Journal of Circuits, Systems, and Computers
Vol. 26, No. 11 (2017) 1750178 (14 pages)

DOI: 10.1142/S021812661750178X

Towards an Optimized Architecture for Unified Binary Huff Curves*

Atif Raza Jafri† and Muhammad Najam ul Islam‡

Department of Electrical Engineering,
Bahria University, Islamabad, Pakistan
†*atif.raza@bui.edu.pk*
‡*najam@bahria.edu.pk*

Malik Imran

Department of Electrical Engineering,
Abasyn University, Islamabad, Pakistan
imran.malik@abasynisb.edu.pk

Muhammad Rashid

Department of Computer Engineering,
Umm Al-Qura University, Makkah, Saudi Arabia
mfelahi@uqu.edu.sa

Received 22 July 2016
Accepted 4 March 2017
Published 20 April 2017

Applying unified formula while computing point addition and doubling provides immunity to Elliptic Curve Cryptography (ECC) against power analysis attacks (a type of side channel attack). One of the popular techniques providing this unifiedness is the Binary Huff Curves (BHC) which got attention in 2011. In this paper we are presenting highly optimized architectures to implement point multiplication (PM) on the standard NIST curves over $GF(2^{163})$ and $GF(2^{233})$ using BHC. To achieve a high throughput over area ratio, first of all, we have used a simplified arithmetic and logic unit. Secondly, we have reduced the time to compute PM through Double and Add algorithm. This is achieved by increasing the frequency of operation through a 2-stage pipelined architecture. The increase in clock cycles caused by consequent pipeline hazards is controlled through optimal scheduling of computations involved in PM. The synthesis results show that our designs can work up to a frequency of 377 MHz on Xilinx Virtex 7 FPGA. Moreover, the overall throughput/area ratio achieved through the adopted approach is up to 20% higher while comparing with available state-of-the-art solutions.

Keywords: Unified; binary huff curves (BHC); crypto processor; FPGA.

*This paper was recommended by Regional Editor Piero Malcovati.
†Corresponding author.

1. Introduction

Nowadays, Elliptic Curve Cryptography (ECC) is a choice in the field of asymmetric key cryptography due to its provision of shorter key lengths as compared to the well-known established Rivest–Shamir–Adleman[1] algorithm. Elliptic curves were first independently proposed by Neil Koblitz[2] and Victor Miller[3] in 1985. Scalar/point multiplication (PM) is the core operation in conventional ECC. In order to implement PM, two types of fields are involved: (1) prime field $GF(p)$ and (2) binary extension field $GF(2^m)$, by adopting either simple affine coordinates or projective coordinates. Elliptic curves over $GF(2^m)$ field are particularly more attractive because they provide efficient hardware implementations of finite field (FF) operations.[1] Projective coordinates are well suited to achieve efficient throughput/area ECC designs as compared to affine coordinates, in which for each point addition (PA)/point doubling (PD) an associated inversion operation is required.

The security strength of ECC mainly depends on the hardness of its discrete logarithmic problem.[1] To address this issue, PA and PD are the necessary operations. Different mathematical formulations for PA and PD to compute PM ensure that the addition laws on ECC are not unified.[4] However, the differences of these operations make the crypto system vulnerable to Side Channel Attacks (SCAs).[5] In order to resists SCAs, different forms of ECC have been introduced which provide unified addition (to compute PA and PD) laws such as Binary Edward Curves (BEC)[6] and Binary Huff Curves (BHC).[7] BEC require higher computational cost than BHC.[4] Hence, the crypto processor based on this curve is expected to be a simple SCA preventive.

The particular domains of applications for highly secure asymmetric key cryptosystems against SCAs are wireless sensor networks,[8] cloud computing,[9] radio frequency identification,[10] Identity-Based Encryption,[11] etc.

Limited hardware based research works, targeting architectures for those algorithms which are resistant against Simple Power Analysis Attacks (SPA) and SCAs, have been proposed previously. By using projective coordinates along with the hybrid Karatsuba multiplier and Itoh–Tsujii inversion algorithms, an FPGA based research work for the computation of PM is available in Ref. 4. To achieve hybrid approach for FF multiplication, they coupled simple and general Karatsuba multipliers. General Karatsuba multiplier is used for the better utilization of LUT over smaller bits while the simple Karatsuba multiplier is used for minimizing the gate counts over longer bits.[12] Additionally, to reduce the number of clock cycles (CC), they adopted the Quad block version of the Itoh–Tsujii algorithm. Another FPGA based approach is found in Ref. 13, where unified PA law has been implemented by adopting projective coordinates to make BHC more secure against SPA.

In this paper we have shown that the unified addition law, as proposed in Ref. 13 for BHC, can be efficiently implemented on FPGA resources by adopting different optimization techniques. On top level, polynomial basis representation along with

projective coordinate system is selected. We have selected Lopez and Dahab as it requires less field multiplications and inversion operations to implement PM.[1]

In order to attain higher throughput, nonpipelined, 2-stage and 3-stage pipelined architectures along with associated pipelining hazards are investigated. With the above stated selections, we have modeled our design for $GF(2^{163})$ and $GF(2^{233})$. Finally both designs are implemented at post place and route level on Xilinx Virtex 4, 5, 6 and 7 devices for performance estimation and, consequently, for comparison with state-of-the-art.

The remainder of this paper is organized as follows. In Sec. 2, preliminaries for BHCs over $GF(2^m)$ are presented. Our proposed architectures for BHC are discussed in Sec 3. Section 4 presents the hardware results and performance estimation of the proposed hardware architecture along with comparison with state-of-the-art. Finally, Sec. 5 concludes the paper.

