Politecnico di Torino

Master's degree in Electronic Engineering

Electronics and Telecommunications Department (DET)

Master Degree Thesis

Binary Edward Elliptic Curve CryptoCore Accelerator

Supervisor:

Prof. Guido Masera
Prof. Maurizio Martina

Candidate:

Gioele Sineo

novembre 2023

Summary

The Internet of Things (IoT) is a vast network connecting billions of devices through the Internet to share data. Many of these devices have limited resources, so they store their data in the cloud, allowing people to access it from anywhere. However, this cloud storage raises security concerns because data owners have little control over how their data is managed.
To address these security concerns and accommodate the limited resources of IoT devices, we use lightweight cryptographic methods. One such method is Elliptic Curve Cryptography (ECC) [1], a type of public-key cryptography. ECC is becoming popular in IoT security, smart card security, and digital signatures because it offers strong security with shorter keys compared to traditional methods like RSA. ECC can be implemented with minimal hardware and low energy use, making it ideal for securing low-power, low-memory IoT devices.

A significant advantage of elliptic curve cryptography is its resistance to current threats from quantum computers. While quantum computers promise to quickly solve problems that are beyond the capabilities of traditional computers, such as factoring large prime numbers, elliptic curves generate computational challenges that require much more time and resources to be handled even by quantum computers. This resilience makes it an attractive choice for the future of cryptography.

ECC can be integrated into small chips to provide fast data encryption and decryption. It also offers secure key agreement protocols that prevent unauthorized access to wireless sensor networks (WSNs) connected to IoT infrastructures. In RFID [2] technology, ECC-based authentication protocols enhance security in smart healthcare environments. Additionally, ECC-based digital signature schemes like ECDSA [3] and EdDSA are used in wireless body area networks (WBANs) to secure real-time health data, such as blood pressure and heart rate. Modern security protocols like TLS and DTLS use these signature schemes for efficient authentication in IoT platforms.

For this purpose i propose an Crypto accelerator based on BEC, implemented reducing as much as possible the area and the critical path for the point multiplication over Galois Prime Field GF(p) implemented following the Montgomery Ladder Algorithm (left-to-right), are also proposed all architectures resulting from meticulous evolution throughout the project of the intermediate modular operation write in VHDL. Finally utilising Xilinx Vivado is done the synthesis for virtex-7 [4] FPGA to obtain an initial approximation of the performance, considering that this result are affected by a big delay of the interconnection and area limitation due to the clb organisation of the board, instead better result are obtained in the last step of ASIC synthesis on silicon chip.

In conclusion, elliptic curve cryptography represents one of the most promising prospects in the direction of cybersecurity in the post-quantum world. Its ability to provide robust data protection with shorter keys and its resistance to quantum computers make it a key choice for ensuring the security of digital communications.

Keywords: elliptic curve cryptography (ECC), elliptic curve point multiplication (ECPM), twisted Edwards curve, unified point addition, unified point doubling, FPGA-ASIC synthesis

Acknowledgements

Questo spazio lo vorrei dedicare a tutte quelle persone che hanno contribuito a motivarmi e supportarmi durante tutto il percorso universitario, nonché nel percorso di crescita personale e professionale.

In primis ringrazio il mio relatore Guido Masera il quale mi ha dato la possibilità d'intraprendere questo percorso di tesi aprendo nuove porte alla mia conoscenza e dispensandomi utili consigli, rendendola un esperienza ancor più gratificante e un eccellente modo per consolidare e applicare ciò che ho appreso in questi cinque anni.

Un ringraziamento speciale va poi ai miei genitori e a mia sorella che mi hanno supportato ogni giorno fin dal primo momento, considerando la smisurata dedizione e impegno necessari per il compimento di questo percorso e il mio carattere molto rigoroso a volte troppo perfezionista posso affermare che sono stati un ottimo supporto. Estendo lo stesso ringraziamento a tutto ciò in cui mi diletto e a cui ambisco al di fuori del regime scolastico.

Ringrazio poi i mie due compagni di corso Lorenzo (Dudo) e Giovanni con i quali ho condiviso molteplici esperienze più o meno positive nell'affrontare esami e situazioni delicate che il magnifico Politecnico ci ha messo di fronte, a volte togliendoci ogni entusiasmo ed energia, siete stati di grande aiuto e supporto e apprezzo infinitamente il vostro sostengo, sia come amici che come colleghi.

Un ringraziamento va poi ai miei nonni i quali non c'e' stato giorno in cui non si siano interessati a come procedessero gli studi e gli esami, e come dice mia nonna se quando ho ricevuto la laurea triennale sono diventato Ingegnere ora con la magistrale devo essere qualcosa in più, quindi "Super Ingegnere".

Ringrazio poi la mia fidanzata Federica che da quando ci conosciamo mi ha sempre sostenuto e appoggiato in ogni piccola cosa incoraggiandomi e facendomi sentire apprezzato nonostante il mio "essere diverso dagli altri", questo mi ha dato più serenita' ed energia per proseguire con le mie ambizioni.

Grazie poi a tutti i miei amici e a tutti coloro che mi conoscono che si sono sempre interessati a sviluppi e progressi, grazie per le parole spese per complimentarvi o supportarmi, sono sicuramente state di aiuto e conforto.

Concludo dandomi una pacca sulla spalla da solo e dicendomi "Bravo", perché nonostante le mille difficoltà che il Politecnico mi ha messo di fronte ho sempre continuato ad andare avanti e non ho mai mollato e ora posso dire di essere un "Super ingegnere".

Contents

Chapter 1

Introduction

1.1 Research Methodology

The objective of this work is starting from some implementation available in the literature and select the more efficient then try to modify and combine them to obtain a better result in term of reducing the number of allocated elementary block and maximize the frequency, all the idea that i have follow are reported in careful way also to the point of view of the architecture also to the point of view the simulation.

Synthesis analyses is done using Vivado and the Virtex-7 FPGA, the results are compared with some result given by the paper, i try to modify the architecture and work with DSP integrated to reduce as much as possible the area and increase the frequency up to the best possible,the setting and the complexity of the software probably allow to obtain a better result in more advanced strategy and with more powerful work-station, but for our purpose are just good.

In the end i syntheses the cryptocore on silicon generating an ASIC using 45 nm technology, to have a complete view of the work, and also because for me is the most interesting way to culminate the project.

1.2 Thesis Organization

The structure of the thesis it is as follow: in the chapter 2 is reported a brief review on the literature about the word of cryptography [5] and some mechanism of secure data exchange, then more in detail Elliptic Curve Cryptography (ECC) and some reference to the mathematical background 2.3 about the arithmetic on the Twisted Edward Curve also for the case of affine coordinate 2.3.1 and also for the specific case under consideration of projective coordinates 2.3.2. In section 2.4 are reported all the parameter for the Twisted Edward Curve Ed25519 under analysis. Furthermore the chapter 3 is completely dedicated to explain all the developed architecture for the hierarchic block, their timing and software validation results also for the case of RADIX2 and RADIX4 multiplayer. In conclusion the chapter 4 and 5 report the synthesis results both for the FPGA case and for the ASIC case.

Chapter 2

Literature Review

2.1 Cryptography introduction

Cryptography is essential for information and security, can be divided into two different categories: private key cryptography 2.1.1 and public key cryptography 2.1.2. These two approaches offer different solutions for guarantee the security of digital communications. Here is a brief overview of the differences between them:

2.1.1 Symmetric Encryption

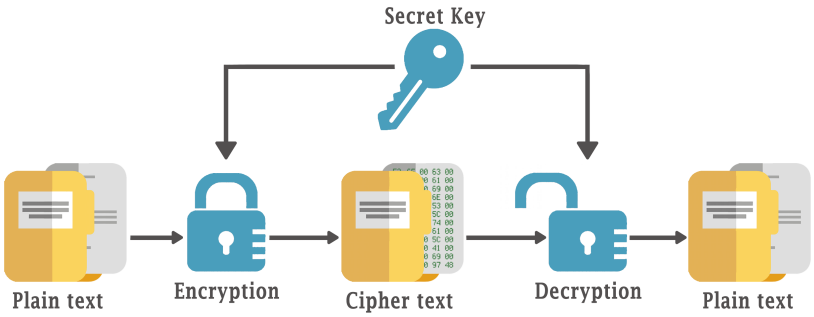


Figure 2.1. Symmetric Encryption

Private key cryptography is also known as symmetric cryptography, involves the use of a single secret key to encrypt and decrypt data. The same key is used for both encryption and decryption, and it is necessary for the sender and receiver to share this key in advance. The key feature of private key cryptography is its efficiency and speed, as it requires fewer computations compared to public key cryptography.

However, the primary problem of private key cryptography derive to the secure distribution of secret keys between the sender and receiver. If the secret key fall into the wrong hands the data security would be compromised. Additionally, this type of cryptography does not provide an authentication mechanism, which means it does not guarantee the identity of the sender.

2.1.2 Asymmetric Encryption

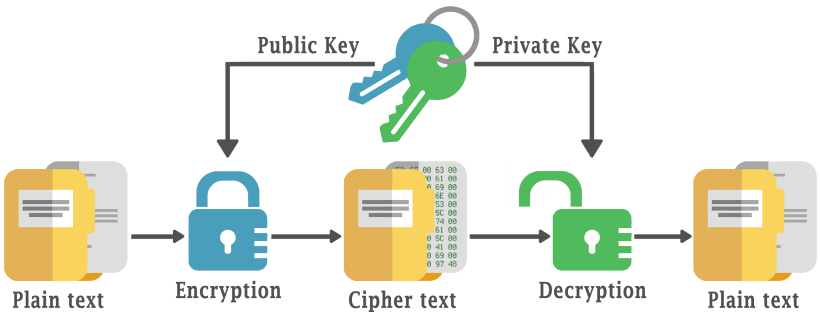


Figure 2.2. Asymmetric Encryption

Public key cryptography, also known as asymmetric cryptography, employs two distinct keys: a public key and a private key. The public key is used to encrypt data, while the private key is required to decrypt it. The public key can be freely distributed to anyone, whereas the private key must be kept secret by its owner.

This approach of key distribution, allow anyone to encrypt data using the public key, but only the holder of the private key can decrypt it. Furthermore, public key cryptography supports authentication, as the owner of the private key can digitally sign data and confirming his identity. However, public key cryptography tends to be slower and is often primarily used to establish secure initial communication, such as the key exchange process, then private key cryptography is frequently used to actually encrypt data during communication.

In summary, private key cryptography is fast and efficient but requires secure channel for the key distribution, whereas public key cryptography resolves the key distribution challenge but can be slower. Often, these two approaches could be combined to have the best of both. Asymmetric cryptography is the one over our interest for this research, we can distinguish basically two different implementation, the RSA (Rivest–Shamir–Adleman) algorithm and the ECC (Elliptic-Curve-Cryptography), the firt one is now outdated due to the fact that ECC guarantee more high security and less key length, but is important to understand his functioning to better apprehend other one.

Rivest–Shamir–Adleman (RSA)

This algorithm is one of the first that use modular operation and it is based on number factorization to generate the private key, the key length must be larger than 1024 bit to guarantee an acceptable level of security, to contextualize with an example traditional computer require around one year to violate the 1024 bit, it is clear that with the arrival of quantum computer this mechanism will not safe also using more high parallelism. RSA algorithm is implemented as follow:

- One chooses two random prime number sufficiently large, typical greater than 1024 bit defined as p and q.
- Now it is time to calculate the product $n = p \cdot q$ where n is the modulo value.
- Compute the Euler's totient function defined as:

$$\varphi(n) = (p-1)(p-1) \tag{2.1}$$

  define the number of integers between 1 and n that are coprime to n, the factorization of n is secret and also who know p and q can recognize.

- Then is chosen another number e that is coprime with $\varphi(n)$ and less the $\varphi(n)$.
- Is computer d that is the private exponent such that its product with e is congruent to 1 modulo $\varphi(n)$ means $ed \equiv 1\ mod(\varphi(n))$

Public-key is defined as (n, e) and the private-key as (m, d). The Power of this algorithm is due to the fact that compute e and d or viceversa is not possible only knowing n but is necessary know $\varphi(n)$ but the only way to obtain this value is factorizing the number and this is a very long time and expencive procedure.

Elliptic-Curve-criptography (ECC)

Elliptic Curve Cryptography is the modern evolution of RSA, give more high level of security and require less key dimension. Private keys in ECC are integers in the range of the curve's field size, typically 256-bit integers are sufficient, as in our case of Twisted Edward Curve.

Generating keys in ECC cryptography is easy, involving the secure generation of a random integer within a certain range, making it extremely fast. Any number within this range is a valid ECC private key, then the public keys is represented as EC points, which is pairs of integer coordinates (x, y) located on the curve. Due to their unique properties, EC points can be compressed into a single coordinate with an additional bit indicating whether it's odd or even, this is the public-key.

Many type of elliptic curve are available and each one give different result and allow to have different level of security and parallelism, to the point of view of the implementation cost each curve is governed by different equation and require different hardware to handle the operation, in section 2.3 are reported the mathematical specification for the curve over analysis in this research.

Elliptic Curve could be expressed with a generic equation in two dimension :

$$Ax^2 + Bx^2y + Cxy^2 + Dy^3 + Ex^2 + Fxy + Gy^2 + Hx + Iy + J = 0 \tag{2.2}$$

but in cryptography are used simplified form, the more relevant are:

Weierstrass Curve $$y^2 = x^3 + ax + b \tag{2.3}$$

Montgomery Curve $$by^2 = x^3 + ax^2 + x \tag{2.4}$$

Edward Curve $$ax^2 + y^2 = 1 + dx^2y^2 \tag{2.5}$$

In the following figure 2.3 is reported an example of Weierstrass curve that look good to explain the operation on the EC.

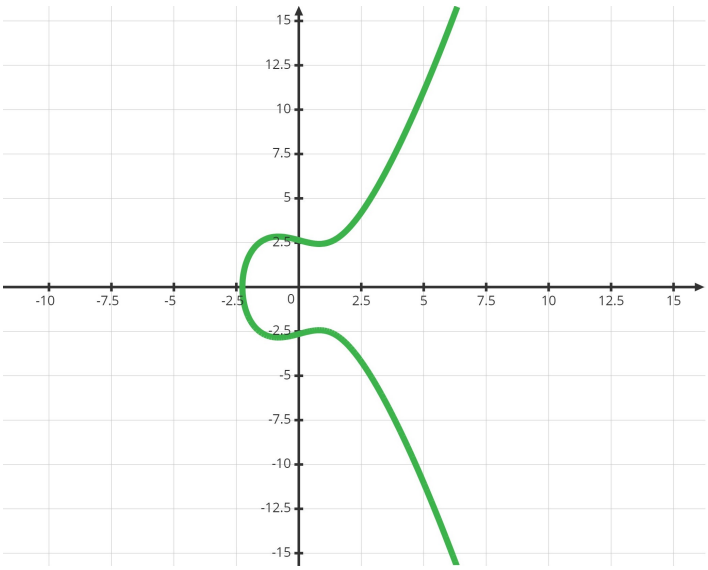


Figure 2.3. Example EC curve

This representation is on infinite field, but in ECC to have a strong level of security we work on finite field $\mathbb{F}p$ where p is the dimension of the prime field, differently from RSA both x and y coordinates work on limited range $[0, p-1]$, the result is a set of integer coordinate point $\{x, y\}$. Figure 2.4 is the visualization of 2.3 defined over a finite field

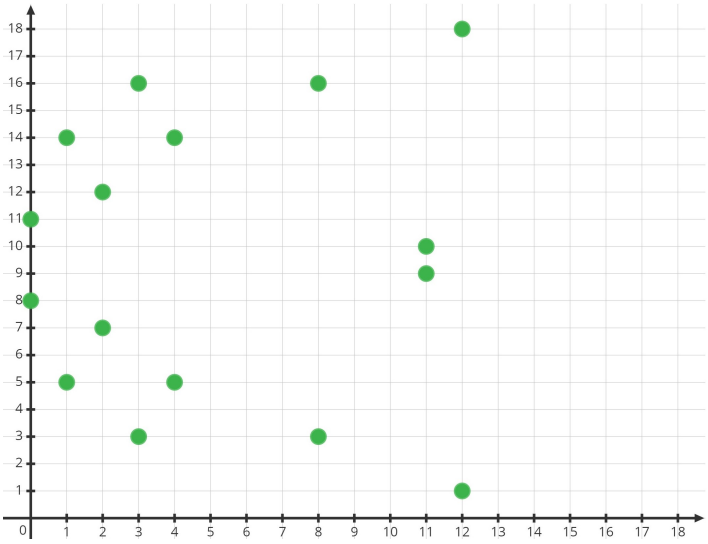


Figure 2.4. Example EC curve over finite field $\mathbb{F}p$ ($p = 19$)

Notice that this example of curve is defined on a field of 4-5 bits in real implementation as we will see there are 256 bit.
Very interesting property of this curve over finite field is cyclic algebraic group, if we add two point belonging to the curve the result is another point on the curve: $G + G = 2G$, $2G + G = 3G$ and so on, this is the base for the implementation of the Point Multiplication eq.2.8 that is the most relevant operation on EC for the key computation. ECC cryptosystems define a unique pre-defined EC point

known as the generator point G (or base point). This generator point has the remarkable property of being able to produce any other point within its subgroup on the elliptic curve by multiplying it by an integer within the range $[0...r]$,where r is the order of the soubgroup.

As mentioned before we generate the public-key multiplying the generator point for the private-key this operation is very fast but is not the same for the inverse operation called Discrete Logarithm Problem ECDLP over $\mathbb{F}p$, is similar to the invers exponentiation in $\mathbb{Z}p$, now the complexity of this algorithm is around $2^{n/2}$, considering Ed25519 curve based on 256 bit means that are necessary $2^{256/2}$ operation, quantity not acceptable for actual technology, it means that to have k bit secure strength we need $2k$ bit curve, in case of the curve just mentioned the secure strength is 128 bit.

This chapter is also an overview on the most relevant topic, in the next section are reported more precisely all the equation and hardware implementation that allow to work with EC.

2.1.3 Diffie–Hellman Key Exchange (DHKE)

All this research work is dedicated to the implementation of the cryptocore for the key-generation but it is also important define the mechanism that allow to exchange the key over insecure channel in a safe way and use them to encrypt data. One of the most relevant method is based on the Diffie–Hellman Key Exchange (DHKE), is based on the idea of excange the key between sender and receiver in exponential form, and then combining the result both of them share the same coomon encript/decript key.

The implementation is conceptually simplest, both part have a private-key we call a for the sender and b for the receiver, is then defined the dimension of the prime field p, and g that is a primitive root of p .

- sender compute the exponentiation $A = g^a \ mod(p)$
- receiver compute the exponentiation $B = g^b \ mod(p)$
- sender receive B and compute $s = B^a \ mod(p)$
- receiver receive A and compute $s = A^b \ mod(p)$

As we can see both part have calculate s that is the shared key, and it is possible because:

$$s = B^a = (g^b)^a = g^{ab} \ mod(p), \quad s = A^b = (g^a)^b = g^{ab} \ mod(p) \tag{2.6}$$

Is important to notice that a and b are secret instead p and g are know to everyone, but as explained in section before to retrieve the private-key value is necessary implement the inverse logarithm that is extremely expansive, this guarantee a secure strategy to exchange key.

Elliptic Curve Diffie–Hellman Key Exchange (ECDH)

ECDH is the same as DHKE but is adapted to work with Elliptic Curve, the mechanism is the same but instead of working whit modular exponentiation we use the operation of point multiplication, that is the aim of all this work, in the figure 2.5 is reported a schematic of ECDH working.

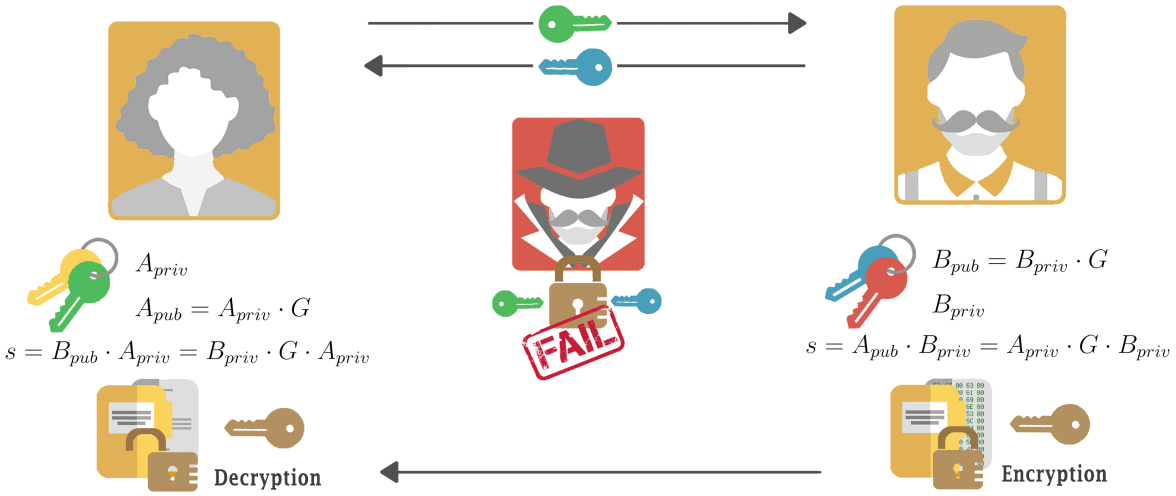


Figure 2.5. ECDH working schematic

Notation used is A_{priv} and A_{pub} to denote the private-key and public-key of the sender, and B_{priv},B_{pub} for the receiver, p is the field size and G is the generator point defined for each curve.

- Sender compute $A_{pub} = A_{priv} \cdot G$
- Receiver compute $B_{pub} = B_{priv} \cdot G$
- Sender receive A_{pub} and compute $s = A_{pub} \cdot B_{priv}$
- Receiver receive B_{pub} and compute $s = B_{pub} \cdot A_{priv}$

Both part shared the same key s now :

$$s = A_{pub} \cdot B_{priv} = A_{priv} \cdot G \cdot B_{priv}, \quad s = B_{pub} \cdot A_{priv} = B_{priv} \cdot G \cdot A_{priv} \tag{2.7}$$

This mechanism for ECDLP allow to have high level of security without share the private-key of both part. There is another interesting thing to notice, encrypting the data with the shared key s allow to have double layer of protection because instead of encrypt also whit the private key and decrypt with public key what we do implicitly is encrypt with our private key and then re-encrypt with public key of the receiver, in the same way at the receiver the decryption is done using the private key of the receiver and the public key of the sender and this is a sort of signature that confirm the source authenticity .

2.2 ECC Cryptography Hierarchy

This section is dedicated to analyse the typical hierarchy of the ECC cryptocore processor, is composed of a sequence of four hierarchical levels, as depicted in Figure 2.6.

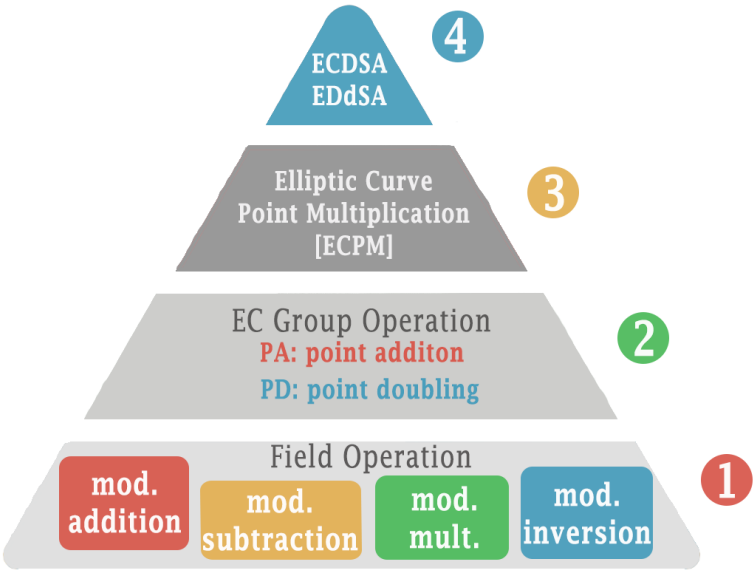


Figure 2.6. Cryptography hierarchy

The low tier contain the finite field calculations, including modular addition 3.2.1,modular subtraction 3.2.2,modular multiplication 3.3, and modular inversion 3.4. Moving on to the second level are covered operations related to elliptic curve groups, such as point addition 3.5 and point doubling 3.6, which implement various modular arithmetic operations reported in section 2.3. The third level is concerned with elliptic curve point/scalar multiplication (ECPM/ECSM), a process that integrates elliptic curve group operations sequentially that allowing to compute the public-key. Finally, the top-level includes ECC protocols, such as the elliptic curve digital signature algorithm (ECDSA) and EdDSA that are not treated in this research work.

Most predominant and time-consuming operation in ECC is ECPM, defined as:

$$Q = K \cdot G \tag{2.8}$$

where G represents a Generator base point on an elliptic curve, k is a scalar and depict the private-key, and Q is another point on the same curve that symbolize the publik-key. The primary objective of ECPM is to generate the public key multiplying the private-key with the base point on the curve. Mathematically, It's very tough to figure out the secret key by reversing the ECPM, as:

$$K = Q \cdot (G^{-1}) \tag{2.9}$$

The process of solving k from points G and Q is commonly referred to as the elliptic curve discrete logarithm problem (ECDLP)[6], which serves as an indicator for the security of ECC schemes. The security of an elliptic curve cryptosystem leveraging on the difficulty of the ECDLP and the secretly of the private key. Various algorithms are employed for ECPM, including the binary method, double-and-add method, non-adjacent form (NAF) method, the Montgomery ladder algorithm, sliding window method, and fixed-base comb method, but many of these are susceptible to side-channel attacks, In contrast, the Montgomery ladder technique enables parallel execution of point addition and point doubling, making it indistinguishable by the bit pattern of the key and thus resilient to side-channel attacks.

2.3 ECC Mathematical Background

It's time to explore the more relevant equation that govern all the operation over the elliptic curve that are the result of a very complex mathematical background, but is not relevant for our scope, it's crucial analyze the difference between the affine and jacobian coordinate to understand why the second is more efficient and allow to reduce the area and the complexity of the algorithm and tacking in consideration the fact that each little savings is important considering the nature of 256 bit parallelism of the field.

The generic equation for the Edward curve define over a prime field F_p is:

$$E_{a,d} = ax^2 + y^2 = 1 + dx^2y^2 \tag{2.10}$$

were the coefficient a,b are defined on F_p and $\in \{0,1\}$, if $a = 1$ the curve is called Untwisted Edward Curve, in other hand if we are in the opposite situation and $a = -1$ the curve is called Twisted Edward Curve, this second case is the one over our interest as we will see.

