FPGA Implementation of Montgomery Modular Multiplier

Velibor Škobić, Branko Dokić, and Željko Ivanović

Abstract – Montgomery architectures of modular multipliers with one or two bits scanning are described in this paper. Multipliers have been described using hardware description language – VHDL, and implemented on FPGA integrated circuit EP4CE115F29C7. Comparative analysis of multiplier regarding minimum calculation time, maximum operating frequency and number of used logic elements of integrated circuit is given. Based on implemented modules, analysis of RSA module for data encryption is performed.

Keywords – Modular multiplication, Montgomery algorithm, RSA algorithm

I. INTRODUCTION

Data protection can be achieved using symmetric and asymmetric algorithms. Encryption procedure using symmetric algorithms occurs by substitution, transposition, shifting, as well as logic operations (XOR) over data bytes (AES). These operations are much simpler for hardware implementation, which leads to lower number of resources and higher operating speed of encryption module. Symmetric algorithms are mostly used for data encryption. Asymmetric algorithms are for one order of magnitude slower than symmetric ones. They are used for keys exchange and digital signatures. Data protection occur using quite demanding mathematical operations. One of the most commonly used asymmetric algorithms is RSA algorithm [1]. Data protection procedure occurs in a way that key message (P) is exponentiated onto public key (e), and then determine the remaining of dividing operation with public key (m). For hardware realization of RSA algorithm, binary algorithm of modular exponentiation is typically used [2, 3]. Using binary algorithm, modular exponentiation procedure is reduced to iterative modular multiplication ($A{\cdot}B\mathrm{mod}m$). Some of modular multiplication algorithms are described in [2]. One of the most effective algorithms is Montgomery algorithm [4]. Calculation procedure is carried out by iterative summation. This algorithm is highly efficient and very simple for hardware implementation. It is used in algorithms with a large number of modular multiplications.

In second chapter, fundamentals of Montgomery algorithm modular multiplication are given. Two ways of Montgomery modular multiplication are described: with scanning of one bit and two bits. Two RSA data encryption algorithms using Montgomery modular multiplier have been described. In third chapter, hardware realization of Montgomery module is described. Fourth chapter presents simulation results of implemented modules. Results are summarized in the conclusion.

II. MONTGOMERY ALGORITHM

A. Montgomery algorithm

Montgomery algorithm is efficient and simple for hardware implementation. The result of modular multiplication is given by the following equation:

$$\mathrm{MonPro}(A,B,m) = A\cdot B\cdot R^{-1} \bmod m \qquad (1)$$

The advantage of this algorithm is that calculation is performed without dividing with m, but dividing with number R. Number R is taken in the form of 2^k, where k is the number of bits needed to represent input data. Number R^{-1} is inverse number of number R modulo m. For hardware implementation, dividing with 2^k is simple shifting operation of k bits to the right. As shown in equation (1), the result of modular multiplication contains number R^{-1}. This number can be eliminated so that input data A and B convert to leftover system modulo m (A=MonPro(A,R,m), B=MonPro(B,R,m)), and then the result of modular multiplication D converts so that multiplies with number 1 (D=Monpro(D,1,m)). As a result, we have $D=A{\cdot}B\mathrm{mod}m$. Montgomery algorithm modular multiplication is given by the following pseudo code:

Result $D=(A{\cdot}B)\mathrm{mod}m$
1. $D=0$
2. from $i=0$ to $i=k$-1
   a. $D = D + A{\cdot}B_i$
   b. $D = (D + D(0){\cdot}m)/2$
3. output D

Listing 1

Number k is the number of bits used for data representation. This algorithm passes through k iterations, where k is the number of bits used for representation of numbers A, B and m. Value of bits of number B (b_i) has been scanned. Depending on the value of scanned bit b_i number D is added to number A. Then, in order to perform reduction with 2 in every iteration, if the current result D is odd, number m (m is a prime number in RSA algorithm) is

added. If not, zero is added. For this realization, two adders, one shift register and control logic is needed.

