# LOW-POWER AND AREA-EFFICIENT FINITE FIELD ARITHMETIC ARCHITECTURE BASED ON IRREDUCIBLE ALL-ONE POLYNOMIALS

A THESIS SUBMITTED TO
THE BOARD OF GRADUATE PROGRAMS
OF
MIDDLE EAST TECHNICAL UNIVERSITY
NORTHERN CYPRUS CAMPUS

BY

SHIMA MOHAGHEGH

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2020

Approval of the Board of Graduate Programs

Prof. Dr. Gürkan Karakaş
Chairperson

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Murat Fahrioğlu
Program Coordinator

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science .

Prof. Dr. Ali Muhtaroğlu
Supervisor, Electrical and Electronics Engineering Program, METU NCC

Assoc. Prof. Satoshi Kondo
Co-Supervisor, Mathematics Group, METU NCC

**Examining Committee Members**

Prof. Dr. Cüneyt F. Bazlamaçcı CNG / IZTECH

Prof. Dr. Ali Muhtaroğlu EEE/ METU NCC

Assoc. Prof. Dr. Muhammed Salamah CMPE / EMU

## ETHICAL DECLARATION

**I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.**

Name, Surname: Shima Mohaghegh

Signature :

## ABSTRACT

## LOW-POWER AND AREA-EFFICIENT FINITE FIELD ARITHMETIC ARCHITECTURE BASED ON IRREDUCIBLE ALL-ONE POLYNOMIALS

Mohaghegh, Shima
M.S., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Ali Muhtaroğlu
Co-Supervisor : Assoc. Prof. Dr. Satoshi Kondo

September 2020, 121 pages

This thesis presents a low-power and area-efficient finite field multiplier based on irreducible all-one polynomials (AOP). The proposed organization implements the AOP multiplication algorithm in three stages, which are reduction network, AND network (multiplication), and three input XOR tree (accumulation), while state-of-the-art implementations distribute reduction, multiplication and accumulation operations in a systolic array. The optimization reduces the overall number of sequential elements and provides lower pipeline latency compared to literature. This leads to the reduction of power dissipation and area for a system clock frequency. Both the previously reported and the proposed architectures have been implemented in Verilog for three different binary field sizes using TSMC 90 $nm$ standard cell library from Artisan Components, and have been synthesized with a target 1.2 GHz system clock frequency using the Cadence Genus Synthesis tool. The proposed architecture offers 14%, 30%, and 19% reduction in average leakage, dynamic power, and area, respectively, compared to the state-of-the-art. Thus, the proposed architecture is better suited for energy-efficient portable systems, including wireless sensors.

Keywords: Galois field multiplier, synthesis, irreducible, low-power, area-efficient, all-one polynomial (AOP), elliptic curve cryptography (ECC).

# ÖZ

## İNDİRGENEMEZ POLİNOMLARA (AOP) DAYALI DÜŞÜK GÜÇ VE ALAN VERİMLİ SONLU ALAN ÇARPANLARI

Mohaghegh, Shima
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Ali Muhtaroğlu
co Ortak Tez Yöneticisi: Doç. Dr. Satoshi Kondo

Eylül 2020 , 121 sayfa

Bu çalışma, indirgenemez polinomlara (AOP) dayalı düşük güç ve alan verimli sonlu alan çarpanları sunmaktadır. Önerilen organizasyon, AOP çarpma algoritmasının azalma ağı, AND ağı (çarpma) ve üç girişli ağacı XOR (birikim) ağı olmak üzere üç aşamada uygulanırken, literatürdeki uygulamalar azaltma, çarpma ve birikim işlemlerini sistolik bir diziye dağıtır. Optimizasyon, sıralı elemanların toplam sayısını azaltır ve literatüre kıyasla daha düşük boru yolu gecikmesi sağlar. Bu, belirlenmiş bir sistem saat frekansı için güç dağılımının ve alanın azaltılmasına yol açar. Daha önce bildirilen ve önerilen mimariler, Artisan bileşenlerinden TSMC 90 $nm$ standart kütüphanesini kullanılarak, üç farklı alan boyutu için Verilog ile uygulanmış, ve Cadence Genus sentez aracını yürüterek, 1.2 GHz sistem saat frekansı ile sentezlenmiştir. Önerilen organizasyon, literatüre kıyasen sırayla ortalama sızıntı, dinamik güç ve alanda $\%14$, $\%30$ ve $\%19$ azalma sunmaktadır. Böylece, önerilen organizasyon, kablosuz sensörler de dahil olmak üzere, enerji tasarruflu taşınabilir sistemler için daha uygundur.

Anahtar Kelimeler: Galois alan çarpanı, sentez, indirgenemez polinomlar (AOP), düşük güç, alan verimliliği, eliptik eğri kriptografisi.

This thesis is dedicated to my parents for their patience and unquestioned support through easy and difficult times.

# ACKNOWLEDGMENTS

I would like to express my gratitude and appreciation for my supervisor Prof. Dr. Ali Muhtaroğlu, for his dedicated support and guidance throughout this research project. I am incredibly grateful to Assoc. Prof. Dr. Satoshi Kondo for his advice on mathematical approaches. I would also like to thank Assist. Prof. Gürtaç Yemişçioğlu for helping with this research. I would like to thank Prof. Dr. Cüneyt F. Bazlamaçcı for motivating us to work in Finite field circuits. To conclude, I cannot forget to thank my family, especially my brother, for all the unconditional support in these very intense academic years. I also wish to thank my best friends Melina, Solmaz, Negar, Molly and Sana, who have been a great source of support and encouragement in all the difficult times during this project. I would like to express my deep appreciation for the support and help of my colleagues Mahnoor, Haroon, Aboulfazl, Arman and Erfan.

**TABLE OF CONTENTS**

APPENDICES

# LIST OF TABLES

# LIST OF FIGURES

FIGURES

## LIST OF ABBREVIATIONS

| | |
|---|---|
| ALU | Arithmetic Logic Unit |
| AOP | All One Polynomials |
| AES | Advanced Encryption Standard |
| CAD | Computer Aided Design |
| CoCalc | Collaborative Calculation in the Cloud |
| ECC | Elliptic Curve Cryptography |
| EPA | Environmental Protection Agency |
| ESP | Equally Spaced Polynomials |
| GCD | Greatest Common Divisor |
| GF | Galois field |
| GPU | Graphical Processing Unit |
| HDL | Hardware Description Language |
| IC | Integrated Circuit |
| IoS | Internet of Services |
| NAOP | Nearly All One Polynomials |
| NEM | Nano-Electro-Mechanical |
| NIST | National Institute of Standards and Technology |
| RC | Resistance and Capacitance |
| SOC | System On Chip |
| TSMC | Taiwan Semiconductor Manufacturing Company |
| VLSI | Very Large Scale Integrated |
| WSN | Wireless Sensor Networks |
| X-ETP | Cross-European Technology Platforms |

# CHAPTER 1

# INTRODUCTION

## 1.1 Modern Computing and Security Trends

With the continual advances as part of the digital revolution dating back to the eighties, there has been an increased demand for digital services. The emergence of the information economy, driven by the development of computers, the expansion of telecommunications, and the Internet has radically changed the way people do business, deal with the government, receive health and education services, entertain themselves, and hold a conversation with friends and families. Private documents that were previously committed to paper and hand-delivered or stored under lock and key are now produced, submitted, processed, and stored electronically on a routine basis [17].

The Cross-European Technology Platforms (X-ETPs) community sees the Internet of Services (IoS) as a core component of the future Internet (see Figure 1.1). In the future of the IoS, cloud computing will play a prominent part, allowing on-demand provisioning of software, networks, and computing infrastructures. Security, privacy, and reliability should be taken into account in all areas of the future of the Internet. Although cloud service providers utilize numerous processes and technologies to ensure the protection and privacy of data and information, there is significant room for improvement concerning the authentication, authorization, and audit frameworks introduced within the current infrastructure as a service [5].

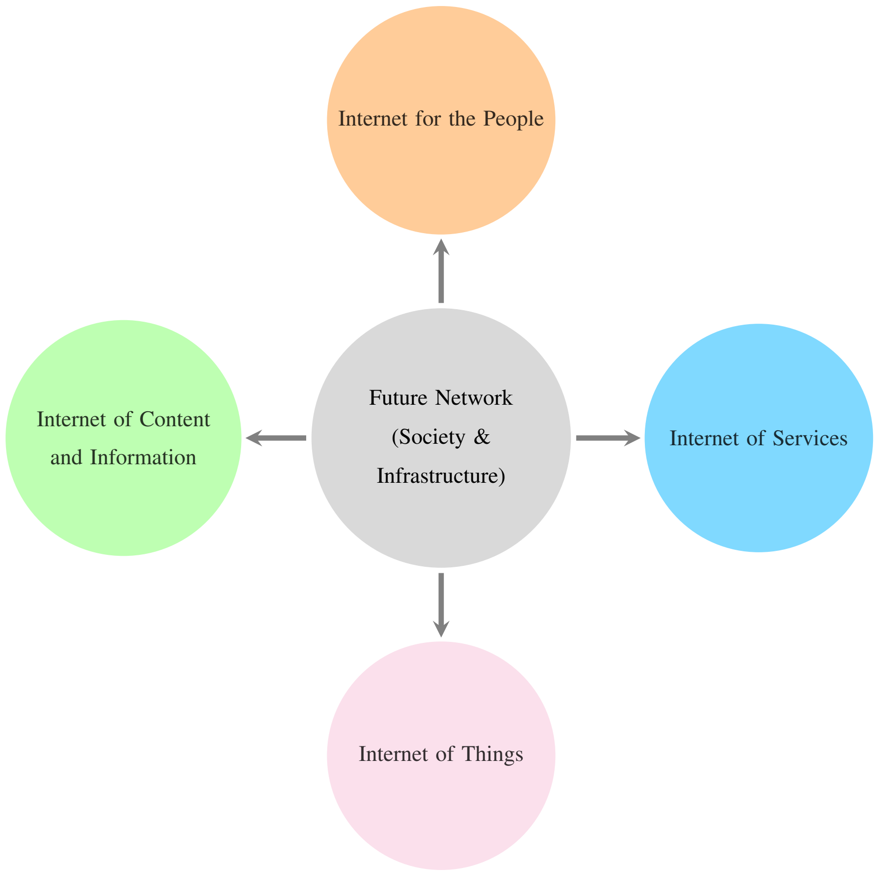

Internet for the People
Internet of Content and Information
Future Network (Society & Infrastructure)
Internet of Services
Internet of Things

**Figure 1.1.** Backbone of Future Internet [5]

Cryptography is a method for information and communication to be secured using codes to be interpreted and processed by those for whom the data is intended. Thus cryptography targets preventing unauthorized access to information. The "crypt" prefix means "hidden," and the "graphy" suffix means "writing" [18].

## 1.2 Use of Finite Field Arithmetic for Cryptography

Finite or Galois field $(GF)$ arithmetic is universally utilized in error-correcting coding systems [19], like Hamming or Reed-Solomon code, and cryptosystems like elliptic curve cryptography (ECC) [20]. ECC based secure communication has recently gained popularity in Wireless Sensor Networks (WSNs) due to the utilization of WSNs in healthcare and other private applications. Since WSN nodes have stringent power and energy consumption requirement, an emerging critical research focus is energy-efficient implementation of ECC systems [21], [22], [23] and [24]. This premise is founded on the fact that all internal operation of Advanced Encryption Standard (ASE) is to be associated with finite fields [25]. Virtually all public-key cryptographic algorithms depend heavily on finite field arithmetic, which is to be effectively performed to satisfy execution speed and design space constraints. Such priorities are significant obstacles that involve interdisciplinary research activities to generate the best algorithms, architectures, implementations, and design practices [26].

Polynomial (or standard), normal, dual, and triangular basis are alternative representations of the elements over the binary field $GF(2^m)$. The polynomial basis is widespread for bit-parallel multipliers due to the fact that it offers the potential for reasonably simple and scalable designs for the fields of the higher-order and does not demand a basis conversion [27]. Multiplication in $GF(2^m)$ is more complicated than addition and subtraction since it involves two steps: polynomial multiplication and reduction modulo of an irreducible polynomial. All-One Polynomials (AOP), Equally Spaced Polynomials (ESP), trinomials, and pentanomials are known as different types of irreducible polynomials [28]. Although the AOP classes of irreducible polynomials offer considerably efficient implementation of finite field multiplication [2], they are not as commonly studied as irreducible trinomials or pentanomials. Thus, the multiplier architecture proposed in this work is based on AOP. AOP multiplication architecture may, in fact, be used as a kernel for field exponentiation, inversion, and division as well as ECC implementation using Nearly AOP (NAOP) [29]. AOPs are relatively simple to implement for finite field generation [30], and are packed as computation cores to be used for trinomial and pentanomials multipliers [3].

## 1.3 Power Dissipation

It is generally believed that the smartphones and computers we use are eco-friendly; however, the fact is a tenth of the electricity used across the world is predominantly consumed by the digital industry resulting in vast consequences for our economy and eco-system [31]. It is globally reported that semiconductor industry sales drastically increased from 21 billion dollars in 1985 to 306 billion dollars in 2013 [32].

Power consumption is one of the critical parameters in integrated circuits customized for embedded and battery-powered applications. The observations regarding power dissipation are particularly relevant for two reasons. Firstly, to the best of found knowledge, the levels of current and power available in a battery are virtually constant. Therefore, the analysis shows that the power dissipation of a circuit or a product determines the battery life. As a result, it can be mentioned that if the power dissipation increases, the battery life will decrease. Another critical issue is that the power dissipation and the heat generated by the microchip or product are not independently variable. Extreme heat dissipation might raise the operating temperature, and this may either increase the risk of gate circuitry drifting out of its regular operating curb bringing the gates to failure in producing appropriate output values. As shown, there are several possible reasons that the power dissipation of any gate implementation must be maintained at the lowest possible value [33].

Recently, the well-known Environmental Protection Agency (EPA) [34] has given an accurate estimation of 1.52 pounds of carbon dioxide per kilowatt-hour (excluding line-losses) [35]. It could be said that carbon dioxide plays a critical role in global warming. On the other hand, critical applications of daily life such as wearable and portable devices, embedded systems, wireless sensor nodes, healthcare, and other private devices are expected to develop at a significant speed and change people's daily routine eminently. Concurrently, privacy and security inherent in the use of these devices gain importance.

All in all, in addition to security and privacy, power and energy dissipation in digital systems have grown equally important. Lower power dissipation in integrated circuits is important to optimize system size and cost in terms of simpler cooling solutions, more compact power electronics components, and smaller batteries [36]. Alternatively, lower power dissipation prolongs battery life forgiven battery capacity. Considering most of the consumer electricity is generated based on fossil fuel consumption, lower power dissipation is also important to improve efficiency and hence the sustainability of the electronic systems.

## 1.4 Motivation and Problem Definition

Security hardware is expected to grow in the near future, along with the digital ecosystem [37]. The binary extension field utilization for secure hardware implementations is of high interest because it leads to carry less arithmetic operation and provides a faster and smaller system [38]. As expected, the most significant operations in binary fields are addition, multiplication, squaring, and inversion. Using low-power and area-efficient hardware implementation will result in smaller, lower cost and more sustainable systems.

It could be concluded that the problems that arise when using applied science for overlapping areas between mathematics, computer engineering, electronic engineering are to be investigated in the present thesis project.

## 1.5 Proposed Methods and Models

Selecting the right variables deeply affects the outcome of the optimization process. Therefore, in order to consider the behavior of digital circuits in the power sector, the first stage is to choose suitable metrics. Through the design optimization process of AOP multipliers, it is expected to keep some of the variables constant in this thesis which are:

1. The cell height,
2. Power supply voltage (1.08 V).

Some of the variables which are subjected to be modified are:

1. Architecture of multiplier ,
2. Word length (bits).

State-of-the-art approaches for implementing reduction, multiplication, and accumulation operations of an AOP multiplication in a systolic array will be studied first. Alternate hardware organizations will then be studied with low-power and cost-efficient design targets. Verilog implementations of previously reported and proposed implementations will then be compared in three different binary field sizes, after synthesis of the circuits using a 90 nm standard cell library from Taiwan Semiconductor Manufacturing Company (TSMC). The synthesis will be done with a 100 MHz system clock frequency using the Genus Synthesis Solution tool from Cadence. The research method is summarized in Figure 1.2.

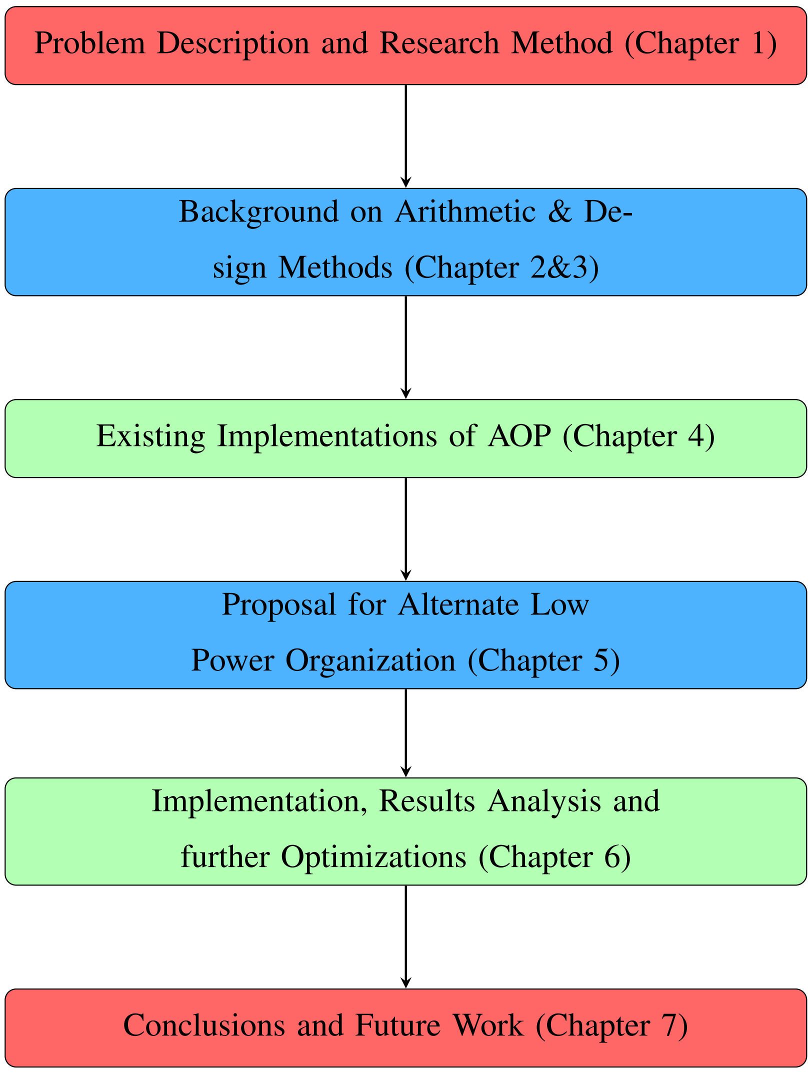

Problem Description and Research Method (Chapter 1)
Background on Arithmetic & Design Methods (Chapter 2&3)
Existing Implementations of AOP (Chapter 4)
Proposal for Alternate Low Power Organization (Chapter 5)
Implementation, Results Analysis and further Optimizations (Chapter 6)
Conclusions and Future Work (Chapter 7)

**Figure 1.2.** Flow-Chart of the Completion Steps of This Research

## 1.6 The Outline of the Thesis

The rest of the thesis is outlined as follows: The mathematical background is presented in Chapter 2. Chapter 3 provides background information on employed implementation and synthesis techniques. Chapter 4 discusses the existing implementations of irreducible AOP. The proposed low-power and area-efficient hardware organization is explained in Chapter 5. In Chapter 6, the proposed organization is implemented in different sizes and is compared against the alternatives in recent literature. Chapter 7 outlines the conclusion and future work.

# CHAPTER 2

# MATHEMATICAL BACKGROUND

This chapter introduces some key definitions and general concepts concerning the finite field's mathematical context.

## 2.1 Field

A field $F$ is a set of numbers which can be added, subtracted, multiplied and divided. The following are the properties of a field:

1. Commutativity
   $a + b = b + a$ for all $a, b \in F$,
   $ab = ba$ for all $a, b \in F$.

2. Associativity
   $a + (b + c) = (a + b) + c$ for all $a, b, c \in F$,
   $a(bc) = (ab)c$ for all $a, b, c \in F$.

3. Identity
   There is an element in $F$, denoted by $0_F$, such that $a + 0_F = a \quad \forall a \in F$,
   There is an element in $F$, denoted by $1_F$, such that $a \cdot 1_F = a \quad \forall a \in F$.

4. Inverse
   For each $a \in F$ there is an element in $F$, denoted by $-a$ , such that $a + (-a) = 0_F$,
   For each $a \neq 0_F$ there is an element in $F$, denoted by $a^{-1}$, such that $a \cdot a^{-1} = 1_F$.

5. Distributive law
   $a(b+c) = ab + ac \quad \forall a, b, c \in F$ [39].

Examples of fields include the real numbers ($\mathbb{R}$), complex numbers ($\mathbb{C}$), and rational numbers ($\mathbb{Q}$).

## 2.2 Finite Field

A finite field $F_q$ is a field that consists exactly of $q$ elements. It has been shown $q$ can be written as $q = p^m$, where p is a prime, and $m$ is a positive integer. A finite field is known as a Galois field in honor of Évariste Galois (1811-1832), a young French mathematician who contributed significantly to the theory of fields (see Figure 2.1). Therefore, it can be written $F_q$ or $GF(p^m)$ for the finite field of $q = p^m$ [18].

**Figure 2.1.** Portrait of Évariste Galois

### 2.2.1 Prime Fields

When $m$ equals one, $GF(p^1)$ is a the prime field. The elements of the prime field $GF(p)$ are the integers $\{0, 1, \ldots, p-1\}$ with operations (addition, subtraction, multiplication, and inverse) computed $mod\, p$.

Let $a, b \in GF(p)$,
$a + b \equiv c \quad mod\, p$,
$a - b \equiv d \quad mod\, p$,
$a.b \equiv e \quad mod\, p$,
$a.a^{-1} = 1 \quad mod\, p$.

In Figure 2.2, common operations are illustrated for a very common prime field, $GF(2)$ with the elements 0, 1.

| $+$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

| $\times$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

| a | 0 | 1 |
|---|---|---|
| -a | 0 | 1 |

| a | 0 | 1 |
|---|---|---|
| $a^{-1}$ | Undefined | 1 |

**Figure 2.2.** Arithmetic Operations for $GF(2)$

### 2.2.2 Extension Fields

We will call the fields $GF(p^m)$ with $m$ is greater than one extension fields. This is because they are extensions of their prime sub fields. Computers use the binary system; therefore, this thesis will focus on binary extension fields $GF(2^m)$. A binary extension field has $2^m$ elements, and there is an irreducible polynomial, say $f$, such that $GF(2^m) \cong GF(2)[x]/(f)$ (that is, polynomials with coefficients in $GF(2) modulo f$.

So the elements of $GF(2^m)$ can be represented by various (not necessarily irreducible) polynomials.

## 2.3 Basis of Finite Fields

Suppose that $V$ is an $n$ dimensional vector space over $F$. A basis is $e_1, e_2, \cdot, e_n \in V$ such that $\{a_1e_1 + a_2e_2 + \cdot + a_ne_n \mid a_1, \ldots, a_n \in F\} = V$. If these $e$'s are $\{x^0, x^1, x^2, \ldots, x^{m-1}, x^m\}$ it will be polynomial or (standard) basis since any element of $GF(2^m)$ can be expressed as bits using $x^0, x^1, x^2, \ldots, x^{m-1}, x^m$ which is called an extended basis of the canonical basis [40], and if the set is similar to $\{x^{q^0}, x^{q^1}, x^{q^2}, \ldots, x^{q^{m-1}}\}$. It is called normal basis. Figure 2.3 shows the hierarchy of finite field and basis.

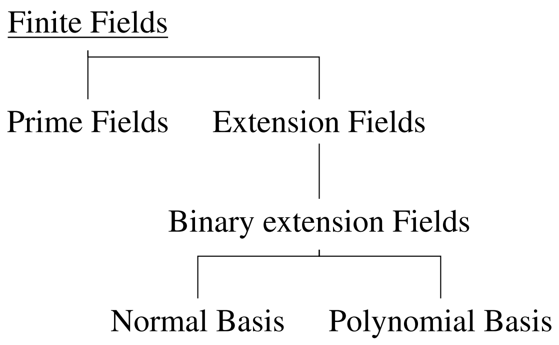


**Figure 2.3.** Finite Field and Basis [6], [7]

The polynomial and normal bases are also recommended by standards institutions for practical use. For hardware implementation, the polynomial basis multiplier is more widely used than the normal basis since the normal basis requires non-regular multiplication [41]; therefore, this thesis focuses on the polynomial basis. Equation (21) describes a set of polynomials

$$P(x) = x^m + a_{m-1}x^{m-1} + \cdots + a_1x + 1, \quad a_i \in GF(2) \quad for \quad 1 \leq i \leq m-1. \tag{21}$$

Table 2.1 summarizes all the concepts which have been studied so far.

**Table 2.1.** Binary Representation for the Polynomial in $GF(2^4)$

| Decimal | Polynomial | Binary |
|---|---|---|
| 0 | $0.x^4 + 0.x^3 + 0.x^2 + 0.x + 0.1$ | 00000 |
| 1 | $0.x^4 + 0.x^3 + 0.x^2 + 0.x + 1.1$ | 00001 |
| 2 | $0.x^4 + 0.x^3 + 0.x^2 + 1.x + 0.1$ | 00010 |
| 3 | $0.x^4 + 0.x^3 + 0.x^2 + 1.x + 1.1$ | 00011 |
| 4 | $0.x^4 + 0.x^3 + 1.x^2 + 0.x + 0.1$ | 00100 |
| 5 | $0.x^4 + 0.x^3 + 1.x^2 + +0.x + 1.1$ | 00101 |
| 6 | $0.x^4 + 0.x^3 + 1.x^2 + 1.x + 0.1$ | 00110 |
| 7 | $0.x^4 + +0.x^3 + 1.x^2 + 1.x + 1.1$ | 00111 |
| 8 | $0.x^4 + 1.x^3 + 0.x^2 + 0.x + 0.1$ | 01000 |
| 9 | $0.x^4 + 1.x^3 + 0.x^2 + 0.x + 1.1$ | 01001 |
| 10 | $0.x^4 + 1.x^3 + 0.x^2 + 1.x + 0.1$ | 01010 |
| 11 | $0.x^4 + 1.x^3 + 0.x^2 + 1.x + 1.1$ | 01011 |
| 12 | $0.x^4 + 1.x^3 + 1.x^2 + 0.x + 0.1$ | 01100 |
| 13 | $0.x^4 + 1.x^3 + 1.x^2 + 0.x + 1.1$ | 01101 |
| 14 | $0.x^4 + 1.x^3 + 1.x^2 + 1.x + 0.1$ | 01110 |
| 15 | $0.x^4 + 1.x^3 + 1.x^2 + 1.x + 1.1$ | 01111 |

## 2.4 Four types of Irreducible Polynomials

An irreducible polynomial of degree $m$ with coefficients in $GF(2)$ is used to produce $GF(2^m)$. To the best of the author's knowledge, irreducible polynomials have four main categories: Trinomial, Pentanomial, EPS, and AOP.

### 2.4.1 Trinomials

If a polynomial contains three nonzero terms, it is called trinomial. The national institute of standards and technology (NIST) recommends five binary finite fields for ECC, two of them are trinomials, and these irreducible polynomials security lifetime (duration of validity) is through 2030. The first trinomial is for $GF(2^{233})$ field as equation (22).

$$P(x) = x^{233} + x^{74} + 1, \tag{22}$$

Another trinomial is for $GF(2^{409})$, and the security lifetime is beyond 2030 [42].

$$P(x) = x^{409} + x^{87} + 1, \tag{23}$$

### 2.4.2 Pentanomials

If a polynomial consist of five nonzero terms, it is called pentanomial.

NIST recommends $GF(2^{163})$ with a security lifetime (duration of validity) through 2010. An irreduciable pentanomial defining $GF(2^{163})$ is

$$P(x) = x^{163} + x^7 + x^6 + x^3 + 1. \tag{24}$$

Another term via security life time beyond 2030 is for $GF(2^{283})$ and $GF(2^{571})$ via the following irreducible polynomials [42],

$$P(x) = x^{283} + x^{12} + x^7 + x^5 + 1, \tag{25}$$

$$P(x) = x^{571} + x^{10} + x^5 + x^2 + 1. \tag{26}$$

### 2.4.3 ESP

An ESP irreducible polynomial is the expression (27),

$$P(x) = x^{ns} + x^{(n-1)s} + \cdots + x^s + 1, \quad m = ns \quad for \quad 1 \leq s \leq \frac{m}{2}. \tag{27}$$

### 2.4.4 AOP

An AOP is a polynomial in the following form:

$$P(x) = x^m + x^{m-1} + \cdots + x + 1, \tag{28}$$

where the coefficients of all terms are one. An AOP is irreducible and may be used to define $GF(2^m)$ if and only if $m + 1$ is a prime and 2 is a primitive root modulo $m + 1$ [43].

As an example, for $m \leq$ 600 the AOP is irreducible for the following values of $m$: 2, 4, 10, 12, 18, 28, 36, 52, 58, 60, 66, 82, 100, 106, 130, 138, 148, 162, 172, 178, 180, 196, 210, 226, 268, 292, 316, 346, 348, 372, 378, 388, 418, 420, 442, 460, 466, 490, 508, 522, 540, 546, 556, 562, 586 respectively [20].

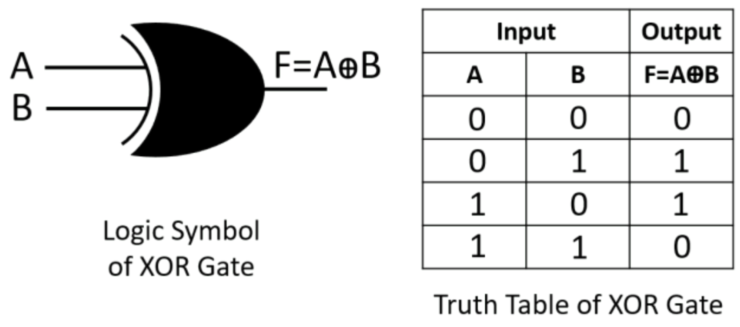
In this thesis, three of these bit lengths have been chosen, namely 162, 268, and 562, to analyze and compare hardware organizations.