2.3.1 Twisted Edwards Curve - affine coordinates on Galois Binary Field $GF(2^n)$

Working on Galois Binary Field $GF(2^n)$ is one of the first approach used in ECC, the operation over the elliptic curve are done in two dimension starting from two point defined as $A(x_1, x_2)$ and $B(x_2, y_2)$ belonging to $E_{a,d}$. Basic operation necessary for the ECPM implementation are the point-addition and the simplified version point-doubling when the input are equal, both explained better in the following:

Unified Point-Addition

This operation allow to add two different affine point on the curve $E_{a,d}$

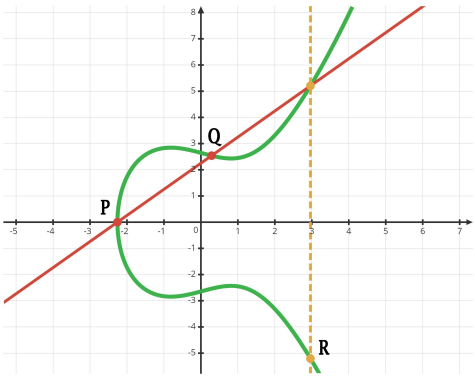


Figure 2.7. graphic point addition on EC

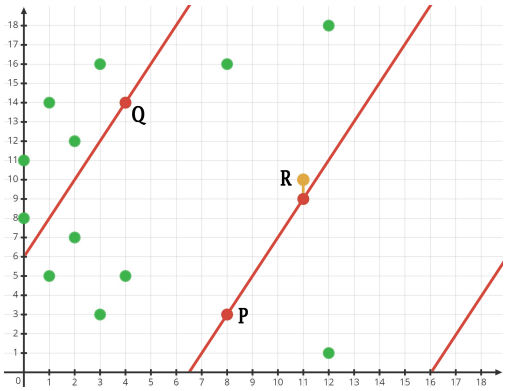


Figure 2.8. graphic point addition on EC over finite field $\mathbb{F}p$

$$P(x_1, y_1) + Q(x_2, y_2) = R(x_3, y_3) \tag{2.11}$$

$$x_3 = \frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2} \tag{2.12}$$

$$y_3 = \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2}$$

Unified Point-Doubling

This operation allow to add two equal affine point, give the nature of their equality the equation could be simplified as follow:

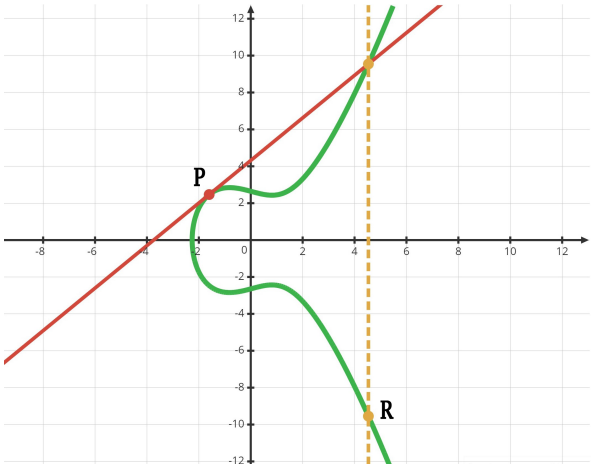


Figure 2.9. graphic point doubling on EC

$$2P(x_1, y_1) = R(x_2, y_2) \tag{2.13}$$

$$\begin{aligned} x_2 &= \frac{2x_1y_1}{y_1^2 + ax_1^2} \\ y_2 &= \frac{y_2^2 - ax_1^2}{2 - y_1^2 - ax_1^2} \end{aligned} \tag{2.14}$$

2.3.2 Twisted Edwards Curve - projective coordinates on Galois Prime Field $GF(p)$

Galois Prime Field $GF(p)$ is the second possible approach, is slightly more complex to the point of view of math derivation but allow to obtain better result in term of hardware implementation, fundamentally how is possible see in the equation reported below there is not the operation of modular inversion for each operation of PA and PD as in the case of affine coordinates, in light of the fact that is a very expensive operation as we will see in 3.4 (take n+n/4 clock cycle to the calculation), but we use it only in te final step of convert the public key result from projective to affine.

The limitation of this approach is due to the fact that working in projective coordinate means work on three dimension and is necessary implement more complex logic and more harware, but this allow to obtain better result in the end. Now we analyze what are the conversion step to move from affine to jacobian word, as reported in 2.15 the conversion is very simple and the Z coordinate is forced to $Z = 1$ instead X and Y are the same.

$$X = x, \quad Y = y, \quad Z = 1 \tag{2.15}$$

The relationship between the quantities is the following, this is especially important for the final conversion from jacobian to affine.

$$x = X/Z, \quad y = Y/Z \tag{2.16}$$

The projective form of Edward curve is obtained substituting 2.16 in 2.10:

$$E_{a,d} : (aX^2 + Y^2)Z^2 = Z^4 + dX^2Y^2 \tag{2.17}$$

Considering the case of Twisted Edward curve $a = -1$ and the final form is:

$$E_d : (-X^2 + Y^2)Z^2 = Z^4 + dX^2Y^2 \tag{2.18}$$

Unified Point-Addition

Point addition of two point $A(X_1, Y_1, Z_1)$ and $B(X_2, Y_2, Z_2)$ on $E_{a,d}$ is give by the formula:

$$P(X_1, Y_1, Z_1) + Q(X_2, Y_2, Z_2) = R(X_3, Y_3, Z_3) \tag{2.19}$$

where

$$\begin{aligned} X_3 &= Z_1Z_2(Z_1^2Z_2^2 - dX_1X_2Y_1Y_2)(X_1Y_2 + Y_1X_2) \\ Y_3 &= Z_1Z_2(Z_1^2Z_2^2 + dX_1X_2Y_1Y_2)(X_1X_2 + Y_1Y_2) \\ Z_3 &= (Z_1^2Z_2^2 + dX_1X_2Y_1Y_2)(Z_1^2Z_2^2 - dX_1X_2Y_1Y_2) \end{aligned} \tag{2.20}$$

Unified Point-Doubling

In case in which the input are equal is possible use the simplified version equation:

$$2P(X_1, Y_1, Z_1) = R(X_2, Y_2, Z_2) \tag{2.21}$$

where

$$\begin{aligned} X_2 &= (2X_1Y_1)(Y_1^2 - X_1^2 - 2Z_1^2) \\ Y_2 &= (X_1^2 - Y_1^2)(X_1^2 + Y_1^2) \\ Z_2 &= (Y_1^2 - X_1^2)(Y_1^2 - X_1^2 - 2Z_1^2) \end{aligned} \tag{2.22}$$

2.4 Twisted Edward Curve Ed25519

All the work developed is based on the definition of Elliptic Curve, as explained in the literature review exist different types of curve and for each one are defined different coefficient and equation, but the math say that is possible convert one in to the other using birational map.

2.4.1 parameter derivation

For our purpose the curve Twisted Edward Curve Ed25519 is used, this curve is derived by a Montgomery curve Curve25519 defined over prime field F_p expressed in form:

$$E_{a,b} : y^2 = x^3 + ax^2 + x \tag{2.23}$$

That curve is characterized by 128 bit security level and a prime field of $2^{255} - 19$, in table 2.1 are reporter all parameter.

Parameter	Value
p	$2^{255} - 19$
b	256 bit
order	2^{252} + 0x14def9dea2f79cd65812631a5cf5d3ed
cofactor	8
a	486662
Gx	0x0000000000000000000000000000000000000000000000000000000000000009
Gy	0x20ae19a1b8a086b4e01edd2c7748d14c923d4d7e6d7c61b229e9c5a27eced3d9

Table 2.1. Curve25519 parameter

Using the birational map equation reported in 2.24 and 2.25 we can transform this curve into the corespective Twisted Edward

$$(u, v) = \left(\frac{1+y}{1-y}\sqrt{(-486664)\cdot\frac{u}{x}}\right) \tag{2.24}$$

$$(x, y) = \left(\sqrt{(-486664)\cdot\frac{u}{v}}, \frac{u-1}{u+1}\right) \tag{2.25}$$

2.4.2 Parameter Ed25519

The Twisted Edward Curve Ed25519 derived by 2.10 is:

$$E_d : -x^2 + y^2 = 1 + \mathrm{d}x^2y^2 \tag{2.26}$$

The parameter obtained by the birational map are reported in table 2.2.

Parameter	Value
p	$2^{255} - 19$
b	256 bit
order	2^{252}+27742317777372353535851937790883648493
d	-121665/121666 =
	0x52036cee2b6ffe738cc740797779e89800700a4d4141d8ab75eb4dca135978a3
a	-1
Gx	0x216936d3cd6e53fec0a4e231fdd6dc5c692cc7609525a7b2c9562d608f25d51a
Gy	0x6666666666666666666666666666666666666666666666666666666666666658

Table 2.2. Ed25519 parameter

Now is possible implement the ECPM architecture to obtain the 256 bit Public Key.

Chapter 3

Proposed Hardware Architectures

An elliptic curve cryptosystem can be realized through either hardware or software approaches. In this research, our exclusive focus is on the hardware approach, as hardware implementation offers significantly enhanced performance in comparison to software implementation. However, achieving a hardware implementation of ECC with minimal hardware consumption and low time complexity presents a substantial challenge. Area and time represent two conflicting parameters for an ECC processor, requiring a compromise between them to achieve heightened efficiency in terms of the other. To optimize performance, the area-time (AT) product of the processor should be minimized. This research effort aims to design an ECC processor well-suited for high-speed, low-power cryptographic devices by reducing both computation time and the area footprint required for ECPM.

In this chapter i will present all the hardware block implemented and eventually their evolution during the design, tacking special care for the timing that is essential to achieving the better performance as possible in the moment in which the overall architecture is assembled. The presentation is organized in a hierarchic way, start with the presentation of the adder architecture that is the elementary block used to implement modular arithmetic operation, conversely are used to implement point addition and point doubling that converge in the top ECPM block.

3.1 elementary adder

First section is dedicated to the adder,is very important search better implementation since the core work with 256 bit key and contribute in massive way to the critical path and on the area balancing. Two subsection are dedicated because of the fact that the first one is for the ASIC implementation and require all the specification for on-silicon realization, second one have as target the utilization of DSP integrated on the FPGA to optimize the syntheses and reduce the allocation of LUT.

3.1.1 Carry Look Ahead(CLA) - for ASIC implementation

For the implementation of the Adder that will be synthesize on silicon is necessary have fast adder that execute all the operation in only one clocks cycle, this is essential to limited the latency of the core, because there is the risk that we are not able to maintain a critical path comparable with the one of the elementary adder in the other part of the architecture, and considering the large amount of clock cycle necessary, the total execution time could be very worst.

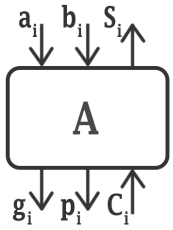
According to this the choice the architecture selected is the Carry-look-Ahead Adder (CLA),his force is the ability to generate carry in parallel, enabling fast bit addition. This approach differs from the ripple carry adder (RCA), where the carry is sequentially propagated from one bit to another. The carry-look-ahead adder employ a series of carry generate and propagate to predict the carry value in advance at each position, without having to wait for propagation from the least significant bit to the most significant one. This parallel design makes CLA significantly faster than the ripple carry adder, especially when we are working with strong word lengths.

CLA implementation

For the Hardware implementation there are fundamentally two approach, i try to implement both of them but then i preferred the second one because allow to obtain better performance and less area.

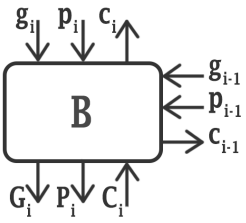
First approach use also elementary block called A and B, in block A is computed the generate g and propagate p of two bit, block B instead compute the combination of two generate and two propagate , in figure 3.1, and 3.2 are reported the schematic and the implemented equation, combining this two block in a proper tree way is possible implement very large adder as will explain later. Second approach instead using also another block called C that compute the combination generate and propagate of four p and g input, obviously the equation used are studied a doc to optimize the process, figure 3.3 illustrate the the schematic.

How is possible to see the block work in two phase, first the computation of signal generate and propagate move from top to the bottom, second the carry start is propagation from bottom and move to the top, this means that the tree is traversed two time in both direction



$$
\begin{aligned}
g_i &= a_i \; b_i \\
p_i &= a_i \oplus b_i \\
S_i &= a_i \oplus b_i \oplus C_i
\end{aligned}
\tag{3.1}
$$

Figure 3.1. architecture A block



$$
\begin{aligned}
G_i &= g_i + g_{i-1} \; p_i \\
P_i &= p_i \; P_{i-1} \\
c_i &= g_{i-1} + p_{i-1} \; C_i \\
c_{i-1} &= c_i
\end{aligned}
\tag{3.2}
$$

Figure 3.2. architecture B block

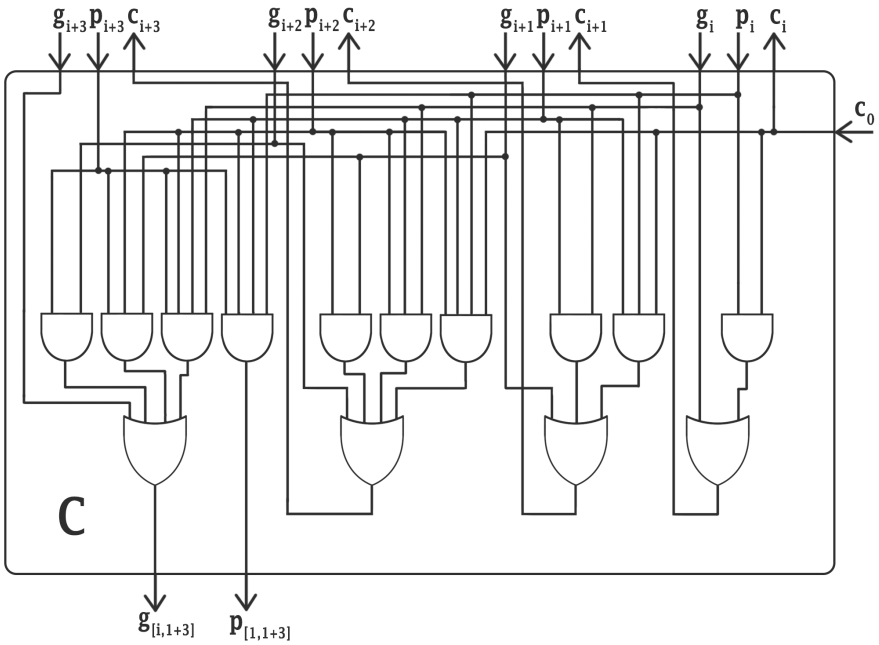


Figure 3.3. architecture C block

defined the elementary blocks is time to develop the overall tree architecture, the architecture proposed is relate to the second approach , the first one follow the same logic but instead of using 4-input block as mono-block create it using three B block. Implementation follow a hierarchic approach, in practise is implemented first of all a 4-bit Adder as in fig. 3.4, and combining four of this block is obtained a 16 bit adder fig.3.5

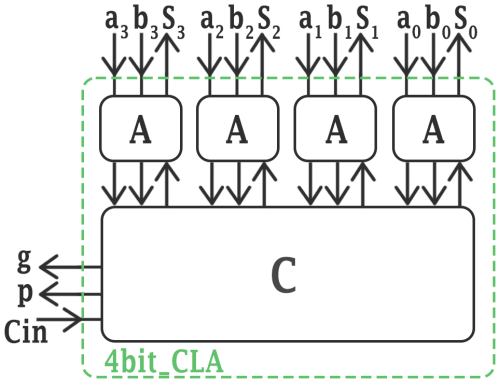


Figure 3.4. architecture 4bit CLA

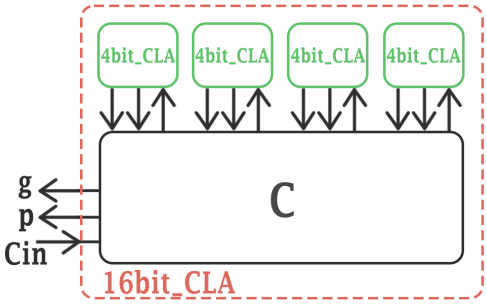


Figure 3.5. architecture 16bit CLA

Four of this block are combined to create a 64-bit adder as in figure 3.6 and in conclusion in turn other four block are combined to create a 256-bit adder as in figure 3.7

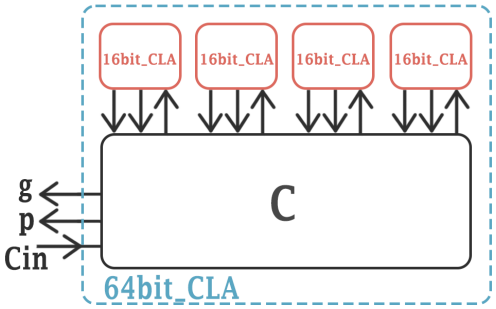


Figure 3.6. architecture 64bit CLA

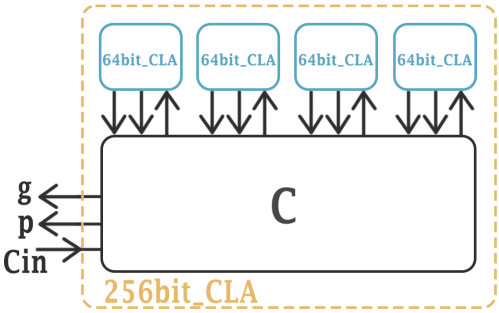


Figure 3.7. architecture 256bit CLA

Analyzing the architecture proposed in next section is clear that to take in consideration sign and overflow are not sufficient 256 bit but is necessary extend the parallelism to 257 bit for the modular adder/subtractor, and 258 bit for the modular multiplier and modular inversion, it is therefore necessary increase the parallelism of the adder in this two case, to do this is possible extend the tree to the right or to the left without distinction but For convenience and to minimize changes i have chose the left approach as explained in figure 3.8, 3.9

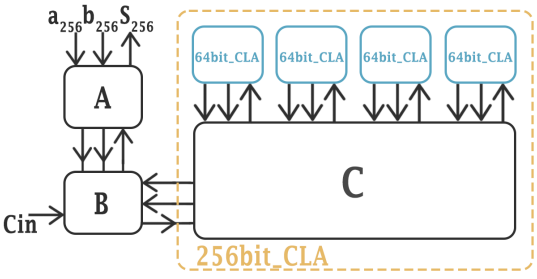


Figure 3.8. architecture 257bit CLA

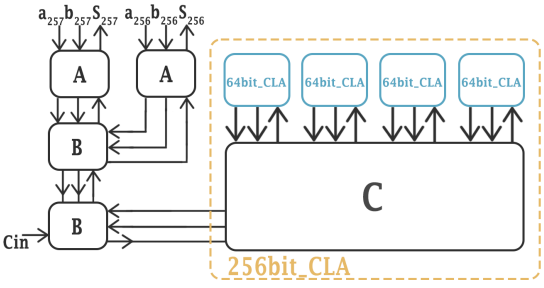


Figure 3.9. architecture 258bit CLA

3.1.2 Carry Select Adder(CSA) using DSP - for FPGA synthesis

In this section is threated the implementation of the elementary adder using DSP available on the Virtex-7 FPGA, this is necessary to reduce the utilization of the LUT and in consequence area and the speed, to reach the better implementtation i tra different way starting from the more simplest, the Ripple Carry Adder As in RTL figure 3.10

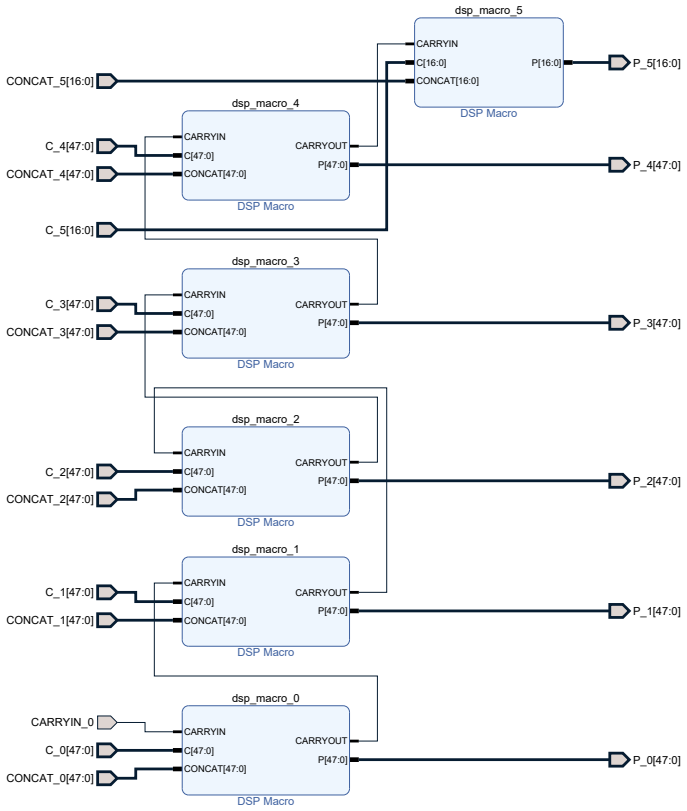


Figure 3.10. RTL view of RCA adder implemented with DSP

As is possible to see are allocated six DSP in which five are implemented with the maximum parallelism of 48 bit, and the last one with 17 bit for a total of 257 bit, the same is done for the 258 bit. Being an RCA adder simply the carry in output from the first stadium propagate along the chain block per block, final result is a very long propagation delay, i notice that i have disabled all the six default pipeline level to respect the timing of overall system. Aware of the fact that is a worst result i try to implement the Carry-Select-Adder (CSA), also in this case i develop may idea in successive step, initially using three level architecture, because i think that increasing the number of multiplexer could be counterproductive considering the high working parallelism, but after the compilation the results are better respect to the RCA but not satisfying, although i decide to implement four-level architecture that is the maximum possible using 48-bit DSP and now the result is good, around $2.7ns$ of critical path. In figure 3.11 is reported the generic architecture of 4-level CSA, in which the carry is identified by the green line and result by the black, instead in fig. 3.12 the RTL implementation,

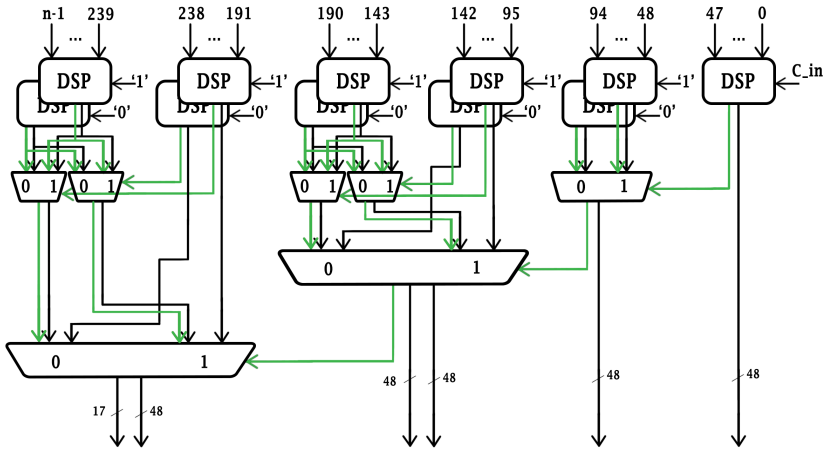


Figure 3.11. architecture CSA 4-level

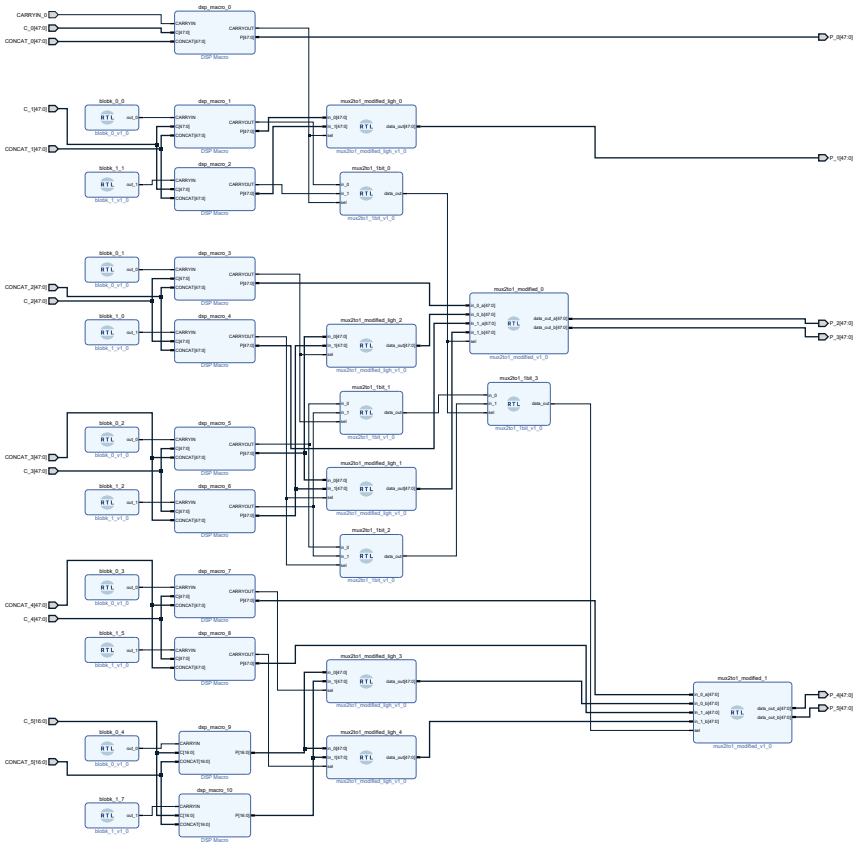


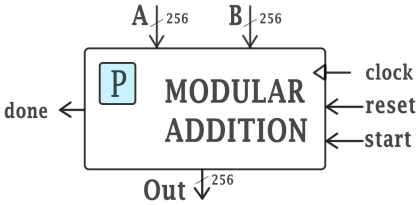
Figure 3.12. RTL view of CSA 4-level adder implemented usng DSP

Using this approach each block calculate the result in both case of $carry = 1$ or and $carry = 0$, then using multiplexer only the correct result according to the carry in output from previous stage can pass trough, this idea could be extended following a tree to high parallelism, to have better performance i dedicate a one-bit multiplexer for the carry, doing this carry propagation is more fast respect pass trough the MUX of the result and enabling the next level of multiplexer to start earlier.

3.2 modular addition/subtraction

In this section are analyzed the modular operation of addition ad subtraction, the concept is very simple, in practice every time we need addition or subtract two number is necessary that the result is confined inside the prime field $\mathbb{F}p$ that could be seen as a watch, every time is reached the end we restart from the beginning and so on.

3.2.1 modular addition



Architecture proposed in fig. 3.13 is used to implement the modular addition, first operation is the addition of two input using the adder proposed in section 3.1, parallelism is extended to 257 bit to preserve the integrity of the result, next step is finite field $\mathbb{F}p$ dimension p subtraction, then a sign check is done, if the sign is negative means $MSB = 1$ the multiplexer enable the passage of $S1$ result because means that it is just confined over the field, otherwise if the sign bit is 0 multiplexer give $S2$ to become the output. Done signal is start signal retarded of one clock cycle to allow the alignment with the result.

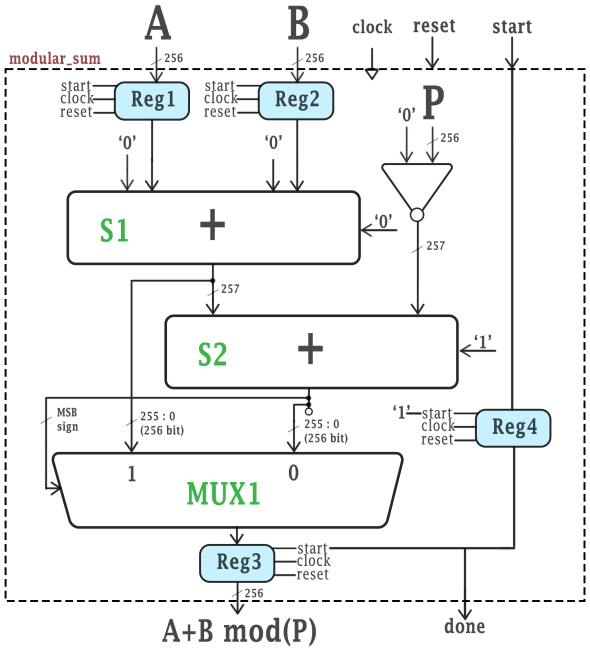
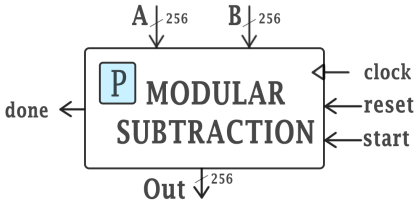


Figure 3.13. schematic modular addition

features:
Cost: 2 Adder + 1 MUX(256to1) + 1 Inv. + 4 Reg.
Critical Path: 2 T_{add} + T_{MUX} + T_{reg}
Latency : 1 T_{ck} + 1

3.2.2 modular subtraction



In this second section is presented the Modular Subtractor fig. 3.14, the working is very similar to the Modular Adder, first step is compute the subtraction of the input also in this case the parallelism is extended to 257 bit to preserve the integrity of the result, his MSB is used as the selector for the multiplexer, if the sign bit is 0 means that the result is positive and in consequence is defined on the prime field $\mathbb{F}p$ for definition and it comes out directly as definite output, otherwise if $sign = 1$means that the subtraction give negative result and it need to be reported on the prime field adding one time his dimension p.

