Asynchronous vs. Synchronous Design of RSA

A. Rezaeinia, V. Fatemi, H. Pedram, B. Sadeghian, M. Naderi
Computer Engineering Department, Amirkabir University of Technology, Tehran, Iran
{rezainia,fatemi,pedram,naderi}@ce.aut.ac.ir

Abstract

Asynchronous designs have the potential to be robust with respect to changes in the physical environment. They also appear to consume less power and provide better performance compared to their synchronous counterparts. In this work, synchronous and asynchronous implementations of the modular multiplication that is the major part of the RSA algorithm is presented at the transistor level. Total currents and transistor counts are reported and compared.

1. Introduction

Difficulties of synchronous circuits such as clock skew, power consumption, worst-case delay, and physical sensitivity pave way for asynchronous designs. There are different asynchronous delay models; among them Delay Insensitive (DI) is one of the most popular models. A circuit is called DI when its correct operation is independent of the delays in the operators and wires, expect that these delays are positive and finite. Choosing DI model is more appropriate when physical conditions like voltage and temperature may vary.

Recently, smart cards are being used in growing number of applications such as banking, telephony access control, health care, and tickets. These cards are good examples for variable physical conditions, especially when they are used in contactless mode.

Designing smart cards through synchronous methods requires observing the worst cases. While asynchronous circuits operate over a wide range of physical conditions and automatically adapt their speed. Also asynchronous circuits have two power properties that make them suitable for such cards: low average power and small current peak.

Data encryptions based on asymmetric key exchange are more popular than the symmetric ones. A public key exists that everyone can use for encryption, but decryption is only possible for those who know the secret key. RSA is one of the most famous asymmetric methods, and was introduced in 1978. RSA is based on repeated modular multiplication that we used Montgomery algorithm to implement it. RSA implementation is presented here at the switch level for both synchronous and asynchronous modes.

This paper first reviews the RSA security method. Montgomery and array based multiplication is discussed in part 2. Design block diagram and a detailed explanation of units is brought in part 3. Simulations are covered in part 4. Conclusions follow in part 5.

2. RSA and Montgomery Algorithms

First we describe discrete logarithm and Fermat's theorem as they are referred in next parts.

Discrete logarithm is defined as follows: imagine order pair (g,s) and prime number N. Is there any positive number such as x, $0 \le x < N$ where:

$x = \log_g s \pmod N$

Femat's theorem is defined as follows: Uler introduced a function named $\varphi(n)$. The function returns the number of positive numbers that are less than n and prime to it. For example $\varphi(6) = 2$ as only 1 and 5 are prime to 6 and less than it.

It is obvious that we can extend the rule to prime number as where $\varphi(p) = p - 1$ if p is prime. Then femat's theorem is stated as:

$a^{p-1} = 1 \pmod N$ if p is prime.

2.1. RSA algorithm

RSA was introduced by Adlman, Shamir, Rivest and is based on two hard math problems, discrete logarithm and factorization.

In RSA message and keys (Public and Secret) belong to $Z_n = \{1,2,\ldots,n\}$. N is $p*q$ where p and q are both prime numbers. Encryption produces C from public key, K_B, and message M.

$C = E_{K_B}(M) = E_B(M) = M^{K_B} \pmod N$

Decryption uses secret key k_B and encrypted text C to obtain M.

$M = D_{k_B}(C) = D_B(C) = C^{k_B} \pmod N$

Surely, the encrypted message must be decrypted uniquely:

$$M = D_B(E_B(M))$$

$K_B k_B = 1 (\operatorname{Mod} \gamma(n))$ where $\gamma(n)$ is least common multiply of p-1 and q-1.

For Example: B is going to send message C. B assumes p=5 and q=7 then $\gamma(n) = lcm(4,6) = 12$. Now if we consider $K_B = 17$, secret key k_B will be

$$17k_B = 1(\text{Mod}\,12) \Rightarrow \text{k}_\text{B} = 5.$$

Ordered pair $(N = 35, K_B = 17)$ is delivered to the other side.

Now if sender want to transmit M=33.

$$C = M^{K_B}\ (\text{Mod N}) \Rightarrow \text{C} = 33^{17} = 3\,(\text{Mod}\,35).$$

Receiver can obtain M using its secret key:

$$M = C^{k_B}\ (\text{Mod N}) \Rightarrow \text{M} = 3^5 = 33\,(\text{Mod}\,35)$$

2.2. Montgomery Algorithm

As it was mentioned above RSA needs repeated multiplication. Montgomery has recommended a simple method which is suitable for both hardware and software implementations. Here is a brief description of the Montgomery algorithm.

