Area-Time-Power of Modular Multipliers implemented in FPGA

Gustavo Sutter[1] , Jean-Pierre Deschamps[2], and Eduardo Boemo[1]

[1]School of Engineering, Universidad Autónoma de Madrid, Spain
{gustavo.sutter, eduardo.boemo}@uam.es
[2] Dept. Eng. Electrònica, Elèctrica i Automàtica, Universitat Rovira i Virgili ,Tarragona, Spain

Abstract. Three modular multiplication algorithms are described and compared: the so-called Multiply and Reduce, the Shift and Add, and finally, the Montgomery product. An estimation of the cost of their combinational implementation using Xilinx FPGAs family is calculated. Practical results in term of area, delay, and power for both combinational and completely sequential implementations are presented.

1 Introduction

For both performance and physical security, it is often advantageous to implement cryptographic algorithms in hardware. These algorithms can be implemented in smart cards and others portable devices, where not only the speed and area are important, but also the power consumed.

Many public-key algorithms are based on finite ring Z_m and finite field $GF(p^k)$ operations like RSA [1] and Elliptic Curves [2]. As a consequence, the development of optimized VHDL models for modular multipliers is the starting point in this area. When used as a computation primitive for calculating the exponential function y^x mod m, the modular multiplication is generally performed using the Montgomery algorithm [3]. It is an efficient method oriented to a software implementation of the modular exponentiation [4], but frequently has been translated to hardware in RSA ciphering/deciphering and key generation circuits [5, 6, 7]. Nevertheless, as is explained in section 3.5.3, the algorithm is inefficient for single modular multiplications [4].

In this paper, three modular multiplication algorithms are described and compared: Multiply and Reduce, Shift and Add, and finally, Montgomery Product. These algorithms are described in section 2. The area and delay figure of their combinational implementation using Xilinx 4K serie FPGAs are presented in section 3. In next section, the analysis is extended to the completely sequential implementation of the three algorithms. In section 5, the power is also analysed and finally, the main conclusions are summarized in section 6.

2 Algorithms

Given three natural numbers *x, y* and *m* such that: $x < m$, $y < m$ and $m < 2^n$, three algorithms are presented - and their corresponding implementations are compared - for calculating: $z = x.y$ mod m. The features of these algorithms are summarized bellow.

2.1 Multiply and reduce

This algorithm consists of: 1) multiplying *x* by *y*, obtaining a 2*n*-bit intermediate result *p*, and 2) reducing *p* modulo *m*.

The first operation, the multiplication of two natural numbers, can be decomposed in a series of left shifts and conditional sums (section 3). It is the classical shift and add algorithm.

algorithm 1

```
p := 0;
for i in 0 .. n-1 loop    p :=( p + x(i)*y)/2;  end loop;
p := p*(2**n);
```

The second operation, the modulo *m* reduction, can be decomposed in a sequence of left shifts, subtractions, and branching. The recovering algorithm is similar to the paper and pencil division method (algorithm 2).

algorithm 2

```
module := m*(2**n);
r(0) := p;
for i in 1 .. n loop
    remainder := (2*r(i-1))-module;
    if remainder < 0 then r(i) := 2*r(i-1);
    else r(i) := remainder; end if;
end loop;
z := r(n) / 2**n;
```

algorithm 3

```
module := m*(2**n);
r(0) := (2*n) - module;
for i in 1 .. n-1 loop
    if r(i-1)<0 then r(i) := (2*r(i-1))+ module;
    else r(i) := (2*r(i-1))- module; end if;
end loop;
z := r(n-1) / (2**n);
if z < 0 then z := z + m; end if;
```

A slightly different algorithm (non-recovering algorithm) uses a sum-subtraction primitive, where the operation selection depends on a previously calculated binary condition (algorithm 3).

Is important to observe that the recovering algorithm includes a branching, based on the bit sign of a result (*remainder*) calculated during the same iteration step, while the non-recovering algorithm branching is based on the bit sign of a result *r*(*i*-1), calculated in the previous iteration step.

