Hardware Implementation of Elliptic Curve Processor over $GF(p)$

Sıddıka Berna Örs, Lejla Batina, Bart Preneel

K.U. Leuven ESAT/COSIC
Kasteelpark Arenberg 10
B-3001 Leuven-Heverlee, Belgium
{Siddika.BernaOrs, Lejla.Batina, Bart.Preneel}@esat.kuleuven.ac.be

Abstract. This paper describes a hardware implementation of an arithmetic processor which is efficient for bit-lengths suitable for both commonly used types of Public Key Cryptography (PKC), i.e., Elliptic Curve (EC) and RSA Cryptosystems. The processor consists of special operational blocks for Montgomery Modular Multiplication, modular addition/substraction, EC Point doubling/addition, modular multiplicative inversion, EC point multiplier, projective to affine coordinates conversion and Montgomery to normal representation conversion.

1 Introduction

In this work we propose a processor for Elliptic Curve Cryptosystems (ECC) over $GF(p)$. The processor is divided into five levels. In the most highest level "main controller" (MC) stands which controls execution of a EC point multiplication algorithm [19]. Level 2 consists of "normal to Montgomery representation converter (NtoM)", "EC point multiplier (EPM)", "projective to affine coordinates converter (PtoA)" and "Montgomery to normal representation converter". There are "EC point multiplier (EPM)" and "projective to affine coordinates converter (PtoA)" in level 3. Montgomery modular multiplication Circuit (MMMC) and modular addition, substraction circuit (MASC) stand in level 4. In the last level, there is addition/substraction circuit (ASC). The operational blocks that are one higher level control the operational blocks in one lower level.

The previous works about hardware implementations of EC processor over $GF(p)$ are from Orlando and Paar [21] and Gura *et al.* [10].

The most time consuming, so critical operation for EC point multiplication is modular multiplication. In 1985 Montgomery introduced a new method for modular multiplication [20]. The approach of Montgomery avoids the time consuming trial division that is a common bottleneck of other algorithms. His method proved to be very efficient and is the basis of many implementations of modular multiplication, both in software and hardware. In this paper we look at a hardware implementation.

Efficient implementation of Montgomery modular multiplication (MMM) in hardware was considered by many authors [7, 13, 33, 22, 17, 24, 34, 27, 14, 25, 28].

A systolic array architecture is one possibility for implementations of public key cryptography in hardware. Various solutions for systolic arrays were proposed, for example [8, 12, 11, 32, 13, 15, 29, 35, 26, 3, 9, 31, 36, 4, 30, 2].

In this work we combine a systolic array architecture, which is assumed to be the best choice for hardware on current integrated circuits (ICs), with a MMM in Field Programmable Gate Array (FPGA). Similar work was done by Blum and Paar [3]. However, their solution is less efficient because they had to use an extra step in the main algorithm (MMM); this step was required because they do not use the optimal bound for the main parameter R (so-called Montgomery parameter). The implementation of RSA is based on modular exponentiation. This algorithm is usually repeating modular multiplication (MMM) around 1500 times (assuming balanced Hamming weight of the exponent) for 1024-bit operands. Therefore, the implementation [3] is far less efficient compared to implementation in this work.

The remainder of this paper is organized as follows. In Section 2, are explained the underlying methods invented by Montgomery in detail and we introduce common notation and parameters. We also give some comments on the bound condition for avoiding subtraction at the end of every multiplication introduced by Walter [37]. Section 3 describes a processor architecture and design steps as well as the results of implementation. Conclusions and benchmarks for future work conclude the paper.

2 Montgomery Modular Multiplication

For modular multiplication Montgomery's technique is chosen [20]. Montgomery multiplication is defined as follows:

$$Mont(x, y) = xyR^{-1} \mod N \tag{1}$$

For a word base $b = 2^\alpha$, R should be chosen such that $R = 2^r = (2^\alpha)^l > N$. There is a one-to-one correspondence between each element $x \in \mathbb{Z}_\mathbb{N}$ and its Montgomery representation $xR \bmod N$. This Montgomery representation allows very efficient modular arithmetic especially for multiplication. Montgomery's method for multiplying two integers x and y (called N-residues) modulo N, avoids division by N which is the most expensive operation in hardware. The method requires conversion of x and y to an N-residue domain and conversion of the calculation result back to $\mathbb{Z}_\mathbb{N}$. The procedure is as follows. To compute $Z = xyR \bmod N$, one first has to compute the Montgomery multiplication of x and $R^2 \bmod N$ to get $Z' = xR \bmod N$. $Mont(Z', y)$ gives the desired result. When computing the Montgomery product $T = Mont(x, y) = xyR^{-1} \bmod N$, the following procedure is performed [18]:

To avoid the subtraction in Step 7 of Algorithm 1 a bound for R is given as $4N < R$ by Walter [37] and Batina and Muurling [2] such that for inputs $X, Y < 2N$ the output is also bounded by $T < 2N$. We will use $4N < R = 2^{l+2}$, by taking $\alpha = 1$ for simplicity and making the iteration starting from Step 2 execute $l + 2$ times. So the algorithm we use is as follows:

Algorithm 1 Montgomery modular multiplication

```
Require: Integers $N = (n_{l-1} \cdots n_1 n_0)_{2^\alpha}$, $x = (x_{l-1} \cdots x_1 x_0)_{2^\alpha}$, $y = (y_{l-1} \cdots y_1 y_0)_{2^\alpha}$
    with $x \in [0, N-1], y \in [0, N-1]$, $R = (2^\alpha)^l$, $gcd(N, 2^\alpha) = 1$ and $N' = -N^{-1}$
    mod $R$ (Notation $T = (t_l t_{l-1} ... t_0)$)
Ensure: $xyR^{-1} \mod N$
 1: $T \leftarrow 0$
 2: for $i$ from 0 to $l-1$ do
 3:     $m_i \leftarrow (t_0 + x_i y_0) N' \bmod 2^\alpha$
 4:     $T \leftarrow (T + x_i y + m_i N)/2^\alpha$
 5: end for
 6: if $T \geq N$ then
 7:     $T \leftarrow T - N$
 8: end if
 9: Return (T)
```

Algorithm 2 Montgomery modular multiplication without final subtraction

```
Require: Integers $N = (n_{l-1} \cdots n_1 n_0)_2$, $x = (x_l \cdots x_1 x_0)_2$, $y = (y_l \cdots y_1 y_0)_2$ with
    $x \in [0, 2N-1], y \in [0, 2N-1]$, $R = 2^{l+2}$, $gcd(N, 2) = 1$ and $N' = -N^{-1} \mod R$
    (Notation $T = (t_l t_{l-1} ... t_0)$)
Ensure: $xyR^{-1} \mod 2N$
 1: $T \leftarrow 0$
 2: for $i$ from 0 to $l+1$ do
 3:     $m_i \leftarrow (t_0 + x_i y_0) N' \bmod 2$
 4:     $T \leftarrow (T + x_i y + m_i N)/2$
 5: end for
 6: Return (T)
```

All the operations will be done modulo $2N$ through EC point multiplication. The last step is to convert the Montgomery representation of coordinates of resulting point back to normal representation. This is done by calculating the Montgomery modular multiplication of the coordinates and 1, $Mont(xR, 1) = xRR^{-1} = x$. Batina and Muurling [2] gives the proof for $Mont(T, 1) \leq N$, if $0 \leq T < 2N$. The coordinates of points on an elliptic curve can not be 0, so $Mont(xR, 1) = x < N$.