N is a positive number (module of operation) and R is a prime number to N. Define :

$$\text{Mont\_Product}\,(\text{A},\text{B},\text{N},\text{R}) = \text{ABR}^{-1}\ (\text{mod N})$$ In fact Montgomery introduces an algorithm for modular reduction.

Assume T, module N, and positive number R which is prime to N,

$$RR^{-1} - NN' = 1$$

Must be true where $RR^{-1} = 1\,(\text{mod N})$ and $NN' = -1\,(\text{mod N})$.

In this case for 0<T<RN modular reduction can be obtained by function REDC.

$$Function REDC(T)$$

$$\text{m} = (\text{T mod R}).\text{N}'\ \text{Mod R}$$

$$\text{T} = (\text{T} + \text{m} * \text{N})/\text{R}$$

Montgomery algorithm results TR^{-1} mod N instead of T mod N. Also in cases where $T>N$, main results are obtained by $T=T-N$.

2.3. Array based modular multiplication

Walter [1] represented a systolic array multiplier using Montgomery algorithm.

A is a number so that $A = \sum_{i=0}^{N-1} A[i]r^i$. Radix r is selected a power of 2 for hardware simplicity. B and M are numbers satisfying B<2M is. Walter's algorithm is represented as ;

$$P = 0;$$

$$for\ \text{i} = 0\ \text{to N - 1 do}$$

$$\text{Q[i]} = ((\text{P[0]} + \text{A[i]} * \text{B[0]}) * (\text{r - M[0]})^{-1}\ \text{mod r}$$

$$\text{P} = (\text{P} + \text{A[i]} * \text{B} + \text{Q[i]} * \text{M})\,\text{Div r}$$

Walter's array is shown in Figure 1. It consists of similar cells except for column one which has no carry and Q inputs. These cells produce Q and feed through the rows.

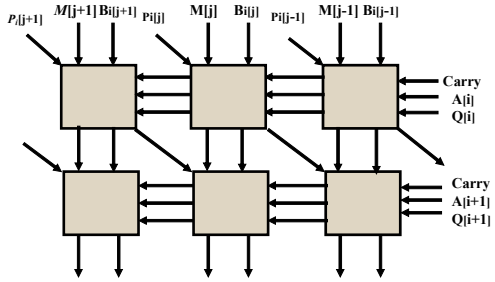

Figure 1. Walter array.

The following example clarifies the array multiplier.

A=5, *B*=3, *M*=13

Then

*A*B* (mod *N*) = 5*3 (mod 13) = 2

Now, Montgomery and Walter's algorithms suggest:

$$A = 5 \Rightarrow (0101)$$

$$\text{B} = 3 \Rightarrow (0011)$$

$$\text{M} = 13 \Rightarrow (1101)$$

$$\Downarrow$$

$$B[0] = 1$$

$$N = 4$$

N=0

$$Q_0 = 0 + 1 * 1 = 1$$

$$P = 0 + 1 * 3 + 1 * 13 = 16 \Rightarrow 8$$

N=1

$$Q_1 = 0 + 0 * 1 = 0$$

$$P = 8 + 0 * 1 + 0 * 13 = 8 \Rightarrow 4$$

N=2

$$Q_2 = 0 + 1 * 1 = 1$$

$$P = 4 + 1 * 3 + 1 * 13 = 20 \Rightarrow 10$$

N=3

$$Q_3 = 0 + 0 * 0 = 0$$
$$P = (0 + 0 * 0 + 0 * 13) = 10 \Rightarrow 5$$

3. Design Block Diagram:

Power in RSA equation is usually large, so it is not possible to arrange multiplier blocks in a pipeline to result those large values. Therefore, data is fed back to the multiplier for further operations. Figure 2 shows the RSA block diagram. Data selection units direct data to/from outputs and inputs or cycle it through the multiplier. Multiplication unit is implemented using Motgomery and Walter algorithms as mentioned earlier.

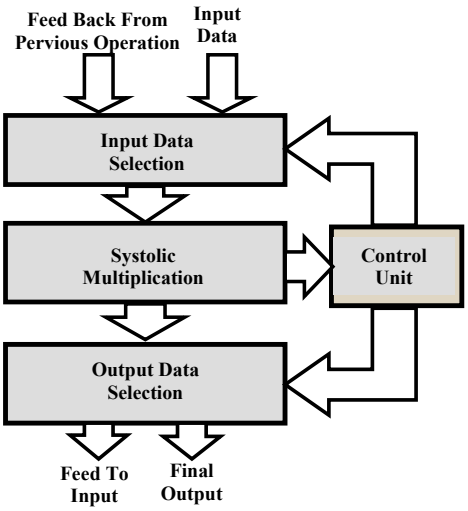

Figure 2. RSA Block Diagram.

