CHAPTER 3

THE PROPOSED SYSTEM OF IMPLEMENTING

ELLIPTIC CURVE CRYPTOGRAPHY

USING VERILOG HDL

This chapter illustrates the proposed system of implementing Elliptic Curve Cryptography using Verilog HDL. This introduces ECC architectures targeted for hardware implementation in programmable hardware. Software architectures have the great advantage that they are portable to multiple hardware platforms. Their main disadvantage are their lower performance when compared to specialized hardware architectures and their inability to protect private keys from disclosure with the same degree of security that is achievable in hardware. These disadvantages are some of the reasons motivating the study of efficient hardware architectures. The program for implementing ECC in hardware using FPGA is written in Verilog Hardware Description language. FPGA’s are reconfigurable hardware devices whose functionality is programmable. The configuration of an FPGA device can be changed over time thus allowing the same FPGA to implement different functions.

3.1 VERILOG DESCRIPTION

Verilog HDL is a Hardware Description Language (HDL). A Hardware Description Language is a language used to describe a digital system, for example, a computer or a component of a computer. One may describe a digital system at several levels. For example, a HDL might describe the layout of the wires, resistors and transistors on an Integrated
Circuit (IC) chip, and the switch level or, it might describe the logical gates and flip-flops in a digital system, i.e., at the gate level. An even higher level describes the registers and the transfers of vectors of information between registers. This is called the Register Transfer Level (RTL). Verilog supports all of these levels. However, this thesis focuses on only the portions of Verilog, which support the RTL level.

Verilog data types are very simple, easy to use and very much geared towards modeling hardware structure. All data types used in a Verilog model are defined by the Verilog language and not by the user. There are net data types, for example wire, and a register data type called reg.

Verilog may be preferred because of its simplicity. There is no concept of packages in Verilog. Functions and procedures used within a model must be defined in the module.

Verilog HDL allows a hardware designer to describe a high level of abstraction at the architectural or behavioral level as the lower implementation levels i.e., gate and switch levels leading to very large scale integration (VLSI) Integrated Circuits (IC) layouts and chip fabrication. A primary use of HDLs is the simulation of designs before the designer must commit to fabrication. This project focuses the use of Verilog HDL at the architectural or behavioral levels and emphasizes design at the Register Transfer Level (RTL).
The major capabilities of the Verilog Hardware Description Languages are listed below.

- Primitive logic gates such as (AND), (OR), and (NAND) are built-in into the language.

- Flexibility of creating a User Defined Primitive (UDP) such as primitive could either be a combinational logic primitive or sequential logic primitive.

- Switch level modeling, primitive gates such as PMOS and NMOS are also built-in into the language.

- Explicit language constructs are provided for specifying pin to pin, delays, path delays, and timing checks of a design.

- There are two data types in Verilog HDL: The net data type and the register data type. The net data type represents a physical connection between structural elements while a register type represents an abstract data storage element.

- A design can be arbitrary size; the language does not impose a limit.

- Verilog HDL is non-proprietary and is an IEEE standard.

Figure 3.1 given below shows the typical design flow in Verilog.
Figure 3.1 Typical Design Flow in Verilog
3.2 PROBLEM DEFINITION

The security of the ECC is based on the apparent intractability of the following elliptic curve discrete logarithm problem (ECDLP): Consider the equation, \( Q = kP \), where \( Q, P \) are points in the elliptic curve \( E(a,b) \) and \( k < P \). It is relatively easy to calculate \( Q \) given \( k \) and \( P \), but it is relatively hard to determine \( k \) given \( Q \) and \( P \). This is called discrete logarithmic problem for elliptic curves (MENE 97).

The elliptic curve consists of all real numbers for the points \( x, y, a \) and \( b \) in the \((x,y)\) coordinate plane. The \( E(a,b) \) curve plane satisfies the following equation: \( y^2 = x^3 + ax + b \mod p \). The prime number \( p \) sets the upper limits of the equation and is used for modulus arithmetic. \( P \) and \( Q \) are the points on the elliptic curve. When using ECC, there are two types of arithmetic, the cartesian coordinates for resolving the elliptic curve and modular arithmetic used for resolving the the points along the coordinate system \( k \) is a very large integer generated at random which is multiplied with the point.

