Design and Evaluation of FPGA Based Hardware Accelerator for Elliptic Curve Cryptography Scalar Multiplication

Kapil A. Gwalani, Omar Elkeelany
Department of Electrical and Computer Engineering
Tennessee Technological University
Cookeville, Tennessee, USA.
Kagwalani21@tntech.edu OElkeelany@tntech.edu

Abstract: Embedded systems find applications in fields such as defense, communications, industrial automation and many more. For majority of these applications, security is a vital issue. Over the last decade, security has become the primary concern when discussing e-commerce. The rapid development in the field of information technology has led to the increase in need of techniques capable of providing security. Cryptography plays an important role in providing data security. Until recently, symmetric key encryption schemes were used for a majority of these applications. Now however, asymmetric key encryption schemes such as Elliptic curve cryptography are gaining popularity as they require less computational power and memory and are still capable of providing equivalent security when compared to their counterparts. Elliptic curve cryptography was first introduced in 1985 and has always been around since. Scalar or point multiplication in elliptic curve cryptography has been a topic of research interest. Improving the performance of scalar multiplication can improve the overall performance of elliptic curve cryptography. One popular method to improve scalar multiplication is by means of hardware accelerators.

The authors of this paper have implemented scalar multiplication, the most time consuming operation in elliptic curve cryptography using binary non-adjacent form algorithm. The results of the software implementation have been presented in section- 4. Methodology to improve the performance of the scalar multiplication by use of hardware accelerators has also been presented in this paper.

Key Words: Binary Non-adjacent Form, ECC, Prime Field, System on Programmable Chip.

1. Introduction

Cryptography is defined as the art of encoding data using a key so that only authorized users can decode and access the data. Cryptography can be classified into two categories, public key cryptography and private key cryptography. Private Key cryptography, also known as symmetric key cryptography uses a single key for encryption and decryption. Examples of such encryption scheme are, Advanced Encryption Standard (AES), Data Encryption Standard (DES), and Triple DES. Public key cryptography also known as asymmetric key encryption, on the other hand uses two keys, one for encryption and other for decryption. Examples of this cryptographic scheme are RSA, Diffie–Hellman and Elliptic curve cryptography (ECC).

Symmetric key algorithms are easy to implement but there is always a possibility of the key being intercepted. Asymmetric key algorithms on the other hand, are immune to this attack and thus provide better security than their counterparts. However, they have the disadvantage of being complex and so reduce the overall performance of the embedded system. Hence the use of asymmetric algorithms in cryptography is a research challenge. Designers of embedded systems are faced with making a decision between providing improved security at the cost of reduced performance or vice versa.

Elliptic curve cryptographic systems are known to provide better security per bit than RSA; at the same time they can be feasibly implemented on embedded systems at higher speeds and less memory requirements. As a result they are now being recognized as a true alternative not only to RSA but also to symmetric key systems such as Triple data encryption standard and advanced encryption standard.

To provide higher security, key sizes are usually in the range of hundred's of bits. The longer the key length the more secure the system becomes. This fact also makes cryptographic procedures slow in software.

ECC relies on the elliptic curve discrete logarithmic problem (ECDLP) to provide security. ECDLP is to find k in the equation Q=kP, where Q and P are points on the elliptic curve. The critical operation k.P is called point multiplication. The overall performance of ECC can be improved drastically if the operation of point multiplication can be accelerated. The aim of this work is, to speed up the operation of point multiplication through the use of a hardware accelerator.

Elliptic Curve Cryptosystems can be implemented over two fields, the prime field GF (p) and the binary field GF (2^m). GF (p) contains a prime ‘p’ number of elements. The elements of this field are the integers modulo p, and the field arithmetic is implemented in terms of the arithmetic of integers modulo p. GF(2^m), contains 2^m elements where m is degree of the field. Elements in GF (2^m) can be considered as polynomials of degree m-1. The coefficients of these polynomials can be 0 or 1 only. This work focuses on speeding up the performance of point multiplication over the prime field.

The paper has been divided into five sections. Section two covers the related work. This section briefly discusses the previous work done in this field. Section three presents the mathematics involved with ECC over prime field, section four illustrates the hardware design and implementation. Section five presents the results of the software implementation. Section five also presents the initial results of hardware implementation. Future work to be done has also been discussed in section five.

2. Related Work

There have been several implementations of hardware accelerators for elliptic curve cryptography but most of them focus on the binary field.

