Implementation of Elliptic Curve Crypto Processor and Its Performance Analysis

Asst. Prof.Y.A.Suryawanshi, Neha Trimbak Khadgi M.tech(VLSI)
Department of Electronics Engineering
Yeshwantrao Chavan College of Engineering
Nagpur (M.S.), India
yogesh_surya8@rediffmail.com, nehakhadgi@gmail.com

ABSTRACT: **ECDSA stands for "Elliptic Curve Digital Signature Algorithm", it's used to create a digital signature of data (a file for example) in order to allow you to verify its authenticity without compromising its security. This paper presents the architecture of finite field multiplication. The proposed multiplier is hybrid Karatsuba multiplier used in this processor. For multiplicative inverse we choose the Itoh-Tsujii Algorithm (ITA). This work presents the design of high performance elliptic curve crypto processor (ECCP) for an elliptic curve over the finite field GF (2^233). The curve which we choose is the standard curve for the digital signature. The processor is synthesized for Xilinx FPGA.**

Keywords: **Elliptic Curve Cryptography (ECC), Field programmable gate ARRAY (FPGA), ITA,Elliptic Curve Crypto Processor(ECCP).**

*****

1. INTRODUCTION

In this era communications using world wide web has increased. The data which is transmitted by using wired or wireless network. Some of the transactions have critical data which need to be confidential and users authenticated. Hence it requires some security. Cryptography is used to provide secure communications. An authorized user is identified by a cryptographic key, a user having the correct key will be able to access the transmitted information while the fake users or the other people will not have rights to use the information.

There are two types of crypto graphic algorithms, symmetric key and asymmetric key algorithms. Symmetric key cryptography contains both encryption and decryption. It is most widely used method because this method is fast and simple. But it can be used only when the two parties are agreed for the secret keys. So it is difficult because exchanging the keys is not easy for the users.

1.1PRINCIPLE OF ECDSA

ECDSA stands for "Elliptic Curve Digital Signature Algorithm", it's used to create a digital signature of data (a file for example) in order to allow you to verify its authenticity without compromising its security. The ECDSA algorithm is basically all about mathematics. You have a mathematical equation which draws a curve on a graph, and you choose a random point on that curve and consider that your point of origin. Then you generate a random number, this is your private key, you do some mathematical equation using that random number and that "point of origin" and you get a second point on the curve, that's your public key. When you want to sign a file, you will use this private key (the random number) with a hash of the file (a unique number to represent the file) into an equation and that will give you your signature. The signature itself is divided into two parts, called ***R*** and ***S***. In order to verify that the signature is correct, you only need the public key (that point on the curve that was generated using the private key) and you put that into another equation with one part of the signature (***S***), and if it was signed correctly using the private key, it will give you the other part of the signature (***R***). So to make it short, a signature consists of two numbers, ***R*** and ***S***, and you use a private key to generate ***R*** and ***S***, and if a mathematical equation using the public key and ***S*** gives you ***R***, then the signature is valid. There is no way to know the private key or to create a signature using only the public key.

ECDSA uses only integer mathematics, there are no floating points (this means possible values are 1, 2, 3, etc.. but not 1.5..), also, the range of the numbers is bound by how many bits are used in the signature (more bits means higher numbers, means more security as it becomes harder to 'guess' the critical numbers used in the equation), as you should know, computers use 'bits' to represent data, a bit is a 'digit' in binary notation (0 and 1) and 8 bits represent one byte. Every time you add one bit, the maximum number that can be represented doubles, with 4 bits you can represent values 0 to 15 (for a total of 16 possible values), with 5 bits, you can represent 32 values, with 6 bits, you can represent 64 values, etc.. one byte (8 bits) can represent 256 values, and 32 bits can represent 4294967296 values (4 Giga).. Usually ECDSA will use 160 bits total, so that makes well, a very huge number with 49 digits in it.