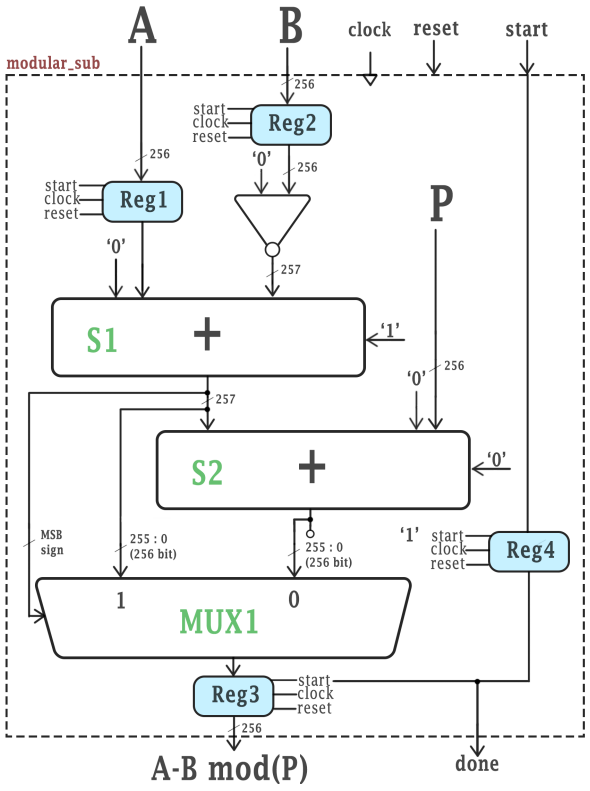
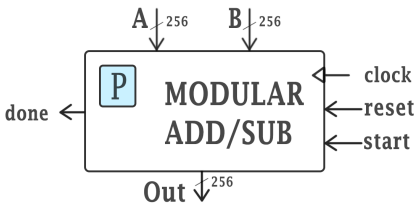


Figure 3.14. schematic modular subtraction

features:
Cost: 2 Adder + 1 MUX(256to1) + 1 Inv. + 4 Reg.
Critical Path: $T_{Inv} + 2\ T_{add} + T_{MUX} + T_{reg}$
Latency : $1\ T_{ck} + 1$

3.2.3 modular sum-subtraction



Third and last part of this section is dedicated to another circuit that combine the operation of Modular Addition and Modular Subtraction, i decide to define this architecture because my research is strongly focused on reducing the number of allocated resources, as we will see wen we talk about the operation of Point Addition and Point Doubling i have done an accurate scheduling job and to reduce more and more the resources block this architecture provide assistance.

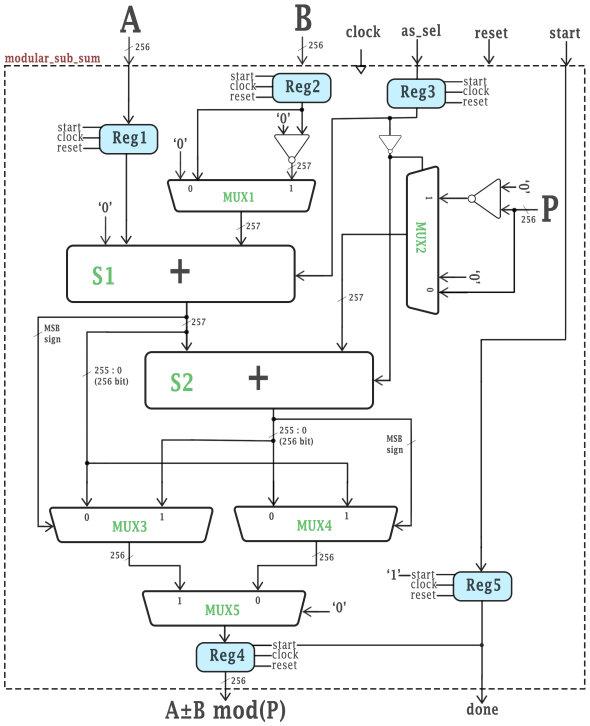


Figure 3.15. schematic modular additon/subtraction

The working of this circuit is the same as explained for Mod. Addition and Subtraction but there is one additional signal the as if $as = 1$ $mux1$ let it pass the negate version of B and it means we implement the subtraction, next step is check if the result is positive or negative according to the sign bit of the result, and in case it is negative $mux2$ define the second input of $S2$ as the not negate value of p and we add it to report the result on the finite field otherwise if the sign is positive the result is already over the field, then the result go in output thank to $mux3$ derived by the sign. opposite situation is when $as = 0$ in this case $mux1$ let it pass unaltered version of B and it means we implement the addition, next step is check if is out of the filed and in case subtract the field dimension P and then tanks to $mux4$ is decided which is the correct result. In both case $mux5$ driven by as select the correct output.

features:
Cost: 2 Adder + 5 MUX + 3 Inv. + 5 Reg.
Critical Path: $T_{Inv} + T_{MUX} + 2\ T_{add} + 2\ T_{MUX} + T_{reg}$
Latency : $1\ T_{ck} + 1$

Apparently could be worst this architecture but considering the cost of the single adder this structure save two adder instead of allocate two different structure for the implementation of Mod. Addition and Subtraction. Obviously could operate only one operation per time and is necessary have the scheduling possibility to work in this condition

timing

in this paragraph is reported the timing diagram of all the three architecture propose in this section, Modular Addition 3.2.1, Modular Subtraction 3.2.2 and Modular Addition/Subtraction 3.2.3 in which are reported the behaviour of the most relevant signal.

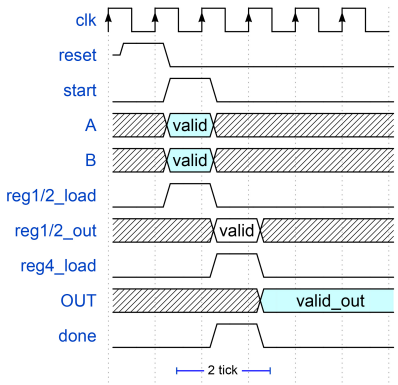


Figure 3.16. timing Modular Addition/Subtraction

VHDL simulation and Matlab validation

In this section is reported the Matlab validation code used to verify the correctness and the simulation of VHDL code done using Modelsim

```
1
2  % field dimension 0x52036cee2b6ffe738cc740797779e89800700a4d4141d8ab75eb4dca135978a3
3  p = sym('57896044618658097711785492504343953926634992332820282019728792003956564819949');
4  %input
5  a = sym('26826903516534128576205990732449246204549756080988166989461034734729464335973');
6  b = sym('57896044618632535464341535436436578412269754886254878645315168469699635187546');
7  %output
8  out = dec2hex(mod(a+b, p)); %result converted in hexadecimal
9  %--------
10 %out = 3B4F7D4345F8078BB201CEBF9AE65CDE6D55CB91B7A23DC13380EB8009A35DD2
```

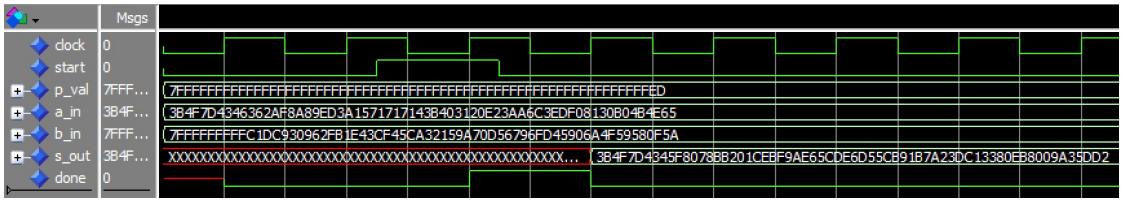


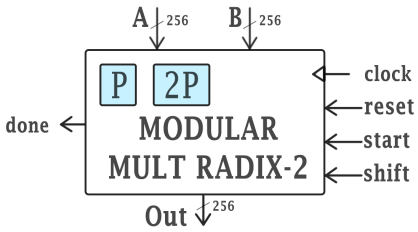
Figure 3.17. VHDL Modular Addition simulation

Results are the same, means Modular Adder works well

3.3 modular multiplication

In this section is explained the architecture of Modular Multiplication, fundamentally the research work is starting with the implementation of the RADIX-2 interleaved modular multiplication but according to the fact that are necessary n clock cycle to handle the operation i try to implement the RADIX-4 architecture that use only the half of clock cycle $n/2$ about the critical path tanks to a precomputation stadium and revision of the architecture i obtain a result very similar to the one given by RADIX-2, is clear that the best choice is the RADIX-4 but i implement all the operation using both of them to have a better comprehensive comparison.

3.3.1 RADIX2 - interleaved modular multiplication



In this architecture is prresented the RADIX-2 implementation , schematic is reported in 3.19, the idea behind is compute partial product multiplying each clock cycle factor A by bit a bit B, to do this A is memorized in a register, factor B is saved in a shift register that each clock cycle shift left of one position and the gap is filled with '0', there is also a '1' added to the LSB this is useful because to detect the end of the multiplication is allocated a OR port that take in input $T[n-1$ to $0]$ output bit of the shift register and the result is 1 until the '1' go out of this set, at this point OR output is 0 and means the multiplication in finish instead done signal is asserted to '1'. $MUX1$ is driven by the output bit of shift register let it pass or the value of previous partial result D multiply by two according to the multiplication rule proposed in pseudo-code Algorithm 1 if the bit in '0', otherwise if the bit is '1' the MUX output is the addition of the previous partial result D and the factor A. Last step is report the result over the finite field $\mathbb{F}p$ to do this is necessary remember that partial result could be at maximum three time the field dimension p it means that in this case to report the result on the field is necessary subtract two time p otherwise one only one time or noting, all of this operation are done in parallel, then according to the sign of the result $MUX2$ let pass only the correct result defined on the field. $REG2$ sample each time instead the OR output is one, but in last cycle signal in 0 and the correct result of multiplication stays memorized on the output.

Algorithm 1 Radix-2 Interleaved Modular Multiplication

Require: $A = \sum_{i=0}^{n-1} a_i 2^i, B = \sum_{i=0}^{n-1} b_i 2^i, p = \sum_{i=0}^{n-1} p_i 2^i; \quad a_i, b_i, p_i \in \{0,1\}$
Ensure: $D = (A \cdot B) \mod p$
1: $D \leftarrow 0$;
2: $T \leftarrow B$& '1';
3: **while** $T(n-1$ downto $0) \neq$ '0' **do**
4: $D \leftarrow 2C$;
5: **if** $T_n = 1$ **then**
6: $D \leftarrow D + A$;
7: **end if**
8: $D \leftarrow D \mod p$;
9: $T \leftarrow (T_{n-1}$ downto $T_0)$& '0'; // **left-shift operation**
10: **end while**
11: **return** D;

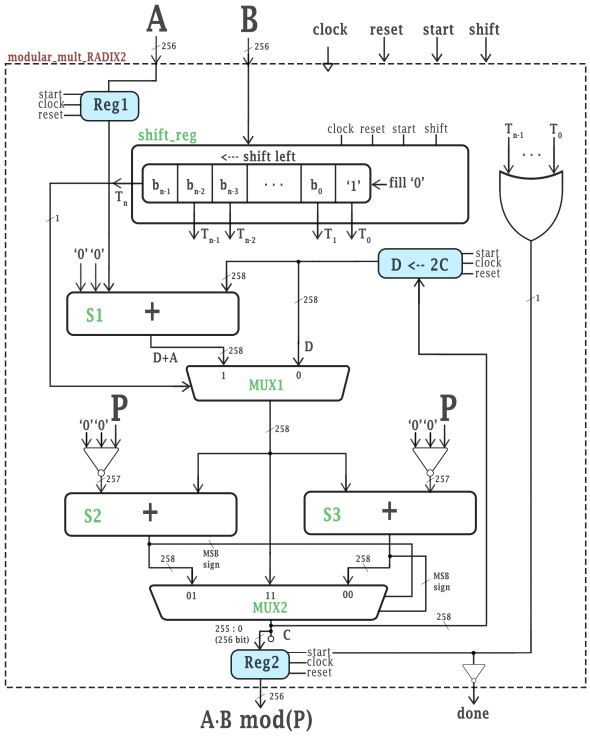


Figure 3.18. timing modular multiplication RADIX-2

The parallelism is defined on 258 bit to preserve the integrity of the signal, one extra bit is for the overflow, and it is sufficient to represent at maximum three time the field dimension $\mathbb{F}p$, other one extra bit is for the sign used by $MUX2$

features:
Cost: 3 Adder + 2 MUX + 2 Inv. + 4 Reg + 1 Shift Reg.
Critical Path: $T_{add} + T_{MUX} + T_{add} + T_{MUX} + T_{reg}$
Latency : $n + 1\ T_{ck} = 257\ T_{ck}$

timing

In fig. 3.19 is reported the timing diagram of Modular Mult. RADIX-2

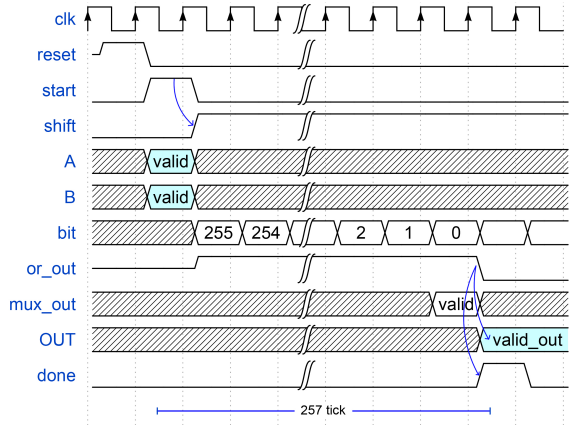


Figure 3.19. schematic modular multiplication RADIX-2

Matlab validation

In this section is reported the Matlab validation code used to verify the correctness

```
1
2  % field dimension 0x52036cee2b6ffe738cc740797779e89800700a4d4141d8ab75eb4dca135978a3
3  p = sym('57896044618658097711785492504343953926634992332820282019728792003956564819949');
4  %input
5  a = sym('814768362534039048724533725658258822622461771343307800652438953110765787357 1');
6  b = sym('151122213495354007725011514095885315114540126930418572060461132839498477622 02');
7  %output
8  out = dec2hex(mod(a*b, p)); %result converted in hexadecimal
9  %--------
10 %out = 803644543CFA1951232C9C47ED0EA89B11ECB00D70501C0B2AAFC2E24C541E4
```

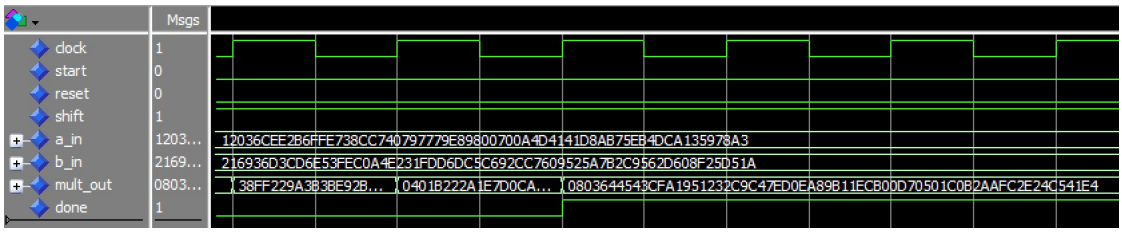
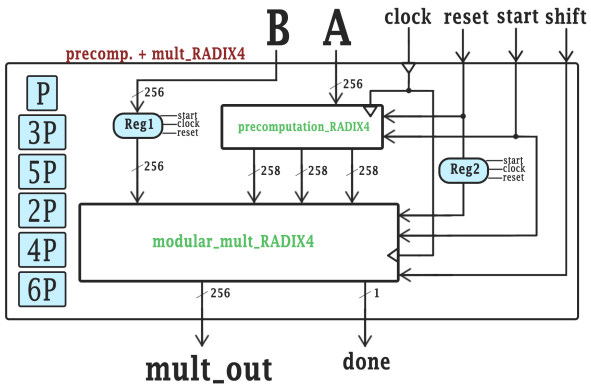


Figure 3.20. VHDL Modular RADIX-2 Multiplication simulation

Results are the same, means Modular Multiplier works well

3.3.2 RADIX4 - interlived modular multiplication



In this architecture is presented the RADIX-4 implementation, the idea behind is the same described before but with a little change, compute partial product multiplying each clock cycle A by a couple of bit at time from B. to do this A is memorized in a register, factor B is saved in a shift register that each clock cycle shift left of two position and the gap is filled with '0', there is also a '01' added to the LSB this is useful because to detect the end of the multiplication is allocated a OR port that take in input T[n - 1 to 0] output bit of the shift register and the result is '1' until the '1' go out of this set, at this point OR output is '0' and means the multiplication in finish instead done signal is asserted to 1. $MUX1$ is driven by two output bit of shift register, this allow to have four case, if sel is '00' means that the new partial result is D, the same as cycle before but shifted left of two position according to the multiplication rule explained in Algorithm 2, in case of sel ='01' means that to the partial result we add one time the factor A, the same rule is valid for sel='10' means addition two time A and in the end sel='11' means adding three time A. Bottom half of the implementation is reported

in two different version explained in paragraph below. Another important consideration is relative to the precomputed value in input to $S1$, $S2$, $S3$ adder, i implement a dedicated block reported in fig. 3.21 for their computation, because inserting another adder block in series increase the critical path of the overall core since of this block is the globally critical one, as we will see in timing for the precomputation there is one clock cycle dedicate.

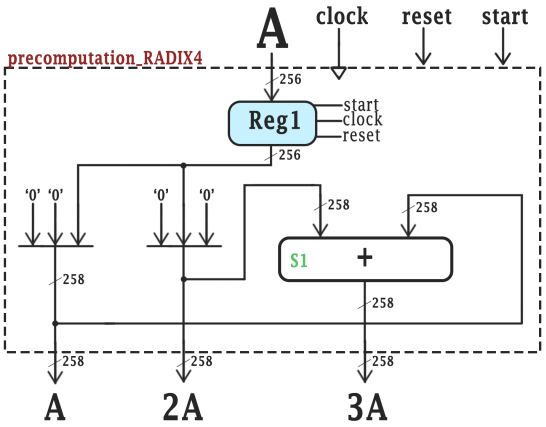


Figure 3.21. precomputation modular multiplication RADIX-4

Algorithm 2 Proposed Radix-4 Interleaved Modular Multiplication

Require: $A = \sum_{i=0}^{n-1} a_i 2^i, B = \sum_{i=0}^{n-1} b_i 2^i; \quad a_i, b_i \in \{0, 1\}$
Ensure: $D = (A \cdot B) \mod p$

```
1:  D ← 0;
2:  T ← B&"01";
3:  while T(n − 1 downto 0) ≠ 0 do
4:     D ← 4D;
5:     if T(n + 1 downto n) = "01" then
6:        E ← D + A;
7:     else if T(n + 1 downto n) = "10" then
8:        E ← D + 2A;
9:     else if T(n + 1 downto n) = "11" then
10:       E ← D + 3A;
11:    else
12:       E ← D;
13:    end if
14:    D ← E mod p;
15:    T ← (T_{n−1} downto T_0)&"00"; // left shift operation
16: end while
17: return D;
```

first implementation This is the first version [7], proposed in fig. 3.22 as in the case of RADIX-2 is necessary report the partial result on the finite field $\mathbb{F}p$ but now the partial result could be at maximum seven time the field dimension p, it means that is necessary define all the possible intermediate case in which we subtract one time the field dimension p, two time, and so on up to six, doing this we guarantee to have the partial result over the field. at this point using $MUX2$ is necessary allow to pass only the correct result, to do this the idea is drive the selector with the three most significant bit of the partial result in output to $MUX1$ if are '000' means that we are already over the field and the partial result pass without change, if sel='001' output is the partial result less p, and so on for each other case. $R4$ sample each result in output instead of the OR output become '0' at this point the modular multiplication is terminated.

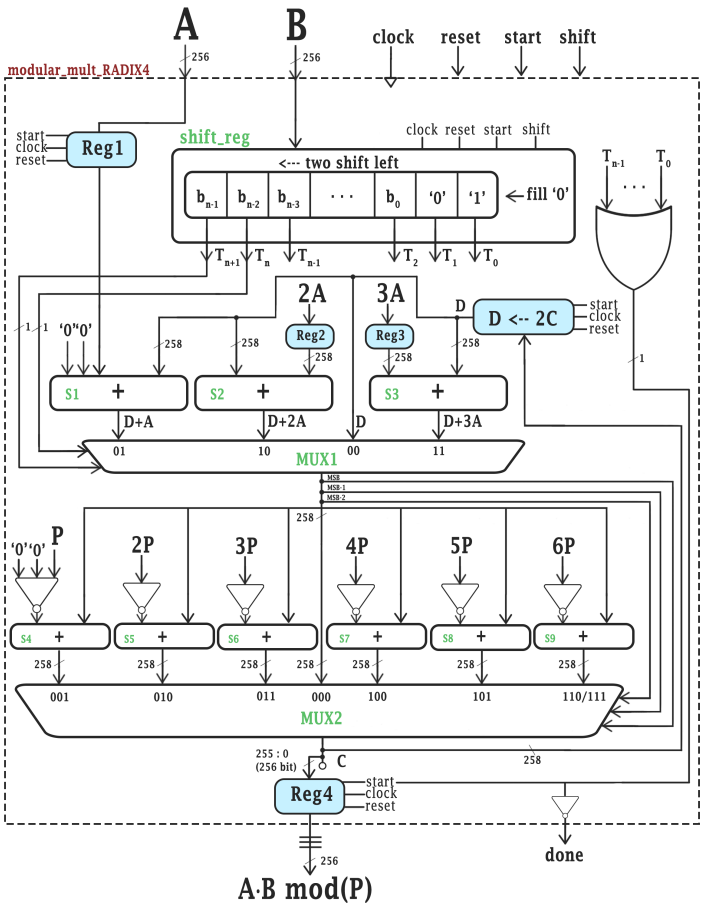


Figure 3.22. VHDL Modular Multiplication first version

This architecture proposed have a problem that i discover during the test with high input close to the maximum field value, in practise the multiplexer $MUX2$ use only the three MSB to decide which is the output, but the finite field dimension P has a value of $2^{255} - 19$, it means that does not saturate the 256 bit dynamic and if the output of the $MUX1$ is a little bit grater respect this value the three MSB are the same but the value is in the next cycle of the field. In conclusion the output result could be defined on two time p and not only on p to solve this problem is necessary allocate another block in proximity of the symbol "$\equiv$" nearness to the output in fig.3.22. Extra block in exam is reported in fig.3.23. it is simply an adder that subtract the field dimension p and then if the sign is positive means that the result was out of bound and now is reported over the field, in case it is negative means that it was just defined over the field and no modification are needed.

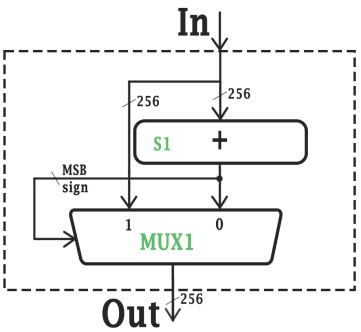


Figure 3.23. check extra block

The parallelism is defined on 258 bit to preserve the integrity of the signal, one extra bit is for the overflow, and it is sufficient to represent at maximum three time the field dimension $\mathbb{F}p$, other one extra bit is for the sign used by $MUX2$

features:
Cost: 10 Adder + 3 MUX + 7 Inv. + 5 Reg + 1 Shift Reg. + 1 OR gate.
Critical Path: T_{add} + 1 T_{MUX} + T_{add} + T_{MUX} + T_{reg}
Latency : $\frac{n}{2} + 2\ T_{ck} = 130\ T_{ck}$

second implementation In this second implementation is presented an optimized architecture finalized to reduce the area, the upper half of the architecture is the same as before but the modification is applied to the bottom half, instead of having six adder allocate i use the $MUX2$ only to select the correct value of field multiple mp to subtract from the partial result in output to $MUX1$ and in consequence there is only one adder allocate instead of six, considering the huge cost in term of area of the elementary adder there is a reduction of the overall area around 45% that is a very good result. Also in this case in correspondence of symbol ≡ is located the block 3.24 to conduct a validation check of the result inclusion within the field.

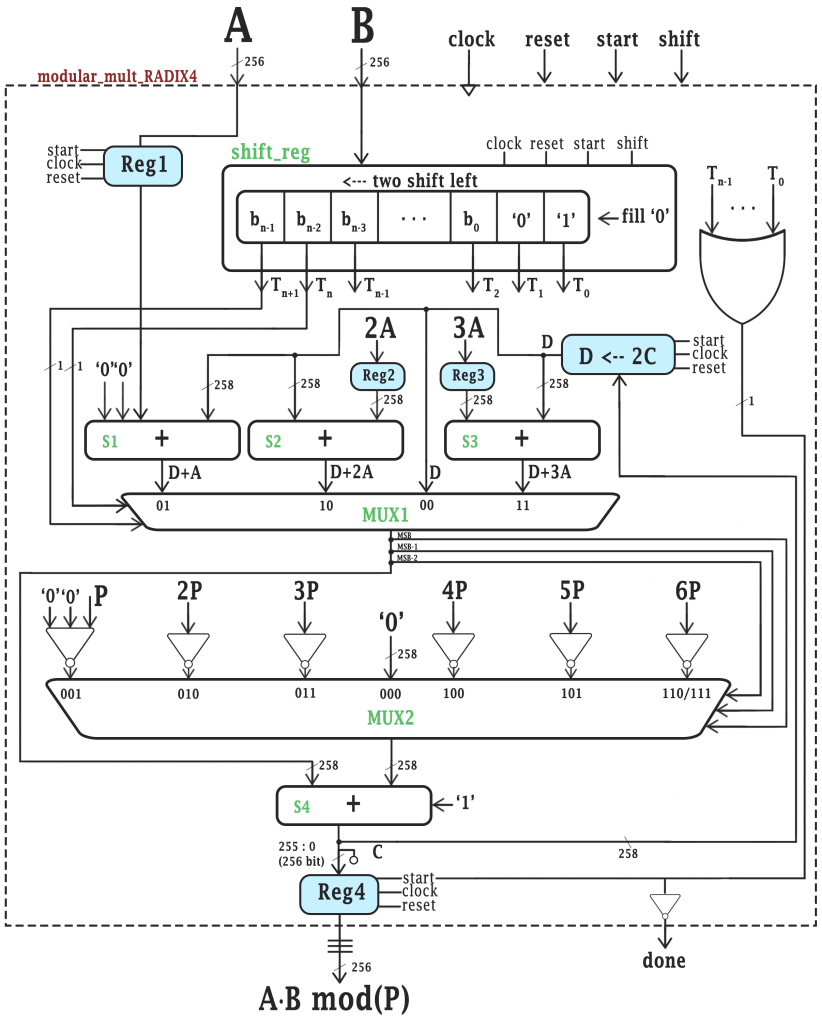


Figure 3.24. modular multiplication RADIX4 second version(definitive)

features:
Cost: 5 Adder + 3 MUX + 7 Inv. + 5 Reg + 1 Shift Reg. + 1 OR gate.
Critical Path: $T_{add} + 2\ T_{MUX} + T_{add} + T_{reg}$
Latency : $\frac{n}{2} + 2\ T_{ck} = 130\ T_{ck}$

About parallelism is defined on 256 bit for the same reason as RADIX-2

timing

In fig. 3.25 is reported the timing diagram of Modular Mult. RADIX-4. How is possible to see there is one dedicated clock cycle to the precomputation of the factor, then $\frac{n}{2} + 1$ clock cycle for the computation of the multiplication.