Siddika Berna et al. [2], in their paper have presented a hardware implementation of ECC over GF (p). The authors of [2] used Montgomery modular multiplication in their hardware implementation. The algorithm used for point multiplication is the Double and Add algorithm. The target hardware platform in [2] was the Xilinx V1000E-BG-560-8 (Virtex E) FPGA. The current work uses Binary non-adjacent form algorithm for the point multiplication and the target hardware platform is the Altera Cyclone II FPGA.

Satoh and Takano [3] present an ASIC elliptic curve processor which supports both the binary and prime fields. The authors of [3] have used a 0.13 μm CMOS ASIC as their target platform and Montgomery multiplier in their hardware implementation. The current implementation concentrates only on the prime field; algorithm used for point multiplication is the binary non-adjacent form algorithm. The authors of the current work use ripple adders in their hardware implementation.

Orlando and Paar [4], in their paper presented elliptic curve processor architecture for computing point multiplication using a high-radix Montgomery multiplier. The authors of [4] use double and add algorithm for point multiplication. The target hardware platform used by the authors in [4] was Xilinx XCV1000E-8-BG680 FPGA.

As seen, a majority of the hardware accelerators are implemented using the double and add algorithm and using Montgomery multiplier hardware architecture. The current work utilizes the Binary Non- Adjacent Form algorithm for point multiplication. The hardware implementation is carried out using a series of ripple adders

Applications of elliptic curve cryptography are many; recently a lot of focus has been placed particularly in the field of wireless communications and e-commerce. Jang Ho and Jou Yang [10], in their paper discussed the need for improved security over the wireless mobile network. In their paper, the authors have made a comparison amongst the several secure connection mechanisms in mobile commerce. A comparison between wireless transport layer security (WTLS) and the KiloByte SSL (KSSL) was made. Even though WTLS came up short, there is still work in progress on ways to improve it.

Po - Hsian Huang [11] in his paper has described the possible application of Elliptic curve cryptography in IPv6. The paper discusses the advantage of employing ECC over RSA in IPv6. Andrew Georgiades et. al. [12], in their paper have focused on security issues pertaining to route optimizations, the solution proposed by them – the trinity protocol utilizes the elliptic curve cryptography for providing security. Some of the other work also includes use of elliptic curve diffie hellman key exchange protocol (ECDH) to establish a secure connection between a base station and a sensor node [13].

3. Mathematics behind Elliptic Curves on Prime Field F_p

The operations which can be performed on points on an elliptic curve are, point addition, subtraction and doubling. All the operations are carried out using modular arithmetic which means that all the elements of the finite field are integers between 0 and p-1 [5], where 'p' is the prime modulus. It's recommended to choose p between 112-521 bits [2]. The elliptic curve over prime field is defined by the equation below:

$$y^2 \bmod p = x^3 + ax + b \bmod p \qquad (1)$$

where p is the prime modulus defined for the finite field F_p. Parameters 'a' and 'b' define the elliptic curve (1). The above mentioned three parameters along with two other parameters 'n' and 'G' constitute the domain parameters of the elliptic curve defined by equation (1). The order of the elliptic curve is denoted by 'n'. 'G' denotes the generator point, a point on the elliptic curve which is chosen for performing cryptographic operations [5].

3.1 Point Addition

Consider two distinct points A and B on an elliptic curve where A and B have the co-ordinates (x_a, y_a) and (x_b, y_b) respectively and a third point C (x_c, y_c) which is the result of adding A and B then,

$$x_c = (m^2 - x_a - x_b) \bmod p \qquad (2)$$
$$y_c = (m(x_a-x_c)-y_a) \bmod p \qquad (3)$$
$$m = (y_a-y_b / x_a - x_b) \bmod p \qquad (4)$$

Where, m is the slope of the line through the points A and B.

3.2 Point Subtraction

Point subtraction utilizes the same equations as that of point addition, consider the points A and B again on an elliptic curve where, A and B have the co-ordinates (x_a, y_a) and (x_b, y_b) respectively and a third point C (x_c, y_c) which is the result of subtracting A from B then,

$$A - B = A + (-B) \text{ where } -B = (x_b, -y_b) \qquad (5)$$

3.3 Point Doubling

Consider the point A (x_a, y_a), where y_a is not equal to zero, then C = 2A is given by the following equations

$$x_c = (m^2 - 2x_a) \bmod p \qquad (6)$$
$$y_c = (m(x_a-x_c)-y_a) \bmod p \qquad (7)$$
$$m = ((3x_a^{\,2} + a)/ 2\, y_a) \bmod p \qquad (8)$$

If y_a is zero then m is zero as a result C = O, the point at infinity. The slope equation (8), above utilizes the curve parameter 'a'.

