Design and Evaluation of Hardware Accelerator for Elliptic Curve Cryptography Point 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. 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. 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, FPGA, Prime Field.

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 \( k \cdot P \) 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 \( \text{GF}(p) \) and the binary field \( \text{GF}(2^m) \). \( \text{GF}(p) \) 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 \( p \). \( \text{GF}(2^m) \), contains \( 2^m \) elements where \( m \) is degree of the field. 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 \( \text{GF}(p) \). 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 \( \mu \)m 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.

3. Mathematics behind Elliptic Curves on Prime Field \( \text{F}_p \)

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:

\[
y^2 \mod p = x^3 + ax + b \mod p \tag{1}
\]

where \( p \) is the prime modulus defined for the finite field \( \text{F}_p \). 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,

\[
x_c = (m^2 - x_a - x_b) \mod p
\]
\[
y_c = (m(x_a-x_c)-y_a) \mod p
\]
\[
m = (y_a-y_b / x_a - x_b) \mod p
\]

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,

\[
A - B = A + (-B) \quad \text{where} \quad -B = (x_b, -y_b)
\]

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

\[
x_c = (m^2 - 2x_a) \mod p
\]
\[
y_c = (m(x_a-x_c)-y_a) \mod p
\]
\[
m = ((3x_a^2 + a)/ 2 y_a) \mod p
\]

If \(y_a\) is zero then \(m\) is zero as a result \(C = O\), the point at infinity. The slope equation (7), above utilizes the curve parameter ‘a’.

4. 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.

4.1 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.

4.1.1 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. 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

\[
\text{while } (k>0)
\]

\[
\text{begin}
\]

\[
\text{if } k \text{ is odd then}
\]

\[
\v = 2 – (k \mod 4)
\]

\[
k = k – \v
\]

\[
\text{if }
\]

\[
\v = 1 \text{ then } Q = Q + P
\]

\[
\v = -1 \text{ then } Q = Q – P
\]

\[
\text{end}
\]

\[
k = k/2
\]

\[
P = 2P
\]

end

return \(Q\)

Fig.1. Binary Non- adjacent Form Algorithm

4.2 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.
4.2.1 Hardware Architecture

From figure 2 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 table 1.

Table 1. Definition of Status Register

<table>
<thead>
<tr>
<th>Bit</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRQ</td>
<td>When set requests for interrupt</td>
</tr>
<tr>
<td>Qr</td>
<td>When set selects the register Q</td>
</tr>
<tr>
<td>Pr</td>
<td>When set selects the register P</td>
</tr>
<tr>
<td>Start</td>
<td>Active high to start processing</td>
</tr>
<tr>
<td>Done</td>
<td>When set indicates completion</td>
</tr>
</tbody>
</table>

The ALU can consist of ripple adders as seen in figure 3; 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.

4.3. 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 and the memory mapped 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 3 [7], presents how the hardware accelerator and processor interact with each other.

Fig.4. Communication between the Accelerator and Processor

5. 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

<table>
<thead>
<tr>
<th>Field Size</th>
<th>Timing (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>This Work - 168</td>
<td>39k</td>
</tr>
<tr>
<td>This Work - 192</td>
<td>50k</td>
</tr>
<tr>
<td>Jian Yang et al.- 163</td>
<td>10k</td>
</tr>
</tbody>
</table>

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. We expect that, the result of the hardware implementation will improve the overall performance of ECC. 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

<table>
<thead>
<tr>
<th></th>
<th>This Work (168 bit)</th>
<th>Before Accelerator Implementation</th>
<th>After Accelerator Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Elements</td>
<td>3262</td>
<td>7276</td>
<td></td>
</tr>
<tr>
<td>Registers</td>
<td>2470</td>
<td>2971</td>
<td></td>
</tr>
</tbody>
</table>

As seen from 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

<table>
<thead>
<tr>
<th>Accelerator</th>
<th>Logic Elements</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>This Work – Multiplication</td>
<td>7276</td>
<td>2971</td>
</tr>
<tr>
<td>Jian Yang et al.- Multiplication</td>
<td>2939</td>
<td>690</td>
</tr>
</tbody>
</table>

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.

Figure 5 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 intend to make a more thorough analysis of the hardware accelerated multiplication and compare the results with the ASIC based ECC processor in [3].

ACKNOWLEDGEMENT

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

References:


