Computing and Informatics, Vol. 23, 2004, 487–499

AN FPGA IMPLEMENTATION OF A MONTGOMERY MULTIPLIER OVER $GF(2^M)^*$

Nele Mentens, Sıddıka Berna Örs
Bart Preneel, Joos Vandewalle

Katholieke Universiteit Leuven, ESAT/COSIC
Kasteelpark Arenberg 10
B-3001 Leuven-Heverlee, Belgium
e-mail: {Nele.Mentens, Siddika.BernaOrs, Bart.Preneel, Joos.Vandewalle}@esat.kuleuven.ac.be

Manuscript received 21 February 2005

Abstract. This paper describes an efficient FPGA implementation for modular multiplication in the finite field $GF(2^m)$ that is suitable for implementing Elliptic Curve Cryptosystems. We have developed a systolic array implementation of a Montgomery modular multiplication. Our solution is efficient for large finite fields ($m = 160 - 193$), that offer a high security level, and it can be scaled easily to larger values of m. The clock frequency of the implementation is independent of the field size. In contrast to earlier work, the design is not restricted to field representations using irreducible trinomials, all one polynomials or equally spaced polynomials.

Keywords: Elliptic Curve Cryptosystems, FPGA, Montgomery's multiplication method, systolic array

1 INTRODUCTION

In 1976, Diffie and Hellman introduced the idea of public key cryptography [1]. They showed that private communication is possible even when one only has an

* Nele Mentens and Sıddıka Berna Örs are funded by research grants of the Katholieke Universiteit Leuven, Belgium. This work was supported by GOA-Mefisto-2000/06, GOA-Ambiorix-2005/11 and FWO.

authenticated channel. Moreover, they have introduced the concept of a digital signature, which allows to uniquely bind a message to its sender. Since then, numerous public-key cryptosystems (PKCs) have been proposed and all these systems based their security on the difficulty of some mathematical problem. Elliptic Curve Cryptosystems (ECCs), which were proposed by Koblitz [2] and Miller [3], are examples of PKCs. The basic operation in an ECC is a point multiplication which relies on an efficient finite field multiplication. Commonly used finite fields for ECC are $GF(p)$ and $GF(2^m)$. As a consequence, a substantial amount of research is focused on efficient and secure implementations of modular multiplication in hardware.

In 1985 Montgomery has introduced a new method for modular multiplication [4]. The approach of Montgomery avoids the time consuming trial division that is a common bottleneck in other algorithms. His method has been proved to be very efficient and is the basis of many implementations of modular multiplication, both in software and hardware [5, 6, 7, 8, 9, 10]. The implementation discussed in [5] is fully systolic but not suitable for comparison because it is done in the finite field $GF(p)$. In 1998 Koç and Acar introduced a method to use Montgomery multiplication over $GF(2^m)$ [11]. They showed that the multiplication operation in the field $GF(2^m)$ can be implemented significantly faster than the standard multiplication. In this paper we look at an efficient hardware implementation of a Montgomery Modular Multiplication (MMM) over $GF(2^m)$ in a Field Programmable Gate Array (FPGA). An efficient implementation of the MMM over $GF(2^m)$ in hardware was considered by Wu [12]; the proposed parallel architecture is restricted to finite fields that are represented using irreducible trinomials. The major drawback of a parallel implementation is the clock frequency's dependency on the field size m. Our contribution consists of an FPGA implementation of the MMM in a systolic array, which allows pipelining and makes the clock frequency of the design independent of the field size m. This is important e.g. in cryptographic applications where a growing field size is necessary for security reasons. The clock frequency of Wu's design decreases with the increase of m. Unlike Wu's design, our implementation is also suitable for finite fields that cannot be represented using a trinomial. This does not restrict our design to applications using this special kind of irreducible polynomials. There are no practical ASIC or FPGA implementation results on previously designed systolic array architectures for Montgomery multiplication. However, Savas et al. presented some unified architectures in [13]. The disadvantage of these architectures is that they are not fully systolic which makes them less flexible.