3 Hardware Implementation

3.1 Design Overview

Elliptic Curve processor (ECP) can be divided into 5 levels hierarchically as shown in Figure 1.

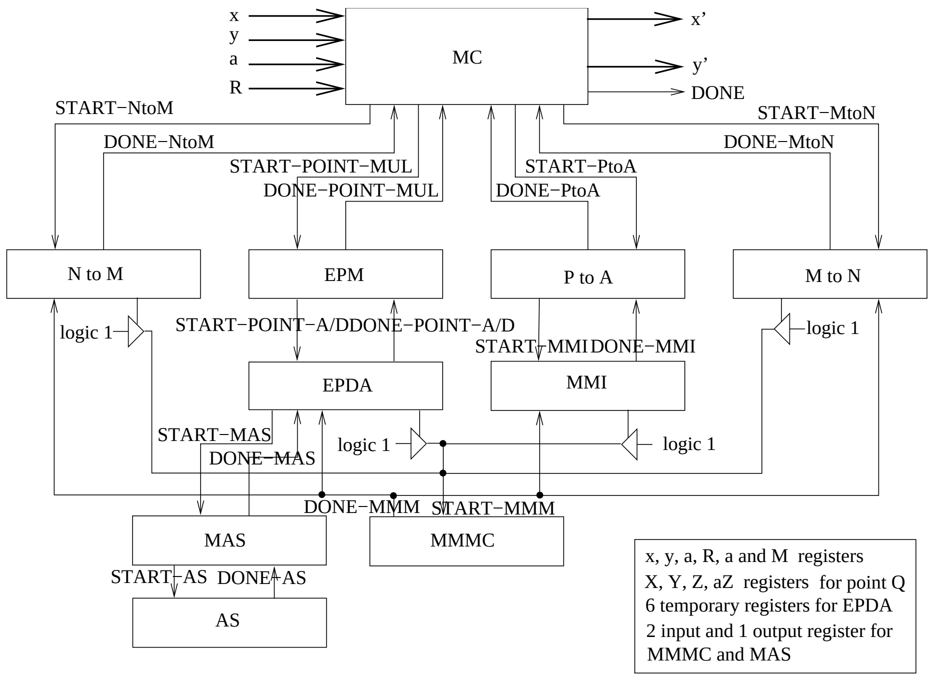


Fig. 1. EC point multiplier circuit block diagram

The operation blocks on each level from top to bottom are as follows:

– Level 1: Main Controller (MC)
– Level 2:

  1. Affine to projective coordinates converter (AtoP): $(x, y) \rightarrow (X, Y, Z, aZ^4)$ such that $X = x$, $Y = y$, $Z = 1$ and $aZ^4 = a$
  2. Normal to Montgomery representation converter (NtoM): $X \rightarrow XR \bmod M$, $Y \rightarrow YR \bmod M$, $1 \rightarrow R \bmod M$ and $a \rightarrow aR \bmod M$.
  3. EC point multiplier (EPM)
  4. Projective to affine coordinates converter (PtoA): $XR \bmod 2M \rightarrow XZ^{-2}R \bmod 2M$, $YR \bmod 2M \rightarrow YZ^{-3}R \bmod 2M$.
  5. Montgomery to normal representation converter (MtoN): $xR \bmod 2M \rightarrow x$, $yR \bmod 2M \rightarrow y$.
- Level 3:
  1. EC Point doubling, addition circuit (EPDA)
  2. Modular Multiplicative Inverter (MMI)
- Level 4:
  1. Montgomery Modular Multiplication Circuit (MMMC)
  2. Modular Addition, Substraction circuit (MASC)
- Level 5: Addition, Substraction circuit (ASC)

All blocks were designed as an separate circuit with own finite state machine and data path for simplicity. So all the blocks can be improved and tested themselves. The VHDL [1] code was written by describing the bit-length, N, of coordinates x and y of P and the bit-length, l of k as parameters. So this design is suitable for any N and l. In the following sections we have described the system using a top-down approach.

3.2 Main Controller

MC includes a finite state machine (FSM) with 5 states. MC waits in first (IDLE) state. After the START signal is set, MC reads the coordinates of P, x, y, modulus M, integer k, coefficient a and R^2 from input ports and writes them in register bank, commands to NtoM to start the operation by setting START-INI signal and goes to the second (S1) state.

MC waits in S1 state until the signal (DONE-INI) is set, which indicates the conversions from normal representation to Montgomery representation of the coordinates of P and a are done. After DONE-INI is set, MC writes (xR, yR, R, aR), to the registers for coordinates of point Q, commands to EPM to start point multiplication by setting START-PM signal and goes to the third (S2) state.

MC stays in S2 state waiting for the DONE-PM signal from EPM which indicates the point multiplication operation is finished, to be set. After DONE-PM signal is set, MC commands to PtoA to start conversion from projective to affine coordinates by setting START-PtoA signal and goes to the forth (S3) state.

MC stays in S3 state waiting for the DONE-PtoA signal from PtoA which indicates the conversion to affine coordinates is finished, to be set. After DONE-PtoA signal is set, MC commands to MtoN to start a conversion from Montgomery to normal representation by setting START-MtoN signal and goes to the fifth (S4) state.

MC stays in S4 state waiting for the DONE-MtoN signal from MtoN which indicates the conversion to normal representation is finished, to be set. After DONE-MtoN signal is set, MC writes resulting x' and y' to outputs, sets DONE signal which indicates to the host that a complete point multiplication operation is finished and the results are ready on output ports and goes back to IDLE state.

3.3 Normal to Montgomery representation converter

Conversion from normal representation to Montgomery representation is done by using Montgomery modular multiplier circuit (MMMC). The conversion of an integer x to Montgomery representation is done as $Mont(x, R^2) = xR^2R^{-1} \bmod M = xR \bmod M$. Multiplication by MMMC of two numbers that are in Montgomery representation will produce the Montgomery representation of product as $Mont(xR, yR) = xRyRR^{-1} \bmod M = xyR \bmod M$. Modular addition and subtraction of two numbers that are in Montgomery representation will produce the Montgomery representation of sum or difference as $xR \bmod M \pm yR \bmod M = (x \pm y)R \bmod M$. Because of previous relations Montgomery representation of the coordinates of P, the coefficient a and number 1 will be calculated in the beginning of point multiplication by NtoM circuit and all the operations during the execution of Algorithm 3 will be done in Montgomery representation.

