Comparing Subtraction-Free and Traditional AMI

Jiří Buček, Róbert Lórencz
Department of Computer Science and Engineering
Faculty of Electrical Engineering, Czech Technical University
Karlovo nám. 13, 121 35 Praha 2, Czech Republic
Email: bucekj@fel.cvut.cz, lorencz@fel.cvut.cz

***Abstract*— This paper presents FPGA implementations of traditional Almost Montgomery Inverse and Subtraction-free Almost Montgomery Inverse and compares their space and time properties. The subtraction-free algorithm with its hardware architecture overcomes the disadvantages of currently known methods (e.g. [2]). The ">" or "<" tests that require either extra clock cycles or extra chip area are completely eliminated.**

I. Introduction

Calculation of the modular inverse forms a part of various cryptographic algorithms, such as the decipherment of the RSA algorithm [10], digital signature systems [9], and so on. It is especially important in computing point operations on elliptic curves [8], or in accelerating the modular exponentiation operation using the so-called addition-subtraction chains [1]. Many cryptographic applications require an efficient hardware implementation of the modular inverse due to its time complexity and also for security reasons.

The majority of algorithms for computing the modular inverse are derived from the Extended Euclidean Algorithm [5]. The modular arithmetic operations are often performed in a Galois Field GF(p), where p is prime. An efficient "left-shifting" algorithm for computing the modular inverse in GF(p) is described in [6]. This algorithm avoids the "less-than/greater-than" tests that are in GF(p) subject to carry propagation delays. The algorithms discussed in this paper assume GF(p) as well.

One class of modular inversion algorithms computes the Montgomery Modular Inverse (MMI, [4]), which is defined as $MMI(a) \equiv a^{-1}2^{2n} \pmod p$, where $n = \lceil log_2 p \rceil$. MMI is computed in two steps. The first step computes the Almost Montgomery Inverse (AMI), $AMI(a) \equiv a^{-1}2^k \pmod p$, where $k \in [n, 2n]$. The second step adjusts the result, multiplying it by 2^{2n-k}. In this paper, we focus only on the first step, thus computing AMI.

II. Subtraction-Free Almost Montgomery Inverse

This paper presents the first implementation of the Subtraction-Free AMI algorithm [7], which is a new variant of AMI published in [4].

The traditional algorithm for AMI uses the test $(u > v)$, which involves subtraction, and the sign of the difference determines the test result. If the condition is true, the result of this subtraction $(u - v)$ can be used in further processing (dividing by two, assigning into u). However, if the condition is false, result of the subtraction is negative and we need to compute the opposite value. When implementing in hardware, this requirement may present a significant overhead, since we must compute the difference $(v - u)$ or negate the previous result. A variant of the traditional AMI is given in Algorithm 1.

Algorithm 1, AMI with Subtractions

Input: $a \in [1, p-1]$ and $p > 2$ is prime
Output: $o \in [1, p-1]$ and k, where
$o = a^{-1}2^k \bmod p$ and $n - 1 \leq k < 2n$

1. $u \leftarrow p, v \leftarrow a, r \leftarrow 0, s \leftarrow 1, k \leftarrow 0$
2. while(1)
3. if ($u_{\mathrm{LSB}} == 0$) then
4. $u \leftarrow u/2, s \leftarrow 2s$
5. else if ($v_{\mathrm{LSB}} == 0$)
6. $v \leftarrow v/2, r \leftarrow 2r$
7. else
8. $x = u - v, y = r + s$
9. if ($x == 0$) then return $o \leftarrow s$ and k
10. if ($CARRY(x) == 1$) then
11. $u \leftarrow x/2, r \leftarrow y, s \leftarrow 2s$
12. else
13. $v \leftarrow (x = (v - u))/2, s \leftarrow y, r \leftarrow 2r$
14. $k \leftarrow k + 1$

Algorithm 2, Subtraction-Free AMI

Input: $a \in [1, p-1]$ and $p > 2$ is prime
Output: $o \in [1, p-1]$ and k, where
$o = a^{-1}2^k \bmod p$ and $n - 1 \leq k < 2n$