2 MATHEMATICAL BACKGROUND

2.1 Polynomial Representation

The algorithm for the MMM used in this paper is defined on the polynomial representation. According to this representation an element a of $GF(2^m)$ is a polynomial with length m, written as

$$a(x) = \sum_{i=0}^{m-1} a_i x^i = a_{m-1}x^{m-1} + \ldots + a_1 x + a_0, \tag{1}$$

where the coefficients $a_i \in GF(2)$. In the word-level description of the algorithms in the following sections these bits are grouped into s words of equal length. Let w be the word length and $m = s \cdot w$. Throughout the remainder of this paper, m, w and s will be used to denote the field size, the word length and the number of words in one $GF(2^m)$ element, respectively. Hence, a can be written as

$$a(x) = \sum_{i=0}^{s-1} A_i(x) x^{iw}, \tag{2}$$

where each polynomial $A_i(x)$ corresponds to a word of length w, or

$$A_i(x) = a_{iw+w-1}x^{w-1} + \ldots + a_{iw+1}x + a_{iw}. \tag{3}$$

The addition of two elements a and b in $GF(2^m)$ is performed by adding the polynomials $a(x)$ and $b(x)$, where the coefficients are added in the field $GF(2)$. This is equivalent to a bit-wise XOR operation on the vectors a and b. In order to multiply two elements a and b in $GF(2^m)$, we need to select an irreducible polynomial of degree m. Note that a different choice of the polynomial leads to a different finite field representation, but all finite fields with the same number of elements are isomorphic. Let $p(x)$ be an irreducible polynomial of degree m over the field $GF(2)$, hence $p_m = 1$ and $p_0 = 1$. The product $c = a \cdot b$ in $GF(2^m)$ is obtained by computing

$$c(x) = a(x)b(x) \bmod p(x), \tag{4}$$

where $c(x)$ is a polynomial of length m, representing the element $c \in GF(2^m)$.

2.2 Montgomery Modular Multiplication Over $GF(2^m)$

The Montgomery multiplication of a and b is defined as follows:

$$c(x) = a(x)b(x)r^{-1}(x) \bmod p(x) \tag{5}$$

where $r(x) = x^m \bmod p(x) = (p_{m-1}p_{m-2}\ldots p_1 p_0)$.

Equation 5 can be evaluated with Algorithm 1 given by Koç and Acar in [11].

Algorithm 1. Word-level Montgomery modular multiplication
Require: $a(x)$, $b(x)$, $p(x)$, $P_0'(x)$
Ensure: $c(x) := a(x)b(x)x^{-m} \bmod p(x)$
1: $c(x) := 0$
2: **for** i from 0 to $(s-1)$ **do**
3: $\quad M(x) := [C_0(x) + A_i(x)B_0(x)]\, P_0'(x) \bmod x^w$

4: $c(x) := x^{-w}\left[c(x) + A_i(x)b(x) + M(x)p(x)\right]$
5: **end for**

Algorithm 1 requires the computation of the polynomial $P_0'(x) = P_0^{-1}(x) \bmod x^w$, where $P_0(x) = \sum_{i=0}^{w-1} p_i x^i$ [11]. $P_0'(x)$ consists of w bits and is computed by using the observation that $P_0(x)$ and its inverse satisfy

$$P_0(x)P_0'(x) = 1 \bmod x^i \tag{6}$$

for $i = 1, 2, \ldots, w$.

The following example helps understand Algorithm 1.

Example 1. Let $a(x) = x^7 + x^4 + x + 1$, $b(x) = x^7 + x^6 + x^4 + x^3 + 1$, $p(x) = x^8 + x^6 + x^5 + x + 1$, $m = 8$ and $w = 4$.

In Algorithm 1 the polynomial $P_0'(x)$ is pre-computed as in Equation 6:

- $i = 1$: $P_0(x)P_0'(x) = 1 \bmod x$
  $\Rightarrow (x+1)(p_3'x^3 + p_2'x^2 + p_1'x + p_0') = 1 \bmod x$
  $\Rightarrow 1 \cdot p_0' = 1$
  $\Rightarrow p_0' = 1$
- $i = 2$: $P_0(x)P_0'(x) = 1 \bmod x^2$
  $\Rightarrow (x+1)(p_3'x^3 + p_2'x^2 + p'1_x + 1) = 1 \bmod x^2$
  $\Rightarrow (p_1' + 1)x + 1 = 1$
  $\Rightarrow p_1' = 1$
- $i = 3$: $P_0(x)P_0'(x) = 1 \bmod x^3$
  $\Rightarrow (x+1)(p_3'x^3 + p_2'x^2 + x + 1) = 1 \bmod x^3$
  $\Rightarrow (p_2' + 1)x^2 + 1 = 1$
  $\Rightarrow p_2' = 1$
- $i = 4$: $P_0(x)P_0'(x) = 1 \bmod x^4$
  $\Rightarrow (x+1)(p_3'x^3 + x^2 + x + 1) = 1 \bmod x^4$
  $\Rightarrow (p_3' + 1)x + 1 = 1$
  $\Rightarrow p_3' = 1$

In short, $P_0'(x) = x^3 + x^2 + x + 1$.

