Very Fast Pipelined RSA Architecture Based on Montgomery's Algorithm

Iput Heri K, Asep Bagja N., Randy S. Purba and Trio Adiono

Electrical Engineering Department, School of Electrical Engineering and Informatics,
Bandung Institute of Technology
10th Ganesha Street, Bandung 40132.

iput_kurniawan@students.ee.itb.ac.id
randy232@students.ee.itb.ac.id
bagja212132@students.itb.ac.id
tadiono@paume.itb.ac.id

Abstract— **This paper present a design of RSA-Encryption using Pipelined radix-2 Montgomery's architecture. The architecture design exploits the algorithm to achieve high speed and efficient computation. The design separates the computation of Montgomery modular multiplication into different clock cycles to achieve high frequency clock. This design supports input from 1 to 14 block data and efficient in the number of total logic element and register. The design has been successfully verified whether functional Verilog RTL simulation, FPGA timing simulation and run in Signal Tap FPGA simulation. The design occupies logic elements 1157, 1030 registers, and able to run up to 261.85 MHz on Altera Cyclone II EP2C35 F672C6. The proposed design has been successfully synthesized using Synopsys with CMOS 0.18µ technology. The area is 63567.5 µm2 and the delay is 3.35 ns.**

Keywords— **RSA, Montgomery, FPGA, Pipeline Architecture**

I. INTRODUCTION

RSA encryption algorithm is widely used for such as email and files encryption, safe connection over insecure network like the internet, e-commerce and so on. In spite of widely used application, this algorithm has heavy computational and very time-consuming due to using of modular arithmetic in very large number. The basic computational of RSA algorithm is modular exponential that can be achieved using repeated modular multiplication. Thus good RSA encryption hardware has to be implemented using fast modular multiplication. However there is always trade-off between achieved speed and needed area.

This paper describes RSA algorithm hardware implementation. It was implemented based on Montgomery algorithm and using pipeline architecture inside Montgomery algorithm.

II. RSA-ENCRIPTION ALGORITHM

Let us define some integer parameters P as plain text, C as an encrypted text, E as the encryption key, D as the decryption key, and M as modulo number. The encryption can be made by following equation

$$C = P^E \bmod M \qquad (1)$$

mod is a modular operation.
On the other hand, the following equation is used for decryption

$$P = C^D \bmod M \qquad (2)$$

As we can see in Eq.(1) and Eq.(2), the RSA encryption expression itself is not complicated.
At first step how to define E, D, and M, we randomly choose quite large number of two prime factor p and q (p≠q). Then modulo M is defined as M = p x q. From p and q we define

$$L = LCM(p-1, q-1) \qquad (3)$$

where LCM stands for least common devisor.

In order to shorten the processing time, E is often chosen to be relatively small number. Lastly, the decryption key D should satisfy the following equation for arbitrary integer number H

$$E \cdot D = H \cdot L + 1 \qquad (4)$$

For handling computational complexity on equation (1) or (2), modular exponential can be simplified using repeated E times modular multiplication

$$C = P^E \bmod M$$

$$C = (((((P \times P) \bmod M) \times P) \bmod M)\ldots) \bmod M$$

One of methods used to compute modular multiplication is Montgomery Modular Multiplication. This algorithm replaces division operation by n operation with division by power of 2. Montgomery algorithm computes

$$MM(A,B) = A.B.r^{-1} \bmod M \qquad (5)$$

Where A, B < N and r such that GCD (M,r) = 1. For application in hardware we can choose r = 2^m, where m is the number of bit in M. In RSA M is odd as the product of two prime number, therefore the above constraint is always true.

A. Montgomery Modular Multiplication

Figure 1 show Montgomery Modular Multiplication Algorithm (radix 2) [1] for computing $A.B.R^{-1} \bmod M$

$$A = \sum_{i=0}^{m-1} 2^i a_i, a_i \in \{0,1\}$$

$$B = \sum_{i=0}^{m-1} 2^i b_i, b_i \in \{0,1\}$$

$$M = \sum_{i=0}^{m-1} 2^i m_i, m_i \in \{0,1\}$$

$$A, B < M; M < R = 2^m; GCD(M,R) = 1$$

```
1.  S = 0;
2.  for i = 0 to m-1
3.  $q_i$ = ($S_i$ + $a_i$B) mod 2
4.  $S_{i+1}$ = ($S_i$ + $q_i$M + $a_i$B)/2
5.  end for
6.  if $S_m \geq$ M
7.        $S_m$ – M
8.  else
9.        $S_m$
10. End
```

