LOW AREA SCALABLE MONTGOMERY INVERSION OVER GF(2^m)

Mohamed N. Hassan and Mohammed Benaissa
Electronic and Electrical Department, University of Sheffield, Mappin Street, Sheffield, U.K.

Keywords: Cryptography, Finite Fields, ECC over GF(2^m), Montgomery Inversion, FPGA.

Abstract: In this work, an improved algorithm for Montgomery modular inversion over GF(2^m) is proposed. Moreover, A novel scalable hardware architecture for the proposed algorithm is presented which is parameterizable and amenable to interfacing to special purpose processors such as microcontrollers. The architecture supports operations over finite fields GF(2^m) up to $m \le 571$ without the need to reconfigure the hardware. The results show that, this work can be exploited to construct low resource elliptic curve cryptosystems (ECC).

1 INTRODUCTION

Since their introduction by Miller and independently Koblitz in 1985 (D. Hankerson, 2004), Elliptic curve cryptosystems are considered the best compromise between the required security and the attainable performance for many low resource constrained security systems. Scalability versus performance, in particular for low resources applications, has always been a challenging trade-off in ECC hardware implementations. The efficiency of of this trade-off depends significantly on the efficient implementation and scalability of the modular arithmetic of the underlying field. The computation of the modular inversion is the most challenging from this perspective. Hence, the contribution of the work presented in this paper.

In the literature, several algorithms for computing the multiplicative inverse in GF(2^m) have been proposed (D. Hankerson, 2004). Some of them are based on performing modular multiplication like Fermat's little theorem. In contrast, others apply the greatest common divisor algorithm GCD which has many variants. All these variants can compute the modular inverse in about 2m iterations. However, the Montgomery inversion algorithm (B. Kaliski, 1995) offers better performance and can perform the inversion in less than 2m iterations. Consequently, this work investigates Montgomery modular inversion and develops algorithmic modifications that reduce the hardware complexity whilst offering scalable and parameterized inversion with low area architecture over FPGAs.

A modified algorithm for Montgomery is therefore proposed and implemented on the smallest and lowest cost Xilinx FPGA. The architecture is parameterized to support variable word lengths and has been implemented with 8, 16, 32 and 64 bit word lengths for finite field lengths m=163 and m=571. The results obtained show that the 32-bit data path designs are the best compromise between the low area requirements and the practical performance in terms of throughput (4.63 Mbps for m = 163).

This paper is organized as follows: section 2 presents a theoretical background about ECC over GF(2^m). Section 3 gives an overview about the Montgomery modular inversion. The proposed improved algorithm is presented in section 4. The description of the circuit operation and the FPGA implementation are in Section 5. Finally, Section 6 shows the performance and results of the implementation on a state of the art FPGA.

2 ELLIPTIC CURVE ITHMETIC OVER GF(2^m)

Briefly, a cryptosystem based on an elliptic curve E over finite fields GF(2^m) is mainly used for encipherment of point P by key K such that, Q=K.P.

$$Y^2 + XY = X^3 + aX + b \quad (1)$$

This operation is called scalar multiplication (N.Koblitz, 1984). Practically, P is a point lies on the curve E or equally the data to be encrypted. Multiplying K by P can be achieved by many methods e.g. double and add or shift and add, etc. Actually, this operation dominates the execution time. Both Q and P must satisfy the equation that represents the elliptic curve E over GF(2^m), namely,

Where $a, b \in GF(2^m)$. Equation (1) is called the simplified Weierstarss equation over the finite field GF(2^m) with characteristic $= 2$ in the affine (Euclidean) coordinates .

For two points P_1 and $P_2 \in GF(2^m)$.

1. Point addition $(P_1 + P_2) = (x_3, y_3)$:

Let $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$ and $(P_1 + P_2) = (x_3, y_3)$ Where $P_1 \neq \pm P_2$ and $P_1, P_2, (P_1 + P_2) \in GF(2^m)$. Then,

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \ \& \ y_3 = \lambda(x_1 + x_3) + x_3 + y_1 \qquad (2)$$

2. Point doubling 2P :

Let $P_1 = (x_1, y_1) \in GF(2^m)$, $P_1 \neq -P_1$ and $2P_1 = (x_3, y_3)$.

$$x_3 = {x_1}^2 + {}^{b}\!/_{{x_1}^2} \ \& \ y_3 = {x_1}^2 + \lambda x_3 + x_3 \qquad (3)$$

Thus, we can observe from equations (2,3) that one inversion is involved in both point addition and point doubling over the elliptic curve E.

