Optimization of the Elliptic Curve Cryptography Implementation On A MicroBlaze Processor

Grant Genie, Daisy Guzman, Newton Le, Laurice Sattouf, Andro Younan, Mohamed El-Hadedy

gtgenie@cpp.edu, dguzman@cpp.edu, newtonle@cpp.edu, lsattouf@cpp.edu, ayounan@cpp.edu, mealy@cpp.edu  
*Department of Electrical and Computer Engineering*

*College of Engineering, California Polytechnic State University, Pomona*

Pomona, California

***Abstract* — The elliptic curve cryptography (ECC) is a fast and more secure method of encryption, compared to the Public-Key Cryptographic algorithms of RSA, ECC provides high security with much shorter algorithms. In this paper, we will discuss the algorithms and implementations to use this network security in a micro blaze processor using a Field - programmable gate arrays (FPGA) based system implementation.  For the cryptographic operations being fast and accurate, the elliptic curve functions over two finite fields, the prime and binary field. There have been many projects that are done over the prime field FP which is chosen with finitely large numbers of points to have more security. Fields of large prime characteristic are very popular since they are capable of implementing techniques borrowed from other finite field based cryptographic systems like the other popular public key cryptography algorithm, RSA. However, using this finite field is not as effective, lacks in performance and slows down the process speed, so we opted to use the binary field algorithm. In this paper, we put forward an effective FPGA implementation of the elliptic curve (EC) point multiplication in the binary field. We used the right-to-left comb algorithm, also called the add-shift algorithm, for the point multiplication module; this algorithm is well-suited for hardware purposes and can be performed in one clock cycle. Comparing the differences in algorithms in the binary field, we designed this algorithm to help demonstrate that the implementation of the FPGA can be fast and as effective in its ability to be secure from its counterparts, using shorter keys intended for study purposes than other multiplication methods, such as the windowed methods we will be comparing it to in this project.**

***Keywords—Elliptic curve cryptography, FPGA, Microblaze, finite field arithmetic, multiplication, point addition, public-key, private-key, circuit design, benchmark***

# Introduction

ECC is one method of public-key cryptography that has many benefits compared to the other forms of public-key cryptography. We have conducted this study to implement into the FPGA-based system to verify its main benefits when it comes to achieving fast high security levels with shorter keys. ECC uses 2*n*-bit keys for achieving *n* bits of security [1]. For study cases, we will use a small bit key to achieve the implementation. Fig. 1 showcases the flows of ECC, which has four steps with different sets of operations at each layer [4]. The main objective of this study is to find and implement a low bit key cryptographic algorithm onto the FPGA. We have taken in consideration the algorithms formulated from other studies to implement more effective cryptographic algorithms onto our project.

Chart

Description automatically generated

Fig 1: Bottom-up Hierarchy of Elliptic Curve Cryptography [4]

# Related Work

ECC has two methods, one of the related work that we looked into, used the prime field method. In this method the equation of the elliptic curve has integers between 0 and p-1 and all of the modules follow this algorithm [2].

*y2 mod p= x3 + ax + b mod p,* (1)

*where 4a3 + 27b2 mod p ≠ 0.*

Prime field processors have been used on general – purpose CPU since common CPUs are given a large integer multiplier circuit, in contrast from a large binary integer multiplier circuit. However, as stated in the abstract section, binary curves are smaller and faster in hardware than the prime field. This is because they have shorter formulas, binary squaring is linear, and the binary field does not contain the carry function.

For the multiplication operation, the biggest difference is the prime field’s multiplier, which uses full adder operations [10]. Fig. 2 and Fig. 3 shows the difference in the multiplication modules, this operation gives the most change in speed and performance between the two finite fields. This binary field method will be tested with the multiplication windowed algorithm to show that it is easier to implement than any other algorithm and takes less power since it is better at using less resources.

Diagram, schematic

Description automatically generated

Fig 2: 4-bit integer multiplier for prime field [10].

Diagram

Description automatically generated

Fig 3: 4-bit carry-less multiplier for binary field [10].

# Design Architecture

## Binary Field Arithmetic

First to improve hardware performance, the curve points exist in the binary field where the elements of the finite field are integers of length at most *m* bits. All of the operations like addition, division, subtraction and multiplication involves the polynomials of degree *m*-1 or less [2]. The *m* is a large finite number of points.

