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

Nele Mentens, Sıddıka Berna Örs, Bart Preneel, Joos Vandewalle
Katholieke Universiteit Leuven
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
{Siddika.BernaOrs, Nele.Mentens, Bart.Preneel,
Joos.Vandewalle}@esat.kuleuven.ac.be

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.*
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 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 (ECC), which were proposed by Koblitz [2] and Miller [3], are examples of PKCs. The basic operation for 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 implementation 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 of 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].
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. 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. There are no practical ASIC or FPGA implementation results on previously designed systolic array architectures for Montgomery multiplication.

2 Mathematical background

2.1 Polynomial representation

The algorithm for the Montgomery modular multiplication 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

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

where the coefficients $a_i \in GF(2)$. In the word-level description of the algorithms, these bits are grouped into s words of equal length. Let w be the word length and m=s·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 $a(x) = \sum_{i=0}^{s-1} A_i(x)x^{iw}$, where each polynomial $A_i(x)$ corresponds to a word of length w, or

$$A_i(x) = a_{iw+w-1}x^{w-1} + \ldots + a_{iw+1}x + a_{iw}. \qquad (2)$$

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·b in GF(2^m) is obtained by computing $c(x) = a(x)b(x) \bmod p(x)$, 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:

$$c(x) = a(x)b(x)r^{-1}(x) \bmod p(x) \qquad (3)$$

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

Equation (3) 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

3: $M(x) = (C_0(x) + A_i(x)B_0(x))P'_0(x) \bmod x^w$

4: $c(x) = x^{-w}(c(x) + A_i(x)b(x) + M(x)p(x))$

5: end for

Algorithm 1 requires the computation of the polynomial $P'_0(x) = P_0^{-1}(x) \bmod x^w$, where $P_0(x) = \sum_{i=0}^{w-1} p_i$ [11]. $P'_0(x)$ consists of w bits. The inverse of the polynomial $P_0(x)$ is computed by using the observation that $P_0(x)$ and its inverse satisfy

$$P_0(x)P_0^{-1}(x) = 1 \bmod x^i \tag{4}$$

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

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 3.1. The implementation of the inversion of $P_0(x)$ is explained in Section 3.2. Section 3.3 describes the read-in and the 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 3.4 the state machine controlling the MMM is discussed. The implementation results are listed in Section 3.5.

3.1 Systolic array

The i-th iteration of Step 4 in Algorithm 1 computes the temporary results

$$c_i(x) = x^w(c_{i-1}(x)A_i(x)b(x) + M_i(x)p(x)) \tag{5}$$

where i=0,…,m-1 and $c^{-1}(x)=0$. c_i(x) can be divided into s words of length w. In (5) the j-th word of $c_i(x)$ is obtained using the recurrence relation

$$C_{i,j}(x) = C_{i-1,j+1}(x) + LAB_{i,j}(x) + LMP_{i,j}(x) + HAB_{i,j-1}(x) + HMP_{i,j-1}(x) \tag{6}$$

where $A_i(x)B_j(x) = HAB_{i,j}(x)x^w + LAB_{i,j}(x)$, $M_i(x)P_j(x) = HMP_{i,j}(x)x^w + LMP_{i,j}(x)$, $i = 0,...,s-1$ and $j = 0,...,s$. $A_i(x)$, $B_j(x)$, $M_i(x)$ and $P_j(x)$ are defined as in (2).

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). Fig. 1 shows a schematic view of the array. The output of the j+1-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.

