst,ct_vector(i),ct_vector(M),
ct_vector(i+1));
   end generate;
   NOFB: if ((i>=1 and i<k) or i>k ) generate
      s3d31_inst2 :  s3d31_PE2 port map(a(i),b,clk,rst,ct_vector(i),
      ct_vector(i+1));
   end generate;
```

```
   end generate PEA; --end "generate" block
ct_vector_out<= ct_vector;
co <= ct_vector(M);
end architecture;
```

Appendix C

VHDL Test Bench

```
LIBRARY ieee;
ENTITY s3d31_array_tb IS
END s3d31_array_tb;
ARCHITECTURE behavior OF s3d31_arra_tb IS
      - - Component Declaration for the Unit Under Test (UUT)
      COMPONENT s3d31_array
         PORT(
            a :  IN std_logic_vector(4 downto 0);
            b :  IN std_logic;
            clk :  IN std_logic;
            rst :  IN std_logic;
            cin :  IN std_logic;
            co :  OUT std_logic
);
      END COMPONENT;
      - -Inputs
      signal a :  std_logic_vector(4 downto 0) := (others => '0');
      signal b :  std_logic := '0';
      signal clk :  std_logic := '0';
      signal rst :  std_logic := '0';
      signal cin :  std_logic := '0';
      - -Outputs
      signal co :  std_logic;
```

```
   -- Clock period definitions
   constant clk_period :  time := 10 ns;
BEGIN
   - - Instantiate the Unit Under Test (UUT)
   uut:  s3d31_array PORT MAP (
   a => a,
   b => b,
   clk => clk,
   rst => rst,
   cin => cin,
   co => co );
   - - Clock process definitions
begin
   clk <= '0';
   wait for clk_period/2;
   clk <= '1';
   wait for clk_period/2;
end process;
   -- Stimulus process
   stim_proc:  process
begin
   rst <= '1';
rst <= '0';
a<="1110"; -- Left bit is LSB
         --wait for 10 ns;
      b <= '0'; - - MSB
         wait for 10 ns;
      b <= '1';
         wait for 10 ns;
      b <= '0';
         wait for 10 ns;
      b <= '0';
         wait for 10 ns
      b <= '0'; - - LSB
   - - hold reset state for 100 ns.
```

```
      wait for 100 ns;
      wait for clk_period*10;
   -- insert stimulus here
      wait;
      end process;
END;
```