FPGA Implementation of RSA Public-Key Cryptographic Coprocessor for Restricted System

Mooseop Kim Yongje Choi and Howon Kim
Electronics and Telecommunications Research Institute
161 Gajeong-Dong, Yuseong-Gu, Daejeon, 305-350, KOREA
e-mail : {gomskim, yjchoi , hwk}@etri.re.kr

Abstract : In this paper, the hardware implementation of the RSA public-key cryptographic algorithm is presented. The RSA cryptographic algorithm is depends on the computation of repeated modular exponentials. The Montgomery algorithm is used and modified to reduce hardware resources and to achieve reasonable operating speed for smart card. An efficient architecture for modular multiplications based on the array multiplier is proposed. We have implemented a 1024bit RSA cryptographic processor based on proposed scheme in IESA system developed for smart card emulating system. As a result, it is shown that proposed architecture contributes to small area and reasonable speed for smart cards.

1. Introduction

As the use of computer and Internet service becomes a more pervasive part of our daily life, the concern with security problems has become growing for recent years.

The same size as a credit card, a smart card stores and processes information through its embedded system in the card chip. From these higher security features than other type cards, smart card is used in many applications such as a telecommunicaton, and an access control, etc.

Public key cryptographic systems such as RSA are very important to smart cards for an authentication, a private key exchange, and a digital signature. The RSA is adopted in Visa, Mastercard, and EMV smart card systems to guarantee its security for credit card transactions such as Secure Electronic Transactions(SET) protocol.

Modular exponentiation of long integers-often 512 or 1024-bits long-is the critical operation for a variety of the most widely accepted cryptosystems. The operation for RSA encryption and decryption are $c = x^e \bmod m$. where x, e, m, and c stand for the message to be encrypted or decrypted, the encryption(decryption) key, the modular and the encrypted(decrypted) message respectively. For security concern, the length of x, e and m are 512~2048 bits, which makes RSA operations intensive. However, the large bit over 1024-bit modular operation makes the RSA system difficult to implement in hardware.

The common ingredients of RSA processor such as long registers, shift registers and adders are expensive in hardware area and power consumption.

Over the past few decades, a considerable number of studies have been conducted on the efficient hardware design of RSA processor[2, 6]. But, many of those studies focused on the decrease of operating time for faster system performance. There are many factors to be considered for a cryptographic hardware design such as cost, area and performance etc. The main issues to design of a cryptographic circuit for smart card system are trade-offs between a feasible speed and a small chip area.

We have studied the method of reducing the chip size for practical hardware implementation and that of achieving reasonable operating time for smart card's data transaction. The proposed arhitecture is implemeted in Xilinx Vertex series FPGA in IESA system developed for smart card emulating.

In section 2, we analyzed the Montgomery algorithm and modified it to make the hardware arcitecture simple. We proposed an efficient hardware architecture for modified algorithm in section 3. An implementation of RSA processor above IESA system is described in section 4. Finally, we analyze the performance of the 1024-bit RSA processor implemented by proposed architecture.

2. Modified Algorithm

There are many algorithms to compute modular reduction of AB to $AB \bmod M$. The naive method to compute modular multiplication is to perform a multiplication of two numbers and then divide with M. This method is inadequate for smart card, because it needs large memory area to save partial products.

Algorithm 1. Montgomery modular multiplication

Inputs : $m = (m_{n-1}\ldots m_1 m_0)_b$, $x = (x_{n-1}\ldots x_1 x_0)_b$,
$y = (y_{n-1}\ldots y_1 y_0)_b$, $x \geq 0$, $m > y$, $R = b^n$,
$gcd(m, b) = 1$, $m' = -m^{-1} \bmod b$

Step 1 : $A = 0$

Step 2 : *for* $i = 0$ *to* $n-1$
$u_i = (a_0 + x_i y_0) m' \bmod b$
$A = (A + x_i y + u_i m)/b$
end loop

Step 3 : *if* $A > m$ *then return* $A-m$
else return A

Output : $xyR^{-1} \bmod m$

The P. L. Montgomery[1] suggested as follows ; instead of computing the complete product of two operand at once,

it might be more efficient using interleaving the addition of a_iB with modular reduction. By doing so, this method saves the partial products without overflow.