## Polynomial Equation

An elliptic curve over GF(2n) is defined by the simplified Weierstrass equation [2]:

*y² + xy = x3 + ax2 + b* (2)

*where a ≠ 0 and b ≠ 0*

This type of curve is considered to be a non-super singular elliptic curve defined over the binary field *K*. When we multiply two polynomials, we will gather results with a degree larger than the original value. In our case, this will be larger than order of four, in our modules we reduce the results from the multiplication and reduce it to fit onto our desired registers. In the degree we have chosen to implement, we choose between three irreducible functions which remains constant throughout our project:

*f1(z) = z4 + z + 1* (3)

*f1(z) = z4 + z3 + 1*  (4)

*f1(z) = z4 + z3 + z2 + x + 1* (5)

## Modules Implemented

To be able to implement the polynomial to perform multiplication, we need to reduce the results given to fit into the registers we chose. ECC uses modular arithmetic or polynomial arithmetic for its operations depending on the field chosen [1].

### Point Addition and Point Doubling

Point addition considers whether *P = Q*, with this case the point doubling algorithm is implemented. There are other cases to consider when *P* does not equal *Q*. Lastly, if all of the sub-cases were to fail, then *P* and *Q* are not the identity and are distinct. Fig. 4 and Fig. 5 shows the procedures when a ∈ {0, 1}.

|  |
| --- |
| **Algorithm 3.24** Point doubling (LD coordinates) |

INPUT: P = (*X1* : *Y1* : *Z1*) in LD coordinates on

*E/K* : *y² + xy = x3 + ax2 + b.*

OUTPUT: 2*P* = (*X3* : *Y3* : *Z3*) in LD coordinates.

|  |
| --- |
| Fig 4: Algorithm 3.24 point doubling [1]. |

|  |
| --- |
| **Algorithm 3.25** Point addition (LD-affine coordinates) |

INPUT: *P* = (*X1* : *Y1* : *Z1*) in LD coordinates,

Q = (*x2, y2*) in affine coordinates on

*E/K*: *y² + xy = x3 + ax2 + b.*

OUTPUT: *P + Q* = (*X3* : *Y3* : *Z3*) in LD coordinates.

|  |
| --- |
| Fig 5: Algorithm 3.25 point addition [1]. |

### Subtraction

Subtraction is used in certain implementation of the multiplication module [2]. Addition and subtraction are the same operation in F2m. Polynomial subtraction can be achieved by a simple XOR of the two numbers in the polynomials, the same as the addition module.

### Point Scalar Multiplication

To be able to implement the polynomial to perform multiplication, we need to reduce the results given to fit into the registers we chose. For computing *kP*, where *k* is an integer and the value *P* is a point on the curve that is defined over the field *Fq* [1]. This module can see precomputed data that depends only on the value of *P* and not on *k.* Point multiplication is the more sensitive module in area, power performance and speed. This will determine if the whole system will outcome with much faster results or not. We concluded that we use the right-to-left method for the point multiplication. Fig 6. shows to procedure on how the module will work.

|  |
| --- |
| **Algorithm 3.26** Right-to-Left Point Multiplication |

INPUT: *k = (kt – 1, …, k1, k0)2, P ∈ E(Fq).*

OUTPUT: *kP.*

|  |
| --- |
| Fig 6: Algorithm 3.26 right-to-left point multiplication [1]. |

### 

### Point Scalar Multiplication Windowed

This method of multiplication is the module we will be comparing to the multiplication mentioned in section 3.3.3. In this module, this algorithm selects a window size (for our project we separate this into two windows) and computes the values, but uses fewer point additions, which is slower than the method previously shown.

|  |
| --- |
| **Algorithm 2.36** Left-to-Right multiplication with windows of width w |

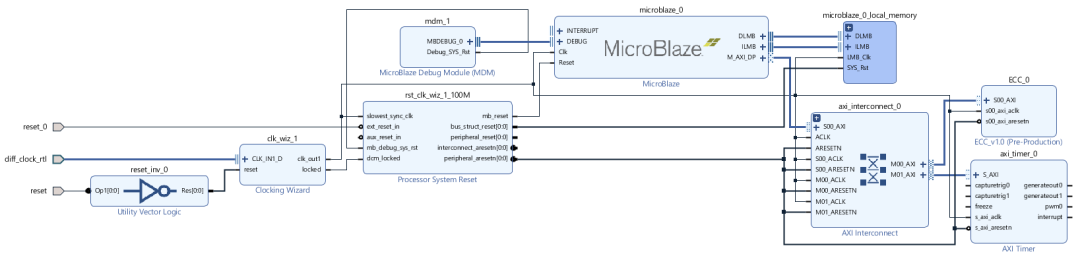
INPUT: Binary polynomials a(z) and b(z) of

degree at most m - 1.