Data selection units do not have the decision capability, and they receive information related to the data direction from the control unit. To produce this information, control unit requires counters and comparators. A detailed description follows.

3.1. Operation Unit

As shown in Figure 2, circuit consists of three major units:

- Input / Output.
- Modular multiplier.
- Control unit.

3.1.1. Input/Output

Data is passed to next cells in each cycle, thus input data must be fed to array with one cycle delay with respect to previous cell. Figure 3 illustrates the idea. In order to reach such an input pattern in synchronous design some registers are needed in input and output sections. Obviously as the array size increases, the number of registers increases in quadratic order.

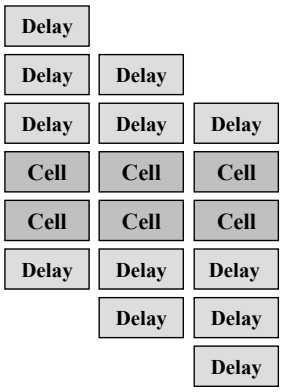

Figure 3. Input Data Feeding.

In asynchronous design each cell pends for its inputs and the cell does not start operation until all inputs get valid data. This mechanism eliminates the need for registers. The numbers of registers are derived from the following equations:

$\frac{n(n-1)}{2}$ Horizontal Inputs

$\frac{n(n-1)}{2}$ Vertical Inputs

$\frac{n(n-1)}{2}$ Outputs

Then $\frac{3n(n-1)}{2}$ registers are totally required.

3.1.2. Multiplier Unit

The multiplier unit is implemented using array-based Walter model. In synchronous design a global clock controls the data movement between cells, while in asynchronous design a data flow control exists. Array is named systolic in synchronous mode, and wavefront in asynchronous case. Again a number of registers are needed to force delayed data in the sysnchronous model. According to Walter's algorithm, results must be divided by r which can be implemented by shift operation if r is a power of 2. Because of the shift operation, data reaches the next cell after two clocks cycle instead of one. Thus forwarded data in column must be delay by one cycle, which is acieved by using registers. These registers are not required in asynchronous mode as each cell waits until all inputs are valid.

3.1.3. Control Unit

In asynchronous design two counters and two comparators are needed. One set is used to determine if the multiplication is complete and the othe set counts the number of multiplications and compares it to power of RSA in order to signal the end of operation.

Counter implementation is not a simple task in asynchronous mode, so we designed the control unit differently. In asynchronous design, a block can detect validation of data and also completion of operation. Using these properties eliminates a counter and a comparator. Instead of counting the number of cycles to detect the completion of multiplication, data reaching the last cell in array signals operation is completed.

As data is pipelined through the array a mechanism is needed to indicate multiplication is completed. This is done by introducing a flag. Flag is set when the first data is pipelined to the array and it will clear in the following cycle. The flag remains cleared until a new cycle of multiplication.

Setting and clearing the flag is done by the control unit. When data reaches the last cell with its flag set a counter increases to keep number of multiplication and if its flag is not set no counting occurs.

Control unit uses first data flag and counter value in order to direct data through the array or input/output. Figure 4 shows block diagram of asynchronous RSA.

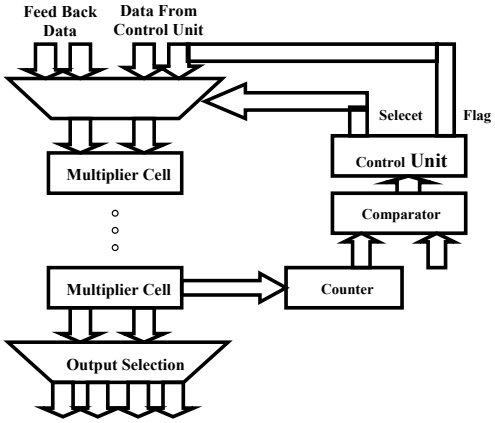

Figure 4. Asynchronous RSA.

4. Simulation

Synchronous version of the multiplication process in our synchronous design always needs 3n-2 cycle. This is called the worst-case delay. The synchronous multiplier must go through 3n-2 cycles regardless of data type. But in asynchronous circuit, delay is related to the average case rather than the worst case. A simulation program was developed to investigate the number of cycles for different data inputs. The program has been run for 5*5 and 8*8 arrays and the following results were obtained.