Where $$\lambda = \begin{cases} \left(\dfrac{y_2 + y_1}{x_2 + x_1}\right) \ldots P_1 \neq P_2 \\ x_1 + \dfrac{y_1}{x_1} \cdots\cdots p_1 = p_2 \end{cases}$$

3 MONTGOMERY INVERSION AND ITS VARIANT

Based on the extended binary algorithm and Montgomery trick for computing the modular multiplication (L. Montgomery, 1985). B.Kaliski was the first to propose the Montgomery inverse algorithm for a given irreducible polynomial p(x) and for any element $a(x) \in GF(p)$ or GF(2^m). Montgomery inversion for element a(x) is defined as, $\mathrm{MonInv}(a(x)) = a^{-1}.2^m \bmod p(x)$

B.Kaliski proposed two phases to compute the inverse of a(x). The first phase is dedicated to compute the GCD of both a(x) and p(x) and concurrently, calculates the number of halvings L. This phase produces the partial Montgomery inverse. $\mathrm{Par.MonInv}(a(x)) = a^{-1}.2^L \bmod p(x)$.

The number L is in the range $m \leq L \leq 2m$. Then, the second phase performs L-m right shifting on the partial Montgomery inverse to produce the final inverse in the Montgomery domain. The number of iterations can be adjusted to right shifts or left shifts of the output from the first phase to get the inverse in the required domain (Montgomery or residue). The Montgomery inversion Algorithm uses 4 vectors to hold the intermediate calculations between the successive iterations. Although the B.Kaliski Algorithm is simple, it has no fixed number of iterations which makes it difficult to be mapped into hardware efficiently. M.Shieh, J.Chen, and C.Ming (M.Shieh, 2006) developed a new modification to the Kaliski's algorithm, as shown in algorithm 1, in which only one phase is required. Consequently, by this improvement the data dependency between the first phase and the second phase has been eliminated. Moreover, this also avoided the zero comparison operation required by the original algorithm.

Algorithm 1.

Input : $a(x).x^m, p(x)$
Output: $a^{-1}(x).2^m.\mathrm{mod}\, p(x)$
Initialize : $u \leftarrow p, v \leftarrow a, r \leftarrow 0, s \leftarrow 1, L \leftarrow 0$

```
For $L \to 0 : 2m$
{If  (u is even)  then
        $u \leftarrow u/2,\ s \leftarrow 2s.\mathrm{mod}\ p(x)$,
else if (v is even) then
      $v \leftarrow v/2,\ r \leftarrow 2r \bmod p(x)$,
else if (u > v) then
   $u \leftarrow (u \oplus v)/2,\ r \leftarrow r+s,\ s \leftarrow 2s \bmod p(x)$
else    $u \leftarrow (u \oplus v)/2,\ s \leftarrow r + s,\ r \leftarrow 2r \bmod p(x)$
 $L \leftarrow L + 1$ }
return  r;
```

4 MODIFIED MONTGOMERY MODULAR INVERSE ALGORITHM

In this section, based on algorithm 1, an improvement for the Montgomery inversion algorithm over GF(2^m) is represented. Kim and Hong (C.H.Kim, 2003) introduced a development based on a modified version of the binary extended great common divisor algorithm BGCD. Their algorithm is suitable for realizing a compact and fast inverters over GF(2^m). Simply, they replaced the degree comparison employed by the BGCD with a counter and state indicator bit. By applying the same idea to Algorithm 1 we can define a new modified version of the Montgomery inversion algorithm,

Algorithm 2 below, which will save m XORs used to perform the degree comparison. Besides, these m XORs lie on the critical path of the data path. Hence, we can have great savings not only in terms of area but also in terms of reducing the delays caused by degree comparison in algorithm 1.
The algorithm 2 proceeds as follows:

At the beginning, the counter, the state bit, the vectors u, v, s and r are initialized. Thus, we have u > v at the beginning. This means that degree of u has to be decremented according to the BGCD algorithm. Further, at the start of the algorithm the value of u_0 always equals to 1.

Algorithm 2.