1. $u \leftarrow (-p), v \leftarrow a, r \leftarrow 0, s \leftarrow 1, k \leftarrow 0$
2. do
3. if ($u_{\mathrm{LSB}} == 0$) then
4. $u \leftarrow u/2, s \leftarrow 2s$
5. else if ($v_{\mathrm{LSB}} == 0$)
6. $v \leftarrow v/2, r \leftarrow 2r$
7. else
8. $x = u + v, y = r + s$
9. if ($CARRY(x) == 0$) then
10. $u \leftarrow x/2, r \leftarrow y, s \leftarrow 2s$
11. else
12. $v \leftarrow x/2, s \leftarrow y, r \leftarrow 2r$
13. $k \leftarrow k + 1$
14. while ($x \neq 0$)
15. return $o \leftarrow s$ and k

1-4244-0185-2/06/$20.00

Algorithm 2 computes AMI completely without subtractions, thus avoids the overhead of computing both $(u - v)$ and $(v - u)$. To achieve the same result as in Algorithm 1, it computes $(u+v)$, where one of the addends must be negative. By keeping u always negative and v always positive, we can compute an equivalent of the differences in the original algorithm, without subtraction.

It can be shown that in every iteration of the AMI calculation, same values appear in v, r, s in Algorithm 2 as in Algorithm 1, while opposite values appear in u [7].

The negative values in u are represented using two's complement code. This allows us to use normal binary adders. Moreover, the information about the sign of u does not need to be stored explicitly, because u cannot be positive (and v cannot be negative). Therefore, we do not need any extra bits for storing the negative values, compared to the number of bits required to represent the always positive values in Algorithm 1.

By avoiding the $(u > v)$ test, our approach either saves chip area (compared to [2]; instead of two subtractors, one needs a single adder) or performs faster (compared to [3]; no extra clock cycles are needed). Furthermore, there is no need for multi-function arithmetic units (adders/subtractors).

There are other, more complex architectures for AMI and MMI, which deal with inversion of very long numbers (thousands of bits), thus focus on scalability [11] or use carry-free arithmetics [12]. Our implementations focus on shorter numbers (hundreds of bits), which are suitable for elliptic curves cryptography, and therefore we focus mainly on small and fast data paths and controllers.

A. Time Complexity

The time complexity of Algorithm 1 and 2 is determined by the number of operations in every iteration, which correspond to the iterations of the binary GCD algorithm. Knuth mentions in [5] that, for binary $\gcd(u, v)$, where $u, v \in [1, 2^n]$, the number of iterations and shifts $k \approx 2qn$, number of additions/subtractions $s \approx qn$, number of $(u > v)$ tests $t \approx (qn)/2$ (Algorithm 1 only), where $q \approx 0.70597$.

Assuming that all operations (additions, subtractions, shifts, tests) are performed in a single cycle and that we are able to add and shift in a single cycle, the speedup S of AMI calculation using Algorithm 2 compared to Algorithm 1 is: $S \approx \frac{k+t}{k} = 1 + \frac{(qn)/2}{2qn} = 1.25$. If additions, subtractions and $(u > v)$ tests take more clock cycles than shifts because of carry propagation delay, which would be true for large n ($n > 512$), then the speedup is

$$S(f(n)) \approx \frac{k+f(n)s+t(1+f(n))}{k+f(n)s} = 1 + \frac{(1+f(n))/2}{2+f(n)}, \quad (1)$$

where $f(n)$ is a real function of a positive integer n such that $0 \leq f(n) \leq f(n+1)$ for all n. The function $f(n)$ represents additional clock cycles due to carry chain propagation delay. Using equation (1) with $f(n) = 0$ for some small n we obtain $S \approx 1.25$ as shown above. If $n \to \infty$, hence $f(n) \to \infty$, we obtain the maximum theoretical speedup

$$S_{max} \approx \lim_{f(n)\to\infty} S(f(n)) = 1.5,$$

which can be achieved with the proposed Algorithm 2 compared to Algorithm 1.