NtoM includes a FSM with five states. NtoM waits in first (ini-IDLE) state until the signal (START-INI) from MC is set. After it is set, NtoM writes 1 and R^2 to the inputs of MMMC, commands to MMMC to start a MMM and goes to the second (ini-S1) state.

NtoM waits for the signal (DONE-MONT), that indicates MMM is finished, to be set. After DONE-MONT is set, NtoM writes the output of MMMC to the register for R, writes the new values to the inputs of MMMC, commands to MMMC to start a MMM and goes to the third (ini-S2) state. This way NtoM makes MMMC to execute 4 MMMs, $Mont(1, R^2) = R \bmod M$, $Mont(x, R^2) = xR \bmod M$, $Mont(y, R^2) = yR \bmod M$, $Mont(a, R^2) = aR \bmod M$. After DONE-MONT is set in last state, NtoM sets DONE-INI signal and goes back to ini-IDLE state.

3.4 EC Point Multiplier

EPM controls the execution of Algorithm 3. It includes a FSM with 4 states. The circuit stays in first (mul-IDLE) state until the START-PM signal from the MC is set. After START-PM signal is set, then EPM goes to the second (mul-S1) state.

In mul-S1 state EPM compares the value of the counter with l. If they are the same, EPM goes to the mul-IDLE state by setting the DONE-PM signal. If the value of the counter is smaller than l, then EPM commands to the counter to increment by 1, writes the coordinates of point Q to the inputs of EPDA, commands to EPDA by setting START-PAD signal to start a point double operation and goes to the third (mul-S2) state.

Algorithm 3 Elliptic Curve Point Multiplication

```
Require: EC point $P = (x, y)$, integer $k$, $0 < k < M$, $k = (k_{l-1}, k_{l-2}, \cdots, k_0)_2$,
    $k_{l-1} = 1$ and $M$
Ensure: $Q = (x', y')$
 1: $Q \leftarrow P$
 2: for $i$ from $l - 2$ downto 0 do
 3:     $Q \leftarrow 2Q$
 4:     if $k_i = 1$ then
 5:         $Q \leftarrow Q + P$
 6:     end if
 7: end for
 8: Return ($Q$)
```

EPM stays in mul-S2 state waiting for the signal DONE-PAD from EPDA, that indicates the point double operation is finished, to be set. After receiving this indication, EPM shifts the register in which k is stored, one bit left, checks if the most significant bit (MSB) of this register is 1. If it is 1 then writes the coordinates of point Q to the inputs of EPDA (the second point to be added is always P, so EPM writes only Q as a input), commands to EPDA to start a point addition operation and goes to the forth (mul-S3) state.

EPM stays in mul-S3 state waiting for DONE-PAD to be set. After it is set, EPM goes to the mul-S1 state back. So another iteration of Algorithm 3 starts.

3.5 Projective to affine coordinates converter

After finishing the point scalar multiplication the result point Q must be converted from J^m coordinates to affine coordinates. This is done as $\left(X, Y, Z, aZ^4\right) \rightarrow (x, y)$ such that $x = XZ^{-2}$ and $y = YZ^{-3}$.

PtoA includes a FSM with six states. PtoA waits in first (PtoA-IDLE) state until the signal (START-PtoA) from MC is set. After it is set, PtoA writes Z coordinate of point Q to the inputs of inverter, commands to inverter by setting START-INV signal to start a multiplicative inversion and goes to the second (PtoA-S1) state.

PtoA stays in PtoA-S1 state until DONE-INV signal, which indicates the multiplicative inversion is done, from inverter is set. After receiving this indication, PtoA writes the proper values to the inputs of MMMC, visits the other four states in the following order and after DONE-MONT signal from MMMC is set in PtoA-S5 state, PtoA goes back to PtoA-IDLE state.

- PtoA-S2: $Z^{-2}R = Mont(Z^{-1}R, Z^{-1}R)$
- PtoA-S3: $xR = XZ^{-2}R = Mont(XR, Z^{-2}R)$
- PtoA-S4: $Z^{-3}R = Mont(Z^{-1}R, Z^{-2}R)$
- PtoA-S5: $yR = YZ^{-3}R = Mont(YR, Z^{-3}R)$

3.6 Montgomery to normal representation converter

Because the coordinates of the product point must be in normal representation, as a last action a conversion from Montgomery representation to normal representation is needed. This operation is done just by executing two more times MMM operation with the inputs xR and 1, then yR and 1, because $x = Mont(xR, 1) = xRR^{-1}$, $y = Mont(yR, 1) = yRR^{-1}$.

MtoN includes three states and waits for the START-MtoN command from MC in the first (MtoN-IDLE) state. After receiving this command it executes two MMM as mentioned above and after receiving the signal DONE-MONT in the last state, sets the signal DONE-MtoN, which indicates the conversion is done and goes back to MtoN-IDLE state.

3.7 EC Point doubling, addition

Cohen *et al.* propose a modified Jacobian coordinates in order to obtain faster EC point doubling in [6]. They represent internally the Jacobian coordinates as a quadruple $\left(X, Y, Z, aZ^4\right)$. This representation is called modified Jacobian coordinate system and denoted by the authors as J^m. Let $P = \left(X_1, Y_1, Z_1, aZ_1^4\right)$, $Q = \left(X_2, Y_2, Z_2, aZ_2^4\right)$ and $P + Q = R = \left(X_3, Y_3, Z_3, aZ_3^4\right)$. The addition formulas in J^m are the following ($P \neq \pm Q$).

$$\begin{array}{l} U_1 = X_1 Z_2^2,\ U_2 = X_2 Z_1^2,\ S_1 = Y_1 Z_2^3,\ S_2 = Y_2 Z_1^3,\ H = U_2 - U_1,\ r = S_2 - S_1 \\ X_3 = -H^3 - 2U_1 H^2 + r^2,\ Y_3 = -S_1 H^3 + r\left(U_1 H^2 - X_3\right),\ Z_3 = Z_1 Z_2 H,\ aZ_3^4 = aZ_3^4 \end{array} \tag{2}$$

The doubling formulas in J^m are the following ($R = 2P$).

$$\begin{array}{l} S = 4X_1 Y_1^2,\ U = 8Y_1^4,\ M = 3X_1^2 + \left(aZ_1^4\right) \\ X_3 = -2S + M^2,\ Y_3 = M(S - X_3) - U,\ Z_3 = 2Y_1 Z_1,\ aZ_3^4 = 2U\left(aZ_1^4\right) \end{array} \tag{3}$$

When we convert the input point, P, from affine coordinates to projective coordinates we take Z as 1. The J^m representation of $P(x, y)$ is $(x, y, 1, a)$. During the execution of point multiplication one of the points to be added is always P. According to these properties we can take $Z_1 = 1$. Because there are both MMMC and modular addition/subtraction (MAS) circuits available, these operations can be executed in parallel. According to these properties, EC point addition can be realized by Algorithm 4 given below.