A 5*5 array needs 13 cycles to complete, where an asynchronous 5*5 array may finish in 8 cycles for particular input data. If we extend the array size to 8, synchronous design needs 22 cycles, where the asynchronous system may complete in 13 cycles in some cases. As the array size increases, the difference between the two methods becomes more evident.

4.1. VHDL Simulation

To test and simulate the recommended asynchronous design, it was modeled by VHDL. The code consists of different blocks that are active only when input data is valid. Handshaking among blocks is by En and Ack signals. En signals the previous block when data is received and ack is the signal indicating that the next block has consumed data. A simple VHDL block code follows. Other blocks follow similar structures.

```
Process(Reset,Inh,Inl,En,Outh,Outl, ack)
Reset : IN STD_LOGIC;
Inh     : IN STD_LOGIC;
Inl     : IN STD_LOGIC;
Ack     : IN STD_LOGIC;
Outh   : OUT STD_LOGIC:
Outl    : OUT STD_LOGIC:
En      : OUT STD_LOGIC:

Begin
  If Reset='1' then
    Outh<='0';
    Outl<='0';
     En<='1';
  End if;
  If Reset='0' then
    If Outh/='0' or Outl/='0' then
      OutV:='1';
     End if;
     If Outv='1' and En='1' and
          Ack='1' then
        // Logical Block
        En<='0';
        Nut<='0';
      End if;
      If En='0' and Ack='0' then
        Outh<='0';
        Outl<='0';
        Nur<='1';
      End if;
      If Outh='0' and Outl='0' then
        OutV:='0';
      End if;
 If OutV='0' and En='0' and Nut='1'
    Then
  En<='1';
 End if;
End if;
End process;
```

The Reset signal causes outputs to turn zero and En set to one. If input data validate goes active and En is zero, the previous block signals that data is received. When Ack goes zero, it indicate the data has been consumed by the next cell and then outputs turn nutral. In the last phase if input data become nutral En turns one. At this point four phases are complete and cell is ready for next operation.

4.2. SPICE Simulation

In this part the production rules of an asynchronous design are mapped to switches. Asynchronous blocks consist of two parts: data and control. Data are coded in dual rail. En and Ack are used for handshake.

Other than the four-phase action that was described earlier, there are situations that all inputs are not used or all outputs are not validated. In these cases only the data lines that participate in an operation must receive or send acknowledgement, which makes the design complex. Also some blocks may start operation even though acknowledgements are zero. Input and output selectors are examples of such cases. The acknowledgement must only be received or transmitted on the data lines that have participated in the operation. A simple asynchronous block is shown in Figure 5.

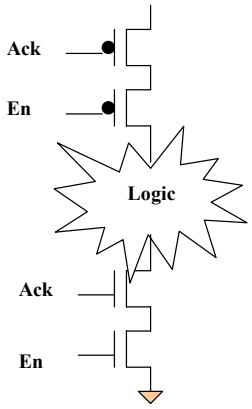

Figure 5. Simple Asynchronous Block.

Operation starts when both Ack and En are active and input data are valid. After the logical blocks compute results, the control unit disables En. This causes the appropriate transistor to become active in pull up circuit and ready for nutralization. As soon as Ack goes zero, pull up becomes active and outputs go nutral. The control unit waits until both input and output are nutral and then activates En again. Everything is now ready for the next operation.

5. Conclusion

The asynchronous and synchronous implementations of the modular multiplication as the major part of the RSA algorithm were presented at the transistor level. Simulation results in SPICE were illustrated in charts.

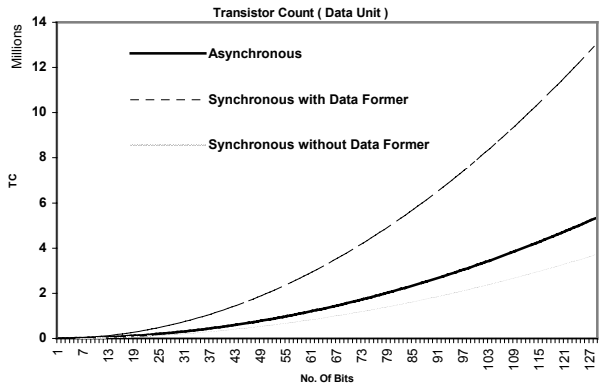

Figure 6. Transistor Count Comparison.

Also registers and delay element that are used to form data and control data flow cause a considerable overhead. Figure 6 shows that simple blocks of multiplier array are less in synchronous case than asynchronous design. That is because asynchronous circuit goes through level of activation and neutralization, which cause increase in the number of transistor. But in order to systolic work right data must be inserted in a special format. In asynchronous case, no former register is required as the data flow is control by handshake mechanism of circuit, while in the synchronous case forming is performed by a number of registers, which increase the number of transitors. As Figure 6 states number of transistors for synchronous circuit with data former is higher than asynchronous circuit.