## 2.5 $GF(2^m)$ Arithmetic Operations

Addition, multiplication, squaring, and inversion are four essential arithmetic operations in $GF(2^m)$. These operations are performed modulo an irreducible polynomial $P(x)$ over $GF(2)$.

### 2.5.1 Addition and Subtraction

The addition of polynomials is performed under modulo two arithmetic. Consequently, the addition of two polynomials becomes the bitwise Exclusive OR (XOR) of their binary representations. Figure 2.4 depicts the XOR logic symbol and the truth table.



| Input | | Output |
|---|---|---|
| A | B | F=A⊕B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

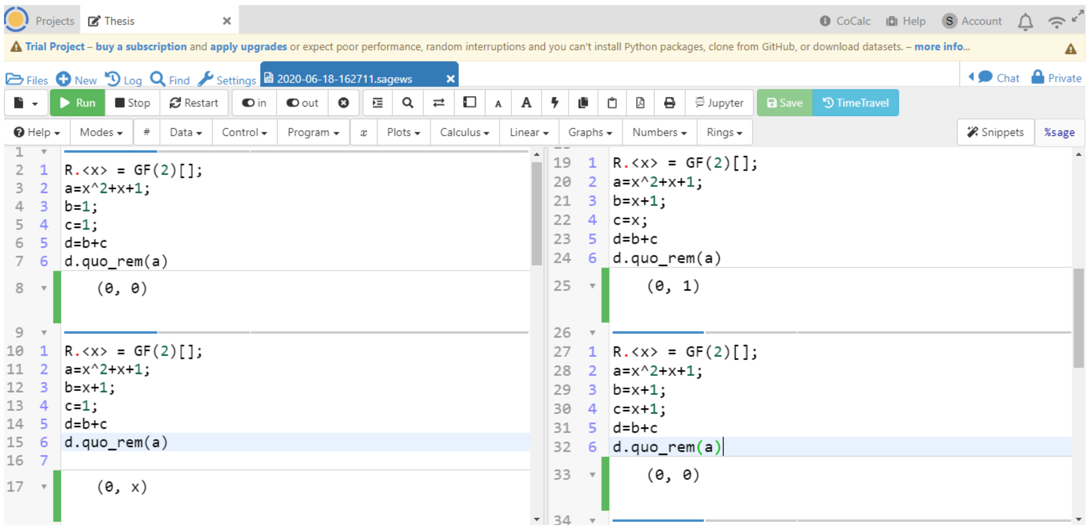
**Figure 2.4.** Logic Symbol and Truth Table of XOR Gate

The following table shows the addition when $m$ is equal to 2.

**Table 2.2.** Addition for $GF(2^2) = \mathbb{Z}_2[x](mod \quad x^2 + x + 1)$

| + | 0 | 1 | $x$ | $x+1$ |
|---|---|---|---|---|
| 0 | 0 | 1 | $x$ | $x+1$ |
| 1 | 1 | 0 | $x+1$ | $x$ |
| $x$ | $x$ | $x+1$ | 0 | 1 |
| $x+1$ | $x+1$ | $x$ | 1 | 0 |

Figures 2.5, and 2.6 display the same values as the Table 2.2, and it computes using Collaborative Calculation in the Cloud (CoCalc), a sophisticated web-based platform for computational mathematics [13].

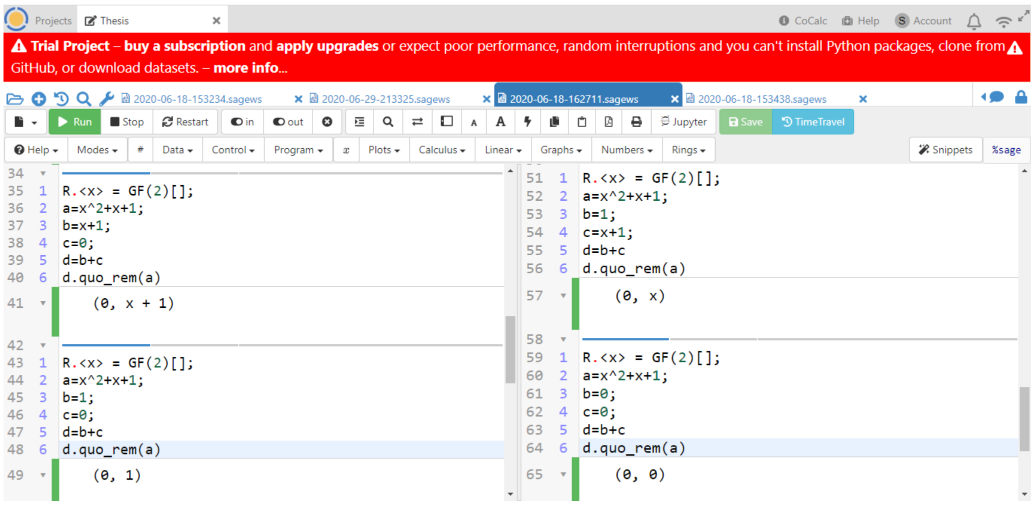


Trial Project – buy a subscription and apply upgrades or expect poor performance, random interruptions and you can't install Python packages, clone from GitHub, or download datasets. – more info...

```
R.<x> = GF(2)[];
a=x^2+x+1;
b=1;
c=1;
d=b+c
d.quo_rem(a)
    (0, 0)

R.<x> = GF(2)[];
a=x^2+x+1;
b=x+1;
c=1;
d=b+c
d.quo_rem(a)

    (0, x)

R.<x> = GF(2)[];
a=x^2+x+1;
b=x+1;
c=x;
d=b+c
d.quo_rem(a)
    (0, 1)

R.<x> = GF(2)[];
a=x^2+x+1;
b=x+1;
c=x+1;
d=b+c
d.quo_rem(a)
    (0, 0)
```

**Figure 2.5.** CoCalc Addition Computation

The other four additional computations are deprecated in Figure 2.6.

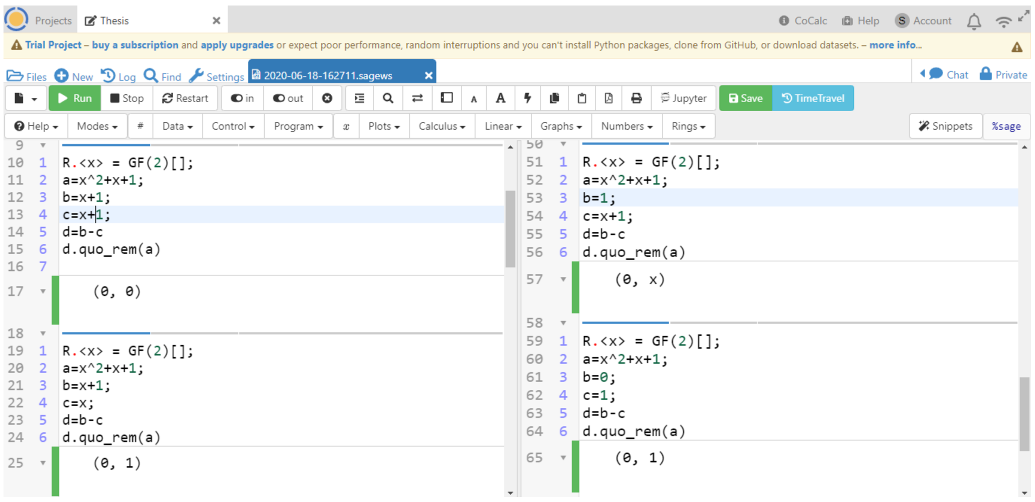


Trial Project – buy a subscription and apply upgrades or expect poor performance, random interruptions and you can't install Python packages, clone from GitHub, or download datasets. – more info...

```
R.<x> = GF(2)[];
a=x^2+x+1;
b=x+1;
c=0;
d=b+c
d.quo_rem(a)
    (0, x + 1)

R.<x> = GF(2)[];
a=x^2+x+1;
b=1;
c=0;
d=b+c
d.quo_rem(a)
    (0, 1)

R.<x> = GF(2)[];
a=x^2+x+1;
b=1;
c=x+1;
d=b+c
d.quo_rem(a)
    (0, x)

R.<x> = GF(2)[];
a=x^2+x+1;
b=0;
c=0;
d=b+c
d.quo_rem(a)
    (0, 0)
```

**Figure 2.6.** Continued CoCalc Addition Computation

Subtraction is identical as an addition in modulo two arithmetic, as illustrated in Figure 2.7.



Trial Project – buy a subscription and apply upgrades or expect poor performance, random interruptions and you can't install Python packages, clone from GitHub, or download datasets. – more info...

```
R.<x> = GF(2)[];
a=x^2+x+1;
b=x+1;
c=x+1;
d=b-c
d.quo_rem(a)

    (0, 0)
```

```
R.<x> = GF(2)[];
a=x^2+x+1;
b=x+1;
c=x;
d=b-c
d.quo_rem(a)
    (0, 1)
```

```
R.<x> = GF(2)[];
a=x^2+x+1;
b=1;
c=x+1;
d=b-c
d.quo_rem(a)
    (0, x)
```

```
R.<x> = GF(2)[];
a=x^2+x+1;
b=0;
c=1;
d=b-c
d.quo_rem(a)
    (0, 1)
```

**Figure 2.7.** CoCalc Subtraction Computation

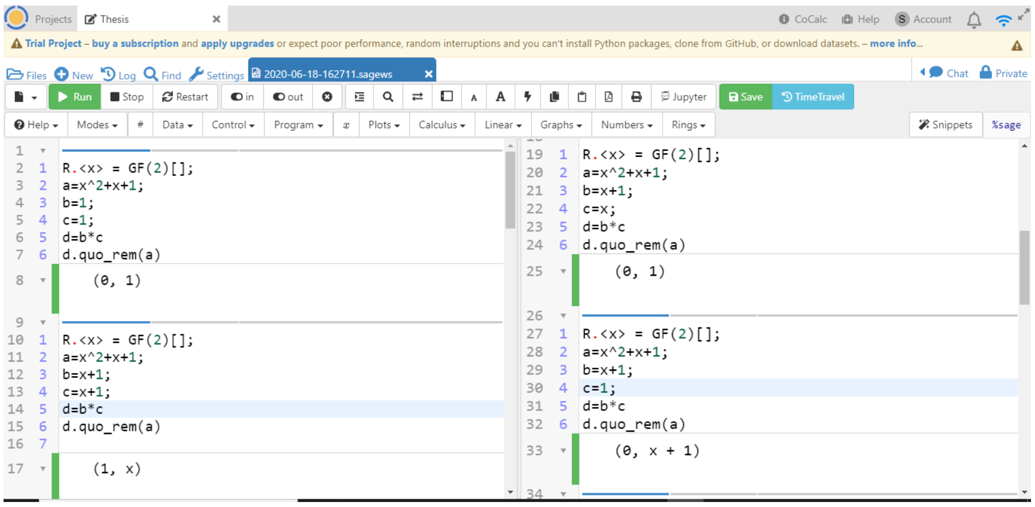
### 2.5.2 Multiplication in $GF(2^m)$

Multiplication is a standard multiplication of polynomials followed by a reduction of modulo irreducible polynomials [44]. Therefore, multiplication in $GF(2^m)$ is more complex than addition and subtraction since it involves two steps: polynomial multiplication and reduction modulo an irreducible polynomial.

**Table 2.3.** Multiplication for $GF(2^2) = \mathbb{Z}_2[x](mod \quad x^2 + x + 1)$

| . | 0 | 1 | $x$ | $x+1$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | $x$ | $x+1$ |
| $x$ | 0 | $x$ | $x+1$ | 1 |
| $x+1$ | 0 | $x+1$ | 1 | $x$ |

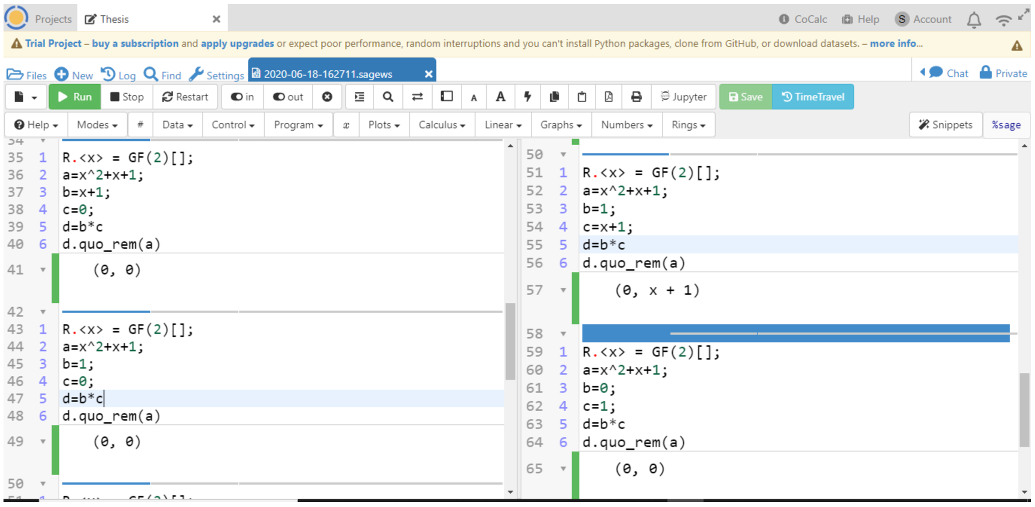
Figures 2.8, and 2.9 illustrate the eight values of Table 2.3.



Trial Project – buy a subscription and apply upgrades or expect poor performance, random interruptions and you can't install Python packages, clone from GitHub, or download datasets. – more info...

```
R.<x> = GF(2)[];
a=x^2+x+1;
b=1;
c=1;
d=b*c
d.quo_rem(a)
    (0, 1)

R.<x> = GF(2)[];
a=x^2+x+1;
b=x+1;
c=x+1;
d=b*c
d.quo_rem(a)

    (1, x)

R.<x> = GF(2)[];
a=x^2+x+1;
b=x+1;
c=x;
d=b*c
d.quo_rem(a)
    (0, 1)

R.<x> = GF(2)[];
a=x^2+x+1;
b=x+1;
c=1;
d=b*c
d.quo_rem(a)
    (0, x + 1)
```

**Figure 2.8.** CoCalc Multiplication Computation



Trial Project – buy a subscription and apply upgrades or expect poor performance, random interruptions and you can't install Python packages, clone from GitHub, or download datasets. – more info...

```
R.<x> = GF(2)[];
a=x^2+x+1;
b=x+1;
c=0;
d=b*c
d.quo_rem(a)
    (0, 0)

R.<x> = GF(2)[];
a=x^2+x+1;
b=1;
c=0;
d=b*c
d.quo_rem(a)
    (0, 0)

R.<x> = GF(2)[];
a=x^2+x+1;
b=1;
c=x+1;
d=b*c
d.quo_rem(a)
    (0, x + 1)

R.<x> = GF(2)[];
a=x^2+x+1;
b=0;
c=1;
d=b*c
d.quo_rem(a)
    (0, 0)
```

**Figure 2.9.** Continued CoCalc Multiplication Computation

Irreducible AOPs, trinomials, and pentanomials are known as different types of irreducible polynomials [28], and each of them has different algorithms for implementation.

#### 2.5.2.1 AOP Multiplication

An AOP is a polynomial in the following form:

$$P(x) = x^m + x^{m-1} + \cdots + x + 1, \tag{29}$$

Then $\{1, x, x^2, \cdots, x^m\}$ are the polynomial extended bases of $GF(2^m)$[40]. Let $x$ be a root of the irreducible AOP. Since $P(x) = 0$, it can first be concluded that:

$$x^m = x^{m-1} + \cdots + x + 1. \tag{210}$$

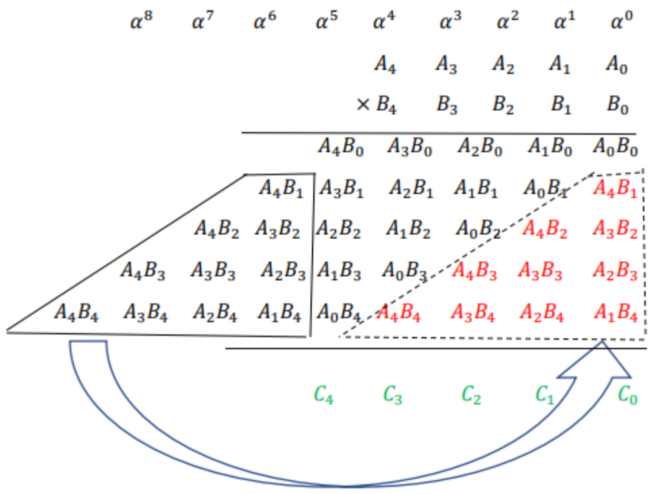
Second,

$$\begin{aligned} P(x) + xP(x) =& (x^m + x^{m-1} + \cdots + 1) \\ & + x(x^m + x^{m-1} + \cdots + 1) = 0. \end{aligned} \tag{211}$$

Then,

$$x^{m+1} = 1. \tag{212}$$

Equation (212) is a property of AOP that can be used to reduce the complexity of algorithm as follows:



$\alpha^8$ $\alpha^7$ $\alpha^6$ $\alpha^5$ $\alpha^4$ $\alpha^3$ $\alpha^2$ $\alpha^1$ $\alpha^0$
$A_4$ $A_3$ $A_2$ $A_1$ $A_0$
$\times B_4$ $B_3$ $B_2$ $B_1$ $B_0$
$A_4B_0$ $A_3B_0$ $A_2B_0$ $A_1B_0$ $A_0B_0$
$A_4B_1$ $A_3B_1$ $A_2B_1$ $A_1B_1$ $A_0B_1$ $A_4B_1$
$A_4B_2$ $A_3B_2$ $A_2B_2$ $A_1B_2$ $A_0B_2$ $A_4B_2$ $A_3B_2$
$A_4B_3$ $A_3B_3$ $A_2B_3$ $A_1B_3$ $A_0B_3$ $A_4B_3$ $A_3B_3$ $A_2B_3$
$A_4B_4$ $A_3B_4$ $A_2B_4$ $A_1B_4$ $A_0B_4$ $A_4B_4$ $A_3B_4$ $A_2B_4$ $A_1B_4$
$C_4$ $C_3$ $C_2$ $C_1$ $C_0$

**Figure 2.10.** AOP Multiplication $GF(2^4)$

Although the explained algorithm makes the computation more straightforward, in hardware implementation is cumbersome, so two more approaches from the literature will be introduced next to reduce the implementation complexity.

**The First Algorithm For Implementation**

Let

$$A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1x + a_0 = \sum_{k=0}^{m-1} a_kx^k, \tag{213}$$

and the next polynomial be

$$B(x) = b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots + b_1x + b_0 = \sum_{k=0}^{m-1} b_kx^k, \tag{214}$$

and the primitive, irreducible polynomial

$$F(x) = x^m + f_{m-1}x^{m-1} + f_{m-2}x^{m-2} + \cdots + f_1x + f_0. \tag{215}$$

Suppose $P$ is the product of A and B,

$$P = A \cdot B, \tag{216}$$

Substitute equation (214) in (216)

$$P = A \cdot \sum_{k=0}^{m-1} b_k x^k, \tag{217}$$

so $A$ can move inside the summation

$$P = \sum_{k=0}^{m-1} (Ax^k) b_k. \tag{218}$$

Since in the finite field, when we multiply one polynomial by $x$, the coefficients rotate, and the main structure remains the same so that we can rewrite (213) as below.

$$Ax^k = a_{m-1}^{(k)} x^{m-1} + a_{m-2}^{(k)} x^{m-2} + \cdots + a_1^{(k)} x + a_0^{(k)} = \sum_{n=0}^{m-1} a_n^{(k)} x^n, \tag{219}$$

according to (219) and (218) it can be written as

$$P = \sum_{k=0}^{m-1} (\sum_{n=0}^{m-1} a_n^{(k)} x^n) b_k, \tag{220}$$

it can be written as

$$P = \sum_{n=0}^{m-1} (\sum_{k=0}^{m-1} a_{n-1}^{(k)} b_k) x^n, \tag{221}$$

Let us set $p_n$ as

$$p_n = \sum_{k=0}^{m-1} a_n^{(k)} b_k. \tag{222}$$

So $P$ can be written as (according to equations (221) and (222))

$$P = \sum_{n=0}^{m} p_n x^n. \tag{223}$$

It can be concluded that the highest degree of $P$ can be $m-1$. Now let's compute $Ax^k$ in a different way. It can be written as

$$Ax^k = (Ax^{k-1})x, \tag{224}$$

According to (219) substituting $k$ instead of $k-1$

$$Ax^{k-1} = (\sum_{n=0}^{m-1} a_n^{(k-1)} x^n), \tag{225}$$

Therefore,

$$Ax^k = \sum_{n=0}^{m-1} a_n^{(k-1)} x^{n+1}. \tag{226}$$

Expand the summation

$$Ax^k = a_{m-1}^{(k-1)} x^m + \sum_{n=0}^{m-2} a_n^{(k-1)} x^{n+1}. \tag{227}$$

We are changing the interval of summation.

$$Ax^k = a_{m-1}^{(k-1)} x^m + \sum_{n=1}^{m-1} a_{n-1}^{(k-1)} x^n. \tag{228}$$

Let us consider (215) when $x$ is the root of $F(x)$ so that $F(x) = 0$. We obtain

$$x^m = f_{m-1}x^{m-1} + f_{m-2}x^{m-2} + \cdots + f_1 x + f_0 = \sum_{n=1}^{m-1} f_n x^n + f_0. \tag{229}$$

Via substituting (229) in the equation (228)

$$Ax^k = a_{m-1}^{(k-1)} (\sum_{n=1}^{m-1} f_n x^n + f_0) + \sum_{n=1}^{m-1} a_{n-1}^{(k-1)} x^n, \tag{230}$$

Let us expand

$$Ax^k = a_{m-1}^{(k-1)} f_0 + a_{m-1}^{(k-1)} \sum_{n=1}^{m-1} f_n x^n + \sum_{n=1}^{m-1} a_{n-1}^{(k-1)} x^n, \tag{231}$$

Factor $x^n$

$$Ax^k = a_{m-1}^{(k-1)} f_0 + \sum_{n=1}^{m-1} (a_{n-1}^{(k-1)} + a_{m-1}^{(k-1)} f_n) x^n, \tag{232}$$

As it is known (219) can be written in

$$Ax^k = a_0^{(k)} + \sum_{n=1}^{m-1} a_n^{(k)} x^n. \tag{233}$$

If (232) and (233) are compared, we can conclude these equations are exactly same as in [12]

$$\begin{cases} a_0^{(k)} = a_{m-1}^{(k-1)} f_0 \\ a_n^{(k)} = a_{n-1}^{(k-1)} + a_{m-1}^{(k-1)} f_n \quad \text{for} \;\; 1 \leq n \leq m-1 \end{cases} \tag{234}$$

The implementation of the algorithm will be discussed in Chapter 4 in detail.

**The Second Algorithm For Implementation**

Let $A(x), B(x), C(x) \in GF(2^m)$.

$$A(x) = \sum_{k=0}^{m} a_k x^k, \quad B(x) = \sum_{k=0}^{m} b_k x^k, \quad C(x) = \sum_{k=0}^{m} c_k x^k, \tag{235}$$

where $a_k$, $b_k$, and $c_k \in GF(2)$ for $0 \leq k \leq m$. Also, $a_m$, $b_m$, and $c_m$ are equal to zero. Galois field multiplication in extension binary field is performed as [25]:

$$C(x) \equiv A(x) \cdot B(x) \quad modP(x). \tag{236}$$

Substituting equation (235) into (236), and applying commutativity property, the following can be derived:

$$C(x) \equiv \sum_{i=0}^{m} b_i(A(x)x^i) \quad modP(x). \tag{237}$$

The next step is computing $(A(x)x^i) \quad modP(x)$ part of (237). Defining $A^{(0)} = A$ and $A^{(i)} = A(x)x^i \quad modP(x)$, we obtain

$$A^{(i+1)} = xA^{(i)} \quad modP(x). \tag{238}$$

and writing $A^{(i)}$ same as $A(x)$:

$$A^{(i)} = a_m^{(i)}x^m + \cdots + a_1^{(i)}x + a_0^{(i)} = \sum_{k=0}^{m} a_k^{(i)}x^k. \tag{239}$$

In the same manner,

$$\begin{aligned} A^{(i+1)} &= a_m^{(i+1)}x^m + \cdots + a_1^{(i+1)}x + a_0^{(i+1)} \\ &= \sum_{k=0}^{m} a_k^{(i+1)}x^k. \end{aligned} \tag{240}$$

If (239) is multiplied by $x$, it will be

$$A^{(i)}x = a_m^{(i)}x^{m+1} + \cdots + a_1^{(i)}x^2 + a_0^{(i)}x. \tag{241}$$

Substituting $x^{m+1}$ from 212, and using (241),

$$A^{(i+1)} = xA^{(i)} = a_{m-1}^{(i)}x^m + \cdots + a_1^{(i)}x^2 + a_0^{(i)}x + a_m^{(i)}. \tag{242}$$

(240) and (242) can be written vectors of coefficients as [18]:

$$A^{(i+1)} = [a_m^{(i+1)}, \cdots, a_1^{(i+1)}, a_0^{(i+1)}], \tag{243}$$

$$A^{(i+1)} = xA^{(i)} = [a_{m-1}^{(i)}, \cdots, a_0^{(i)}, a_m^{(i)}]. \tag{244}$$

Comparing (243), and (244),

$$\begin{cases} a_0^{(i+1)} = a_m^{(i)} & \text{if } k = 0 \\ a_k^{(i+1)} = a_{k-1}^{(i)} & \text{if } 1 \leq k \leq m. \end{cases} \tag{245}$$

The presented algorithm is the same as [2], which is utilized in different hardware implementations.

#### 2.5.2.2 Trinomial Multiplication

The trinomial general form is

$$P(x) = x^m + x^s + 1. \tag{246}$$

Therefore,

$$x^m = x^s + 1. \tag{247}$$

Substituting (247) in (240) we obtain

$$A^{(i+1)} = xA^{(i)} = a_{m-1}^{(i)}(x^s + 1) + \cdots + a_1^{(i)}x^2 + a_0^{(i)}x + a_m^{(i)}. \tag{248}$$

Comparing (240) and (248) will be the following

$$\begin{cases} a_0^{(i+1)} = a_{m-1}^{(i)} \\ a_s^{(i+1)} = a_{s-1}^{(i)} + a_{m-1}^{(i)} \\ a_k^{(i+1)} = a_{k-1}^{(i)} & \text{for } \ 1 \leq k \leq m-1 \quad and \quad k \neq s. \end{cases} \tag{249}$$

So to implement this the circuit we will need some wiring and one XOR gate to do the addition.

#### 2.5.2.3 Pentanomial Multiplication

Let

$$P(x) = x^m + x^s + x^p + x^t + 1. \tag{250}$$

be a general pentanomial. Therefore,

$$x^m = x^s + x^p + x^t + 1. \tag{251}$$

Substituting (251) at (240) will be

$$A^{(i+1)} = xA^{(i)} = a_{m-1}^{(i)}(x^s + x^p + x^t + 1) + \cdots + a_1^{(i)}x^2 + a_0^{(i)}x + a_m^{(i)}. \tag{252}$$

Comparing (240), and (252) will be the following equation

$$\begin{cases} a_0^{(i+1)} = a_{m-1}^{(i)} \\ a_s^{(i+1)} = a_{s-1}^{(i)} + a_{m-1}^{(i)} \\ a_p^{(i+1)} = a_{p-1}^{(i)} + a_{m-1}^{(i)} \\ a_t^{(i+1)} = a_{t-1}^{(i)} + a_{m-1}^{(i)} \\ a_k^{(i+1)} = a_{k-1}^{(i)} & \text{for } 1 \leq k \leq m-1 \quad and \quad k \neq s, p, t. \end{cases} \tag{253}$$

In order to implement this equation, we will need wiring and 3 XOR gates.

Although the AOP class of irreducible polynomials offers considerably efficient implementation of finite field multiplication [2], they are not as commonly studied as irreducible trinomials or pentanomials. Thus, the multiplier architecture proposed in this work is based on AOP. AOPs are relatively simple to implement for finite field generation [45], and are packed as computation cores to be used for trinomial and pentanomials multipliers [3].

### 2.5.3 Squaring

Multiplication with only one entry is one method for applying to the square in the finite field [28].

$$C(x) \equiv A(x) \cdot A(x) \ \ modP(x) = A(x)^2 \ \ modP(x). \tag{254}$$

In addition, according to the author's knowledge in the normal basis shifting is squaring.

### 2.5.4 Inversion

If $P(x)$ is an irreducible polynomial and $A(x)$ is a nonzero polynomial of the finite field $GF(2^m)$ has to inverse such as the following:

$$A(x) \cdot A(x)^{-1} \ \ modP(x) = 1. \tag{255}$$

It is the most complex operation that the Euclidean algorithm is required to be used. For this part, the modulus $P(x)$ is critical to be irreducible [44]. Euclidean algorithm for polynomials computes the greatest common divisor (GCD) polynomial of two polynomials. The optimized algorithm based on the Euclidean algorithm for calculating inversion is discussed in [46],[47].

# CHAPTER 3

# BACKGROUND ON DESIGN METHODS

This chapter is categorized into three main sections: Physical design, design organization, and VLSI custom digital design flow to addresses various hardware organization and hardware system architecture techniques for implementation; in other words, all parameters that can affect the design of IC's are discussed. The design methods and tools are outlined in the last sections of this chapter.

## 3.1 Physical Design

In this section, relevant IC physical design principles are discussed to provide a robust system.

### 3.1.1 Process Corner

Process lots are defined as unique improved wafers designed to adapt substantiating chip schematic strength for confirming process alterations that statistically occur in wafer manufacturing throughout the years. Process lots are kind of products proposed by semiconductor manufacturers. Corner lot wafers are a group of wafers skewed by the fab to different corners. The function of process lots is to verify the immunity of the planned design to process alterations in the future. A two-letter classification system is utilized by the industry to identify the different corners, in which the initial letter stands for the NMOS device, and the second means the PMOS device. There are five exemplary corners:

1. FF (fast-fast),

2. SF (slow-fast),

3. SS (slow-slow),

4. FS (fast-slow),

5. TT (typical typical) [48].

### 3.1.2 Thermal Effects

Analysis performed on data ascertains that architectures with high logic density in arithmetic and logic unit (ALU) in processors affect heat generation through non-uniform power density. Hot spots located in the heat map of a single-core processor, as illustrated in Figure 3.1, can be seen, especially on ALU.