OUTPUT: c(z) = a(z) \* b(z).

|  |
| --- |
| Fig 7: Algorithm 2.36 Multiplication with Windows [1]. |

Fig. 8 shows the MicroBlaze design of the functional block diagram. As seen in the diagram implementation, we focused on using less resources to have a fast and reliable outcome to encrypt and decrypt the polynomial equation. Many aspects of the MicroBlaze was used to configure better performance.



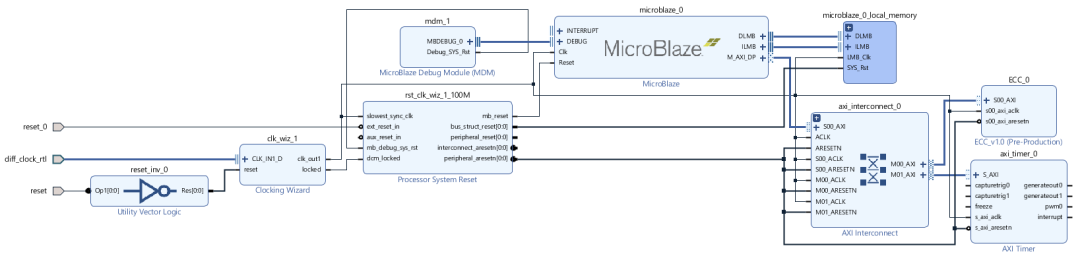
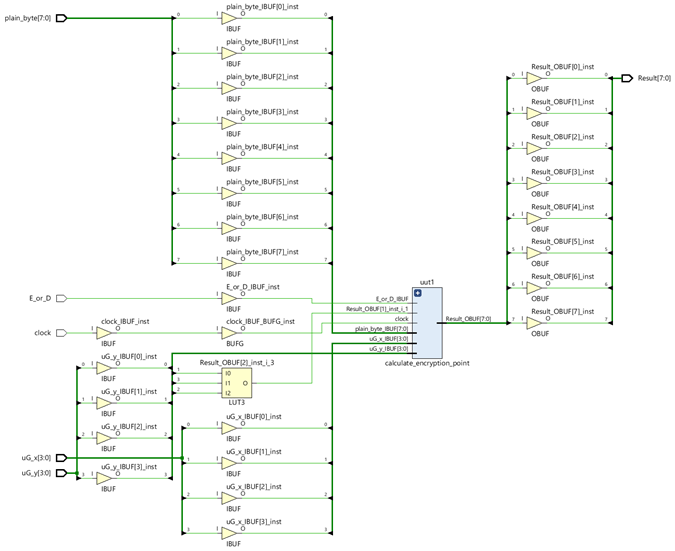


Fig 8: Output Block Diagram

Fig. 9 shows the end results of the schematic that was implemented into our project. With the modules that were chosen, we condensed the schematic down to use less power.