14 states and 6 temporary registers are needed for completing EC point addition algorithm. The multiplications and the squares are done by using MMMC. Addition, subtraction and double operations are done by using modular addition/subtraction circuit. Because completing one MAS operation takes shorter time than one MMM, the latency of one state is the same as one MMMC. So the total execution time of EC point addition is 14 MMMs.

Point doubling algorithm is given by Algorithm 5 below.

14 states and 6 temporary registers are needed for completing EC point doubling algorithm. The total execution time of the algorithm is 8 MMMs+6 MAS.

Algorithm 4 EC point addition

```
Require: $P_1 = (X_1, Y_1, 1, a)$, $P_2 = (X_2, Y_2, Z_2, aZ_2^4)$
Ensure: $P_1 + P_2 = P_3 = (X_3, Y_3, Z_3, aZ_3^4)$
  1.  $T_1 \leftarrow Z_2^2$
  2.  $T_2 \leftarrow xT_1$          $(U_1)$
  3.  $T_1 \leftarrow T_1 Z_2$                $T_3 \leftarrow X_2 - T_2$ $(H)$
  4.  $T_1 \leftarrow yT_1$          $(S_1)$
  5.  $T_4 \leftarrow T_3^2$                  $T_5 \leftarrow Y_2 - T_1$ $(r)$
  6.  $T_2 \leftarrow T_2 T_4$
  7.  $T_4 \leftarrow T_4 T_3$                $T_6 \leftarrow 2T_2$
  8.  $Z_3 \leftarrow Z_2 T_3$                $T_6 \leftarrow T_4 + T_6$
  9.  $T_3 \leftarrow T_5^2$
  10. $T_1 \leftarrow T_1 T_4$                $X_3 \leftarrow T_3 - T_6$
  11. $aZ_3^4 \leftarrow Z_3^2$               $T_2 \leftarrow T_2 - X_3$
  12. $T_3 \leftarrow T_5 T_2$
  13. $aZ_3^4 \leftarrow \left(aZ_3^4\right)^2$        $Y_3 \leftarrow T_3 - T_1$
  14. $aZ_3^4 \leftarrow a\left(aZ_3^4\right)$
```

Algorithm 5 EC point doubling

```
Require: $P_1 = (X_1, Y_1, Z_1, aZ_1^4)$
Ensure: $2P_1 = P_3 = (X_3, Y_3, Z_3, aZ_3^4)$
  1.  $T_1 \leftarrow Y_1^2$                  $T_2 \leftarrow 2X_1$
  2.  $T_3 \leftarrow T_1^2$                  $T_2 \leftarrow 2T_2$
  3.  $T_1 \leftarrow T_2 T_1$         $(S)$  $T_3 \leftarrow 2T_3$
  4.  $T_2 \leftarrow X_1^2$                  $T_3 \leftarrow 2T_3$
  5.  $T_4 \leftarrow Y_1 Z_1$                $T_3 \leftarrow 2T_3$   $(U)$
  6.  $T_5 \leftarrow T_3\left(aZ_1^4\right)$          $T_6 \leftarrow 2T_2$
  7.  $T_2 \leftarrow T_6 + T_2$
  8.  $T_2 \leftarrow T_2 + \left(aZ_1^4\right)$ $(M)$
  9.  $T_6 \leftarrow T_2^2$                  $Z_3 \leftarrow 2T_4$
  10. $T_4 \leftarrow 2T_1$
  11. $X_3 \leftarrow T_6 - T_4$
  12. $T_1 \leftarrow T_1 - X_3$
  13. $T_2 \leftarrow T_2 T_1$                $aZ_3^4 \leftarrow 2T_5$
  14. $Y_3 \leftarrow T_2 - T_3$
```

EPDA includes a FSM with 29 states. EPDA waits in first (epda-IDLE) state until the signal (START-PAD) from MC is set. After it is set, if ADD-DOUBLE signal from MC is set, EPDA writes Z coordinate of point Q to the both inputs of MMMC, commands MMMC to start a MMM by setting START-MONT signal and goes to the first (epda-S1) state of point addition. Otherwise it writes Y coordinate of point Q to the both inputs of MMMC, commands MMMC to start a MMM by setting START-MONT signal, writes X coordinate of point Q to the both inputs of MAS circuit, commands MAS circuit to start a MAS by setting START-MOD-AS signal and goes to the first (epda-S15) state of point doubling.

EPDA waits in epda-S1 or epda-S15 state for the signal DONE-MONT to be set. After DONE-MONT signal is set, if EPDA is in epda-S1 state, EPDA writes the output of MMMC to the register for T_1, writes the new values to the inputs of MMMC, commands to MMMC to start a MMM and goes to the second (epda-S2) state of point addition. After DONE-MONT signal is set, if EPDA is in epda-S15 state, EPDA writes the output of MMMC to the register for T_1, writes the output of MAS circuit to the register for T_2, writes the new values to the inputs of MMMC and MAS circuit, commands to MMMC and MAS circuit to start a MMM and a MAS and goes to the second (epda-S16) state of point doubling. This way EPDA executes Algorithm 4 or Algorithm 5 according if ADD-DOUBLE signal is set or not. When DONE-MONT signal is set during the last state, EPDA sets the signal DONE-PAD, which indicates Algorithm 4 or Algorithm 5 finished and goes to epda-IDLE state.

3.8 Modular Multiplicative Inverter

According to Fermat's theorem, $a^{-1} = a^{p-2} \bmod p$, if $gcd(a,p) = 1$ [18]. Because the curves we are interested in are defined over $GF(p)$, p is prime, we can use this theorem to find the multiplicative inverses modulo p. So multiplicative inversion can be done by modular exponentiation of a by $p-2$.

Modular exponentiation can be realized by using the square and multiply algorithm given below [18].

Algorithm 6 Modular exponentiation

```
Require: integers $0 \leq M < N$, $0 < E < N$, $E = (e_{t-1}, e_{t-2}, \cdots, e_0)_2$, $e_{t-1} = 1$ and $N$
Ensure: $M^E \bmod N$
1: $A \rightarrow M$
2: for $i$ from $t-2$ to 0 do
3:     $A \rightarrow AA \bmod N$
4:     if $e_i = 1$ then
5:         $A \rightarrow AM \bmod N$
6:     end if
7: end for
8: Return (A)
```

MMI controls the execution of Algorithm 6. It includes a FSM with 4 states. The circuit stays in first (inv-IDLE) state until the START-INV signal from the PtoA is set. After START-INV signal is set, then MMI goes to the second (inv-S1) state by writing $M-2$ to one of the internal registers ($T1$) and Z coordinate of point Q to ($T2$).