Fig. 1: Montgomery modular multiplication $A.B.R^{-1} \bmod M$

B. Optimized Montgomery Modular Multiplication

As we can see from the algorithm above in step 4, S_i is added with one of four possible values (0, B, M, B + M) during iteration. Thus, we can pre-compute these values and make selector to select value that is used inside the iteration [2]. Second improvement is done by removing the final comparison and reduction in step 6 to step 9 [1], since this operation needs a lot of resources and makes the pipelined operation of the algorithm difficult. It can easily be verified that $S_{i+1} < 2M$ always true if A, B < M. However, S_m can not be reused as input A or B for the next modular multiplication because S_m is overflow. Example: if A = 243, B = 252, M = 253 then $S_m = 296$ (more than 8-bit). If we perform two more executions of the loop with $a_{m+1} = 0$ and inputs A, B < 2M, the inequality $S_{m+2} < 2M$ is still satisfied. Now, S_{m+2} can be used as input A or B for the next modular multiplication. As the result, we just allow S to have two more bits for intermediate results. Figure 2 show the optimized Montgomery Modular Multiplication Algorithm (radix 2) for computing $A.B.R^{-1} \bmod M$ with (*m+2*)-bit operands X $(0,0,x_{m,}x_{m-1},......,x_1,x_0)$, Y and M.

$$A = \sum_{i=0}^{m+1} 2^i a_i, a_i \in \{0,1\}, a_{m+1} = 0;$$

$$B = \sum_{i=0}^{m} 2^i b_i, b_i \in \{0,1\}$$

$$M = \sum_{i=0}^{m-1} 2^i m_i, m_i \in \{0,1\}$$

$$A, B < 2M; M < R = 2^{m+2}; GCD(M,R) = 1$$

```
1.  S = 0; C = 0;
2.  for i = 0 to m+1
3.    $q_i$ = ($S_i$(0) + $C_i$(0) + $A_i$B(0)) mod 2
4.    if($A_i$ = 0)
5.          if(q = 0)
6.              I = 0;
7.          else
8.              I = M;
9.          end
10. else
11.      if(q = 0)
12.          I = B;
13.      else
14.          I = B + M;
15.      end
16. end
17. $C_{i+1}$ + $S_{i+1}$ = $C_i$ + $S_i$ +I;
18. $C_{i+1}$ = ($C_{i+1}$)/2;
19. $S_{i+1}$ = ($S_{i+1}$)/2;
20. end
21. return P = $S_{m+2}$ + $C_{m+2}$;
```

Fig. 2: The Optimized Montgomery modular multiplication $A.B.R^{-1} \bmod M$

The difference of Montgomery modular multiplication and the other modular multiplication is multiplication is done in Montgomery domain. Montgomery domain uses the R^{-1} values for calculating modular multiplication, $MM(A,B) = A.B.R^{-1} \bmod M$. Thus for modular exponentiation, we have to transform the input value into the Montgomery domain, and then we calculate modular multiplication. Next, the output is transformed into normal representation. For calculating modular exponentiation we use following algorithm. Figure 3 show modular exponentiation Algorithm based on Montgomery modular exponentiation for computing $X^E \bmod M$ [2].

```
$E = \sum_{i=0}^{t-1} 2^i b_i ; b_i \in \{0,1\} ; E_{(t-1)} = 1;$
$M = \sum_{i=0}^{m-1} 2^i m_i ; m_i \in \{0,1\};$
$Integer\ X,\ 1 \le X < M;$
$M < R = 2^{m+2}; GCD(M,R) = 1;$
A = R mod M;
G = $R^2$ mod M;
$\overline{X}$ = Mont(X,G);
for i = t to 0
    A = Mont(A,A);
    if($E_i$ = 1)
        A = Mont(A, $\overline{X}$)
    End if
End for
A = Mont(A,1);
Return A;
```

Fig. 3: Montgomery Modular Exponentiation

III. ARCHITECTURE DESIGN

Proposed architecture is based on pipeline architecture for Montgomery Modular Multiplication. The architecture uses repeated Montgomery Modular Multiplication to compute modular exponentiation. For implementation the optimized Montgomery modular multiplication as shown in Figure 5, our architecture consists of

1st Adder: This adder has function to determine register I = Y + N as shown in Figure 2 step 14. N is modular number and Y is register input for Montgomery multiplication.