The Montgomery's modular multiplication algorithm, given in Algorithm 1, is one of the most widely used to increase the performance of modular arithmetic.

Where x, y and m are n-bit numbers. Generally, this algorithm is used in radix-2 because of its simple hardware implementation. But, this algorithm still has a critic problem to be used in a smart card system.

The algorithm 1 needs n iterations in each modular multiplication and requires two additions per iteration. These additions over a long integer cause the use of a large bit registers. So it is necessary to modify Montgomery's algorithm for an efficient smart card system.

2.1. Modified Modular Multiplication

The long bit data addition in each iteration is an expensive in terms of hardware area and power consumption. As an alternative approach, we revised algorithm 1 such that only 32-bit data multiplication and addition are required in each iteration for more efficient design of smart card.

A state diagram for the modified algorithm using this idea is shown in Figure 1.

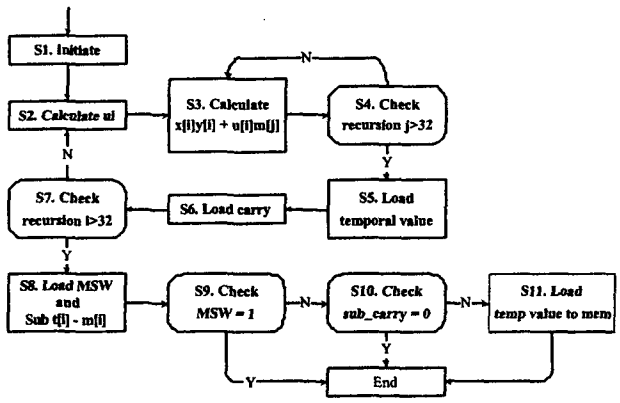

Figure 1. The state diagram for modified multiplication

Instead using a long bit data arithmetic, all arithmetic operations are reduced to 32-bit multiplications, additions, subtractions and binary shifts.

The modified algorithm's core operating routine, the modification of step 2 in the algorithm 1, is shown in algorithm 2. In this algorithm, only 32-bit data multiplications and addtions are required in each iteration. The number of iteration, however, is increased up to 32 times maximum that of original algorithm. But the overall computation time of the iteration is reduced to a reasonale time by using faster clock and using an efficient controller circuit.

Algorithm 2. Modified modular multiplication

for $i = 0$ *to* $n-1$ *loop*
  $(C_1, S) = x_i y_0 + a_0$
  $(Rt_1, R_u) = m'S$
  $(C_2, S) = m_0 R_u + S$
  for $j = 1$ *to* $n-1$ *loop*
    $(C_1, S) = a_i y_j + a_j + C_1$
    $(C_2, S) = m_j R_u + S + C_2$
    $a_{j-1} = S$
  end loop j
  $(R_p, S) = C_1 + C_2$
  $(Rt_1, S) = a_n + S$
  $a_{n-1} = S$
  $(Rt_2, R_p) = a_{n+1} + R_p$
  $(Rt_1, R_p) = Rt_1 + R_p$
  $a_n = R_p$
end loop i

The advantage of this algorithm is that all the operations are reduced in 32-bit data range, hence, it leads to simpler architecture and, thus, smaller chip area.

2.2. Modified Modular Exponentiation

In RSA cryptographic algorithm a message X is encoded by computing X^e *mod* M for some M and encryption key e. Modular exponentiation is accomplished by repeated modular multiplications. But a RSA cryptographic circuit based on Montgomery modular multiplication needs two extra operations to perform a modular exponentiation.

According to Montgomery's theorem[1, 2], input data X is converted to montgomery residue class type Xr *mod* M. From this converting, the output of modular exponentiation becomes $X^e r$ *mod* M Another modular multiplication of Mont($X^e r$ *mod* M, 1) perfo rmed to remove the extra factor r for correct output. Finally the last output result is the desired form X^e *mod* M.