In inv-S1 state MMI compares the value of the counter with N. If they are the same, MMI goes to the inv-IDLE state by setting the DONE-INV signal. If the value of the counter is smaller than N, then MMI commands to the counter to increment by 1, writes $T2$ to the inputs of MMMC, commands to MMMC to start a MMM operation and goes to the third (inv-S2) state.

MMI stays in inv-S2 state waiting for the signal (DONE-MONT) from MMMC to be set. After receiving this indication, MMI shifts $T1$, one bit left, checks if the MSB of $T1$ is 1. If it is 1 then writes the result to one input of MMMC and Z to the other one, commands to MMMC to start a MMM operation and goes to the forth (inv-S3) state.

MMI stays in inv-S3 state waiting for DONE-MONT to be set. After it is set, MMI goes to the inv-S1 state back. So another iteration of Algorithm 6 starts.

3.9 Montgomery Modular Multiplication Circuit

Systolic Array Cells:

The i-th iteration of Step 2 in Algorithm 2 computes the temporary results

$$T_i = 2^{-1}(T_{i-1} + x_i \times Y + m_i \times N)\ i = 0, \cdots, l-1 \tag{4}$$

where $T_{-1} = 0$ [35]. The j-th digit of T_i is obtained using the recurrence relation

$$\begin{aligned} &2^2 \times c1_{i,j} + 2 \times c0_{i,j} + t_{i,j} = \\ &t_{i-1,j+1} + x_i \times y_j + m_i \times n_j + 2 \times c1_{i,j-1} + c0_{i,j-1} \end{aligned} \tag{5}$$

$i = 0, \cdots, l-1$, $j = 0, \cdots, l$, $c1_{i,-1} = 0$ and $c0_{i,-1} = 0$. In Eq. (5), $2 \times c1_{i,j} + c0_{i,j}$, $j = -1, \cdots, l$, denotes the carry chain up the adder.

The regular cell of the systolic array consists of two full-adders (FA), one half-adder (HA) and two AND-gates as shown in Fig. 2.A.

$-N^{-1}$ mod 2 can be written as $(2 - n_0)^{-1}$ mod 2. Because N is odd prime for ECC, $n_0 = 1$. By using this property we can calculate m_i by the following equation:

$$m_i = (t_{i-1,1} + x_i \times y_0) \bmod 2 = t_{i-1,1} \oplus x_i \times y_0 \tag{6}$$

$i = 0, \cdots, l-1$ and $t_{-1,1} = 0$. Here m_i is not an input to the rightmost cell, but obtained in the rightmost cell.

Because there is no carry input to the rightmost cell, the equation for calculating $t_{i,0}$ can be simplified as shown by Eq. (7).

$$2 \times c0_{i,0} + t_{i,0} = t_{i-1,1} + x_i \times y_0 + m_i \tag{7}$$

$i = 0, \cdots, l-1$ and $t_{-1,1} = 0$. The truth table of $c0_{i,0}$ and $t_{i,0}$ is given in Table 1. According to Table 1, $t_{i,0} = 0$ and the equation for $c0_{i,0}$ is as follows:

Table 1. The truth table of $c0_{i,0}$ and $t_{i,0}$

$t_{i-1,1}$	$x_i \times y_0$	m_i	$c0_{i,0}$	$t_{i,0}$
0	0	0	0	0
0	0	1	x	x
0	1	0	x	x
0	1	1	1	0
1	0	0	x	x
1	0	1	1	0
1	1	0	1	0
1	1	1	x	x

x: Do not care conditions

$$c0_{i,0} = t_{i-1,1} + x_i \times y_0 \tag{8}$$

$i = 0, \cdots, l-1$ and $t_{-1,1} = 0$. The rightmost cell of the systolic array consists of one AND, one OR and one XOR gate as shown in the Fig. 2.B.

Because there is only one carry input from rightmost cell, Eq. (5) can be simplified for $t_{i,1}$ as follows, which is obtained by the cell shown in Fig. 2.C. It consists of one FA, two HAs and two AND-gates.

$$2^2 \times c1_{i,1} + 2 \times c0_{i,1} + t_{i,1} = t_{i-1,2} + x_i \times y_1 + m_i \times n_1 + c0_{i,0} \tag{9}$$

$i = 0, \cdots, l-1$ and $t_{-1,2} = 0$.

Because $n_l = 0$, the equation of $t_{i,l}$ can be simplified as follows:

$$2 \times c0_{i,l} + t_{i,l} = t_{i-1,l+1} + x_i \times y_l + 2 \times c1_{i,l-1} + c0_{i,l-1} \tag{10}$$

$i = 0, \cdots, l-1$ and $t_{-1,l+1} = 0$. This equation is implemented by the l-th cell, which is shown in Fig. 2.D. This cell consists of one FA, one AND-gate and one XOR-gate.

The i-th row computes T_i from T_{i-1}. Each cell operates in a single clock cycle. Then the i,j-th cell processes the digits of Eq. (5) at clock cycle time $2i + j$.

Systolic Array

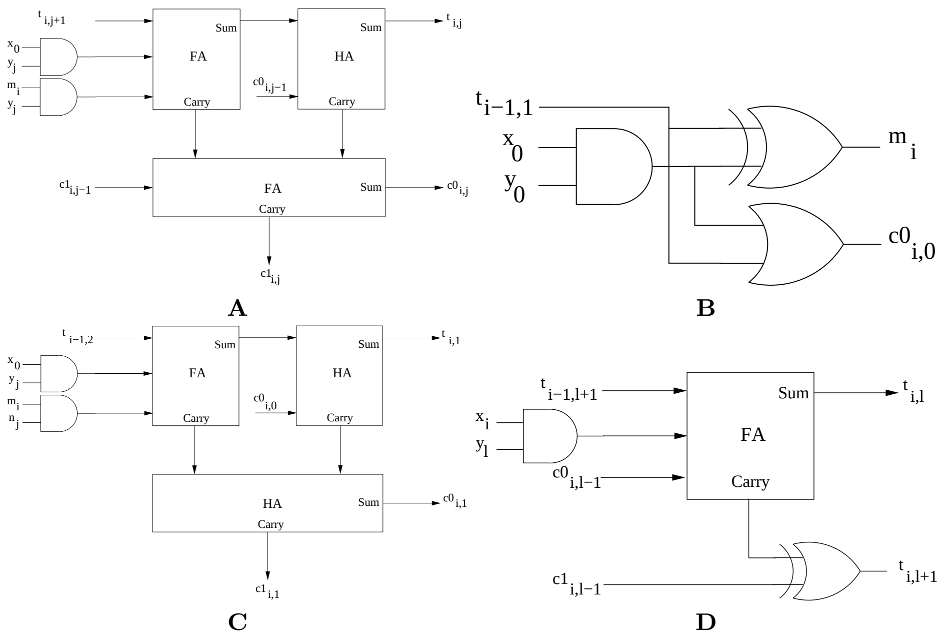
To obtain a linear, pipelined modular multiplier, only one row of cells is taken. The j-th cell behaves like cell (i,j), computing Eq. (5) at time $2i + j$ for $i = 0, \cdots, l+1$.