ECDSA is used with a SHA1 cryptographic hash of the message to sign (the file). A hash is simply another mathematical equation that you apply on every byte of data which will give you a number that is unique to your data. Like for example, the sum of the values of all bytes may be considered a very dumb hash function. So if anything changes in the message (the file) then the hash will be completely different. In the case of the SHA1 hash algorithm, it will always be 20 bytes (160 bits). It's very useful to validate that a file has not been modified or corrupted, you get the 20 bytes hash for a file of any size, and you can easily recalculate that hash to make sure it matches. What ECDSA signs is actually that hash, so if the data changes, the hash changes, and the signature isn't valid anymore.

Elliptic Curve cryptography is based on an equation of the form :

$$y^2 = (x^3 + a * x + b) \bmod p$$

First thing you notice is that there is a modulo and that the ‘y‘ is a square. This means that for any x coordinate, you will have two values of y and that the curve is symmetric on the *X axis*. The modulo is a prime number and makes sure that all the values are within our range of 160 bits and it allows the use of "modular square root" and "modular multiplicative inverse" mathematics which make calculating stuff easier (I think). Since we have a modulo (p) , it means that the possible values of y^2 are between 0 and p-1, which gives us p total possible values. However, since we are dealing with integers, only a smaller subset of those values will be a "perfect square" (the square value of two integers), which gives us N possible points on the curve where $N < p$ (N being the number of perfect squares between 0 and p). Since each x will yield two points (positive and negative values of the square-root of y^2), this means that there are $N/2$ possible ‘x‘ coordinates that are valid and that give a point on the curve. So this elliptic curve has a finite number of points on it, and it's all because of the integer calculations and the modulus. Another thing you need to know about Elliptic curves, is the notion of "point addition". It is defined as adding one point P to another point Q will lead to a point S such that if you draw a line from P to Q, it will intersect the curve on a third point R which is the negative value of S (remember that the curve is symmetric on the *X axis*). In this case, we define $R = -S$ to represent the symmetrical point of R on the *X axis*.

There is also point multiplication where $k*P$ is the addition of the point P itself to k times. One particularity of the point multiplication is that if you have a point $R = k*P$, where you know R and you know P, there is no way to find out what the value of ‘k‘ is. Since there is no point subtraction or point division, you cannot just resolve $k = R/P$. Also, since you could be doing millions of point additions, you will just end up on another point on the curve, and you would have no way of knowing "how" you got there. You can't reverse this operation, and you can't find the value ‘k‘ which was multiplied with your point P to give you the resulting point R.

This thing where you can't find the multiplicand even when you know the original and destination points is the whole basis of the security behind the ECDSA algorithm, and the principle is called a "trap door function".

For ECDSA, you first need to know your curve parameters, those are a, b, p, N and G. You already know that ‘a‘ and ‘b‘ are the parameters of the curve function ($y^2 = x^3 + ax + b$), that ‘p‘ is the prime modulus, and that ‘N‘ is the number of points of the curve, but there is also ‘G‘ that is needed for ECDSA, and it represents a ‘reference point' or a point of origin if you prefer. Those curve parameters are important and without knowing them, you obviously can't sign or verify a signature. Yes, verifying a signature isn't just about knowing the public key, you also need to know the curve parameters for which this public key is derived from.

So first of all, you will have a private and a public key.. the private key is a random number (of 20 bytes) that is generated, and the public key is a point on the curve generated from the point multiplication of G with the private key. We set ‘dA‘ as the private key (random number) and ‘Qa‘ as the public key (a point), so we have : $Qa = dA * G$ (where G is the point of reference in the curve parameters).

First, you need to know that the signature is 40 bytes and is represented by two values of 20 bytes each, the first one is called R and the second one is called S. so the pair (R, S) together is your ECDSA signature. First you must generate a random value ‘k‘ (of 20 byes), and use point multiplication to calculate the point $P=k*G$. That point's x value will represent ‘R‘. Since the point on the curve P is represented by its (x, y) coordinates (each being 20 bytes long), you only need the ‘x‘ value (20 bytes) for the signature, and that value will be called ‘R‘. Now all you need is the ‘S‘ value.