2. Preliminaries

2.1. *BHC over GF(2^m)*

Initially, the Huff model was introduced in 1963.[14] Later on, the Huff model was revisited in 2010,[15] where the description and formulation for the odd characteristic fields were provided. Furthermore, an outline for binary field was presented and defined as the set of projective points $(X: Y: Z)$ over $GF(2^m)$ by satisfying the following equation:

$$E : aX(Y^2 + YZ + Z^2) = bY(X^2 + XZ + Z^2)\,. \tag{1}$$

In Eq. (1), the variables 'a' and 'b' are curve parameters and they $\in GF(2^m)$ while considering $a \neq b$.

2.2. *Unified addition law over GF(2^m)*

In 2011, Devigne and Joye developed the formal construction of a Huff model for binary field.[7] This construction provides the unified addition law (Unif_Add) which is illustrated in Table 1 for binary field.

Thereafter in 2013, Unif_Add for BHC was evaluated as explained in Ref. 13. They observed that the unified addition formulas, as described in Ref. 7, show behavioral differences when computing PA and PD which is vulnerable to the SCAs.

Table 1. Addition law (Unif_Add).[7]

$m_1 = X_1.X_2,\ m_2 = Y_1.Y_2,\ m_3 = Z_1.Z_2,\ m_4 = (X_1 + Z_1)(X_2 + Z_2) + m_1 + m_3$
$m_5 = (Y_1 + Z_1)(Y_2 + Z_2) + m_2 + m_3,\ m_6 = m_1.m_3,\ m_7 = m_2.m_3$
$m_8 = m_1.m_2 + (m_3)^2,\ m_9 = m_6(m_2 + m_3)^2,\ m_{10} = m_7(m_1 + m_3)^2$
$m_{11} = m_8(m_2 + m_3),\ m_{12} = m_8(m_1 + m_3),\ X_3 = \alpha.m_9 + m_4 m_{11}$
$Y_3 = \beta.m_{10} + m_5 m_{12},\ Z_3 = m_{11}(m_1 + m_3)$

Table 2. Addition law (Unif_Add).[13]

$m_1 = X_1.X_2,\ m_2 = Y_1.Y_2,\ m_3 = Z_1.Z_2,\ m_4 = (X_1 + Z_1)(X_2 + Z_2)$
$m_5 = (Y_1 + Z_1)(Y_2 + Z_2),\ m_6 = m_1.m_3,\ m_7 = m_2.m_3,\ m_8 = m_1.m_2 + (m_3)^2$
$m_9 = m_6(m_2 + m_3)^2,\ m_{10} = m_7(m_1 + m_3)^2,\ m_{11} = m_8(m_2 + m_3)$
$X_3 = \alpha.m_9 + m_4 m_{11} + Z_3,\ Y_3 = \beta.m_{10} + m_5 m_8(m_1 + m_3) + Z_3$
$Z_3 = m_{11}(m_1 + m_3)$

Based on this observation, they constructed another Unif_Add which provides prevention from the SCAs and SPA attacks and is presented in Table 2. For further mathematical formulations, interested readers can consult Refs. 7 and 13.

Where X_3, Y_3 and Z_3 are the projective points on the defined Huff curve, as shown in Tables 1 and 2. Moreover, 'α' and 'β' are the curve constants and they can be computed as, $\alpha = (a + b)/b$ and $\beta = (a + b)/a$. In this work, pre-computed curve constants i.e., 'α' and 'β' have been used to increase the throughput. This work utilizes the '*Unif_Add*' formulations, presented in Table 2, which provide immunity against the SCA and SPA attacks at algorithmic level.

2.3. *PM on BHC*

PM is defined by the following equation:

$$Q = k.P = k(P + P + \cdots + P)\,. \tag{2}$$

Equation (2) is a basic equation for PM, where 'Q' is a resultant point on the curve, 'k' is a scalar multiplier and 'P' is an initial point on the curve. To compute PM operation, we have used the following Double and Add algorithm (Algorithm 1) as in Ref. 16. A good comparative review over different PM algorithms is presented in Ref. 17.

Algorithm 1. Double and Add algorithm[16] over $GF(2^{\mathrm{m}})$.

Input: $k = (k_{n-1}, \ldots, k_1 k_0)$ with $k_{n-1} = 1, P = (x, y) \in GF(2^{\mathrm{m}})$
Output: $Q = k.P$

Initializations: $X_1 = x_p, Y_1 = Y_2 = y_p, Z_1 = 1, X_2 = x_{p^4} + b, Z_2 = x_{p^2}$

Point Multiplication:
for (i from $n - 2$ down to 0) do
  $Q = \text{Unif\_Add}(Q, Q)$
  if ($k_i = 1$) then
    $Q = \text{Unif\_Add}(P, Q)$
  end if
end for

Reconversion: $x_q = X_2/Z_2, y_q = Y_2/Z_{2^2}$

Whereas the "Unif_Add" represents the set of equations as presented in Tables 1 and 2 for point double and add to compute PM.

3. Proposed Crypto Processor for BHC

The hardware architecture of the proposed crypto processor for BHC is shown in Fig. 1. The placement of pipeline registers is not shown in the figure, however, it is discussed later in this section. The initial curve parameters for the proposed design have been selected from National Institute of Standards and Technology (NIST).[18]

3.1. *Memory unit*

The memory unit (MU) of the proposed dedicated 2-stage pipelined design contains a 16 locations register file with data size of 'm' bits. The main purpose of this unit is to store different parameters such as $X_1, X_2, Y_1, Y_2, Z_1, Z_2$ and the intermediate results $m_1, m_2, m_3, m_4, m_5, m_6, T_1, T_2, T_3$ and T_4, while implementing Algorithm-1 for the specified curve. It constitutes of two multiplexers (Mux M1 and Mux M2) which are used to read operands (OP1 and OP_2) from the MU by using the corresponding control signals (C1 and C2) and a single de-multiplexer (Demux) which is used to update the MU contents (Mplex_out) with a specified register address (C3).