Number of iterations can be reduced by scanning two bits of number B (b_i, b_{i+1}) in one iteration. Now, number of cycles needed for calculation is halved ($g=k/2$). Based on this, Montgomery algorithm pseudo code can be written as follows:

Result $D=(A\cdot B)\text{mod}m$
1. $D=0$
2. from $i=0$ to $i=g-1$
   a. $D = D + A\cdot B_i$
   b. $D = (D + D(0)\cdot m)/2$
   c. $D = D + A\cdot B_{i+1}$
   e. D = $(D + D(0)\cdot m)/2$
3. output D

Listing 2

Summation, depending of value of scanned bits b_i, b_{i+1}, can be grouped in one equation, as well as the condition for summation with number m. At the end, result is divided by 4, i.e. content is shifted two positions to the right. Based on these transformations, following algorithm is formed:

Result $D=(A\cdot B)\text{mod}m$
1. $D = 0$
2. from $i=0$ to $i=g-1$
   a. $D = D + A\cdot B_i + 2A\cdot B_{i+1}$
   b. $D = (D + U_1\cdot m + U_2\cdot 2m)$
   c. D = $D/4$
3. output D

Listing 3

Condition U_1 for summation of number m with result D has been defined using the equation:

$$U_1 = b_i A(0) \oplus D(0) \tag{2}$$

while condition U_2 for summation of number $2m$ with result is defined with:

$$\begin{aligned} U[1,0] &= D[1,0] + b_i \cdot A[1,0] + \\ &+ U_1 \cdot M[1,0] \\ &+ b_{i+1} \cdot 2 \cdot A[1,0] \\ U_2 &= U(1) \end{aligned} \tag{3}$$

First algorithm that scans only one bit b_i passes through k iterations. Second algorithm that scans two consecutive bits b_i and b_{i+1} passes through $k/2$ iterations. In the first one, summation is performed with numbers A and m, depending on current result value and value of bit b_i. Second one, in one iteration adds numbers A, $2\cdot A$, m and $2\cdot m$, depending on the state of conditions U_1, U_2 and values of b_i and b_{i+1}.

B. Carry Save Adders

For implementation of Montgomery algorithm modular multiplication, special attention is paid to the implementation of the adder. Better performances regarding speed can be adjusted using CSA (*Carry Save Adders*). CSA has three input and two output vectors. Summation result consists of vector C and vector S, represented in redundant form. Vector C represents carry bit vector, while vector S is vector of the current bit sum. The summation result is:

$$(C, S) = X + Y + Z \tag{4}$$

Carry bit in CSA realization does not propagate through full adders, but it's remembered in the shape of vector C. Propagation time of carry bit is eliminated, and hence result waiting time has been reduced. In order to get final result, vectors C and S needs to be summated using full adders, e.g. RCA (*Ripple Carry Adder*).

C. RSA algorithm

Montgomery modular multipliers are used for implementation of RSA module for data protection. Data encryption using RSA algorithm is performed as follows:

$$C = P^e \bmod m$$

where P is data that needs to be encrypted, e and m are public keys, and C encrypted data. Data decryption is performed as follows:

$$P = C^d \bmod m$$

where the pair of numbers d and m is a private key. Encryption and decryption procedures are very similar. For encryption, public key e has been used as exponent, and private key d for decryption. For hardware implementation of modular exponentiation, method of exponent bit scanning is suitable [2]. Depending of the direction of scanning, there are two methods: scanning from left to right and from right to left. Algorithm from right to left is shown by the following pseudo code [3]:

From right to left

Result $C = P^e \bmod m$
1. $K = 2^{2n} \bmod m$
2. $Z = Monpro(1, K, M)$
3. $P = Monpro(P, K, m)$

4. from $i = 0$ to $i = k - 1$
   a. if $e_i = 1$ then $Z = Monpro(Z, P, m)$
   b. $P = Monpto(P, P, m)$
5. $Z = Monpro(1, Z, m)$
6. $C = Z$

Listing 4

Algorithm from left to right is presented as [3]:

From left to right

Result $C = P^e \bmod m$

1. $K = 2^{2n} \bmod m$
2. $Z = Monpro(1, K, M)$
3. $P = Monpro(P, K, m)$
4. from $i = k - 1$ to $i = 0$
   a. $Z = Monpto(Z, Z, m)$
   b. if $e_i = 1$ then $Z = Monpro(Z, P, m)$
5. $Z = Monpro(1, Z, m)$
6. $C = Z$

Listing 5

For right to left algorithm, two Montgomery modular multipliers are needed working in parallel, while for left to right algorithm, only one sequentially operated multiplier is needed. Algorithm from left to right saves the number of Montgomery multipliers, while the number of calculation iterations doubles. Number of clock cycles for right to left algorithm is (k+3)(k+2), and for left to right algorithm 2(k+3)(k+2).