4. Elliptic Curve Encryption

The mathematics of elliptic curves can be used for encrypting a message as well as decrypting it. Assume there are two users, user A - Alice and user B - Bob who want to share top secret data. The process of encryption is the same, users convert plaintext to cipher text by means of a key and the send this to the other users involved in the communication. The difference here is how the key is calculated at either end and how the message is decoded.

The first step to encrypting data is deciding on the generator point 'G' [8]. The point is chosen such that the smallest value of n – order of the curve for which n.G = 0 is still a very large prime number [8]. The elliptic curve group and the generator point G are made public. Now, if the two users want to communicate with each other, they first select their private key, this is then used to generate the public key. The relation between public key and private key can be understood from the equation below.

Public Key = Private Key * Generator point 'G' (9)
The public key is a point on the elliptic curve; the private key is a random number.

The algorithm for encryption is described in below.

Algorithm - Encryption

- **User A – Alice first select a random generator point (x, y) lying on the elliptic curve.**
- **Message (M) to be encrypted is coded on to an elliptic curve point P_m = (xm, ym).**
- **Alice selects a random private key 'n_A' and then computes the public key as:**

$$\mathbf{P_A = n_A\,(x,y)} \qquad (10)$$

- **To encrypt her message, Alice uses her private key and Bobs (user B) public key.**
- **The encrypted message denoted by Cm is created as follows:**

$$\mathbf{Cm = \{\, P_A,\ (P_m + n_A \,.\, P_B\,)\}} \qquad (11)$$

P_B is the public key of Bob – user B.

As it can be seen from the above algorithm, point multiplication plays a major role during the encryption process. The same hold during decryption too.

The encrypted message is then communicated to the receiver. The receiver - bob then decrypts the message using the decryption mechanism. The algorithm for decryption at the receivers end is as follows:

Algorithm - Decryption

- **When Bob receives the encrypted message, he first multiplies the public key of Alice, which happens to be the first point in the encrypted message with his private key $n_{B.}$**
- **The result of this is then subtracted from the second point the cipher text in equation 11.**
- **This gives him the original message $P_{m.}$**

The advantage here is, Bob's private key is only known to him and not anyone else and therefore only bob can extract the original message by subtracting the product of his private key and Alice's public key with the second point.

Elliptic curve cryptography can also used for key generation and digital signatures as well. ECC can also be used for the verification of digital signatures. In all of the above ECC primitives, scalar multiplication is the core operation. The implementation of this operation is discussed in the next section.

5. Design and Implementation

There are two phases of implementation for the current work, software and hardware. Software implementation of point multiplication was carried out on Altera Cyclone II 2C35C6 FPGA. The results achieved have been discussed and compared with [1] in the next section. The second phase of implementation is performed on hardware; this is done to improve the results obtained in software.

5.1 Software Implementation

The software implementation was carried out on Nios II processor.

The algorithm used for the software implementation of point multiplication is binary non-adjacent form (NAF) and the co-ordinate system used in the current implementation is affine co-ordinate system.

5.1.1 Binary Non – Adjacent Form Algorithm

Non adjacent form, as the name suggests is, to ensure that no two non-zero numbers are adjacent to each other. This algorithm is used to better represent the scalar k. When this algorithm is used k is of the form:

$$\mathbf{k} = \sum_{i=0}^{m-1} \mathbf{k_i\ 2^{\,i}} \qquad (12)$$

k_i can have a value of {-1, 0,1}. Every positive integer has an exclusive non adjacent form. It's also represented as NAF (k).The algorithm utilizes both addition and subtraction to compute the product k.P.

The algorithm used in the current work is presented in figure1.

Inputs: P and k where, P is a point on the elliptic curve having coordinates (x, y) and k is a scalar.
Output Q = k.P; product of k and P.

```
Algorithm
        while (k>0)
        begin
    if k is odd then
        v = 2 – (k mod 4)
        k = k – v
    if
        v =1 then Q = Q + P
        v = -1 then Q = Q – P
        end
  k = k/2
  P = 2P
end
return Q
```

Fig.1. Binary Non- adjacent Form Algorithm

5.2 Hardware Implementation

The target platform for the current implementation is the Altera DE2 board with Cyclone II FPGA (EP2C35672C6). To the best of the authors' knowledge no earlier implementation on this hardware exists.