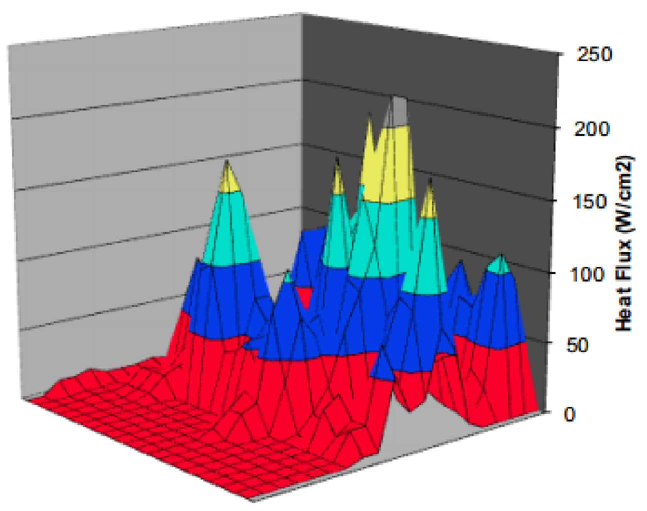


**Figure 3.1.** Power Density [8]

Furthermore, Figure 3.2 illustrates the Niagara T1 Processor's heat map. Additionally, the figure shows the power apex in proximity to the arithmetic logic unit (ALU). Overall, in ALU and especially in graphical processing units (GPU), multiplication is considered as the most time and power-consuming process because a large amount of energy is consumed to compress and deliver an enormous body of data.

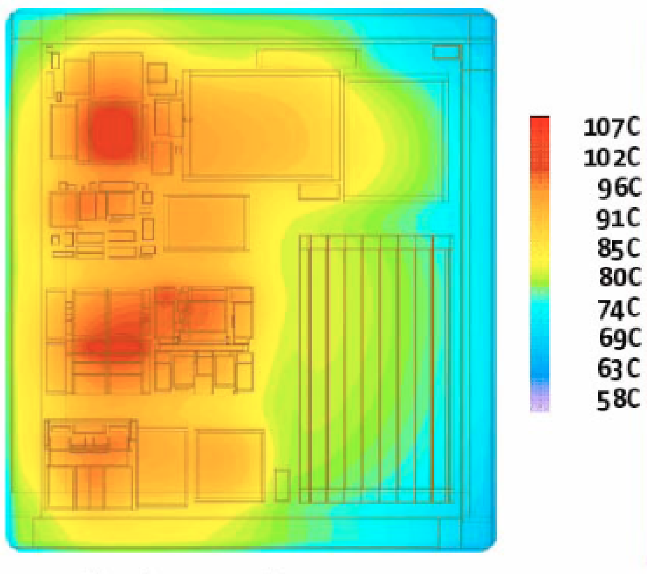


**Figure 3.2.** Map of Temperature for Single Core Processor [9]

### 3.1.3 Routing Layers

Another touchstone for the analysis of technological progress and compression of chips is the number of routing layers. The number of routing layers in a circuit refers to the number of surfaces that hold the circuit connections and stack them in layers without contact with each other. Evidently, with a higher number of routing layers, chip production technology is more advanced, and the implementation is more compact. Table 3.1 shows the number of connection layers in chips in recent years.

**Table 3.1.** Maximum Number of Routing Layers [1]

| Year | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 | 2007 | 2010 | 2013 | 2016 |
|---|---|---|---|---|---|---|---|---|---|---|
| Number of Routing Layers | 7 | 8 | 8 | 8 | 9 | 9 | 9 | 10 | 10 | 10 |

### 3.1.4 Power Supply Voltage and Allowable Maximum Power

Another factor for examining the agents influential in the development of chip technology is the power supply voltage. The power supply voltage connected to a chip has a significant impact on its total power. The total power of a chip is a sum of two parameters: Dynamic power and static power. Dynamic power dissipation in the electrical circuits is obtained from the following relation.

$$P_D = \alpha.C_L.V_{DD}^2.f_s, \tag{31}$$

In equation (31), $\alpha$ is activity factor. $C_L$ is capacitive loading, $V_{DD}$ is power supply voltage and $f_s$ is switching frequency.

The dynamic power of integrated circuits is positively related to its power supply voltage square. In addition, static power in integrated circuits, which mainly consists of idle leakage power when transistors are not switching, has a complicated superlinear relationship to supply voltage. Accordingly, reducing supply voltage leads to a significant reduction in power dissipation due to both dynamic and static effects.

Table 3.2 shows how power supply voltage and a maximum allowable power of chips have changed in the last ten years (2001 to 2010) and foresees the maintenance of this process until 2016.

**Table 3.2.** Processing of Power supply Voltage (Low operation Power, high $V_{DD}$ transistor) and Maximum Power High Performance with Heat-sink [1]

| Year | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 | 2007 | 2010 | 2013 | 2016 |
|---|---|---|---|---|---|---|---|---|---|---|
| Power supply Voltage $(V)$ | 1.2 | 1.2 | 1.1 | 1.1 | 1.0 | 1.0 | 0.9 | 0.8 | 0.7 | 0.6 |
| Max Power $(W)$ | 130 | 140 | 150 | 160 | 170 | 180 | 190 | 218 | 251 | 288 |

The optimal voltage level is critical in digital circuits since it affects power, whereas delay is inversely equivalent to the voltage [49].

### 3.1.5 Transistor Density

Gordon Moore, by inspecting the compression of the components and the complexity of their routing network, introduced his well-known theory that the number of transistors placed on a fixed surface is almost doubled every 1.5 to 2 years. He made this observation of the process and claimed that such progress would be everlasting.

It could be concluded that it is possible to trace the integration of the components to exponential function by following the significant points of the digital chips examination. In general, if the number of transistors on a chip in 1965 is equal to $k$, in the specified year, the number of transistors is calculated as follows, according to Moore's Law.

$$Number\ of\ Transistor\ in\ a\ Chip\ in\ Year\ y = k.2^{(\frac{year-1965}{2})}. \tag{32}$$

Although the theory has remained valid until recent years, studies show that it is no longer applicable.

Moreover, it is essential to note that in addition to processors, the technology of manufacturing other electronic components such as memories, microcontrollers, and data transmission equipment, etc. has also improved according to Moore's law; simultaneously, it has made the modern advanced chips compatible with each other [50].

### 3.1.6 Transistor Length

In order to examine the mentioned factors in the process of manufacturing chips in recent years, Figure 3.1 shows that the history of the development and growth of chips in the world are to be explored. As the first observation of the figure below reveals, it shows the process of gate length changes in digital chips since 1995. Obviously, any change in the process of development of the construction of chip should not be at the expense of accuracy or processing speed of the chip. As it is clear from the diagram, the length of the gate is reduced by about 30% for each two or three-year period. The length of the transistor was articulated as micrometers until the early 1990s, but today, due to the sharp decline in this size, its evaluation unit is expressed in Nano terms. For instance, this chart postulates that the dimensions of transistors inside the chip will have reached less than 10 nm by 2020, which is proven true [10].

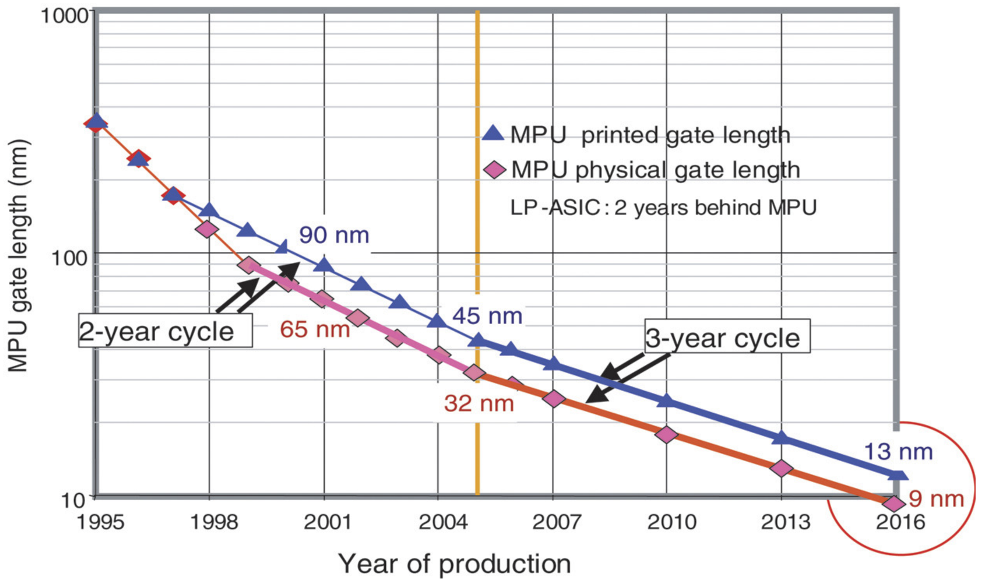


**Figure 3.3.** MOSFET channel length scaling in IC's [10] Reprinted from Materials Today, Wolfgang Arden, Review key challenges, Page No. 41, 2003, with permission from Elsevier. Copyright © 2003 Elsevier Science Ltd. (appendix)

### 3.1.7 Clock Speed

Another factor is the processing speed of CPU chips. The clock pulse is a common signal in synchronous sequential circuits. By adjusting the clock frequency, the processing speed of circuits inside the chip can be changed. The clock period depends on the delay of the circuits between consecutive registers. The delay is the time that the transition of input through the output of the circuit is expected to have an effect. A circuit's maximum operating frequency is equal to the inverse of the maximum delay in the circuit. This introduces the concept of the critical path.

**The critical path in sequential circuits**: The path between two modules (usually two registers) synchronized by a common clock, which has the longest delay among all available paths between successive synchronized modules. Figure 3.4 shows a path in sequential circuits as well. The two elements 1 and 2, refer to two successive synchronized modules. It is clear that the constitutional elements of the path are all combinational.

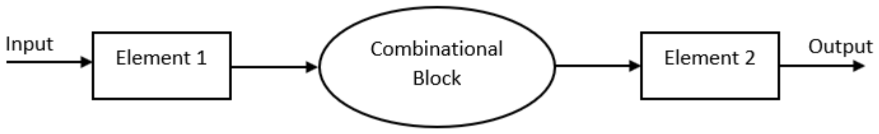


**Figure 3.4.** Path in Sequential Circuits

After understanding the concept of the clock, the process of increasing clock speed in microprocessors from 1975 to 2020 is to be examined. The process of growing microprocessors clock speed continued exponentially until 2000, however it slowed down by entering the third millennium. This problem is raised by obstacles such as elements delay inside the chip, and total power is a severe obstacle to technological progress. The chart shows that before the year 2000; the microprocessors clock speed has almost doubled every two years. This rate, nevertheless, has occurred every 2.5 years since 2000.

### 3.1.8 Cost

In addition to all the aforementioned factors, the cost of manufacturing chips is also one of the important issues in evaluating the level of technological progress, which must be examined at each level of manufacturing and production of chips. It is logical to say that, when a new technology ramps in production, at first, the costs are greatly reduced, and the market changes direction towards new technology, yet gradually with the passage of time and the normalization of the situation, this process slows down, and everything gets ready for new technology to come out. [50].

### 3.1.9 Average Cost

The last topic discussed in the development of IC is the average price of each transistor in digital chips. The price of transistors manufactured in the past 30 years has changed in US dollars. In the 70s and 80s, the price of transistors was almost halved per year. On the other hand, after these two decades, this price reduction has occurred every 1.5 years [50]. It could be concluded that advances made in manufacturing technology affect and reduce the price of transistors in the final plan.

## 3.2 Design Organization

### 3.2.1 Pipelines or Single Cycle

Latency and throughput can describe the speed of a system. The latency of a system is the time needed for a group of inputs to pass through the system from start to end. The throughput is the number of groups of output that is being produced per unit time. Consequently, throughput can be improved by processing several groups of information at the same time. One form of this process is called pipelining. Pipelining breaks a task into stages, and it speeds up a circuit without doubling the hardware [32].

### 3.2.2 Systolic or Non-Systolic Designs

Multiplication over $GF(2^m)$ in aspects of their design scheme can be divided into two main forms.

1. Systolic or semi-systolic,
2. Non-systolic [51].

The key goal of the non-systolic designs [52], [53], [54] is to reduce the number of partial products to achieve multipliers with lower hardware and shorter latency. The systolic designs typically have a higher latency compared to non-structural designs, but their benefits are linked to regularity and modularity of design and simplicity [51].

### 3.2.3 Input/Output Architectures

The $GF(2^m)$ multiplier can be divided into three main types as regards the input-output structure:

1. Serial-in, serial-out,
2. Parallel-in, parallel-out,
3. Serial-in, parallel-out.

Within the serial in serial out systems, only one new input bit is taken, and one bit of output per cycle is generated. Such structures are lightweight and can be used in resource-restricted systems; they have disadvantages in high-speed applications for a given system clock frequency. The bit-level Parallel-in parallel-out structures offer extremely high throughput rates at relatively high hardware costs [55],[56]. The serial-in parallel-out architecture, as the name implies, is a hybrid of the above structures.

## 3.3 VLSI Custom Digital Design Flow

Electronic systems are ubiquitously used in modern society, and integrated circuits are crucially important for modern compact, energy-efficient, high-performance electronic systems. Today's Very Large Scale Integrated (VLSI) circuits have thousands to billions of transistors and interconnections within a tiny area. Such a design is an elaborate and lingering process, and the complexity of the process is continually increasing. The past decades have thus witnessed an increased use of designer productivity applications. The result is the emergence of sophisticated Computer-Aided Design (CAD) tools. Contemporary digital design requires description through a Hardware Description Language (HDL) (which is a high-level description akin to a programming language), and benefits greatly from synthesis tools to produce optimized circuit layouts that could be sent off to a silicon IC manufacturer in a short time [57]. VLSI chips used in most designs appear in three types: Full custom, application-specific integrated circuit (ASIC), or system on chip (SOC) design. SOCs consists of the integration of previously generated design blocks on a single chip. In the full custom approach, VLSI chips or some of their parts are hand-designed MOSFET by MOSFET, allowing each transistor and each wire individually to be defined by a set of polygons. Photographic masks needed in the fabrication process are thus generated. Because full custom design is slow and cumbersome, a unique methodology to create a standard cell-based design has been introduced to decrease the design time by making use of standard cells or pre-designed components as building blocks, where-from large logic circuits can be developed. All cells adapt certain vertical or horizontal grid size to optimize the overall layout area and cost [33]. A standard cell ASIC methodology incorporates a standard cell library and automated design tools to make use of this library so that higher designer productivity can be attained. The designer describes the circuit behavior in a hardware description language (HDL) such as Verilog or VHDL. This advanced specification is then annexed to a library of standard cells that execute miscellaneous logic functions. Further, synthesis flow iterations are executed to meet latency/frequency, power, or area targets specified by the designer. The capacitance of the wires is determined by a wire load model because, at the synthesis stage, the final structure of the chip is not determined. Once the standard cells are placed, wires are dispatched between them, and a clock tree network is filled in to supply the clock signal [36].

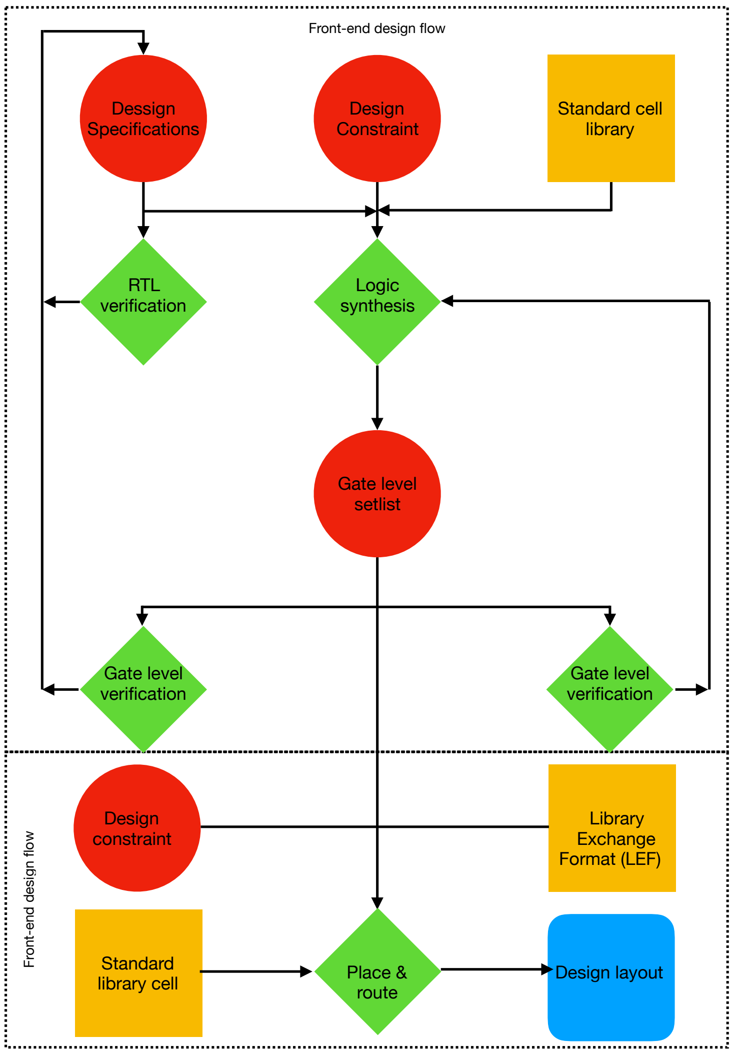


**Figure 3.5.** Design Flow for Standard Cell ASIC [11]

The design flow is put into two categories called front end and back end.

### 3.3.1 Front-End Design Flow

As depicted in Figure 3.5, front-end design flow contains logic synthesis, register transfer level (RTL) verification, gate-level net listing and verification, and gate-level optimization. Logic synthesis is a method of changing a design structure to a gate-level netlist. HDL is employed to describe the design. The logic synthesizer first scrutinizes the HDL code, the standard cell library, and the design constraints. Subsequently, the HDL code is rendered into logic blocks that are contained in a technology-independent library. Ultimately, the synthesizer projects the logic netlist to a specific technology library, improving the design based on constraints declared by the designer throughout the mapping.

### 3.3.2 Back-End Design Flow

The netlist based on the standard cell library is transformed into a design layout in the back-end design flow. Herein, an EDA tool implements and routes the cells. For back-end design flow, the Genus synthesis method could be applied.

### 3.3.3 Synthesis

Logic synthesis transforms HDL code into a netlist in terms of hardware (e.g., the logic gate and the wires connecting them). The logic synthesizer may enact optimizations to decrease the required amount of hardware. The netlist might be a text file, or else it may be viewed as a schematic to help visualize the circuit. As part of the Cadence IC Mixed-Signal Design Tool Suite, the Genus tool has been utilized throughout this research to generate synthesized circuits and layout from the Verilog design description. Cadence specifications report that:

"The Genus Synthesis Solution is a next-generation RTL synthesis and physical synthesis tool that provides up to a 10X boost in RTL design productivity with up to 5X faster turnaround times. The solution can scale its capacity to well beyond 10 million instances flat. It also supplies tight timing and wire length correlation to within 5% of place and route. Using the Genus Synthesis Solution, you can experience a 2X or more reduction in iterations between block-level and unit-level synthesis [58]."

A review of the flow in the Genus synthesis solution will be provided in the following sections.

### 3.3.4 Basic Design Flow in Genus Synthesis Solution

To ensure reproducible results, the experiments should be carried out in a controlled environment. Figure 3.6 shows the specific execution flow of the Genus synthesis solution.

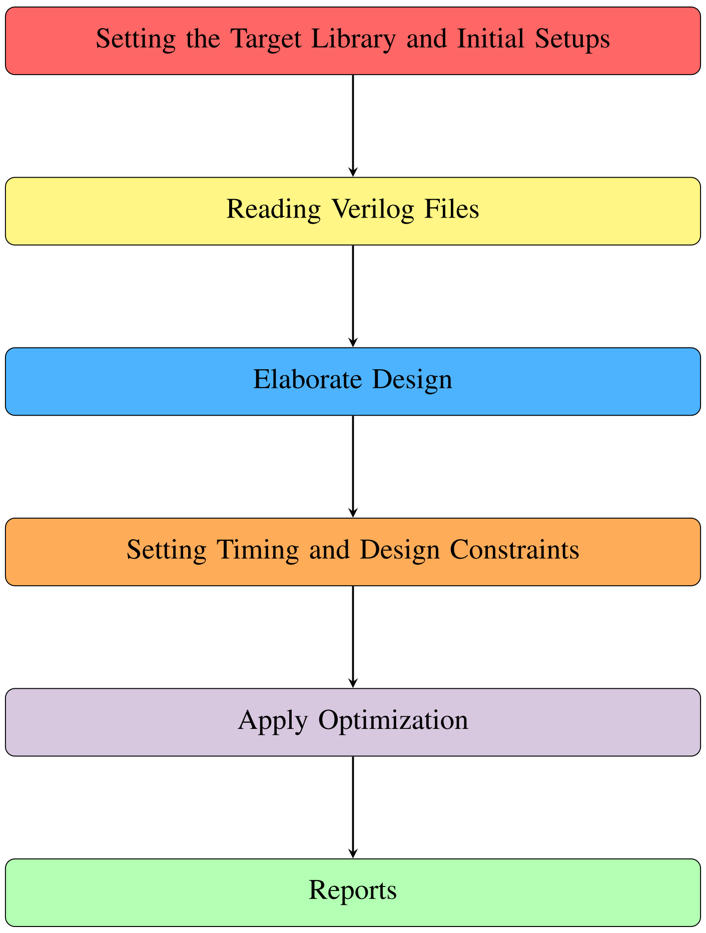

Setting the Target Library and Initial Setups
Reading Verilog Files
Elaborate Design
Setting Timing and Design Constraints
Apply Optimization
Reports

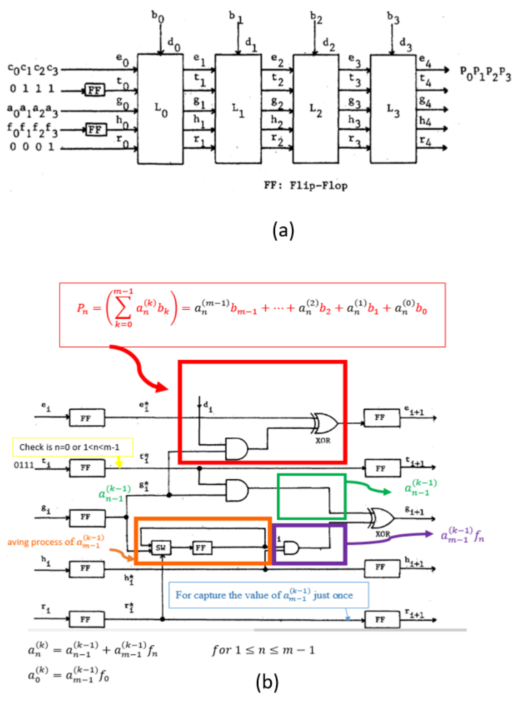
**Figure 3.6.** Genus Specific Execution Flow

# CHAPTER 4

# EXISTING IMPLEMENTATIONS OF IRREDUCIBLE FINITE FIELD MULTIPLIER

## 4.1 Yeh et al. Implementation

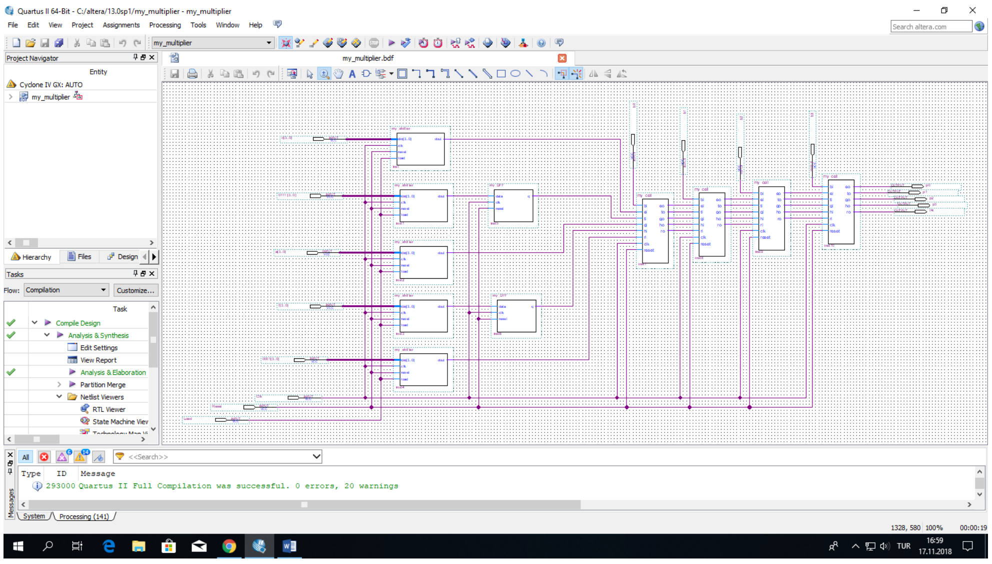
Yeh et al. have presented a serial-in, serial-out, one-dimensional systolic array over a finite field multiplier [12]. Figure 4.1 depicts the details of their implementation. This architecture is based on the algorithm, which was discussed in Chapter 2, section 2.5.2.1.



$b_0$ $b_1$ $b_2$ $b_3$
$d_0$ $d_1$ $d_2$ $d_3$
$c_0c_1c_2c_3$
0 1 1 1
$a_0a_1a_2a_3$
$f_0f_1f_2f_3$
0 0 0 1
FF
$L_0$ $L_1$ $L_2$ $L_3$
$p_0p_1p_2p_3$
FF: Flip-Flop
(a)
$P_n = \left(\sum_{k=0}^{m-1} a_n^{(k)} b_k\right) = a_n^{(m-1)} b_{m-1} + \cdots + a_n^{(2)} b_2 + a_n^{(1)} b_1 + a_n^{(0)} b_0$
Check is n=0 or 1<n<m-1
XOR
SW
$a_{n-1}^{(k-1)}$
aving process of $a_{m-1}^{(k-1)}$
$a_{m-1}^{(k-1)} f_n$
For capture the value of $a_{m-1}^{(k-1)}$ just once
$a_n^{(k)} = a_{n-1}^{(k-1)} + a_{m-1}^{(k-1)} f_n$ for $1 \le n \le m-1$
$a_0^{(k)} = a_{m-1}^{(k-1)} f_0$
(b)

**Figure 4.1.** (a) a Serial-in, Serial-out Systolic Multiplier for the Finite Field (b) the Circuit of the Cell $L_i$ [12]

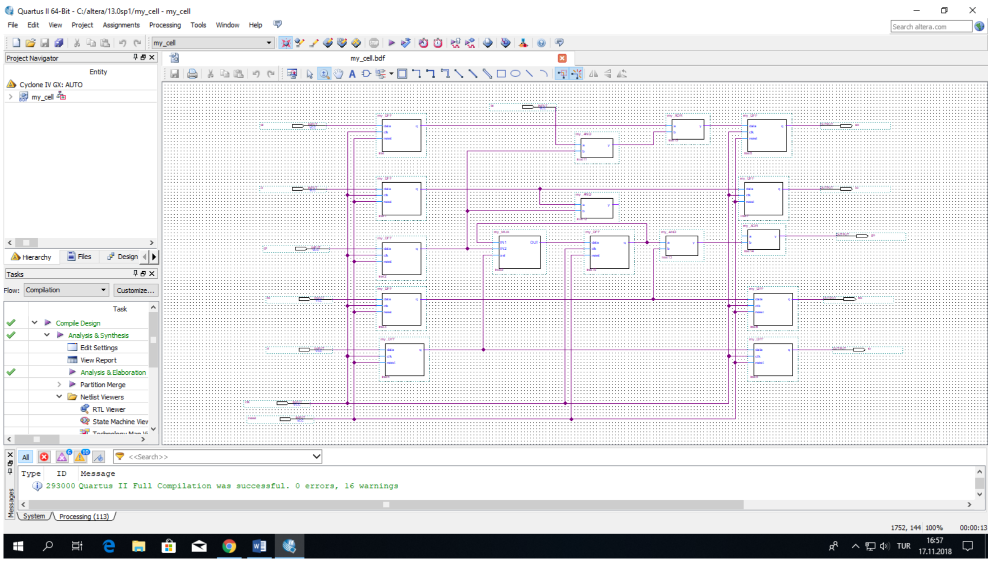
To comprehend the function of this system author implemented this circuit by utilizing Altera Quartus II. Figure 4.2 is a serial-in, serial-out systolic multiplier for the finite field, functionally the same as in Figure 4.1.



```
Type  ID     Message
      293000 Quartus II Full Compilation was successful. 0 errors, 20 warnings
```

**Figure 4.2.** Implementation of the Systolic Multiplier for the Finite Field via Quartus

Figure 4.3 shows a cell of the design the same as part b of Figure 4.1.



```
Type  ID     Message
      293000 Quartus II Full Compilation was successful. 0 errors, 16 warnings
```

**Figure 4.3.** Implementation of the Cell $L_i$ via Altera Quartus

Figure 4.4 shows the waveform of this design, and then it is compared with calculated on paper 4.5, and at Colac 4.6, as it can see it, all of them show the same result, which is correct. In the red box, it can be seen the result 10000 in the five clock cycle.