This system enhances the security of data transfer as well as reduces the size of the cipher text thereby eliminating the drawbacks of Diffie Heilman and ElGamal algorithms. The working of the proposed system is as follows. In Figure 3.2, module A is assumed to be the sender and module B to be the receiver. A message, \( M \) has to be transmitted from \( A \) to \( B \). This message has to be encrypted before transmission and the receiver must be able to obtain the original message after decryption.
P is a public key used for encryption. x is a private key known only to module A. Module A calculates xP and makes it public. The values of x and P are chosen such that even with the knowledge of P, it would be nearly impossible to calculate x. Similarly, y is a private key of module B. Module B calculates yP and makes it public (Cryptography and network security, third edition, William Stallings, Prentice Hall of India, 2005). The encryption and decryption steps involved in transmission and reception of a message using ECC is described below.

3.3 STEPS INVOLVED

Encryption:

- Let x, y be the private keys used by the transmitter and receiver respectively. The transmitter secret key x is multiplied with the public value of the receiver yP i.e., xyP.
- The message is encrypted using the formula M + xyp, where M is the plain text.
• Decryption:
  • The receiver’s secret key \( y \) is multiplied with the public value of the transmitter \( xP \) i.e., \( yxP \)
  • The message is decrypted by subtracting the value \( yxP \) from the received message i.e., \( M + xyP - xyP = M \). The same is illustrated in Figure 3.1.

The above said encryption and decryption process are used to implement ECC algorithm in the proposed hardware module. Figure 3.3 shows the flowchart for encryption algorithm and Figure 3.4 shows the flowchart for decryption algorithm.

---

**Figure 3.3  Flowchart for Encryption**
Figure 3.4 Flowchart for Decryption
3.4 MAIN MODULES OF ELLIPTIC CURVE CRYPTOGRAPHY

The basic unit of description in verilog is the module. A module describes the structure or functionality of a design and also describes the ports through which it communicates externally with the other modules.

The structure of a design is described using switch-level primitives, gate-level primitives and user defined primitives. Dataflow behaviour of a design is described using continuous assignments and sequential behaviour is described using procedural constructs. A module can also be instantiated inside another module. The basic syntax of a module is given below.

```verilog
module module_name (port_list);
    declarations
    reg, wire, parameter
    input, output, inout.
    Functions, tasks,
    statements
    initial statements
    always statements,
    .......
    Endmodule
```

The implementation in verilog HDL consists of three main modules namely,

i) Main Controller

ii) Multiplier and

iii) Adder
Figure 3.5 shows the above three modules. The main controller controls the functioning of the adder and multiplier components. The multiplier block is selected when the Enable line is ‘00’. The multiplier performs multiplication of an integer with a point on the elliptic curve. The multiplication is done by successive addition.

The adder block is selected when the Enable line is ‘01’. The adder performs addition of two points on an elliptic curve. Addition is based on the rules of Elliptic Curve Arithmetic known as point addition (Gerardo Orlando, ‘Efficient elliptic curve processor architecture for field programmable logic’, Thesis for Ph.D in Electrical Engineering, Worcester Polytechnic Institute, 2002).

![Diagram of Main Modules in Elliptical Curve Cryptography](image)
Figure 3.6 shows the block diagram of the main controller.

**MAIN CONTROLLER**

The *main controller* controls the functioning of the *adder* and *multiplier* components. It has several internal signals, the functions of which are mentioned below.

*Clock*: The internal clock

*Reset*: The reset signal is used to bring back all the components to their initial conditions, when set to ‘1’.