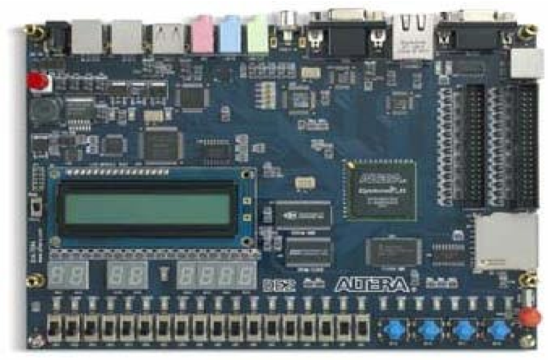


Fig.2. Target Hardware – Altera DE2 board with the Cyclone II FPGA

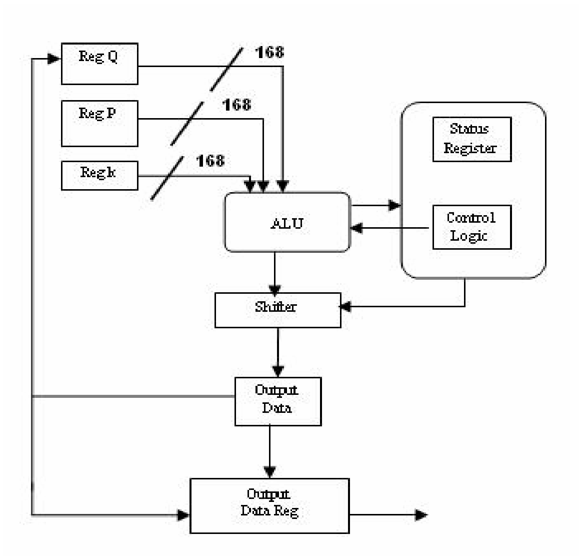


Fig.3. Hardware Architecture for the Accelerator Implementation

5.2.1 Hardware Architecture

From figure 3 it's clear that the ALU has inputs from three registers, each of these registers are 168 bit long. The ALU has a function select input which determines whether an addition is to be performed or subtraction. The shifter is used to left shift the input P by one and right shift the value of k once, as needed in the ECC algorithm. The status register definition is specified in the table1.

Table.1. Definition of Status Register

Bit	Definition
IRQ	When set requests for interrupt
Qr	When set selects the register Q
Pr	When set selects the register P
Start	Active high to start processing
Done	When set indicates completion

The ALU can consist of ripple adders as seen in figure 4; the layout of ripple carry adder is simple, which allows for fast design time.
The advantage with using ripple adders is that the delay is proportional to the number of

adders, in our implementation it's twenty one. The figure below represents a general ripple adder.

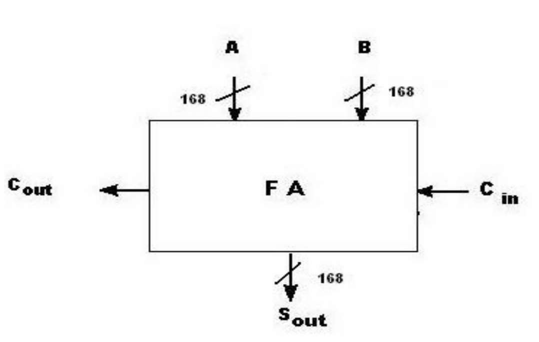


Fig.4. 168 bit Ripple Adder

5.3. Integrating Hardware and Software

The interface between the Nios II processor and the hardware accelerator is provided by means of an Avalon bus [1]. The Avalon bus allows us to connect several components in the FPGA thereby reducing the complexity of the design [6]. There are several standard interfaces available; the ones used in this work are the Avalon interrupt interface, clock interface, the master – slave memory mapped interface and the conduit interface.

Hardware accelerators can operate in parallel with a host processor; interrupt interfaces allow slave components (hardware accelerator here) to signal events to master (processor) [6].