The schematic view of the systolic array is shown in Fig. 3. $X(0)$ denotes the least significant bit (LSB) of the register in which the input X is stored. T denotes the intermediate value register. The carry chain is stored in the $C0$ and $C1$ registers.

Fig. 3 shows that the T_{j+1} output of $(j+1)$-th cell is used as an input for j-th cell during the next iteration. This way the division by 2 in Step 4 of Algorithm 2 is realized.

Total area of the systolic array is $(5l-3)XOR + (7l-7)AND + (4l-5)OR$ gates and $4l$ flip-flops. The critical path is the same as the critical path of one

Fig. 2. Schematic view of systolic array cells; A) Regular, B) Rightmost, C) 1-st bit, D) Leftmost



regular cell and it is independent of the bit length of the operands. So it is $2T_{FA}(c_{in} \rightarrow c_{out}) + T_{HA}(c_{in} \rightarrow c_{out})$.

Modular Montgomery Multiplication Circuit

The MMMC has three l-bit data inputs X, Y and N, one START instruction input, one DONE output, which indicates that the operation is ended, and an l-bit RESULT output.

The MMMC is designed using the algorithmic state machine (ASM) approach. For detailed information about ASM approach, reader is referred to [16]. The circuit consists of controller and data path as shown in Fig. 4. The controller

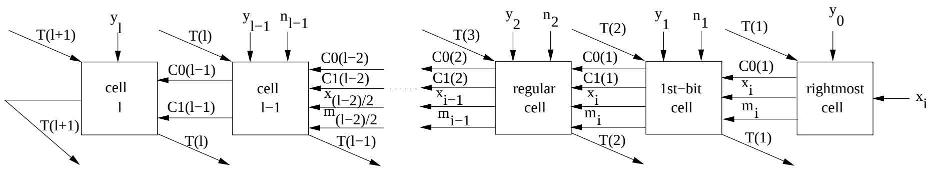


Fig. 3. Schematic view of complete systolic array

has four states, IDLE, MUL1, MUL2 and OUT. The data path consists of a systolic array, four internal registers, a counter and a comparator.

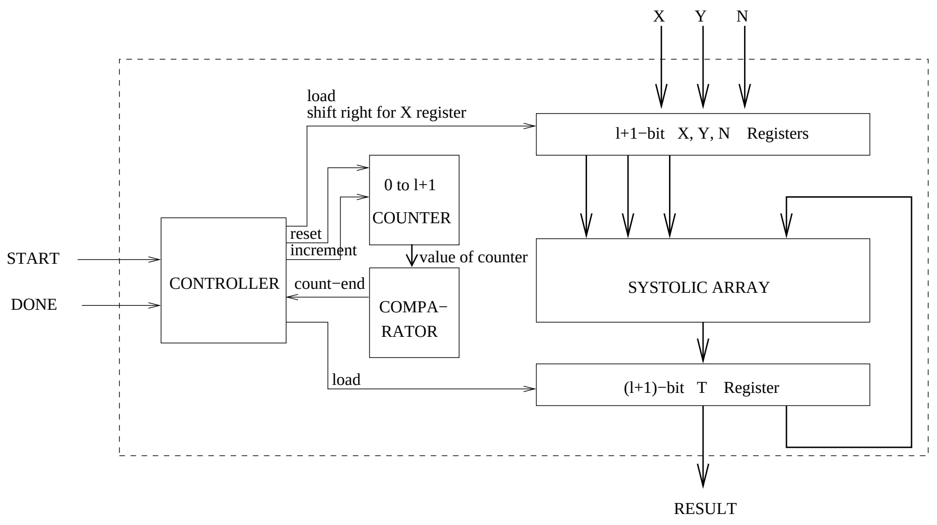


Fig. 4. Architecture of the Montgomery modular multiplier circuit

The controller stays in the IDLE state waiting for the START instruction. When the START input is set, X, Y and N registers are loaded by input values, the T register and the counter are reset.

In MUL1, the outputs of the systolic array cells are written to the T register and controller goes to the MUL2 state.

When the controller is in the MUL2 state, the counter is incremented by 1. When the counter value reaches $2(l+1)$, the comparator sets the "count-end" signal. Then the controller goes to the OUT state in which the value of the T register is written to the RESULT output and the acknowledgement signal DONE is set.

In the MUL2 state, the X register is shifted one bit right and the most significant bit (MSB) of the X register is filled 0. This ensures that, during the last iteration of Step 2 of Algorithm 1, the value of $X(0)$ will be 0.

As mentioned before $t_{i,j}$ is calculated at the $(2i+j)$-th clock cycle, $i = 1, \cdots, l+2$ and $j = 1, \cdots, l$. $t_{l+2,l}$ is calculated at the $(2(l+2)+l)$-th clock cycle. Hence, the total number of clock cycles for completing one modular Montgomery multiplication equals $3l+4$.

3.10 Modular Addition, Substraction

Modular addition and subtraction are executed according to Algorithm 7 and Algorithm 8 [5] given below, respectively.

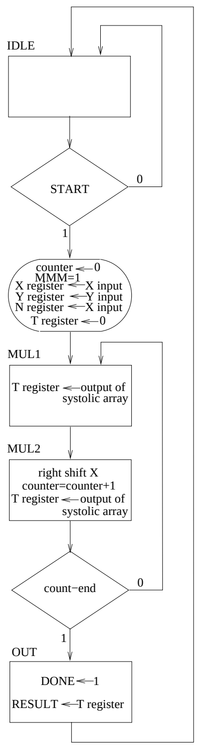


Fig. 5. Algorithmic state machine of Montgomery modular multiplier

MAS includes a FSM with 4 states. The circuit stays in first (mas-IDLE) state until the START-MAS signal from the EPDA is set. After START-MAS signal is set, MAS commands ASC to start an addition operation by setting START-AS signal. If MOD-AS signal from EPDA is "0", which indicates a modular addition operation must be performed, then MAS goes to the second (mas-S1) state. If MOD-AS signal is "1", which indicates a modular subtraction operation must be performed, then MAS goes to the fifth (mas-S4) state.

In mas-S1 or mas-S4 states MAS waits for DONE-AS signal from ASC, which indicates the addition operation is finished. So after receiving START-MAS, according to MOD-AS signal from EPDA, MAS executes Algorithm 7 for modular addition or Algorithm 8 for modular subtraction. When DONE-AS signal is set in the last state of MAS's FSM, it sets DONE-MAS signal, which indicates a modular addition or subtraction operation is finished to EPDA and goes back to mas-IDLE state.