For the computation of $c(x)$ we follow the steps in Algorithm 1:

1: $c(x) := 0$
2: $i = 0$
3: $M(x) := [0 + (x+1)(x^3+1)](x^3 + x^2 + x + 1) \bmod x^4 = x^3 + 1$
4: $c(x) := x^{-4}[0 + (x+1)(x^7 + x^6 + x^4 + x^3 + 1) + (x^3+1)(x^8 + x^6 + x^5 + x + 1)]$
$= x^7 + x^5 + x^4 + 1$

2: $i = 1$
3: $M(x) := [1 + (x^3+1)(x^3+1)](x^3 + x^2 + x + 1) \bmod x^4 = 0$

4: $c(x) := x^{-4}[(x^7 + x^5 + x^4 + 1) + (x^3 + 1)(x^7 + x^6 + x^4 + x^3 + 1) + 0]$
$= x^6 + x^5 + x^3 + x$

The result of the Montgomery multiplication of $a(x)$ and $b(x)$ is $c(x) = x^6 + x^5 + x^3 + x$.

2.3 Montgomery Modular Multiplication Circuit

The architecture of the Montgomery Modular Multiplication Circuit (MMMC) consists of a systolic array, a circuit to compute $P'_0(x)$, a read-in and read-out mechanism and a state machine to control the MMM. The systolic array performs Algorithm 1 and will be discussed in Section 2.3.1. The implementation of the inversion of $P_0(x)$ is explained in Section 2.3.2. Section 2.3.3 describes the read-in and read-out mechanism. The read-in mechanism makes sure the inputs to the systolic array arrive at the correct moment. The read-out mechanism registers the resulting bits of the MMM when they are ready. In Section 2.3.4 the state machine controlling the MMM is discussed. The implementation results are listed in Section 3.

2.3.1 Systolic Array

The i^{th} iteration of Step 4 in Algorithm 1 computes the temporary results

$$c_i(x) = x^{-w} \left(c_{i-1}(x) + A_i(x)b(x) + M_i(x)p(x)\right), \tag{7}$$

where $i = 0, \ldots, s-1$ and $c_{-1}(x) = 0$. A division into s words of length w can be done for $c_i(x)$. In our implementation, the j^{th} word of $c_i(x)$, without the right-shift, is obtained using the recurrence relation

$$C_{i,j}(x) = C_{i-1,j+1}(x) + LAB_{i,j}(x) + LMP_{i,j}(x) + HAB_{i,j-1}(x) + HMP_{i,j-1}(x). \tag{8}$$

To include the right-shift of the previous $c(x)$, we use $C_{i-1,j+1}(x)$ instead of $C_{i-1,j}(x)$. In the following, the rest of the notation used in Equation 8 is explained.

The multiplication of two w-bit words with coefficients in $GF(2)$ results in a $2w-1$-bit word. That is why only the w least significant bits of $A_i(x)B_j(x)$ are used in the calculation of $C_{i,j}(x)$. The most significant part, consisting of $w-1$ bits, is used in $C_{i,j+1}(x)$, so $C_{i,j}(x)$ uses the $w-1$ most significant bits of $A_i(x)B_{j-1}(x)$ as an input. For the division into the most and the least significant parts the following notation is used:

$$A_i(x)B_j(x) = HAB_{i,j}(x)x^w + LAB_{i,j}(x). \tag{9}$$

Likewise, this notation is also used for the multiplication of $M_i(x)$ and $P_j(x)$:

$$M_i(x)P_j(x) = HMP_{i,j}(x)x^w + LMP_{i,j}(x). \tag{10}$$

In Equations 9 and 10, $i = 0, \ldots, s-1$ and $j = 0, \ldots, s-1$. $A_i(x)$, $B_j(x)$, $M_i(x)$

and $P_j(x)$ are defined as in Equation 3:

$$A_i(x)B_j(x) = \tag{11}$$

$$\sum_{k=2w-2}^{w-1} \left(\sum_{l=2w-2}^{k} a_{iw+l-w+1} b_{jw+k-l+w-1} \right) x^k + \tag{12}$$

$$\sum_{k=w-2}^{0} \left(\sum_{l=k}^{0} a_{iw+l} b_{jw+k-l} \right) x^k \tag{13}$$

with $i = 0, \ldots, s-1$ and $j = 0, \ldots, s-1$.