*Mx*: X-coordinate of the message to be transmitted
My : Y - coordinate of the message to be transmitted

aPx : X - coordinate of the quantity “xP”, (required in the encrypter part)

aPy : Y - coordinate of the quantity “yP”, (required in the encrypter part)

Enc_Dec : Selects encryption/decryption

‘0’ – Encryption

‘1’ – Decryption

k_l : A very large integer (Private key) generated at random

En : Enables Multiplier/point Adder

‘00’ – Multiplier

‘01’ – Point Adder

To Multiplier

k : A very large integer generated at random which is multiplied with the point

oPx : X - coordinate of the point which is to be multiplied with the integer

oPy : Y - coordinate of the point which is to be multiplied with the integer

To Point Adder

oPx : X - coordinate of the addend

oPy : Y - coordinate of the addend
oQx : X - coordinate of the augend
oQy : Y - coordinate of the augend

From Multipler

iPx : X - coordinate of the result
iPy : Y - coordinate of the result

From Point Adder

\text{add\_iPx} : X - coordinate of the result
\text{add\_iPy} : Y - coordinate of the result

Final Outputs

kPx : X - coordinate of the product “xP” (Encryption)
kPy : Y - coordinate of the product “yP” (Encryption)
outx : X - coordinate of the expression “xyP+M”
outy : Y - coordinate of the expression “xyP+M”

The Figure 3.7 shows the block diagram of multiplier. The multiplier block is selected when the En line is ‘00’. The multiplier performs multiplication of an integer with a point on the elliptic curve. The multiplication is done by successive addition. This addition follows the rules of Elliptic Curve Arithmetic, known as \textit{Point Doubling}. 
MULTIPLIER

Figure 3.7  Block Diagram of Multiplier

From Main Controller

\( k \) : it is a randomly generated integer which is multiplied with a point on the curve

\( Pax \) : x-coordinate of a point on the elliptic curve

\( Pay \) : y-coordinate of a point on the elliptic curve

\( En \) : enables the multiplier block

To Main Controller

\( kPax \) : x-coordinate of the resultant

\( kPay \) : y-coordinate of the resultant

The number of processes required is more in affine co-ordinates. Hence we go for computation in projective co-ordinates. The final result is given in affine co-ordinates. The formulae for the relevant conversions are given in Table 3.1.
Table 3.1  Table for conversions from affine to projective and projective to affine

<table>
<thead>
<tr>
<th>Affine to Projective</th>
<th>Projective to Affine</th>
</tr>
</thead>
<tbody>
<tr>
<td>X = x</td>
<td>x = X/Z^2</td>
</tr>
<tr>
<td>Y = y</td>
<td>y = Y/Z^3</td>
</tr>
<tr>
<td>Z = 1</td>
<td></td>
</tr>
</tbody>
</table>

The basic formulae used for multiplication in projective coordinates are given below:

Point doubling (*in projective coordinates*)

\[
\begin{align*}
(X_3, Y_3, Z_3) &= 2(X_1, Y_1, Z_1) \\
M &= 3X_1^2 + aZ_1^4 \mod p \\
Z_3 &= 2Y_1Z_1 \mod p \\
S &= 4X_1Y_1^2 \mod p \\
X_3 &= M_2 - 2S \mod p \\
T &= 8Y_1^4 \mod p \\
Y_3 &= M (S - X_3) - T \mod p
\end{align*}
\]

Figure 3.8 shows the block diagram of adder module. The adder block is selected when the *En* line is ‘01’. The adder performs addition of two points on an elliptic curve. Addition is based on the rules of Elliptic Curve Arithmetic known as point addition. Similar to multiplication, addition is performed on projective co-ordinates and the formulae are given in the above table. The result is in affine.
**ADDER**

**Figure 3.8 Block Diagram of Adder**

**From Main Controller**

- \( x_P \): x-coordinate of a point on the elliptic curve
- \( y_P \): y-coordinate of a point on the elliptic curve
- \( x_Q \): x-coordinate of a point on the elliptic curve
- \( y_Q \): y-coordinate of a point on the elliptic curve
- \( En \): enables the adder block