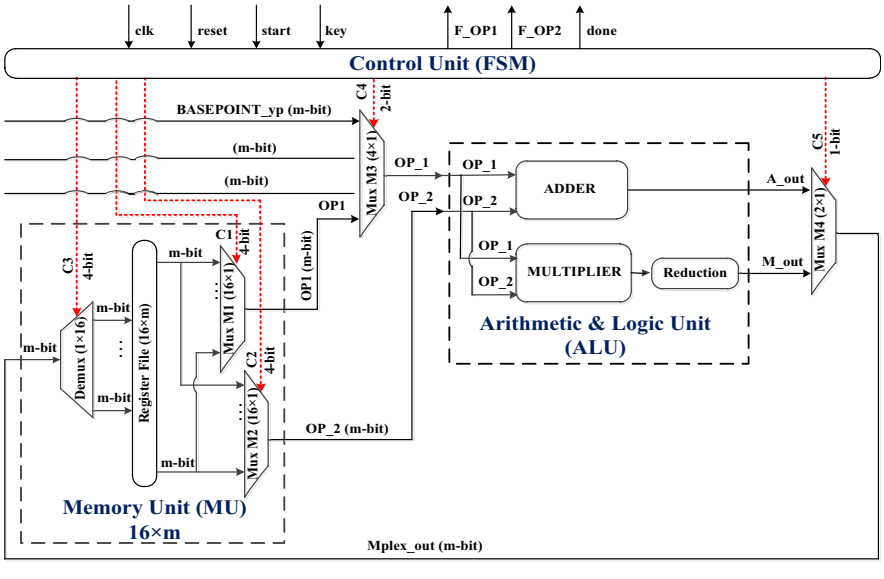

Fig. 1. BHC design for PM.

3.2. *Arithmetic and logic unit*

The arithmetic and logic unit (ALU) contains adder, multiplier and reduction units, as shown in Fig. 1. The addition unit is implemented through 'm' number of bit-wise exclusive-or (XOR) gates.

In order to perform multiplication of two 'm' bit polynomials ($A(x)$ and $B(x)$) over $GF(2^{163})$, a serial digit level multiplier with digit size of 41 bits is implemented in Ref. 19 where each multiplication requires 4 clock cycles (CC). In this paper, we have implemented the parallel Least Significant Digit (LSD) multiplier with a digit size of $s = 32$ bits, as shown in Fig. 2. The digits with $s = 32$ bits of polynomial $B(x)$ is created (i.e., B1–B8) and then parallel multiplication of each 's' bit digit with 'm' bit polynomial ($A(x)$) is performed by generating partial products. To compute FF multiplication operation over $GF(2^{163})$, a total of six digits are required. Out of these six digits, five digits are with 32 bit size whereas one digit is with three bit size. Similarly, for $GF(2^{233})$ a total of eight digits are required. Out of these eight digits, seven digits (B1–B7) are with 32 bit size whereas one digit (B8) is with nine bit size, as shown in Fig. 2. Parallel multiplication of each B1–B8 digit with an 'm' bit polynomial $A(x)$ results in '$s + m - 1$' bits of polynomials and these resultant polynomials are represented as D1–D8, as shown in Fig. 2. Once multiplication of each 's' bit digit with an 'm' bit polynomial is completed, the final resultant polynomial of size $2 \times m - 1$ bit is created by XOR and shift operations of D1–D8. In this paper, NIST reduction algorithms over $GF(2^{163})$ and $GF(2^{233})$ are implemented, as described in Algorithm 2.41 and Algorithm 2.42 of Ref. 1.

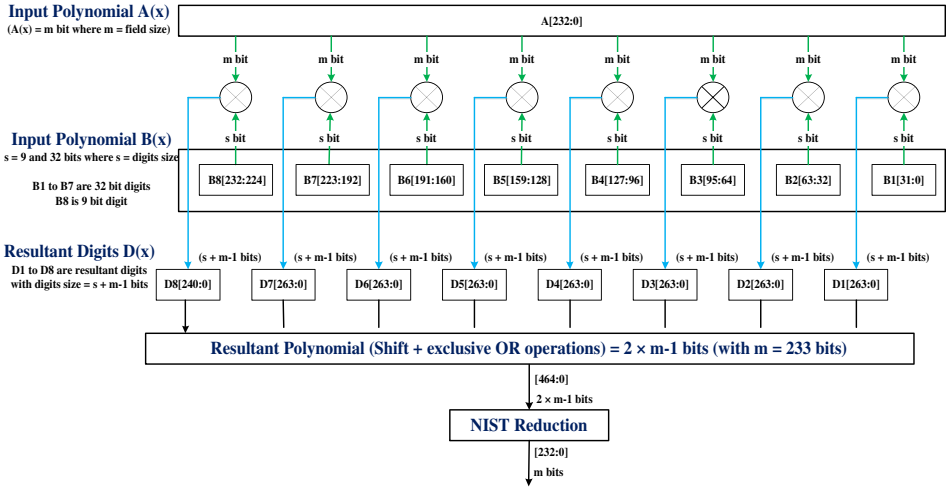

Fig. 2. Parallel LSD multiplier.

As a multiplication along with reduction operation uses one CC, in order to reduce hardware resources, squaring instructions (presented in Table 3) are performed by providing the same inputs to the parallel LSD multiplier. Moreover, inversion is achieved by implementing the Itoh–Tsujii inversion algorithm.[20] To compute the inversion operation, Itoh–Tsujii requires nine field multiplications when implementing over $GF(2^{163})$,[21] while 10 field multiplications are required over $GF(2^{233})$ field.[22]

3.3. *Routing networks*

The proposed design constitutes of two routing networks, the first one (Mux M3) is from the input curve parameters and MU to the input of ALU and second one (Mux M4) is from the output of ALU to the input of MU. In order to perform routing operations, the corresponding control signals are C4 and C5, as shown in Fig. 1.