$$HAB_{i,j}(x) = \sum_{k=2w-2}^{w-1} \left(\sum_{l=2w-2}^{k} a_{iw+l-w+1} b_{jw+k-l+w-1} \right) x^k \tag{14}$$

with $i = 0, \ldots, s-1$ and $j = 0, \ldots, s-1$.

$$LAB_{i,j}(x) = \sum_{k=w-2}^{0} \left(\sum_{l=k}^{0} a_{iw+l} b_{jw+k-l} \right) x^k \tag{15}$$

with $i = 0, \ldots, s-1$ and $j = 0, \ldots, s-1$. $HMP_{i,j}(x)$ and $LMP_{i,j}(x)$ are determined in the same way.

In the i^{th} clock cycle the array computes $c_i(x)$ by using $c_{i-1}(x)$, $A_i(x)$, $b(x)$ and $p(x)$. Figure 1 shows a schematic view of the array. The output of the $j+1^{\text{th}}$ cell is used as the input for the j^{th} cell during the next iteration. This way the division by x^w in Step 4 of Algorithm 1 is implemented.

Every word $C_{i,j}(x)$ of $c_i(x)$ is computed in a separate cell. These cells are contained in the systolic array. There are three different kinds of cells. Most of the words are calculated by a regular cell (cell 1, ..., cell $s-1$). Two special cells, the rightmost cell (cell 0) and the leftmost cell (cell s), perform the rest of the calculations. Figure 2 shows the different cells in the systolic array. In iteration i all cells take $A_i(x)$ as an input.

Regular cell. Figure 2 a) shows the regular cell (cell 1, ..., cell $s-1$). It consists of two different operations:

- an addition ($+$): This is implemented as a bit-wise XOR array which consists of w XOR gates.
- a multiplication/addition ($*$): This performs a multiplication and sends the most significant part to the next cell (on the left). The least significant part is added to the most significant part of the calculation result of the previous cell (on the right).

Rightmost cell. Based on Step 3 in Algorithm 1 we can compute $M_i(x)$ as follows:

$$M_i(x) = \left(C_{i-1,1}(x) + LAB_{i,0}(x)\right) P'_0(x) \bmod x^w \tag{16}$$

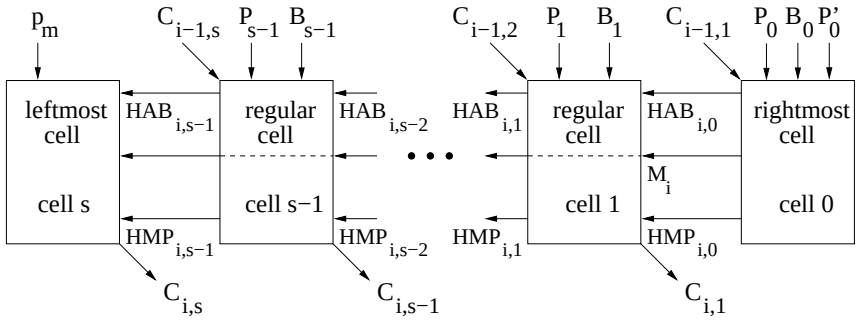


Fig. 1. Schematic view of the complete systolic array

with $i = 0, \ldots, s-1$, $C_{-1,1}(x) = 0$ and $LAB_{i,0}(x) = A_i(x)B_0(x)x^w$. According to Step 4 in Algorithm 1 $M_i(x)$ is not an input to the rightmost cell, but obtained in the rightmost cell. The least significant word (LSW) of $c(x)$ can be obtained by the following equation:

$$C_{i,0}(x) = C_{i-1,1}(x) + LAB_{i,0}(x) + LMP_{i,0}(x) \tag{17}$$

with $i = 0, \ldots, s-1$ and $C_{-1,1}(x) = 0$. It follows immediately from Equations 15 and 14 that $C_{i,0}(x)$ is always equal to 0.