To calculate S, you must make a SHA1 hash of the message, this gives you a 20 bytes value that you will consider as a very huge integer number and we'll call it ‘z‘. Now you can calculate S using the equation :

$$S = k^{-1} (z + dA * R) \bmod p$$

Note here the k^{-1} which is the ‘*modular multiplicative inverse*‘ of k, it's basically the inverse of k, but since we are dealing with integer numbers, then that's not possible, so it's a number such that $(k^{-1} * k) \bmod p$ is equal to 1. And k is the random number used to generate R, z is the hash of the message to sign, dA is the private key and R is the ***x*** coordinate of ***k*G*** (where ***G*** is the point of origin of the curve parameters).

Now that you have your signature, you want to verify it, you only need the public key (and curve parameters of course) to do that. You use this equation to calculate a point ***P*** :

P= S^-1*z*G + S^-1 * R * Qa

If the ***x*** coordinate of the point ***P*** is equal to ***R***, that means that the signature is valid, otherwise it's not.

Now we are going to verify it and this require some mathematics to verify :

We have :

P = S^-1*z*G + S^-1 * R *Qa

but ***Qa = dA*G***, so:

P = S^-1*z*G + S^-1 * R * dA*G = S^-1 (z + dA* R) * G

But the ***x*** coordinate of ***P*** must match ***R*** and ***R*** is the ***x*** coordinate of ***k * G***, which means that :

k*G = S^-1 (z + dA * R) *G

we can simplify by removing ***G*** which gives us :

k = S^-1(z + dA * R)

by inverting ***k*** and ***S***, we get :

S = k^-1 (z + dA *R)

and that is the equation used to generate the signature.. so it matches, and that is the reason why you can verify the signature with it.

You can note that you need both '***k***' (random number) and '***dA***' (the private key) in order to calculate ***S***, but you only need ***R*** and ***Qa*** (public key) to validate the signature. And since ***R=k*G*** and ***Qa = dA*G*** and because of the trap door function in the ECDSA point multiplication (explained above), we cannot calculate ***dA*** or ***k*** from knowing ***Qa*** and ***R***, this makes the ECDSA algorithm secure, there is no way of finding the private keys, and there is no way of faking a signature without knowing the private key.

The ECDSA algorithm is used everywhere and has not been cracked and it is a vital part of most of today's security.

2. Implementation of Elliptic Curve Crypto Processor

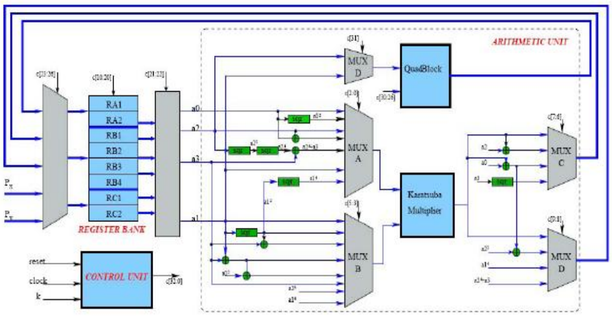

Figure1. Block Diagram of Elliptic Curve Crypto Processor.

3. Description of the blocks used in Elliptic Curve Crypto Processor

The quad based ITA architecture was shown to have best performance compared to other architectures, therefore we design our 2^n circuit based ITA as a generalization of this architecture.

We now give the brief description of the 2^n based ITA architecture which is shown in block diagram. This design presents the high performance ECCP for the curve over finite field GF(2^233). The finite field multiplier used is of combinational type and follows the Karatsuba Algorithm. The Quad block is used to raise the power of its input to any power of 2. Buffers are used to latch the output of the multiplier and the quad block respectively. Control signal is used in each clock cycle to latch the output of either the multiplier or the power block but not both. Any intermediate result that is required in a later step is stored in a register bank. A control block is used to generate all the control signals that drive a Finite State Machine(FSM).

3.1 Register Bank

The register file contains eight registers each of size 233 bits. The registers are used to store the results of the computations done at every clock cycle.