3.4. *Choices for pipeline inclusion*

In order to achieve an optimal throughput, the first action was to explore the choices of pipelining. Hence, the circuit was divided into three parts, i.e., (1) circuit made through M1, M2 and M3 used for read operation (R), (2) ALU alone for execution (E) and (3) combination of M4 and Demux for write back (WB) operation. With this division, we have three possible solutions i.e., no pipeline registers in the architecture, hence, R, E and WB in a single CC as carried out in Refs. 4 and 13. Secondly, using the registers at the input of ALU (2-stage pipelined architecture) i.e., R in one CC and E & WB in second CC. Finally, using registers both at the input and output of the ALU (3-stage pipelined architecture), causing R, E and WB in three separate cycles.

The unified algorithm of Table 2, requires a total of $23 \times m$ storage elements in MU for intermediate results i.e., for $X_1, X_2, Y_1, Y_2, Z_1, Z_2, T_1$ to T_3, m_1 to m_{11}, X_3, Y_3 and Z_3 as shown in Table 3 (column 2). It requires a total of 31 CC for each unified PA and point double when R, E and WB is performed in one clock cycle i.e., no pipelining of the architecture (column 1 of Table 3). Moreover, the unified algorithm of Table 2 in its given form, can cause read after write (RAW) hazard in the context of pipelining, as shown in Table 3 (column 3) e.g., the first RAW hazard occurs during the execution of inst_5 when a write operation is performed on T_2 and in the very next cycle the value of T_2 is read. In the 2-stage pipelined architecture context, one cycle delay will be required to execute inst_6 as the new value of T_2 will take two cycles for its computation. By considering the RAW hazards presented above, the proposed sequences of instructions for a 2-stage pipeline architecture (R and [E, WB] in two different cycles) are shown in Table 3 (column 4). For a 2-stage pipeline architecture, it requires a total of 37 CC for each PA and point double. Similarly, for a 3-stage pipeline architecture (R, E and WB in separate cycles), a total of 43 CC are required for each PA and point double.

Table 3. Proposed scheduling of unified addition law for PM on BHC.