Algorithm 7 Modular addition

```
Require: $M$, $0 \leq A < M$, $0 \leq B < M$
Ensure: $C = A + B \bmod M$
1: $C' = A + B$
2: $C'' = C' - M$
3: if $C'' < 0$ then
4:     $C = C'$
5: else
6:     $C = C''$
7: end if
```

Algorithm 8 Modular subtraction

```
Require: $M$, $0 \leq A < M$, $0 \leq B < M$
Ensure: $C = A - B \bmod M$
1: $C' = A - B$
2: $C'' = C' + M$
3: if $C' < 0$ then
4:     $C = C''$
5: else
6:     $C = C'$
7: end if
```

3.11 Addition, Substraction

The numbers represented in *two's complement* representation [23]. In this representation, addition is done as following, $A, B \in \mathbb{Z}_{2^l}$, $A = (a_{l-1}, \cdots, a_0)_2$, $B = (b_{l-1}, \cdots, b_0)_2$ and $S + Cout = A + B$, $(S = (s_{l-1}, \cdots, s_0)_2$,

$$\begin{aligned} s_i &= a_i \oplus b_i \oplus c_i \\ c_{i+1} &= a_i b_i + a_i c_i + b_i c_i \end{aligned} \tag{11}$$

$i = 0, \cdots, l-1$, $c_0 = 0$ and $Cout = c_l$. If $Cout = 1$, then $S \geq 2^l$.

Subtraction is done as following, $A, B \in \mathbb{Z}_{2^l}$, $A = (a_{l-1}, \cdots, a_0)_2$, $B = (b_{l-1}, \cdots, b_0)_2$ and $S + Cout = A - B = A + \left(2^l - B\right)$, $(S = (s_{l-1}, \cdots, s_0)_2$, $2^l - B = (1 \oplus b_{l-1}, \cdots, 1 \oplus b_0)_2 + 1$. So the same circuit for addition can be used for subtraction as following,

$$\begin{aligned} b' &= AS \oplus b_i \\ s_i &= a_i \oplus b' \oplus c_i \\ c_{i+1} &= a_i b' + a_i c_i + b' c_i \end{aligned} \tag{12}$$

$i = 0, \cdots, l-1$, $c_0 = AS$ and $Cout = c_l$. If $Cout = 1$, then $A \geq B$, otherwise $A < B$.

ASC includes a bit-serial adder with one FA, two shift registers and one flip-flop, a counter and a controller as shown in Figure 6. FSM of ASC has two states. ASC waits in first (as-IDLE) state waiting for START-AS signal from MAS to be set. After START-AS signal is set, ASC commands A, B registers and carry flip-flop to load the inputs, resets the counter and goes to the second (asc-S1) state.

In as-S1 state, ASC checks if the counter value is l. If this is the case, then ASC writes the A to Sum output, sets the DONE-AS signal and goes to as-IDLE state. Otherwise, ASC commands to A and B registers to right shift.

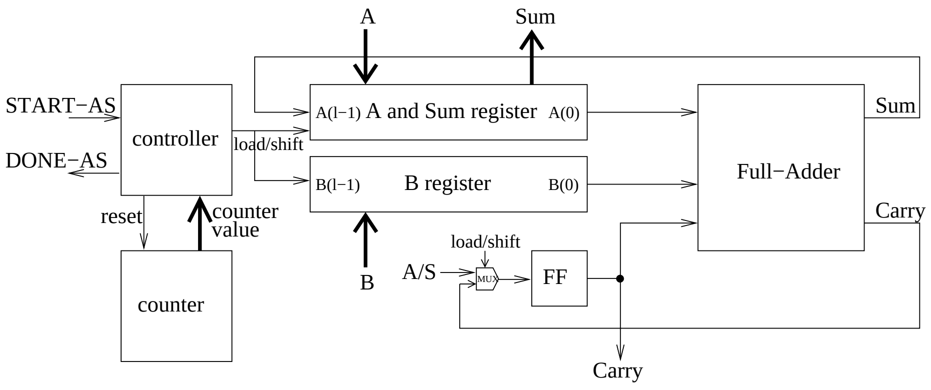


Fig. 6. Architecture of addition, subtraction circuit

3.12 Implementation Results of The Elliptic Curve Processor

The ECP is implemented on Xilinx V1000E-BG-560-8 (Virtex E) FPGA by taking the bit length of EC parameters N and the bit length of k, l as 160. 6055 out of 12288 SLICEs were used for the implementation of ECP. This is equivalent to 115,520 gates. Minimum period of clock is 10.952ns (Maximum frequency: 91.308MHz).

Table 2. Latency of the operations executed in ECP

NtoM	4 MMM	$12N + 16$	0.021
EPM	l EC point double+$l/2$ EC point double	$l(51N + 66)$	14.414
PtoA	MMI+4 MMM	$3N^2 + 16N + 16$	0.397
MtoN	2 MMM	$6N + 8$	0.011
EC point doubling	8 MMM+6 MAS	$30N + 38$	0.070
EC point addition	14 MMM	$42N + 56$	0.074
MMI	$3N/2$ MMM	$9/2N^2 + 6N$	1.272
MMM		$3N + 4$	0.005
MAS		$2N + 1$	0.003

4 Conclusions

We have described an architecture of elliptic curve processor over $GF(p)$. The processor is designed by taking the bit-length of EC parameters as generic. So it is suitable for any bit-length and the clock frequency of the processor does not depend on it. For the most critical, modular multiplication operation, we use the method of Montgomery, which is proven to be very secure in hardware. Namely, the optimal bound is achieved which, with some savings in hardware, omits completely all reduction steps that are presumed to be vulnerable to side-channel attacks.

The EC processor is implemented on Xilinx V1000E-BG-560-8 (Virtex E) FPGA by taking the bit length of EC parameters N and the bit length of k, l as 160. The latency of the operations are given according to results of this implementation.