Table 1. Utility of Registers in Register Bank

Register	Description
RA1	1.During initialization it is loaded with Px.

	2.Stores the x co ordinate of the result. 3.Also used for temporary storage.
RA2	Stores Px.
RB1	1.During initialization it is loaded with Py. 2.Stores the y co ordinate of the result. 3.Also used for temporary storage.
RB2	Stores Py
RB3	Used for temporary storage.
RB4	Stores the curve constant b.
RC1	1.During initialization it is set to 1. 2.Store z co ordinate of the projective result.
RC2	Used for temporary storage.

3.2 FINITE FIELD ARITHMETIC UNIT

The arithmetic unit is build using finite field arithmetic circuit. The AU has five inputs (A0 to A3 and Qin) and 3 outputs (C0,C1 and Qout). The main components of AU is quad block and a multiplier. The multiplier is based on the hybrid karatsuba algorithm. The quad block consists of 14 cascaded quad circuits and is capable of generating the output. The quad block is used only for inversion operation. The AU has several adders and squarer circuits.

233 Bit Hybrid Karatsuba Multiplier.

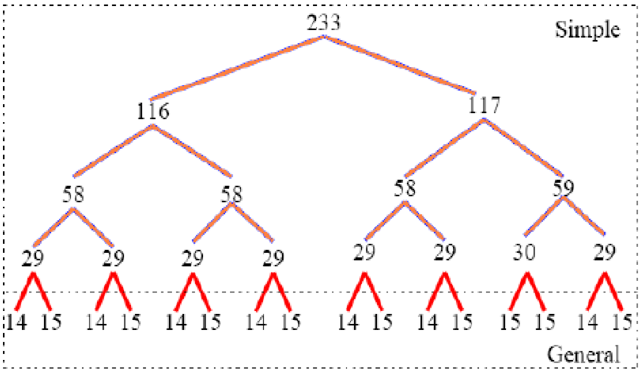

Figure 2. Architecture of 233 Bit Hybrid Karatsuba Multiplier.

3.3 CONTROL UNIT

At every clock cycle the control unit produces a control word. The control word signals control the flow of data and also decide the operations performed on the data. There are 33 control signal generated by the control word.

4.RESULT OF ALU BLOCK USED IN ELLIPTIC CURVE CRYPTO PROCESSOR.

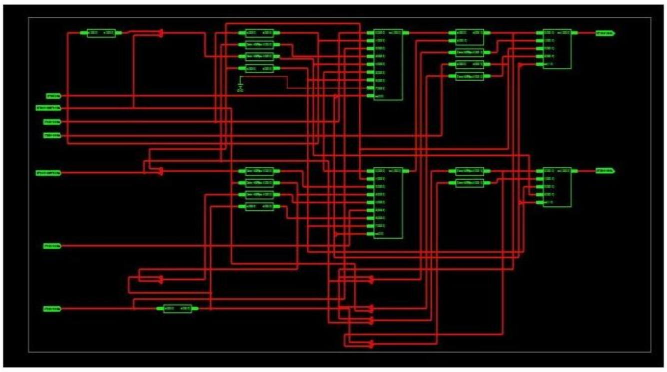

Figure 3.RTL view of Arithmetic Unit.

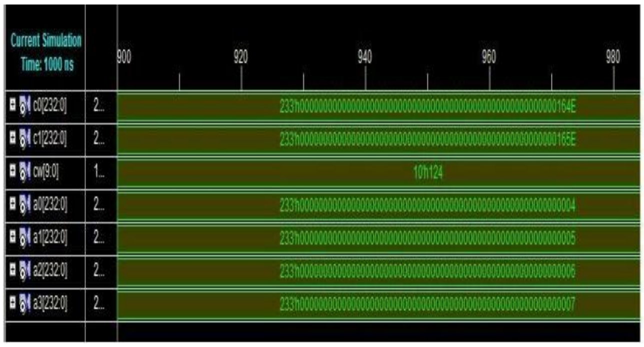
Current Simulation Time: 1000 ns
900
920
940
960
980
c0[232:0]
2...
233'h0000000000000000000000000000000000000000000000000000000164E
c1[232:0]
2...
233'h0000000000000000000000000000000000000000000000000000000165E
cw[9:0]
1...
10'h124
a0[232:0]
2...
233'h00000000000000000000000000000000000000000000000000000000004
a1[232:0]
2...
233'h00000000000000000000000000000000000000000000000000000000005
a2[232:0]
2...
233'h00000000000000000000000000000000000000000000000000000000006
a3[232:0]
2...
233'h00000000000000000000000000000000000000000000000000000000007