2.2 Shift and add

In this case, instead of multiplying, obtaining a 2*n*-bit result, and reducing modulo *m,* the solution is reducing modulo *m* in every step of the shift and add algorithm:

algorithm 4

```
z := 0;
for i in 1 .. n loop
    z := (z*2 + x(n-i)*y) mod m;
end loop;
```

The maximum value of $2.z+x(n-i).y$ is: $2.(m-1) + (m-1) = 3.(m-1)$.
Thus: $2.z + x(n-i).y = m.q + r$, where $q \in \{0,1,2\}$.

As a consequence, the calculation of $z.2+x(n-i).y.$ modulo *m* can be performed as is done in algorithm 5. Additionally, this algorithm can be simplified. On the one hand *p*2 and *p*3 cannot be simultaneously negative: $p2 = 2.z + x(n-i).y – m.$ So that $-m \le p2 < 2.m$; if $p2 < 0$ then: $p3 = p2 + m \geq -m + m = 0$.

On the other hand, instead of computing: $p2 = p1 + x(n-i).y - m$, the value of $k = m-y$ can be previously computed (outside the *for* loop) so that $p2 = p1-m$ or $p2 = p1-k$. The final algorithm is called algorithm 6.

algorithm 5

```
p1 := z*2;
p2 := p1 + x(n-i)*y - m;
if p2 < 0 then   p3 := p2 + m; z := p3;
else
   p3 := p2 - m;
   if p3 < 0 then z := p2; else z := p3; end if;
end if;
```

algorithm 6

```
z := 0; k := m-y;
for i in 1 .. n loop
   if x(n-i) = 0 then w := m; else w := k; end if;
   p1 := z*2; p2 := p1 - w;
   if p2<0 then p3:=p2+m;else p3:=p2-m; end if;
   if p3<0 then z := p2; else z := p3; end if;
end loop;
```

2.3 Montgomery multiplication

If m is odd, then the greatest common divisor of 2^n and m is 1, so that there exits a natural number, denoted 2^{-n}, such that $2^{-n}.\ 2^n = 1 \bmod m$. Montgomery 3. proposed an algorithm that calculates: $z = x.y.\ 2^{-n} \bmod m$. In the so-called Montgomery product, every iteration step consists of two conditional sums:

algorithm 7

```
r(0) := 0;
for i in 1 .. n loop
    a := r(i-1) + x(i-1)*y;
    r(i) := (a + a(0)*m)/2;
end loop;
if r(n) < m then z := r(n); else z := r(n) - m; end if;
```

This algorithm does not calculate $x.y \bmod m$. Nevertheless, it can be observed that if x, y and $z = x.y \bmod m$, are substituted by $x' = x.2^n \bmod m$, $y' = y.2^n \bmod m$ and $z' = z.2^n \bmod m$, then: $z' = x'.y'.2^{-n} \bmod m$.

The result z' is the Montgomery product of x' and y'. In other words, a transform that applies Z_m in Z_m can be defined ($a \rightarrow a.2^n \bmod m$) such that the mod m product is substituted by the Montgomery product within the transformed domain. A direct transform $a \rightarrow a.2^n \bmod m$ is equivalent to the Montgomery multiplication of a by $2^{2.n} \bmod m$, and the inverse transformation $a' \rightarrow a'.2^{-n} \bmod m$ is equivalent to the Montgomery multiplication of a' by 1.

The classical exponentiation algorithm, based on a sequence of multiplications, that calculates $e = y^x$ is depicted in algorithm 8.

algorithm 8

```
e := 1;
for i in 1 .. n loop
   e := e*e;
   if x(n-i) = 1 then e := e*y; end if;
end loop;
```

algorithm 9

```
e := one_m;
y := MM(y,two_m);
for i in 1 .. n loop
   e := MM(e,e);
   if x(n-i) = 1 then   e := MM(e,y);   end if;
end loop;
e := MM(e,1);
```

In order to calculate $e = y^x \bmod m$, the previous algorithm must be modified: 1 and y are substituted by $1.2^n \bmod m$ and $y.2^n \bmod m$, the integer product is substituted by the Montgomery operation, and the final result e is substituted by $e.2^{-n} \bmod m$. Suppose that the value of *one_m* = 2^n mod m and *two_m* = $2^{2.n}$ mod m have been previously calculated for all usefull values of m. Then, the algorithm 9, where *MM* is a procedure that calculates de Montgomery product, computes $e = y^x$ mod m.