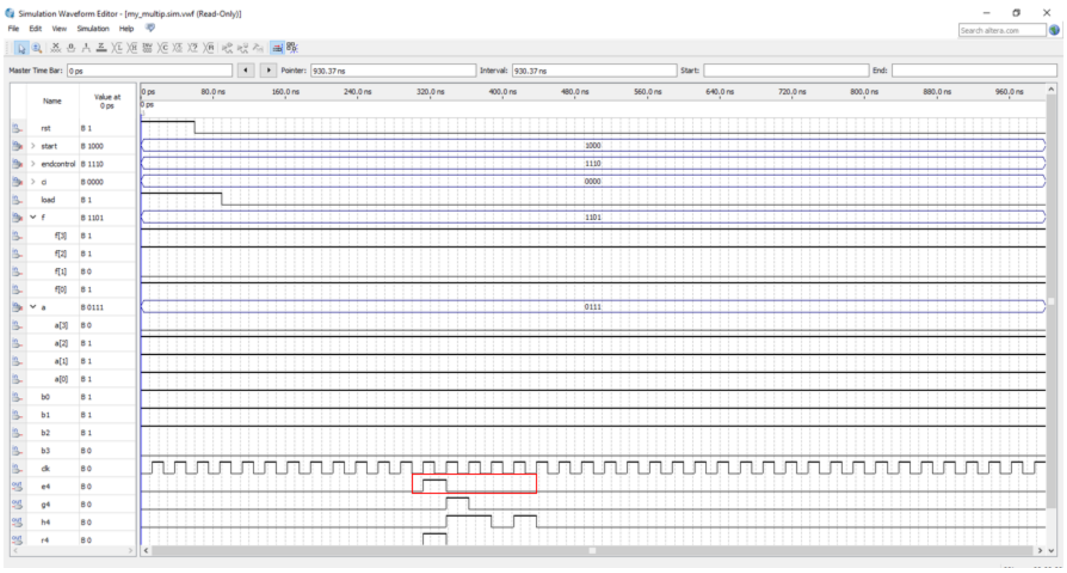
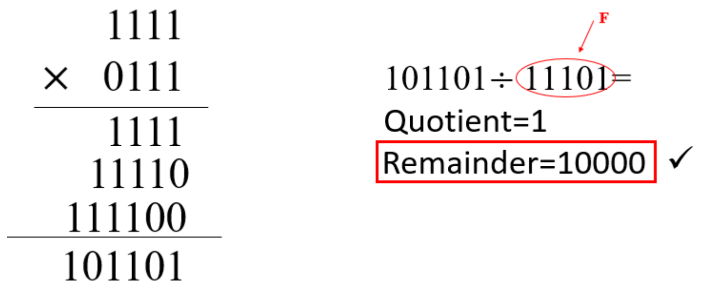


**Figure 4.4.** Quartus Waveform

It can be computed as follows.



$$\begin{array}{r} 1111 \\ \times\ 0111 \\ \hline 1111 \\ 11110 \\ 111100 \\ \hline 101101 \end{array}$$

F

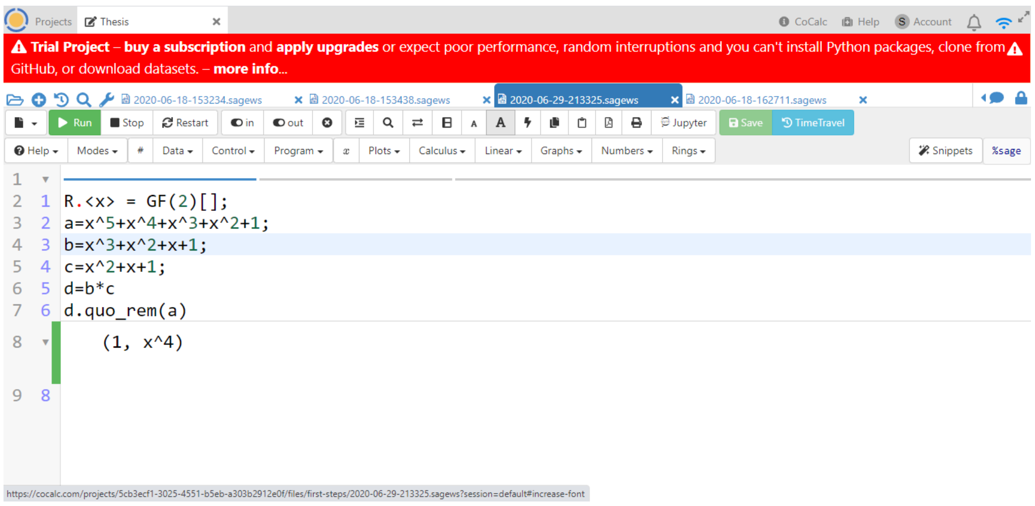
101101 ÷ 11101 =

Quotient=1

$\boxed{\text{Remainder=10000}}$ ✓

**Figure 4.5.** Calculation

The same example is done in Colac [13]. As $x^4$ is the same as 10000.



⚠ **Trial Project** – **buy a subscription** and **apply upgrades** or expect poor performance, random interruptions and you can't install Python packages, clone from ⚠ GitHub, or download datasets. – **more info**...

```
R.<x> = GF(2)[];
a=x^5+x^4+x^3+x^2+1;
b=x^3+x^2+x+1;
c=x^2+x+1;
d=b*c
d.quo_rem(a)
```

```
    (1, x^4)
```

https://cocalc.com/projects/5cb3ecf1-3025-4551-b5eb-a303b2912e0f/files/first-steps/2020-06-29-213325.sagews?session=default#increase-font

**Figure 4.6.** Colac Calculation [13]

## 4.2 Implementations of Irreducible AOP Finite Field Multiplier

The first bit-parallel multiplier over $GF(2^m)$ based on Irreducible AOP was proposed by [40]. Although bit-parallel multipliers require fewer clock cycles to perform a complete computation, they often utilize more areas and may not provide high throughput. Regular systolic architectures are compelling for implementing efficient synthesized circuits in large binary extension fields and are common due to their pertinence to pipelining, scalability, and other benefits [59], [60]. A systolic bit-parallel AOP-based multiplier was proposed by [61], which was improved further in the area-delay aspect by [2].

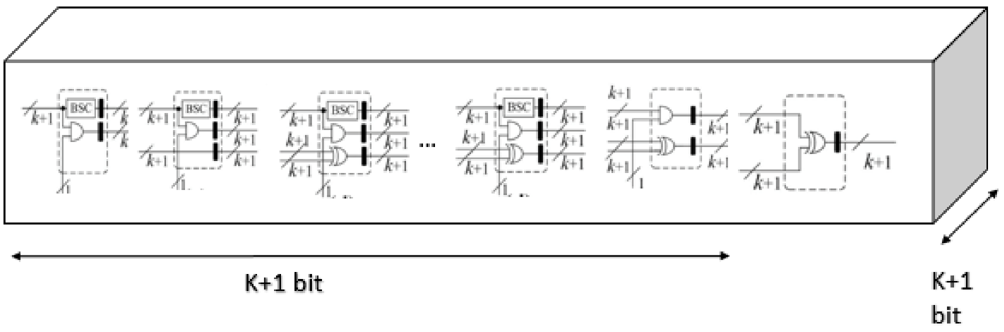


**Figure 4.7.** Multiplier Architecture for $GF(2^m)$ Based on AOP [2]

As shown in the 4.7 diagram, it contains AND gates, XOR gates, Register, and a box by name BCD. Figure 4.8 indicates BCD (BCD1) implementation and needs only wiring. This cabling hardware is based equation (245), which is discussed in chapter two.

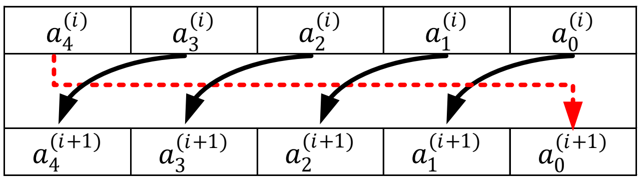


**Figure 4.8.** BSC1 structure

**Table 4.1.** Hardware and Time Complexity Multipliers of $GF(2^m)$ Based on AOP [2]

| Latency | AND Count | XOR Count | Register Count |
|---|---|---|---|
| $k+2$ | $(k+1)^2$ | $((k+1)-2+1)(k+1) = K^2+K$ | $(3k+2+1)(k+1) = 3K^2+6K+3$ |

## 4.3 Lower Register Complexity Implementation

Another systolic AOP multiplier implementation with lower register complexity was presented in [3], which was then used as a computing core to construct a trinomial base multiplier. As can be seen in Figure 4.9, this architecture, therefore, uses share registers, decreases the number of registers, uses NAND gates instead of AND gates, and uses XNOR gates instead.

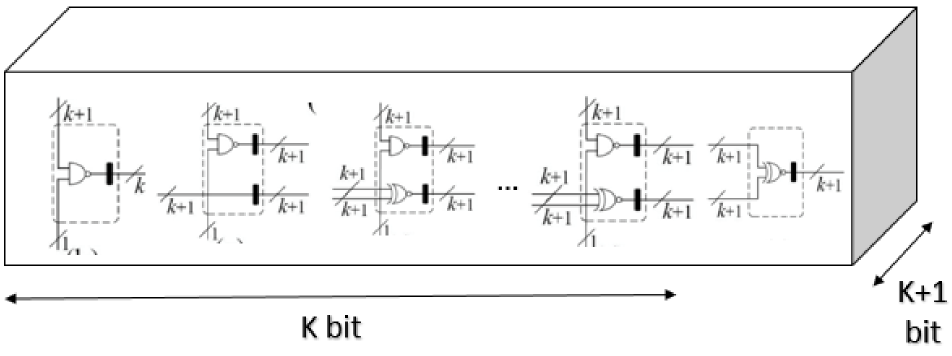


**Figure 4.9.** Multiplier Architecture of $GF(2^m)$ Based on AOP [3]

**Table 4.2.** Hardware and Time Complexity Multipliers of $GF(2^m)$ based on AOP [3]

| Latency | NAND Count | XNOR Count | Register Count |
|---|---|---|---|
| $k+1$ | $k(k+1) = k^2 + k$ | $((k-2)+1)(k+1) = K^2 - 1$ | $(2(k-1)+1+1)(k+1) = 2K^2 + 2K$ |

## 4.4 Low Cost, Low Latency, Energy-Efficient Designs

Power dissipation and energy have been considered as important metrics in a number of studies [62], [63], [64]. Recently, Meher and Lou [4] presented a regular and efficient recursive formulation for the bit-parallel systolic multiplier to reduce power and latency consumption for computing multiplication over $GF(2^m)$ using AOP basis, which is further re-architected in this work for both energy and layout area (cost) improvements.

In [4], BSC1, BSC$l$, and BSC$(l+1)$ are defined. BSC1s are located in the first row, BSC$(l+1)$s in the next r (value of reminder) rows, and BSC$l$s are positioned in the rest of the rows. The result of each BSC is multiplied with corresponding b using AND operation. The results of AND gates go through XOR, and this cycle repeats recursively. The final result of each row is input to a pipelined adder-tree (XOR tree). Figure 4.10 depicts the implementation in [4] for $GF(2^m)$.

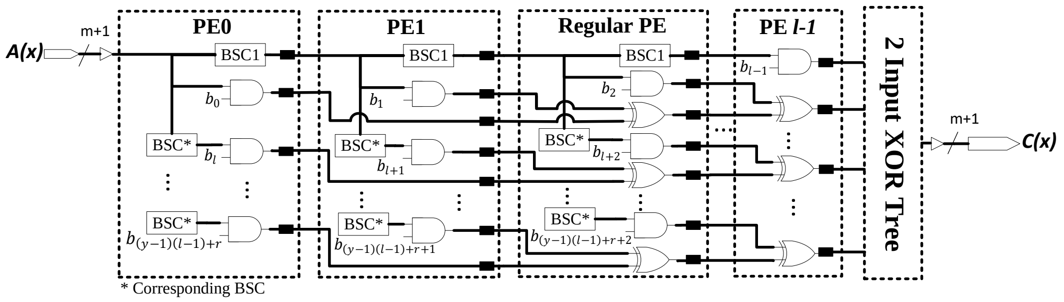


**Figure 4.10.** Multiplier Architecture for $GF(2^m)$ Based on AOP [4]

For example, when m is 10 is finite field irreducible, as shown in figure 4.11.

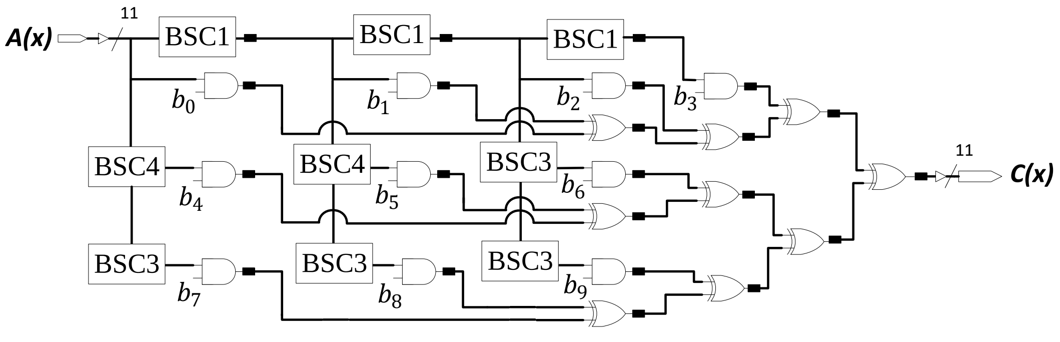


**Figure 4.11.** Multiplier Architecture for $GF(2^{10})$ Based on AOP [4]

# CHAPTER 5

# PROPOSED LOW-POWER AND AREA-EFFICIENT HARDWARE ORGANIZATION

## 5.1 Approach

There are a variety of aspects of the implementation [4] that can be optimized, such as a large number of registers and the extent of the XOR tree, which are the critical causes for the dissipation of power in this organization. Besides, there is an increased reliance on the number of rows of architecture that can affect latency and defining BSC and programming aspects. This research has, therefore, tried to encourage the most recent state-of-the-art organization to make an original contribution by providing low-power, area-efficient, and lower latency.

## 5.2 Proposed Design

The proposed design contains three main computation blocks, as depicted in Figure 5.1. The first module is the BSC network, which contains BSC0 to BSC$m-1$. This unit consolidates the reduction calculations, which can be implemented through wiring, as shown in figure 4.8. In other words, all $A^i$s will be computed in this unit. Calculated $A^i$s are multiplied with the corresponding $b_i$ in the AND network. In the last unit, the results from bit multiplication are processed through the three input XOR tree (accumulation) to compute the final output(see equation (51)).

$$C(x) \equiv \sum_{i=0}^{m} b_i(A^i). \tag{51}$$

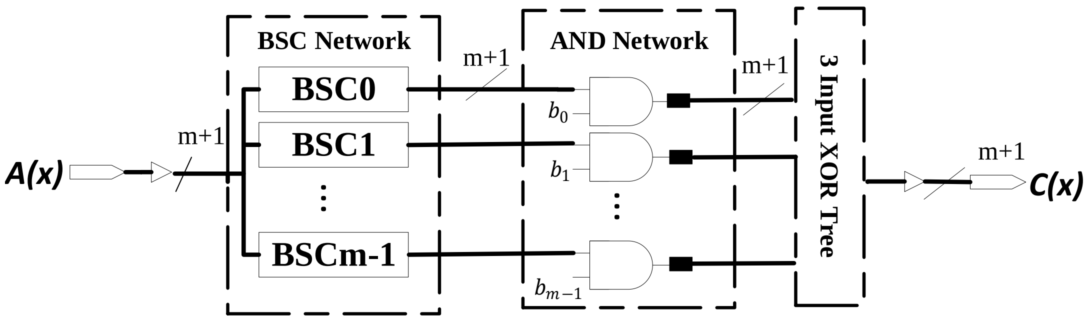


**Figure 5.1.** Proposed Multiplier Architecture for $GF(2^m)$ Based on AOP

Figure 5.2 depict implementations of proposed architectural schemes with $GF(2^{10})$ based on AOP. It is clear from this simple example that the proposed approach flattens the logic network, and henceforth creates an overall latency reduction opportunity through reduction of serial paths and overall hardware when compared to Figure 4.11. In applications that are more sensitive to single AOP multiplication execution time than pipeline throughput, the latency margin can be translated to even lower power consumption by reducing clock frequency.

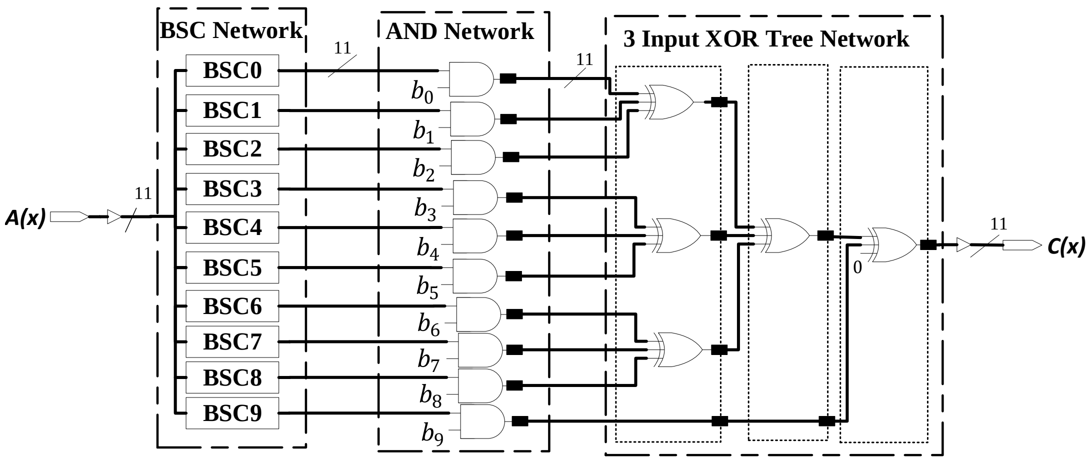


**Figure 5.2.** Proposed Multiplier Architecture for $GF(2^{10})$ Based on AOP

## 5.3 Analysis and Complexity

Table 5.1 presents the hardware complexity and latency comparison of hardware organization proposed here and in reference [4]. The logic delay path in the proposed implementation includes an AND network followed by a 3-input XOR tree, so the logic latency of the proposed architecture is $2 + [\log_3 m]$. However, in [4] the latency is $l + [\log_2 y] + 1$, where $l$ represents the number of columns and $[\log_2 y]$ represents the delay in the XOR tree for $y$ processing elements in a row. The throughput in both implementations corresponds to one operation per clock cycle. All XOR and AND gates are defined as $m + 1$ bit so the number of XOR and AND gates will be $(\sum_{i=0}^{[\log_3 m]-1} 2^i)(m + 1)$ and $m(m + 1)$ respectively for the proposed implementation. Data is staged at synchronous registers at the output of all AND and XOR gates.

**Table 5.1.** Architecture Level Hardware and Time Complexity Comparison Between Bit Parallel Multipliers of $GF(2^m)$

| Structure | Latency | AND Count | XOR Count | Register Count |
|---|---|---|---|---|
| [4] | $l^a + [\log_2 y^b] + 1$ | $m(m+1)$ | $(m + \sum_{i=0}^{[\log_2 y]-1} 2^i)(m+1)$ | $(2m + l + \sum_{i=0}^{[\log_2 y]-1} 2^i)(m+1) + \frac{ml}{2}$ |
| This Work | $2 + [\log_3 m]$ | $m(m+1)$ | $(\sum_{i=0}^{[\log_3 m]-1} 2^i)(m+1)^c$ | $(m + \sum_{i=0}^{[\log_3 m]-1} 2^i)(m+1)$ |

[a] $l$ is the number of the processing element in rows.

[b] $y$ is the number of the processing element in columns.

[c] Three input XOR count.

## 5.4 Summary of Theoretical Benefits

It may seem the proposed architecture relinquishes regularity compared to the state of the art systolic array approaches since it is not made up of processing elements with similar features. Nevertheless, the design is not less regular than the one shown in Figure 5.1, is scalable, and is easily synthesized using parameterized design approaches. Divergence from the architecture based on systolic processing element array for the benefit of lower cost and power dissipation is well justified in the synthesized digital VLSI environment.

**Table 5.2.** Compared Architectural Schemes for AOP Multipliers Over $GF(2^m)$

| m | Structure | Latency | AND Count | XOR Count | Register Count |
|---|---|---|---|---|---|
| 162 | [4] | 15 | 26406 | 28688 | 59486 |
| | This Work | 7 | 26406 | 2445[a] | 28851 |
| 268 | [4] | 15 | 72092 | 75858 | 161249 |
| | This Work | 7 | 72092 | 8339[a] | 80431 |
| 562 | [4] | 15 | 316406 | 324288 | 693022 |
| | This Work | 8 | 316406 | 17453[a] | 333859 |

[a] Three input XOR count.

As shown in Table 5.2 number of AND gates is unchanged, and the number of XOR gates is reduced at the proposed design since it uses three XOR gate inputs such that the height of the XOR tree (accumulation) is reduced. While registers are necessary after each level in the XOR tree, the tree height in the proposed work is less than state-of-the-art and decreases the number of registers used throughout the section of accumulation. Importantly, in the proposed organization, the majority of register reductions are due to the decrease in latency (flatting of the architecture). As can be seen in Table 5.3, the number of registers is reduced by about 50%.

**Table 5.3.** % Register Reduction

| Mult. Size | % Reduction in Register Count |
|---|---|
| 162 | 51 |
| 268 | 50 |
| 562 | 52 |

# CHAPTER 6

# IMPLEMENTATION AND RESULTS

## 6.1 Functional Verification

[2], [4] and Proposed architectures have been implemented. These three organizations were implemented in structural Verilog hardware description language at RTL.

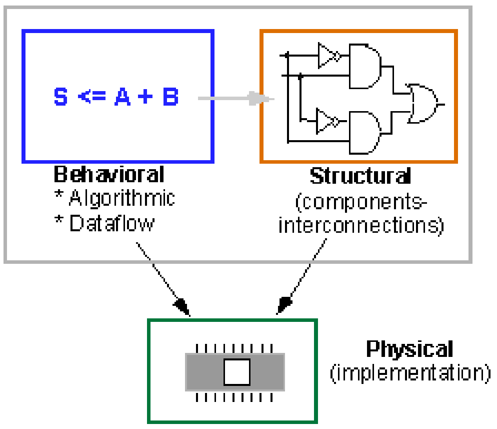


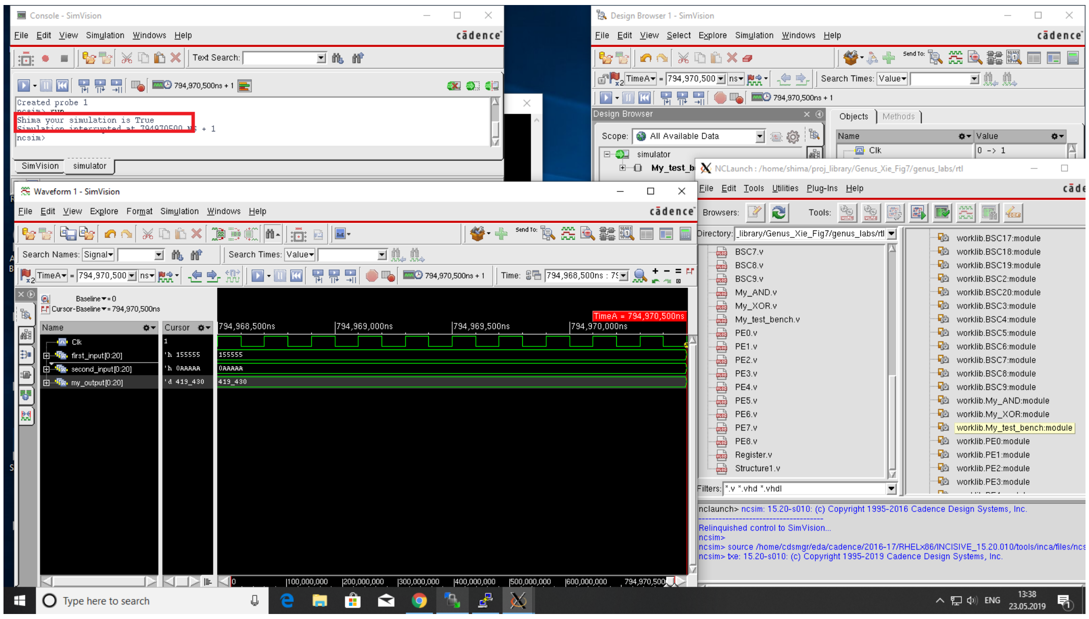
**Figure 6.1.** Different Levels of Representation in Digital Circuit [14]

Programming via HDL's has two main reasons.

1. Logic verification (and simulation),
2. Synthesis.

To ensure that the module operates correctly during simulation, inputs to a module are stimulated with test vectors, and outputs are compared against expected values. This effectively eliminates human errors (bugs) in the system. Debugging and correcting errors after the system is built can be devastatingly expensive. Therefore, logic verification and simulation are essential steps to verify a system before it is built by [32].

In this work, NCsim from Cadence has been utilized to test the functionality of the circuits Figure 6.4).



```
Created probe 1
ncsim> run
Shima your simulation is True
Simulation interrupted at 794970500 NS + 1
ncsim>
```

```
nclaunch> ncsim: 15.20-s010: (c) Copyright 1995-2016 Cadence Design Systems, Inc.
-----------------------------------
Relinquished control to SimVision...
ncsim>
ncsim> source /home/cdsmgr/eda/cadence/2016-17/RHELx86/INCISIVE_15.20.010/tools/inca/files/ncs
ncsim> txe: 15.20-s010: (c) Copyright 1995-2019 Cadence Design Systems, Inc.
```

**Figure 6.2.** Simulation Result at NCsim from Cadence to Test the Functionality

It is good to mention that buffers with equal sizes were utilized at the input and output interface for all implementations in order to achieve electrically equivalent input and output loading conditions. The approach of inserting buffers to isolate fan-in from fan-out is compatible with standard techniques in the literature.

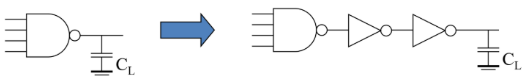


**Figure 6.3.** Buffer Insertion [15]

Buffer insertion separates large fan-in from large $C_L$ with buffers, and it reduces $C_L$ on large fan-in gates [15]. All three organization was proven correct as a result of functional simulations.

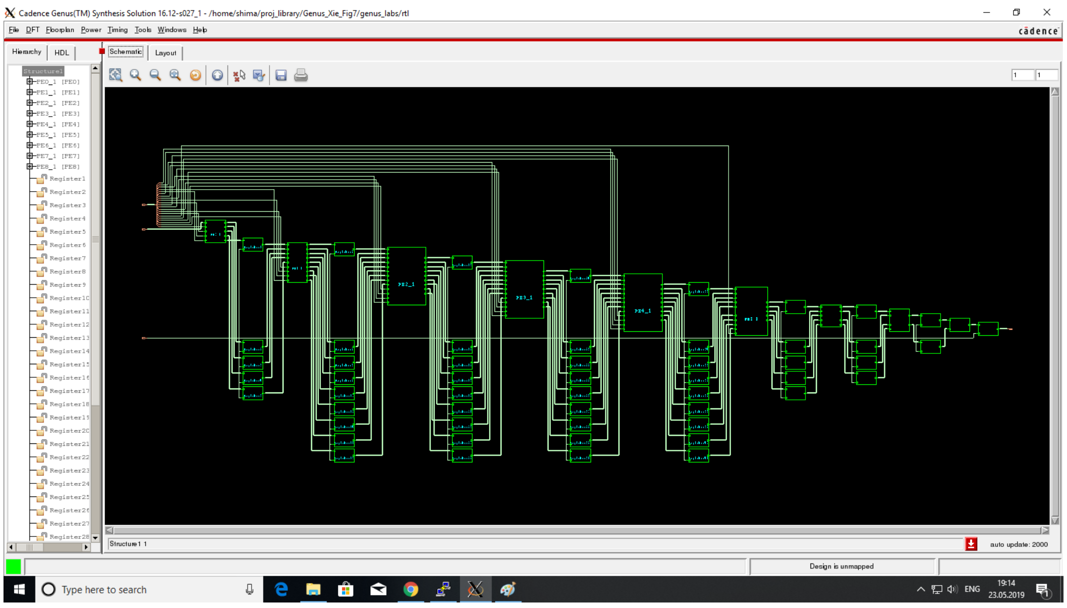
## 6.2 Verification of Synthesis Method

[2] uses 100 MHz frequency and 90nm TSMC library. So to verify the synthesis method, this work was repeated by the author, and all the results were approximately the same as the values presented in the article. The following table shows the comparison synthesis result for $m = 20$ [2] and Genus synthesis solution report.

**Table 6.1.** Synopsys Design Compiler and Genus synthesis Solution for $m = 20$ TSMC 90 nm at 100 MHz

| | Power (mW) | Area ($\mu m^2$) |
|---|---|---|
| Synopsys Design Compiler (paper reported) | 3.893 | 17871 |
| Genus Synthesis Results | 2.842 | 24332.881 |

Figure 6.4 shows the RTL level of [2].



Cadence Genus(TM) Synthesis Solution 16.12-s027_1 - /home/shima/proj_library/Genus_Xie_Fig7/genus_labs/rtl
File DFT Floorplan Power Timing Tools Windows Help
cadence
Hierarchy
HDL
Schematic
Layout
Structure1 1
Design is unmapped
auto update: 2000
Type here to search

**Figure 6.4.** RTL Level in the Genus Synthesis Solution

There are two basic requirements for the flow to be executed on the Verilog design: An Electronic Design Automation (EDA) tool and a standard cell library delivered by ASIC manufacturers. Cadence Inc. is one of the most well-known corporations that develop the EDA tool, and TSMC is an established manufacturing company that provides standard cell libraries. The libraries include precise information about the specifications of the logic blocks, and they carry the wire models compatible with EDA tools as well.

[4] and proposed implementations have been synthesized for a variety of bit lengths $GF(2^{163})$, $GF(2^{268})$, and $GF(2^{562})$ using TSMC 90nm process CMOS standard cell library via Cadence Genus Synthesis Solution with 1.2 GHz clock frequency target. Simulations have been done at the following characteristic conditions: $1.08V$ supply voltage, 125°C temperature, SS (Slow-Slow) process, and WCCOM corner. 0.2 activity factor is assumed by the tool at input nets. Leakage, dynamic power, area, number of sequentials, inverters, buffers, tristates, and logic size were extracted from synthesis, and simulation results were obtained for the compared hardware organizations. The following figure shows the result of the RTL synthesis in the Genus synthesis Solution environment.