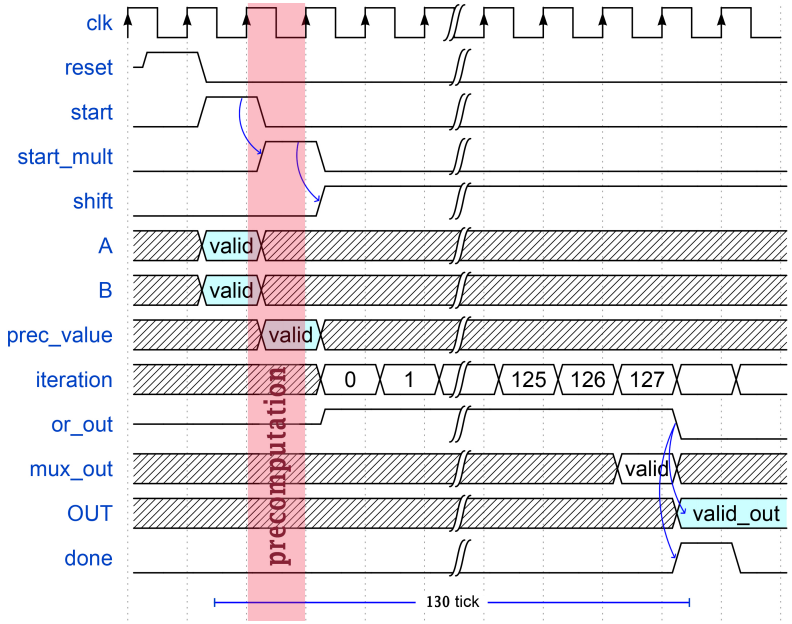


Figure 3.25. timing RADIX-4 multiplication

VHDL and Matlab validation

In this section is reported the Matlab validation code used to verify the correctness and the result of the VHDL test done with Modelsim

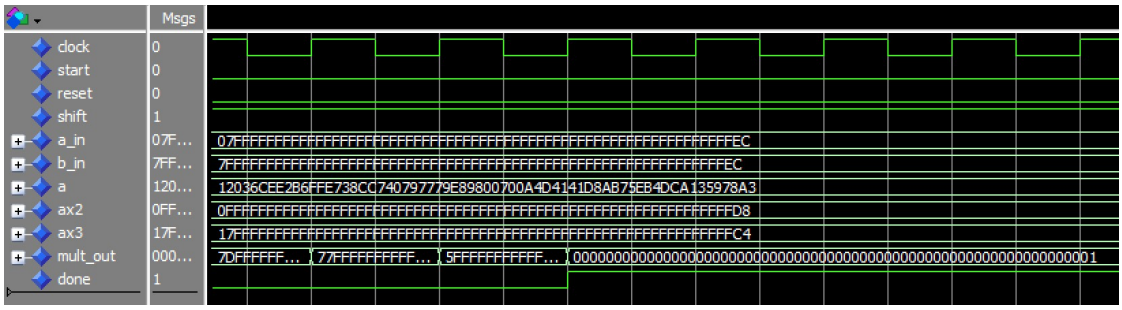
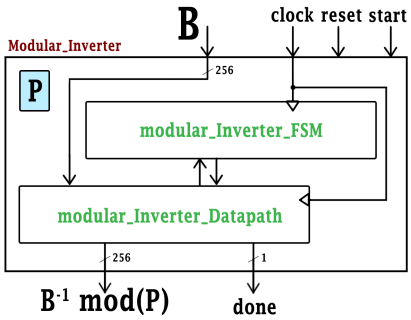


Figure 3.26. VHDL Modular RADIX-4 Multiplication simulation

```
1
2  % field dimension 0x52036cee2b6ffe738cc740797779e89800700a4d4141d8ab75eb4dca135978a3
3  p = sym('57896044618658097711785492504343953926634992332820282019728792003956564819949');
4  % input
5  a = sym('57896044618658097711785492504343953926634992332820282019728792003956564819948');
6  b = sym('57896044618658097711785492504343953926634992332820282019728792003956564819948');
7  B = dec2bin(b,256);
8  reg_x4 = 0;
9  n=0;
10
11 for i = 1:2:256
12     if i + 1 <= 256
13         bits = B(i:i+1);
14         %fprintf('Bits estratti: %s\n', bits);
15          fprintf('partial result: %d\n',n);
16
17       if bits == '01'
18           mux1_out = a+reg_x4;
19           fprintf('mux_1: %s       mux1_out: %s \n',bits,dec2hex(mux1_out));
20           fprintf('mux1_out: %s \n',dec2bin(mux1_out,258));
21       end
22       if bits == '10'
23           mux1_out = 2*a+reg_x4;
24           fprintf('mux_1: %s       mux1_out: %s \n',bits,dec2hex(mux1_out));
25           fprintf('mux1_out: %s \n',dec2bin(mux1_out,258));
26       end
27       if bits == '11'
28           mux1_out = 3*a+reg_x4;
29           fprintf('mux_1: %s       mux1_out: %s \n',bits,dec2hex(mux1_out));
30           fprintf('mux1_out: %s \n',dec2bin(mux1_out,258));
31       end
32       if bits == '00'
33           mux1_out = reg_x4;
34           fprintf('mux_1: %s       mux1_out: %s \n',bits,dec2hex(mux1_out));
35           fprintf('mux1_out: %s \n',dec2bin(mux1_out,258));
36       end
37
38       mux2_out = mod(mux1_out,p);
39       fprintf(' %s\n\n', dec2hex(mux2_out));
40       reg_x4 = mux2_out*4;
41       n=n+1;
42
43     else
44         bits = B(i);
45         fprintf('remaining bit: %s\n\n', bits);
46     end
47 end
48
49 %--------
50 %out = 0000000000000000000000000000000000000000000000000000000000000001
```

In this test i try the test with two input value at the bound of maximum dimension of the field p to verify the correct working also in worst case condition, and result are the same as the software simulation, means that circuit work well. Notice also the plot of precomputated value nominate a, $ax2$ and $ax3$.
Matlab code is implemented following exactly the same algorithm used in the hardware and proposed in the pseudocode to have all the intermediate value available and test in the better way as possible. Final consideration is that with RADIX-4 multiplier we have halved the number of clock cycle necessary and the critical path is accurately maintained the same as before, is clear that this one is the best implementation.

3.4 modular inversion



In this section is presented the modular inversion block, how we can see is very expensive and require many clock cycle to perform the calculation, and this is one of the reason why we have move in the direction of working with projective coordinate and not with the affine that require one of this operation for each point addition and point doubling.

3.4.1 Modular inversion Architecture

Algorithm 3 Proposed Modular Inverter

Require: $B = \sum_{i=0}^{n-1} b_i 2^i; \quad b_i \in \{0,1\}$
Ensure: $C = B^{-1} \mod p$

$C \leftarrow 0, q \leftarrow B, r \leftarrow p, s \leftarrow 1, t \leftarrow 0;$
while $q \neq 1$ **do**
  while $q(0) = 0$ **do**
    $q \leftarrow q/2;$
    if $s(0) = 0$ **then**
      $s \leftarrow s/2;$
    else
      $s \leftarrow (s+p)/2;$
    end if
  end while
  while $r(0) = 0$ **do**
    $r \leftarrow r/2;$
    if $t(0) = 0$ **then**
      $t \leftarrow t/2;$
    else
      $t \leftarrow (t+p)/2;$
    end if
  end while
  if $q > r$ **then**
    $q \leftarrow q - r;$
    if $s > t$ **then**
      $s \leftarrow s - t;$
    else
      $s \leftarrow s + p - t;$
    end if
  else
    $r \leftarrow r - q;$
    if $t > s$ **then**
      $t \leftarrow t - s;$
    else
      $t \leftarrow t + p - s;$
    end if
  end if
end while
return $s \mod p;$

The algorithm behind this architecture is very complex and it would require a very in-depth mathematical analysis, but that is not the purpose of this research, and i only present the pseudo-code in Algorithm 3 and the implementation. Generally this schematic is divided in four different block, each one with with a register called respectively $RegQ$, $RegR$, $RegS$, $RegT$ and the are done some calculation according to the algorithm, input multiplexer $MUX1$, $MUX2$, $MUX3$, $MUX4$, select the first value to charge in the register and then switch to connect the feedback for next iteration, then there is the final block adder $S9$ that perform a check to verify if $q < 2$ if this condition is reached means that the result of inversion is ready, otherwise means that is necessary do other calculation. $S10$ as usual compute the check to verify that the result is defined on the finite field $\mathbb{F}p$ subtracting the field dimension p and the tacke the decision using the sign.

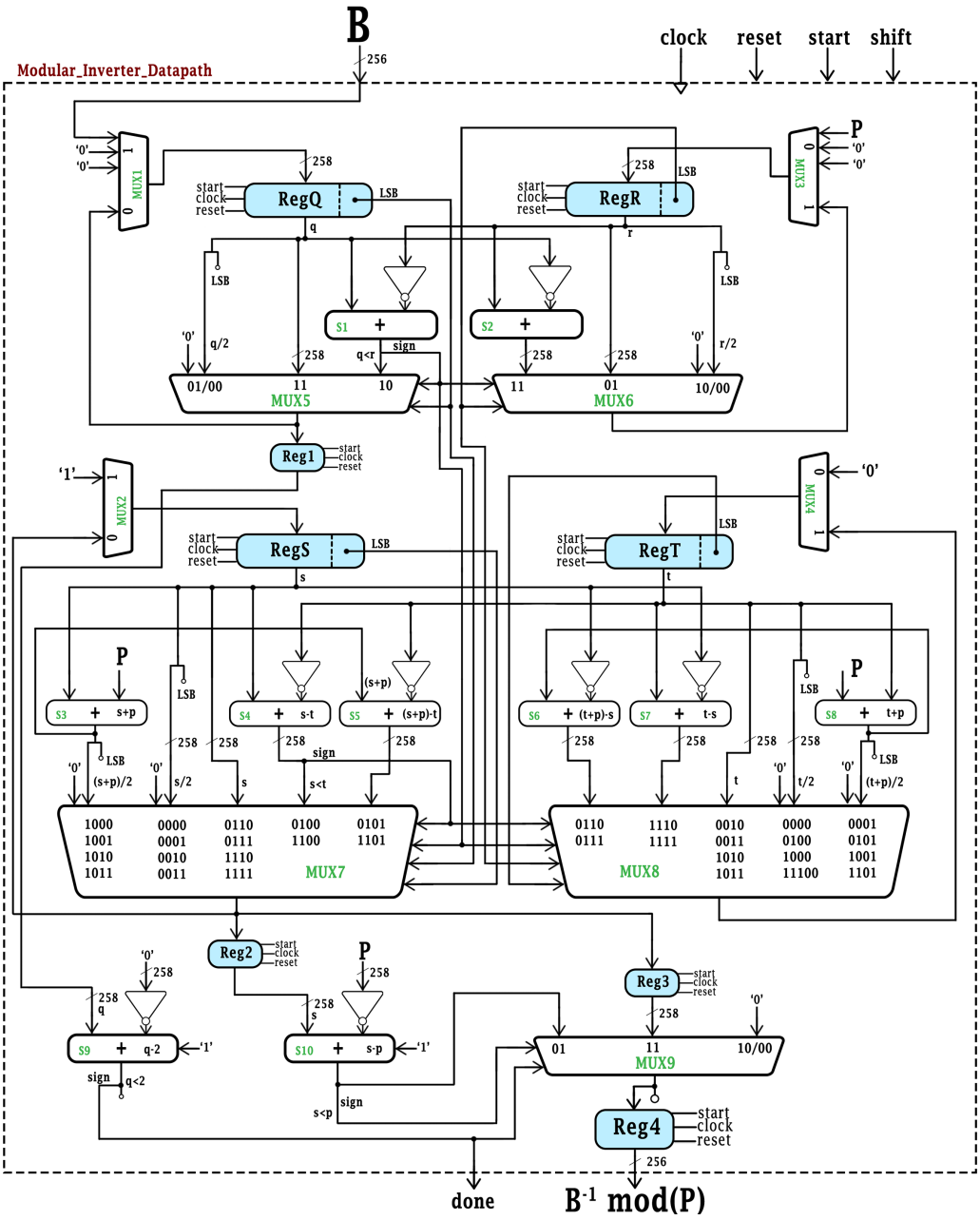


Figure 3.27. Datapath modular inverter

How in presented in the block abstraction the architecture is drive by a FSM, this is necessary because i have split the circuit in two part using pipeline register $reg1$, $reg2$, $reg3$ to reduce the critical path, now the two half part work simultaneously and this allow to have the same latency as the original circuit but with a Cp less.
In the 3.28 is reported the flow chart that describe logic sequence and for each state define the control signal send to the datapath.

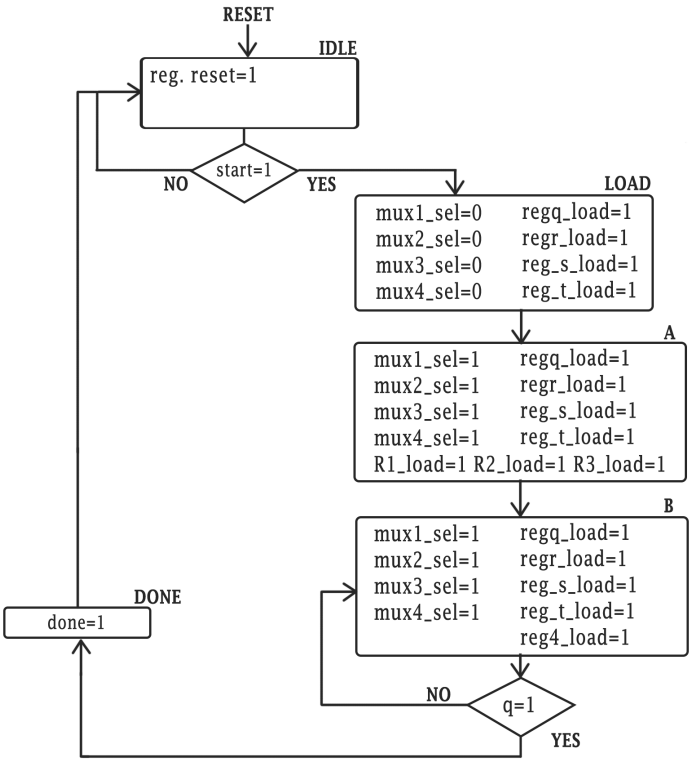


Figure 3.28. FSM Modular Inverter

features:
Cost: 10 Adder + 9 MUX + 7 Inv. + 8 Reg .
Critical Path: 2 T_{add} + T_{MUX} + T_{reg}
Latency : $n + \frac{n}{4}$ T_{ck} = Max. 320 T_{ck}

The parallelism is defined on 258 bit to preserve the integrity of the signal, one extra bit is for the overflow, and it is sufficient to represent at maximum three time the field dimension $\mathbb{F}p$, other one extra bit is for the sign.

3.4.2 timing

In fig. 3.25 is reported the timing diagram of Modular Inverter. How is possible to see there is a timing pattern described in the FSM that is repeated many time instead of the output of $S9$ give a result that is negative, at this point $q < 2$ means q=1 considering the positive nature of the partial results. At maximum $n + \frac{n}{4}$ clock cycle are necessary for the inversion.

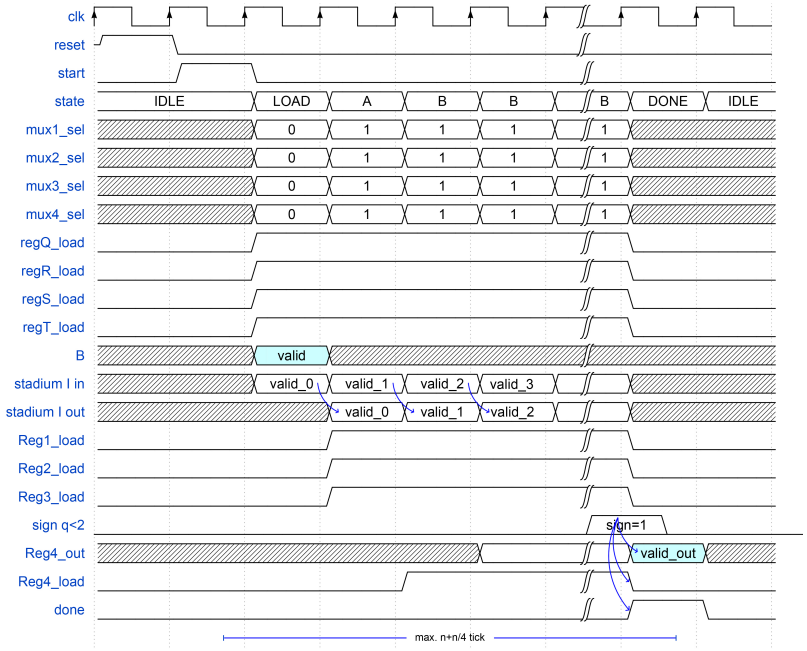


Figure 3.29. timing Modular Inverter

3.4.3 VHDL and Software validation

```
1
2  ############################
3  # python mod_inversion.py
4  ############################
5
6  from sympy import mod_inverse
7
8  number = 32574380800717110134092858959791422721477056992005136709695267593247745763631
9  modulus = 57896044618658097711785492504343953926634992332820282019728792003956564819949
10
11 inverse = mod_inverse(number, modulus)
12 print("Inversione Modulare:", inverse)
13
14 #------------
15 # out = 3B096388255DB31880D26F49D83D9129118727536B2C8ADFAB2CB245C2BC67E4
```

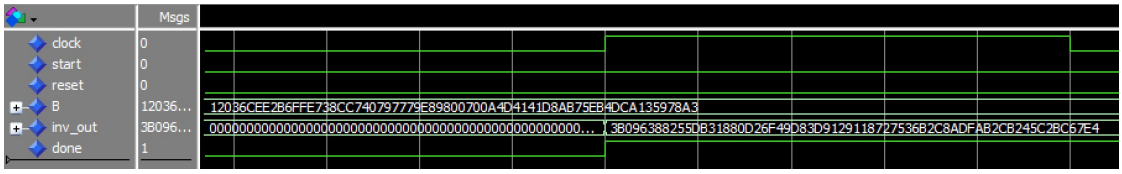
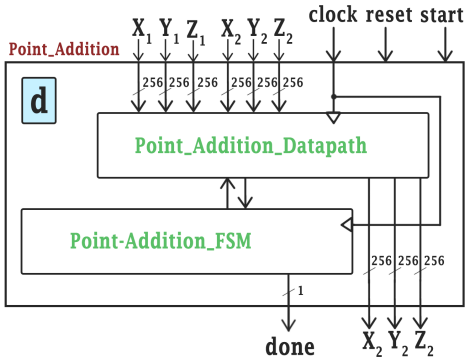


Figure 3.30. VHDL Modular inverter simulation

How is possible to see the result are the same, also in this case the architecture work well, this time the software simulator is a Python code that use specific library to compute the inversion.

3.5 Point Addition



Starting from this section we initiate assembling the more relevant block that are essential for computation of point multiplication, to do this are used all the modular block defined in the past paragraph. Also in this case the algorithm for the Point Addition is a derivation of a very complex math problem, but we develop our research around a set of formula that are just given and are just reported in eq. 2.12 if we consider affine coordinate, but we work in projective to be more efficient therefore the reference are 3.3.

3.5.1 Point Addition Architecture

Starting point is define the scheduling of the resources to implement the equation in the more efficient way reducing as much as possible the hardware allocated and the time necessary, in fig. 3.31 is reported the scheduling flow-chart.

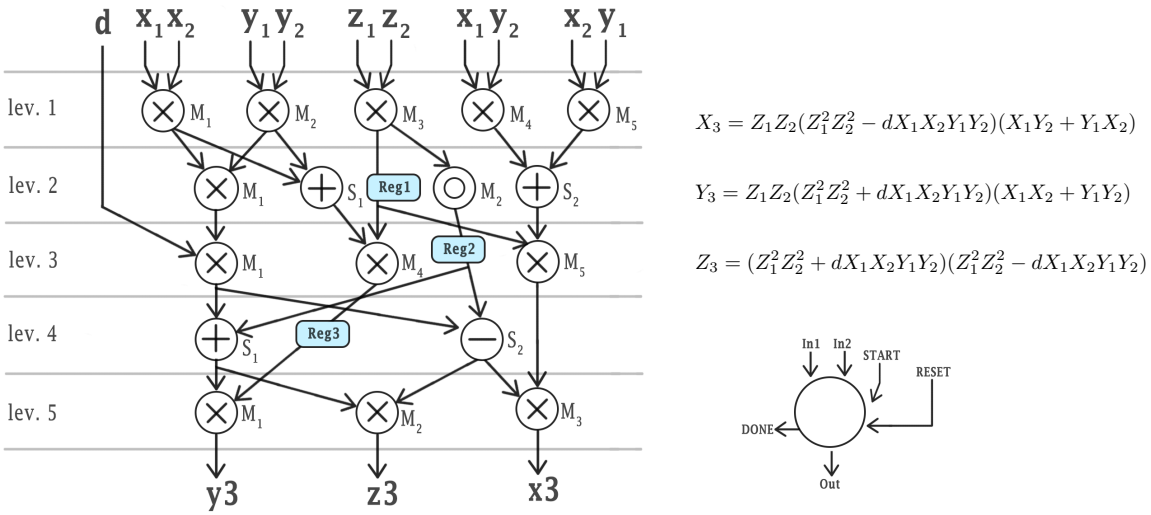


Figure 3.31. resource scheduling Point Addition

the block take in input two different value that will added following the Elliptic Curve Ed25519, each point is defined on three dimension $A(x, y, z)$ and $B(x, y, z)$ then there is a third input that is the constant d defined in eq.2.26.
Each graphic circle is a modular operation respectively $\oplus$ is the modular additio, $\otimes$ is the modular multiplication, by the way i have implemented both case RADIX-2 multiplier and RADIX-4 multiplier

although we just say that RADIX-4 is the best one, then $\ominus$ is the modular subtraction and $\circledcirc$ is the modular square root.

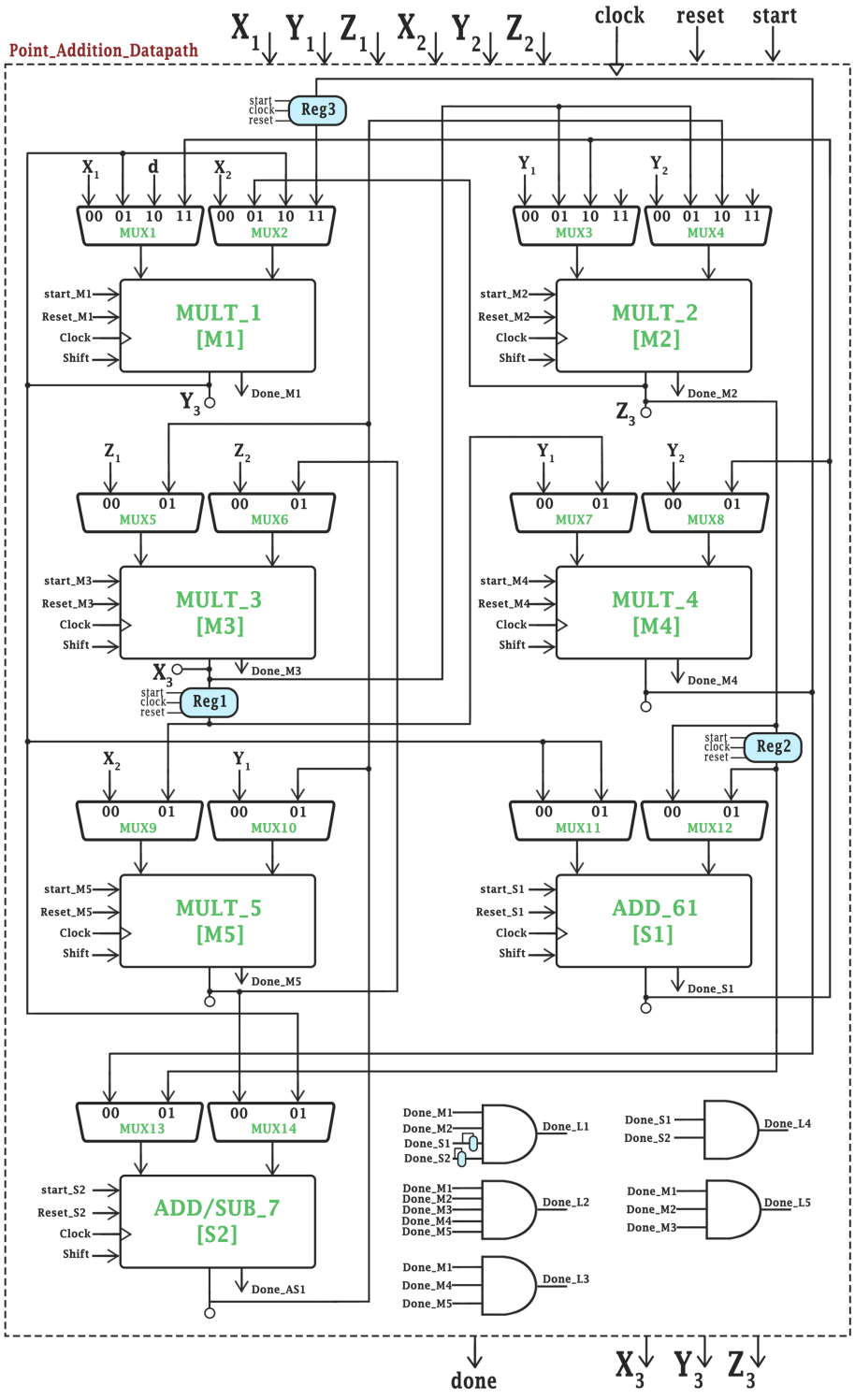


Figure 3.32. architecture Point Addition

One of the Target of this research is reduce as much as possible the area, to do this i try to schedule the block in the best possible way reusing all the resource many time, near each block is possible see a label that i will associate to the allocated block in the *point_addition_Datapath* fig. 3.32 block that have same label means that share the same component, obviously the sharing is done in such a way that the level of computation are always five and the execution time is the less as possible.
As is possible to see the resource allocated are strongly reduced respect the scheduling diagram, but we see better:

features:
Scheduling Resource: 4 Mod. Adder + 12 Mod. Multiplier + 1 Mod. squarer.
Datapath Resource: 1 Mod. Adder + 1 Mod. Adder/Subtractor + 5 Mod. Multiplier

Datapath Cost: Datapath Resource + 14 MUX + 5 AND + 3 Reg.
Critical Path: Critical Path Mod. Multiplier
Latency using RADIX-2 : $(n+1) \cdot 4\ T_{ck} + 2\ T_{ck} + 1\ T_{ck}\ (L1\_A) = 1031 T_{ck}$ (1030 in cycling working)
Latency using RADIX-4 : $(\frac{n}{2} + 2) \cdot 4\ T_{ck} + 2\ T_{ck} + 1\ T_{ck}\ (L1\_A) = 523 T_{ck}$ (522 in cycling working)

Is clear that using scheduling we have saved seven multiplier and two adder, now it is more clear the reason why i have decide to implement the block of addition/subtraction, because instead of having only two adder we would have also another subtractor, that occupies area. About the working of datapath there are two multiplexer for each arithmetic block, according to the level in which we are the FSM drive the respective selector to let pass the correct input, to define the end of a level five AND gate are allocated and their input are asserted to '1' only in the moment in which all the block on the level are finish and his done is high, at this point the FSM enable to start the next level. About the three register they are the same visible in the scheduling flow chart and are used to memorize data that require pass trough more than one level. In the following fig. 3.34 and fig. 3.35 are reported the FSM of respectively both RADIX-2 implementation and RADIX-4 implementation, and in 3.33 the starting non optimized version.

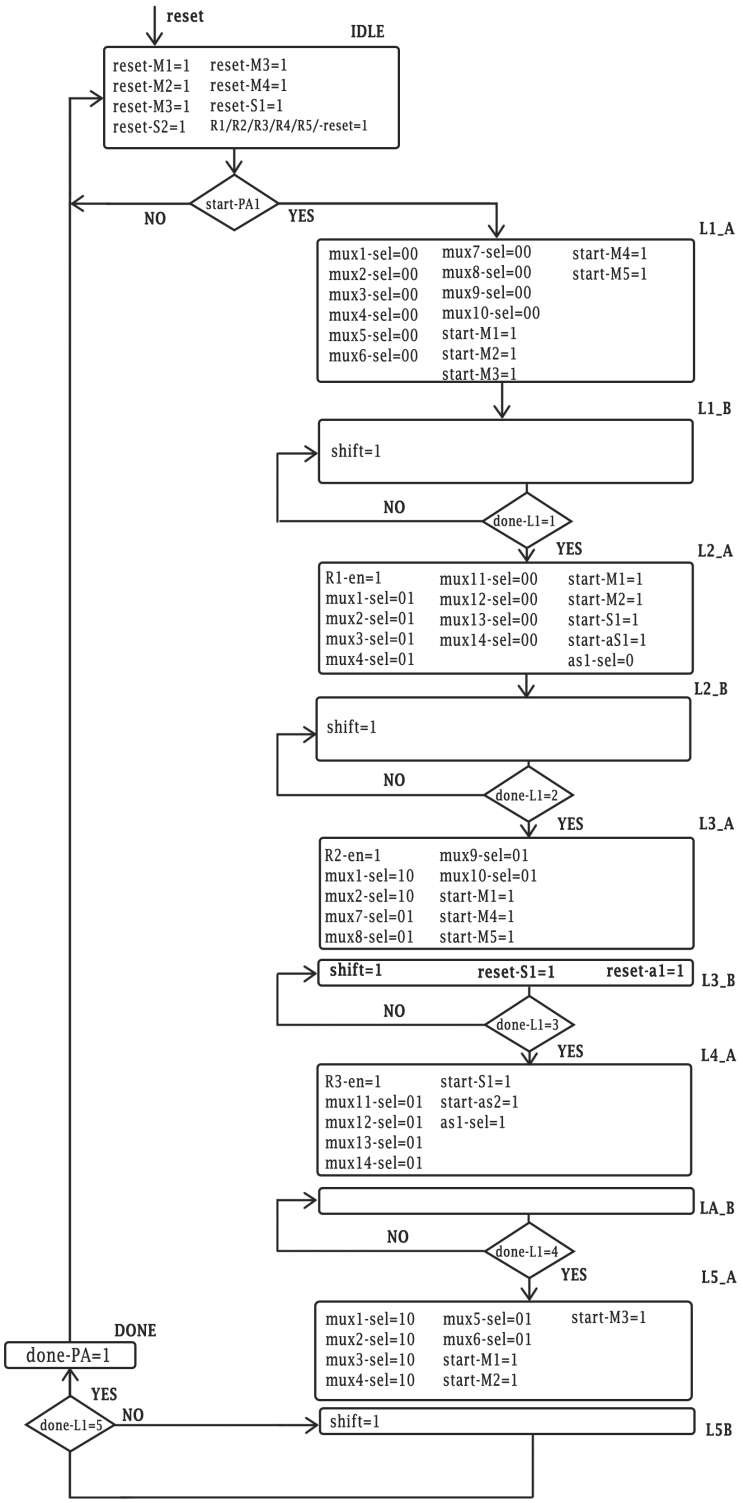


Figure 3.33. FSM Point Addition RADIX-2 non-optimized

In this implementation the switch between two state is defined by the assertion of the done signal, and in case is zero we continue to loop instead of the calculation are finished.