References

1. P. J. Ashenden. *The Designer's Guide to VHDL.* Morgan Kaufmann Publishers, Inc., San Francisco, California, 1996.
2. L. Batina and G. Muurling. Montgomery in practice: How to do it more efficiently in hardware. In B. Preneel, editor, *Proceedings of RSA 2002 Cryptographers' Track*, number 2271 in Lecture Notes in Computer Science, pages 40–52, San Jose, USA, February 18-22 2002. Springer-Verlag.
3. T. Blum and C. Paar. Montgomery modular exponentiation on reconfigurable hardware. *Proceedings of 14th IEEE Symposium on Computer Arithmetic*, 47(2):70–77, 1999.
4. T. Blum and C. Paar. High-radix Montgomery modular exponentiation on reconfigurable hardware. *IEEE Transactions on Computers*, 50(7):759–764, July 2001.
5. Ç. K. Koç. RSA hardware implementation. Technical report, RSA Laboratories, RSA Data Security, Inc., Redwood City, CA, August 1995.
6. H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation using mixed coordinates. In K. Ohta and D. Pei, editors, *Proceedings of ASIACRYPT 1998*, number 1514 in Lecture Notes in Computer Science, pages 51–65. Springer-Verlag, 1998.
7. S. E. Eldridge and C. D. Walter. Hardware implementation of Montgomery's modular multiplication algorithm. *IEEE Transactions on Computers*, 42:693–699, 93.
8. S. Even. Systolic modular multiplication. In A. J. Menezes and S. A. Vanstone, editors, *Advances in Cryptology: Proceedings of CRYPTO'90*, number 537 in Lecture Notes in Computer Science, pages 619–624. Springer-Verlag, 1990.
9. W. L. Freking and K. K. Parhi. Performance-scalable array architectures for modular multiplication. In *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, pages 149–160. IEEE, 2000.
10. N. Gura, S. C. Shantz, H. Eberle, D. Finchelstein, S. Gupta, V. Gupta, and D. Stebila. An end-to-end systems approach to elliptic curve cryptography. In Burt Kaliski J., Ç. K. Koç, and C. Paar, editors, *Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2002)*, Lecture Notes in Computer Science, 2002.

11. K. Iwamura, T. Matsumoto, and H. Imai. High-speed implementation methods for RSA scheme. In R. A. Rueppel, editor, *Advances in Cryptology: Proceedings of EUROCRYPT'92*, number 658 in Lecture Notes in Computer Science, pages 221–238. Springer-Verlag, 1992.
12. K. Iwamura, T. Matsumoto, and H. Imai. Systolic-arrays for modular exponentiation using Montgomery method. In R. A. Rueppel, editor, *Advances in Cryptology: Proceedings of EUROCRYPT'92*, number 658 in Lecture Notes in Computer Science, pages 477–481. Springer-Verlag, 1992.
13. K. Iwamura, T. Matsumoto, and H. Imai. Montgomery modular multiplication method and systolic arrays suitable for modular exponentiation. *Electronics and Communications in Japan*, 77(3):40–50, 1994.
14. Y. S. Kim, W. S. Kang, and J. R. Choi. Implementation of 1024-bit modular processor for RSA cryptosystem. In *Proceedings of Asia-Pasific Conference on ASIC (AP-ASIC)*, Cheju Island, Korea, August 28-30 2000.
15. P. Kornerup. A systolic, linear-array multiplier for a class of right-shift algorithms. *IEEE Transactions on Computers*, 43(8):892–898, August 1994.
16. M. M. Mano and C. R. Kime. *Logic and Computer Design Fundamentals*. Prentice Hall, Upper Saddle River, New Jersey 07458, second edition, 2001.
17. W. P. Marnane. Optimised bit serial modular multiplier for implementation on field programmable gate arrays. *Electronics Letters*, 34(8):738–739, April 1998.
18. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.
19. A. J. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.
20. P. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, Vol. 44:519–521, 1985.
21. G. Orlando and C. Paar. A scalable GF(p) elliptic curve processor architecture for programmable hardware. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Proceedings of Workshop on Cryptograpic Hardware and Embedded Systems (CHES 2001)*, number 2162 in Lecture Notes in Computer Science, pages 356–371, Paris, France, May 14-16 2001. Springer-Verlag.
22. H. Orup. Simplifying quotient determination in high-radix modular multiplication. In *Proceedings of the 12th Symposium on Computer Arithmetic*, pages 193–199. IEEE, 1995.
23. B. Parhami. *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford University Press, Inc., New York, 2000.
24. J. Poldre, K. Tammemae, and M. Mandre. Modular exponent realization on FPGAs. In *Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, From FPGAs to Computing Paradigm (FPL'98)*, pages 336–347, Tallinn, Estonia, August 31-September 3 1998.
25. E. Savaş, A. F. Tenca, and Ç. K. Koç. A scalable and unified multiplier architecture for finite fields GF(p) and GF(2^m). In C. Paar and Ç. K. Koç, editors, *Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2000)*, number 1965 in Lecture Notes in Computer Science, pages 281–296. Springer-Verlag, 2000.
26. C.-Y. Su, S.-A. Hwang, P.-S. Chen, and C.-W. Wu. An improved Montgomery's algorithm for high-speed RSA public-key cryptosystem. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7(2):280–284, June 1999.
27. A. F. Tenca and Ç. K. Koç. A scalable architecture for Montgomery multiplication. In Ç. K. Koç and C. Paar, editors, *Proceedings of Cryptographic Hardware and Embedded Systems (CHES 1999)*, number 1717 in Lecture Notes in Computer Science, pages 94–108. Springer-Verlag, 1999.

28. A. F. Tenca, Georgi Todorov, and Ç. K. Koç. High-radix design of a scalable modular multiplier. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2001)*, number 2162 in Lecture Notes in Computer Science, pages 189–205. Springer-Verlag, 2001.
29. A. A. Tiountchik. Systolic modular exponentiation via Montgomery algorithm. *Electronics Letters*, 34(9):874–875, April 1998.
30. E. Trichina and A. Tiountchik. Scalable algorithm for Montgomery multiplication and its implementation on the coarse-grain reconfigurable chip. In D. Naccache, editor, *Proceedings of Topics in Cryptology - CT-RSA 2001*, number 2020 in Lecture Notes in Computer Science, pages 235–249. Springer-Verlag, 2001.
31. W.-C. Tsai, C. B. Shung, and S.-J. Wang. Two systolic architectures for modular multiplication. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(1):103–107, February 2000.
32. C. D. Walter. Systolic modular multiplication. *IEEE Transactions on Computers*, 42:376–378, 1993.
33. C. D. Walter. Still faster modular multiplication. *Electronics Letters*, 31(4):263–264, February 1995.
34. C. D. Walter. Techniques for the hardware implementation of modular multiplication. In *Proceedings of 2nd IMACS International Conference on Circuits, Systems & Computers*, volume 2, pages 945–949, Athens, October 1998.
35. C. D. Walter. Montgomery's multiplication technique: How to make it smaller and faster. In Ç. K. Koç and C. Paar, editors, *Proceedings of Cryptographic Hardware and Embedded Systems (CHES 1999)*, number 1717 in Lecture Notes in Computer Science, pages 80–93. Springer-Verlag, 1999.
36. C. D. Walter. An improved linear systolic array for fast modular exponentiation. *IEEE Computers and Digital Techniques*, 147(5):323–328, September 2000.
37. C. D. Walter. Precise bounds for Montgomery modular multiplication and some potentially insecure RSA moduli. In B. Preneel, editor, *Proceedings of Topics in Cryptology- CT-RSA 2002*, number 2271 in Lecture Notes in Computer Science, pages 30–39, 2002.