The processor moves data into the internal registers of the accelerator and sets the start bit in the status register, from here on all the calculations are done in the hardware which are going to be faster than the software. While the accelerator processes registered data, the host processor can perform other tasks. Once the calculations are done, the result is moved onto the output register. At that time, the 'done' bit will be set, and an interrupt (IRQ) will be signaled to the processor, to collect the result from the output register. The processor can then clear the interrupt by writing to the status register using the accelerator slave interface [7]. The timing of the (IRQ) signal must be synchronous to the rising edge of its associated clock. The figure 5 [7], presents how the hardware accelerator and processor interact with each other.

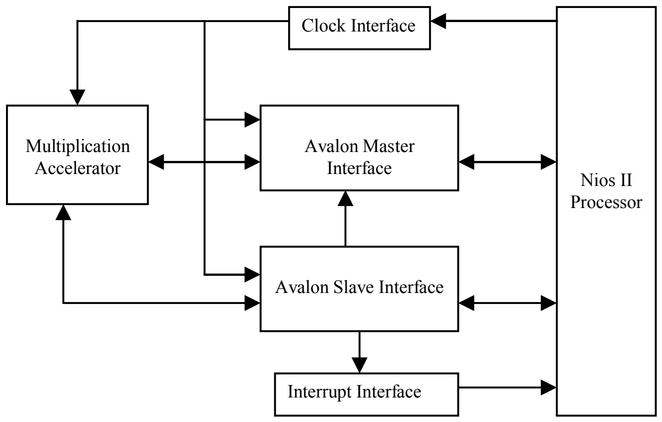


Fig.5. Communication between the Accelerator and Processor

The communication between the Nios II processor and the FPGA is further explained in the next subsection.

5.3.1 Software Design

The software design deals with the firmware which is executed by the Nios II processor. The firmware was developed using the C programming language. The core function of the firmware was, to write data to and to read data from the custom user component system, i.e. from the multiplication accelerator.

The software for the firmware of the Nios II processor was developed in the Nios II Integrated Development Environment (Nios II IDE). A standard Nios II system for specified target hardware consists of an application file which contains the user code and the system file that contains the Hardware Abstraction Layer (HAL) drivers. These drivers enable the hardware to interface with the software [9]. The Nios II IDE thus provides the user with the flexibility to access the hardware of a system without having to write any device driver software.

The goal of this work is to fasten the point multiplication operation; for this reason, the whole operation was shifted to hardware. The processors task is to write the data into the accelerators internal register and read the data from the external register once the whole operation is completed.

The read and write operations are carried out using the IORD and IOWR macros respectively. IORD macro reads data from a register or device port and IOWR writes data to a device port. The syntax of IORD is IORD (base address, offset). The syntax of IOWR is IOWR (base address, offset, data). This macro writes data to a register at the specified offset.

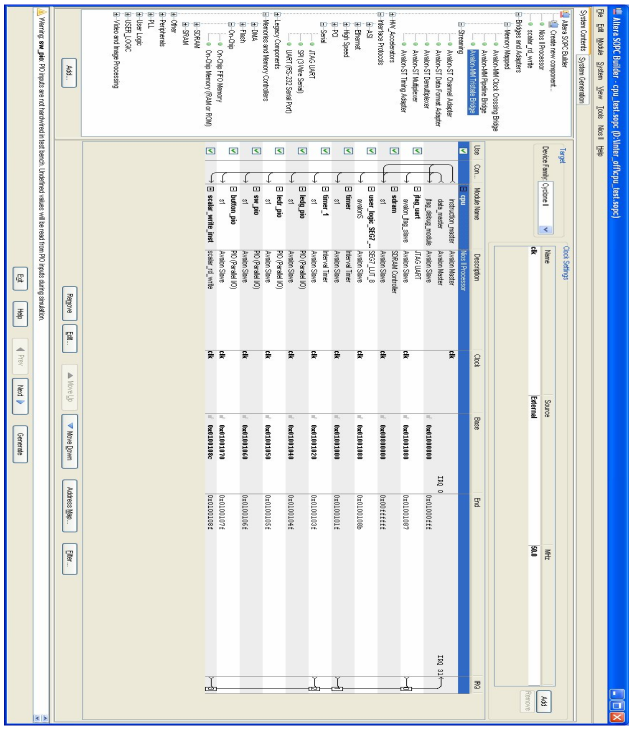
5.3.2 Hardware System Design

The hardware system design consists of building a custom component that carries out the point multiplication operation. For the current work the custom component is called scalar_rd_write. The functionalities of the component have been described in the previous section.