**To Main Controller**

- \( x_R \): x-coordinate of the resultant
- \( y_R \): y-coordinate of the resultant

The basic formulae used for addition in projective coordinates are given below:

**Point addition** (in projective coordinates)

\[ (X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (X_2, Y_2, Z_2) \]

\[ U_1 = X_1Z_2^2 \mod p \]
\[ S_1 = Y_1Z_2^3 \mod p \]
\[ U_2 = X_2Z_1^2 \mod p \]
\[ S_2 = Y_2Z_1^3 \mod p \]
\[ W = U_1 - U_2 \mod p \]
\[ R = S_1 - S_2 \mod p \]
\[ T = U_1 + U_2 \mod p \]
\[ M = S_1 + S_2 \mod p \]
\[ Z_3 = Z_1 Z_2 W \mod p \]
\[ X_3 = R^2 - T W^2 \mod p \]
\[ V = T W^2 - 2 X_3 \mod p \]
\[ Y_3 = (V R - M W^3) \cdot 2^1 \mod p \]

3.5 HARDWARE IMPLEMENTATION STEPS

The easiest and most efficient language considered for condensing the larger circuits is Verilog HDL. It is easily understandable. It is used for specification, simulation and synthesis of an electronic system. Digital circuits could be described at register transfer level (RTL) by use of HDL.

Figure 3.9 shows that the program written in verilog HDL run by Xilinx 7.1 ise tool when synthesised gives the hardware module of the program upto the gate level and then given for implementation using FPGA which does the chip layout. Before implementation the program is verified by using simulation softwares such as modelsim 5.7 which gives the output as signal waveform and even the testbench program can be written and verified using simulation software before synthesis using xilinx 7.1.

The logic synthesis pushed the HDLs into the forefront to digital design. And there is no need to manually place gates to build digital circuits. They could describe complex circuits at an abstract level in terms of functionality and data flow by designing those circuits in HDLs. Logic synthesis tools implemented the specified functionality in terms of gates and
gate interconnections. Logic synthesis tools have cut design cycle times significantly. Logic synthesis is the process of converting a high level description of the design into an optimised gate level representation, given a standard cell library and certain design constraints.

Simulation is used for verification of HDL based design. They load the verilog HDL code and simulate the behaviour in software. Modelsim 5.7 is one of the software which is used for verification of the program in this thesis. The output will be in the form of the signal waveform.

Figure 3.9 Data Flow in Verilog HDL
The gate level netlist is verified for functionality and timing and then given for implementation using FPGA. It does the chip layout, checks that the post layout circuit meets timing requirements and then fabricates the IC chip.

3.6 SIMULATION OF ENCRYPTION AND DECRYPTION

In order to implement ECC cryptographic system three modules namely (i) main controller (ii) point adder and (iii) multiplier are to be simulated using ModelSim SE 5.7g simulator. The program for the above three modules are written in verilog HDL.

There are three ways to simulate a design, software simulation, hardware acceleration and hardware emulation. Software simulation is used here using modelsim 5.7 which is used to run Verilog HDL based design. They load the verilog hdl code and simulate the behaviour in software.

When designs started exceeding one million gates, software simulation began to consume large amount of time and became a bottle neck in the verification process. Thus other techniques such as hardware acceleration and hardware emulation are used to accelerate these simulations.

Verilog simulation requires two main blocks, namely

- Design Block
- Stimulus Block

HDL uses design block to describe the actual circuit. All modules are defined down to the lowest level leaf cells in the design methodology. Once the design block is completed it must be tested checking results can test
the functionality of the design block. The block that is used to perform this operation is called as the stimulus block. The stimulus and design blocks are placed separately in order to avoid confusion (Samir Palnitkar, ‘Verilog HDL’, 2nd edition, Pearson education, April 2006).

When the encryption decryption is low (0), then the encryption part is selected. The simulation waveforms of multiplier block, adder block and main controller are obtained. The simulation waveform for multiplier block is given below. When the enable line is ‘00’ the multiplier block is selected. Then the clock is set as high and the reset as low, the system will multiply the constant K with the combination of public and private key and gives the product as KPax and KPay. The resulting output for encryption and decryption of multiplier are shown in the screen shots below in Figure 3.10.