3 Synthesis and hardware mapping

In this section, the XC4025 implementation of the three proposed algorithm (Multiply and Reduce, Shift and Add, and Montgomery product) are compared. The proposed algorithms (1 and 2 or 3, 6 and 7) can be implemented with the following computation primitives:

sum: $r : a+b$
subtraction: $r = a + (2^n - b)$
sum - subtraction: $r = a + (1-x).b + x. (2^n - b)$
conditional sum: $r = a + x.b$
conditional subtraction: $r = a + x. (2^n - b)$
selection: $r = (1-x).a + x.b$

where a and b are n-bit numbers and x a one-bit number.

All of them can be synthesized with $n/2 + 1$ Configurable Logic Blocks (CLB) of the XC4K FPGA family, except the selection (a two-to-one n-bit multiplexer) that is synthesized with $n/2$ CLBs.

As main result, the cost of a two-to-one multiplexer is practically the same as the cost of a programmable adder-subtractor ($n/2$ vs. $n/2 + 1$). That is, n look-up tables. This fact is a consequence of the FPGA structure, based on LUTs. In the case of a Standard Cell implementation, the conclusion would be quite different.

3.1 Multiply and Reduce

The product (algorithm 1) includes n conditional sums, so that the corresponding cost is equal to $n.(n/2 + 1)$ CLBs.

The reduction, performed with algorithm 2, would need n conditional subtractions and n multiplexers, while algorithm 3 only includes n-1 sum - subtractions, an additional initial subtraction and a final conditional sum. The corresponding cost is $n.(n/2 + 1) + n/2 + 1$ CLBs. The total cost is equal to:

$C_{\text{multiply and reduce}} = n^2 + 2{,}5.n + 1$ CLBs.

3.2 Shift and Add

The implementation of algorithm 6 needs: an initial subtraction (computation of k), n multiplexers (selection of w), n n+1-bit subtractions (computation of $p2$; an extra bit is necessary in order to detect the sign of $p2$), n n+1-bit sum-subtractions (in the computation of $p3$, an extra bit is necessary in order to detect the sign of $p2$), and finally, n multiplexers (selection of z). The corresponding cost is equal to $n/2 + 1 + n.(n/2 + (n+1)/2 + 1 + (n+1)/2 + 1 + n/2)$. That is:

$C_{\text{shift and add}} = 2.n^2 + 3{,}5.n + 1$ CLBs.

3.3 Montgomery multiplier

Algorithm 7 includes: n n+1-bit conditional sums (a), n n+2-bit conditional sums ($r(i)$), an n+1-bit subtraction ($r(n) - m$), and an n-bit multiplexer (selection of z). The corresponding cost is equal to $n.((n+1)/2 + 1 + (n+2)/2 + 1) + (n+1)/2 + 1 + n/2$, that is to say

$C_{\text{Montgomery}} = n^2 + 4{,}5.n + 1{,}5$ CLBs.

3.4 Comparison

The three types of modular multipliers (named m_r: multiply and reduce, s_a: shift and add, and mont: Montgomery multiplier) have been implemented in an XC4025E array. The initial description is a synthesizable VHDL model using the arithmetic function packages of the IEEE. The sum-subtraction, conditional sum, conditional subtraction, and selection primitives have been modelled using *if then else* sentences:

```
if x='0' then r <= a+b; else r <= a-b; end if;
if x='0' then r <= a; else r <= a + b; end if;
if x='0' then r <= a; else r <= a - b; end if;
if x='0' then r <= a; else r <=  b; end if;
```

In Table 1, the implementation results are summarized. The 24 and 32-bit versions do not fit in the selected FPGA model.