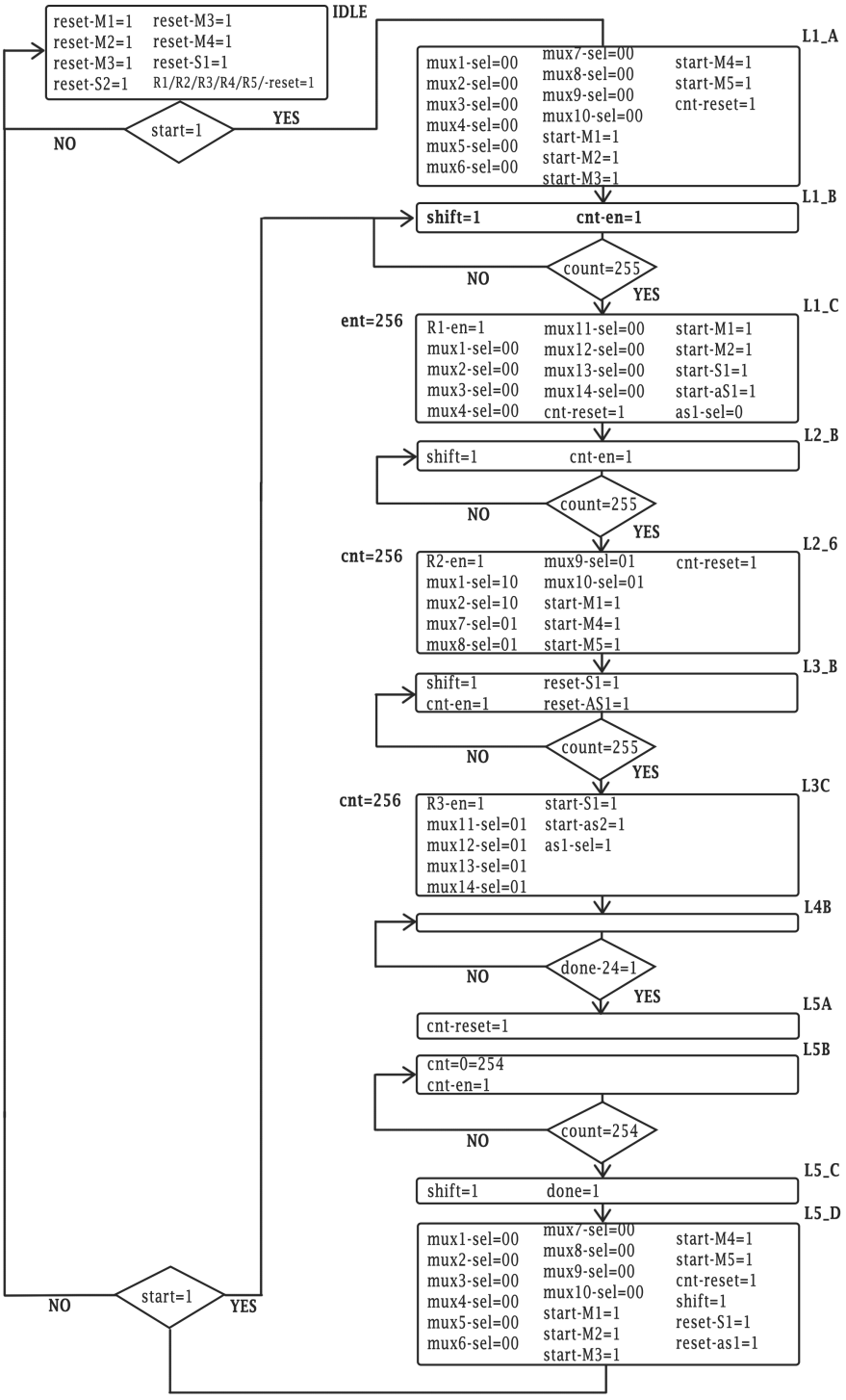


Figure 3.34. FSM Point Addition RADIX-2

Now differently from before we do not have the done signal to define the end of the level, but we use a counter, this allow to start the following level across the last one without loosing clock cycle. Then there is the possibility to work in cycling way passing in the fork.

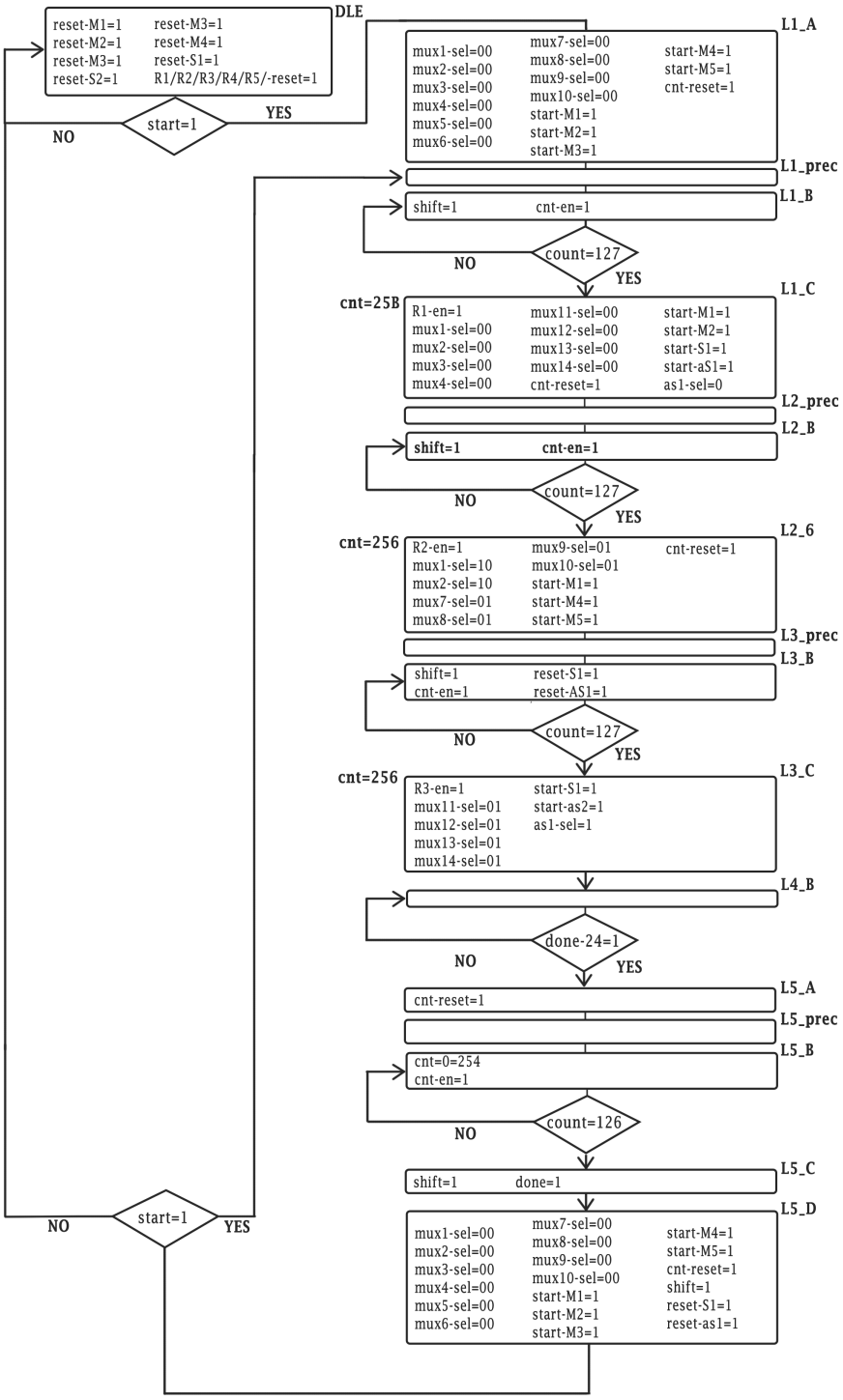


Figure 3.35. FSM Point Addition RADIX-4

How is possible to see are very similar, RADIX-4 differ fundamentally from having more state and this is due to the fact that there is the precomputation stadium that occupy one clock cycle, and then the counting discriminant are different because require half iteration respect RADIX-2. globally for both the generic idea is enable each level one at time and when the done signal of the level is asserted we can start the next one. Notice that that the FSM is projected taking in consideration the fact that

in the point multiplication the point addition is the bottle neck because the point doubling require less number of iteration, moreover the PA iterate many and many time consequentially, therefore is necessary optimize the FSM to allow the best performance as possible, and this is done making sure that when the $Level.5$ is completed in the next clock cycle the $Level.1$ is just in execution, to do that is necessary anticipate the done signal in state $L5\_A$ of one clock cycle as we will see in the timing and define a fork to verify if the start is asserted and exceed directly to the $L1\_B$ state without wasting time. Proof of that is the latency reported in the feature chapter,first implementation require 525 iteration, then optimized version require 522 T_{ck} for one computation and one less for the cyclic, means 521 T_{ck}.

3.5.2 timing

In fig.3.36 3.37 and 3.38 are reported the timing diagram in which are distinguishable the five level denominate with the letter L and the number of the level, then each one is subdivided in step defined before in the FSM. In the first implementation 3.36 that is non optimized for cyclic working is possible to see that each iteration end when the done of the level is asserted, then in next state the start become high and the next iteration start, this is correct but the problem is that every time we loose one clock cycle for the restart and other two for the passage for the $IDLE$ state. Is also necessary optimize this architecture to reduce as much as possible the number of iteration primarily to the point of view of the cyclic working as in 3.37 and 3.38 , to do this first improvement is define a counter because is necessary anticipate the assertion of the done, but using also the done signal of each level is impossible due to the fact that we are not able to predict the future, second improvement is define a fork of the flow in which if the start is high the flow is redirect to state $L1\_B$ saving other two clock cycle, otherwise it return to the $IDLE$ state. In conclusion with this betterment's the latency is reduced of three clock cycle and as we will see being the PA the bottle neck of the ECPM in total we have saved $3 \cdot 255$ iteration $= 765$ clock cycle in point multiplication operation.

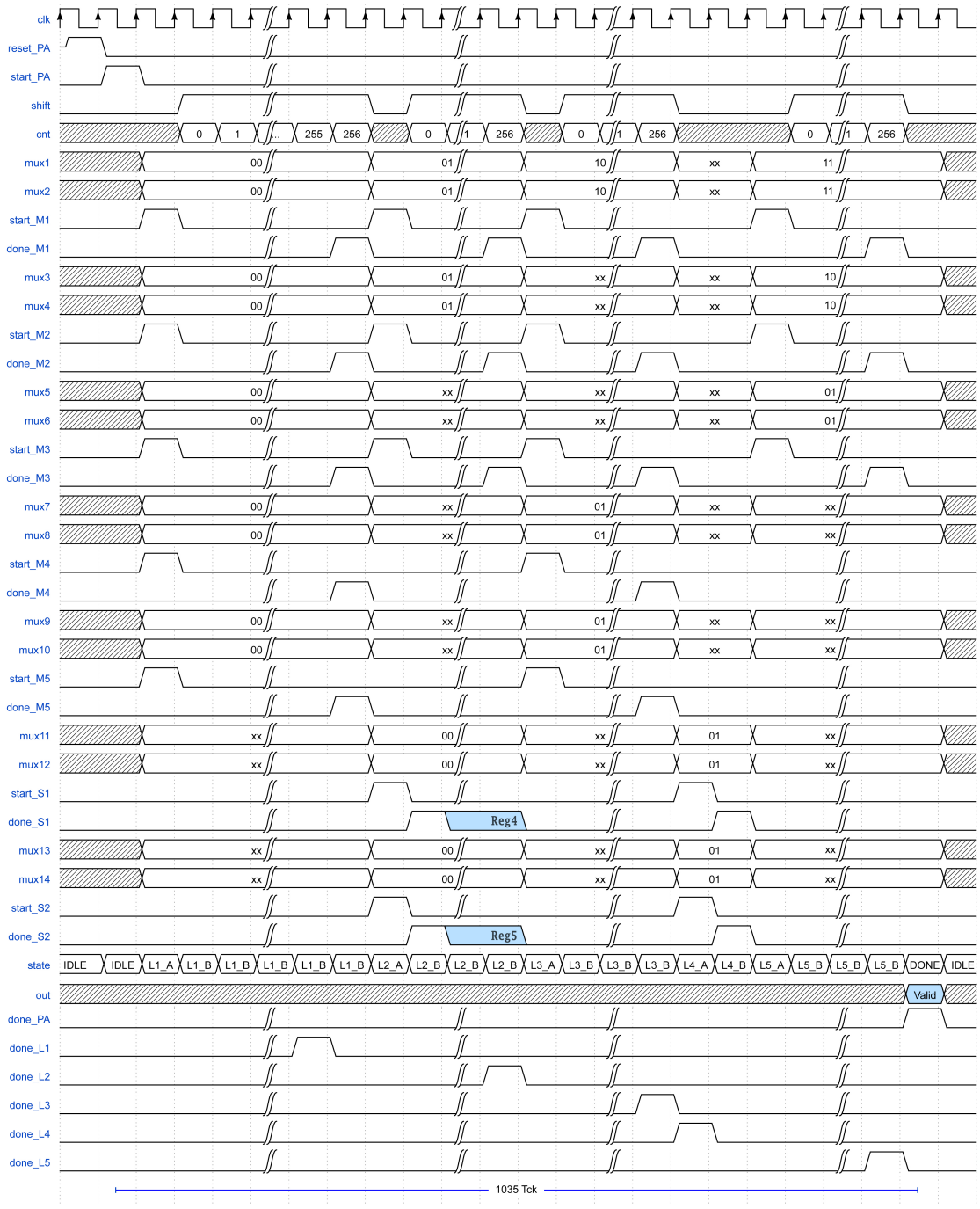


Figure 3.36. timing Point Addition RADIX-2 non-optimized

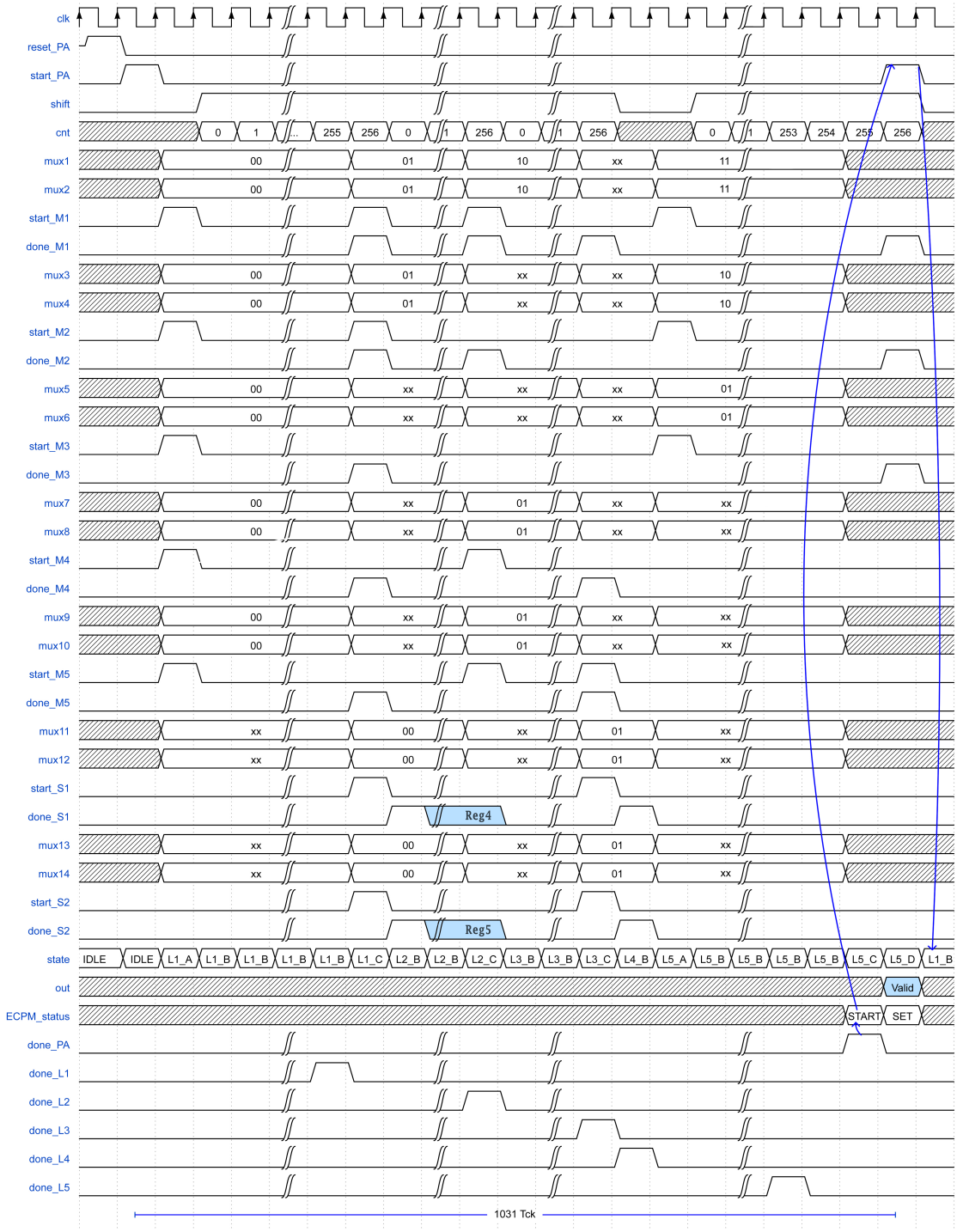


Figure 3.37. timing Point Addition RADIX-2

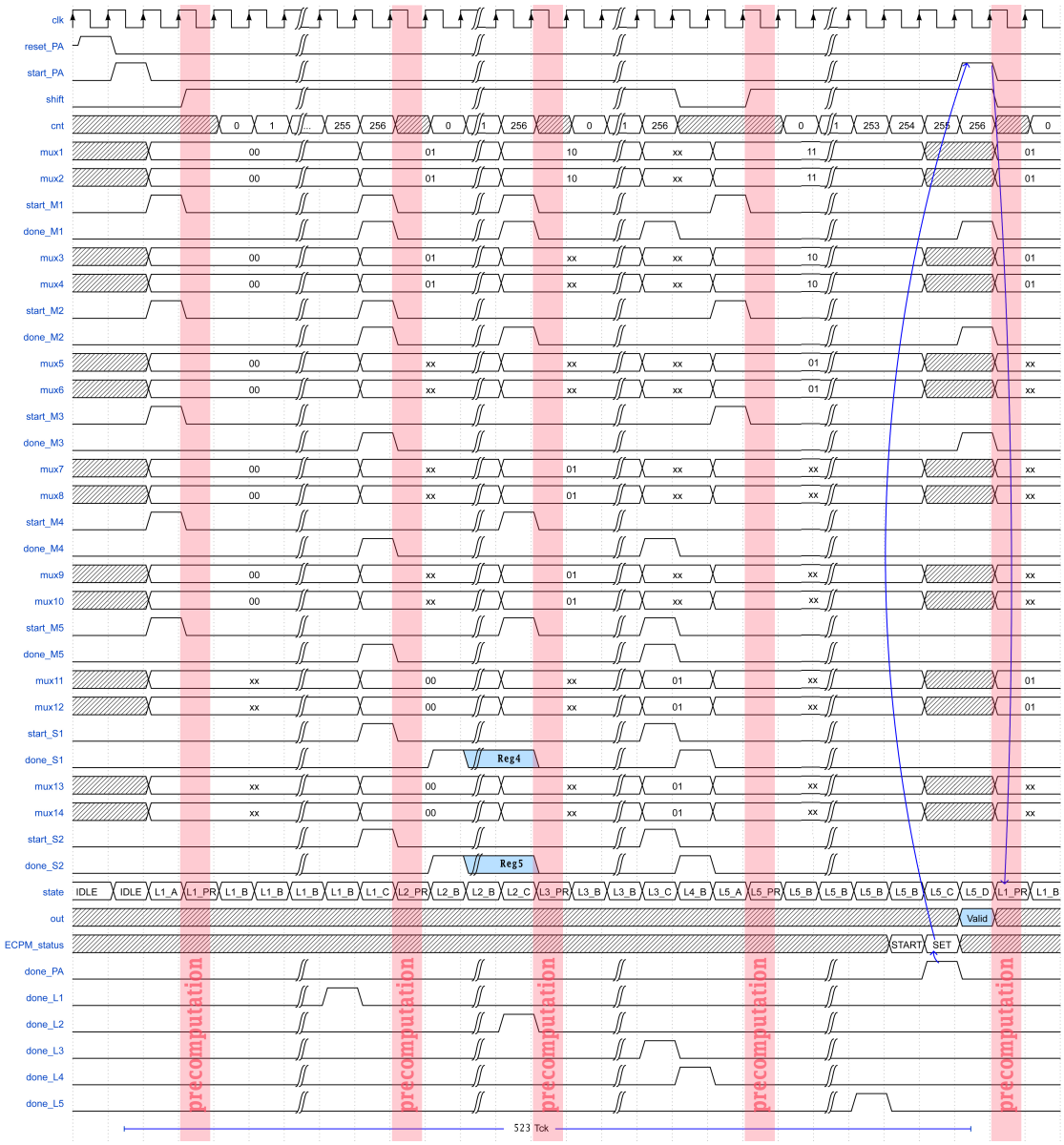


Figure 3.38. timing Point Addition RADIX-4

How is possible to see in reed are highlighted the precomputation stadium, in both case i report the case of cyclic working stress the attention on the done signal that is anticipate respect the valid output of one clock cycle to allow the restarting without loosing time, i also report the state of the ECPM FSM to have a more clear overview of the time scheduling.

3.5.3 VHDL simulation and Matlab validation

In the following are reported the Matlab validation code and VHDL simulation using Modelsim

```
% field dimension 0x52036cee2b6ffe738cc740797779e89800700a4d4141d8ab75eb4dca135978a3
% constant d 0x52036cee2b6ffe738cc740797779e89800700a4d4141d8ab75eb4dca135978a3
p = sym('57896044618658097711785492504343953926634992332820282019728792003956564819949');
d = sym('37095705934669439343138083508754565189542113879843219016388785533085940283555');
%input
x1 = sym('151122213495354007725011514095885315114540126930418572060461132839498477622 02');
y1 = sym('463168356949264781694283940034751631413079938662562256157830336031652518559 60');
z1 = sym('1');
x2 = sym('222271421460536153836867114565920545334817230652383280794910861657546885719 91');
y2 = sym('231326128979357639473761188163029369619457538555924972125273302060347140013 67');
z2 = sym('477309695254115434863233454915946082009688224541037287466376761790956438073 39');
% level 1
m11 = mod((x1*x2),p);
m21 = mod((y1*y2),p);
m31 = mod((z1*z2),p);
m41 = mod((x1*y2),p);
m51 = mod((x2*y1),p);
% level 2
m12 = mod((m11*m21),p);
s12 = mod((m11+m21),p);
m22 = mod((m31*m31),p);
s22 = mod((m41+m51),p);
% level 3
m13 = mod((d*m12),p);
m43 = mod((s12*m31),p);
m53 = mod((m31*s22),p);
% level 4
s14 = mod((m13+m22),p);
s24 = mod((m22-m13),p);
% level 5
y3 = mod((s14*m43),p);
dec2hex(y3)
z3 = mod((s14*s24),p);
dec2hex(z3)
x3 = mod((s24*m53),p);
dec2hex(x3)

%------------
% x3 = 6B8E2FBC776B7E13CE4AA95335C6B1F390A06B8F524BAF4EC23753B1C9EB9985
% y3 = 169A317C0CBFFF7865588CE7671602170B9DF112A9F752EEDF75F68803EC7664
% z3 = 61E9BE98A68F5565E79965B0C47B14857CBD699B053F91018B5F4B631D02D8E9
```

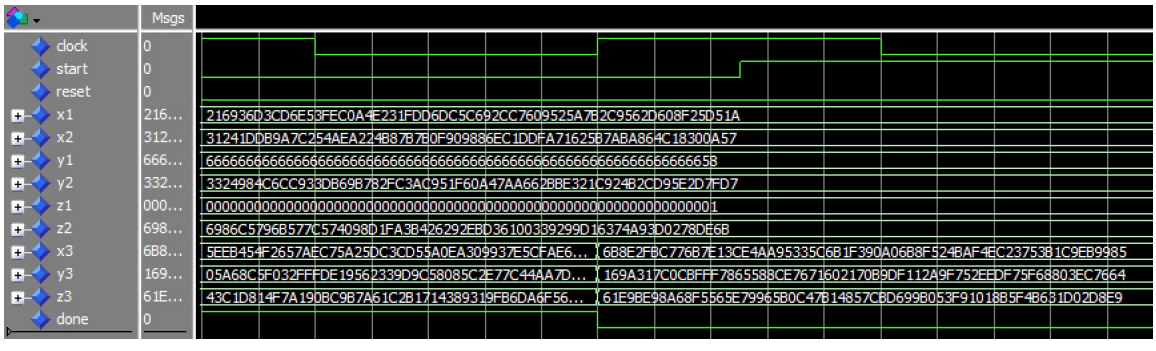
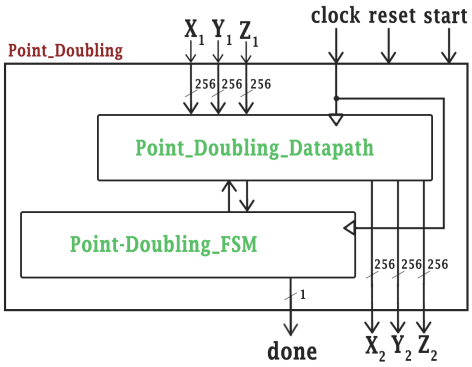


Figure 3.39. VHDL simulation point addition RADIX-4

After simulation the result are the same, circuit work well.

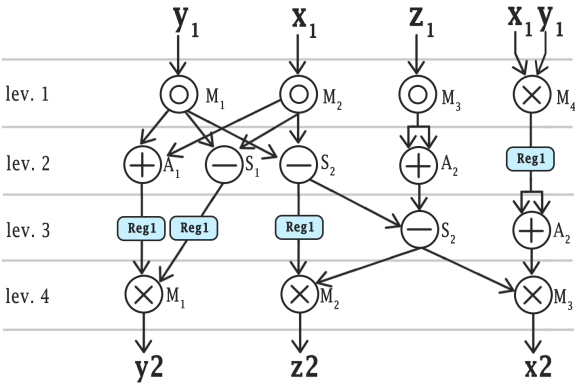
3.6 Point Doubling



In this second section is treated the Point Doubling operation, the implementation follow everything described for point Addition, unique difference is in the more easiest equation implemented.