Encryption/Multiplier
When the enable line is ‘01’ the adder block is selected. Then the clock is set as high and the reset as low. The adder performs the addition of two points on the elliptic curve. The message coordinates are added with the public key coordinates and the result is given to the main controller. The inputs are Mx, My, aPx and aPy and the result is Mx+aPx and My+aPy. The output screen shots obtained for the above operations are in Figure 3.11.
Decryption/Adder

The main controller controls the function of adder and multiplier. It has several internal signals for controlling the operation of the whole system. The clock is set as high and the reset is low. It has the input as the message coordinates and the public and private keys. The encrypted message coordinate is obtained as an output in the encryption process and the original message is obtained back in the decryption process. The screen shots corresponding to the main controller’s output are shown in Figure 3.12.
3.7 SYNTHESIS

The advent of computer aided logic synthesis tools has automated the process of converting high level descriptions to logic gates. Verilog HDL has become one of the popular HDL’s for writing high level description. Automated logic synthesis has reduced the time for conversion...
from high level design representation to gates. This has allowed designers to spend more time on designing at higher level of representation, because less time is required for converting the design to gates (Bhasker J, A Verilog HDL Primer, 2nd edition, B.S.Publications, 2001).

Verilog HDL allows to design each module independently and thus the complexity involved in the design is reduced to a great extent. In addition to this it supports parallel design which reduces the time required. The synthesised output gives the timing summary, number of multiplexers, comparators, flipflops, IOB’s, subtractors, adders, multipliers, registers, GCLK’s, Latches, etc.,

Field Programmable Gate Array (FPGAs) enable rapid development and implementation of complex digital circuits. FPGA devices can be reprogrammed, allowing the same hardware to be employed for entirely new designs or for iteration of the same design. While much of traditional IC logic synthesis methods apply, FPGA circuits have special requirements that affect synthesis. The FPGA device consists of a number of configurable Logic Blocks (CLBs), interconnected by a routing matrix. Pass transistors are used in the routing matrix to connect segments of metal lines.

3.7.1 Synthesis Result

The architecture has been synthesized by using Xilinx ISE 7.1i. The Place and Route report from ISE7.1i shows the number of logic gates consumed. Table 3.2 shows the device utilization summary of encoder and decoder and Table 3.3 gives the timing summary.
### Table 3.2 Device Utilization Summary

*Encoder / Decoder*

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Name</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Multiplexer</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>Adder</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>Subtractor</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>Multiplier</td>
<td>58</td>
</tr>
<tr>
<td>5</td>
<td>Comparator</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>Latches</td>
<td>22</td>
</tr>
<tr>
<td>7</td>
<td>LUT's</td>
<td>43</td>
</tr>
<tr>
<td>8</td>
<td>IOB’s</td>
<td>124</td>
</tr>
<tr>
<td>9</td>
<td>GCLK’s</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>Flip flops</td>
<td>118</td>
</tr>
<tr>
<td>11</td>
<td>Registers</td>
<td>6</td>
</tr>
</tbody>
</table>

### Table 3.3 Timing Summary

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Name</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Minimum period</td>
<td>5.043 ns</td>
</tr>
<tr>
<td>2</td>
<td>Maximum frequency</td>
<td>198.295 MHz</td>
</tr>
<tr>
<td>3</td>
<td>Total memory usage</td>
<td>269824 Kilobytes</td>
</tr>
</tbody>
</table>
For the purpose of logic synthesis, designs are currently written in a HDL at a register transfer level (RTL). The term RTL is used for a HDL description style that utilises a combination of dataflow and behavioral constructs. Logic synthesis takes RTL based HDL description and converts it to an optimised gate-level netlist.

Chapter 4 provides the experimental results upto the RTL level and the gate level netlist of the encoder and the decoder and a synthesis report of the same.