Register I: Based on Matlab source, register I has four possible values that are 0, Y+N, Y and N. N is module number, Y is register input for Montgomery modular multiplication. This architecture uses Sout and Cout from CSA together with register input to select value of register I. This register I has 10 bit word length.

Carry Save Adder (CSA): CSA has function to calculate the value of sum and carry from three inputs. Each register input has 10 bit word length. The Logic gate of our CSA can be described in figure 4.

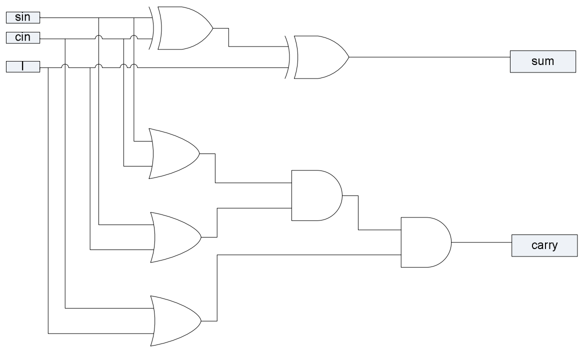

Fig. 4: Logic Gate of Carry Save Adder

Output from CSA has value out = Sout + 2Cout. Thus, division by 2 is done by insert present register (Sout) to the bit n-1 position of the next PE and (Cout) is connected with the same bit position PE for next iteration while register. This operation is shown in block diagram figure 4.

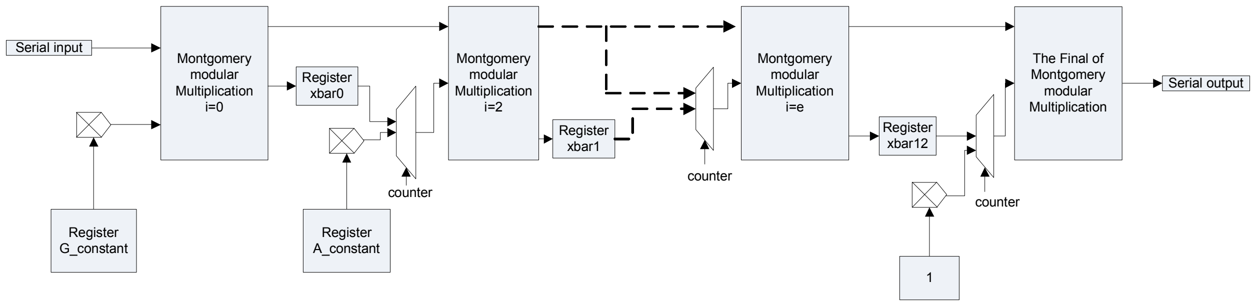
Serial input
Montgomery modular Multiplication i=0
Register xbar0
Register G_constant
Register A_constant
counter
Montgomery modular Multiplication i=2
Register xbar1
counter
Montgomery modular Multiplication i=e
Register xbar12
counter
1
The Final of Montgomery modular Multiplication
Serial output