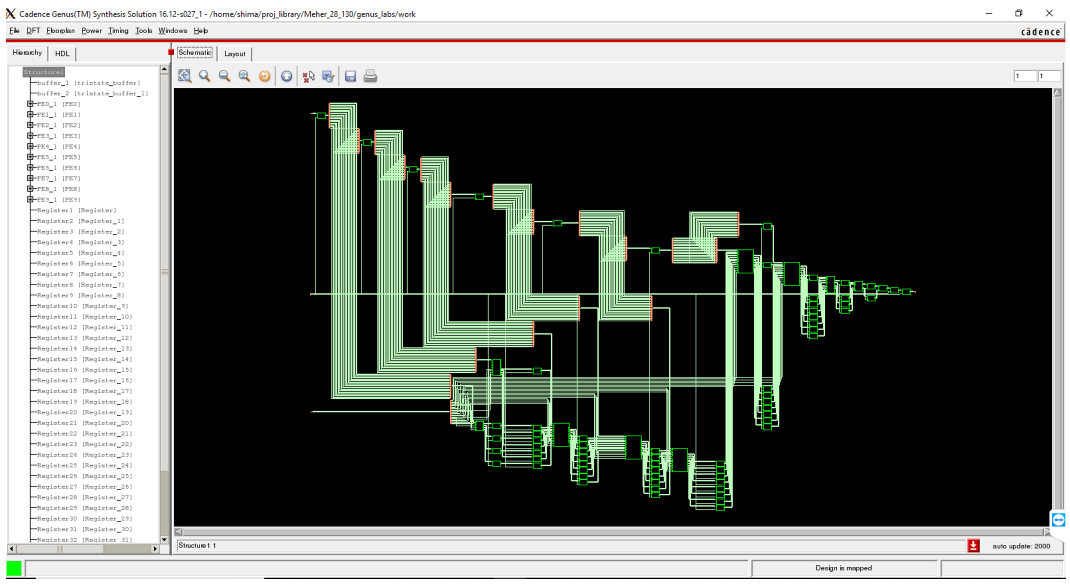


**Figure 6.5.** RTL Level in the Genus Synthesis Solution for [4]

## 6.3 Performance Verification

In order to present the result of the synthesis and simulation, data visualization has been done to observe trends and patterns easily.

### 6.3.1 Critical Path Delay

Time slack is the period of time that an operation can be delayed past its earliest start or earliest finish without limiting the execution. In this work, critical path delay is calculated using time slack from Genus Synthesis Solution, and subtraction of clock period from slack time is computed as a critical path delay.

$$Critical\ Path\ Delay = Clock\ Period - Slack\ Time. \tag{61}$$

The following bar chart shows the critical path slack for the proposed architecture remains stable; however, for implementation [4], this parameter is varying by changing the number of bits. This variation depends on y, which is the number of processing elements in rows.

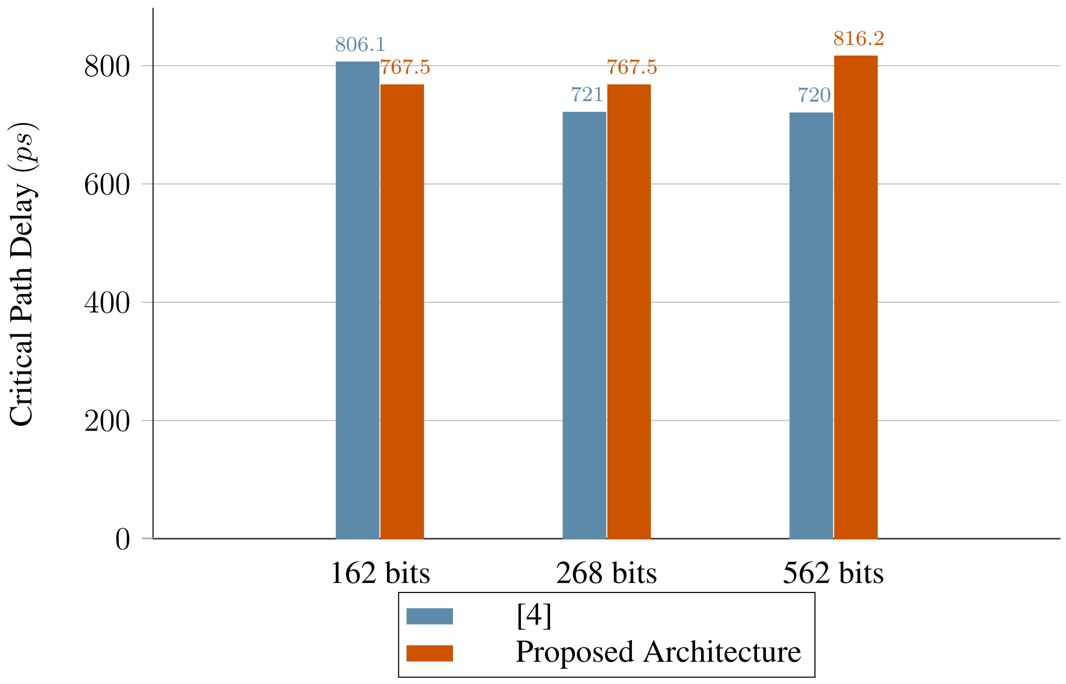


**Figure 6.6.** Comparison of Critical Path Delay for Different Binary Field Size

### 6.3.2 Power

Genus Synthesis Solution utilizes the following formula to compute the total power.

$$TotalPower = weight \times Leakage\,power + (1 - weight) \times Dynamic\,power. \quad (62)$$

Since, in many cases, the circuit is idle, to achieve the best result, the weight is set to a value close to one. In this synthesis, weight is set as 0.99.

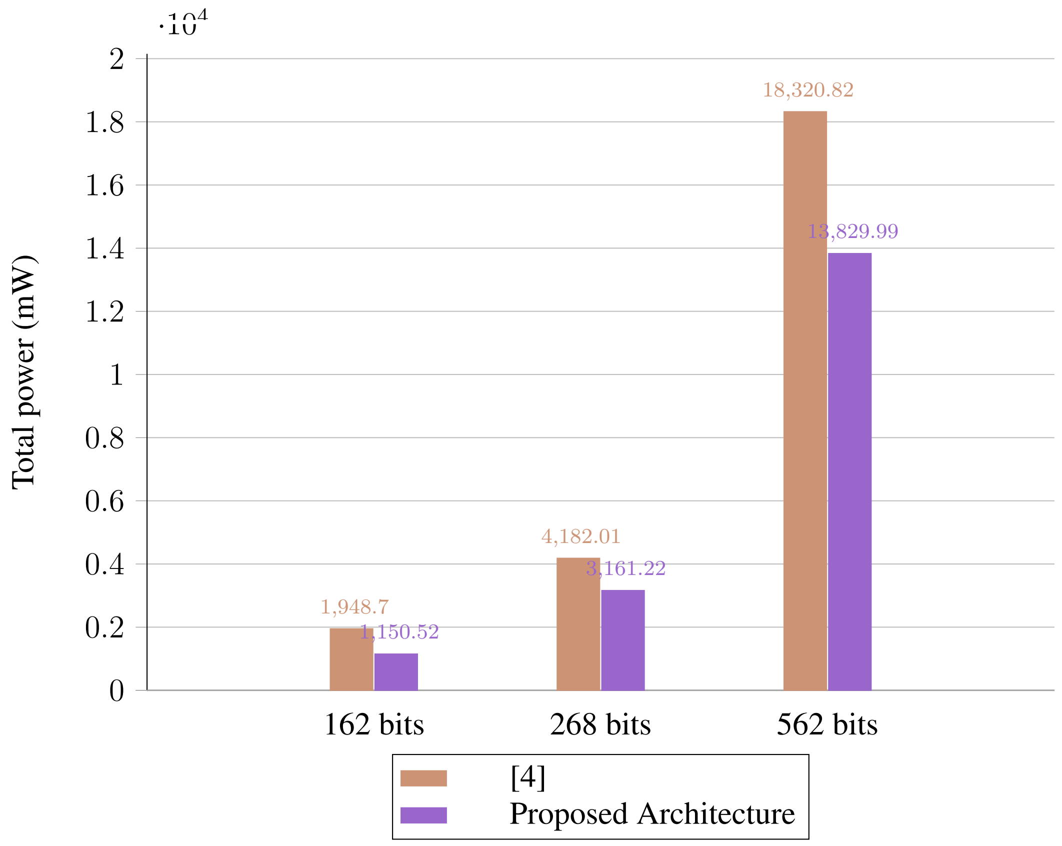


**Figure 6.7.** Comparison of Total Power for Different Binary Field Size

The above bar chart compares the amount to total power between the proposed organization and [4]. It can be seen that the amount of total power is being reduced by 41%, 24%, and 25%, respectively, at three different binary fields size (162, 268, 562 bits).

The power dissipation in a circuit includes two distinct categories; leakage (static) power and dynamic power.

#### 6.3.2.1 Leakage Power

The leakage power is defined as the power that is lost by the cell in a steady-state condition after the vector is applied, and all transistors are stabilized. The leakage power is often assumed in the library as a constant number.

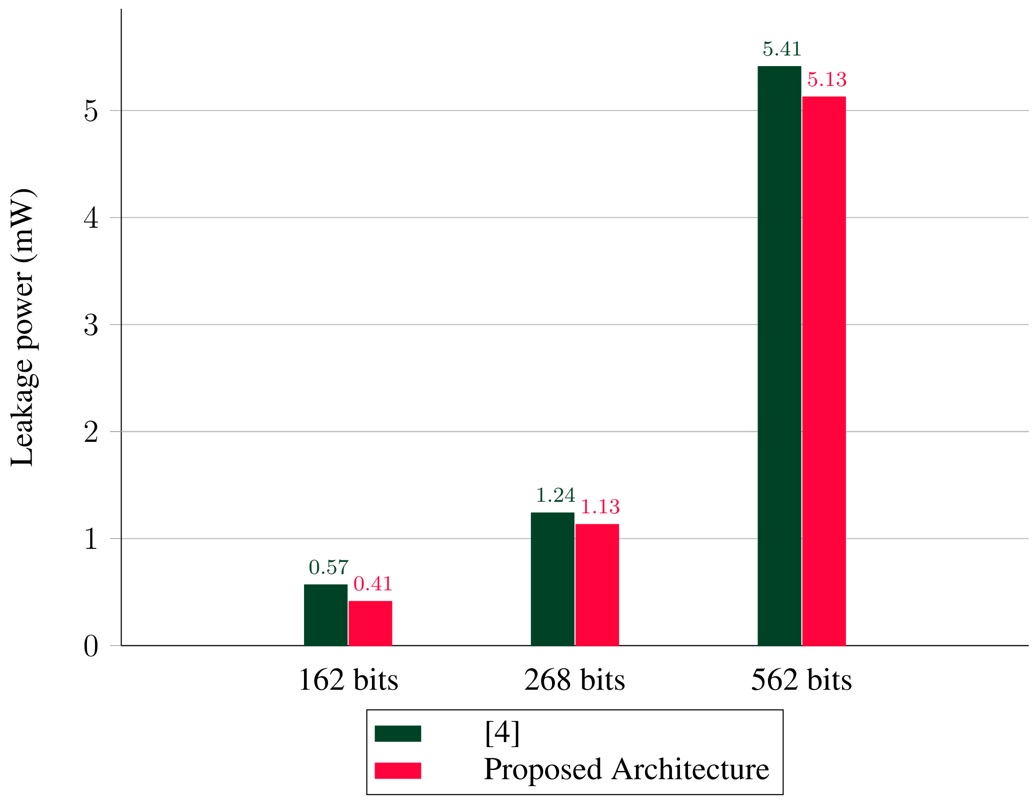


**Figure 6.8.** Comparison of Leakage Power for Different Binary Field Size for 1.2 $GHz$

The above graph shows that the proposed architecture reduced the leakage power consumption by 27%, 9%, and 5%, respectively, at three different binary fields size (162, 268, 562 bits).

As it is shown, as bit length keeps increasing, the leakage power for the proposed organization yields more favorable results.

The simulations were repeated for two other frequencies in order to more accurately characterize critical design parameters such as leakage and $C_{dyn}$

**Table 6.2.** Leakage Power $(mW)$

| | [4] | | | Proposed | | |
|---|---|---|---|---|---|---|
| Mult. Size \ Freq (MHz) | 100 | 800 | 1200 | 100 | 800 | 1200 |
| 162 | 0.395 | 0.400 | 0.567 | 0.289 | 0.370 | 0.412 |
| 268 | 0.991 | 1.210 | 1.239 | 0.792 | 1.016 | 1.131 |
| 562 | 4.681 | 5.373 | 5.410 | 3.382 | 4.785 | 5.126 |

The below table shows the percent of leakage power reduction.

**Table 6.3.** % Leakage Reduction

| Mult. Size \ Freq (MHz) | 100 | 800 | 1200 |
|---|---|---|---|
| 162 | 27 | 8 | 27 |
| 268 | 28 | 16 | 9 |
| 562 | 28 | 11 | 5 |

#### 6.3.2.2 Dynamic Power

The dynamic power dissipation is defined as the power dissipated while a circuit is actively switching at a targeted frequency; this frequency target is set at 1.2 GHz in this thesis in order to (barely) meet speed-path requirements. However, two other frequencies were characterized by simulations.

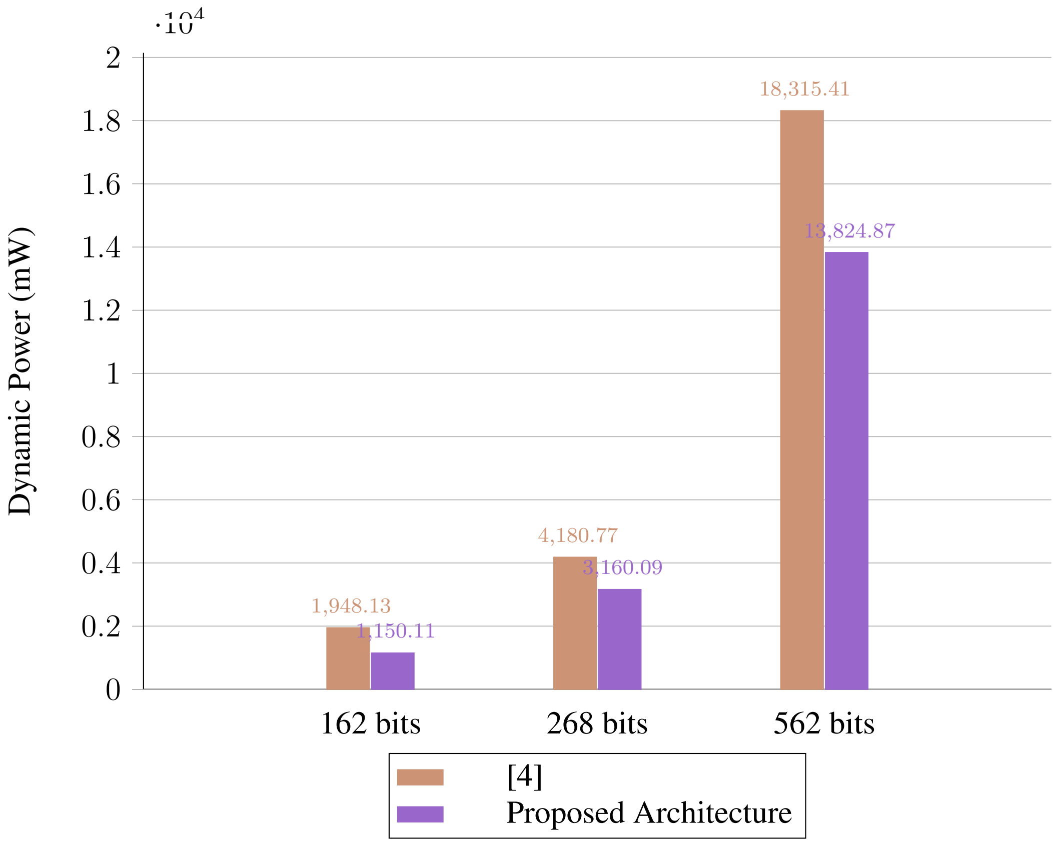


**Figure 6.9.** Comparison of Dynamic Power for Different Binary Field Size

The above figure represents the amount of power dissipation in three different binary fields size (162, 268, 562 bits), which is reduced by 40.96%, 24.41%, and 24.52% respectively while comparing this thesis with [4]. Dynamic capacitance is an essential parameter that reveals the dynamic power characteristics of an implementation running a certain activity.

$$P_{dyn} = C \cdot V^2 . f_{clk}. \tag{63}$$

Therefore,

$$\frac{P_{dyn}}{V^2} = C_{dyn} \cdot f_{clk}. \tag{64}$$

Dynamic capacitance is essentially a property of circuit size and activity, whereas voltage and frequency are operating conditions. If the dynamic capacitance is divided into activity factor, the result will be the basic intrinsic parasitic capacitor in the implementation. Therefore, knowing the dynamic capacitance is important to compare two implementations.

The following table shows the synthesis result of three different frequencies,

**Table 6.4.** Dynamic Power $(mW)$

| | [4] | | | Proposed | | |
|---|---|---|---|---|---|---|
| Freq (MHz) / Mult. Size | 100 | 800 | 1200 | 100 | 800 | 1200 |
| 162 | 293.931 | 1177.590 | 1948.700 | 216.201 | 791.564 | 1150.105 |
| 268 | 697.360 | 3094.669 | 4180.772 | 590.516 | 2173.935 | 3160.085 |
| 562 | 2819.94 | 13653.479 | 18315.409 | 2548.596 | 10058.405 | 13824.866 |

if we try to estimate a linear equation for these curves using the best fitting line technique, we will come up with the following equations in which slope represents dynamic capacitance.

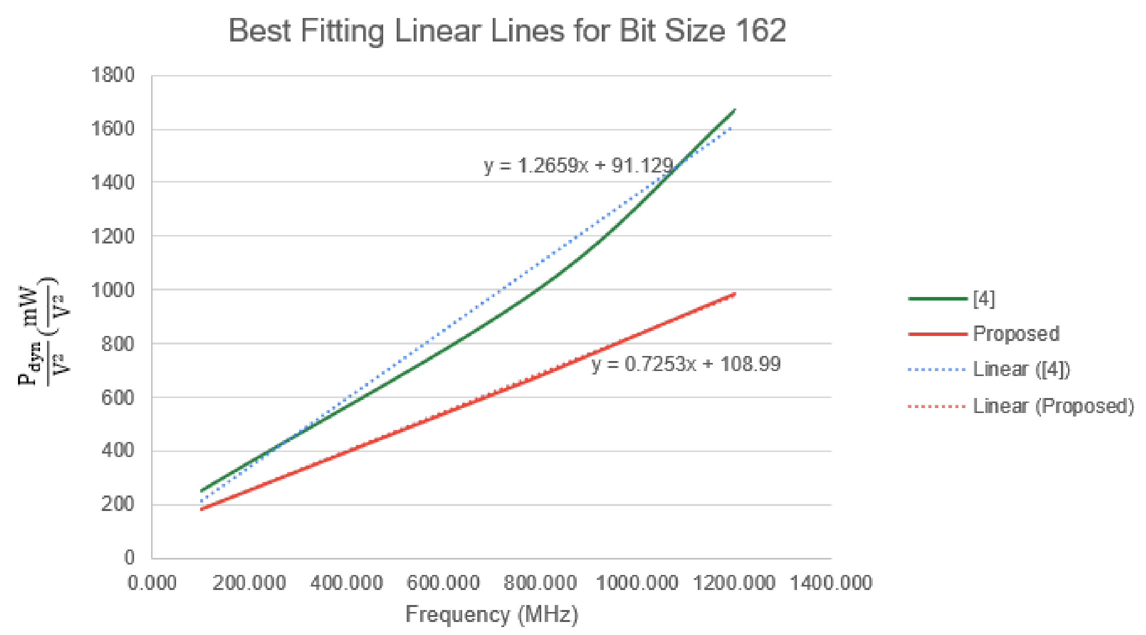


**Figure 6.10.** $\frac{P_{dyn}}{V^2}$ vs. Frequency Curves for Size 162 Used to Extract $C_{dyn}$

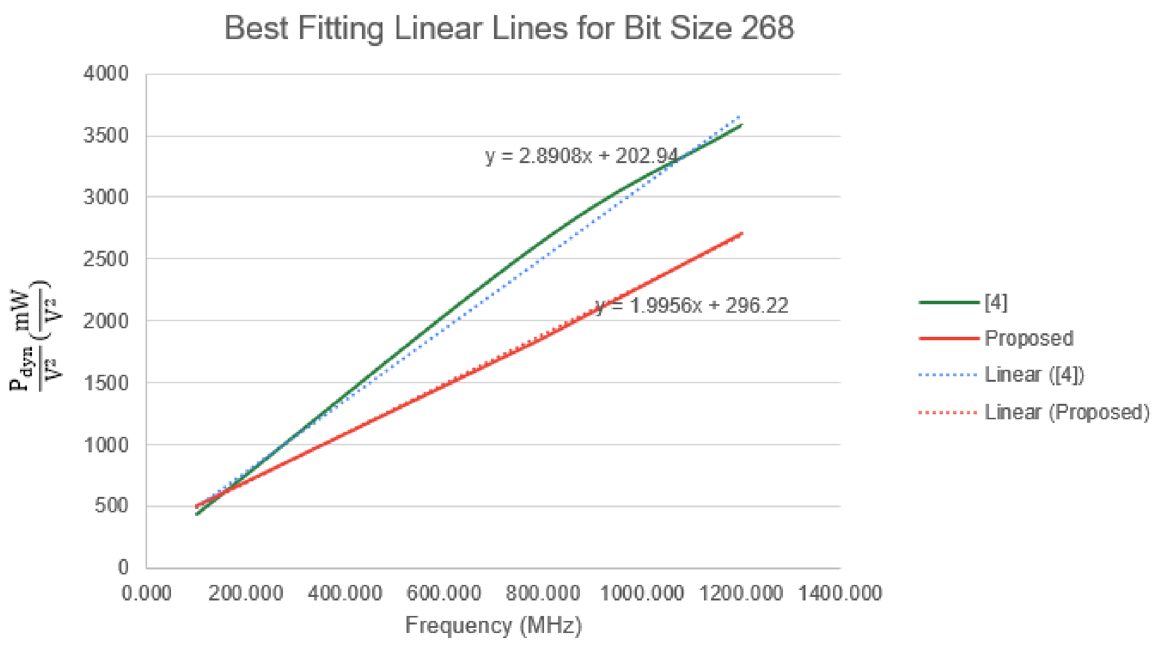


**Figure 6.11.** $\frac{P_{dyn}}{V^2}$ vs. Frequency Curves for Size 268 Used to Extract $C_{dyn}$

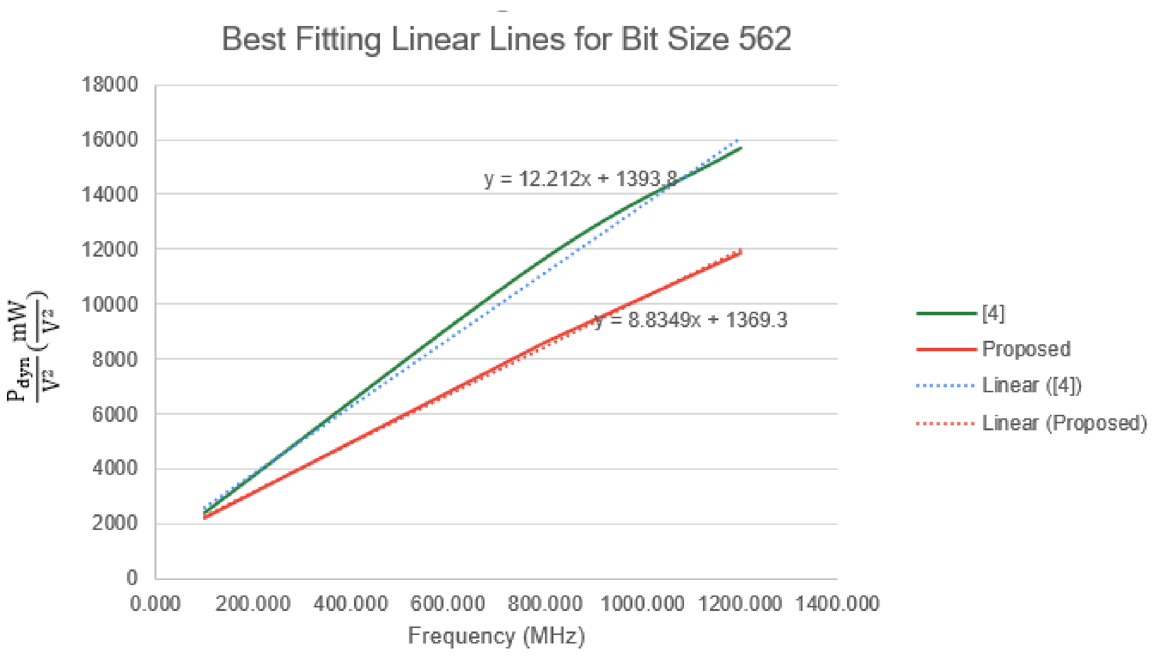


**Figure 6.12.** $\frac{P_{dyn}}{V^2}$ vs. Frequency Curves for Size 562 Used to Extract $C_{dyn}$

**Table 6.5.** Total $C_{dyn}$ $(nF)$

| Mult. Size | [4] | proposed |
|---|---|---|
| 162 | 1.2559 | 0.7253 |
| 268 | 2.8908 | 1.9956 |
| 562 | 12.212 | 8.8349 |

**Table 6.6.** % $C_{dyn}$ Reduction

| Mult. Size | % $C_{dyn}$ reduction |
|---|---|
| 162 | 42 |
| 268 | 31 |
| 562 | 28 |

### 6.3.3 Area

The area falls into two categories, cell area and an estimate of the net area. The following figure compares the total area for three different binary field sizes between the proposed design and [4]. As can be seen, the total area reduced in all three binary field sizes. The total area decreased by 24.75%, 13.76%, and 16.08%, respectively.

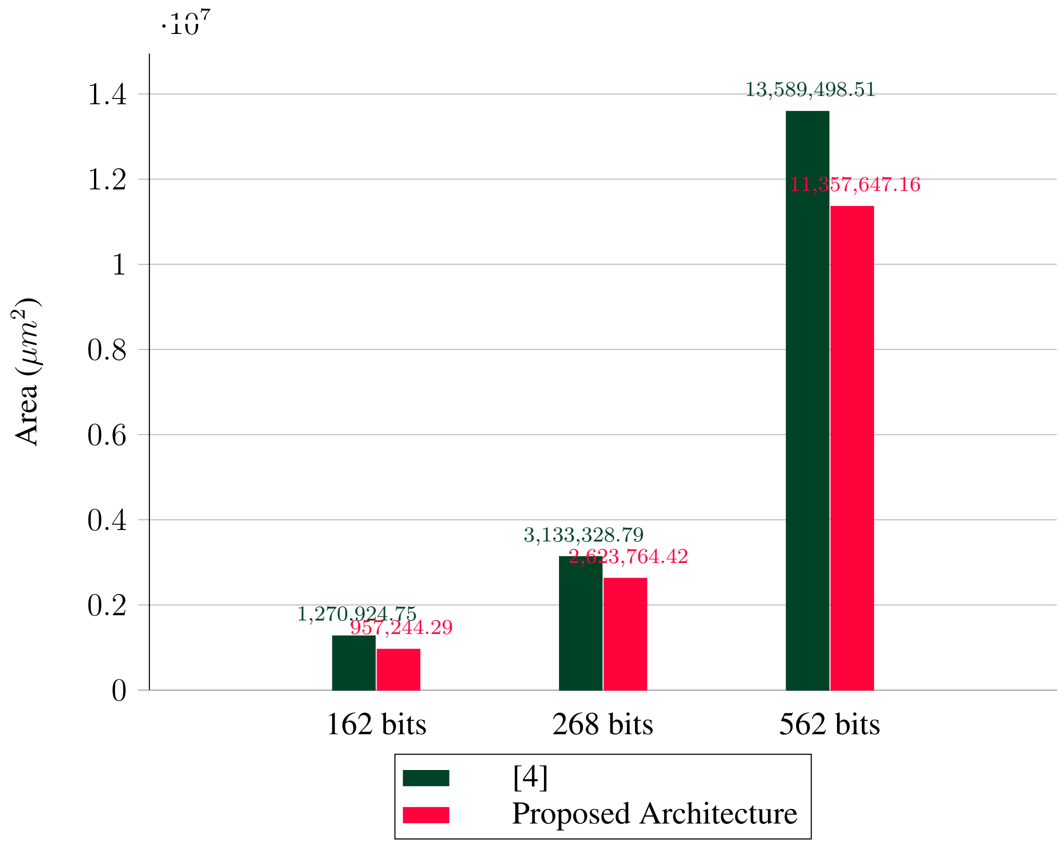


**Figure 6.13.** Comparison of Total Area for Different Binary Field Size

**Table 6.7.** Area ($\mu m^2$)

| Mult. Size \ Freq (MHz) | [4] 100 | [4] 800 | [4] 1200 | Proposed 100 | Proposed 800 | Proposed 1200 |
|---|---|---|---|---|---|---|
| 162 | 1309224.011 | 1308303.909 | 1270924.749 | 985192.402 | 966790.354 | 957244.292 |
| 268 | 3131208.046 | 3138006.601 | 3133328.786 | 2700256.395 | 2649767.892 | 2623764.415 |
| 562 | 13868879.236 | 13587700.973 | 13589498.505 | 11638060.049 | 11321449.567 | 11357647.161 |

**Table 6.8.** % Area Reduction

| Mult. Size \ Freq (MHz) | 100 | 800 | 1200 |
|---|---|---|---|
| 162 | 25 | 26 | 25 |
| 268 | 14 | 16 | 16 |
| 562 | 16 | 16 | 16 |

The cell area and net area comparison bar charts have been shown in the following section. It is worth noting that the most substantial reduction occurs in the cell area since the number of sequential instances is less in the proposed architecture, and this fact is one of the advantages of the proposed design.