Input : $a(x).x^m, p(x)$
Output : $r = a^{-1}(x).2^m.\text{mod}\, p(x)$,
Initialize : $u \leftarrow p, v \leftarrow a, r \leftarrow 0, s \leftarrow 1, L \leftarrow 0$
$\text{State} \leftarrow 0, \text{Count} \leftarrow 0$

```
For L ← 1 : 2m
{ If (state=0) then
   if (u is even) then
      { u ← u/2, s ← 2s.mod p(x)
           if (count = 0) then
              count=count +1 ; state = 1
           end if }
   else if (v is even) then
    { v ← v/2, r ← 2r.mod p(x) , count=count +1}
    else
{ u ← (u ⊕ v)/2, r ← r+s, s ← 2s mod p(x),
     count=count -1
        if (count = 0 )then
         count=count +1 ; state = 1
         end if } }
 else if (state= 1) then
   if (u is even) then
     { u ← u/2, s ← 2s.mod p(x),count=count +1}
   else if (v is even) then
     { v ← v/2, r ← 2r.mod p(x),count=count - 1
        if (count = 0) then
         count=count +1 ; state = 0
        end if }
      else
{ v ← (u ⊕ v)/2, s ← r+s, r ← 2r mod p(x),
      count=count -1
       if (count = 0 )then
         count=count +1 ; state = 0
        end if } }
}
return r ;
```

We have two possible conditions for the vector v. If v_0=1, hence, in the second iteration the counter will be incremented by one and the state bit will equal one. The procedure for decreasing the degree of u is performed by XORring u and v, dividing the value by 2 and saving the result in u. In parallel, vector s is XORed with r and vector s is doubled. The results of the two operations will be stored in r and s respectively. The other possible condition is v_0=0. Thus, the vector v is even. Hence, the counter will increment by one but the state bit will remain zero. Next, the vector v will be divided by two and the vector r will be doubled. Accordingly, the value of the state bit =0 and the counter >0. For the state bit = 1, If u_0 = 1 and v_0=1. This means that the degree of v>u. Hence, the degree of v has to be reduced. Thus, the vector v is XORed with u and the result will be stored back in v. In parallel, vector r is XORed with s and vector r is doubled; the results of the two operations will be stored in s and r respectively and the counter value will be decremented by one. If the value of the counter becomes zero the state bit will be equal to zero otherwise will remain one. The algorithm keeps track as the procedures in algorithm 2 until 2m iterations. After 2m iterations, the value of the vectors u converges to one. Meanwhile, the values of the vectors v and s converge to zero. Finally, the inverse of the vector a(x) represented in the Montgomery domain will be the value in the vector r

5 CIRCUIT DESIGN

Figure 1 depicts the new architecture for the Montgomery inverter. The data path consists of two blocks, namely, u-v block and s-r block. The first is to compute the intermediate values for vectors u and v and the second to compute in parallel the intermediate values for vectors s and r. A control block is designed for, interfacing the dual block RAMs (DBRAM), decisions required by the algorithm and the operations necessary for computing the inverse (shifting operation, reduction, checking the even-non even condition. etc).

As shown in figure 2 and figure 3, both u-v and s-r blocks have a (DBRAM) that acts to hold the vectors u, v, s, and r. The (DBRAM) in each block is addressed by a counter controlled by the control block. Counters are scalable and they accommodate addressing the (DBRAM) up to 2*((m-m.modWord-Length)/Word- Length+1) memory depth, where m is the length of the vector a(x). Both u-v and s-r blocks have two shifting units. In the u-v block, the shifting unit is right shifting. Meanwhile, in the s-r block, the unit is left shifting. Both units load the word to be shifted, storing the most significant digit MSD for the left shift unit or the least significant digit LSD for the right shift unit to be added to the next word, shif left or right by the corresponding number of shift counts, and then write the shifted word to the (DBRAM) port. The Reduction unit is designed to be parameterized and scalable to accommodate finite fields up to $m \le 571$ in addition to different data path widths. NIST recommended reduction polynomials (NIST, 2000) are used to implement the reduction unit as they are designed to provide both security and high performance.

6 CONCLUSIONS AND RESULTS

The proposed modified algorithm for Montgomery inversion has been fully modelled in VHDL and implemented on the smallest and lowest cost chip available from Xilinx Spartan III family (XC3S50). The proposed architecture is parameterized in order to support variable word lengths. A scalable architecture has been implemented with 8, 16, 32 and 64 bit word lengths. Table 1-2 shows the implementation results for the different widths after place and rout for finite field lengths m=163 and m=571. As expected, the control block and counters dominate the critical path of the design. Thus, the increment of the operand size has a lesser effect on the working frequency. The results show that the 32-bit data path designs are the best compromise between the low area requirements and the practical performance in terms of throughput (4.63 Mbps for m = 163). Further, the proposed architecture with low hardware resources is expected to yield correspondingly lower power budgets and therefore would be suited for low resource ECC implementations.