Figure 2(b) shows the rightmost cell (cell 0), which consists of three different operations:

- an addition ($+$): This is the same operation as the addition in the regular cell
- a multiplication ($*$): This is the same as the multiplication/addition in the regular cell, except for the addition that is omitted because there is no previous cell (on the right)
- a modular multiplication ($\circledast$): This is a multiplication modulo 2^w.

Leftmost cell. Because $C_{i,s+1}(x) = 0$, $B_s(x) = 0$ and $P_s(x) = 1$, the equation of $C_{i,s}(x)$ can be simplified as follows:

$$C_{i,s}(x) = M_i(x) + HAB_{i,s-1}(x) + HMP_{i,s-1}(x) \tag{18}$$

with $i = 0, \ldots, s-1$. This equation is implemented by the leftmost cell (cell s), which is shown in Figure 2(c). It consists of:

- an addition ($+$): This is the same operation as the addition in the regular cell.
- a multiplication ($*$): This is the multiplication of a w-bit word with one bit. The result of the multiplication is added to the most significant part of the multiplication/addition result of the previous cell (on the right).

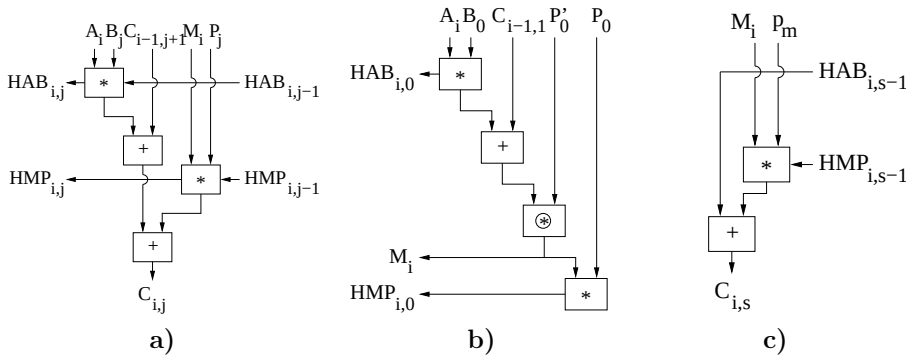


Fig. 2. Schematic view of the array cells ($*$ = multiplication, $+$ = bitwise XOR, $\circledast$ = multiplication modulo 2^w); a) regular, b) rightmost, c) leftmost

2.3.2 Inversion

Equation 6 finds the inversion of the polynomial $P_0(x)$ modulo x^w. When $P_0(x) = p_{w-1}x^{w-1} + \ldots + p_1 x + p_0$ and $P'_0(x) = p'_{w-1}x^{w-1} + \ldots + p'_1 x + p'_0$, the coefficients p'_i, can be found as follows:

$$\begin{aligned}
&p_0 p'_0 = 1 \Rightarrow p'_0 = p_0 = 1 \\
&p_1 p'_0 \oplus p'_1 p_0 = 0 \Rightarrow p'_1 = p_1 \\
&p_2 p'_0 \oplus p_1 p'_1 \oplus p'_2 p_0 = 0 \Rightarrow p'_2 = p_2 \oplus p_1 \\
&p_3 p'_0 \oplus p_2 p'_1 \oplus p_1 p'_2 \oplus p_0 p'_3 = 0 \Rightarrow p'_3 = p_3 \oplus p_1 \\
&p_4 p'_0 \oplus p_3 p'_1 \oplus p_2 p'_2 \oplus p_1 p'_3 \oplus p_0 p'_4 = 0 \Rightarrow p'_4 = p_4 \oplus (p_1 + p_2) \\
&\ldots
\end{aligned} \tag{19}$$

We have designed a combinatorial circuit that calculates the coefficients of the polynomial P'_0 during the first clock cycle and writes the result in a register. Then the value of this register is used as an input for the MMMC.