#### 6.3.3.1 Cell Area and Net Area

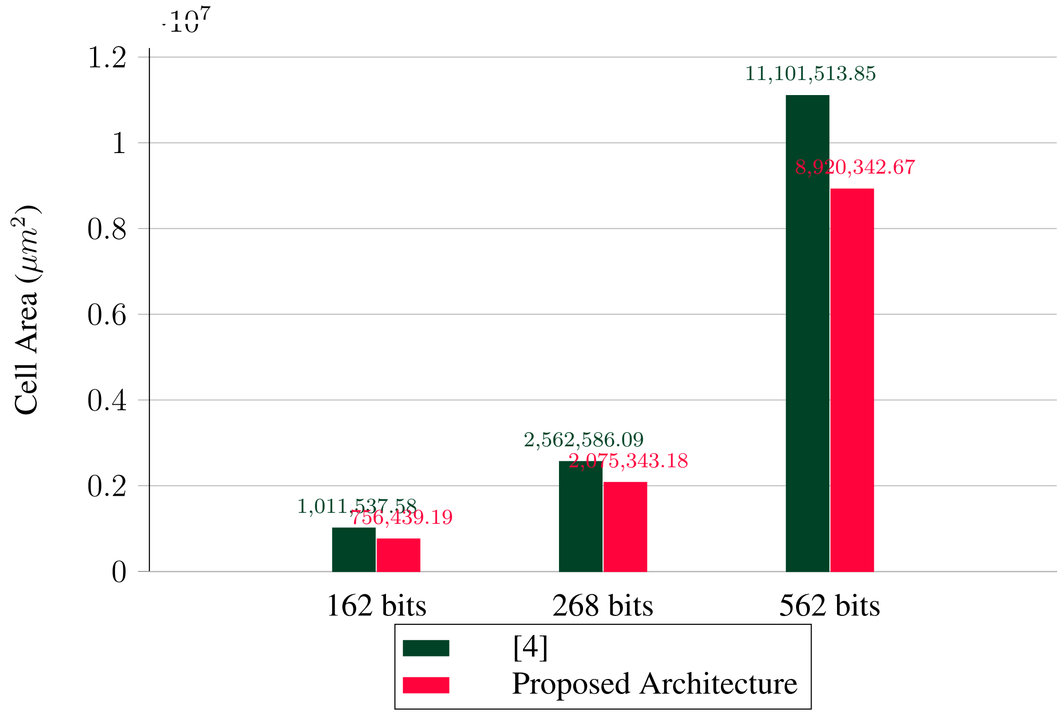


**Figure 6.14.** Comparison of Cell Area for Different Binary Field Size

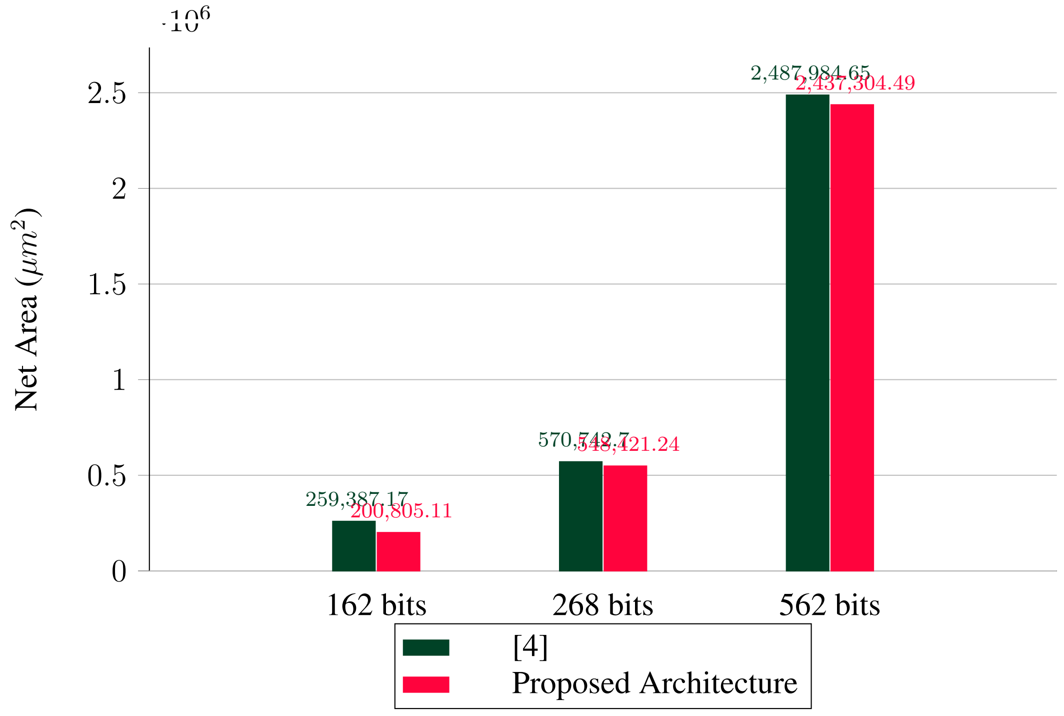


**Figure 6.15.** Comparison of Net Area for Different Binary Field Size

## 6.4 Net Report & Gate Level Details

Genus synthesis solution reports each net information in detail. The net report tables show a summary of approximately 30000 lines of the net report for each circuit. This information is calculated by using Cap-Table.

Some of the terms in the tables are discussed here.
**Load**:
The load of a net refers to the wires that can carry power to other devices further along the circuit.
**Wire Deliver**:
The wire deliver refers to the wires that deliver power from the source to a system.
**Wire Capacity**:
The wire capacity is determined by how much energy it can dissipate and in continuous operation, whether it causes problems or not.
**Wire Load Resistor**:
More wire load resistance results in more heat dissipation.

Since the number of D flip flops is decreased with the proposed topology, the number of loads for the clock signal is reduced in all the three different binary field sizes. The wire capacity of design in [4] for input A in these different field size is lower than the proposed design since input A needs to provide data for the processing elements in each row (y) and input A propagates through all design via D flip flops but in the proposed it needs to provide the values to all of the $m$ gates. Net report details will be discussed in the following sections.

There are clear benefits of the proposed AOP hardware organization. An advantage of the proposed scheme is the reduced register count (D flip-flop), which directly affects the area and power dissipation. Each implementation also has a somewhat distinct internal signal fan-out profile, which also influences the final power dissipation and area.

### 6.4.1 Net Report for AOP Multipliers over $GF(2^{162})$

**Table 6.9.** Net Report for AOP Multipliers over $GF(2^{162})$

| Structure | Type of net | Load | Wire Deliver | Wire Capacity $(fF)$ | Wire Load Resistor $(k\Omega)$ |
|---|---|---|---|---|---|
| [4] | Input A after Buffer | 2 | 1 | 33 | 0.100 |
| | Input B | 1 | 1 | 270.2 | 0.821 |
| | Internal Net | 1 | 1 | 3.4 | 0.010 |
| | CLK | 352 | 1 | 0 | 0 |
| | Enable | 2 | 1 | 538.7 | 1.636 |
| This work | Input A after buffer | 1 | 1 | 268.6 | 0.816 |
| | Input B | 1 | 1 | 270.2 | 0.821 |
| | Internal Net | 1 | 1 | 3.4 | 0.010 |
| | CLK | 243 | 1 | 0 | 0 |
| | Enable | 2 | 1 | 538.7 | 1.636 |

The wire capacity and resistor values for input B are equal for both designs. At the proposed model, wire capacity and wire load resistor is the same for both input A and B since it provides data once and at the same time to $m$ gates.

### 6.4.2 Gate Level Details for AOP Multipliers over $GF(2^{162})$

Table 6.10 and 6.11 indicate the number of D flip flops is decreased in the proposed approach, and it can be implicit that for both organizations, it is possible to use a tri-state buffer with high enable. The tri-state buffer is used in this circuit as they allow multiple logic devices to be attached to the same wire or bus without damage or loss of data [65]. Tri-state buffers are used as an insertion buffer in the input and output interface of the circuit.

**Table 6.10.** Gate Level Details for AOP Multipliers over $GF(2^{162})$ based on [4]

| Gate | Instances | Area ($\mu m^2$) | Leakage Power ($mW$) | Internal Power ($mW$) |
|---|---|---|---|---|
| 2-Input AND | 26406 | 93160.368 | 0.030 | 17.030 |
| 3-State Buffer | 326 | 2300.256 | 0.001 | 0.520 |
| D flip flop sync clear, single output | 3097 | 41519.621 | 0.017 | 53.282 |
| D flip flop single output | 54279 | 689386.723 | 0.453 | 1411.582 |
| 2-input exclusive OR | 26243 | 185170.608 | 0.066 | 93.471 |

**Table 6.11.** Gate Level Details for AOP Multipliers over $GF(2^{162})$ Proposed Organization

| Gate | Instances | Area ($\mu m^2$) | Leakage Power ($mW$) | Internal Power ($mW$) |
|---|---|---|---|---|
| 2-Input AND | 26406 | 93160.368 | 0.029 | 11.198 |
| 3-state buffer | 163 | 1150.128 | 0.000 | 0.371 |
| 3-state buffer (higher driving strength) | 163 | 1495.166 | 0.001 | 0.432 |
| D flip-flop | 39609 | 531014.098 | 0.339 | 904.533 |
| 2-Input exclusive OR | 163 | 1150.128 | 0.000 | 0.678 |
| 3-Input exclusive OR | 13040 | 156417.408 | 0.043 | 32.062 |

### 6.4.3 Net Report for AOP Multipliers over $GF(2^{268})$

**Table 6.12.** Net Report for AOP Multipliers over $GF(2^{268})$

| Structure | Type of net | Load | Wire deliver | Wire Capacity $(fF)$ | Wire Load Resistor $(k\Omega)$ |
|---|---|---|---|---|---|
| [4] | Input A after Buffer | 34 | 1 | 57.7 | 0.175 |
| | input B | 269 | 1 | 444.8 | 1.351 |
| | Internal Net | 1 | 1 | 3.4 | 0.010 |
| | CLK | 156020 | 1 | 0 | 0 |
| | Enable | 2 | 1 | 5 | 0.015 |
| This work | Input A after buffer | 1 | 1 | 443.1 | 1.346 |
| | Input B | 1 | 1 | 444.8 | 1.351 |
| | Internal Net | 1 | 1 | 3.4 | 0.010 |
| | CLK | 405 | 1 | 0 | 0 |
| | Enable | 2 | 1 | 887.8 | 2.696 |

As can be argued from the table, the load of the clock increased significantly; however, the advantage of the proposed work is that the number of clocks does not increase as much as the [4] architecture.

### 6.4.4 Gate Level Details for AOP Multipliers over $GF(2^{268})$

**Table 6.13.** Gate Level Details for AOP Multipliers over $GF(2^{268})$ based on [4]

| Gate | Instances | Area ($\mu m^2$) | Leakage Power ($mW$) | Internal Power ($mW$) |
|---|---|---|---|---|
| Non-inverting buffer | 1 | 14.818 | 0.000 | 0.007 |
| 3-State buffer | 269 | 1898.064 | 0.001 | 0.449 |
| 3-State buffer (higher driving strength) | 269 | 1898.064 | 0.001 | 0.418 |
| Clock buffer with balanced fall/rise time | 51 | 575.770 | 0.001 | 0.173 |
| D flip flop | 15333 | 216379.296 | 0.133 | 346.169 |
| D flip flop sync clear, single output | 72092 | 966494.189 | 0.365 | 1270.414 |
| D flip flop single output | 64880 | 824027.904 | 0.0533 | 1622.969 |
| D flip flop single output (higher driving strength) | 1832 | 24560.525 | 0.017 | 38.873 |
| Inverter | 1884 | 3988.051 | 0.001 | 0.383 |
| Inverter (higher driving strength) | 8 | 22.579 | 0.000 | 0.003 |
| Scan D flip flop | 1883 | 29230.186 | 0.012 | 47.005 |
| 2-Input Exclusive NOR | 16947 | 119578.032 | 0.044 | 50.528 |
| 2-Input exclusive OR | 52993 | 373918.608 | 0.134 | 104.482 |

One of the crucial details derived from Table 6.13 is that the proposed organization used the exclusive NOR gate rather than the AND gates.

**Table 6.14.** Gate Level Details for AOP Multipliers over $GF(2^{268})$ proposed

| Gate | Instances | Area ($\mu m^2$) | Leakage Power ($mW$) | Internal Power ($mW$) |
|---|---|---|---|---|
| 2-Input AND | 72092 | 254340.576 | 0.078 | 29.223 |
| 3-State buffer | 269 | 1898.064 | 0.001 | 0.619 |
| Non-inverting buffer | 269 | 3985.934 | 0.004 | 1.222 |
| D flip flop async clear and set | 108945 | 1460560.248 | 0.927 | 3179.091 |
| 2-input exclusive OR | 269 | 1898.064 | 0.001 | 1.115 |
| 3-input exclusive OR | 35777 | 429152.270 | 0.117 | 88.471 |

### 6.4.5 Net Report for AOP Multipliers over $GF(2^{562})$

**Table 6.15.** Net Report for AOP Multipliers over $GF(2^{562})$

| Structure | Type of net | Load | Wire deliver | Wire Capacity ($fF$) | Wire Load Resistor ($k\Omega$) |
|---|---|---|---|---|---|
| [4] | Input A after Buffer | 1 | 1 | 3.4 | 0.100 |
| | Input B | 563 | 1 | 929 | 2.822 |
| | Internal Net | 1 | 1 | 3.4 | 0.010 |
| | CLK | 675037 | 1 | 0 | 0 |
| | Enable | 2 | 1 | 5 | 0.015 |
| This work | Input A after buffer | 1 | 1 | 16.6 | 0.050 |
| | Input B | 1 | 1 | 3.4 | 0.0100 |
| | Internal Net | 1 | 1 | 3.4 | 0.010 |
| | CLK | 477424 | 1 | 0 | 0 |
| | Enable | 515 | 1 | 850.0 | 2.581 |

As it can be seen, the load of "enable net" in the proposed architecture is more than [4], since the proposed architecture is utilizing the inverting clock buffer with balance fall or rising edge.

### 6.4.6 Gate Level Details for AOP Multipliers over $GF(2^{562})$

**Table 6.16.** Gate Level Details for AOP Multipliers over $GF(2^{562})$ based on [4]

| Gate | Instances | Area ($\mu m^2$) | Leakage Power ($mW$) | Internal Power ($mW$) |
|---|---|---|---|---|
| Non-Inverting Buffer | 1 | 13.406 | 0.000 | 0.006 |
| 3-state Buffer | 563 | 3972.528 | 0.001 | 0.857 |
| 3-state Buffer (higher driving strength) | 563 | 3972.528 | 0.002 | 0.851 |
| Clock buffer with balanced fall/raise time | 2337 | 14840.885 | 0.020 | 2.937 |
| D flip flop | 24209 | 341637.408 | 0.209 | 539.335 |
| D flip flop sync clear | 316406 | 4241865.398 | 1.604 | 5573.297 |
| D flip flop async clear and set | 330025 | 4191581.520 | 2.705 | 82.96.830 |
| D flip flop, single output | 3271 | 50776.387 | 0.050 | 80.719 |
| Inverter | 1127 | 2385.634 | 0.001 | 0.183 |
| Inverter (highest driving strength) | 12 | 33.869 | 0.000 | 0.005 |
| Inverter (higher driving strength) | 2815 | 7945.056 | 0.010 | 1.386 |
| Inverter (higher driving strength) | 3 | 16.934 | 0.000 | 0.004 |
| Scan D flip flop | 1126 | 17479.123 | 0.007 | 26.819 |
| 2-input Exclusive NOR | 25335 | 178763.760 | 0.065 | 73.852 |
| 2-input exclusive OR | 289382 | 2041879.392 | 0.730 | 623.103 |

In [4], scan D-flip flop is used, for which a multiplexer is added at the input of the flip flop with D input and SI input (Figure 6.17). In this design, the Genus synthesis solution used scan D flip flop instead of generic D flip flop. The synthesizer found it useful to use a multiplexer followed by a flip-flop when trying to map the code to the netlist. The multiplexer was useful in a larger multiplier because it can cause area saving, but it can cause speed loss [66].

**Table 6.17.** Gate Level Details for AOP Multipliers over $GF(2^{562})$ proposed

| Gate | Instances | Area ($\mu m^2$) | Leakage Power ($mW$) | Internal Power ($mW$) |
|---|---|---|---|---|
| Non-inverting buffer | 4504 | 12712.090 | 0.005 | 2.296 |
| 3-state buffer | 1126 | 7945.056 | 0.002 | 1.894 |
| Clock buffer (balance rise/fall time) | 5 | 45.864 | 0.000 | 0.014 |
| Inverting clock buffer (balance rise/fall time) | 4504 | 19068.134 | 0.016 | 3.559 |
| D flip flop single output | 477424 | 6063666.739 | 4.084 | 10859.333 |
| Inverter (higher driving strength) | 562 | 8327.491 | 0.015 | 2.296 |
| Inverter | 563 | 1589.011 | 0.001 | 0.219 |
| 2-Input NOR | 316406 | 893024.294 | 0.142 | 54.224 |
| 2-Input exclusive NOR | 17453 | 123148.368 | 0.44 | 45.810 |
| 2-Input exclusive NOR (higher driving strength) | 563 | 3972.528 | 0.003 | 0.804 |
| 2-Input exclusive OR | 563 | 3972.528 | 0.001 | 1.921 |
| 3-Input exclusive OR | 59115 | 709096.248 | 0.200 | 198.957 |
| 3-Input exclusive OR (higher driving strength) | 89517 | 1073774.318 | 0.611 | 221.067 |

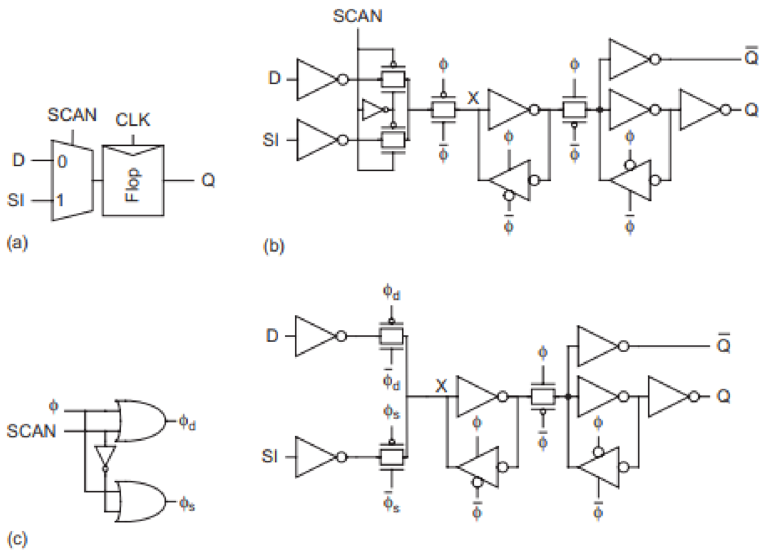


**Figure 6.16.** Scannable Flip Flops [16]

In the proposed architecture for 562-bit length, since the circuit has a tapering clock path, it is advantageous to use equal rise and fall delay time to maintain the initial duty cycle and to ensure that there is no clock signal overlap due to any difference in propagation delays. This overlapping clock signal becomes more critical when dealing with very high-speed ASIC designs. These buffers are known as a clock or balance buffers and have different attributes from the normal buffers in the standard cell library. It is necessary to correctly use balance buffers or inverters during a clock tree synthesis, especially when addressing the requirements of high-speed clocking (small clock pulse width) in a large clock tree [67].

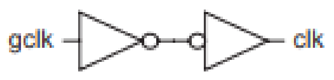


**Figure 6.17.** Clock Buffer [16]

Additionally, the Genus synthesis solution uses XNOR and inverters instead of AND gate for [4], and for proposed architecture, it uses NOR gates. Figure 6.18 indicates the implementation of the AND function using NOR gates.

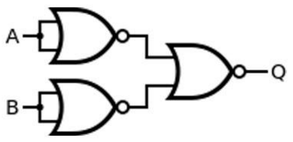


**Figure 6.18.** AND Implementation via NOR Gates

## 6.5 Benchmarking against Literature

The following table shows the result of synthesis for the recent state of the art and proposed architecture through in-depth details.

**Table 6.18.** Synthesis Results from Compared Organizations for AOP Multipliers over $GF(2^m)$ in 90 $nm$

| m | Structure | $y^1$ | Critical path Delay $(ps)$ | Leakage Power $(mW)$ | Dynamic Power $(mW)$ | Total Area $(\mu m^2)$ | Latency$^2$ |
|---|---|---|---|---|---|---|---|
| 162 | [4] | 16 | 806.1 | 0.567 | 1948.133 | 1270924.749 | 15 |
| | This Work | Not Applicable | 767.5 | 0.412 | 1150.105 | 957244.292 | 7 |
| 268 | [4] | 31 | 721 | 1.239 | 4180.772 | 3133328.786 | 15 |
| | This Work | Not Applicable | 767.5 | 1.131 | 3160.085 | 2623764.415 | 7 |
| 562 | [4] | 63 | 720 | 5.410 | 18315.409 | 13589498.505 | 15 |
| | This Work | Not Applicable | 816.2 | 5.126 | 13824.866 | 11357647.161 | 8 |

$^1$ $y$ is the number of processing elements in rows.

$^2$The number of clock cycles to compute a multiplication.

Genus Synthesis Solution optimizes power dissipation by using smaller library components and tries to meet the timing requirement at the same time. It achieves target frequency with some accuracy and reliability, and it is expected that users will not always have the same critical path delay.

**Table 6.19.** Synthesized Component Types from Compared Organizations for AOP Multipliers over $GF(2^m)$ in 90 $nm$

| m | Structure | Sequential | Tristate | Logic | Inverter | Buffer |
|---|---|---|---|---|---|---|
| 162 | [4] | 57376 | 326 | 52649 | 0 | 0 |
|  | This Work | 39609 | 326 | 39609 | 0 | 0 |
| 268 | [4] | 156020 | 538 | 69940 | 1892 | 52 |
|  | This Work | 108945 | 538 | 108138 | 0 | 0 |
| 562 | [4] | 675037 | 1126 | 314717 | 7527 | 1 |
|  | This Work | 477424 | 1126 | 483617 | 0 | 0 |

As discussed in the previous chapter, the theoretical number of registers was expected to reduce 50% in the proposed organization. It can be observed that in practice, the number of sequentials reduced by about 30% on the average. Additionally, a lower number of logic elements in this work are because of a reduction in the number of XORs. The total number of AND and XOR gates in Table 5.2 corresponds to the number of logic instances in Table 6.19, as expected.

# CHAPTER 7

# CONCLUSIONS AND FUTURE WORK

## 7.1 Conclusions

In this thesis, the existing state of the art $GF(2^m)$ multiplier algorithm based on AOP is studied. An alternative hardware organization is proposed to optimize power and area, compared to the previous systolic array design approaches. The proposed scheme moves away from distributing reduction, multiplication, and accumulation steps to systolic processing elements, but separates these steps into distinct stages. The proposed architecture has been implemented for a variety of bit lengths $GF(2^{162})$, $GF(2^{268})$, and $GF(2^{562})$ along with state of the art systolic array recently reported in the literature. Equally sized buffers were utilized at the input and output interface for both implementations in order to achieve electrically equivalent input and output loading conditions. Both designs were implemented in structural Verilog HDL at the register transfer level (RTL). The resulting implementation thus achieves power and area reductions for the same frequency, compared to the best known previous AOP implementation in the literature. Among the three relevant and different bit lengths compared, on the average 25%, 17%, and 18% reduction in leakage, dynamic power, and the area is achieved, respectively, with 1.2 GHz target frequency.

## 7.2 Future Work

The following ideas can be studied in the future:

1. The digital system can reduce the power consumption at different design flow steps, and the digital IC implementation includes the following steps:

(a) RTL Design and Verification,

(b) Synthesis,

(c) Physical (place and route),

(d) Sign off.

In this thesis, RTL design and verification for the last state of the art and proposed is done by NC sim, and synthesis is done via Genus Synthesis solution. The place and route and sign off steps can be done in the future by Innovous for a physical implementation and Assura for physical verification, which are other powerful tools from Cadence [58].

2. As AOP multiplication can be used as a core of trinomial and pentonomial circuits for future work, it would be good to try the proposed approach as the core of these circuits and analyze against best implementations in the literature for trinomial/pentonomial circuits.

3. Low power design of multipliers on the other basis may be investigated.

4. As multiplication is utilized for division and inverse circuits, such circuits could be added to the analysis.

5. Further study can be carried out on the design of Nano-Electro-Mechanical (NEM) relays as energy-saving devices for finite field multipliers. Newly proposed NEM relay technology render ultra-low-power IC's [68],[69]. The relay is an ideal switch as it possesses ideal on or off switching behavior and zero off-state leakage current, so its supply voltage can be decreased to close to zero in theory. This technology can, therefore, possibly overcome CMOS technology's energy-efficiency limits [70].

# REFERENCES

[1] A. Allan, D. Edenfeld, W. H. Joyner, A. B. Kahng, M. Rodgers, and Y. Zorian, "2001 technology roadmap for semiconductors," *Computer*, vol. 35, no. 1, pp. 42–53, 2002.

[2] J. Xie, P. K. Meher, and J. He, "Low-Complexity Multiplier for $GF(2^m)$ Based on All-One Polynomials," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, pp. 168–173, Jan 2013.

[3] P. Chen, S. N. Basha, M. Mozaffari-Kermani, R. Azarderakhsh, and J. Xie, "FPGA Realization of Low Register Systolic All-One-Polynomial Multipliers Over $GF(2^m)$ and Their Applications in Trinomial Multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, pp. 725–734, Feb 2017.

[4] P. K. Meher and X. Lou, "Low-Latency, Low-Area, and Scalable Systolic-Like Modular Multipliers for $GF(2^m)$ Based on Irreducible All-One Polynomials," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, pp. 399–408, Feb 2017.

[5] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, "Key challenges in cloud computing: Enabling the future internet of services," *IEEE Internet Computing*, vol. 17, no. 4, pp. 18–25, 2013.

[6] K. B. Leboeuf, *GPU and ASIC Acceleration of Elliptic Curve scalar point Multiplication*. PhD thesis, University of Windsor, 2012.

[7] F. Gebali and T. Al-Somani, "Finite field multiplication using reordered normal basis multiplier," in *2011 International Conference on Broadband and Wireless Computing, Communication and Applications*, pp. 320–326, 2011.

[8] F. J. Pollack, "New microarchitecture challenges in the coming generations of cmos process technologies (keynote address)(abstract only)," in *the 32Nd An-*

*nual ACM/IEEE International Symposium on Microarchitecture, Washington, DC, USA*, p. 2, 1999.

[9] https://cs.nyu.edu/ lerner/spring10/projects/multicore-niagara.pdf.

[10] W. Arden, "Roadmap key challenges," *Materials Today*, vol. 6, no. 5, pp. 40–44, 2003.

[11] S. Hashemi Namin, *Low power finite field multiplication with wireless security application*. PhD thesis, University of Windsor, 2015.

[12] Yeh, Reed, and Truong, "Systolic multipliers for finite fields $gf(2^m)$," *IEEE Transactions on Computers*, vol. C-33, no. 4, pp. 357–360, 1984.

[13] W. A. Stein, "CoCalc - Collaborative Calculation and Data Science." `https://cocalc.com/`. [Online; accessed 18-June-2020].

[14] K. P. Gnawali, *A Training Manual on Digital Design using VHDL*. 03 2016.

[15] A. C. Jan M. Rabaey and B. Nikolic, *Digital Integrated Circuits: A Design Perspective*. Prentice Hall, 2002.

[16] H. Weste, *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley, 2011.

[17] M. A. Caloyannides, "Encryption wars: early battles," *IEEE Spectrum*, vol. 37, no. 4, pp. 37–43, 2000.

[18] A. Stanoyevitch, *Introduction to Cryptography with Mathematical Foundations and Computer Implementations*. Chapman & Hall/CRC, 2011.

[19] E. Berlekamp, "Bit-serial reed - solomon encoders," *IEEE Transactions on Information Theory*, vol. 28, pp. 869–874, November 1982.

[20] A. J. Menezes, I. F. Blake, X. Gao, R. C. Mullin, S. A. Vanstone, and T. Yaghoobian, *Applications of finite fields*. The Kluwer international series in engineering and computer science, 1993.

[21] P. Kumar, A. Gurtov, J. Iinatti, M. Sain, and P. H. Ha, "Access control protocol with node privacy in wireless sensor networks," *IEEE Sensors Journal*, vol. 16, pp. 8142–8150, Nov 2016.

[22] T. R. Halford, T. A. Courtade, K. M. Chugg, X. Li, and G. Thatte, "Energy-efficient group key agreement for wireless networks," *IEEE Transactions on Wireless Communications*, vol. 14, pp. 5552–5564, Oct 2015.

[23] X. Le, S. Lee, I. Butun, M. Khalid, R. Sankar, M. Kim, M. Han, Y. Lee, and H. Lee, "An energy-efficient access control scheme for wireless sensor networks based on elliptic curve cryptography," *Journal of Communications and Networks*, vol. 11, pp. 599–606, 12 2009.

[24] H. Du, Q. Wen, and S. Zhang, "An efficient certificateless aggregate signature scheme without pairings for healthcare wireless sensor network," *IEEE Access*, vol. 7, pp. 42683–42693, 2019.

[25] C. Paar and J. Pelzl, *Understanding Cryptography - A Textbook for Students and Practitioners.* Springer Verlog, 2010.

[26] E. Savaş and K. Koç, "Finite field arithmetic for cryptography," *IEEE Circuits and Systems Magazine*, vol. 10, no. 2, pp. 40–56, 2010.

[27] I. S. Hsu, T. K. Truong, L. J. Deutsch, and I. S. Reed, "A comparison of vlsi architecture of finite field multipliers using dual, normal, or standard bases," *IEEE Transactions on Computers*, vol. 37, pp. 735–739, June 1988.

[28] J.-P. Deschamps, J. L. Imaña, and G. D. Sutter, *Hardware Implementation of Finite-Field Arithmetic.* McGraw-Hill, Inc., 2009.

[29] C. Negre, "Quadrinomial modular arithmetic using modified polynomial basis," in *International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, vol. 1, pp. 550–555 Vol. 1, April 2005.

[30] C. K. Koc and B. Sunar, "Low-complexity bit-parallel canonical and normal basis multipliers for a class of finite fields," *IEEE Transactions on Computers*, vol. 47, no. 3, pp. 353–356, 1998.

[31] B. Walsh, "The Surprisingly Large Energy Footprint of the Digital Economy [UPDATE]." `https://science.time.com`, 2013. [Online; accessed 19-Feb-2020].

[32] S. Harris and D. Harris, *Digital Design and Computer Architecture ARM Edition*. Morgan Kaufmann, 2015.

[33] D. D. Gajski, *Principles of Digital Design*. Prentice-Hall, Inc, 1997.

[34] "United States Environmental Protection Agency." `https://www.epa.gov/energy`. [Online; accessed 30-June-2020].

[35] "BlueSkyModel." `https://blueskymodel.org/kilowatt-hour`. [Online; accessed 30-June-2020].

[36] D. Chinnery and K. Keutzer, *Closing the Power Gap Between ASIC Custom Tools and Techniques for Low Power Design*. Springer Science+Business Media, LLc, 2007.