The System on Programmable Chip (SOPC) builder, which is included in the Altera Quartus design suite, is used to build and configure the embedded microprocessor systems [9]. The SOPC builder is a tool which can be used to develop systems not necessarily having a Nios processor also [15]. The SOPC builder consists of several standard library controllers like memory controllers, network controllers and many more. Apart from the standard library components, the sopc builder provides the user with the flexibility of developing custom components and then connecting them to the processor by means of the Avalon interconnect fabric.

The SOPC builder allows the user to specify the components required in the system through a graphical user interface (GUI) and then generates the Avalon interconnect logic automatically [15].

The figure 6 depicts the graphical interface of the SOPC builder.


Altera SOPC Builder - cpu_test.sopc (D:\Inter_off\cpu_test.sopc)
File Edit Module System View Tools Nios II Help
System Contents System Generation
Altera SOPC Builder
Create new component...
Nios II Processor
scalar_rd_write
Bridges and Adapters
Memory Mapped
Avalon-MM Clock Crossing Bridge
Avalon-MM Pipeline Bridge
Avalon-MM Tristate Bridge
Streaming
Avalon-ST Channel Adapter
Avalon-ST Data Format Adapter
Avalon-ST Demultiplexer
Avalon-ST Multiplexer
Avalon-ST Timing Adapter
HW_Accelerators
Interface Protocols
ASI
Ethernet
High Speed
PCI
Serial
JTAG UART
SPI (3 Wire Serial)
UART (RS-232 Serial Port)
Legacy Components
Memories and Memory Controllers
DMA
Flash
On-Chip
On-Chip FIFO Memory
On-Chip Memory (RAM or ROM)
SDRAM
SRAM
Other
Peripherals
PLL
User Logic
USER_LOGIC
Video and Image Processing
Add...
Remove Edit... Move Up Move Down Address Map... Filter...
Exit Help Prev Next Generate

Target
Device Family: Cyclone II

Clock Settings

Name	Source	MHz
clk	External	50.0

Use	Con...	Module Name	Description	Clock	Base	End	IRQ
✓		cpu	Nios II Processor				
		instruction_master	Avalon Master	clk			
		data_master	Avalon Master		IRQ 0		IRQ 31
		jtag_debug_module	Avalon Slave		0x01000800	0x01000fff	
✓		jtag_uart	JTAG UART				
		avalon_jtag_slave	Avalon Slave	clk	0x01001080	0x01001087	0
✓		sdram	SDRAM Controller				
		s1	Avalon Slave	clk	0x00800000	0x00ffffff	
✓		user_logic_SEG7_...	SEG7_LUT_8				
		avalonS	Avalon Slave	clk	0x01001088	0x0100108b	
✓		timer	Interval Timer				
		s1	Avalon Slave	clk	0x01001000	0x0100101f	1
✓		timer_1	Interval Timer				
		s1	Avalon Slave	clk	0x01001020	0x0100103f	2
✓		ledg_pio	PIO (Parallel I/O)				
		s1	Avalon Slave	clk	0x01001040	0x0100104f	
✓		ledr_pio	PIO (Parallel I/O)				
		s1	Avalon Slave	clk	0x01001050	0x0100105f	
✓		sw_pio	PIO (Parallel I/O)				
		s1	Avalon Slave	clk	0x01001060	0x0100106f	
✓		button_pio	PIO (Parallel I/O)				
		s1	Avalon Slave	clk	0x01001070	0x0100107f	
✓		scalar_write_inst	scalar_rd_write	clk	0x0100108c	0x0100108f	3

Warning: sw_pio: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.

Fig.6. Snapshot of the SOPC builder GUI

6. Results & Conclusion

The results of the software implementation have been presented in the table below and a comparison of the achieved results with that of [1] has been made.

Table.2. Results of Software Implementation

Field Size	Timing (cycles)
This Work - 168	39k
This Work - 192	50k
Jian Yang et al.- 163	10k

From table 2 it's clear that even though the present implementation takes more time, it's still comparable to [1]. This research is still a work in progress. Table 3 presents the initial results of our hardware implementation. The table presents a rough estimate of the number of registers and logic elements our implementation utilizes.

Table 4 presents a comparison between the number of registers and logic elements utilized in [1] and the current implementation.

Table.3. Number of Logic Elements and Registers Utilized

This Work (192 bit)	Before Accelerator Implementation	After Accelerator Implementation
Logic Elements	3262	3757
Registers	2470	2803

As seen form table 4, our implementation requires only 3918 logic elements out of 33,216 available (only 12% of the FPGAs resources).