III. HARDWARE IMPLEMENTATION

Figure 1 shows Montgomery modular multiplier architecture with one bit scanning (Listing 1). It consists of *C_sig* and *S_sig* registers, shift register, multiplexer for signal routing, CSA network and control logic. Role of *C_sig* and *S_sig* registers is to temporary memorize the current result. Shift register on its output generate bit b_i. Signals 0 and *A* are routed depending of the value of bit b_i towards CSA network output. CSA network consists of two CSA. Inputs of CSA network are *C_sig* and *S_sig* signals, and output signals of first and second multiplexer. CSA network output signal states are memorized in *C_sig* and *S_sig* registers. Control logic controls the operation of the shift register, generates signal U_1 and controls drives the state of *C_sig* and *S_sig* registers. For the final result to come, k+2 clock cycles are needed. In first clock cycle, registers *C_sig* and *S_sig* reset and data *B* is written into shift register. Through next k cycles, summation defined by loop inside Listing 1 is performed. At k+2 clock cycles, result of modular multiplication (*C_sig* and *S_sig*) is converted to normal form (*D*) using full adder.

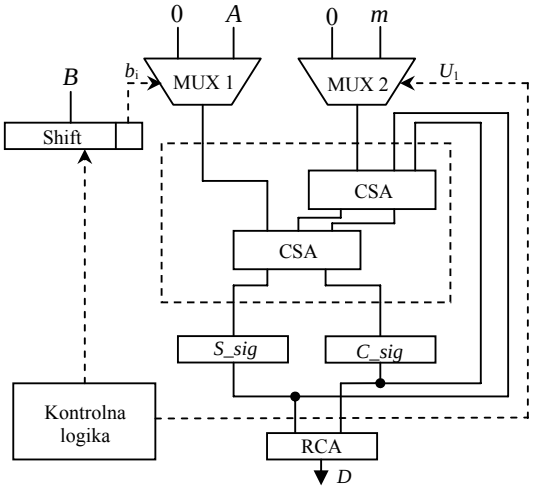

Fig. 1. Architecture of Montgomery modular multiplier with one bit scanning

Figure 2 shows architecture of Montgomery modular multiplier with two bits scanning (Listing 3). It consists of *C_sig* and *S_sig* registers, shift register, multiplexer for signal routing, CSA network and control logic. Role of registers *C_sig* and *S_sig* is to temporary memorize current result. Shift register on its output generate bites b_i and b_{i+1}. In dependence of bits b_i and b_{i+1} value, signals 0, *A* and 2*A* are routed towards CSA network input. Depending of the state of signal U_1, signals 0, *m* and 2*m* are routed towards CSA network over MUX2 multiplexer.

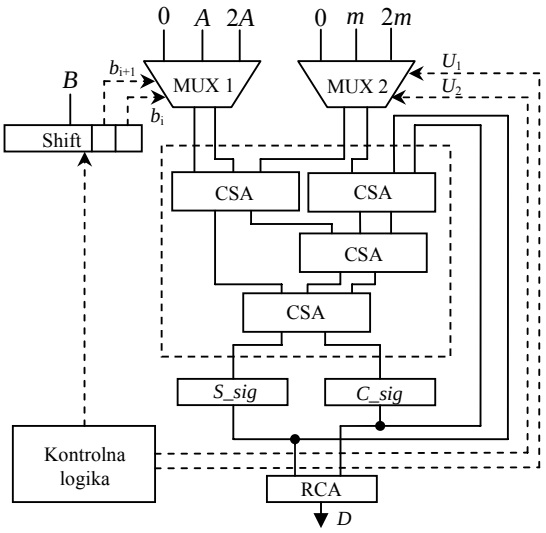

Fig. 2. Architecture of Montgomery modular multiplier with two bits scanning

This CSA network architecture consists of four CSA adders. CSA network inputs are *C_sig* and *S_sig* signals,

and output signals of first and second multiplexer. Output signal state of CSA network is then memorized into registers *C_sig* and *S_sig* registers. Control logic that runs shift register, generates U_1 and U_2 signals and control operating states of *C_sig* and *S_sig* registers. In order to obtain the final result, $k/2+2$ clock cycles are needed. In the first clock cycle, registers *C_sig* and *S_sig* reset and data *B* is written into the shift register. During the next $k/2$ cycles, summation defined by loop inside Listing 3 is performed. At $k/2+2$ clock cycle, result of modular multiplication (*C_sig* and *S_sig*) is converted in normal form (*D*) using full adder.

Based on presented architectures, two modules for Montgomery modular multiplication have been implemented. The first one is with on bit scanning *Montgomery_b1*, and second one with two bits scanning *Montgomery_b2*. Modules are described using hardware description language (VHDL) and synthesized by Quartus II and ModelSim software packages. Analyze of number of required resources for module implementation, maximum operating frequency and minimum calculation time has been performed.