Table 1. Number of CLBs and maximum delay (*ns*)

	Area (CLBs)			Delay (ns)		
bits	m_r	s_a	mont.	m_r	s_a	mont.
8	85	157	102	186	201	167
16	297	563	334	454	724	325
24	637	1232	694			
32	1104	2160	1166			

Observe that the actual cost values are very similar to the predicted ones. The practical conclusions are:

1. The cost of the Multiply and Reduce algorithm and of the Montgomery multiplier is almost the same ($n^2 + 2{,}5.n + 1$ vs. $n^2 + 4{,}5.n + 1{,}5$). Nevertheless, the Montgomery multiplier is the fast circuit.
2. The cost of the Shift and Add algorithm is almost twice the two previous ones ($2.n^2 + 3{,}5.n + 1$), due to the relatively high weight of the multiplexers.

The Shift and Add algorithm should be discarded (at least for a combinational implementation). As regards the choice between the Multiply and Reduce and the Montgomery product, the following facts must be taken into account:

1. The Montgomery multiplication (algorithm 7) is the faster.
2. The Montgomery exponentiation (algorithm 9) needs the previous computation of 2^n mod m and $2^{2.n}$ mod m for all the useful values of m. These values should be stored in a memory. An alternative solution could be to design a specific configuration of the FPGAs as a function of the value of m.

3. The Montgomery multiplier does not compute $z = x.y$ mod m but $z'' = x.y.2^{-n}$ mod m. In order to obtain z from z'' , the value of $2^{2.n}$ mod m must be known, and a second Montgomery product is necessary to compute $z = z''.2^{2.n}.2^{-n}$ mod m. As result, the Montgomery method is inefficient for computing a single product.

4 Sequential implementation

For large values of n, the circuit must be sequentialized, at least partially. The three types of modular multipliers have been implemented and compared under the following hypothesis: the (completely) sequentialized circuit implements the body of the main iteration of algorithms 1 and 3 (m_r), 6 (s_a) and 7 (mont.), respectively, as well as the additional registers, counter, and control logic.

The following additional resources are necessary: registers, shift registers, and counters. In a XC4K-family CLB, an n-bit register can be synthesized with $n/2$ CLBs. The minimum number of CLBs for an n-bit counter (up to 2^n states) is equal to $n/2$. Nevertheless, additional CLBs can be necessary; the exact number of them depends on the specific features of the counter (bidirectional, programmable, with clock enable, etc.). As a rule of thumb, it will be assumed that the cost of an n-bit counter is of the order of n CLBs.

The sequential execution of algorithms 1 and 2 (Multiply and Reduce) needs the following blocks: an n-bit conditional sum, an n-bit sum-subtraction, an n-bit conditional sum (final step), a $2.n$-state counter, two n-bit shift registers, a $2.n$-bit register, and a 4-state machine. The corresponding cost is of the order of $n/2 + 1 + n/2 + 1 + n/2 + 1 + \log_2(2.n) + 2.(n/2) + (2.n)/2 + 4$. That is to say:

$$C_{\text{Multiply and Reduce}} = 3{,}5.n + \log_2 n + 8.$$

The sequential version of algorithm 6 (Shift and Add) includes: an n-bit subtraction, two n-bit multiplexers, an n+1-bit subtraction, an n+1-bit sum-subtraction, an n-state counter, an n-bit shift register, and an n-bit register.
The corresponding cost is of the order of $n/2 + 1 + 2.n/2 + (n+1)/2 + 1 + (n+1)/2 + 1 + \log_2 n + n/2 + n/2$. As a consequence:

$$C_{\text{Shift and Add}} = 3{,}5.n + \log_2 n + 4.$$

The sequential execution of algorithm 7 (Montgomery) needs: an n+1-bit conditional sum, an n+2-bit conditional sum, an n+1-bit subtraction, an n-bit multiplexer, an n-state counter, an n-bit shift register, and an n+1-bit register. The corresponding cost is of the order of $(n+1)/2 + 1 + (n+2)/2 + 1 + (n+1)/2 + 1 + n/2 + \log_2 n + n/2 + (n+1)/2$. That is to say:

$$C_{\text{Montgomery}} = 3.n + \log_2 n + 5{,}5.$$

Table 2. Number of CLBs, Flip-Flops and Maximum frequency (*MHz*)

bits	Multiply and reduce			Shift and Add			Montgomery		
	CLBs	FF	Frec	CLBs	FF	Frec	CLBs	FF	Frec
8	57	67	25	33	37	17,2	34	31	32,1
16	72	124	22,4	63	70	12,7	59	56	25,8
32	126	237	16,9	119	135	7,1	108	105	24,4
64	240	462	-	232	264	-	204	31	-
128	465	911	-	457	521	-	398	56	-
256	915	1808	-	905	1034	-	783	105	-

Table 2 shows the number of CLBs and the maximum clock frequency (in Megahertzs). The total number of clock cycles is equal to n in the case of the Shift and Add and Montgomery multipliers, and equal to $2.n$ in the case of the Multiply and Reduce one. Observe that the actual cost values are very similar to the computed ones.

5. Power Consumption

In order to measure the power consumption, random vectors sequences were generated. The dynamic power was isolated from the others components using the technique of Table 3. Each circuit was measured at 100 Hz, 2, 3, 4 and 5 MHz, and the static power consumption was extrapolated.

All prototypes include tri-states buffers at the output pads to measure the off-chip power. Besides, each pad support the load of the logic analyzer, lower than 3pf [11].

The VHDL code was synthesized using the FPGA Express [8, 9] and the Xilinx tools [10] into a XC4010EPC84-1 FPGA sample. All circuits has been implemented and tested under identical conditions.

Table 3: Determination of power component in arithmetic circuits

Dynamic Power	In a CMOS circuits as: $P = \sum_{all\ nodes} c_n f_n V_{DD}^2$ where, c_n is the load capacitance at the output of the node n, f_n the frequency of switching and V_{DD} supply voltage. To calculate it, the total power is measured and then the static, off-chip and synchronization power is deducted.
Static power	The chip is configured but neither stimulus nor clocking is applied. The pull-up resistors and other external elements that require the FPGAs remain connected.
Off-chip power	The circuit is measured twice. First, during normal operation. Second, by disabling the tri-state output buffers. Thus, the off-chip component can be approximated to the difference between the two results. In addition, the use of the tri-state buffers in low-power design is also useful to separate the results from a particular PCB.
Synchro-nization power	A constant data (for example, all bit zeroed) is inputted to the circuit, meanwhile the clock signal is applied. Thus, only the clock tree has activity. Is important to note that FPGAs use multiplexers to emulate the effect of a clock enable. As a consequence, the use of the clock enable pin of a CLB does not interrupt the clocking of the flip-flops.

5.1 Combinational implementations

The input/output of the sequential multipliers was registered. Eight bit wide data path was chosen in order to fit into the targeted FPGA. The table 4 shows the Power-Area-Time figure of the circuits.

Table 4. Area-Time-Power of the combinational multipliers

	M_r	s_a	mont.
Energy (nJoules)	96,0	186,4	92,7
Area (CLBs)	85	157	102
Time (ns)	186	201	167

Observe that Montgomery implementation consumes a less power than multiply and reduce, in spite of the bigger area used. The Montgomery algorithm has about 4% less output transition for the test pattern utilized. This is caused by the fact that Montgomery not compute $z = x.y \bmod m$ but $z'' = x.y.2^{-n} \bmod m$.

The measurements shows that the Multiply and Reduce and the Montgomery algorithms, has almost de same values not only in area-delay but also in power. Nevertheless, the Montgomery multiplier is a little faster and consumes less power. The power of the Shift and Add algorithm (as the area) is almost twice the previous ones.

5.2 Sequential implementations

In this case, the dynamic power was divided into clock power (due to clock and FF) and combinational power (due to datapath). Main result in that the synchronization power in the multiply and reduce circuit is increase linearly with the number of flip-flop.