Table.4. Comparison of Hardware Implementation Results

Accelerator	Logic Elements	Registers
This Work – Multiplication	3757	2803
Jian Yang et al.- Multiplication	2939	690

6.1. Hardware Accelerator VS Software

The communication of data between the Nios II processor and the hardware accelerator was carried out in two ways. The results acquired from both have been presented in this section.

The first way was to pass the data from Nios II to the FPGA sequentially 32 bits at a time. The FPGA would then carry out the scalar multiplication operation on the received data and then send the result back. To send 192 bits of data the whole process is repeated 6 times.

The second method is to transmit 192 bits of data at once, i.e. 192 bits of k, P (p_x, p_y) are transmitted. The method of transmission is still the same; the Nios processor still transmits 32 bits at a time but in this case all of them are combined into single registers of 192 bits. Once this is done, the scalar multiplication operation is carried out and the result calculated. The result is then sent back to the processor. Table 5 below presents the results obtained from both the implementation schemes.

Table.5. Fmax achieved for the Hardware Implementation Schemes

Implementation method	F_{max}
Transmitting 32 bits serially	148.06 MHz
Transmitting 192 bits serially	132.49 Mhz

Another factor that was calculated is the speed-up achieved through hardware implementation. The value of speed up is calculated as the ratio of software to hardware performance, based on Fmax set to 50MHz for both hardware and software for comparison fairness.

If we denote Speed up by SP, then SP can be calculated as:

SP = Hardware Performance/ Software Performance

Software performance (SW)

Considering the case with field size of 192, the timing is 50000 cycles per product (table 2). The frequency of operation is 50 MHz or 50000000 cycles per second.

Cycles per product / Cycles per second:
50000 / 50000000 = 0.001

SW = 1 / 0.001 = 1000 products per second

Therefore, the number of products per second on the software platform would be 1000.

Hardware Performance (HW)

Hardware performance in ideal case would be as follows:

1/ [1 cycle (192 bit) running at 50000000]
= 1/ 20 nsec
= 50,000,000 products per second.

The Speed up with this ideal case is 50,000. Thus for fairness and presenting a real scenario, the delay is taken into account and the Fmax for both hardware and software is set to 50MHz for fairness.

We need 8 cycles for transmitting px, 8 for py, 8 for k, 16 for receiving qx 16 for qy, plus 2 cycle for calculating products qx, and qy.

The speed up now, based on total of [8+8+8+16+16+2] /2 = 29 cycles is:
SP =50,000/29 = 1724. The speed up achieved after taken delay in to account is almost 1725 times faster.

Table 6 presents a comparison between the results obtained by [1] with that of the current work.

Table.6. Comparison of Hardware Results

Reference	Field Size	Logic Elements	Registers	Fmax (MHz)	Speed - Up
Jian Yang et al.	163	2939	690	94	1666
This Work	192	3757	2803	132.49	1724

The results included in the table above are for the second hardware – software communication scheme where 192 bits of data are sent. Jian Yang et al. [1] in their paper have mentioned that the above results were available in 6 iteration cycles, for the current work the number of cycles as shown above would be 29. The results achieved imply that the current implementation is able to accelerate the overall performance of scalar multiplication, thus the goal of improving the performance through hardware acceleration is achieved.

Figure 7 presents simulation results of our hardware accelerated multiplication. First half of the figure presents output generated with random 192 bit inputs. The second half is for verification where inputs were provided in the final clock cycle of the test bench, the output is acquired in the next cycle. The inputs provided were of powers of two for ease of verification. The simulation was carried out using ModelSim.

The authors of this paper use [1] as a standard for comparison even though the field of operation used in [1] is different. This has been done because literature survey revealed that the authors of [1] used the same target hardware as the current implementation does.

.