2.3.3 Read-In and Read-Out Mechanism

If the execution of each iteration of Step 3 and 4 in Algorithm 1 would happen in one clock cycle, M_i would have to be broadcasted through all cells in one clock cycle. This would make the fan-out of the rightmost cell depending on m, which implies that the minimum clock period would increase with m. Because the maximum clock frequency should not become too low and remain independent of the value of m, the design is implemented as a systolic array. Now, the maximum clock frequency only depends on the word length w, because this value determines the number of logic gates in one cell. The critical path T_{crit} of the systolic array is the same as the critical path of one regular cell and is independent of the bit length m of the operands. It is equal to

$$T_{crit} = T_{MULT/ADD} + 2T_{ADD} = T_{2AND} + T_{wXOR} + 2T_{2XOR}, \tag{20}$$

where $T_{MULT/ADD}$, T_{ADD}, T_{2AND}, T_{wXOR} and T_{2XOR} are the latencies of the multiplication/addition, the addition, a 2-input AND-gate, a w-input XOR-gate and a 2-input XOR-gate, respectively. The w-input XOR-gate can be implemented as a balanced tree of 2-input XORs, which makes the critical path proportional to the logarithm of w. $C_{i,j}$ is calculated at the $2(i+1)+(j-1)^{\text{th}}$ clock cycle as the output of the $j+1^{\text{th}}$ cell.

Because of the registers in between, each cell evaluates another word of $a(x)$ in the same clock cycle. A left-shift register, *a_temp*, is used to provide every cell with the correct word of $a(x)$. Figure 3 shows an example for $m = 16$ and $w = 4$. For this example, the starting value, *a_start*, of *a_temp* is 0000 0000 0000 A_0 0000 A_1 0000 A_2 0000 A_3. In the same manner $M_i(x)$ is placed in the least significant word of a left-shift register, *m_temp*, to provide each cell with the correct version of $M_i(x)$.

Because the output words of $c(x)$ are not valid at the same time, a right-shift register, *RSR*, is used to determine when the words are loaded in the output register as shown in Figure 3. The length of *RSR* is always $3s$. For this example, the starting value of *RSR* is 100000000000. Every clock cycle the '1' in this register shifts one place to the right. At the end, this '1' is sent to the enable (E) of the output register.

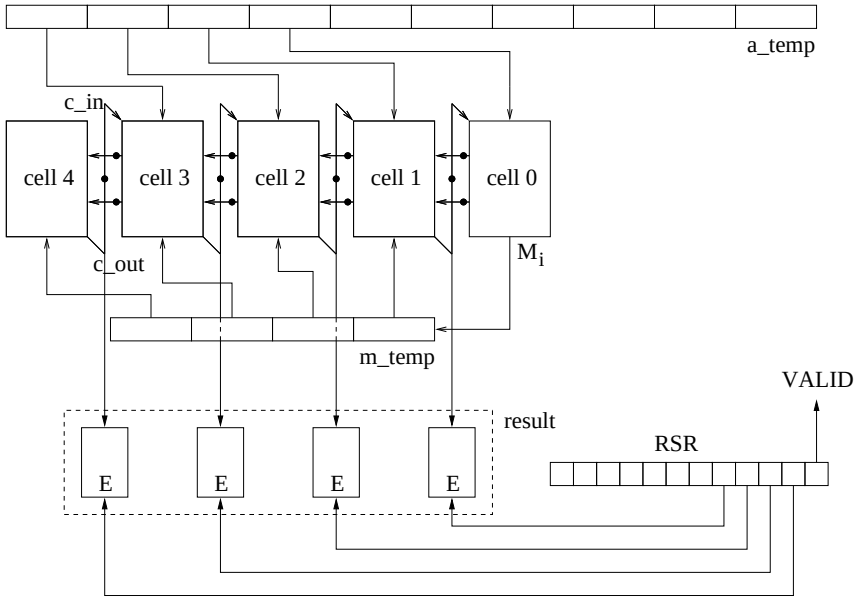


Fig. 3. Architecture of the Montgomery modular multiplier circuit for $m = 16$ and $w = 4$ ($\bullet$ = register (updated every clock cycle), E = enable)

2.3.4 State Machine

Figure 4 shows the Algorithmic State Machine (ASM) chart of the MMMC. When the reset signal (RST) arrives, all the registers are reset. The circuit waits in the IDLE state for the START signal. When the START signal comes *a_start* is loaded into *a_temp*, the most significant bit of *RSR* is set and the circuit goes to state S_1. In every clock cycle *a_temp*, *m_temp* and *RSR* are shifted w, w bits and 1 bit, respectively. Also, the outputs of the systolic array cells are returned to the inputs. When the least significant bit of *RSR* is 1, after $3s$ clock cycles, a VALID signal is produced to indicate that the value of the output register, *result*, is ready.