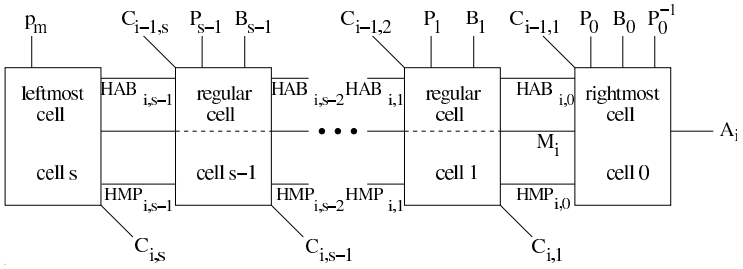
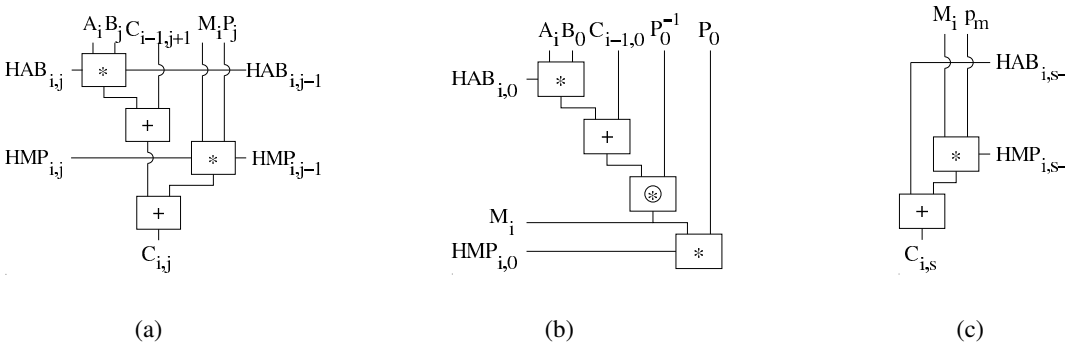


Figure 1: Schematic view of the complete systolic array.

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. Fig. 2 shows the different cells in the systolic array.



(a) (b) (c)

Figure 2: Schematic view of the array cells (∗ = multiplication, + = bitwise XOR, ⊛= multiplication modulo 2^w); (a) Regular, (b) Rightmost, (c) Leftmost.

The regular cell (cell 1,…,cell s-1) in Fig. 1(a) consists of two different operations:

- an addition (+): a bit-wise XOR array which consists of w XOR gates.
- a multiplication/addition (∗): a multiplication which 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).

To understand the logic in the rightmost cell, we look at the computation of $M_i(x)$ in Step 3 of Algorithm 1:

$$M_i(x) = (C_{i-1,1}(x) + LAB_{i,0}(x))P'_0(x) \bmod x^w \tag{7}$$

with i=0,…,s-1, $C_{-1,1}(x)=0$ and $LAB_{i,0}(x) = A_i(x)B_0(x) \bmod x^w$. 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:

$$C_{i,0}(x) = C_{i-1,1}(x) + LAB_{i,0}(x) + LMP_{i,0}(x) \tag{8}$$

with $i = 0,...,s-1$ and $C_{-1,1}(x) = 0$.

It follows immediately from (7) and (8) that $C_{i,0}(x)$ is always equal to 0.

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

- an addition (+): the same operation as the addition in the regular cell.
- a multiplication ($*$): 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 ($\circledast$): a multiplication modulo 2^w.

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:

$$C_{i,s}(x) = M_i(x) + HAB_{i,s-1}(x) + HMP_{i,s-1}(x) \quad (9)$$

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

This equation is implemented by the leftmost cell (cell s), which is shown in Fig. 2(c). It consists of:

- an addition: the same operation as in the regular cell.
- a multiplication/addition ($*$): 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).

3.2 Inversion

Equation (4) finds the inversion of the polynomial $P_0(x)$ modulo x^w. We have designed a circuit that calculates the coefficients of the polynomial P_0^{-1} 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.

3.3 Read-in and read-out mechanism

If the execution of every iteration of Step 3 and 4 in Algorithm 1 would happen in one clock cycle, the delay from $C_{i-1,1}$ to $C_{i,s}$ would be too large. The leftmost cell would have to wait for $HAB_{i,s-1}$ and $HMP_{i,s-1}$ coming from cell s-1, while this cell has to receive $HAB_{i,s-1}$ and $HMP_{i,s-1}$ first, etc. This would imply 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 of the systolic array is the same as the critical path of one regular cell and it is independent of the bit length m of the operands. It is equal to $T_{MULT/ADD}+2T_{ADD}=T_{AND}+(w+3)T_{XOR}$, where T_{MULT} and T_{ADD} are the latencies of the multiplication/addition and the addition respectively. $C_{i,j}$ is calculated at the 2(i+1)+(j-1)-th clock cycle as the output of the j+1-th cell.