3.6.1 Point Doubling Architecture

Starting point is define the scheduling of the resources to implement the equation in the more efficient way reducing as much as possible the hardware allocated and the time necessary, in fig. 3.31 is reported the scheduling flow-chart.



$$X_2 = (2X_1Y_1)(Y_1^2 - X_1^2 - 2Z_1^2)$$

$$Y_2 = (X_1^2 - Y_1^2)(X_1^2 + Y_1^2)$$

$$Z_2 = (Y_1^2 - X_1^2)(Y_1^2 - X_1^2 - 2Z_1^2)$$

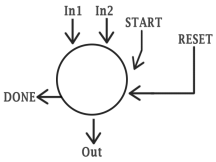


Figure 3.40. resource scheduling Point Doubling

the block take in input only one coordinate that will added to herself following the Elliptic Curve Ed25519, the point is defined on three dimension $A(x, y, z)$, in this case there is not the constant d because this equation are the result of point Addition equation simplification
Each graphic circle is a modular operation respectively $\oplus$ is the modular additio, $\otimes$ is the modular multiplication, by the way i have implemented both case RADIX-2 multiplier and RADIX-4 multiplier although we just say that RADIX-4 is the best one, then $\ominus$ is the modular subtraction and $\circledcirc$ is the modular square root.
One of the Target of this research is reduce as much as possible the area, to do this i try to schedule as in the case of point Addition the block in the best possible way reusing all the resource many time, near each block is possible see a label that i will associate to the allocated block in the *point_Doubling_Datapath* fig. 3.41 block that have same label means that share the same component, obviously the sharing is done in such a way that the level of computation are always 4 and the execution time is the less as possible. Notice that in this implementation the level are only four and two are just addition operation means 2 T_{ck}.

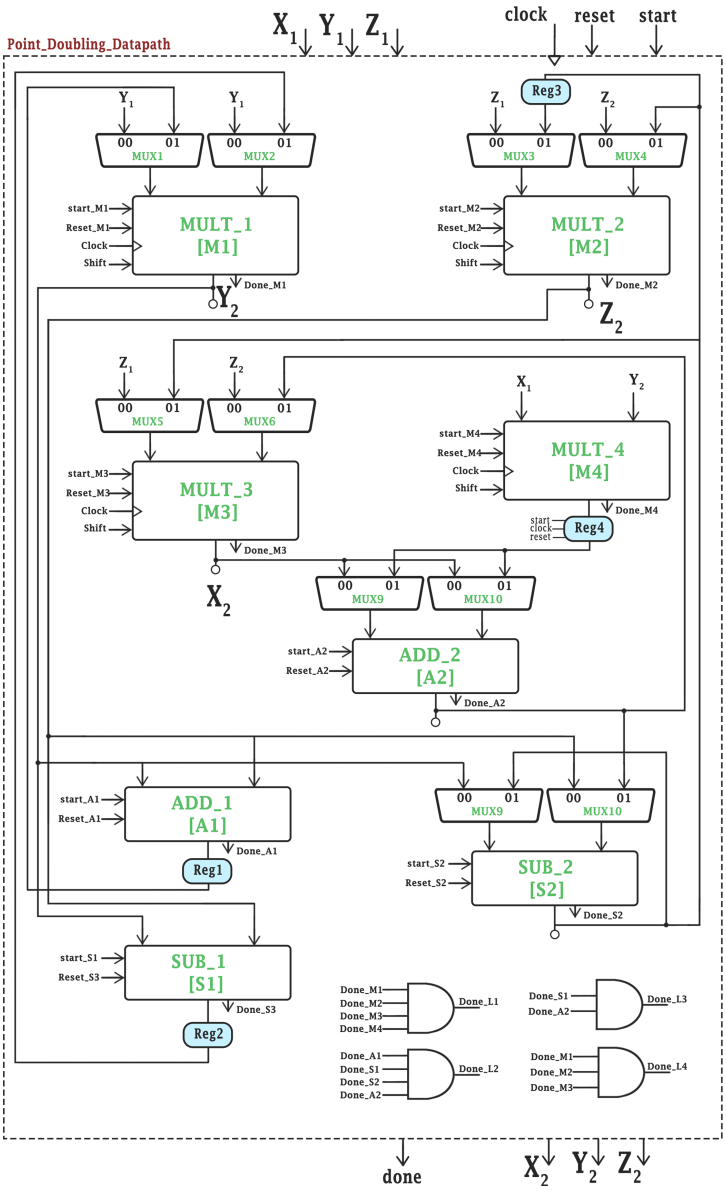


Figure 3.41. Datapath Point Doubling

features:
Scheduling Resource: 3 Mod. Adder + 3 Mod. Sub. + 4 Mod. Multiplier + 1 Mod. squarer.
Datapath Resource: 2 Mod. Adder + 2 Mod. Subtractor + 4 Mod. Multiplier

Datapath Cost: Datapath Resource + 10 MUX + 4 AND + 4 Reg.
Critical Path: Critical Path Mod. Multiplier
Latency using RADIX-2 : $(n+1)\cdot 2\ T_{ck} + 2\ T_{ck} + 2\ T_{ck}\ (L1\_A/\text{done}) = 520T_{ck}$
Latency using RADIX-4 : $(\frac{n}{2}+2)\cdot 2\ T_{ck} + 2\ T_{ck} + 2\ T_{ck}\ (L1\_A/\text{done}) = 264T_{ck}$

As we have presented for the point Addition, i implement the architecture also using both RADIX-2 and RADIX-4 multiplier to have a comparison although we just say that RADIX-4 is the best one. In paragraph *feature* are reporter the data about the resources necessary ad the number of iteration,

using RADIX-2 are needed 520 T_{ck} instead using RADIX-4 264 T_{ck} that are more or less the half. About cyclic optimization is not necessary because in the ECPM as we will see Point Doubling and Point Addition work in parallel, but given that PA require 520 T_{ck} and PD 264 T_{ck} it would just be a waste of resource considering that is necessary allocate a counter and define more logic for the extra status in the FSM.

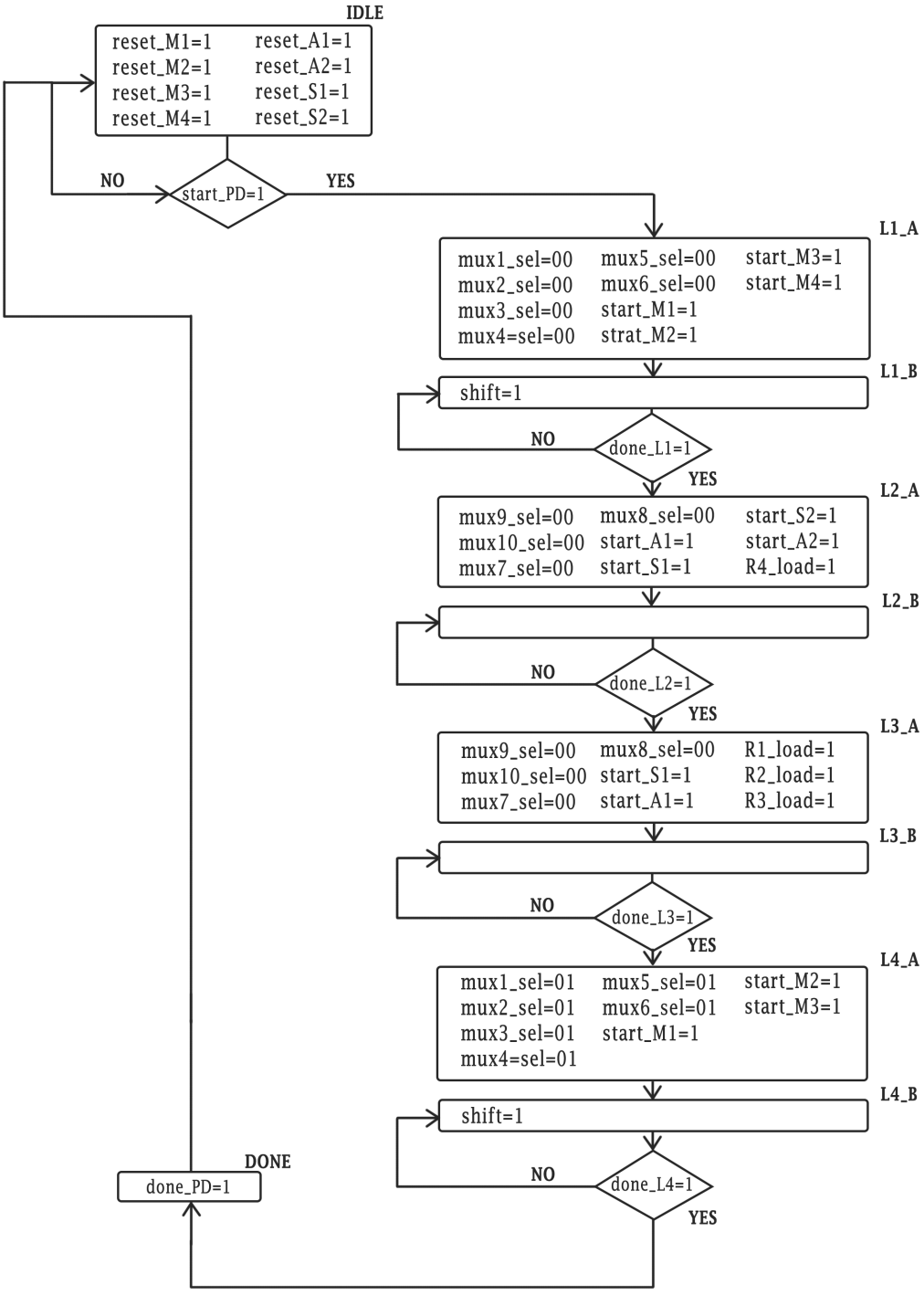


Figure 3.42. FSM Point Doubling RADIX-2

How for the case of Point Addition Datapath require the support of the FSM to work properly. Also in this case FSM manage the state transition according to the four level f the flow chart and enable the switch between two state only in the moment in which the done signal of the level is asserted, in general each state of the machine is defined according to the signal scheduling defined in the timing diagram proposed in next section.
3.42 and 3.43 report respectively the FSM implemented using RADIX-2 multiplier and RADIX-4 that need precomputation stadium.

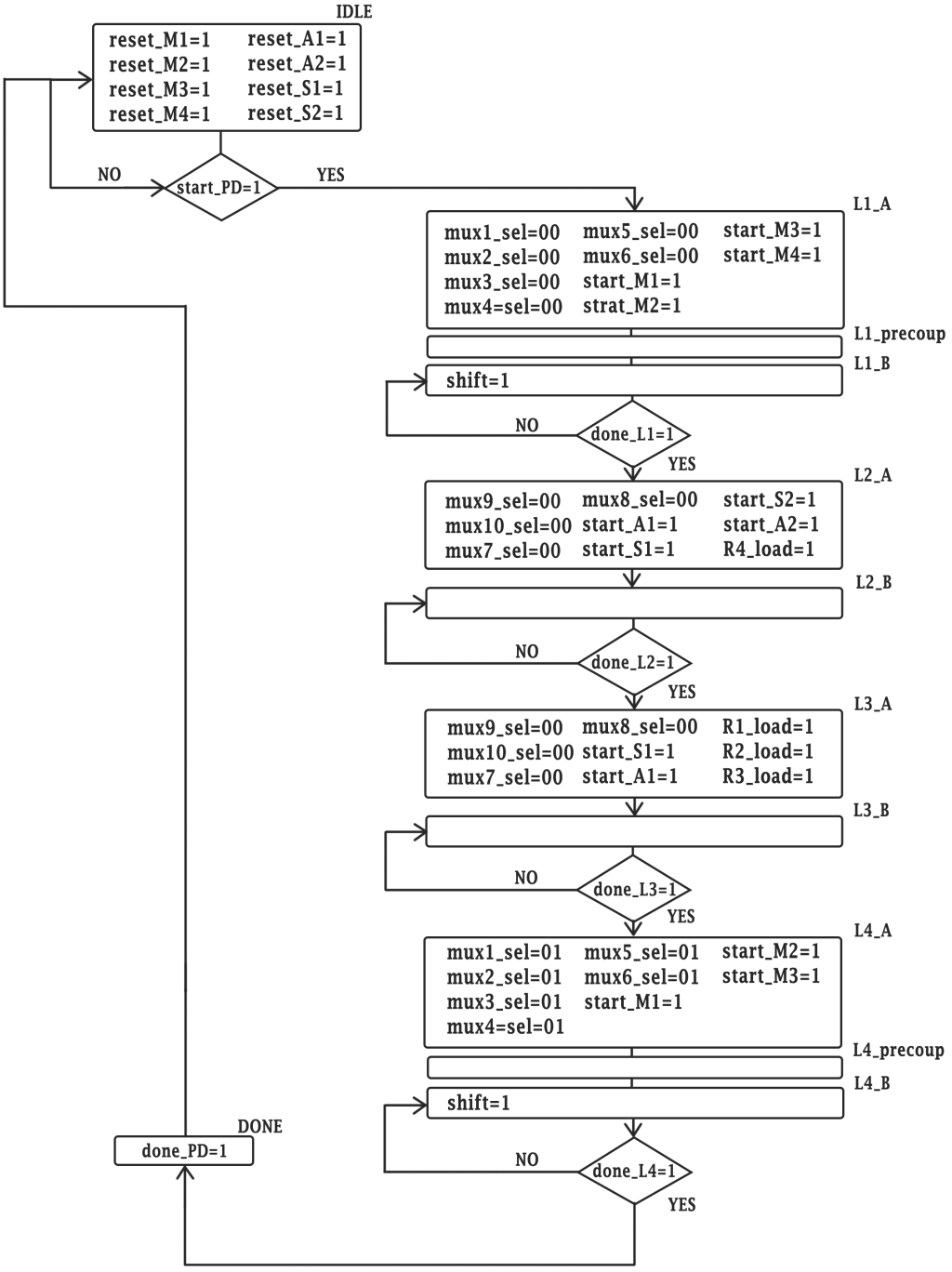


Figure 3.43. FSM Point Doubling RADIX-4

3.6.2 timing

I have reported two timing diagram the first using RADIX-2 multiplier and second one using RADIX-4 multiplier and precomputation cycle are highlighted in red.

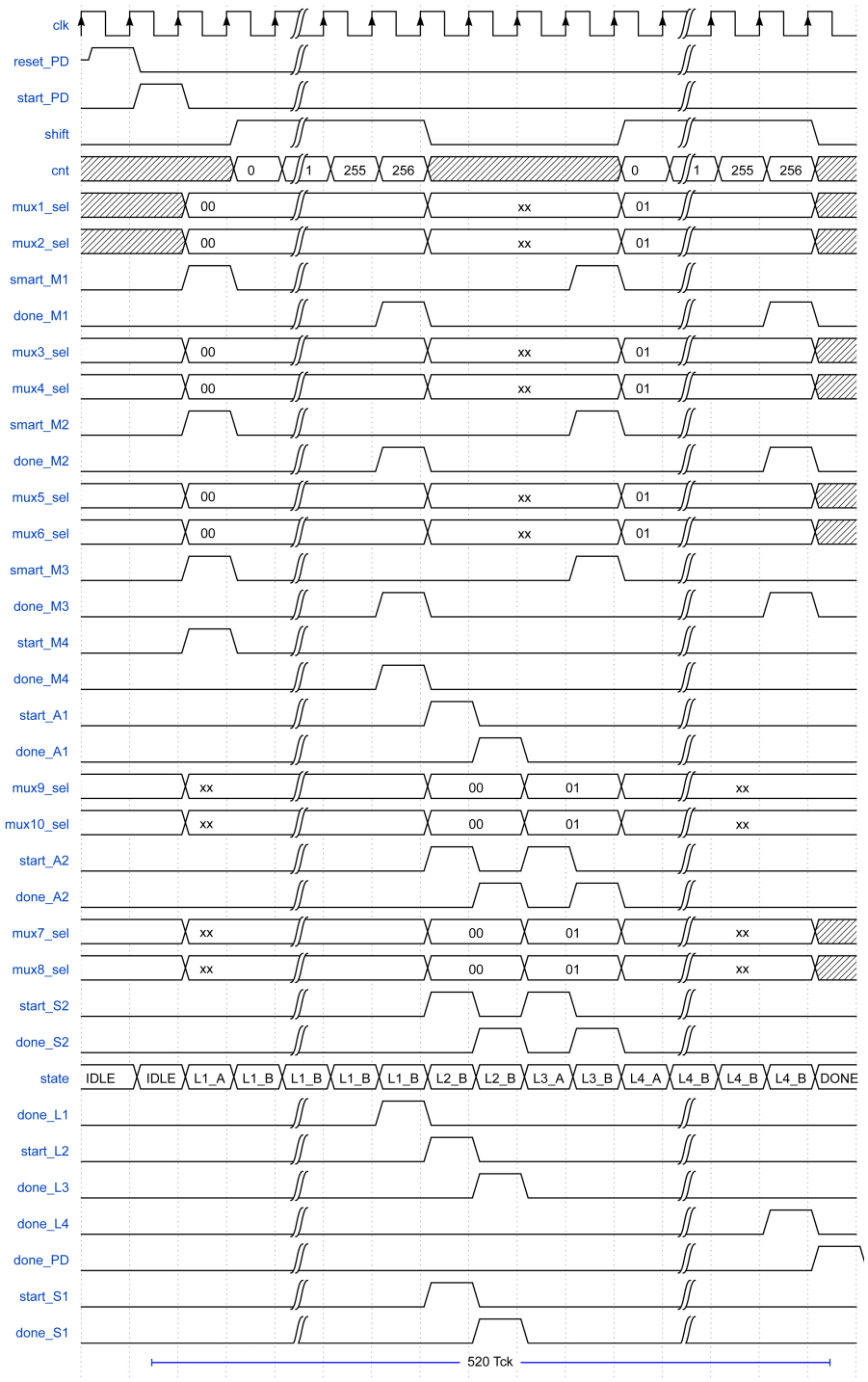


Figure 3.44. timing Point Doubling RADIX-2

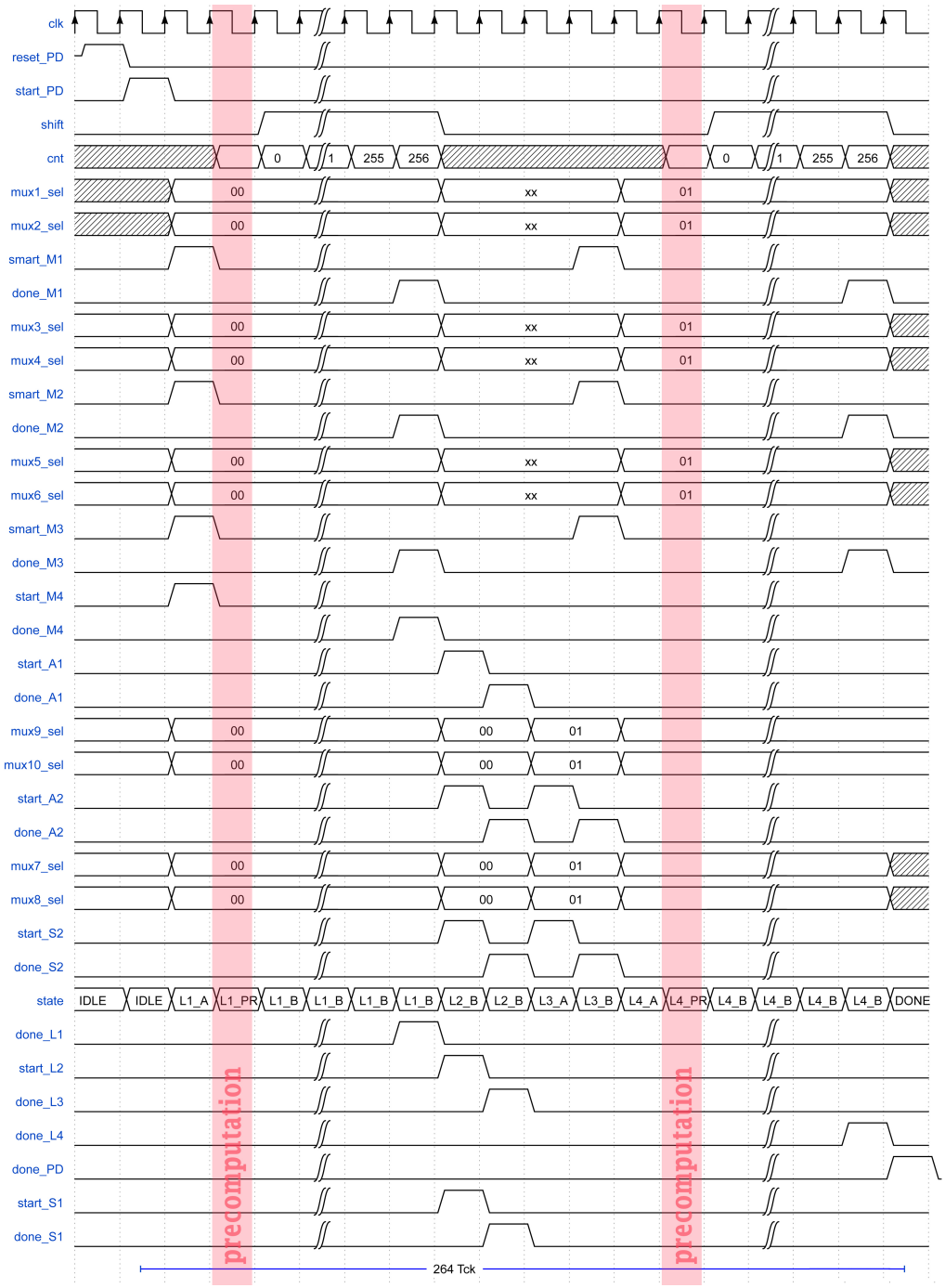


Figure 3.45. timing Point Doubling RADIX-4

3.6.3 VHDL simulation and Matlab validation

In the following are reported the Matlab validation code and VHDL simulaton using Modelsim

```
% field dimension 0x52036cee2b6ffe738cc740797779e89800700a4d4141d8ab75eb4dca135978a3
% constant d 0x52036cee2b6ffe738cc740797779e89800700a4d4141d8ab75eb4dca135978a3
p = sym('57896044618658097711785492504343953926634992332820282019728792003956564819949');
d = sym('37095705934669439343138083508754565189542113879843219016388785533085940283555');
%input
x1 = sym('151122213495354007725011514095885315114540126930418572060461132839498477622 02');
y1 = sym('46316835694926478169428394003475163141307993866256225615783033603165251855960');
z1 = sym('1');
% level 1
m11 = mod((y1*y1),p);
m21 = mod((x1*x1),p);
m31 = mod((z1*z1),p);
m41 = mod((x1*y1),p);
% level 2
a12 = mod((m11+m21),p);
s12 = mod((m21-m11),p);
s22 = mod((m11-m21),p);
a22 = mod((m31+m31),p);
% level 3
s23 = mod((s22-a22),p);
a23 = mod((m41+m41),p);
% level 4
m14 = mod((a12*s12),p);
y2 = dec2hex(m14)
m24 = mod((s22*s23),p);
z2 = dec2hex(m24)
m34 = mod((s23*a23),p);
x2 = dec2hex(m34)

%------------
% x3 = 31241DDB9A7C254AEA224B87B7B0F909886EC1DDFA71625B7ABA864C18300A57
% y3 = 3324984C6CC933DB69B782FC3AC951F60A47AA662BBE321C924B2CD95E2D7FD7
% z3 = 6986C5796B577C574098D1FA3B426292EBD36100339299D16374A93D0278DE6B
```

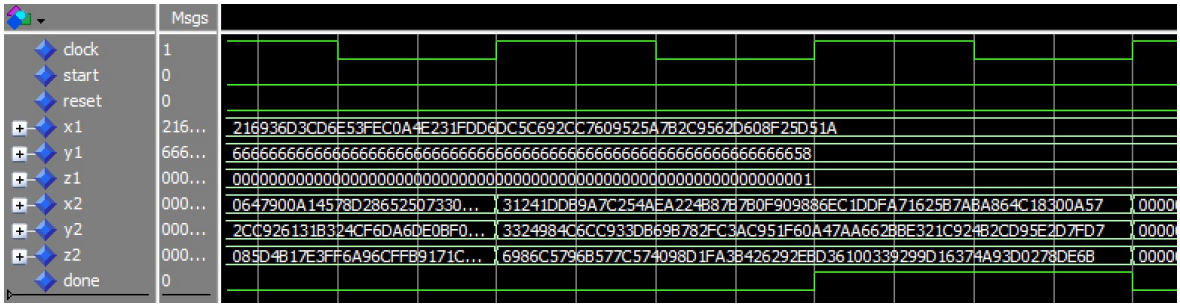
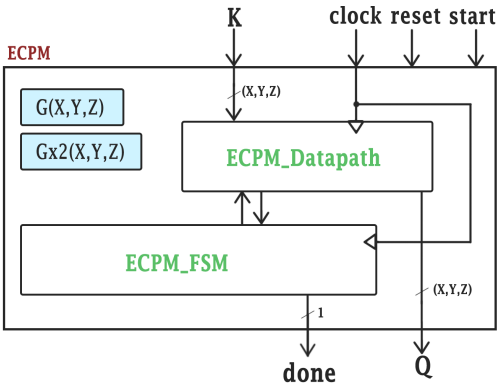


Figure 3.46. VHDL simulation point Doubling RADIX-4

Result are the same, circuit work well

3.7 Elliptic Curve Point Multiplication(ECPM)



In this last section we are arrived to assemble the Elliptic Curve Point Multiplication ECPM using Point Addition and Point Doubling defined before, what we do finally is define the value of the public-key Q starting from a generative point G and multiply it for a Private-key K on the Ed25519 curve, as we have explained more in detail in the introductory chapter.

$$Q = K \cdot P$$

Basic idea of this algorithm is adding P to itself $K - 1$ time:

$$Q = G + \underbrace{G + ... + G}_{k-1 \text{ times}} \tag{3.3}$$

or equally doubling P $\log_2 K$ time if K is even:

$$Q = \underbrace{...2(2(2(G)))}_{\log_2 K \text{ times}} \tag{3.4}$$

There are many algorithm that perform this operation in different way using the Point Doubling and Point Addition following the bit pattern of K. ECPM can be performed using the double-and-add method, as demonstrated in Algorithm 4. In this method, point doubling is executed in each iteration, and point addition occurs only when $K_i = 1$. This approach exhibits two different timing and power consumption profiles: one involving only point doubling and the other involving point addition following point doubling. However, analyzing the power consumption profiles through Simple Power Analysis (SPA) 3.47 reveals that the binary bit pattern of the secret key can be easily extracted, consequently this method is susceptible to Side-Channel Attacks (SCAs).

Algorithm 4 Double-and-Add Point Multiplication (Left to Right)

Require: $P, k = (\sum_{i=0}^{l-1} k_i 2^i)_{10}; \quad k_i \in \{0,1\}, \quad k_{l-1} = 1$
Ensure: Q
1: $Q \leftarrow P$;
2: **for** i from $l-2$ downto 0 **do**
3: $\quad Q \leftarrow 2Q$; $\quad$ **//Point Doubling**
4: $\quad$ **if** $k_i = 1$ **then**
5: $\quad\quad Q \leftarrow Q + P$; $\quad$ **//Point Addition**
6: $\quad$ **end if**
7: **end for**
8: **return** Q;

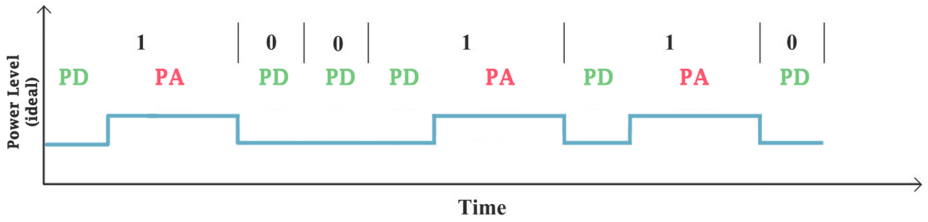


Figure 3.47. power profile DAA algorithm for ECPM

For enhanced security against SCAs, i have proposed the ECPM scheme employs the Montgomery ladder algorithm Algorithm 5. In this algorithm, point addition and point doubling are executed simultaneously to introduce uncertainty into the secret key. 3.48 validates the resistance of the Montgomery algorithm-based ECPM against SCAs by displaying its power tracing profile. In each iteration, the initial power consumption results from the combined power consumption of both the point addition (PA) and point doubling (PD) modules due to their parallel operation. Since the latency of point addition is greater than that of point doubling, point doubling is completed before point addition. Once the point doubling operation is finished, power is just consumed by the PA module. This approach effectively conceals the bit pattern of the secret key, making it impossible to deduce through power analysis.