Table 5. Area-Time-Power of the sequential multipliers

	m_r	s_a	mont.
Dynamic Energy (nJoules)	71,5	52,4	38,6
Synchronization Energy (nJoules)	46,8	26,2	27,2
Combinational Energy (nJoules)	24,7	26,2	11,1
Area (CLBs)	57	33	34
Flip - Flops	67	37	31
Total Time (ns)	320	465	249

The sequential implementation the Montgomery algorithm consume less power than the others alternatives. The multiply and reduce circuit has the worst power figure and uses twice cycles to compute the result.

Observe that the energy consumed (power x time) is lower in the sequential implementation. It can be explained due to the reduction of glitches produced by the registered stages [12]. Nevertheless, the clock power is grater than combinational power.

6. Conclusions

For calculating $e = y^x \bmod m$, where m belongs to a known set of values (in such a way that the values of 2^n and $2^{2.n}$ modulo m can be previously tabulated), the Montgomery algorithm is definitely exhibit the best Area-Time-Power figure, independently of the type of implementation (combinational or sequential).

For calculating $z = x.y \bmod m$, the combinational implementation of the Multiply and Reduce algorithm is better than the Shift and Add algorithm. However, in the sequential implementation, both approach (Multiply and Reduce, Shift and Add) present similar results in area and bandwidth (taking into account that the Multiply and Reduce version needs $2.n$ cycles instead of n), but the power consumption is lower in the Shift and Add algorithm.

Other approaches are under study. Among others, the definition of serial arithmetic algorithms and the generation of hard-macros optimized at the physical level are being considered. Another interesting approach is the use of reconfigurability to define specific circuits for every value of *m*. Finally, other granularity of sequential implementation can be studied to obtain different area-time-power trade-off.

Acknowledgement: This work is supported in part by Project TIC2001-2688-C03-03 of the Spanish Ministry of Education and Science, and in part by Project 07T/0052/2003-3 of the *Consejería de Educación de la Comunidad de Madrid*, Spain

References

1. R. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Commic. of the ACM, vol.21,n^{o}2, pp.120-126, February 1978.
2. I.Blake, G.Seroussi and N.Smart, "Elliptic Curves in Cryptography", Cambridge University Press, 1999.
3. P.Montgomery, "Modular multiplication without trial division", Mathematics of Computation, vol.44, pp. 519 - 521, April 1895.
4. Menezes, P. van Oorschot and S. Vanstone, "A Handbook of Applied Cryptography", CRC Press, 1996.
5. V.Fisher and M.Drutarovský, "Scalable RSA Processor in Reconfigurable Hardware - a SoC Building Block", XVI Conference on Design of Circuits and Integrated Systems, November 2001, Porto, pp. 327 - 332.
6. D.Matilla, M.López-Vallejo and A.Rojo, "Hardware - Software Co-design of a Cryptographic Application", XVI Conference on Design of Circuits and Integrated Systems, November 2001, Porto, pp. 100 - 105.
7. T.Blum and C.Paar, "Montgomery Modular Exponentiation and Reconfigurable Hardware", 14th IEEE Symp. on Computer Arithmetic, April 1999, Adelaide, Australia.
8. FPGA Compiler II / FPGA Express VHDL Reference Manual, Version 1999.05, Synopsys, Inc.,May 1999
9. FPGA Express page. Synopsis, inc.; www.synopsys.com/products/fpga/fpga_express.htm
10. Software Manuals and documentation for Foundation Series 3.1i. http://toolbox.xilinx.com/docsan/3_1i/
11. Tektronix inc., “TLA 700 Series Logic Analyzer User Manual”, available at http://www.tektronix.com.
12. E. Boemo, G. Gonzalez de Rivera, S.Lopez-Buedo and J. Meneses, "Some Notes on Power Management on FPGAs", Lecture Notes in Computer Science, No.975, pp.149-157. Berlin: Springer-Verlag 1995.