Clock cycles	Scheduling according to sequence of Table 2 ($23 \times m$ size of MU)			Proposed scheduling using $16 \times m$ size of MU		
	Inst_i/Operation	Hazard	Scheduling for 2-stage pipelined architecture	Inst_i/Operation	Hazard	Scheduling for 2-stage pipelined architecture
1	$\text{Inst}_1/m_1 = X_1 \times X_2$	—	Inst_1[R]	$\text{Inst}_1/m_1 = X_1 \times X_2$	—	Inst_1[R]
2	$\text{Inst}_2/m_2 = Y_1 \times Y_2$	—	Inst_1[E,WB], Inst_2[R]	$\text{Inst}_2/m_2 = Y_1 \times Y_2$	—	Inst_1[E,WB], Inst_2[R]
3	$\text{Inst}_3/m_3 = Z_1 \times Z_2$	—	Inst_2[E,WB], Inst_3[R]	$\text{Inst}_3/m_3 = Z_1 \times Z_2$	—	Inst_2[E,WB], Inst_3[R]
4	$\text{Inst}_4/T_1 = (X_1 + Z_1)$	—	Inst_3[E,WB], Inst_4[R]	$\text{Inst}_4/T_1 = (X_1 + Z_1)$	—	Inst_3[E,WB], Inst_4[R]
5	$\text{Inst}_5/T_2 = (X_2 + Z_2)$	—	Inst_4[E,WB], Inst_5[R]	$\text{Inst}_5/T_2 = (X_2 + Z_2)$	—	Inst_4[E,WB], Inst_5[R]
6	$\text{Inst}_6/m_4 = T_1 \times T_2$	RAW:T_2	Inst_5[E,WB],Inst_{10}[R]	$\text{Inst}_6/m_6 = m_1 \times m_3$	—	Inst_5[E,WB], Inst_6[R]
7	$\text{Inst}_7/T_1 = (Y_1 + Z_1)$	—	Inst_{10}[E,WB], Inst_6[R]	$\text{Inst}_7/m_4 = T_1 \times T_2$	—	Inst_6[E,WB], Inst_7[R]
8	$\text{Inst}_8/T_2 = (Y_2 + Z_2)$	—	Inst_6[E,WB], Inst_7[R]	$\text{Inst}_8/T_1 = (Y_1 + Z_1)$	—	Inst_7[E,WB], Inst_8[R]
9	$\text{Inst}_9/m_5 = T_1 \times T_2$	RAW:T_2	Inst_7[E,WB], Inst_8[R]	$\text{Inst}_9/T_2 = (Y_2 + Z_2)$	—	Inst_8[E,WB], Inst_9[R]
10	$\text{Inst}_{10}/m_6 = m_1 \times m_3$	—	Inst_8[E,WB], Inst_{11}[R]	$\text{Inst}_{10}/x_q = m_2 + m_3$	—	Inst_9[E,WB], Inst_{10}[R]
11	$\text{Inst}_{11}/m_7 = m_2 \times m_3$	—	Inst_{11}[E,WB], Inst_9[R]	$\text{Inst}_{11}/m_5 = T_1 \times T_2$	—	Inst_{10}[E,WB], Inst_{11}[R]
12	$\text{Inst}_{12}/T_1 = m_3^2$	—	Inst_9[E,WB], Inst_{12}[R]	$\text{Inst}_{12}/T_1 = m_2 \times m_3$	—	Inst_{11}[E,WB], Inst_{12}[R]
13	$\text{Inst}_{13}/T_2 = m_1 \times m_2$	—	Inst_{12}[E,WB], Inst_{13}[R]	$\text{Inst}_{13}/T_2 = m_1 + m_3$	—	Inst_{12}[E,WB], Inst_{13}[R]
14	$\text{Inst}_{14}/m_8 = T_1 + T_2$	RAW:T_2	Inst_{13}[E,WB]	$\text{Inst}_{14}/T_4 = x_q^2$	—	Inst_{13}[E,WB], Inst_{14}[R]
15	$\text{Inst}_{15}/T_1 = m_2 + m_3$	—	Inst_{14}[R]	$\text{Inst}_{15}/T_2 = T_2^2$	—	Inst_{14}[E,WB], Inst_{15}[R]
16	$\text{Inst}_{16}/T_2 = T_1^2$	RAW:T_1	Inst_{14}[E,WB], Inst_{15}[R]	$\text{Inst}_{16}/T_3 = m_3^2$	—	Inst_{15}[E,WB], Inst_{16}[R]
17	$\text{Inst}_{17}/m_9 = T_2 \times m_6$	RAW:T_2	Inst_{15}[E,WB]	$\text{Inst}_{17}/T_1 = T_1 \times T_2$	—	Inst_{16}[E,WB], Inst_{17}[R]
18	$\text{Inst}_{18}/T_2 = m_1 + m_3$	—	Inst_{16}[R]	$\text{Inst}_{18}/T_2 = m_1 \times m_2$	—	Inst_{17}[E,WB], Inst_{18}[R]
19	$\text{Inst}_{19}/T_3 = T_2^2$	RAW:T_2	Inst_{16}[E,WB], Inst_{21}[R]	$\text{Inst}_{19}/X_2 = \beta \times T_1$	—	Inst_{18}[E,WB], Inst_{19}[R]
20	$\text{Inst}_{20}/m_{10} = m_7 \times T_3$	RAW:T_3	Inst_{21}[E,WB], Inst_{17}[R]	$\text{Inst}_{20}/T_2 = (T_2 + T_3)$	—	Inst_{19}[E,WB], Inst_{20}[R]
21	$\text{Inst}_{21}/m_{11} = m_8 \times T_1$	—	Inst_{17}[E,WB], Inst_{18}[R]	$\text{Inst}_{21}/T_3 = m_6 \times T_4$	—	Inst_{20}[E,WB], Inst_{21}[R]
22	$\text{Inst}_{22}/Z_3 = m_{11} \times T_2$	RAW:m_{11}	Inst_{18}[E,WB], Inst_{23}[R]	$\text{Inst}_{22}/T_4 = T_2 \times x_q$	—	Inst_{21}[E,WB], Inst_{22}[R]
23	$\text{Inst}_{23}/T_1 = \alpha \times m_9$	—	Inst_{23}[E,WB], Inst_{19}[R]	$\text{Inst}_{23}/x_q = \alpha \times T_3$	—	Inst_{22}[E,WB], Inst_{23}[R]
24	$\text{Inst}_{24}/T_3 = m_4 \times m_{11}$	—	Inst_{19}[E,WB], Inst_{22}[R]	$\text{Inst}_{24}/y_q = m_4 \times T_4$	—	Inst_{23}[E,WB], Inst_{24}[R]
25	$\text{Inst}_{25}/X_3 = T_1 + T_3$	RAW:T_3	Inst_{22}[E,WB], Inst_{20}[R]	$\text{Inst}_{25}/m_6 = m_5 \times T_2$	—	Inst_{24}[E,WB], Inst_{25}[R]
26	$\text{Inst}_{26}/X_3 = X_3 + Z_3$	RAW:X_3	Inst_{20}[E,WB], Inst_{24}[R]	$\text{Inst}_{26}/m_4 = x_q + y_q$	—	Inst_{25}[E,WB], Inst_{26}[R]
27	$\text{Inst}_{27}/T_1 = \beta \times m_{10}$	—	Inst_{24}[E,WB]	$\text{Inst}_{27}/x_q = m_1 + m_3$	—	Inst_{26}[E,WB], Inst_{27}[R]
28	$\text{Inst}_{28}/T_3 = m_5 \times m_8$	—	Inst_{25}[R]	$\text{Inst}_{28}/X_2 = m_4 + Z_2$	—	Inst_{27}[E,WB], Inst_{28}[R]
29	$\text{Inst}_{29}/Y_3 = T_3 \times T_2$	RAW:T_3	Inst_{25}[E,WB], Inst_{27}[R]	$\text{Inst}_{29}/m_2 = m_6 \times x_q$	—	Inst_{28}[E,WB], Inst_{29}[R]
30	$\text{Inst}_{30}/Y_3 = T_1 + Y_3$	RAW:Y_3	Inst_{27}[E,WB], Inst_{28}[R]	$\text{Inst}_{30}/Z_2 = x_q \times T_4$	—	Inst_{29}[E,WB], Inst_{30}[R]
31	$\text{Inst}_{31}/Y_3 = Y_3 + Z_3$	RAW:Y_3	Inst_{28}[E,WB], Inst_{26}[R]	$\text{Inst}_{31}/m_5 = m_2 + T_1$	—	Inst_{30}[E,WB], Inst_{31}[R]
32	—	—	Inst_{26}[E,WB], Inst_{29}[R]	$\text{Inst}_{32}/y_q = m_5 + Z_2$	RAW: m_5	Inst_{31}[E,WB]
33	—	—	Inst_{29}[E,WB]	—	—	Inst_{32}[R]
34	—	—	Inst_{30}[R]	—	—	Inst_{32}[E,WB]
35	—	—	Inst_{30}[E,WB]	—	—	—
36	—	—	Inst_{31}[R]	—	—	—
37	—	—	Inst_{31}[E,WB]	—	—	—

Further rescheduling of instructions can be performed in order to optimize the solution in terms of less used hardware resources and less number of instructions to achieve a PM. Hence, the unified algorithm of Table 2, is presented according to the proposed $16 \times m$ size of MU as shown in Table 3 (column 5). It requires a total of 32 CC when R, E and WB takes one CC. In the context of pipelining, the presented algorithm (unified algorithm of Table 2) for $16 \times m$ size of MU can cause only single RAW hazard as shown in Table 3 (column 6). By considering the RAW hazard (column 6 of Table 3), the corresponding sequences of instructions for a 2-stage pipeline architecture are shown in Table 3 (column 7) and it requires a total of 34 CC

for each PA and point double. Furthermore, 36 CC are required for each PA and point double when considering a 3-stage pipeline architecture.