Fig. 5: RSA Architecture

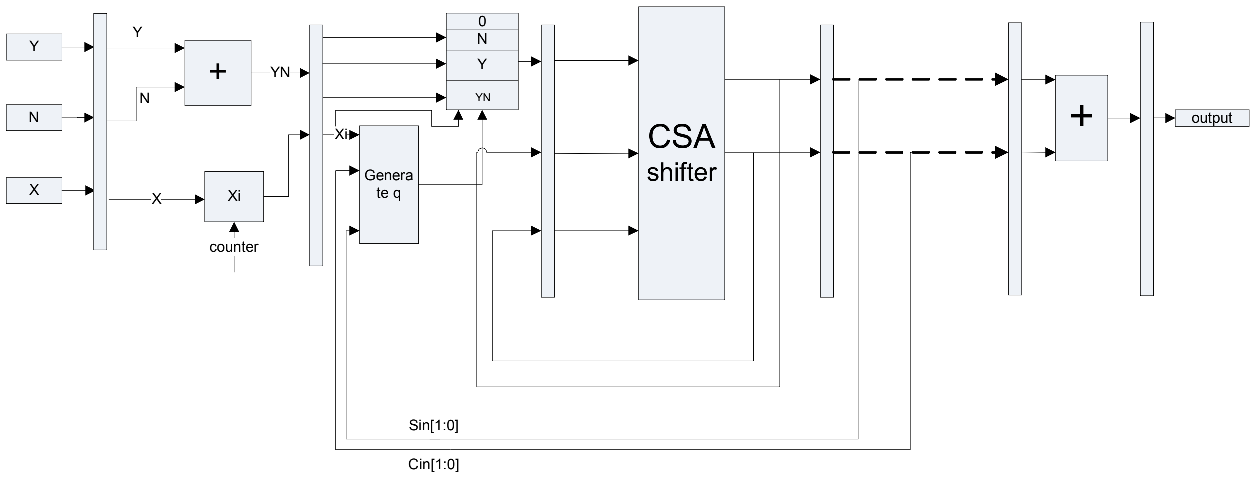

Fig. 6: Montgomery modular multiplication architecture

2nd Adder: this part has function to add the output of register sum and carry from CSA unit. The result will be used as input for the next modular multiplication.

Controller: The architecture consists of an CSA, generator q, register I, first Adder, and second adder. Every stages of the circuit uses counter as the controller. The counter is activated using the rst signal. Controller has function to determine when the data is processed. This counter has 8 bit word length. In order to get better performance, we make the architecture for Montgomery modular multiplication on pipeline form where we insert register D Flip-flop between them.

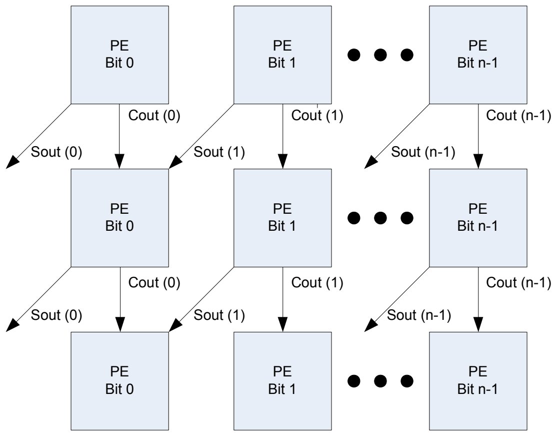

Fig. 7: Processing Element of CSA

IV. VERIFICATION

In order to verify the algorithm, we use Matlab simulation by using key (E = 101, D = 61, M = 253). The result is as follows:

The Original message

```
The  fist  step  in  implementing  this
algorithm  is  simulation  using  high  level
language.
The  purpose  of  this  process  is  to  verify
the true functionality of the algorithm
and  to  provide  data  test  for  simulation
using verilog HDL.
```

Fig. 8: input file : *message.txt*

Matlab Output after encryption: `message.txt`

```
3ì#ÛõxstÛst#ôÛxÑÛx
ô #
#ÑtxÑàÛtìxsÛû àzrxtì
ÛxsÛsx
I ûtxzÑÛIsxÑàÛìxàìÛ #ï# Û ûÑàIûà#.Û{™3ì#Ûô
IrôzS#ÛzõÛtìxsÛôrzB#ssÛxsÛtzÛï#rxõÜÛtì#ÛtrI
#ÛõIÑBtxzÑû xtÜÛzõÛtì#Ûû àzrxtì
Û{™ûÑ…ÛtzÛôrzïx…#Û…ûtûÛt#stÛõzrÛsx
I ûtxzÑÛIsxÑàÛï#rx zàÛÁD+.{™
```

Fig. 9: Matlab Output for Encryption file: *chipper.txt*

Matlab Output after decryption: `chipper.txt`

```
The fist step in implementing this algorithm
is simulation using high level language.
The purpose of this process is to verify the
true functionality of the algorithm and to
provide  data  test  for  simulation  using
verilog HDL.
```

Fig. 10: Matlab Output for Decryption, file **: *decrypt.txt***

From above we can observe that the output is correct. The architecture of RSA was first modeled in Verilog and functionally verified using RTL simulator. The outputs from the ModelSim are validated against a standard modular exponential in Matlab function. Figure 11 show the simulation waveform produced by RTL simulator. Figure 11 shows the simulation waveform produced by ModelSim simulator. It can be observed that the latency is 179 clock cycles. The design is also synthesized using Altera Quartus II version 6.0. From the synthesis result, our RSA circuit can achieve frequency clock is 176.88 MHz.