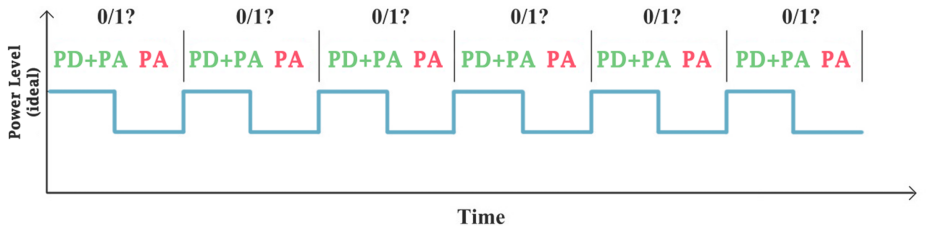


Figure 3.48. power profile Montgomery Ladder algorithm for ECPM

Algorithm 5 Montgomery Ladder Point Multiplication (Left to Right)

Require: $P, k = (\sum_{i=0}^{l-1} k_i 2^i)_{10}; \quad k_i \in \{0,1\}, \quad k_{l-1} = 1$
Ensure: Q
1: $Q1 \leftarrow P; Q2 \leftarrow 2P;$
2: **for** i from $l-2$ downto 0 **do**
3: **if** $k_i = 1$ **then**
4: $Q1 \leftarrow Q1 + Q2;$ **//Point Addition**
5: $Q2 \leftarrow 2Q2;$ **//Point Doubling**
6: **else**
7: $Q2 \leftarrow Q1 + Q2;$ **//Point Addition**
8: $Q1 \leftarrow 2Q1;$ **//Point Doubling**
9: **end if**
10: **end for**
11: **return** $Q1;$

3.7.1 ECPM architecture

Consolidated that Montgomery Ladder Multiplication is the best in the following is proposed the developed architecture that implement the algorithm, as presented in the abstraction there are both the datapath and the FSM that work together.

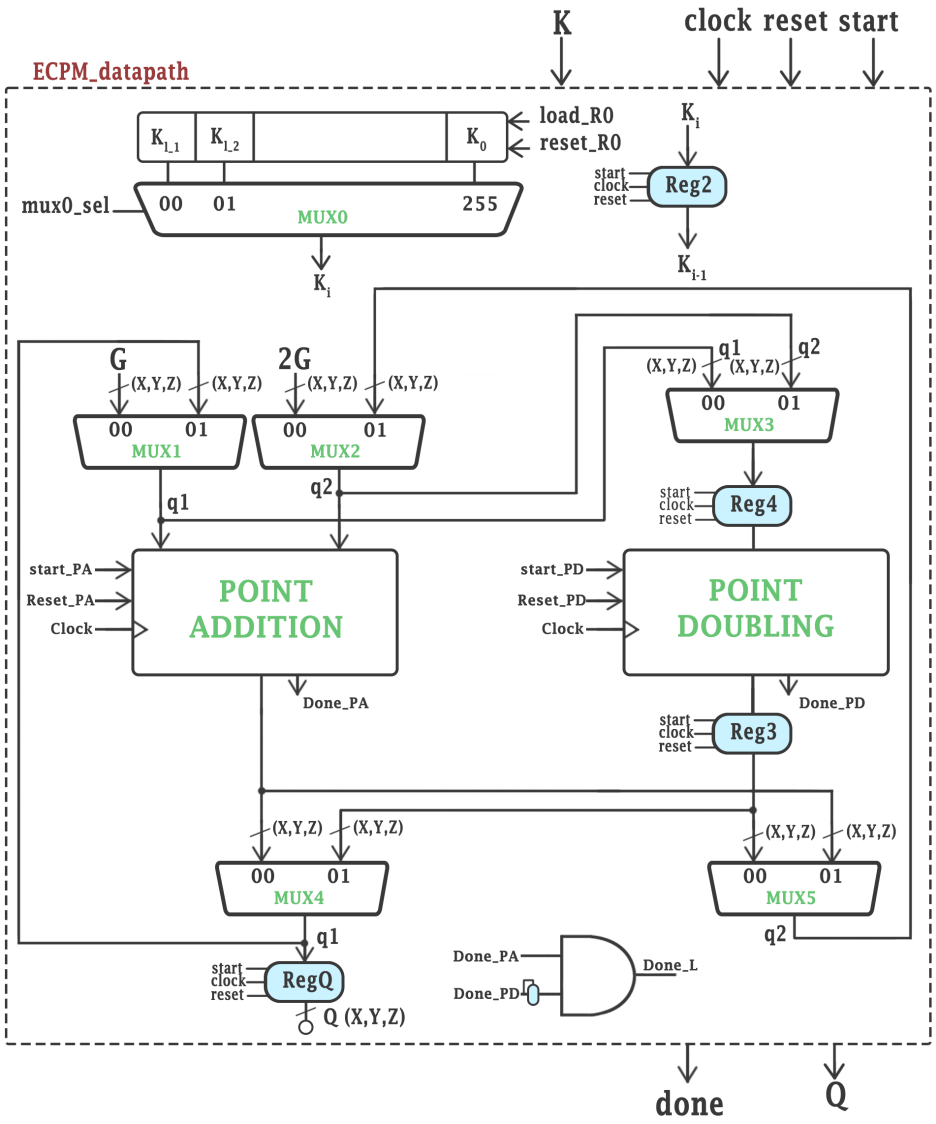


Figure 3.49. Architecture Montgomery Ladder algorithm

In fig. 3.49 is proposed the hardware implementation for the datapath, according to the Algorithm 5, pay attention to the fact that architecture work with value defined on three dimension (X, Y, Z) it means that each wire must be triplicate and also each multiplexer and register. Before describe the circuit i present the idea behind this algorithm, in practise we can write the constant K in binary form:

$$K = K_0 + 2K_1 + 2^2K_2 + +2^lK_l \quad \text{where} \quad l = [\log_2 K] \tag{3.5}$$

at this point we start to initialize two variable to the value of the generating point G and his double $2G$ then according to the value of the K_1 bit this points are doubled ore added, as follow i report the

first step of the algorithm also to understand why we start from the K_{l-1} bit, notice that i have used K ='11'.

$$\begin{aligned}
&Q_1 = \mathcal{O} \\
&Q_2 = G \\
&Q_1 = \mathcal{O} + G = G && //K_l = 1 && \text{point addition} \\
&Q_2 = G + G = [2]G && //K_l = 1 && \text{point doubling} \\
&Q_1 = [2]G + G = [3]G && //K_{l-1} = 1 && \text{point addition} \\
&Q_2 = [2]G + [2]P = [4]G && //K_{l-1} = 1 && \text{point doubling} \\
&... && ... \\
&Out = Q_1
\end{aligned} \tag{3.6}$$

It is now clear that the first step is starting from the infinite point that is not easy to manage therefore we compute the first iteration assuming and indirectly impose that $K - l$ must be equal to '1'.In the datapath $MUX1$ and $MUX2$ are used to charge as starting value G and $2G$ then shift register load the value of the constant K and tanks to $MUX0$ let pass out only one bit at time wen the past iteration are ended, then there are the two core block Point Addition and Point Doubling that are managed by the FSM, $MUX4$ and $MUX5$ instead are used to discriminate if we are in the condition of $K_i = 1$ or $K_i = 0$ and according to this refresh the Q_1 and Q_2 value necessary for the next iteration, and also in the end the REG_Q is loaded to fix the result in output. Notice that the loop between the input and the output of PA is pure cobinatorial, this allow to optimize the timing to reduce as much as possible the number of iteration and work in a more efficient cycling way, and this is not a problem for the critical path because inside the block there are just register in input and output, this allow to isolate the Cp.

feature:
Datapath Cost: 6 MUX + 1 Point Add. + 1 Point Doub. + 5 Reg. + 1 AND.
Critical Path: Critical Path Mod. Multiplier
Latency using RADIX-2 : $PD\_latency \cdot 255 + 1\ T_{ck}$ (DONE) $= (1030 \cdot 255) + 1 = 262651 T_{ck}$
Latency using RADIX-4 : $PD\_latency \cdot 255 + 1\ T_{ck}$ (DONE) $= (522 \cdot 255) + 1 = 133111 T_{ck}$

In fig. 3.50 is reported the FSM, as is possible to see seems very articulate because according to the algorithm each iteration is done checking the value of K_i to drive $MUX1$, $MUX2$, $MUX3$, but not also to manage the $MUX4$ and $MUX5$ is necessary have the previous value K_{i-1} according to the fact that in the meantime the value is updated, this create a net of decision that depend on both value creating many fork of the process. each level is iterate 255 time according to the fact that we start to the K_{l-1} bit, the number of iteration is memorized by a counter and when reach value '0' means that the cycle is ended and is possible to start new iteration.

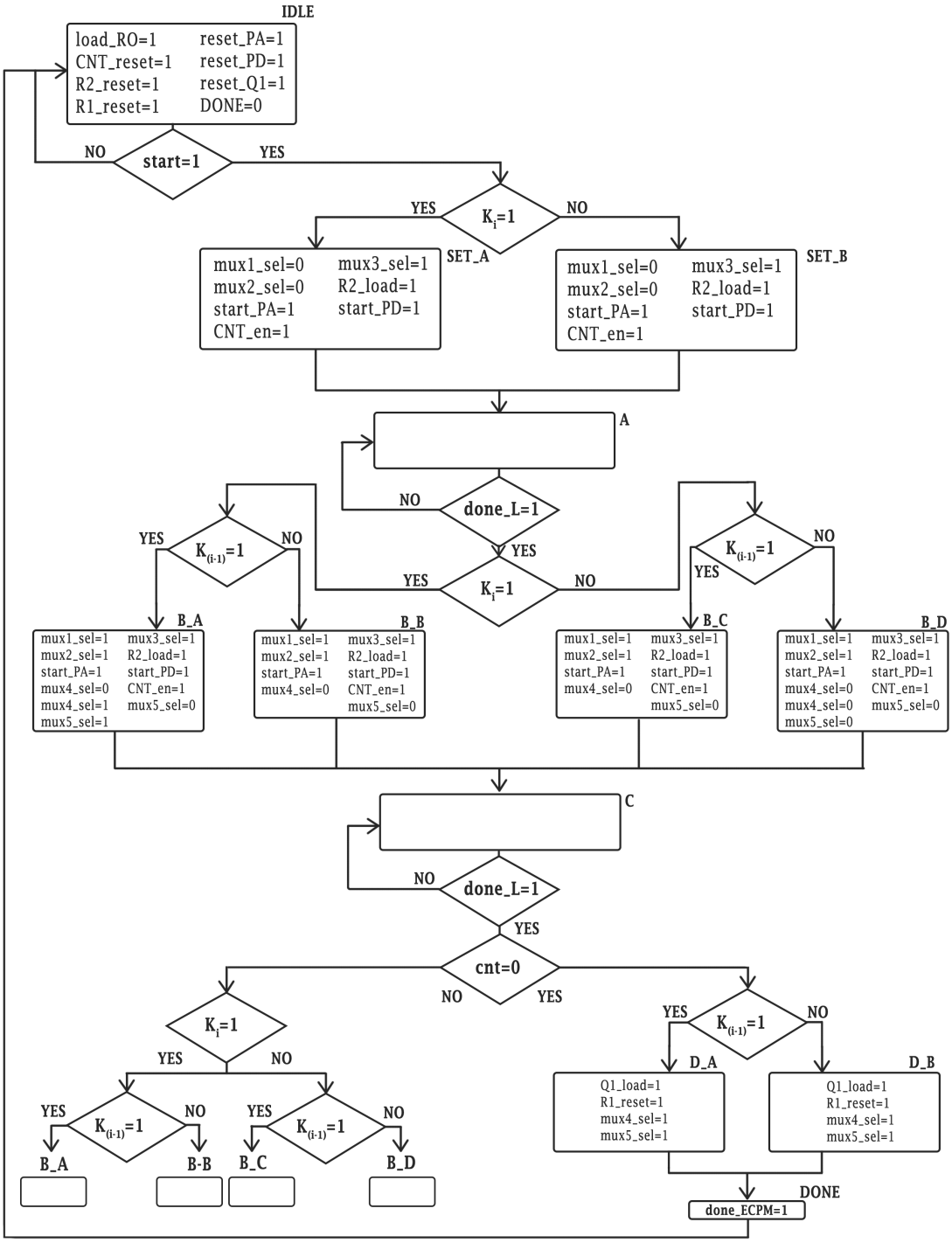


Figure 3.50. FSM ECPM

3.7.2 timing

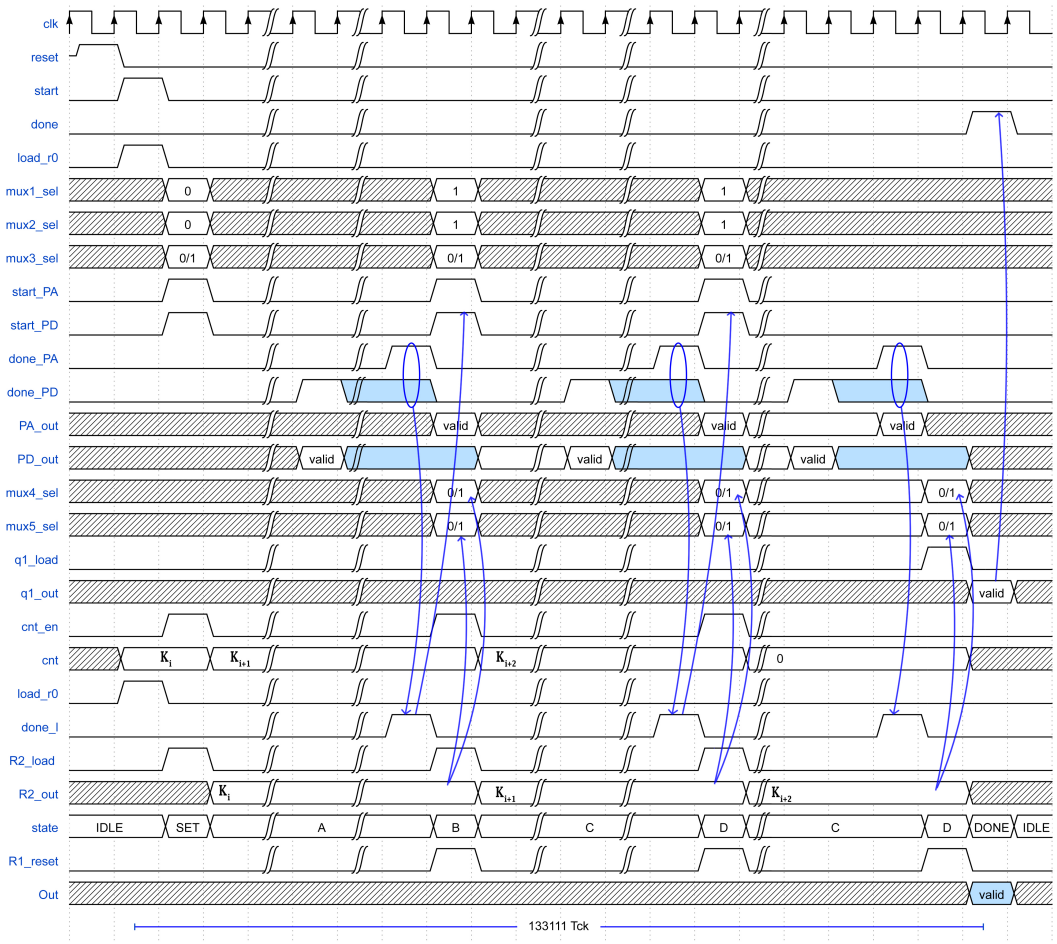


Figure 3.51. timing ECPM using RADIX-4 multiplier

3.7.3 VHDL simulation and Matlab validation

In this paragraph is proposed the Matlab code use to test the implemented hardware.

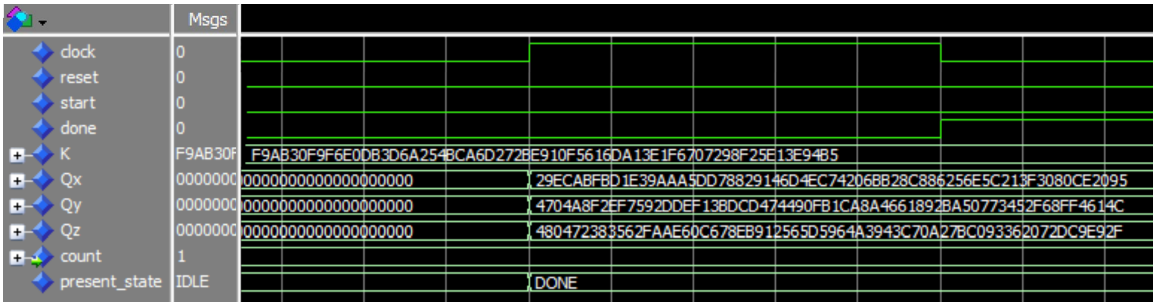


Figure 3.52. simulation ECPM using RADIX-4 multiplication

In the code are used the function *point_addition* and *point_doubling* that contain the code proposed in the section before, all the intermediate results are plotted to have e better comparison

and test of the hardware, in the end i also implement a test to verify the belonging of the output point Q to the curve Ed25519 substituting the point in the equation end verify the equality of both member.

```
% field dimension 0x52036cee2b6ffe738cc740797779e89800700a4d4141d8ab75eb4dca135978a3
% constant d 0x52036cee2b6ffe738cc740797779e89800700a4d4141d8ab75eb4dca135978a3
p = sym('57896044618658097711785492504343953926634992332820282019728792003956564819949');
d = sym('37095705934669439343138083508754565189542113879843219016388785533085940283555');
% PUBBLIC KEY : f9AB30F9F6E0DB3D6A254BCA6D272BE910F5616DA13E1F6707298F25E13E94B5
K = '1111100110101011001100001111100111110110111000001101101100111101011010100010010100
0101111001010011011010010011100101011111010010001000011110101011000010110110110100001001
1111000011111011001110000011100101001100011110010010111100001001111101001010010110101';
%generation point G(x,y,z)
Gx =sym('151122213495354007725011514095885315114540126930418572060461132839498477622 02');
Gy =sym('463168356949264781694283940034751631413079938662562256157830336031652518559 60');
Gz =sym('1');

[G2x,G2y,G2z]=point_doub(Gx,Gy,Gz);

%initialization
Q1x = Gx;
Q1y = Gy;
Q1z = Gz;
Q2x = G2x;
Q2y = G2y;
Q2z = G2z;

% Montgomery Ladder Point Multiplication
for i = 2:length(K)  %ilbit 255 e' escluso dall'algoritmo
    extractedBit = str2double(K(i));
    fprintf('counter : %d \n',i);

     if extractedBit == 1

        [Q1x,Q1y,Q1z]=point_add(Q1x,Q1y,Q1z,Q2x,Q2y,Q2z);
        [Q2x,Q2y,Q2z]=point_doub(Q2x,Q2y,Q2z);

        fprintf('Q1x: %s \n',dec2hex(Q1x,64));
        fprintf('Q1y: %s \n',dec2hex(Q1y,64));
        fprintf('Q1z: %s \n',dec2hex(Q1z,64));
        fprintf('Q2x: %s \n',dec2hex(Q2x,64));
        fprintf('Q2y: %s \n',dec2hex(Q2y,64));
        fprintf('Q2z: %s \n',dec2hex(Q2z,64));

    elseif extractedBit == 0
        [Q2x,Q2y,Q2z]=point_add(Q1x,Q1y,Q1z,Q2x,Q2y,Q2z);
        [Q1x,Q1y,Q1z]=point_doub(Q1x,Q1y,Q1z);
        fprintf('Q1x: %s \n',dec2hex(Q1x,64));
        fprintf('Q1y: %s \n',dec2hex(Q1y,64));
        fprintf('Q1z: %s \n',dec2hex(Q1z,64));
        fprintf('Q2x: %s \n',dec2hex(Q2x,64));
        fprintf('Q2y: %s \n',dec2hex(Q2y,64));
        fprintf('Q2z: %s \n',dec2hex(Q2z,64));
   end
end
%% checks that the point belong to the curve
lsh = mod((-(Q1x)^2+(Q1y)^2)*(Q1z)^2,p);
rhs = mod((Q1z)^4+d*(Q1x)^2*(Q1y)^2,p);
if lsh == rhs fprintf('THE POINT BELONG TO THE CURVE \n'); end
% --------------------------
% Q1x: 29ECABFBD1E39AAA5DD78829146D4EC74206BB28C886256E5C213F3080CE2095
% Q1y: 4704A8F2EF7592DDEF13BDCD474490FB1CA8A4661892BA50773452F68FF4614C
% Q1z: 480472383562FAAE60C678EB912565D5964A3943C70A27BC093362072DC9E92F
% THE POINT BELONG TO THE CURVE
```

As is possible to see the result of the Matlab simulation is the same as the VHDL simulation, means work well.

3.8 Conversion Projective to Affine coordinates

In this last section of hardware implementation is presented final harware in wich is managed also the conversion of the result from projective to affine in practise the result defined on three dimension $Q(X, Y, Z)$ is transposed in $Q(x, y)$ to do this we apply the eq. 2.16

$$x = \frac{X}{Z} \quad y = \frac{Y}{Z}$$

In figure 3.53 is proposed a brief recap of the ECC process

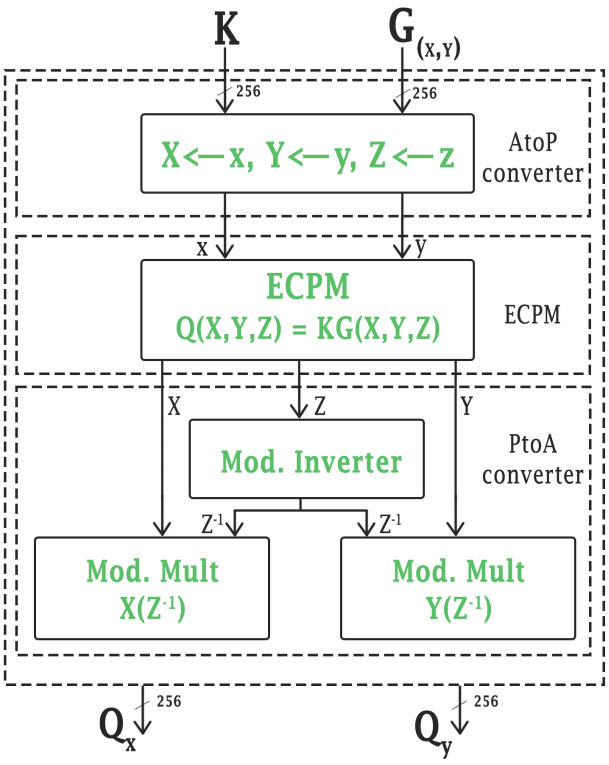


Figure 3.53. ECC proces

3.8.1 implementation

About the implementation is necessary allocate the inversion block described in section before use to compute Z^{-1}, then with two modular multiplier is computed the affine conversion of Q_x and Q_y multiplying projective value for the inverted Z coordinate. For this implementation is not necessary define an FSM .

feature:
conversion Cost: 1 Point Inversion + 2 mod. mult RDIX-4 + 1 Reg.
Critical Path: Critical Path Mod. Multiplier
Latency using RADIX-4 : $(n + \frac{n}{4}) + (\frac{n}{2} + 1) + 1$(Done) $T_{ck} = (320 + 129) + 1$ $T_{ck} = 450$ T_{ck}

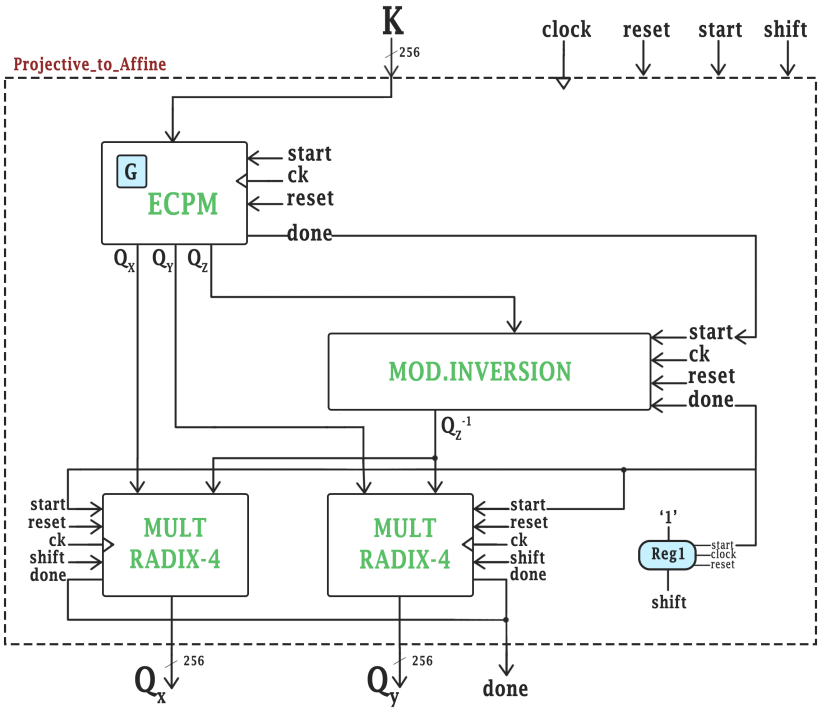


Figure 3.54. Architecture projective to affine conversion

3.8.2 timing

The three step work in sequence because is necessary have the result of previous stage to proceed, time require for modular inversion is very high and this is the reason why using projective coordinate we spend this time only once.

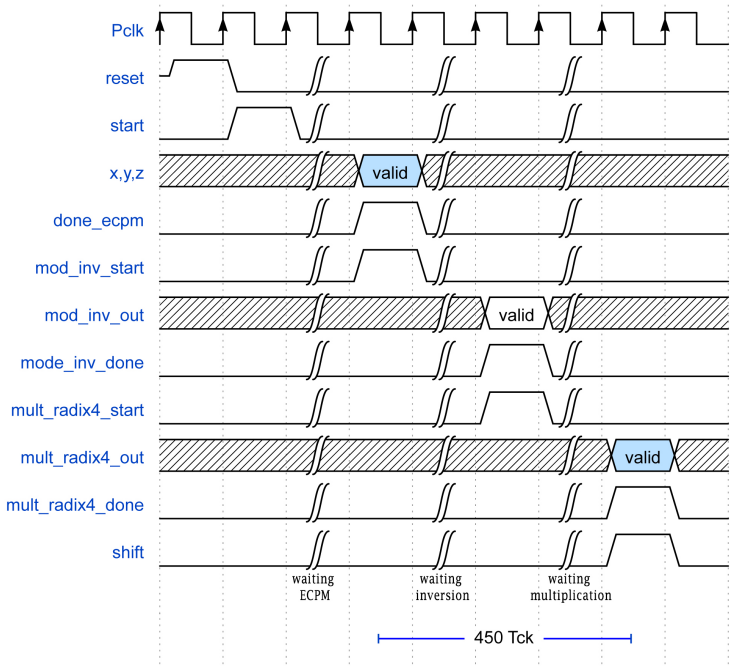


Figure 3.55. timing projective to affine conversion

3.8.3 software validation

This time for the validation we ca use the result of the inversion implemented in Matlab in the dedicated section or otherwise i propose a java code that implement the overall ECPM taking in input the private-key K and use some library for the parameter of generating point, constant, and overall algorithm for the Ed25519, the output is public-key defined in affine coordinate Q_x and Q_y.

java validation

```
1
2  ////////////////////////////////
3  // apt install npm
4  // npm install elliptic
5  //
6  // node ECC.js
7  ////////////////////////////////
8
9  const elliptic = require('elliptic');
10 const ed25519 = new require('elliptic').eddsa('ed25519');
11 const BN = require('bn.js');
12
13 function printPointInfo(desc, P) {
14   console.log();
15   console.log(`${desc}: x-coord: 0x` + P.getX().toString(16));
16   console.log(`${desc}: y-coord: 0x` + P.getY().toString(16));
17   console.log();
18 }
19 //generating point
20 let G = ed25519.curve.g;
21 //public-key
22 let K = 'f9ab30f9f6e0db3d6a254bca6d272be910f5616da13e1f6707298f25e13e94b5';
23
24 printPointInfo('G', G);
25 printPointInfo('2G', G.mul(new BN(2)));
26 printPointInfo('5G', G.mul(new BN(5)));
27 printPointInfo('KG', G.mul(new BN(K, 16)));
28 //---------------------------------------------
29 //G: x-coord: 0x216936d3cd6e53fec0a4e231fdd6dc5c692cc7609525a7b2c9562d608f25d51a
30 //G: y-coord: 0x6666666666666666666666666666666666666666666666666666666666666658
31
32 //2G: x-coord: 0x36ab384c9f5a046c3d043b7d1833e7ac080d8e4515d7a45f83c5a14e2843ce0e
33 //2G: y-coord: 0x2260cdf3092329c21da25ee8c9a21f5697390f51643851560e5f46ae6af8a3c9
34
35 //5G: x-coord: 0x49fda73eade3587bfcef7cf7d12da5de5c2819f93e1be1a591409cc0322ef233
36 //5G: y-coord: 0x5f4825b298feae6fe02c6e148992466631282eca89430b5d10d21f83d676c8ed
37
38 //KG: x-coord: 0x6040cfb1c92d97a508a4a25260e0c9fdf1a737af9bc20b0816a88d2c89266f44
39 //KG: y-coord: 0x63b3e360cf5aeee53678ecd967a4c119b895a37f3f39e7c2fb2be3cdd2611108
```

There are also other test calculation for intermediate check during the development of the architecture, for this verification KG result are the Target.