For both $16 \times m$ and $23 \times m$ sizes of memory units, the 2-stage pipelined architecture gives better throughput/area as compared to the no pipelined architecture (R, E and WB in 3 separate CC). Addition of a third pipeline stage for WB is not efficient as it adds more CC due to RAW hazards. Moreover, addition of registers at the output of the ALU further reduces the overall throughput/area ratio.

3.5. *Control unit* (*CU*)

FSM based efficient control unit is designed in this work to perform control functionalities. The used control signals are shown as dotted lines with red color in Fig. 1, whereas the corresponding FSM generating these signals is shown in Fig. 3.

In order to implement Algorithm 1 for BHC, the FSM consists of 127 states for a 2-stage pipelined architecture, as shown in Fig. 3. St: 0 is an idle state, while during St: 1 to St: 6, control signals for initializations part of Algorithm 1 (i.e., affine to projective conversion) are generated. In order to implement the PM step of Algorithm 1, for each inspected bit of key equals to zero ($k_i = 0$) and $count! = m - 1$, during states St: 7 to St: 40, PD control signals are generated for the proposed scheduling, as presented in Table 3. Initially, the count is set to '0' and is used to count the number of points on the specified curve. When the inspected bit of key is one ($k_i = 1$) and $count! = m - 1$, then PA is also performed followed by PD from St: 41 to St: 74. St: 40 and St: 74, are also responsible to check the status of the count

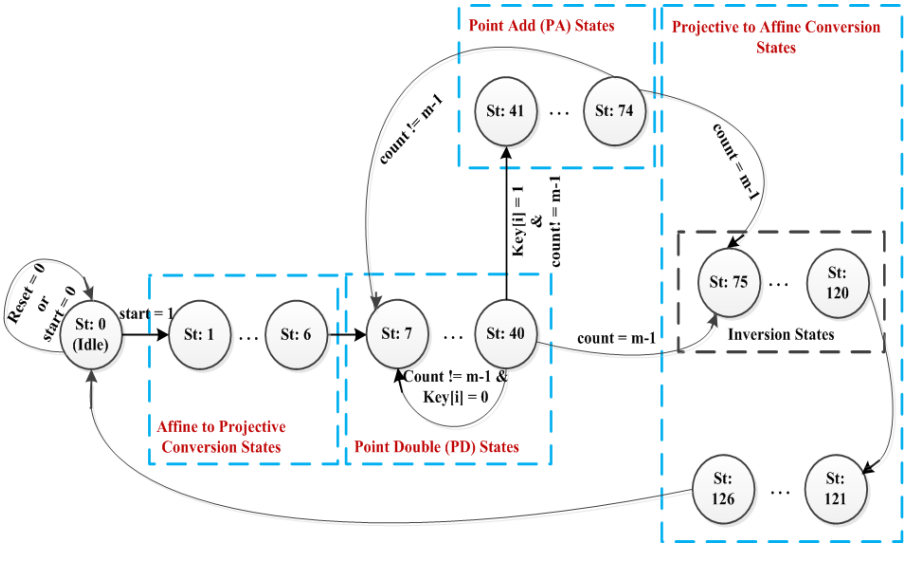

Fig. 3. Finite state machine for BHC.

Table 4. CCs information for $GF(2^{163})$ and $GF(2^{233})$.

	For $GF(2^{163})$		For $GF(2^{233})$	
Operations	Estimated	Behavioral simulated	Estimated	Behavioral simulated
Initializations	6	6	6	6
Unif_Add	8262	8262	11832	11832
FF Inversion	502	502	709	709
Reconversion	6	6	6	6
Total CC	8776	8776	12553	12553

signal. Once $count = m - 1$, then the control unit generates signals for the coordinate reconversion step of Algorithm 1. Finally, during St: 75 to St: 126, the coordinate reconversion step of Algorithm 1 including FF inversion (St: 75 to St: 120) is performed, as shown in Fig. 3.

To compute PM, the initializations step of Algorithm 1 requires six CC. For the most time consuming part, i.e., for each unified addition (PA or PD), the proposed design requires 34 CC. Finally, the reconversion step requires FF inversion (inv) + six CC. To calculate the total number of CC, the worst case scenario as presented in Ref. 4 has been considered i.e., key contains alternate $1's$ and $0's$. Consequently, the total number of CC for the proposed architecture can be calculated by using the expression given in Eq. (3).

$$6 + 34(m-1) + 34\left(\frac{m-1}{2}\right) + \text{inv} + 6\,. \tag{3}$$

The estimated CC using Eq. (3) and the exact CC, which are obtained through behavioral simulations, are provided in Table 4.

4. Results and Performance Estimation

4.1. *Synthesis and place and route results*

For our proposed 2-stage pipeline architecture over $GF(2^{163})$ and $GF(2^{233})$, two Verilog HDL models are created. In first step, to perform verifications of the proposed Verilog (HDL) design, its behavioral results are compared with its C-based functional model. The proposed designs are then synthesized on Virtex 4 (xc4vfx140-11ff1517), Virtex 5 (xc5vfx130t-3ff1738), Virtex 6 (xc6vlx550t-2ff1760) and Virtex 7 (xc7vx690t-3ffg1930) devices from Xilinx using ISE (14.7) design suite. The results after synthesis and place and route for our proposed designs are tabulated in Table 5. From the two designs, one can also estimate the overall additional hardware cost while moving from $GF(2^{163})$ to $GF(2^{233})$.