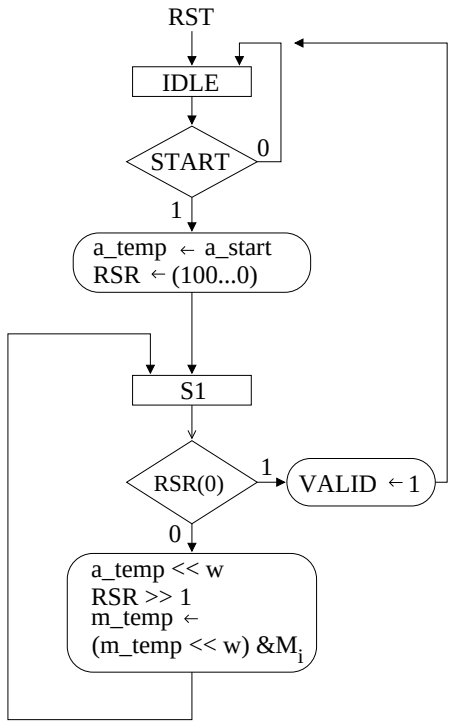


Fig. 4. Algorithmic state machine chart of the Montgomery modular multiplier ($\ll x$ = left shift over x bits, $\gg x$ = right shift over x bits, & = concatenation of two words)

3 IMPLEMENTATION RESULTS

The implementation results of the MMMC on a Xilinx Virtex XCV800-4 FPGA are indicated in Table 1. When the word length w increases, the number of clock cycles needed for completing one MMM decreases, but the minimal clock period increases.

The total MMM latency reaches an optimum for $w = 16$. The circuit becomes larger when w increases.

	$w = 1$	$w = 4$	$w = 8$	$w = 16$	$w = 32$
# of clock cycles	160	120	60	30	15
Minimum clock period (ns)	16.030	17.895	19.798	23.162	62.560
Total MMM latency (μs)	2.564	2.147	1.187	0.694	0.938
# of gates	18 940	32 564	37 017	49 112	72 690

Table 1. FPGA implementation results of the Montgomery modular multiplier over $GF(2^{160})$

4 CONCLUSIONS

In this paper we have presented an efficient hardware implementation of the Montgomery modular multiplication over $GF(2^m)$ in an FPGA. The design has a systolic array architecture to allow pipelining and to make the clock frequency independent of the operand bit length $m = sw$. In this way, the clock frequency does not change when the bit length is enlarged for security reasons. The clock frequency only depends on the word length w, which determines the amount of logic in one cell of the systolic array. The word length is an input parameter for the implementation of the circuit. The design is not restricted to field representations using irreducible trinomials, all one polynomials or equally spaced polynomials. Every irreducible polynomial of degree m can be used. When the word length is chosen to be 1, the maximum frequency is 62.38 MHz. The minimum delay for the execution of one Montgomery multiplication is 0.694 μs with a word length of 16.

REFERENCES

[1] Diffie, W.—Hellman, M. E.: New Directions in Cryptography. IEEE Transactions on Information Theory. Vol. 22, 1976, pp. 644–654.

[2] Koblitz, N.: Elliptic Curve Cryptosystem. Math. Comp., Vol. 48, pp. 203–209, 1987.

[3] Miller, V.: Uses of Elliptic Curves in Cryptography. Advances in Cryptology: Proceedings of CRYPTO '85, H. C. Williams, Ed. 1985, number 218 in Lecture Notes in Computer Science, pp. 417–426, Springer-Verlag.

[4] Montgomery, P.: Modular Multiplication without Trial Division. Mathematics of Computation, Vol. 44, 1985, pp. 519–521.

[5] Örs, S. B.—Batina, L.—Preneel, B.—Vandewalle, J.: Hardware Implementation of a Montgomery Modular Multiplier in a Systolic Array. The 10th Reconfigurable Architectures Workshop (RAW), Nice, France, April 22, 2003.

[6] Batina, L.—Muurling, G.: Montgomery in Practice: How to Do It More Efficiently in Hardware. Proceedings of RSA 2002 Cryptographers' Track, B. Preneel,

Ed., San Jose, USA, February 18–22, 2002, Number 2271 in Lecture Notes in Computer Science, pp. 40–52, Springer-Verlag.