[37] J. Navarro-Ortiz, N. Chinchilla-Romero, J. J. Ramos-Munoz, and P. Munoz-Luengo, "Improving hardware security for lorawan," in *2019 IEEE Conference on Standards for Communications and Networking (CSCN)*, pp. 1–6, 2019.

[38] T. Oliveira, J. López, D. Cervantes-Vázquez, and F. Rodríguez-Henríquez, "Koblitz curves over quadratic fields," *Journal of Cryptology*, vol. 32, 04 2018.

[39] D. W. Hardy, F. Richman, and C. L. Walker, *Applied Algebra: Codes, Ciphers and Discrete Algorithms*. Chapman and Hall/CRC, 2009.

[40] T. Itoh and S. Tsujii, "Structure of parallel multipliers for a class of fields $gf(2^m)$," *Inf. Comput.*, vol. 83, pp. 21–40, Oct. 1989.

[41] Q. Shao, Z. Hu, S. N. Basha, Z. Zhang, Z. Wu, C. Lee, and J. Xie, "Low complexity implementation of unified systolic multipliers for nist pentanomials and trinomials over $GF(2^m)$," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 8, pp. 2455–2465, 2018.

[42] F. P. .-. Federal Information Processing Standards Publication, *Digital Signature Standard (DSS)*. 2013.

[43] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren, *Handbook Of Elliptic and Hyperelliptic Curve Cryptography*. Chapman & Hall/CRC, 2006.

[44] P. Garrett, *The Mathematics of Coding Theory İnformation, Compression, Error Correction, and Finite Field.* Pearson Education, Inc, 2004.

[45] C. K. Koc and B. Sunar, “Low-Complexity Bit-Parallel Canonical and Normal Basis Multipliers for a Class of Finite Fields,” *IEEE Transactions on Computers*, vol. 47, pp. 353–356, March 1998.

[46] H. Brunner, A. Curiger, and M. Hofstetter, “On computing multiplicative inverses in $GF(2^m)$,” *IEEE Transactions on Computers*, vol. 42, no. 8, pp. 1010–1015, 1993.

[47] K. Kobayashi, N. Takagi, and K. Takagi, “An algorithm for inversion in $gf(2^m)$ suitable for implementation using a polynomial multiply instruction on gf(2),” in *18th IEEE Symposium on Computer Arithmetic (ARITH ’07)*, pp. 105–112, 2007.

[48] “Understanding Process Corner (Corner Lots).” https://anysilicon.com/understanding-process-corner-corner-lots/.

[49] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, “Low-power cmos digital design,” *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, 1992.

[50] K. Ray, *The Singularity in Near.* Penguin books, 2005.

[51] P. K. Meher, “Systolic and Super-Systolic Multipliers for Finite Field $GF(2^m)$ Based on Irreducible Trinomials,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 55, pp. 1031–1040, May 2008.

[52] B. Sunar and C. K. Koc, “Mastrovito multiplier for all trinomials,” *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 522–527, 1999.

[53] Tong Zhang and K. K. Parhi, “Systematic design of original and modified mastrovito multipliers for general irreducible polynomials,” *IEEE Transactions on Computers*, vol. 50, no. 7, pp. 734–749, 2001.

[54] H. Fan and M. A. Hasan, “Fast bit parallel-shifted polynomial basis multipliers in $GF(2^n)$,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 12, pp. 2606–2615, 2006.

[55] S. K. Jain, L. Song, and K. K. Parhi, "Efficient semisystolic architectures for finite-field arithmetic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 1, pp. 101–113, 1998.

[56] Chiou-Yng Lee, Jenn-Shyong Horng, I-Chang Jou, and Erl-Huei Lu, "Low-complexity bit-parallel systolic montgomery multipliers for special classes of $GF(2^m)$," *IEEE Transactions on Computers*, vol. 54, no. 9, pp. 1061–1070, 2005.

[57] S. Devadas, A. Ghosh, and K. Keutzer, *Logic Synthesis*. McGraw-Hill, Inc, 1994.

[58] "Cadence Design Systems." `https://cadence.com/`. [Online; accessed 24-June-2020].

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

[60] P. K. Meher, "Systolic and Non-Systolic Scalable Modular Designs of Finite Field Multipliers for Reed–Solomon Codec," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, pp. 747–757, June 2009.

[61] Chiou-Yng Lee, Erl-Huei Lu, and Jau-Yien Lee, "Bit-parallel systolic multipliers for $GF(2^m)$ fields defined by all-one and equally spaced polynomials," *IEEE Transactions on Computers*, vol. 50, pp. 385–393, May 2001.

[62] A. G. Wassal, M. A. Hassan, and M. I. Elmasry, "Low-power design of finite field multipliers for wireless applications," in *Proceedings of the 8th Great Lakes Symposium on VLSI (Cat. No.98TB100222)*, pp. 19–25, Feb 1998.

[63] J. Xie, C. Lee, P. K. Meher, and Z. Mao, "Novel Bit-Parallel and Digit-Serial Systolic Finite Field Multipliers Over $GF(2^m)$ Based on Reordered Normal Basis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, pp. 2119–2130, Sep. 2019.

[64] J. Goodman and A. P. Chandrakasan, "An Energy-Efficient Reconfigurable Public-Key Cryptography Processor," *IEEE Journal of Solid-State Circuits*, vol. 36, pp. 1808–1820, Nov 2001.

[65] "Digital buffer tutorial." `www.electronics-tutorials.ws`. [Online; accessed 10-Jun-2020].

[66] P. Jamieson and J. Rose, "Mapping multiplexers onto hard multipliers in fpgas," in *The 3rd International IEEE-NEWCAS Conference, 2005.*, pp. 323–326, 2005.

[67] K. Golshan, *Physical Design Essentials: An ASIC Design Implementation Perspective*. Springer, 2007.

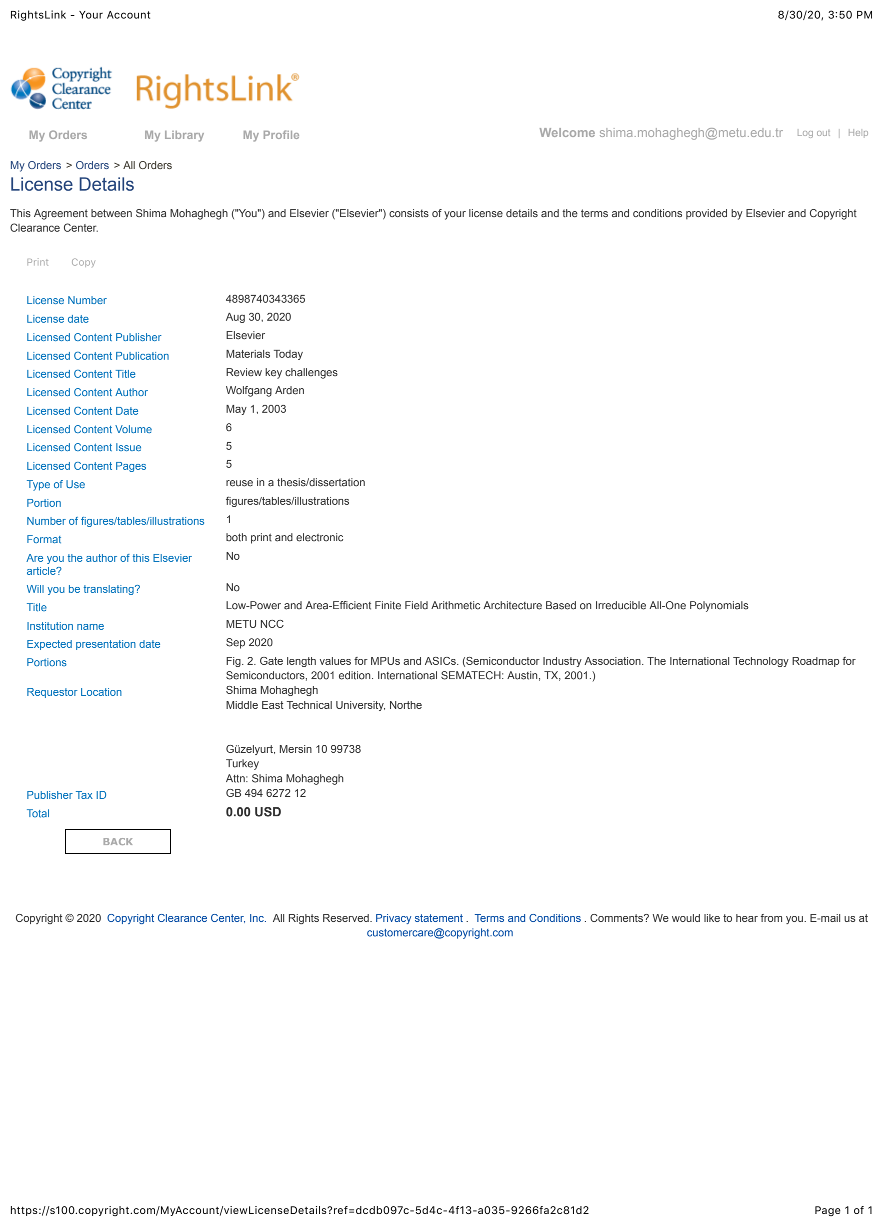
[68] K. Akarvardar, D. Elata, R. Parsa, G. C. Wan, K. Yoo, J. Provine, P. Peumans, R. T. Howe, and H. . P. Wong, "Design considerations for complementary nanoelectromechanical logic gates," in *2007 IEEE International Electron Devices Meeting*, pp. 299–302, 2007.

[69] F. Chen, H. Kam, D. Markovic, T. K. Liu, V. Stojanovic, and E. Alon, "Integrated circuit design with nem relays," in *2008 IEEE/ACM International Conference on Computer-Aided Design*, pp. 750–757, 2008.

[70] Hei Kam, Tsu-Jae King-Liu, E. Alon, and M. Horowitz, "Circuit-level requirements for mosfet-replacement devices," in *2008 IEEE International Electron Devices Meeting*, pp. 1–1, 2008.

# Appendix A

## COPYRIGHT CLEARANCE OF FIGURE 3.3



RightsLink - Your Account 8/30/20, 3:50 PM

Copyright Clearance Center
RightsLink®
My Orders
My Library
My Profile
Welcome shima.mohaghegh@metu.edu.tr
Log out | Help
My Orders > Orders > All Orders

License Details

This Agreement between Shima Mohaghegh ("You") and Elsevier ("Elsevier") consists of your license details and the terms and conditions provided by Elsevier and Copyright Clearance Center.

| | |
|---|---|
| License Number | 4898740343365 |
| License date | Aug 30, 2020 |
| Licensed Content Publisher | Elsevier |
| Licensed Content Publication | Materials Today |
| Licensed Content Title | Review key challenges |
| Licensed Content Author | Wolfgang Arden |
| Licensed Content Date | May 1, 2003 |
| Licensed Content Volume | 6 |
| Licensed Content Issue | 5 |
| Licensed Content Pages | 5 |
| Type of Use | reuse in a thesis/dissertation |
| Portion | figures/tables/illustrations |
| Number of figures/tables/illustrations | 1 |
| Format | both print and electronic |
| Are you the author of this Elsevier article? | No |
| Will you be translating? | No |
| Title | Low-Power and Area-Efficient Finite Field Arithmetic Architecture Based on Irreducible All-One Polynomials |
| Institution name | METU NCC |
| Expected presentation date | Sep 2020 |
| Portions | Fig. 2. Gate length values for MPUs and ASICs. (Semiconductor Industry Association. The International Technology Roadmap for Semiconductors, 2001 edition. International SEMATECH: Austin, TX, 2001.) |
| Requestor Location | Shima Mohaghegh<br>Middle East Technical University, Northe<br><br>Güzelyurt, Mersin 10 99738<br>Turkey<br>Attn: Shima Mohaghegh |
| Publisher Tax ID | GB 494 6272 12 |
| Total | **0.00 USD** |

Copyright © 2020 Copyright Clearance Center, Inc. All Rights Reserved. Privacy statement . Terms and Conditions . Comments? We would like to hear from you. E-mail us at customercare@copyright.com

https://s100.copyright.com/MyAccount/viewLicenseDetails?ref=dcdb097c-5d4c-4f13-a035-9266fa2c81d2 Page 1 of 1

# Appendix B

## SAMPLE LOG FILE OF GENUS SYNTHESIS SOLUTION

```
Cadence Genus(TM) Synthesis Solution.
Copyright 2016 Cadence Design Systems, Inc.
All rights reserved worldwide.
Cadence and the Cadence logo are
registered trademarks and Genus
is a trademark
of Cadence Design Systems, Inc.
in the United States and other countries.

Version: 16.12-s027_1, built Wed Oct 05 2016
Options: -legacy_ui
Date:    Sat Sep 12 20:09:04 2020
Host:    energy (x86_64 w/Linux 2.6.32-431.el6.x86_64)
(6cores*12cpus*Intel(R) Xeon(R)
CPU X5650 @ 2.67GHz 12288KB)
        (16312764KB)
OS:      CentOS release 6.5 (Final)

Checking out license: Genus_Synthesis

Loading tool scripts...
Finished loading tool scripts (7 seconds elapsed).

WARNING: This version of the tool is 1438 days old.
legacy_genus:/> rm $DESIGN
can’t read "DESIGN": no such variable
 (Sat Sep 12 20:09:33 +0300 2020)...
Sat Sep 12 20:09:33 +0300 2020
  Setting attribute of message ’LBR-30’: ’max_print’ = 0
  Setting attribute of message ’LBR-31’: ’max_print’ = 0
  Setting attribute of message 'LBR-40': 'max_print' = 0
  Setting attribute of message 'LBR-41': 'max_print' = 0
  Setting attribute of message 'LBR-72': 'max_print' = 0
  Setting attribute of message 'LBR-77': 'max_print' = 0
  Setting attribute of message 'LBR-162': 'max_print' = 0
Info   : Enabled hdl_track_file_row_column attribute.
       : Setting this attribute to 'true' can have an
impact on the run time. Use this attribute only
when filename, line number, and column information
are needed in reports.
  Setting attribute of root
'/': 'hdl_track_filename_row_col' = true
  Setting attribute of root
'/': 'lp_power_unit' = mW
  Setting attribute of root
'/': 'init_lib_search_path' = /home/shima/proj_library
/Meher_162_909090/genus_labs/work
/../libraries/TIMING/CUSTOM /
home/shima/proj_library/
Meher_162_909090/
genus_labs/work/../libraries/
TIMING/STDCELL /home/shima/proj_library/
Meher_162_909090/
.
.
.
Warning : An attribute is used before
it is defined. [LBR-511]
        : The library level attribute
default_operating_conditions on line 5153111 is
defined after
at least one cell definition. The attribute will
be ignored. (File /home/shima/proj_library
/Meher_162_909090
/genus_labs/work/../libraries/TIMING/STDCELL/
tcbn90lphpwc_ccs.lib)
Warning : An attribute is used before it
is defined. [LBR-511]
        : The library level attribute
default_wire_load_selection
```

```
on line 5153112 is defined after
at least one cell definition.
The attribute will be ignored. (File /home/shima/proj_library/Meher_162_909090/
genus_labs/work/../libraries/TIMING/
STDCELL/tcbn90lphpwc_ccs.lib)
Warning : An attribute is used before
it is defined. [LBR-511]
        : The library level attribute
default_wire_load on line 5153113 is
defined after at least one cell definition.
The attribute will be ignored.
(File /home/shima/proj_library/
Meher_162_909090/
genus_labs
/work/../libraries/TIMING/STDCELL/
tcbn90lphpwc_ccs.lib)

  Message Summary for Library tcbn90lphpwc_ccs.lib:
  *************************************************
  Could not find an attribute in the library. [LBR-436]: 1021
  Missing a function attribute in the output pin definition. [LBR-518]: 1
  An attribute is used before it is defined. [LBR-511]: 3
  An unsupported construct was detected in this library.
[LBR-40]: 1
  *************************************************

Info    : Created nominal operating condition. [LBR-412]
        : Operating condition '_nominal_' was
created for the PVT values (1.000000, 1.080000, 125.000000)
in library 'tcbn90lphpwc_ccs.lib'.
        : The nominal operating condition represents
either the nominal PVT values if specified in the
library source,
or the default PVT values (1.0, 1.0, 1.0).

Warning : Unusable clock gating integrated cell. [LBR-101]
        : Clock gating integrated cell name
: 'CKLHQD20'
        : To use the cell in clock gating
, Set cell attribute 'dont_use'
```

```
false in the library.
Warning : Unusable clock gating
integrated cell. [LBR-101]
        : Clock gating integrated
cell name: 'CKLHQD24'
Warning : Unusable clock gating integrated cell. [LBR-101]
        : Clock gating integrated cell name: 'CKLNQD20'
.
.
.

  Setting attribute of root '/': 'library' =
tcbn90lphpwc_ccs.lib

  According to lef_library, there are total 9 routing layers
[ V(4) / H(5) ]

  Library has 470 usable logic and 282 usable
sequential lib-cells.
  Setting attribute of root '/':
'lef_library' = /home/shima/proj_library/
Meher_162_909090/
genus_labs/work/../libraries/LEF/
STDCELL/tcbn90lphp_9lmT2.lef

  According to cap_table_file,
there are total 9
routing layers [ V(4) / H(5) ]

Warning : Wire parameter is missing. [PHYS-15]
        : 'WIDTH' parameter is missing
for layer 'M10' [line 283 in file
/home/shima/proj_library
/Meher_162_909090/genus_labs/work/../
libraries
/cln90_1p09m_top2_typical.ict.captable]

        : Check the parameter in
technology section.
Warning : Cap table has more layers than lef.
```

```
[PHYS-27]
        : Lef 'tcbn90lphp_9lmT2.lef' has '9'
routing
layers, cap table
'cln90_1p09m_top2_typical.ict.captable'
has '10' layers. Use lef layers
  Setting attribute of root '/': 'cap_table_file' = /home/shima/proj_library
/Meher_162_909090
/genus_labs/work/../
libraries/
cln90_1p09m_top2_typical.ict.captable
Freeing libraries in memory (  tcbn90lphpwc_ccs.lib  )

Info    : Created nominal operating condition. [LBR-412]
        : Operating condition '_nominal_'
          was created for the PVT values
(1.000000, 1.080000, 125.000000) in library
'tcbn90lphpwc_ccs.lib'.

Warning : Unusable clock gating integrated cell. [LBR-101]
        : Clock gating integrated cell name: 'CKLHQD20'
Warning : Unusable clock gating integrated cell. [LBR-101]
        : Clock gating integrated cell name: 'CKLHQD24'
Warning : Unusable clock gating integrated cell. [LBR-101]
        : Clock gating integrated cell name: 'CKLNQD20'
Warning : Unusable clock gating integrated cell. [LBR-101]
        : Clock gating integrated cell name: 'CKLNQD24'
Warning : Library cell has no output pins defined. [LBR-9]
        : Library cell 'DCAP' must have an output pin.
.
.
.
        : Library cell 'GDCAP10' must have an output pin.
  Library has 470 usable logic and 282
usable sequential lib-cells.
  Setting attribute of root '/': 'library' =
tcbn90lphpwc_ccs.lib

  According to lef_library, there are
total 9 routing layers [ V(4) / H(5) ]
```

```
Warning : Cap table has more layers than lef. [PHYS-27]
        : Lef 'tcbn90lphp_9lmT2.lef'
has '9' routing layers, cap table 'cln90_1p09m_top2_typical.ict.captable'
has '10' layers. Use lef layers
  Setting attribute of root '/': 'lef_library' = /home/shima/proj_library/Meher_162_909090/
genus_labs/work
/../libraries/LEF/STDCELL/tcbn90lphp_9lmT2.lef


  According to cap_table_file, there are total 9
routing layers [ V(4) / H(5) ]

Warning : Wire parameter is missing. [PHYS-15]
        : 'WIDTH' parameter is missing for layer
'M10' [line 283 in file /home/shima/proj_library/
Meher_162_909090/genus_labs/work/../
libraries/cln90_1p09m_top2_typical.ict.captable]
Warning : Cap table has more layers
than lef. [PHYS-27]
        : Lef 'tcbn90lphp_9lmT2.lef' has '9'
routing layers,
cap table 'cln90_1p09m_top2_typical.ict.captable'
has '10' layers. Use lef layers
  Setting attribute of root '/':
'cap_table_file' = /home/shima/proj_library
/Meher_162_909090/
genus_labs/work/../libraries/
cln90_1p09m_top2_typical.ict.captable
Info    : Elaborating Design. [ELAB-1]
        : Elaborating top-level block
'Structure1' from file '/home/shima/proj_library
/Meher_162_909090/genus_labs/
work/../rtl/Structure1.v'.
Warning : Undriven module input port. [ELABUTL-127]
        : Undriven bits of port 'D'
of instance 'Register19' of module
'Register' in file '/home/
shima/proj_library
/Meher_162_909090/genus_labs/work/../
```

```
rtl/Structure1.v' on line 186, column 21.
        : Use the 'hdl_undriven_signal_value'
attribute to control
treatment of undriven input port
during elaboration.
Info    : Done Elaborating Design. [ELAB-3]
        : Done elaborating 'Structure1'.

Info: Checking for source rtl...
Info: Check completed for source rtl...
/designs/Structure1

Statistics for commands executed by read_sdc:
 "create_clock"             - successful      1 , failed      0 (runtime  0.00)
 "current_design"           - successful      1 , failed      0 (runtime  0.00)
 "get_clocks"               - successful      1 , failed      0 (runtime  0.00)
 "get_ports"                - successful      1 , failed      0 (runtime  0.00)
 "set_clock_transition"     - successful      1 , failed      0 (runtime  0.00)
Total runtime 0
  Setting attribute of root
'/': 'leakage_power_effort' = high
Warning : Total power has skewed contributions
from leakage and dynamic power. [POPT-502]
        : With 'lp_power_optimization_weight'
set to 0.99, the total power of 'CKND2D0'
is being computed as:
  Total Power = (Leakage Power * 0.99) + (Dynamic Power * (1 - 0.99))
  Leakage Power:     0.0000004875 mW
  Dynamic Power:     0.0019000000 mW
  Total Power:       0.0000194826 mW
  Leakage Power is contributing 2.48% to
the Total Power.
        : Dynamic power is typically
calculated/specified for some 'active period'.
For combination of
leakage and dynamic power
lp_power_optimization_weight must specify
the percentage
of overall time for which the design is not
in the 'active
```

```
period' but in 'idle mode', i.e. no dynamic
power but only
leakage power is consumed.
For a reasonable
optimization across dynamic and
leakage power,
leakage contribution is expected to be above
5% and below 95%. A contribution of less than 5% will
result in limited leakage optimization and
contribution of more than 95% will result
in limited dynamic optimization. Adjust 'lp_power_optimization_weight' so tha
t leakage contribution
comes to an intermediate value if you
intend both optimizations to occur.
  Setting attribute of design 'Structure1': 'lp_power_optimization_weight' = 0.99

  Setting attribute of design 'Structure1':
'lp_default_toggle_rate' = 0.20000

  Setting attribute of root '/': '
syn_generic_effort' = medium
legacy_genus:/> syn_generic
Info    : Deleting instances
not driving any primary outputs. [GLO-34]
        : Deleting 162 hierarchical instances.
        : Optimizations such as constant
propagation or redundancy removal could
change the connections so a hierarchical
instance does not drive any primary
outputs anymore. To see the list of
deleted hierarchical instances,
set the 'information_level'
attribute to 2 or above.
If message is truncated set message attribute
'truncate' to false to see the complete list.
To prevent this optimization, set the
'delete_unloaded_insts' root/subdesign
attribute to 'false' or 'preserve'
instance attribute to 'true'.
.
```

```
.
.

Info    : Synthesizing. [SYNTH-1]
        : Synthesizing 'Structure1'
to generic gates using 'medium' effort.
Info    : Partition Based Synthesis
execution skipped. [PHYS-752]
        : Design size is less than
the threshold size '300000' for
distributed generic optimization to kick in.
.
.
.
Info    : Optimizing RTL. [RTLOPT-1]
        : Optimizing RTL in
'Structure1' using 'medium' effort.
Info    : Done optimizing RTL. [RTLOPT-2]
        : Done optimizing RTL in 'Structure1'.
  Setting attribute of root '/'
: 'pbs_stage_start_elapsed_time' = 313
  Setting attribute of root '/':
'pbs_stage_start_st_time' = 75
Info    : Done synthesizing. [SYNTH-2]
        : Done synthesizing
'Structure1' to generic gates.


Warning : Total power has skewed
contributions from leakage and dynamic power. [POPT-502]
        : With 'lp_power_optimization_weight'
set to 0.99, the total power of 'CKND2D0'
is being computed as:
  Total Power =
(Leakage Power * 0.99) + (Dynamic Power * (1 - 0.99))
  Leakage Power:     0.0000004875 mW
  Dynamic Power:     0.0019000000 mW
  Total Power:       0.0000194826 mW
  Leakage Power is contributing 2.48% to the Total Power.
Info    : Mapping. [SYNTH-4]
```

```
        : Mapping 'Structure1' using 'medium' effort.
Mapper: Libraries have:
Library based partitioning cannot be
done as either there is a single library
or there are no cells to partition.
domain _default_: 470 combo usable
cells and 282 sequential usable cells
leakage powergroup of combo usable
cells (no_value: 470) sequential usable cells
(no_value: 282)
Info    : Partition Based Synthesis
execution skipped. [PHYS-752]
        : Design size is less than
the partition threshold '300000' for
distributed mapping optimization to kick in.
.
.
.
Warning : Total power has skewed
contributions from leakage and dynamic power.
[POPT-502]

        : With 'lp_power_optimization_weight'
set to 0.99, the total power of 'CKND2D0'
is being computed as:
  Total Power =
(Leakage Power * 0.99) + (Dynamic Power * (1 - 0.99))
  Leakage Power:     0.0000004875 mW
  Dynamic Power:     0.0019000000 mW
  Total Power:       0.0000194826 mW
  Leakage Power is contributing 2.48%
to the Total Power.
Mapper: Libraries have:
domain _default_:
470 combo usable cells and 282 sequential usable cells
leakage powergroup of
combo usable cells (no_value: 470) sequential usable cells (no_value: 282)
Multi-threaded constant propagation [1|0] ...
  Setting attribute of root '/':
'super_thread_servers' = localhost localhost
```

```
localhost localhost
localhost localhost localhost
localhost
                  Library loading done successfully
on server 'localhost_1_0'.
Multi-threaded Virtual Mapping
(8 threads per ST process, 8 of 12 CPUs usable)
        Distributing super-thread jobs: tristate_buffer
          Sending 'tristate_buffer' to server 'localhost_1_0'...
            Sent 'tristate_buffer' to server 'localhost_1_0'.
          Received 'tristate_buffer' from server 'localhost_1_0'. (634 ms elapsed)
        Distributing super-thread jobs:
Register My_AND My_AND_3 My_AND_7
My_AND_15 My_AND_16 My_AND_8
My_AND_17 My_AND_1 My_AND_4
My_AND_11 My_AND_9 My_AND_10
My_AND_2 My_AND_5 My_AND_13
My_AND_12 My_AND_6 My_AND_14
          Sending 'Register' to server 'localhost_1_0'...
            Sent 'Register' to server 'localhost_1_0'.
          Sending 'My_AND' to server 'localhost_1_1'...
            Sent 'My_AND' to server 'localhost_1_1'.
          Sending 'My_AND_3' to server 'localhost_1_2'...
            Sent 'My_AND_3' to server 'localhost_1_2'.
          Sending 'My_AND_7' to server 'localhost_1_3'...
            Sent 'My_AND_7' to server 'localhost_1_3'.
          Sending 'My_AND_15' to server 'localhost_1_4'...
            Sent 'My_AND_15' to server 'localhost_1_4'.
          Sending 'My_AND_16' to server 'localhost_1_6'...
            Sent 'My_AND_16' to server 'localhost_1_6'.
          Sending 'My_AND_8' to server 'localhost_1_5'...
            Sent 'My_AND_8' to server 'localhost_1_5'.
          Sending 'My_AND_17' to server 'localhost_1_7'...
            Sent 'My_AND_17' to server 'localhost_1_7'.
          Received 'Register' from server 'localhost_1_0'. (292 ms elapsed)
          Sending 'My_AND_1' to server 'localhost_1_0'...
            Sent 'My_AND_1' to server 'localhost_1_0'.
          Received 'My_AND_1' from server 'localhost_1_0'. (189 ms elapsed)
          Sending 'My_AND_4' to server 'localhost_1_0'...
            Sent 'My_AND_4' to server 'localhost_1_0'.
          Received 'My_AND_4' from server 'localhost_1_0'. (264 ms elapsed)
          Sending 'My_AND_11' to server 'localhost_1_0'...
            Sent 'My_AND_11' to server 'localhost_1_0'.
          Received 'My_AND' from server 'localhost_1_1'. (815 ms elapsed)
          Sending 'My_AND_9' to server 'localhost_1_1'...
            Sent 'My_AND_9' to server 'localhost_1_1'.
          Received 'My_AND_7' from server 'localhost_1_3'. (831 ms elapsed)
          Sending 'My_AND_10' to server 'localhost_1_3'...
            Sent 'My_AND_10' to server 'localhost_1_3'.
          Received 'My_AND_3' from server 'localhost_1_2'. (858 ms elapsed)
          Sending 'My_AND_2' to server 'localhost_1_2'...
            Sent 'My_AND_2' to server 'localhost_1_2'.
          Received 'My_AND_11' from server 'localhost_1_0'. (225 ms elapsed)
          Sending 'My_AND_5' to server 'localhost_1_0'...
            Sent 'My_AND_5' to server 'localhost_1_0'.
          Received 'My_AND_10' from server 'localhost_1_3'. (568 ms elapsed)
          Sending 'My_AND_13' to server 'localhost_1_3'...
            Sent 'My_AND_13' to server 'localhost_1_3'.
          Received 'My_AND_9' from server 'localhost_1_1'. (596 ms elapsed)
          Sending 'My_AND_12' to server 'localhost_1_1'...
            Sent 'My_AND_12' to server 'localhost_1_1'.
          Received 'My_AND_17' from server 'localhost_1_7'. (1376 ms elapsed)
          Sending 'My_AND_6' to server 'localhost_1_7'...
            Sent 'My_AND_6' to server 'localhost_1_7'.
          Received 'My_AND_8' from server 'localhost_1_5'. (1403 ms elapsed)
          Sending 'My_AND_14' to server 'localhost_1_5'...
            Sent 'My_AND_14' to server 'localhost_1_5'.
.
.
.

  'localhost_1_2'. (434 ms elapsed)
          Received 'My_AND_52' from server 'localhost_1_7'. (450 ms elapsed)
          Received 'My_AND_37' from server 'localhost_1_1'. (295 ms elapsed)
          Received 'My_AND_53' from server 'localhost_1_3'. (312 ms elapsed)
        Distributing super-thread jobs:
Register_94 Register_92 Register_84
Register_66 Register_64 Register_62
Register_82 Register_60 Register_58
Register_90 Register_80 Register_78
```

```
Register_88 Register_76 Register_74
Register_72 Register_86 Register_70
Register_68 My_XOR_11 My_AND_59 My_XOR
My_AND_67 My_AND_60 My_XOR_14 My_AND_65
My_XOR_12 My_AND_61 My_AND_70
My_AND_68 My_AND_62 My_AND_71 My_AND_66
My_XOR_13 My_AND_63 My_XOR_7
My_XOR_16 My_XOR_4 My_XOR_15 My_XOR_1
My_AND_69 My_AND_58 My_AND_57 My_AND_56
My_AND_55 My_XOR_17 My_XOR_8 My_XOR_5
My_XOR_2 My_XOR_3 My_XOR_10 My_AND_54
My_XOR_6 My_AND_64 My_XOR_9
          Sending 'Register_94' to server 'localhost_1_0'...
            Sent 'Register_94' to server 'localhost_1_0'.
          Sending 'Register_92' to server 'localhost_1_1'...
            Sent 'Register_92' to server 'localhost_1_1'.
          Sending 'Register_84' to server 'localhost_1_2'...
            Sent 'Register_84' to server 'localhost_1_2'.
          Sending 'Register_66' to server 'localhost_1_3'...
            Sent 'Register_66' to server 'localhost_1_3'.
          Sending 'Register_64' to server 'localhost_1_4'...
            Sent 'Register_64' to server 'localhost_1_4'.
          Sending 'Register_62' to server 'localhost_1_5'...
            Sent 'Register_62' to server 'localhost_1_5'.
          Sending 'Register_82' to server 'localhost_1_6'...
            Sent 'Register_82' to server 'localhost_1_6'.
          Sending 'Register_60' to server 'localhost_1_7'...
            Sent 'Register_60' to server 'localhost_1_7'.
          Received 'Register_92' from server 'localhost_1_1'. (170 ms elapsed)
          Sending 'Register_58' to server 'localhost_1_1'...
            Sent 'Register_58' to server 'localhost_1_1'.
          Received 'Register_94' from server 'localhost_1_0'. (240 ms elapsed)
          Sending 'Register_90' to server 'localhost_1_0'...
            Sent 'Register_90' to server 'localhost_1_0'.
          Received 'Register_58' from server 'localhost_1_1'. (602 ms elapsed)
          Sending 'Register_80' to server 'localhost_1_1'...
            Sent 'Register_80' to server 'localhost_1_1'.
          Received 'Register_66' from server 'localhost_1_3'. (766 ms elapsed)
          Sending 'Register_78' to server 'localhost_1_3'...
            Sent 'Register_78' to server 'localhost_1_3'.
          Received ’Register_60’ from server ’localhost_1_7’. (720 ms elapsed)

.
.
.
’localhost_1_1’. (768 ms elapsed)
          Received ’My_XOR_6’ from server ’localhost_1_5’. (512 ms elapsed)
        Distributing super-thread jobs:
Register_131 Register_127 Register_119
Register_103 Register_71 Register_69
Register_101 Register_67 Register_65
Register_117 Register_99 Register_63
Register_61 Register_97 Register_59
Register_125 Register_115 Register_95
Register_93 Register_113 Register_91
Register_89 Register_129 Register_123
Register_111 Register_73 Register_75
Register_77 Register_79 Register_81
Register_83 Register_87 Register_85
Register_109 Register_121 Register_107
Register_105 My_AND_83 My_AND_81 My_AND_77
My_AND_74 My_AND_89 My_AND_82 My_AND_73
My_AND_72 My_AND_80 My_AND_88 My_AND_84
My_AND_78 My_AND_87 My_AND_75 My_AND_79
My_AND_85 My_AND_86 My_AND_76
          Sending ’Register_131’ to server ’localhost_1_0’...
            Sent ’Register_131’ to server ’localhost_1_0’.
          Sending ’Register_127’ to server ’localhost_1_1’...
            Sent ’Register_127’ to server ’localhost_1_1’.
          Sending ’Register_119’ to server ’localhost_1_2’...
            Sent ’Register_119’ to server ’localhost_1_2’.
.
.
.

          Received ’Register_117’ from server ’localhost_1_0’. (405 ms elapsed)
          Sending ’Register_95’ to server ’localhost_1_0’...
            Sent ’Register_95’ to server ’localhost_1_0’.
          Received ’Register_115’ from server ’localhost_1_1’. (473 ms elapsed)
          Sending ’Register_93’ to server ’localhost_1_1’...
```

```
  Sent ’Register_93’ to server ’localhost_1_1’.
Received ’Register_125’ from server ’localhost_1_5’. (543 ms elapsed)
Sending ’Register_113’ to server ’localhost_1_5’...
  Sent ’Register_113’ to server ’localhost_1_5’.
Received ’Register_59’ from server ’localhost_1_3’. (614 ms elapsed)
Sending ’Register_91’ to server ’localhost_1_3’...
  Sent ’Register_91’ to server ’localhost_1_3’.
Received ’Register_97’ from server ’localhost_1_2’. (745 ms elapsed)
Sending ’Register_89’ to server ’localhost_1_2’...
  Sent ’Register_89’ to server ’localhost_1_2’.
Received ’Register_61’ from server ’localhost_1_6’. (878 ms elapsed)
Sending ’Register_129’ to server ’localhost_1_6’...
  Sent ’Register_129’ to server ’localhost_1_6’.
```