This of course assumes that operations with r and s are performed in parallel with operations performed with u, v as specified in Algorithm 1 and 2.

Unlike Algorithm 1, Algorithm 2 needs the negative value of p. It can be pre-calculated once for use in AMI as well as in other operations that require reductions modulo p. For example, $-p$ is actually more useful than p even for subsequent AMI to MMI conversion. The AMI only needs to be shifted certain number of times, subtracting p (that is, adding $-p$) whenever necessary to keep the result within $[1, p-1]$. Again, only an adder is needed.

III. AMI Implementations with Subtractors

When implementing AMI in hardware, we can divide the data path into the master part that computes $\gcd(u, v)$, and the slave part that computes the inverse (uses r, s). We have studied three different hardware units for computing AMI, which differ in the implementation of the master part of the data path, and in the corresponding controller.

These include AMI with one subtractor and AMI with two subtractors, which implement Algorithm 1, and Subtract-free AMI, which implements Algorithm 2. The slave part of the data path is always the same, it contains one adder for computing $(r + s)$.

Variants of implementation of Algorithm 1 involve computations of both $(u - v)$ and $(v - u)$. There are two possible approaches to this requirement – extra time or extra space (chip area). The first approach leads to AMI with one subtractor, which uses an extra clock cycle to compute the opposite difference on the same subtractor as the first difference. The second approach, AMI with two subtractors, uses the two subtractors to compute both differences concurrently.

A. AMI with One Subtractor

This implementation uses the time approach. The master part contains one subtractor to compute $x = (u - v)$. The output x of the subtractor is then used for the test $(u > v)$, and if the condition is true, x is used in subsequent computation. If the test condition is false (x is negative), the same subtractor is used again in the next clock cycle to compute the opposite difference $x = (v - u)$.

By using only one subtractor in the master part, we save area at the expense of extra clock cycles each time the test fails. However, some area overhead still applies – we have to connect multiplexers to the inputs of the subtractor. Hence the area saved by using only one subtractor depends on the particular technology platform, namely the area sizes of subtractors and multiplexers.

The number of clock cycles needed to compute AMI is greater than the number of iterations of the main loop of Algorithm 1

B. AMI with Two Subtractors

This implementation uses the space approach. There are two subtractors in the master part, therefore both differences, $(u - v)$ and $(v - u)$, are always available. The area is larger due to the additional subtractor, but there are no additional clock cycles, therefore this implementation is faster than AMI with one subtractor. The number of clock cycles needed to compute the inverse corresponds to the number of iterations of the algorithm main loop.

IV. Subtraction-Free AMI Implementation

The subtraction-free AMI implementation contains one adder in the master part, and one adder in the slave part. The result of addition $x = (u + v)$ is always divided by 2 (shifted right), and written into either u or v; the destination register determined by the sign of the result – this information is contained in carry.

The resulting architecture computes AMI in the same number of cycles as AMI architecture with two subtractors, but it is considerably smaller (contains 1 less adder). It is even possible to be slightly faster due to more simple logic (simpler multiplexers, no need to switch between $(u-v)$ and $(v-u)$).

The subtraction-free architecture is faster than AMI architecture with one subtractor while being approximately equal in area occupation. It is even possible to be both smaller and faster because it lacks the extra multiplexers.

V. Results

We have implemented in VHDL three different architectures for computing AMI: Subtraction-free AMI (SF-AMI, Section IV), AMI with one subtractor (1-SUB-AMI, Section III-A), and AMI with two subtractors (2-SUB-AMI, Section III-B). The target platform is a FPGA device Xilinx Virtex2 3000.

We have synthesized the generic designs using bit lengths of $n = 64, 128, 162$ and 256 bits, the results are presented in Table I. Area occupation is given in slices, the numbers are gathered from the map report of the Xilinx toolchain. The minimum clock period is gathered from the post place and route static timing report.

An important metric is the Time×area product (Table II), which reflects both the minimum time needed for the computation of AMI and the chip area occupied by the unit. Therefore, the numbers for AMI with one subtractor also reflect the average slowdown of 1.25 caused by the extra clock cycles needed when an opposite subtraction must be performed.