Using implemented modules *Montgomery_b1* and *Montgomery_b2*, analyze of RSA module for data encryption is carried out. Based on RSA algorithm left to right and Montgomery modular multipliers, modules *RSA_Montgomery_dl_b1* and *RSA_Montgomery_dl_b2* for data encryption are implemented. These modules are analyzed in terms of number of used resources and maximum data encryption speed.

IV. Results

Implementation of the RSA algorithm on FPGA integrated circuit EP4CE115F29C7, family Cyclone IV, Altera [5] is done in this paper. This component contains 266 embedded multipliers (18x18 bits), 4 PLL blocks, 3888 Kbits of embedded memory, 528 I/O pins and 114480 logic elements. Preference for FPGA circuit is caused by availability, ease of system testing, flexibility, relatively good performances in the means of speed and power consumption.

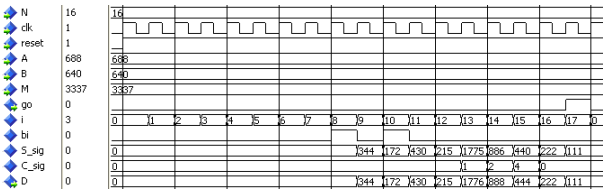

Fig. 3. Signal waveforms of Montgomery_b1 module

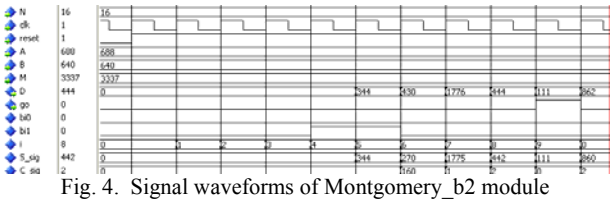

Fig. 4. Signal waveforms of Montgomery_b2 module

Figures 3 and 4 shows results of *Montgomery_b1* and *Montgomery_b2* modules, respectively. For input signal values such as: *A*=688, *B*=640 and *M*=3337, where *k*=16, result of Montgomery modular multiplication is the number *D*=111.

Figure 5 presents result of logic resources analysis needed for implementation of previously mentioned modules. *Montgomery_b1* module occupy less resources comparing to *Montgomery_b2* module, as a consequence of smaller CSA network and logic for signal routing.

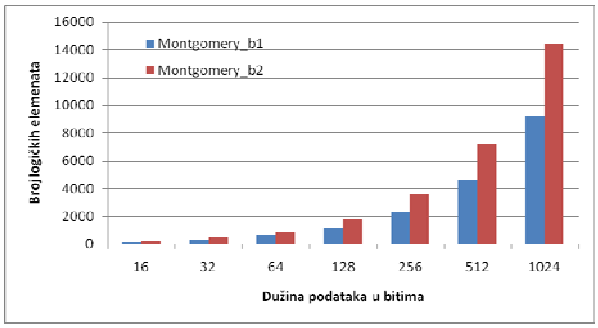

Fig. 5. Number of used logic elements as a function of data length

Table 1 shows module analysis results regarding maximum operating frequency. Based on given results, maximum operating frequency is obtained for *Montgomery_b1* module, at various data length. This is a consequence of lower signal propagation time through CSA network, consisted of two CSA adders.

TABLE I
MAXIMUM OPERATING FREQUENCY [MHZ]

k	*Montgomery_b1*	*Montgomery_b2*
16	323.94	160.77
32	188.04	159.08
64	206.83	157.65
128	234.03	152.86
256	252.4	152.84
512	239.87	145.31
1024	298.33	126.2
2048	173.19	126.98

Multiplying maximum operating frequency (Table I) with number of clock cycles used for processing one data, we get minimum calculation time for one data as a function

of data length. Those results are shown on Fig. 6. Lower calculation time is obtained for *Montgomery_b2* implementation. Maximum operating frequency of *Montgomery_b2* module is slightly lower then for *Montgomery_b1* module, but the number of clock cycles needed for calculation is twice lower, whereby better results are obtained with respect to calculation time.

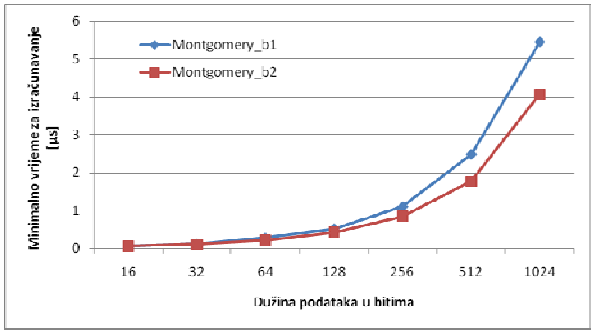

Fig. 6. Minimum calculation time as a function of data length

Analyzing RSA module for data encryption, following results are obtained. On Fig. 7, number of logic elements of implemented modules as a function of data length is shown.