VHDL simulation

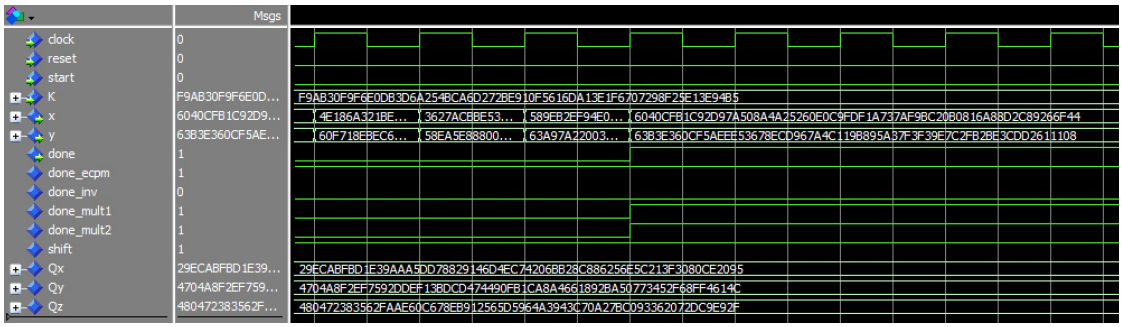


Figure 3.56. ECPM projective to affine conversion

As is possible to see the result are the same it means that the overall architecture of crypto-core based on Edward elliptic curve Ed25519 work well and all the improvement ad modification applied are good. In next chapters are presented the FPGA synthesis and some result comparison with other implementation taken in some paper, and in conclusion an owerview of a possible ASIC synthesis.

Chapter 4

Virtex-7 FPGA synthesis

In this chapter is proposed the FPGA syntheses of the Crypto-core implemented in VHDL and using Xilinx Vivado tool, synthesis is done for the Virtex-7 (XC7V2000TFLG1925-1) FPGA, then the results are compared with other implementation proposed in some paper.

4.1 implementation results

in this section are reported the results of the synthesis, these are just an overview of the general trend of the architecture because is necessary remember that FPGA are composed by preallocated block called CLB that contain in turn other elementary block as adder, multiplier and some other that are called DSP. The big problem of this device is that is necessary interconnect all this block using LUT, it means very long interconnection due to the fact that different CLB could be far apart from each other and according to the scheduling approach used by the software to synthesize the hardware code this block could allocate in many way and each approach generate very different result in term of critical path and area, in conclusion it means that this analysis is just done to have an idea of the performance of the devise.

In table 4.1 there is a summery of more relevant block syntheses.

Operation	Clock cycles	Number of slice (k)	Number of LUTs/DSP	Maximum frequency(MHz)	Time (ms)	Area x time (AT)	Throughput (bps)
elem. Adder (258 bit)	1	0.02	11 LUT,11 DSP	324.6	$3.08 \cdot 10^{-6}$	$61.6 \cdot 10^{-9}$	$8.30 \cdot 10^{10}$
modular ADD/SUB	2	0.434	1197 LUT,22 DSP	105.19	$19.01 \cdot 10^{-6}$	$8.25 \cdot 10^{-6}$	$1.346 \cdot 10^{10}$
mod. Mult. RADIX-4	129	1.258	2550 LUT,66 DSP	95.1	$1.356 \cdot 10^{-3}$	$1.68 \cdot 10^{-3}$	$1.88 \cdot 10^{8}$
Point Addition	523	4.230	12908 LUT,374 DSP	~ 95	$5.5 \cdot 10^{-3}$	$2.32 \cdot 10^{-2}$	$4.65 \cdot 10^{7}$
Point Doubling	264	1.135	5378 LUT,352 DSP	~ 95	$2.77 \cdot 10^{-3}$	$3.15 \cdot 10^{-3}$	$9.21 \cdot 10^{7}$
ECPM	133111	8.375	25685 LUT,726 DSP	~ 95	1.40	11.725	$1.82 \cdot 10^{5}$
ECPM + PtoA conv.	133561	8.970	26350 LUT,1078 DSP	~ 95	1.405	12.602	$1.82 \cdot 10^{5}$

Table 4.1. implementation results of ECC synthesis on FPGA

Results are overall slightly good but in some part are enough different from paper as we will see probably because paper use more advanced constraint to obtain better result to contain as much as possible the area and in consequence also the critical path could be more content, i have tried a few strategies, keeping in mind that they take very long time but more or less the result are very close, in general i can say that i am satisfied.

4.2 performance comparison

Operation	Publishing year	Platform	CCs	Number of slice (k)	Maximum freq.(MHz)	Time (ms)	Area x time (AT)	Throughput (kbps)
My-ECPM	-	Virtex-7	133561	8.970	~ 95	1.405	12.602	182.09
Mainul *et al.*[7]	2020	Virtex-7	198700	6.5	104.39	1.9	12.35	134.49
Mainul *et al.*[8]	2019	Virtex-7	262700	8.9	177.7	1.48	13.17	173.2
Shah *et al.*[9]	2018	Virtex-6	153200	65.60	327.00	0.47	30.83	546.42
Liu *et al.*[16]	2017	Virtex-4	459900	12.00	36.50	12.60	151.20	20.32
Asif *et al.*[13]	2017	Virtex - 7	215900	24.20	72.90	2.96	71.63	1816.20
Javeed *et al.*[14]	2017	Virtex - 4	191600	20.60	49.00	3.91	80.55	65.47
Hossain *et al.*[11]	2016	Virtex-7	397300	11.30	121.50	3.27	36.95	78.28
Marzouqi *et al.*[12]	2016	Virtex-5	361600	8.70	160.00	2.26	19.66	113.27
Javeed *et al.*[15]	2016	Virtex - 4	200000	13.20	40.00	5.00	66.00	51.00
Javeed *et al.*[16]	2016	Virtex - 4	207100	35.70	70.00	2.96	105.67	86.53
Loi *et al.*[17]	2015	Virtex - 4	993700	7.00	182.00	5.46	38.22	46.88
Lee *et al.*[18]	2014	Virtex - II Pro	163200	8.30	37.00	4.41	36.60	58.04
Marzouqi *et al.*[19]	2013	Virtex - 5	442200	10.20	66.70	6.63	67.63	38.61
Ghosh *et al.*[20]	2011	Virtex - II - Pro	337700	12.00	36.00	9.38	112.56	27.29
Ghosh *et al.*[21]	2009	Virtex - 4	331100	20.10	43.00	7.70	154.77	33.25
Ananyi *et al.*[22]	2009	Virtex - 4	414000	20.80	60.00	6.10	126.08	37.10
Schinianakis *et al.*[23]	2009	Virtex - E	156800	16.40	39.70	3.95	64.78	64.82
McIvor *et al.*[24]	2006	Virtex - II Pro	151400	15.80	39.50	3.86	60.98	66.74

Table 4.2. implementation comparison

in table 4.2 are reported some results of other implementation using different approach and syntheses strategy and also different board , paper [7] and [8] are the two most relevant reference also used to understand and project my ECPM, they use the Projective coordinate approach, this means result comparison is more relevant. In the following i have proposed three bare graph to have a better overview on the performance comparison, graph 4.1 report the number of clock cycle necessary for the completion of ECPM and conversion of the result from projective to affine.

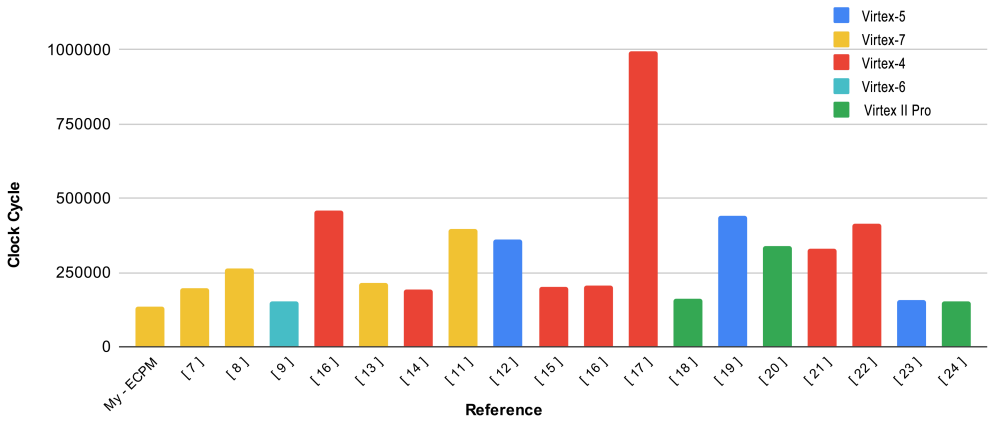


Figure 4.1. Clock Cycle comparison graph

4.2 report the time Area product, is a parameter of quality that take in consideration two of the more relevant design parameter.

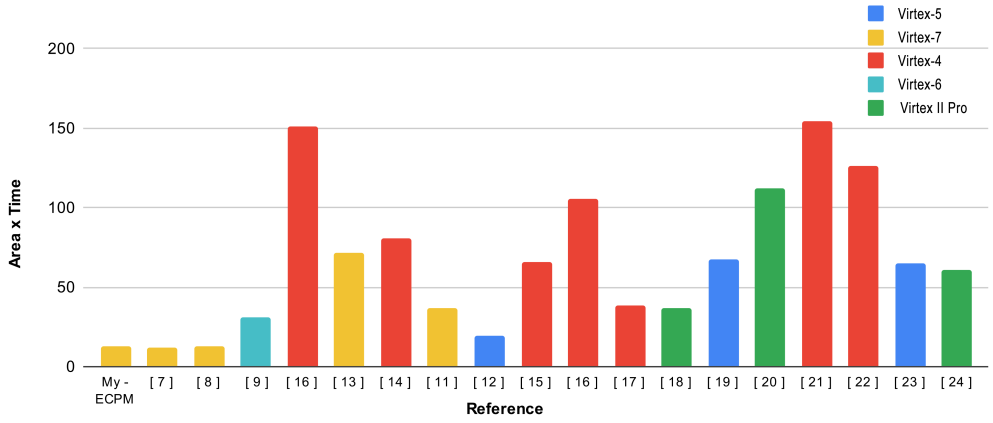


Figure 4.2. Area x Timing comparison graph

In the end the third graph 4.3 the throughput is the amount of work that can be completed in a given amount of time and also this case is a goodness parameter.

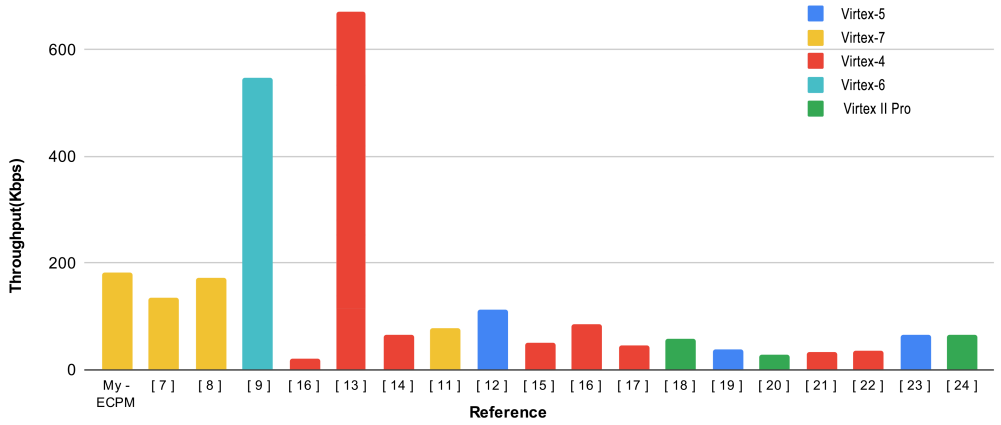


Figure 4.3. Throughput comparison graph

As is possible to see the result are good in particular to the point of view of the number of clock cycle, this is because I invested a lot of time trying to get the best count possible, the area as i say before is not the best as possible but for our scope in any case is good and despite of this i have obtained also a very good result in term of time area product and throughput.

More in general seeing the trend can be observed that over the year the performance become more and more better also thank to the fact that the board and the software become more complex and efficient and also because are introduced new mathematical approach as i have described in dedicated paragraph.

Next and last step is the implementation on silicon ASIC of the crypto-core, in this case we expect a very good improvement of the performance.

Chapter 5

Application Specific Integrated Circuit (ASIC)

In this last chapter i have propose the synthesis made with Synopsys using $Nangate45$ library, instead of using an FPGA the result we will expect should be better because of the fact that in this case we do not have LUT and CLB, that are fixed unit, but we can customize so much the hardware only allocate the standard cell that are very small elementary block, and defining all the interconnection on many layer, this allow to obtain very small area and in consequence reducing the path length we reduce also the critical path, means high frequency.

5.1 Synthesis with Synopsys

First step is charging the developed $VHDL$ code of the crypto-core and the $Nangate45$ library in Synopsys, then before starting the syntheses is necessary applying constraint to the clock and load buffer for each output. About the load for each output i have used a buffer defined in the technology library called $BUFF\_X4$ this allow to have more realistic report results, instead about the clock i have fixed for first iteration the period to 0 ns, doing this i will obtain a negative slack that is exactly equal to the critical path, i substitute it and i relaunch the synthesis to obtain a slack equal to 0.

In table 5.1 is reported the Timing report abstract, with clock feature

clock uncertainty	0.07 ns
library setup time	0.04 ns
input delay	0.5 ns
output delay	0.5 ns
Critical path	3.73 ns
Maximum frequency	268.09 Mhz

Table 5.1. Timing report

The clock uncertainty is fixed to 0.07 ns, very small value respect the clock period, this allow to

implement a circuit whit better performance as possible, about the input/output delay of the signal respect to the clock, much more is reduced much more the path are optimized.

In table 5.2 is reported the report Area abstract

Number of ports	844404
Number of nets	1231141
Number of cells	392246
Number of combinational cells	316722
Number of sequential cells	37113
Number of buff/inv	71425
Combinational Area	333923.631657 μm^2
Buf/Inv area	43279.795835
Non combinational area	188463.666081 μm^2
Total cell area	522387.297738 μm^2

Table 5.2. Report area

As is possible to notice the results are so better respect the FPGA synthesis, about the critical path there is a reduction of 64.5% , from 10.52 ns of the FPGA to 3.73 ns, also about the area the there is a substantial reduction but is not directly comparable view the nature of the FPGA, computing approximately the dimension we could say that is $\sim 730\mu m \times 730\mu m$ which is certainly much less respect the FPGA implementation.

5.2 Place and Rout with Cadence Innovus

In this last step is implemented the physical circuit on silicon, the technology used is always the 45 nm, after having generated the netlist we import that in Innovus and proceed with the *ASIC* synthesis. About the dimension of the core is used an aspect ration of 1, means square chip, about the utilization is fixed to 0.6 and the boundary of the core is forced to 5 μm. After having inserted the power rings with a width of 0.8 μm i decide also to insert Stripes considering the dimensions not exactly small o the core, this allow to improve the power supply distribution and have more controlled and better performance. Then the placement is done on eight layer using the standard cell implemented in the library, doing two optimization both before the clock tree synthesis and after, about the clock the maximum transition time is fixed to 0.08 ns and the maximum skew at 0.5 ns. Then is done the routing and another optimization of the design, in the end is done a place filler to guarantee all the continuity interconnection between $n+$ and $p+$ wells in each row.

Also in this case there are the reports of the timing and area of the physical on silicon Core in 5.3

Gate area	0.7980 μm^2
Number of gates	585040
Number of cells	300992
Total area	466861.9 μm^2
Critical path	3.910 ns
Max frequency	255.75 Mhz

Table 5.3. Report Place and Rout

The value reported are very similar to those obtained in the previous synthesis phase, critical path is slightly increased probably due to the fact that now the parasitic due to the interconnection are more well defined and overall delay suffer from it, about the are now we can approximate the square core to $\sim 683\mu m \times 683\mu m$ in this case it is slightly reduced respect the one approximate in synthesis step.

For what concern the Power in 5.4 are reported all the contribution of sequential, combinational and clock power, how is possible to see the $\sim 67.91\%$ of the total power is due to the combinational path and only ~ 4.7

	Internal Power $[mW]$	Switching Power $[mW]$	Leakage Power $[mW]$	Total Power $[mW]$	Percentage [%]
Sequential	71.88	22.59	3.062	97.53	27.38
Combinational	113.9	121.5	6.536	241.9	67.91
Clock	1.508	15.25	0.01685	16.77	4.708
Total	187.3 mW	159.3 mW	9.615 mW	356.2 mW	100 %

Table 5.4. Report Power

In conclusion is reported the overall ASIC layout synthesized with Innovus in fig. 5.1

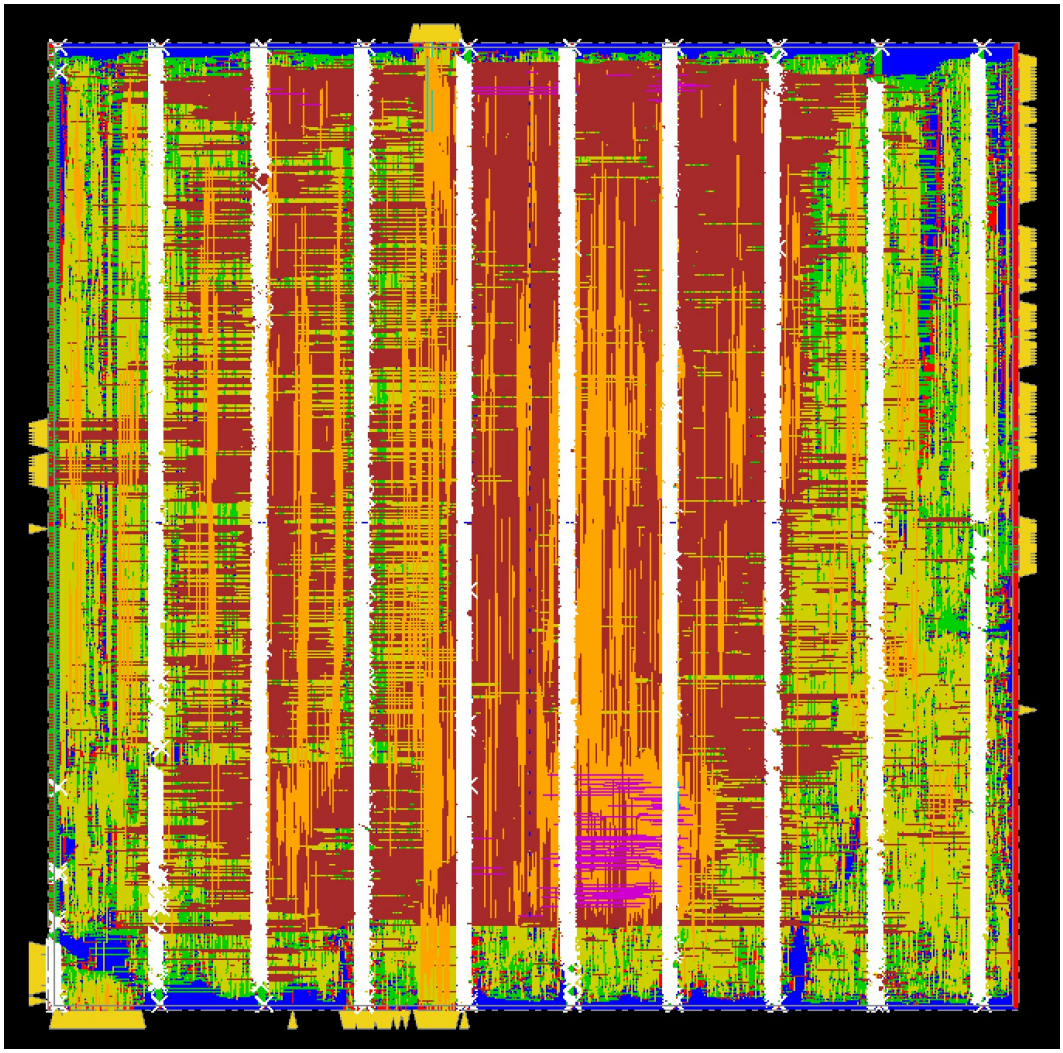


Figure 5.1. ASIC layout

Chapter 6

conclusion and future work

6.1 Conclusion

In conclusion this thesis work It was complex but interesting, allowing me to discover new things and refine my knowledge in the field of digital design, much of the work was done on the dedicated architectures for the various blocks, trying to optimize them as best as possible by reducing as much as possible the number of iterations necessary for completing the operations, and at the same time, through careful scheduling work, also reducing the number of resources allowing to have an area as small as possible. Both objectives have been achieved and the results are quite satisfactory, the core has been implemented with both a RADIX-2 and a RADIX-4 multiplier, in the second case 133111 T_ck are needed to complete the algorithm. Then synthesizing with Vivado and using a virtex-7 FPGA we have obtained medium good performances, as we expected, having an execution time of around 1,405 ms and a maximum working frequency of 95 MHz, then I have resynthesized with Synopsys and I have implemented the on silicon ASIC with a 45 nm technology the results were decidedly better, as we expected, obtaining an overall execution time of 0.52 ms about a third of the FPGA implementation, working at a maximum frequency of 255.75 Mhz with a silicon core of approximately $\sim 683\mu m \times 683\mu m$

6.2 future work

Possible improvement could be done on both front,starting from this implementation first trying to modify the modular multiplier reducing the critical path considering that is the bottle neck of the architecture, possible way is unrolling the architecture and inserting a level of pipe in the middle this allows the critical path to be reduced and the new critical block becomes the modular inverter, in this case this one is just developed with one level of pipe, it means that is necessary again try to unroll and test the performance. About the synthesis possible improvements could be done trying to reach better constraint in Vivado to have a more better allocation of the CLB a and DSP and trying to change the technology in the ASIC synthesis.

Acronym

ECC Elliptic Curve Cryptography

EC Elliptic Curve

ECDLP Elliptic Curve Discrete Logarithm Problem

ECDSA Elliptic Curve Digital Signature Algorithm

EdDSA Edwards curve Digital Signature Algorithm

ECPM Elliptic Curve Point Multiplication

ECSM Elliptic Curve Scalar Multiplication

DHKE Diffie–Hellman Key Exchange

ECDH Elliptic Curve Diffie–Hellman Key Exchange

ASIC Application Specific Integrated Circuit

DSP Digital Signal Processing

FPGA Field Programmable Gate Array

GF Galois Field

LUT Look Up Table

NIST National Institute of Standards and Technology

PA Point Addition

PD Point Doubling

SCAS Side-Channel Attacks

IoT Internet of Things

P Prime field dimension

Gx, Gy Generators point coordinates EC

Bibliography

[1] Yuhan Yan The Overview of Elliptic Curve Cryptography (ECC) 2022 J. Phys.: Conf. Ser. 2386 012019

[2] Weng Chun Tan, Manjit Singh Sidhu *Review of RFID and IoT integration in supply chain management*, Operations Research Perspectives, Volume 9,2022,100229,ISSN 2214-7160.

[3] Mishall Al-Zubaidie, Zhongwei Zhang and Ji Zhang , Efficient and Secure ECDSA Algorithm and its Applications: A Survey, Vol. 11, No. 1, April 2019

[4] Virtex-7 T and XT FPGAs Data Sheet: DC and AC Switching Characteristics, DS183 (v1.29) March 23, 2021

[5] Katz, J., & Lindell, Y. (2007). Introduction to Modern Cryptography: Principles and Protocols (1st ed.). Chapman and Hall/CRC. https://doi.org/10.1201/9781420010756

[6] Delaplace, Claire and May, Alexander. "Can we Beat the Square Root Bound for ECDLP over Fp2 via Representation?" Journal of Mathematical Cryptology, vol. 14, no. 1, 2020, pp. 293-306. https://doi.org/10.1515/jmc-2019-0025

[7] Islam, Md. Mainul & Hossain, Md & Hasan, Moh Khalid & Shahjalal, Md & Jang, Yeong Min. (2020). Design and Implementation of High-Performance ECC Processor with Unified Point Addition on Twisted Edwards Curve. Sensors. 20. 5148. 10.3390/s20185148.

[8] Islam, Md. Mainul & Hossain, Md & Hasan, Moh Khalid & Shahjalal, Md & Jang, Yeong Min. (2019). FPGA Implementation of High-Speed Area-Efficient Processor for Elliptic Curve Point Multiplication Over Prime Field. IEEE Access. 7. 178811-178826. 10.1109/ACCESS.2019.295849

[9] Shah, Y.A.; Javeed, K.; Azmat, S.; Wang, X. Redundant signed digit based high-speed elliptic curve cryptographic processor. J. Circuits Syst. Comput. 2018, 28, 1950081.

[10] Liu, Z.; Liu, D.; Zou, X. An efficient and flexible hardware implementation of the dual-field elliptic curve cryptographic processor. IEEE Trans. Ind. Electron. 2017, 64, 2353–2362.

[11] Hossain, M.S.; Kong, Y.; Saeedi, E.; Vayalil, N. High-performance elliptic curve cryptography processor over NIST prime fields. IET Comput. Digit. Tech. 2016, 11, 33–42.

[12] Marzouqi, H. ; Al-Qutayri, M.; Salah, K.; Schinianakis, D.; Stouraitis, T. A high-speed FPGA implementation of an RSD-based ECC processor. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 2016, 24, 151–164.

[13] S. Asif, M. S. Hossain, and Y. Kong, “High-throughput multi-key elliptic curve cryptosystem based on residue number system,” IET Comput. Digit. Techn., vol. 11, no. 5, pp. 165–172, 2017.

[14] K. Javeed, X. Wang, and M. Scott, “High performance hardware support for elliptic curve cryptography over general prime field,” Microprocess. Microsyst., vol. 51, pp. 331–342, Jun. 2017

[15] K. Javeed and X. Wang, "Low latency flexible FPGA implementation of point multiplication on elliptic curves over GF(p)," Int. J. Circuit Theory Appl., vol. 45, no. 2, pp. 214–228, 2016.

[16] K. Javeed and X. Wang, "FPGA based high-speed SPA-resistant elliptic curve scalar multiplier rchitecture," Int. J. Reconfigurable Comput., vol. 2016, no. 5, pp. 1–10, 2016.

[17] K. C. C. Loi and S. B. Ko, "Scalable elliptic curve cryptosystem FPGA processor for NIST prime curves," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 23, no. 11, pp. 2753–2756, Jan. 2015.

[18] J.-W. Lee, S.-C. Chung, H.-C. Chang, and C.-Y. Lee, "Efficient poweranalysis-resistant dual-field elliptic curve cryptographic processor using heterogeneous dual-processing-element architecture," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 22, no. 1, pp. 49–61, Feb. 2013.

[19] H. Marzouqi, M. Al-Qutayri, and K. Salah, "An FPGA implementation of NIST 256 prime field ECC rocessor," in Proc. IEEE Int. Conf. Electron. Circuits Syst. (ICECS), Dec. 2013, pp. 493–496.

[20] S. Ghosh, D. Mukhopadhyay, and D. Roychowdhury, "Petrel: Power and timing attack resistant elliptic curve scalar multiplier based on programmable GF(p) arithmetic unit," IEEE rans. Circuits Syst. I, Reg. Papers, vol. 58, no. 8, pp. 1798–1812, Jan. 2011.

[21] S. Ghosh, M. Alam, D. R. Chowdhury, and I. S. Gupta, "Parallel cryptodevices for GF(p) elliptic curve ultiplication resistant against sidechannel attacks," Comput. Electr. Eng., vol. 35, no. 2, pp. 329–338, 2009.

[22] K. Ananyi, H. Alrimeih, and D. Rakhmatov, "Flexible hardware processor for elliptic curve cryptography over NIST prime fields," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 17, no. 8, pp. 1099–1112, Aug. 2009.

[23] D. Schinianakis, A. Fournaris, H. Michail, A. Kakarountas, and T. Stouraitis, "An RNS implementation of an Fp elliptic curve point multiplier," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 56, no. 6, pp. 1202–1213, Jun. 2009.

[24] C. J. Mcivor, M. Mcloone, and J. V. Mccanny, "Hardware elliptic curve cryptographic processor over GF(p)," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 53, no. 9, pp. 1946–1957, Sep. 2006.