In Table 5, the respective curves are shown in the first column whereas the considered FPGA devices are mentioned in column 2. Third column presents the utilized FPGA area (slices) used. The number of CC to perform one PM operation,

Table 5. Implementation results for $GF(2^{163})$ and $GF(2^{233})$ on Xilinx Virtex 4, 5, 6 and 7.

				Results after synthesis			Results after post place and route		
Curve	Platform	Slices	Clock cycles	Freq. (MHz)	$k.P$ (μs)	$\frac{\left(\frac{10^6}{s}\right)}{\text{Slices}}$	Freq. (MHz)	$k.P$ (μs)	$\frac{\left(\frac{10^6}{s}\right)}{\text{Slices}}$
				Results over $GF(2^{163})$					
BHC	XC4VFX140	11567	8776	173	50.7	1.70	109	80.5	1.07
	XC5VFX130t	4430	8776	253	34.6	6.52	121	72.5	3.11
	XC6VLX550t	4024	8776	301	29.1	8.53	162	54.1	4.59
	XC7VX690t	3880	8776	377	23.2	11.10	201	43.6	5.91
				Results over $GF(2^{233})$					
BHC	XC4VFX140	17393	12553	162	77.4	0.74	101	124.2	0.46
	XC5VFX130t	6676	12553	239	52.5	2.85	116	108.2	1.38
	XC6VLX550t	7681	12553	296	42.4	3.07	148	84.8	1.53
	XC7VX690t	6342	12553	369	34.0	4.63	191	65.7	2.39

computed through Eq. (3), are shown in column 4. Column 5 (results after synthesis) and column 6 (results after post place and route), are further sub-partitioned into three subcolumns to present the information about the achieved operational frequency (Freq (MHz), time for one PM i.e., $k\,P.$ (μs) and throughput/area ($10^6/k\,P(s)$/slices) ratio).

The time for one PM operation is computed by dividing the CC with operational frequency and is calculated using the below expression

$$k\,P(\mu\text{s}) = \frac{\text{Number of Clock Cycles}}{\text{Freq (MHz)}}\,. \tag{4}$$

Finally, the throughput/slices ($10^6/kP(s)$/slices) ratio is obtained by using Eq. (5) which is considered as a metric to analyze the efficiency of the architecture.

$$\frac{\text{throughput}}{\text{slices}} = \frac{\frac{1}{k.P\times 10^{-6}(s)}}{\text{slices}} = \frac{\frac{1}{k.P(s)}\times 10^6}{\text{slices}}\,. \tag{5}$$

4.2. *Performance results*

For our designs, the number of CC are 8776 and 12553 for $GF(2^{163})$ and $GF(2^{233})$ respectively. Consider the first case of PM with $GF(2^{163})$, implementation on XC4VFX140. In this case the maximum after synthesis operational frequency is 173 MHz which results in 50.7 μs to compute on PM. Finally, by using the expression in Ref. 5, the throughput/area ratio becomes 1.70. On the other hand, if after post place and route frequency is used, this ratio reduces to 1.07. The performances for other cases are computed in the same way. The best results are achieved for the Virtex 7 (xc7vx690t-3ffg1930) device, where the throughput/slices metrics are equal to 11.10 and 4.63 for $GF(2^{163})$ and $GF(2^{233})$, respectively.

Table 6. Comparison with state-of-the-art for BHC over $GF(2^{233})$.

Source	Platform	Slices	Freq. (MHz)	Clock cycles	$k.P$ (μs)	$\frac{\left(\frac{10^6}{s}\right)}{\text{Slices}}$
Ref. 4	Virtex 4	20437	81	5913	73	0.67
Proposed	Virtex 4	17393	162	12553	77	0.74

Table 7. Comparison with state-of-the-art for BHC without inversion over $GF(2^{233})$.

Source	Platform	Slices	Freq. (MHz)	Clock cycles	$k.P$ (μs)	$\frac{\left(\frac{10^6}{s}\right)}{\text{Slices}}$
Ref. 13	Virtex 6	7150	172	7370	43	3.25
	Virtex 7	6032	183	7370	40	4.14
Proposed	Virtex 6	7681	296	11838	39	3.33
	Virtex 7	6342	369	11838	32	4.92

From the hardware implementation perspective, the result of comparison with the state-of-the-art is challenging due to the limited relevant published work. However, similar application is targeted in Refs. 4 and 13 over $GF(2^{233})$ only. To perform a fair comparison, we synthesized our design for those FPGAs which were used in Refs. 4 and 13. The work presented in Ref. 4 performs PM including reconversion, whereas in Ref. 13 reconversions were not considered while presenting their results. Consequently, we have performed our comparison with Ref. 4 including the reconversion steps of Algorithm 1 whereas comparison with Ref. 13 is considered without reconversion of Algorithm 1. The comparisons with state-of-the-art are presented in Tables 6 and 7.

As shown in Table 6, our proposed design consumes 17393 slices, which are 85% of the hardware resources used in Ref. 4 when synthesized over the same Virtex 4 device. This is due to the use of the single 'm' bit adder in the data path whereas the work presented in Ref. 4 utilizes five 'm' bit adders in the data path. Moreover, the work presented in Ref. 4 uses a total of seven multiplexers in the data path. Out of these seven multiplexers, two 8:1 multiplexers are used on the inputs of the multiplier. Other two 4:1 multiplexers are used for fetching the initial curve parameters and precomputed values ('α' and 'β'), and finally, three 9:1 multiplexers are used for fetching data from the register file. However, we have used only four multiplexers in the data path. Out of these four multiplexers, two 16:1 multiplexers are used for fetching register contents from MU whereas a single 4:1 multiplexer is used for fetching the initial curve parameters and precomputed values ('α' and 'β'). Finally, the single 2:1 multiplexer is used to update the MU contents.