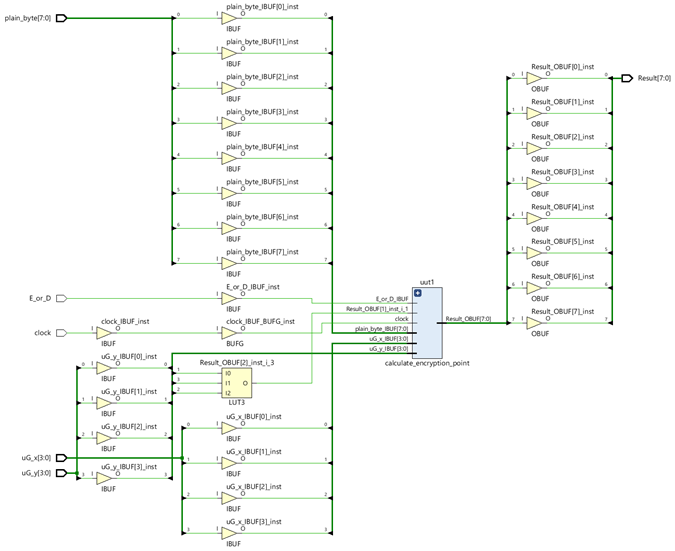


Fig 9: Output Schematic

# Results

## Evaluation

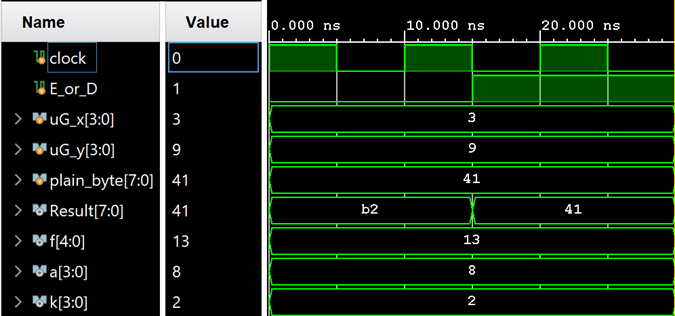


Fig 10: Simulation Results

Looking at the simulation results, we can see that The value of the input E\_or\_D tells us when we are encrypting - a value of 0, or decrypting - a value of 1. At 15 ns, the value of input E\_or\_D changes from 0 to 1, or goes from encrypting to decrypting, which is precisely when the value of the Result changes back to the original message. Inputs uG\_x and uG\_y are the key we start with to encrypt and is used in the scalar multiplication. The input, plain\_byte is the message which is the pure data that is encrypted. The variable, f, is the irreducible polynomial function, of which there are 3 options. In this case, the value refers to z^4 + z + 1. The variable, a, is the curve characteristic, which refers to what kind of curve being used. In this case, the curve being used is z^3. The variable, k, is an integer that is randomly generated every time to be used in generating the keys in the scalar multiplication.

## Comparison

The results of both multiplication methods, point scalar multiplication algorithm and the multiplication windowed algorithm are presented as follows:

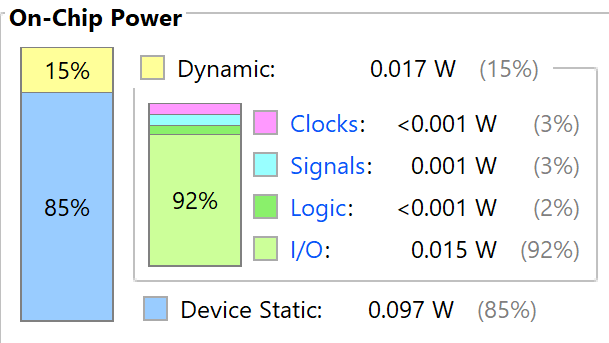




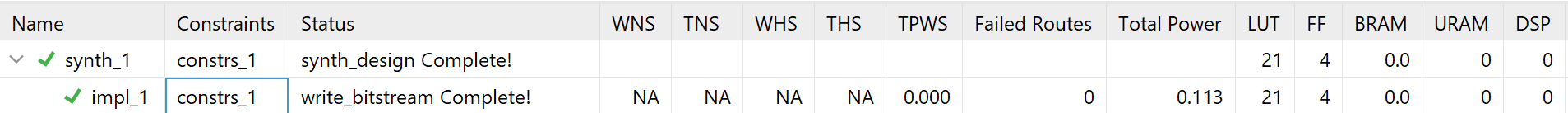
Fig 11: Power Output

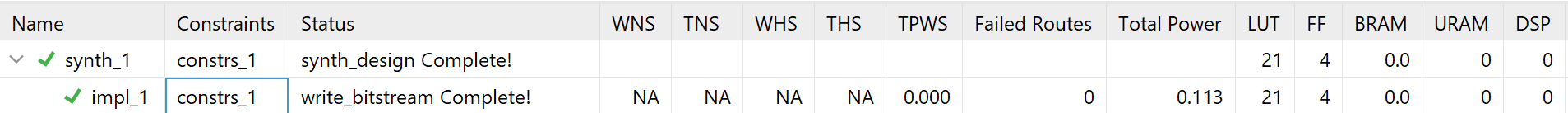
Both algorithms had the same power utilization. Both had total on-chip power equals to 0.11 W.



Fig 12: Slack Time

The pulse width slack was 4.5 ns for both algorithms, which means that the clock pulse width was larger than the minimum required by 4.5 ns as an indication that no violations had occurred in synthesizing nor in implementation.





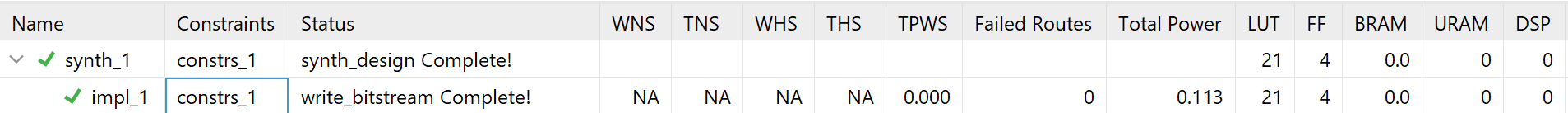


Fig 13: Completion Image

We end up using 21 look up tables and 4 flip-flops for both algorithms.