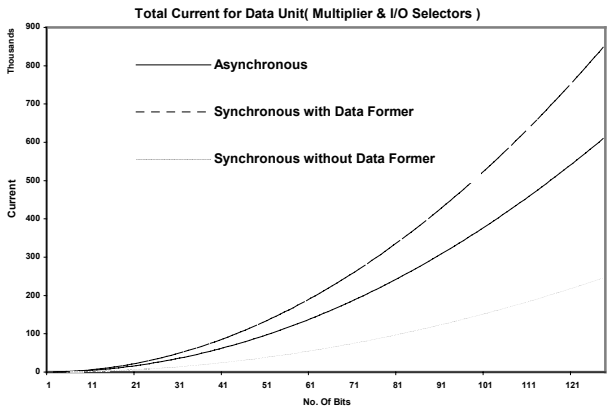

Figure 7. Total Current of Data Unit.

Current consuming graphs for three cases above mentioned are represented in Figure 7.

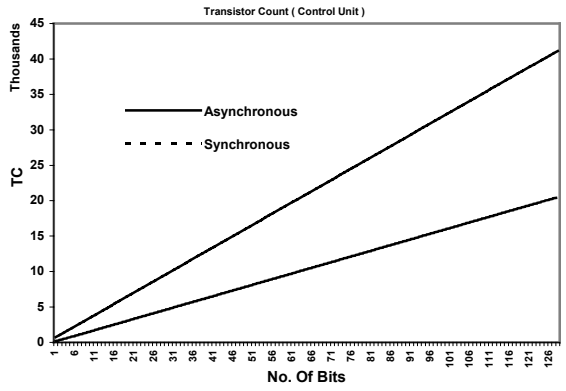

Figure 8. Transistor Count of Control Unit.

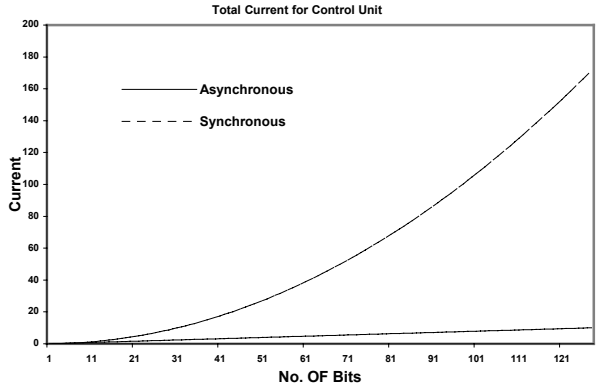

Figure 9. Total Current of Control Unit.

Figure 8 and Figure 9 indicate although the numbers of transistors are more in asynchronous circuit than synchronous one (because of handshake implementation and valid-neutralization of asynchronous circuit), the total current consuming is less in asynchronous case. That is because the control part of the asynchronous circuit is only operates when there is a valid data for processing, but control part of the synchronous circuit is always operational and receive clocks.

It was concluded that although the transistor count in the asynchronous design is more than that of the synchronous one, the total current consumed in asynchronous mode is less. This result is due to the fact that in synchronous mode the control unit is always functioning, while in asynchronous mode it functions only when needed.

Reference

[1] Colin. D. Walter, "Systolic Modular multiplication", *IEEE TRANSACTION ON COMPUTERS*, Vol 42, No.3, Pages 376-378, March 1993.

[2] Andrew Matthew Lines, "Pipelined Asynchronous Circuits", *IEEE Computer Society*, June 1998.

[3] Joep Kessels, Torsten Kramer, "Applying Asynchronous Circuits in Cantactless Smart Cards ", *IEEE TRANSACTION ON COMPUTERS*, Pages 36-44, 2000.

[4] Cetin Kaya Koc, "RSA Hardware Implementation", RSA Data Security Inc., August 1995.

[5] Po-Song, Chen,Shih, Arn Hwang, Cheng Wen Wu, "A Systolic RSA Public Key Cryptosystem", *IEEE TRANSACTION*, Pages 408-411, 1996.

[6] Scott Hauck, "Asynchronous Design Methodologies", *Proceedings Of the IEEE*, Vol 83, No. 1, Pages 69-92, January 1995.

[7] Alain J. Martin, "Synthesis of Asynchronous VLSI Circuits", *Technical Report*, Department of Computer Science California Institute of Technology, August 1991.