Because of the registers in between, every cell evaluates another word of a(x) in the same clock cycle. A left-shift register (LSR), a_temp, is used to provide every cell with the correct word of a(x). Fig. 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 LSR, m_temp, to provide every 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), counter, is used to determine when the words are loaded in the output register as shown in

Fig. 3. The length of counter is always 3s. For this example, the starting value of counter 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.

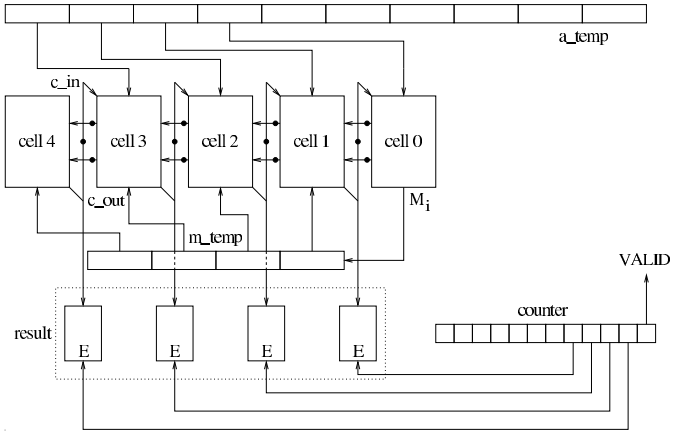


Figure 3: Architecture of the Montgomery modular multiplier circuit for m=16 and w=4
(• = register (updated every clock cycle), E = enable).

3.4 State machine

Fig. 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 counter is set and the circuit goes to state S1. In every clock cycle a_temp, m_temp and counter 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 counter is 1, after 3s clock cycles, a VALID signal is produced to indicate that the value of the output register, result, is ready.

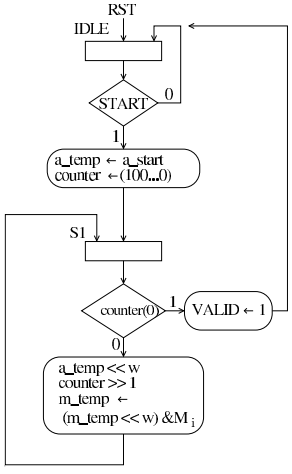


Figure 4: Algorithmic state machine chart of the Montgomery modular multiplier
(<<x = left shift over x bits, >>x = right shift over x bits, & = concatenation of two words).

3.5 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.

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

	w=1	w=4	w=8	w=16	w=32
# of clock cycles	160	120	60	30	15
Minimum clock period	16.030ns	17.895ns	19.798ns	23.162ns	62.560ns
Total MMM latency	2.564µs	2.147µs	1.187µs	0.694µs	0.938µs
# of gates	18 940	32 564	37 017	49 112	72690

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 62,38MHz. The minimum delay for the execution of one Montgomery multiplication is 0,694µs 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, 1987, pp. 203-209.

[3] Miller, V.: Uses of elliptic curves in cryptography. In: 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. In: 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. In: 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. In: 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. In: Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors, 2002, 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, number 1, pp. 103-107, February 2000.

[10]Eldridge, S., E., Walter, C., D.: Hardware Implementation of Montgomery's Modular Multiplication Algorithm. IEEE Transactions on Computers, Vol. 24, number 6, pp. 693-699, June 1993.

[11] Koç, C., K., 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). In: Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2000), C. Koç and C. Paar, Eds., Worcester, MA, USA, August 2000, number 1965 in Lecture Notes in Computer Science, pp. 264-276, Springer-Verlag.