[7] Trichina, E.—Tiountchik, A.: Scalable Algorithm for Montgomery Multiplication and Its Implementation on the Coarse-Grain Reconfigurable Chip. Proceedings of Topics in Cryptology – CT-RSA 2001, D. Naccache, Ed. 2001, Number 2020 in Lecture Notes in Computer Science, pp. 235–249, Springer-Verlag.

[8] Freking, W. L.—Parhi, K. K.: Performance-Scalable Array Architectures for Modular Multiplication. Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors. 2000, pp. 149–160, IEEE.

[9] Tsai, W.-C.—Shung, C. B.—Wang, S.-J.: Two Systolic Architectures for Modular Multiplication. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 8, 2000, No. 1, pp. 103–107.

[10] Eldridge, S. E.—Walter, C. D.: Hardware Implementation of Montgomery's Modular Multiplication Algorithm. IEEE Transactions on Computers, Vol. 42, 1993, pp. 693–699.

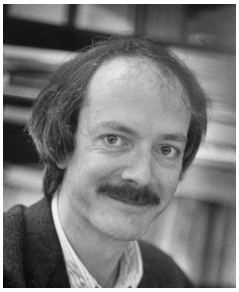
[11] K. Koç, Ç.—Acar, T.: Montgomery Multiplication in GF(2^k). Designs, Codes and Cryptography, Vol. 14, pp. 57–69, 1998.

[12] Wu, H.: Montgomery Multiplier and Squarer in GF(2^m). Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2000), Ç. Koç and C. Paar, Eds., Worcester, MA, USA, August 2000, Number 1965 in Lecture Notes in Computer Science, pp. 264–276, Springer-Verlag.

[13] Savaş, E.—Tenca, A. F.—K. Koç, Ç.: A Scalable and Unified Multiplier Architecture for Finite Fields GF(p) and GF(2^m). Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2000), C. Paar and Ç. K. Koç, Eds. 2000, Number '1965 in Lecture Notes in Computer Science, pp. 281–296, Springer-Verlag.

Nele Mentens obtained the degree of Master of Industrial Science in electronics, chip design, in 2000 at the Katholieke Hogeschool Limburg (KHLim), Diepenbeek. In 2000–2001 she was a research assistant at the KHLim. In 2003, she received the Master degree in applied electrotechnical science, microelectronics. Since 2000, she is a part-time lecturer at the KHLim. In 2003, she started a Ph.D. program at the Katholieke Universiteit Leuven, Belgium. She is working on efficient hardware implementations of cryptographic algorithms and side-channel attacks.

Siddika Berna Örs obtained the degree in electronics and telecommunications engineering from Istanbul Technical University (I. T. U.), Electrical and Electronic Engineering Faculty, Turkey. In 1998, she received the Masters degree in electronics and communication engineering from I. T. U., Institute of Science and Technology. She obtained a doctoral degree on the subject "Hardware Design of Elliptic Curve Cryptosystems and Side-Channel Attacks" in February 2005 at the Katholieke Universiteit Leuven, Belgium. At the moment she is an associate professor at I. T. U.



Bart Preneel received the Doctorate in Applied Sciences from the Katholieke Universiteit Leuven (Belgium). He is currently professor at the K. U. Leuven and visiting professor at the T. U. Graz (Austria). He was visiting professor at several universities in Europe. His main research interests are cryptography and network security. He has authored more than 180 scientific publications. He is vice-president of the IACR (International Association for Cryptologic Research) and a member of the Editorial Board of the Journal of Cryptology and of the ACM Transactions on Information Security. In 2003, he has received the European Information Security Award in the area of academic research.

Joos Vandewalle obtained the degree of M.Sc. in electrical engineering, a doctoral degree and a special doctoral degree from K. U. Leuven in 1971, 1976 and 1984, respectively. He was a postdoctoral researcher in 1976–78 and visiting assistant professor in 1978–79 at the University of California, Berkeley. Since 1979 he is appointed at the K. U. Leuven, where he is full professor since 1986. He is a fellow of IEEE and is currently vice-president for Region 8 of the IEEE Society on Circuits and Systems. He is currently the head of the Department of Electrical Engineering ESAT, and of the research group ESAT/SISTA-COSIC. He is one of the 3 coordinators of the Interdisciplinary Center for Neural Networks (ICNN).