TABLE I
AREA OCCUPATION (SLICES), MINIMUM CLOCK PERIOD (NS)

	SF-AMI		1-SUB-AMI		2-SUB-AMI	
n (bits)	area (slices)	period (ns)	area (slices)	period (ns)	area (slices)	period (ns)
64	347	15.328	411	14.631	393	14.855
128	638	14.369	801	15.189	710	17.777
162	917	19.660	955	19.709	1044	19.380
256	1272	24.591	1269	24.542	1491	24.610

TABLE II
TIME×AREA PRODUCT, SCALED (SLICES*NS/1000)

n (bits)	SF-AMI	1-SUB-AMI	2-SUB-AMI
64	5.3	7.5	5.8
128	9.2	15.2	12.6
162	18.0	23.5	20.2
256	31.3	38.9	36.7

VI. Conclusion and Future Work

We have implemented the recently published Subtraction-free Almost Montgomery Inverse algorithm in FPGA and compared it to two different architectures for the traditional Almost Montgomery Inverse algorithm with subtractions.

Implementation results show that the Subtraction-free AMI algorithm [7] is suitable for hardware implementation and its implementation is equally fast as the implementation of AMI with two subtractors, yet about 13–17% smaller in area.

The Subtraction-free AMI implementation is equally small as the implementation of AMI with one subtractor, yet it is about 25% faster. We have observed that for smaller bit lengths, the Subtraction-free AMI implementation is both smaller and faster.

Our experiments with FPGA implementations were influenced by the fact that FPGAs contain dedicated hardware structures for carry chains (adders and subtractors). Our future work will focus on implementing the Subtraction-free algorithm in ASIC. Adding and subtracting numbers to about 256 bits is implementable using conventional binary adders and these word lengths are suitable for elliptic cryptography, finding its application, among others, in smart cards.

References

[1] Ö. Eğecioğlu, Ç. K. Koç, Exponentiation Using Canonical recoding, Theoretical Computer Science 129 (2), 1994, pp. 407–717.

[2] A. Gutub, A. Tenca, Ç. Koç, Scalable VLSI Architecture for GF(p) Montgomery Modular Inverse Computation, Proceeding of the IEEE Computer Society Annual Symposium on VLSI, 2002.

[3] J. Hlaváč, R. Lórencz, Ordinary Modular Inverse Using AMI – Hardware Implementation, Proc. IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems–DDECS'03, Poznań, Poland, 2003, pp. 309–310.

[4] B. S. Kaliski Jr., The Montgomery Inverse and Its Application, IEEE Transaction on Computers 44 (8), 1995, pp. 1064–1065.

[5] D. E. Knuth, The Art of Computer Programming, Vol. 2 / Seminumerical Algorithms, Addison-Wesley, Reading, Mass. Third edition, 1998.

[6] R. Lórencz, New Algorithm for Classical Modular Inverse, Cryptographic Hardware and Embeded Systems - CHES 2002, Springer-Verlag LNCS 2523, 2003, pp. 57–70.

[7] R. Lórencz, J. Hlaváč, Subtraction-free Almost Montgomery Inverse Algorithm, Information Processing Letters 94 (1), 2005, pp. 11–14.

[8] A. J. Menezes, Elliptic curve Public Key Cryptosystem, Kluwer Academic Publishers, Boston, MA, 1993.

[9] Nat'l Inst. of Standards and Technology (NIST), FIPS Publication 186: Digital Signature Standard, 1994.

[10] J.-J. Quisquarter, C. Couvreur, Fast Decipherment Algorithm for RSA Public-key Cryptosystem, Electronics Letters 18 (21), 1982, 905–907.

[11] E. Savaş, M. Naseer, A. A-A. Gutub, Ç. K. Koç, Efficient unified Montgomery inversion with multibit shifting, IEE Proceedings – Computers and Digital Techniques 152 (4), 2005, pp. 489–498

[12] E. Savaş, A Carry-Free Architecture for Montgomery Inversion, IEEE Transactions on Computers 54 (12), 2005, pp. 1508–1519