Table 1: FPGA Implementation Results for Different Data Path Widths on Spartan III XC3S50 assuming *m =163.*

Data-path width	Look-up tables	Area (slice)	Freq. (MHz)	Throughput (Mbps)	Throughput /area kbit/s.Slices
8	596	319	96	0.87	2.73
16	796	439	94.417	1.63	3.72
32	1005	583	85.096	2.7	4.63
64	1247	697	82.066	5.2	7.5

Table 2: FPGA Implementation Results for Different Data Path Widths on Spartan III XC3S50 assuming *m =571.*

Data-path width	Look-up tables	Area (slice)	Freq. (MHz)	Throughput (Mbps)	Throughput /area kbit/s.Slices
8	596	319	96	0.256	0.8
16	796	439	94.417	0.53	1.2
32	1005	583	85.096	0.9	1.55
64	1247	697	82.066	1.74	2.5

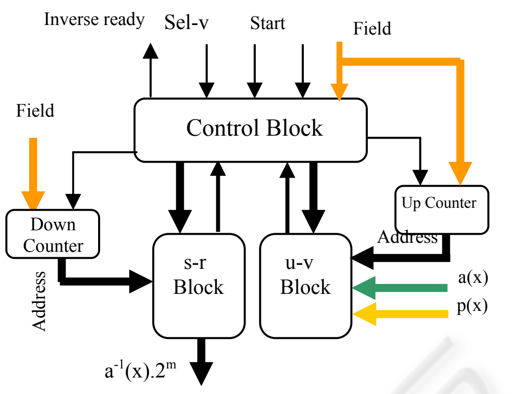

Figure 1: Montgomery inverse basic building block.

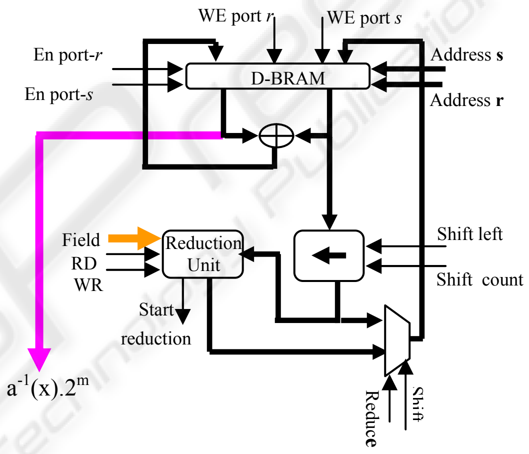

Figure 2: Montgomery inverse s-r block.

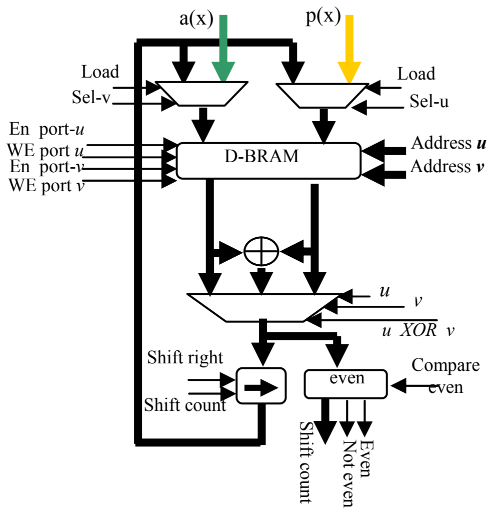

Figure 3: Montgomery inverse u-v block.

REFERENCES

D. Hankerson, A. Menezes, and S. Vanstone." *Guide to Elliptic Curve Cryptography*." Springer-Verlag, 2004.

N. Koblitz, "*Introduction to Elliptic Curves and Modular Forms*" Graduate Texts in Mathematics, Vol. 97, Springer, 1984.

P. L. Montgomery. "*Modular Multiplication without Trial Division*" Mathematics of Computation, vol.44. April 1985.

B. Kaliski."*The Montgomery inverse and its applications*". IEEE Transactions on Computers, Vol. 44**,** No.8, August 1995.

NIST "*Recommended elliptic curves for federal government tuse*", Available at http:// csrc.nist.gov/encryption/.2000.

M. Shieh. J.Chen, And C.Ming "*High-Speed Design of Montgomery Inverse Algorithm over GF(2^m)*" IEICE Trans. Fundamentals, Vol.E89–A, February 2006.

C. H. Kim, S. Kwon, J.J. Kim, C.P. Hong, "*A Compact and Fast Division Architecture for a Finite Field GF(2^m)*". ICCSA 2003, LNCS 2667, pp. 855-864, 2003.