Apparently, there were no major differences found between the two algorithms concerning the resources usage and the power utilization on the chip, the fact that led us to consider future work, to be discussed in the conclusion section. This is mostly because we have used only 4 bits for the keys and k equals 2. If we were to use a larger binary space and a value for k larger than 2, the differences between both algorithms would be more noticeable.

# Conclusion

The information provided in this paper has armed us with the necessary knowledge about the fundamentals of the binary field for the elliptic curve cryptography. With the general issues related to ECC implementation discussed before, we now came to the conclusion that the point scalar multiplication and the windowed multiplication algorithms are the same in terms of clock cycles and power utilization as long as we are using small keys. In the long run with just one value for encryption from the sender and decryption from the receiver, it can result in a fast implementation, but it is difficult to gather all the information that is being shared.

## Future Work

To further our work, many different adaptations and experiments have been left for the future due to lack of time. Future work concerns applying more evaluation metrics and deeper analysis of software based elliptic curve cryptography and the benefits of using the hardware based path over the software one.

# References

1. Hankerson, D., Menezes, A. and Vanstone, S., n.d. *Guide To Elliptic Curve Cryptography*. [Online] Citeseerx.ist.psu.edu. Available at: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.394.3037&rep=rep1&type=pdf> [Accessed 18 November 2020].
2. A. MS, "Elliptic Curve Cryptography - An Implementation Tutorial - dkrypt.com", *Sites.google.com*. [Online]. Available: https://sites.google.com/site/csnewandold/home/ecc#TOC-Conclusion. [Accessed: 18- Nov- 2020]
3. D. Susantio and I. Muchtadi-Alamsyah, "Implementation of Elliptic Curve Cryptography in Binary Field", *Iopscience.iop.org*, 2016. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/710/1/012022/pdf. [Accessed: 19- Nov- 2020]
4. M. Kashif, "A Hardware Eﬃcient Elliptic CurveAccelerator for FPGA BasedCryptographic Applications", *Researchgate.net*, 2020. [Online]. Available: https://www.researchgate.net/publication/342067400\_A\_Hardware\_Efficient\_Elliptic\_Curve\_Accelerator\_for\_FPGA\_Based\_Cryptographic\_Applications. [Accessed: 19- Nov- 2020]
5. J. Huang, H. Li and P. Sweany, "An FPGA Implementation of Elliptic Curve Cryptography for Future Secure Web Transaction", *Citeseerx.ist.psu.edu*. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.1823&rep=rep1&type=pdf. [Accessed: 25- Nov- 2020]
6. N. Smart, "A Comparison of Different Finite Fields for Elliptic Curve Cryptosystems", *Science Direct Assets*, 2000. [Online]. Available: https://pdf.sciencedirectassets.com/271503/1-s2.0-S0898122100X01204/1-s2.0-S089812210100133X/main.pdf. [Accessed: 25- Nov- 2020]
7. "THE PROPOSED SYSTEM OF IMPLEMENTING ELLIPTIC CURVE CRYPTOGRAPHY USING VERILOG HDL", *Shodhganga.inflibnet.ac.in*, 2020. [Online]. Available: https://shodhganga.inflibnet.ac.in/bitstream/10603/27232/8/08\_chapter3.pdf. [Accessed: 25- Nov- 2020]
8. J. Bauer, "Elliptic Curve Cryptography Tutorial", *Johannes-bauer.com*, 2020. [Online]. Available: https://www.johannes-bauer.com/compsci/ecc/. [Accessed: 25- Nov- 2020]
9. N. Sullivan, "A (Relatively Easy To Understand) Primer on Elliptic Curve Cryptography", *The Cloudflare Blog*, 2013. [Online]. Available: https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/. [Accessed: 25- Nov- 2020]
10. E. Wenger and M. Hutter, "Exploring the Design Space of Prime Field vs.Binary Field ECC-Hardware Implementations", *ResearchGate*, 2011. [Online]. Available: https://www.researchgate.net/publication/262246446\_Exploring\_the\_Design\_Space\_of\_Prime\_Field\_vs\_Binary\_Field\_ECC-Hardware\_Implementations. [Accessed: 27- Nov- 20
11. A. Mahboob, "Efficient Hardware and Software Implementation of Elliptic Curve Cryptography", *ResearchGate*, 2004. [Online]. Available: https://www.researchgate.net/publication/216485702\_EFFICIENT\_HARDWARE\_AND\_SOFTWARE\_IMPLEMENTATION\_OF\_ELLIPTIC\_CURVE\_CRYPTOGRAPHY. [Accessed: 08- Dec- 2020