On the other hand, in the data path of the architecture in Ref. 4, firstly multiple operators and multiplexer are connected without the pipeline registers. Due to this fact they achieve a maximum operational frequency of 81 MHz which is comparably 50% lesser than our work. Although, the reported time to perform PM in Ref. 4 is

4 μs less than the time consumed by our solution, the overall throughput/slices ratio of our work is 10% higher than the work in Ref. 4.

Another FPGA based solution over newer technologies (Virtex 6 and 7) for BHC has been presented in Ref. 13, as shown in Table 7. Their work utilizes an area of 6032 slices which is 5% lesser than our design (6342) over Virtex 7. They attain a maximum operational frequency of 183 MHz which is almost half when comparing with this work (369 MHz) using the same technology. To compute each PM, their design requires 40 μs which is comparably 20% higher than the proposed solution (32 μs) over Virtex 7. Finally, they achieved maximum throughput/area figures of 4.14, hence, our solution gives 20% higher throughout/area ratio.

Acknowledgments

This project is funded by NSTIP (National Science Technology, Innovative Plan), Saudi Arabia. We acknowledge the support of KACST (King Abdul-Aziz City for Science and Technology) and STU (Science and Technology Unit) Makkah.

References

1. D. Hankerson, A. Menezes and S. Vanstone, *Guide to Elliptic Curve Cryptography* (Springer-Verlag, New York, 2004).
2. N. Koblitz, Elliptic curve cryptosystems, *Math. Comp.* **48** (1987) 203–209.
3. V. Miller, Use of elliptic curves in cryptography, in *Proc. CRYPTO 1985*, H. C. Williams ed. Vol. 218, Lecture Notes in Computer Science (1986), pp. 417–426.
4. A. Chatterjee and I. Sengupta, High-speed unified Elliptic curve cryptosystem on FPGAs using binary huff curves, *Progress in VLSI Design and Test (VDAT)*, Lecture Notes in Computer Science, Vol. 7373 (2012), pp. 243–251.
5. T. Izu and T. Takagi, Exceptional procedure attack on Elliptic curve cryptosystems, in Y. G. Desmedt eds. *Public Key Cryptography — PKC 2003*, PKC 2003. Lecture Notes in Computer Science, Vol. 2567 (Springer, Berlin, Heidelberg, 2002), pp. 224–239.
6. D. J. Bernstein, T. Lange and R. R. Farashahi, Binary edwards curves, 2008.
7. J. Devigne and M. Joye, Binary huff curves, *CT-RSA 2011*, Lecture Notes in Computer Science, Vol. 6558 (2011), pp. 340–355.
8. Z. Dyka and P. Lagendorfer, Improving the security of wireless sensor networks by protecting the sensor nodes against side channel attacks, *Wireless Networks and Security*, Signals and Communication Technology (2013), pp. 303–328.
9. B. Harris, Security intelligence, Last accessed: February 2016, https://securityintelligence.com/platform-as-a-service-paas-cloud-side-channel-attacks-part-i/.
10. T. Kasper, D. Oswald and C. Paar, Side-channel analysis of cryptographic RFIDs with analog demodulation, *RFID. Security and Privacy*, Vol. 7055 of the series Lecture Notes in Computer Science (LNCS), 2012, pp. 61–77.
11. T. H. Kim, T. Takagi, D. Han, H. W. Kim and J. Lim, Side channel attacks and countermeasures on pairing based cryptosystems over binary fields, *Cryptology and Network Security*, Lecture Notes in Computer Science, Vol. 4301 (2006), pp. 168–181.
12. M. Imran and M. Rashid, Architectural review of polynomial basis finite field multipliers over $GF(2^m)$, *IEEE International Conference on Communication, Computing and Digital Systems*, Islamabad, Pakistan, March 2017, pp. 8–9.

13. S. Ghosh, A. Kumar, A. Das and I. Verbauwhede, On the implementation of unified arithmetic on binary huff curves, *Cryptographic Hardware and Embedded Systems-CHES 2013*, Lecture Notes in Computer Science, Vol. 8086 (2013), pp. 349–364.
14. D. L. Huff, A probabilistic analysis of shopping center trade areas, *Land Econ.* **39** (1963) 81–90.
15. M. Joye, M. Tibouchi and D. Vergnaud, Huff's model for elliptic curve, *Algorithmic Number Theory (ANTS-IX)*, Lecture Notes in Computer Science, Vol. 6197 (2010), pp. 234–250.
16. C. Bach, Elliptic Curves–Double and Add Algorithm, Last accessed: January 2016, http://hyperelliptic.blogspot.com/2009/06/double-and-add-algorithm.html.
17. M. Rashid, M. Imran and A. R. Jafri, Comparative analysis of flexible cryptographic implementations, *11th IEEE Int. Symp. Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC)*, Tallinn, 2016, pp. 1–6.
18. National Institute of Standards and Technology (NIST), Recommended Elliptic Curves for Federal Government Use (1999), http://csrc.nist.gov/CryptoToolkit/dss/ecdsa/NISTReCur.pdf.
19. Z. Khan and M. Benaissa, Throughput/area-efficient ECC processor using Montgomery point multiplication on FPGA, *IEEE Trans. Circuits Syst. II* **62** (2015) 1078–1082.
20. T. Itoh and S. Tsujii, A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases, *J. Inform. Comput.* **78** (1988) 171–177.
21. V. Dimitrov and K. Jarvinen, Another look at inversions over binary fields, 21st *IEEE Int. Symp. Computer Arithmetic (ARITH21)*, Austin, 2013, pp. 211–218.
22. L. Parrilla, A. Lloris, E. Castillo and A. Garcia, Minimum-clock-cycle Itoh-Tsujii algorithm hardware implementation for cryptography applications over GF(2^m) fields, *Electron. Lett.* **48** (2012) 1126–1128.