Figure 4. Output of ALU.

Result of Complete Processor with Hybrid Karatsuba Multiplier.

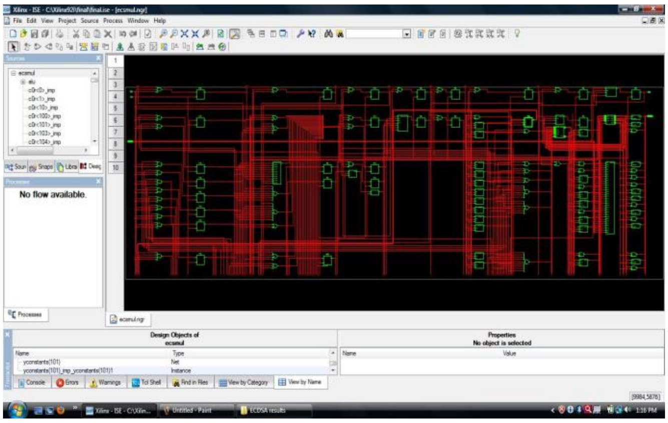

Figure 5. RTL View of ECCP Processor using Hybrid Karatsuba Multiplier

.

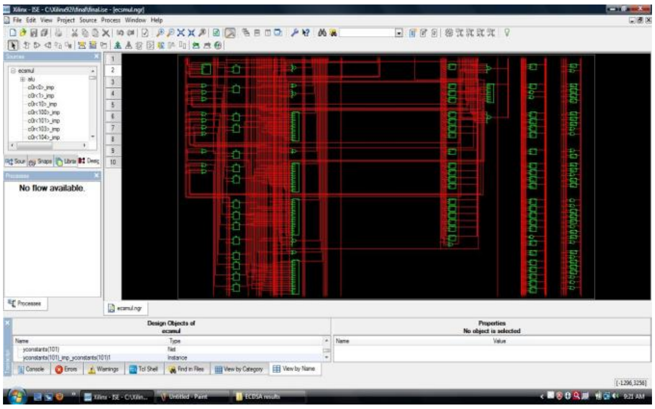

Figure 6. RTL View Of ECCP Processor.

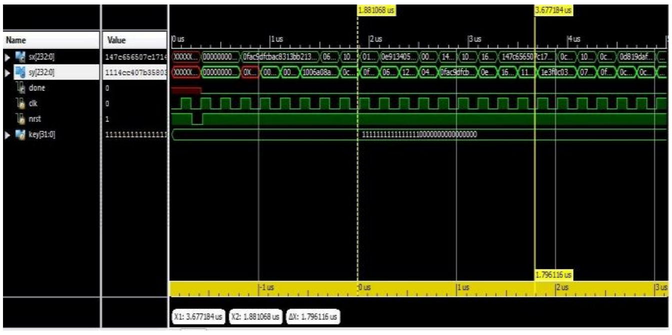

Figure 7. Output of the ECCP Processor.

CONCLUSION

The arithmetic unit is used in a Elliptic Curve Crypto Processor to compute the scalar product *kP*. This design has more efficient FPGA utilization .High speed is obtained by implementation of quad and squarer circuits. The Quad ITA algorithm is used to reduce computation time. We use carry look ahead adder in ALU because it is faster than any other adder. It will generate and propagate the carry so that computation time requirement is less.
Also the area and power requirements are less in this design. The most important factor contributing the performance is the finite field multiplication and finite field inversion.