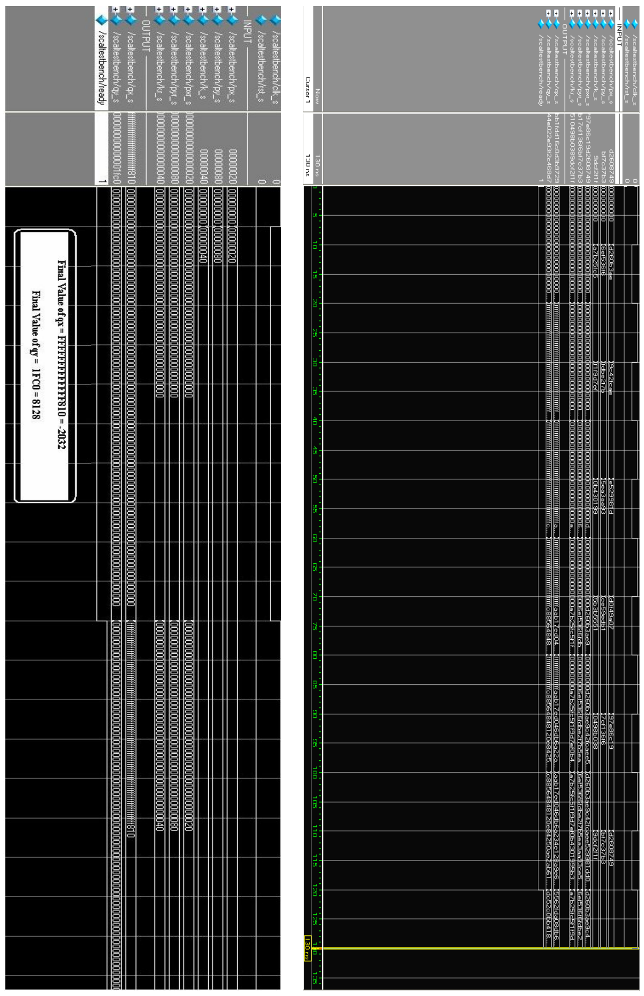

Final Value of qx = FFFFFFFFFFFFF810 = -2032
Final Value of qy = 1FC0 = 8128

Fig.7. Implementation Results on ModelSim

ACKNOWLDEGEMENT

The authors of the paper would like to sincerely thank Mohammed Abdallah, for his help during the course of this work.

References:

[1] Jian - Yang Zhou and Xiao - Gang Jiang, "Accelerating Elliptic Curve Cryptography On System-on- Programmable- Chip," *IEEE International Workshop on Anti-Counterfeiting, Security,* 2007, pp. 292-295.

[2] S. B. Ors, L. Batina, B. Preneel and J. Vandewalle, "Hardware Implementation of an Elliptic Curve Processor over GF (p)," *IEEE International Conference on Application-Specific Systems, Architectures, and Processors,* 2003, pp. 433-443.

[3] A. Satoh and K. Takano, "A Scalable Dual-Field Elliptic Curve Cryptographic Processor," *IEEE Transactions on Computers*, Vol. 52 NO.4, 2003, pp. 449-460.

[4] G. Orlando and C. Paar, "A scalable GF (p) Elliptic Curve Processor Architecture for Programmable Hardware," *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems*, 2001, pp. 356–371.

[5] J. Lopez and R. Dahab, An Overview of Elliptic Curve Cryptography, Technical report, Institute of Computing, State University of Campinas, 2000.

[6] Altera SOPC Builder Component Development Walkthrough, available on: http://www.altera.com/literature/hb/qts/qts_qii54007.pdf

[7] Altera Avalon Interface Specification, available on: http://www.altera.com/literature/manual/mnl_avalon_spec.pdf

[8] K.Rabah, "Implementation of Elliptic Curve Diffie – Hellman and EC Encryption Schemes," *Information Technology Journal,* 2005, pp. 132 – 139.

[9] Altera Nios II Software Developer's Handbook, available on: http://www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf

[10] J. Ho and R. Yang, "A Comparison of Secure Mechanisms for Mobile Commerce," *Proceedings of the 7th WSEAS International Conference on Mathematics & Computers in Business & Economics*, 2006, pp. 24 -28.

[11] P.H. Huang, "The Investigation of the Elliptic Curve Cryptology Applies to the New Generation Protocol," *WSEAS Transactions on Computers*, Vol. 7 No. 6, 2008, pp. 694-703.

[12] A. Georgiades et al., "Trinity Protocol for Authentication of Binding Updates in Mobile IPv6," *Proceedings of the 9th WSEAS International Conference on Communications*, 2005.

[13] S. Kumar et al., "Embedded End-to-End Wireless Security with ECDH Key Exchange," *Proceedings of the 46th IEEE Midwest Symposium on Circuits and Systems*, 2003.

[14] Altera Introduction to SOPC builder, available on: http://www.altera.com/literature/hb/qts/qts_qii54001.pdf