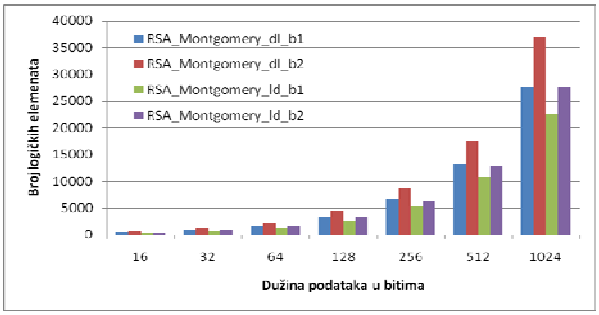

Fig. 7. Number of used logic elements as a function of data length

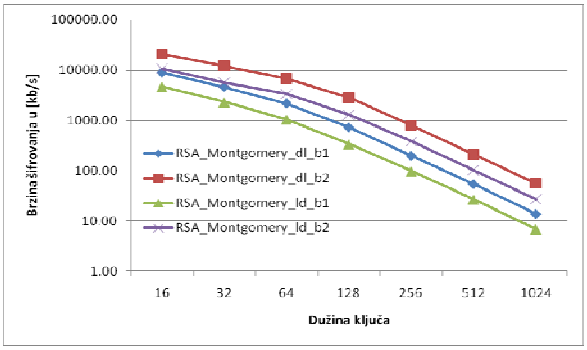

Fig. 8. Maximum data encryption speed as a function of key length

Least number of logic elements occupies implementation of left to right algorithm and Montgomery modular multiplier with one bit scanning. Most resources have been used by implementation of right to left algorithm and Montgomery modular multiplier with two bits scanning. Fig. 8 presents results of module analysis in terms of maximum data encryption speed. Best results have been obtained by right to left algorithm and Montgomery modular multiplier with two bits scanning. For key length of 1024 bits, maximum operating frequency is 55.32 [kb/s].

IV. Conclusion

Multipliers are implemented on Altera's Cyclone family FPGA integrated circuit EP4CE115F29C7. Synthesis and simulation are performed by Quartus II and ModelSim software packages. Number of used logic elements depends of data length and is higher for multiplies architectures with two bits scanning. For example, for data length of k =128 bits it is a 55% increase, and for k=1024 bits is 56%. Maximum operating frequency and minimum calculation time also depends of data length. Modular multiplier with one bit scanning has higher operating frequency, and lower calculation time. Maximum frequency of *Montgomery_b1* module is in the range about 324 MHz for k=16 up to 188 MHz for k=1024 bits. For *Montgomery_b2* module, this frequency is in the range of 161 MHz up to 126 MHz Calculation time is in the range of 0.05µs (k=16 bits) up to 5.45µs (k=1024 bits) for *Montgomery_b1*, and from 0.06µs up to 4.07µs for *Montgomery_b2*. Average calculation time in *Montgomery_b2* implementation is decreased for about 23% comparing to *Montgomery_b1* implementation. Data encryption speed is highest for *Montgomery_dl_b2* implementation. For key length of k=1024 bits, maximum encryption speed is 55.32 kb/s, and number of used logic elements is 36960. The lowest number of used logic elements is at *Montgomery_ld_b1* implementation. For key length of k=1024 bits, number of used logic elements is 22649, and maximum encryption speed 26.28 kb/s.

References

[1] R. L. Rivest, A. Shamir, L. Adleman, "*A Method For Obtaining Digital Signatures And Public-Key Crypto Systems*," Communications of the ACM, vol. 21, no. 2, pp. 120-126, Feb., 1978.

[2] C. K. Koc. "RSA Hardware Implementation". TR 801, RSA Laboratories, April 1996.

[3] V. Škobić, B. Dokić, Ž. Ivanović. "*FPGA Implementacija RSA algoritma*," Proccedings of 57th ETRAN Conference, Zlatibor, Serbia, June 3-6, 2013, pp.EL3.8.1-5.

[4] P. L. Montgomery, "*Modular Multiplication Without Trial Division*," Mathematics of Computation, vol. 44, no. 170, pp. 519-521, Abbrev. Apr., 1985.

[5] "Cyclone IV EP4CE115F29C7 Data Sheets," http://www.altera.com.