FUTURE WORK

I design ALU and ECCP processor completely for the curve over the finite field GF(2^233) . In ALU hybrid karatsuba multiplier is used so I will design modified karatsuba multiplier and compare the results of both multipliers. Then I will decide which multiplier gives the best results. I will calculate the time requirement, area and power of the ECCP Processor by using both multipliers in ALU unit.

RELATED WORK

Revisiting the Itoh-Tsujii Inversion Algorithm for FPGA Platforms [1]paper generates components of several cryptographic implementations such as elliptic curve cryptography. For binary fields generated by irreducible trinomials, this paper process a modified ITA algorithm for efficient implementations on field programmable gate array(FPGA) platforms.

Theoretical Modelling of the Itoh-Tsujii Inversion Algorithm for Enhanced Performance on k- LUT based FPGAs [5],this paper maximizing the performance of the ITA algorithm on FPGAs requires tunning of several design parameters. This is often time consuming and difficult. This paper presents a theoretical model for the ITA foe any Galois field and k-input LUT based FPGA. Such a model would aid a hardware designer to select the ideal design parameters quickly.

A high performance elliptic curve cryptographic processor for general curves over GF(p) based on a systolic arithmetic unit [4]this paper brief presents high performance elliptic curve cryptographic processor for general curves over GF(p), which features a systolic arithmetic unit. This paper propose a new unified systolic array that efficiently implements addition.substraction and multiplication.

The Elliptic Curve digital signature algorithm [9]this paper shows that ECDSA algorithm is the elliptic curve analogue of the digital Signature Algorithm. It was accepted in 1999 as an ANSI standard, and was accepted in 2000 as IEEE and NIST standards. Unlike the ordinary discrete logarithm problem and the integer factorization problem, no sub exponential time algorithm is known for the elliptic curve discrete logarithm problem.

REFERENCES

[1] Chester Rebeiro, Sujoy Sinha “Revisiting The Itoh-Tsujii Inversion Algorithm for FPGA Platforms” IEEE transactions on VLSI Systems, VOL. 19, NO. 8, August 2011.

[2] Surabhi Mahajan“Security and Privacy in VANET” International Journal of Computer Applications volume1,February 2010.

[3] Kuldeep Singh “Implementation of Elliptic Curve Digital Signature Algorithm” International Journal of Computer Applications, volume2 No.2, May 2010.

[4] Gang Chen and Guoqiang Bai “A High Performance Elliptic Curve Cryptographic Processor for General Curves over GF (p) Based On a Systolic Arithmetic Unit.” IEEE transactions on circuit and systems-II: Express Briefs, VOL 54, No.5, May 2007.

[5] Sujoy Sinha Roy and Chester Rebeiro "Theoretical Modelling of the Itoh-Tsujii Inversion Algorithm for Enhanced Performance on k- LUT based FPGAs."

[6] Debdeep mukhopadhay " High Performance Elliptic Curve Crypto Processor for FPGA Platforms" NTT Labs, Japan, June 2010.

[7] Qiucheng Deng, Xuefei Bai, Li Guo and Yao wang " Hardware Implementation of Multiplicative Inversion in GF(2^m)",Department of Electronics Science and Technology Of China.

[8] Mohen Machhout, Zied Guitouni, Kholdoun Torki and Rached "Coupled FPGA/ASIC Implementation Of Elliptic Curve Crypto Processor", International Journal Of Network Security and its Applications (IJNSA), Volume 2, Number 2, April 2010.

[9] Don Johnson, Alfred Menezes and Scott Vanstone " The Elliptic Curve Digital Signature Algorithm (ECDSA)", Certicom Research, Canada.

[10] Felipe Tellez and Jorge Ortiz " Behaviour of Elliptic Curve Cryptosystems for the Wormhole Intrusion in Manet : A Survey and Analysis", IJCSNS International Journal of Computer Science and Network Security, VOL.11 No.9, September 2011.