V. Conclusion

RSA architecture based on Montgomery algorithm and pipeline architecture has been presented. We use pipeline architecture in modular multiplication, as the result we can get 14 data each 179 clock cycles. The architecture has been modeled with hardware description language Verilog. First data output appear in 179 clock cycles (for CE = 101) with maximum allowable frequency is 261.85 MHz on Altera Cyclone II EP2C35F672C6. The validity of the proposed design architecture has been verified.

VI. References

[1] Blum, Thomas. 1999. "*Modular Exponen-tiation on Reconfigurable Hardware*". Master Thesis. Worcester Polytechnic Institute.

[2] A.J. Menezes, P.C. Van Oorschot, and S.A. Vanstone.1997. *"Handbook of Applied Cryptography*". CRC Press.

[3] A.P. Fournaris and O. Koufopavlou. 2005. *"A New RSA Encryption Architecture and Hardware Implementation based on Optimized Montgomery Multiplication"*. IEEE.

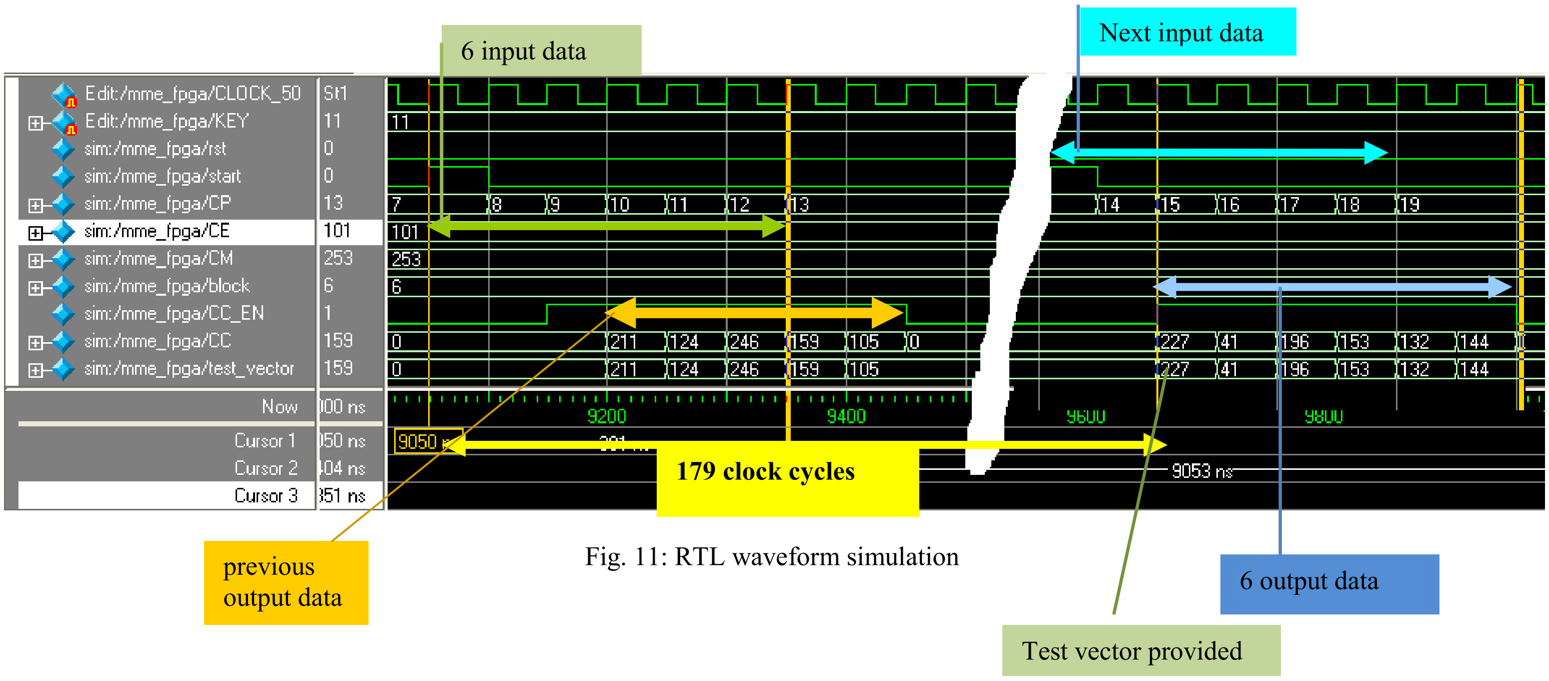

Fig. 11: RTL waveform simulation