.

.

.

```
Sending ’Register_168’ to server ’localhost_1_0’...
  Sent ’Register_168’ to server ’localhost_1_0’.
Sending ’Register_166’ to server ’localhost_1_1’...
  Sent ’Register_166’ to server ’localhost_1_1’.
Sending ’Register_162’ to server ’localhost_1_2’...
  Sent ’Register_162’ to server ’localhost_1_2’.
Sending ’Register_154’ to server ’localhost_1_3’...
  Sent ’Register_154’ to server ’localhost_1_3’.
Sending ’Register_138’ to server ’localhost_1_4’...
  Sent ’Register_138’ to server ’localhost_1_4’.
Sending ’Register_136’ to server ’localhost_1_6’...
  Sent ’Register_136’ to server ’localhost_1_6’.
Sending ’Register_152’ to server ’localhost_1_5’...
  Sent ’Register_152’ to server ’localhost_1_5’.
Sending ’Register_134’ to server ’localhost_1_7’...
  Sent ’Register_134’ to server ’localhost_1_7’.
Received ’Register_166’ from server ’localhost_1_1’. (171 ms elapsed)
Sending ’Register_132’ to server ’localhost_1_1’...
  Sent ’Register_132’ to server ’localhost_1_1’.
Received ’Register_168’ from server ’localhost_1_0’. (229 ms elapsed)
```

.

.

.

```
’
          Sending ’Register_189’ to server ’localhost_1_0’...
            Sent ’Register_189’ to server ’localhost_1_0’.
          Sending ’Register_191’ to server ’localhost_1_1’...
            Sent ’Register_191’ to server ’localhost_1_1’.
          Sending ’Register_193’ to server ’localhost_1_2’...
            Sent ’Register_193’ to server ’localhost_1_2’.
          Sending ’Register_195’ to server ’localhost_1_3’...
            Sent ’Register_195’ to server ’localhost_1_3’.
          Sending ’Register_197’ to server ’localhost_1_4’...
            Sent ’Register_197’ to server ’localhost_1_4’.
          Sending ’Register_199’ to server ’localhost_1_5’...
            Sent ’Register_199’ to server ’localhost_1_5’.
          Sending ’Register_187’ to server ’localhost_1_6’...
.
.
.
’localhost_1_3’. (244 ms elapsed)
          Received ’My_AND_120’ from server ’localhost_1_7’. (547 ms elapsed)
        Distributing super-thread jobs:
Register_242 Register_240 Register_238
Register_236 Register_234 Register_232
Register_230 Register_228 Register_226
Register_224 Register_222 Register_220
Register_218 Register_216 Register_214
Register_212 Register_210 Register_208
Register_206 My_XOR_41 My_AND_142 My_AND_139
My_XOR_42 My_XOR_43 My_AND_141 My_XOR_36 My_AND_138
My_AND_135 My_AND_136 My_AND_134 My_AND_143
My_XOR_51 My_XOR_50 My_AND_129 My_XOR_46
My_AND_140 My_AND_131 My_AND_130 My_AND_132
My_XOR_52 My_XOR_48 My_XOR_44 My_XOR_45
My_XOR_39 My_XOR_38 My_AND_126 My_AND_127
My_AND_128 My_XOR_53 My_XOR_49 My_AND_137
My_XOR_47 My_AND_133 My_XOR_40 My_XOR_37
          Sending ’Register_242’ to server ’localhost_1_0’...
            Sent ’Register_242’ to server ’localhost_1_0’.
          Sending ’Register_240’ to server ’localhost_1_1’...
            Sent ’Register_240’ to server ’localhost_1_1’.
          Sending ’Register_238’ to server ’localhost_1_2’...
```

```
            Sent 'Register_238' to server 'localhost_1_2'.
          Sending 'Register_236' to server 'localhost_1_3'...
            Sent 'Register_236' to server 'localhost_1_3'.
          Sending 'Register_234' to server 'localhost_1_4'...
            Sent 'Register_234' to server 'localhost_1_4'.
          Sending 'Register_232' to server 'localhost_1_6'...
            Sent 'Register_232' to server 'localhost_1_6'.
          Sending 'Register_230' to server 'localhost_1_5'...
            Sent 'Register_230' to server 'localhost_1_5'.
          Sending 'Register_228' to server 'localhost_1_7'...
            Sent 'Register_228' to server 'localhost_1_7'.
          Received 'Register_240' from server 'localhost_1_1'. (160 ms elapsed)
          Sending 'Register_226' to server 'localhost_1_1'...
            Sent 'Register_226' to server 'localhost_1_1'.
          Received 'Register_242' from server 'localhost_1_0'. (216 ms elapsed)
          Sending 'Register_224' to server 'localhost_1_0'...
            Sent 'Register_224' to server 'localhost_1_0'.
          Received 'Register_226' from server 'localhost_1_1'. (588 ms elapsed)
          Sending 'Register_222' to server 'localhost_1_1'...
            Sent 'Register_222' to server 'localhost_1_1'.
          Received 'Register_228' from server 'localhost_1_7'. (660 ms elapsed)
          Sending 'Register_220' to server 'localhost_1_7'...
            Sent 'Register_220' to server 'localhost_1_7'.
 .
.
.

Global mapping target info
==========================
Cost Group 'Clk' target slack:    14 ps
Target path end-point (Pin: Register277/Q_reg[162]/d)

Multi-threaded Technology Mapping (8 threads per ST process, 8 of 12 CPUs usable)
        Distributing super-thread jobs: tristate_buffer_1
          Sending 'tristate_buffer_1' to server 'localhost_1_0'...
            Sent 'tristate_buffer_1' to server 'localhost_1_0'.
          Received 'tristate_buffer_1' from server 'localhost_1_0'. (152 ms elapsed)
        Distributing super-thread jobs: Register_352
          Sending 'Register_352' to server 'localhost_1_0'...
            Sent 'Register_352' to server 'localhost_1_0'.
          Received 'Register_352' from server 'localhost_1_0'.
(245 ms elapsed)
        Distributing super-thread jobs: My_XOR_160
          Sending 'My_XOR_160' to server 'localhost_1_0'...
            Sent 'My_XOR_160' to server 'localhost_1_0'.
          Received 'My_XOR_160' from server 'localhost_1_0'.
(409 ms elapsed)
        Distributing super-thread jobs: Register_350
          Sending 'Register_350' to server 'localhost_1_0'...
            Sent 'Register_350' to server 'localhost_1_0'.
          Received 'Register_350' from server 'localhost_1_0'.
(284 ms elapsed)
        Distributing super-thread jobs: My_XOR_159
          Sending 'My_XOR_159' to server 'localhost_1_0'...
            Sent 'My_XOR_159' to server 'localhost_1_0'.
          Received 'My_XOR_159' from server 'localhost_1_0'.
(408 ms elapsed)
        Distributing super-thread jobs: Register_348 Register_347
          Sending 'Register_348' to server 'localhost_1_0'...
            Sent 'Register_348' to server 'localhost_1_0'.
          Sending 'Register_347' to server 'localhost_1_1'...
            Sent 'Register_347' to server 'localhost_1_1'.
          Received 'Register_348' from server 'localhost_1_0'.
(298 ms elapsed)
          Received 'Register_347' from server 'localhost_1_1'.
(404 ms elapsed)
        Distributing super-thread jobs: Register_351 My_XOR_157 My_XOR_158
          Sending 'Register_351' to server 'localhost_1_0'...
            Sent 'Register_351' to server 'localhost_1_0'.
          Sending 'My_XOR_157' to server 'localhost_1_1'...
            Sent 'My_XOR_157' to server 'localhost_1_1'.
          Sending 'My_XOR_158' to server 'localhost_1_2'...
            Sent 'My_XOR_158' to server 'localhost_1_2'.
          Received 'Register_351' from server 'localhost_1_0'. (282 ms elapsed)
          Received 'My_XOR_158' from server 'localhost_1_2'. (492 ms elapsed)
          Received 'My_XOR_157' from server 'localhost_1_1'. (634 ms elapsed)
        Distributing super-thread jobs:
Register_349 Register_342 Register_345
Register_343 Register_344
          Sending 'Register_349' to server 'localhost_1_0'...
```

```
                Sent 'Register_349' to server 'localhost_1_0'.
              Sending 'Register_342' to server 'localhost_1_1'...
                Sent 'Register_342' to server 'localhost_1_1'.
.
.
.
              Sending 'Register_72' to server 'localhost_1_0'...
                Sent 'Register_72' to server 'localhost_1_0'.
              Sending 'Register_78' to server 'localhost_1_1'...
                Sent 'Register_78' to server 'localhost_1_1'.
              Sending 'Register_84' to server 'localhost_1_2'...
                Sent 'Register_84' to server 'localhost_1_2'.
              Sending 'Register_82' to server 'localhost_1_3'...
                Sent 'Register_82' to server 'localhost_1_3'.
              Sending 'Register_92' to server 'localhost_1_4'...
                Sent 'Register_92' to server 'localhost_1_4'.
              Sending 'Register_74' to server 'localhost_1_5'...
                Sent 'Register_74' to server 'localhost_1_5'.
              Sending 'Register_66' to server 'localhost_1_6'...
                Sent 'Register_66' to server 'localhost_1_6'.
              Sending 'Register_58' to server 'localhost_1_7'...
                Sent 'Register_58' to server 'localhost_1_7'.
              Received 'Register_78' from server 'localhost_1_1'. (306 ms elapsed)
              Sending 'Register_68' to server 'localhost_1_1'...
                Sent 'Register_68' to server 'localhost_1_1'.
              Received 'Register_68' from server 'localhost_1_1'. (314 ms elapsed)
              Sending 'Register_70' to server 'localhost_1_1'...
.
.
.

              Received 'My_AND_4' from server 'localhost_1_2'. (1942 ms elapsed)
              Received 'My_AND_8' from server 'localhost_1_5'. (1852 ms elapsed)
            Distributing super-thread jobs: tristate_buffer
              Sending 'tristate_buffer' to server 'localhost_1_0'...
                Sent 'tristate_buffer' to server 'localhost_1_0'.
              Received 'tristate_buffer' from server 'localhost_1_0'. (153 ms elapsed)

Global mapping status
=====================
                                  Group
                                 Tot Wrst
                          Total  Weighted     Leakage
Operation                  Area   Slacks       Power
--------------------------------------------------------------------------------
 global_map             1453910      -26     1162683
            Worst cost_group: Clk, WNS: -26.2
            Path: Register0/Q_reg[0]/CP --> Register21/Q_reg[0]/D

    Cost Group            Target    Slack    Clock
--------------------------------------------------
           Clk                14      -26      500



Global incremental target info
==============================
Cost Group 'Clk' target slack:   -26 ps
Target path end-point (Pin: Register277/Q_reg[162]/D (DFQD4/D))



Global incremental optimization status
======================================
                                  Group
                                 Tot Wrst
                          Total  Weighted     Leakage
Operation                  Area   Slacks       Power
--------------------------------------------------------------------------------
 global_incr            1436038      -70     1020828
            Worst cost_group: Clk, WNS: -70.5
            Path: Register0/Q_reg[0]/CP --> Register21/Q_reg[0]/D

    Cost Group            Target    Slack    Clock
--------------------------------------------------
           Clk               -26      -70      500

  Setting attribute of root '/':
'pbs_stage_start_elapsed_time' = 2220
  Setting attribute of root '/':
'pbs_stage_start_st_time' = 1029
Info    : Done mapping. [SYNTH-5]
```

```
        : Done mapping 'Structure1'.

Info    : Incrementally optimizing. [SYNTH-7]
        : Incrementally optimizing 'Structure1' using 'medium' effort.
Info    : The given (sub)design is already uniquified. [TUI-296]
        : /designs/Structure1.
        : Try running the 'edit_netlist uniquify'
command on the parent hierarchy of this (sub)design,
if there exists any.
 Uniquify netlist ...
 Done
 Swap or remap avoided cells ...
 Done
 hi_fo_buf              1437669      -70  -1449705         0        0
            Worst cost_group: Clk, WNS: -70.5
            Path: Register0/Q_reg[0]/CP --> Register21/Q_reg[0]/D

           Trick     Calls     Accepts   Attempts    Time(secs)
-----------------------------------------------------------
       hi_fo_buf       163  (       21 /       21 )  8.76




                                  Group
                                 Tot Wrst     Total - - DRC Totals - -
                           Total  Weighted      Neg      Max       Max     Leakage
Operation                   Area   Slacks      Slack    Trans      Cap      Power
-------------------------------------------------------------------------------
 init_delay             1437669      -70  -1449705         0        0    1008453
            Worst cost_group: Clk, WNS: -70.5
            Path: Register0/Q_reg[0]/CP --> Register21/Q_reg[0]/D
 incr_delay             1456746       -8   -172242         0        0    1038252
            Worst cost_group: Clk, WNS: -8.3
            Path: Register0/Q_reg[0]/CP --> Register21/Q_reg[0]/D
 incr_delay             1462691       -4     -1548         0        0    1048553
            Worst cost_group: Clk, WNS: -4.8
            Path: Register242/Q_reg[1]/CP --> Register280/Q_reg[1]/D

           Trick     Calls     Accepts   Attempts    Time(secs)
```

```
-----------------------------------------------------------
       crit_upsz      1239  (     1141 /     1141 )  78.96
            fopt      1728  (     1304 /     1471 )  105.34

 init_tns                1462691       -4     -1548         0        0    1048553
 Worst cost_group: Clk, WNS: -4.8
Path: Register242/Q_reg[1]/CP --> Register280/Q_reg[1]/D
 incr_tns                1463151        0         0         0        0    1048732

           Trick     Calls     Accepts   Attempts    Time(secs)
-----------------------------------------------------------
        setup_dn       489  (      326 /      326 )  145.17

The inst 'pbs_iopt_n_1' is selected as pbs candidate.
The instance cnt is: '54643'.
The inst 'pbs_iopt_n_0' is selected as pbs candidate.
The instance cnt is: '57528'.
Info    : Connection established with super-threading
server. [ST-110]
        : The server 'localhost_1_8' is forked process '19885' on this host.
        : The tool is entering super-threading mode
and has established a connection
with a CPU server process.
This is enabled by the root attributes
'super_thread_servers'
or 'auto_super_thread'.
Info    : Connection established with super-threading server. [ST-110]
        : The server 'localhost_1_9' is forked process '19886' on this host.
Info    : Connection established with super-threading server. [ST-110]
        : The server 'localhost_1_10' is forked process '19887' on this host.
Info    : Connection established with super-threading server. [ST-110]
        : The server 'localhost_1_11' is forked process '19888' on this host.
Info    : Connection established with super-threading server. [ST-110]
        : The server 'localhost_1_12' is forked process '19889' on this host.
Info    : Connection established with super-threading server. [ST-110]
        : The server 'localhost_1_13' is forked process '19890' on this host.
Info    : Connection established with super-threading server. [ST-110]
        : The server 'localhost_1_14' is forked process '19891' on this host.
        Distributing super-thread jobs:
{pbs_iopt_n_0 ./.pbs_energy_14060/
pbs_iopt_n_0.etf} {pbs_iopt_n_1 ./.pbs_energy_14060/pbs_iopt_n_1.etf}
          Sending 'pbs_iopt_n_0 ./.pbs_energy_14060/
pbs_iopt_n_0.etf' to server
'localhost_1_0'...
            Sent 'pbs_iopt_n_0 ./
.pbs_energy_14060/pbs_iopt_n_0.etf'
to server 'localhost_1_0'.
          Sending 'pbs_iopt_n_1
./.pbs_energy_14060/
pbs_iopt_n_1.etf' to server 'localhost_1_8'...
            Sent 'pbs_iopt_n_1
./.pbs_energy_14060/pbs_iopt_n_1.etf'
to server 'localhost_1_8'.
          Received 'pbs_iopt_n_0 ./.pbs_energy_14060/pbs_iopt_n_0.etf'
from server 'localhost_1_0'.
(209481 ms elapsed)
          Received 'pbs_iopt_n_1
./.pbs_energy_14060/
pbs_iopt_n_1.etf' from server
'localhost_1_8'. (260449 ms elapsed)
       Start - Partition:
pbs_iopt_n_1  Slack: 0.4
TNS: 0  Cell-Count: 54643
Cell-Area: 572809  Init-Memory: 787.87
       Done  - Partition: pbs_iopt_n_1
Slack: 0.8  TNS: 0  Cell-Count:
54643  Cell-Area: 517503  Peak-Memory
: 830.90  Elapsed: 251
       Start - Partition: pbs_iopt_n_0
Slack: 0.4  TNS: 0  Cell-Count: 57528  Cell-Area: 627493  Init-Memory: 807.87
       Done  - Partition: pbs_iopt_n_0
Slack: 1.4  TNS: 0  Cell-Count: 57365
Cell-Area: 568290  Peak-Memory: 853.65
Elapsed: 199
 hi_fo_buf              1348332        0         0         0        0     843660

           Trick     Calls     Accepts   Attempts    Time(secs)
-----------------------------------------------------------
```

```
                                  Group
                                 Tot Wrst     Total - - DRC Totals - -
                          Total  Weighted      Neg       Max       Max     Leakage
Operation                  Area   Slacks      Slack     Trans      Cap      Power
-------------------------------------------------------------------------------
 init_delay             1348332        0          0         0        0     843660

           Trick     Calls     Accepts   Attempts    Time(secs)
-----------------------------------------------------------


 init_tns               1348332        0          0         0        0     843660

           Trick     Calls     Accepts   Attempts    Time(secs)
-----------------------------------------------------------

                                  Group
                                 Tot Wrst     Total - - DRC Totals - -
                          Total  Weighted      Neg       Max       Max     Leakage
Operation                  Area   Slacks      Slack     Trans      Cap      Power
-------------------------------------------------------------------------------
 init_power             1348332        0          0         0        0     843660
 p_rem_inv              1348274        0          0         0        0     843564
 p_merge_bi             1347939        0          0         0        0     843043
 glob_power             1346214        0          0         0        0     839977
 power_down             1342579        0          0         0        0     837499

           Trick     Calls     Accepts   Attempts    Time(secs)
-----------------------------------------------------------
       p_rem_buf      1141  (         0 /      163 )  1.45
       p_rem_inv         2  (         2 /        2 )  0.06
      p_merge_bi         1  (         1 /        1 )  0.55
      glob_power         4  (         2 /        4 )  19.67
      power_down     14451  (       654 /      654 )  284.85



                                  Group
                                 Tot Wrst     Total - - DRC Totals - -
                          Total  Weighted      Neg      Max       Max     Leakage
Operation                  Area   Slacks      Slack    Trans      Cap       Power
--------------------------------------------------------------------------------
 init_delay             1342579        0          0        0         0     837499

           Trick     Calls     Accepts   Attempts    Time(secs)
-----------------------------------------------------------


 init_tns               1342579        0          0        0         0     837499

           Trick     Calls     Accepts   Attempts    Time(secs)
-----------------------------------------------------------


 init_delay             1342579        0          0        0         0     837499

           Trick     Calls     Accepts   Attempts    Time(secs)
-----------------------------------------------------------
                                  Group
                                 Tot Wrst     Total - - DRC Totals - -
                          Total  Weighted      Neg      Max       Max     Leakage
Operation                  Area   Slacks      Slack    Trans      Cap       Power
--------------------------------------------------------------------------------
 init_power             1342579        0          0        0         0     837499

           Trick     Calls     Accepts   Attempts    Time(secs)
-----------------------------------------------------------


 init_area              1342579        0          0        0         0     837499

           Trick     Calls     Accepts   Attempts    Time(secs)
-----------------------------------------------------------



Local DRC optimization status
=============================
                                  Group
                                 Tot Wrst     Total - - DRC Totals - -
                          Total  Weighted      Neg      Max       Max     Leakage
Operation                  Area   Slacks      Slack    Trans      Cap       Power
```

```
-------------------------------------------------------------------------------
 init_drc                1342579        0          0          0         0     837499

           Trick     Calls     Accepts   Attempts    Time(secs)
-----------------------------------------------------------


           Trick     Calls     Accepts   Attempts    Time(secs)
-----------------------------------------------------------


           Trick     Calls     Accepts   Attempts    Time(secs)
-----------------------------------------------------------


Local DRC optimization status
=============================
                                   Group
                                  Tot Wrst     Total - - DRC Totals - -
                           Total  Weighted      Neg      Max       Max     Leakage
Operation                   Area   Slacks      Slack    Trans      Cap      Power
-------------------------------------------------------------------------------
 init_drc                1342579        0          0          0         0     837499

           Trick     Calls     Accepts   Attempts    Time(secs)
-----------------------------------------------------------


           Trick     Calls     Accepts   Attempts    Time(secs)
-----------------------------------------------------------


           Trick     Calls     Accepts   Attempts    Time(secs)
-----------------------------------------------------------


# Incremental Optimization Runtime Summary:
.
.
.
```

```
Done incrementally optimizing.
##>===== Cadence Confidential (Iopt-Logical) =====
##>ST Summary (2 partitions in total):
##>----------------------------------
##>PARTITION              1         0
##>----------------------------------
##>PRE_WNS                0         0
##>PRE_TNS                0         0
##>PRE_CNT            54643     57528
##>PRE_PORT_CNT       15414     15397
##>PRE_AREA          572809    627493
##>----------------------------------
##>POST_WNS               0         1
##>POST_TNS               0         0
##>POST_CNT           54643     57365
##>POST_PORT_CNT      15414     15397
##>POST_AREA         517503    568290
##>----------------------------------
##>I:Misc               259       208
##>Total Elapsed        259       208
##>==================================
##>Main Thread Summary:
##>------------------------------------------------------
##>STEP                Elapsed     Insts      Area    Memory
##>------------------------------------------------------
##>I:Initial                 6    110804   1175789      1550
##>I:Launch ST              34         -         -      2002
##>I:Distributed           297         -         -      2002
##>I:Cleanup              1005    111985   1080084      1162
##>I:Misc                    0
##>------------------------------------------------------
##>Total Elapsed          1010
.
.
.
Info    : Done incrementally optimizing. [SYNTH-8]
        : Done incrementally optimizing ’Structure1’.
```

```
============================================================
  Generated by:           Genus(TM) Synthesis Solution 16.12-s027_1
  Generated on:           Sep 12 2020  09:11:35 pm
  Module:                 Structure1
  Technology libraries:   tcbn90lphpwc_ccs 150.000000
                          physical_cells
  Operating conditions:   _nominal_
  Interconnect mode:      global
  Area mode:              physical library
============================================================

Timing
--------

Clock Period
-------------
Clk    500.0


  Cost    Critical       Violating
 Group   Path Slack TNS    Paths
-----------------------------------
Clk             1.1   0          0
default    No paths   0
-----------------------------------
Total                 0          0

Instance Count
--------------
Leaf Instance Count             111985
Sequential Instance Count        57376
Combinational Instance Count     54609
Hierarchical Instance Count        692
Warning : Potential error generating
clock gating report. [RPT_CG-12]
        : 'lp_insert_clock_gating'
root attribute is set to 'false'.
        : The 'report clock_gating'
command depends on the 'lp_insert_clock_gating'
attribute.  Set it to 'true' before calling this command.
```

```
Area
----
Cell Area                         1080084.499
Physical Cell Area                0.000
Total Cell Area (Cell+Physical)   1080084.499
Net Area                          262494.360
Total Area (Cell+Physical+Net)    1342578.859

Power
-----
Leakage Power                     0.849 mW
Dynamic Power                     3173.997 mW
Total Power                       3174.845 mW
Max Fanout                        57376 (Clk)
Min Fanout                        0 (PE1_1/P17[0])
Average Fanout                    1.9
Terms to net ratio                2.9140
Terms to instance ratio           3.0116
Runtime                           2105.561769 seconds
Elapsed Runtime                   3799 seconds
Genus peak memory usage           2001.43
Innovus peak memory usage         no_value
Hostname                          energy

                                              Leakage  Leakage  Internal  Internal
     Type      Instances     Area    Area % Power (mW) Power % Power (mW)  Power %
-----------------------------------------------------------------------------------
sequential         57376  736541.971   68.2      0.488    57.5   2315.295      89.0
inverter             330    1954.512    0.2      0.003     0.3      0.454       0.0
buffer              1304   23922.662    2.2      0.043     5.1     32.310       1.2
tristate             326    2300.256    0.2      0.001     0.1      0.545       0.0
logic              52649  315365.098   29.2      0.313    36.9    251.552       9.7
physical_cells         0       0.000    0.0      0.000     0.0      0.000       0.0
-----------------------------------------------------------------------------------
total             111985 1080084.499  100.0      0.849   100.0   2600.155     100.0
=============================================
PBS_iopt Index: 0.90  PBS_overall Index: 0.90
=============================================
Normal exit.
```