Algorithm 3 : Modified Exponentiation

Inputs : $m = (m_{n-1}...m_1 m_0)_r$, $R = r^l$, $m' = -m^{-1}$ *mod* r,
  $e = (e_t...e_1 e_0)_2$ *with* $e_t = 1$ *and a message* x,
  $1 \le x \le m$.
Step 1 : $xp = xR$ *mod* m, $A = R$ *mod* m.
Step 2 : *for* $i = t$ *down to* 0
    $A = Mont(A, A)$.
    if $e_i = 1$ *then* $A = Mont(A, xp)$.
    else $A = A$.
  end loop
Step 3 : $A = Mont(A, 1)$.
Step 4 : *return* A.
Output : x^e *mod* m

3. Modular Multiplier Architecture

The designed modular multiplier performs the modular reduction operation by computing Montgomery modular multiplication. In hardware implementation, every step in Montgomery algorithm is designed by 32-bit multiplication, binary shift and addition.

The critical issue in design of modular multiplier is the design of 32×32 multiplier. A 32×32 bit multiplier is too large for smart card system. Furthermore the delay may be important and reduce the speed of whole RSA processor. So we used repeated small multiplications to compute whole 32-bit multiplication as most microprocessor manufactures using in their processor.

To carry out 32×32 bit multiplication, we used 32-bit by 8-bit array multiplier as shown in Figure 2[3, 4]. Although this repeated multiplication takes 4 clock cycles for 32-bit multiplication, we could use the additive array multiplier because it performs 32-bit multiplication and addition at the same time.

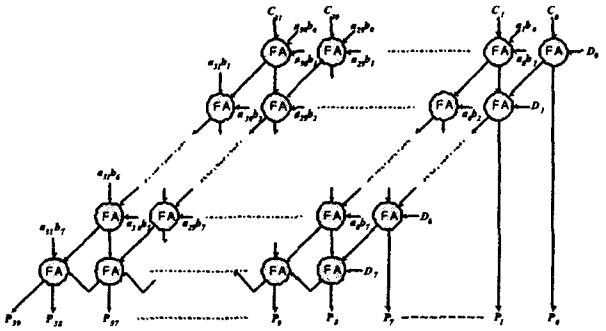

Figure 2. The architecture of 32×8 bit array multiplier

This architecture reduces the complexity of hardware and achieves small chip area which proper for smart card. This simple architecture needs many clock cycles to perform multiplication steps.

The architecture of additive multiplier, a core operation module of Montgomery modular multiplication is shown in Figure 3.

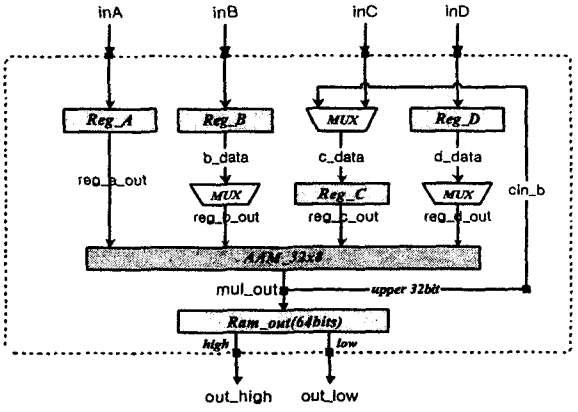

Figure 3. The architecture of 32-bit additive multiplier

For its reasonable performance, special care is needed to design an efficient control circuit. The additive multiplier in Figure 3 makes it possible to design simple modular multiplier. Actually, the designed modular multiplier is consists of only additive multiplier, some 32-bit registers and control block.

4.Imple mentation of RSA processor

Modular exponentiation is executed in three steps; data converting to a Montgomery residue class, exponentiation and reconverting to original data class.

Figure 4 shows the architecture of the 1024-bit RSA porcessor adopting the modified algorithm. The designed RSA processor consists of interface, control register, controller, several 32bit data registers, modular multiplier, memory and output register.

First, each input data for RSA cryptograhic operation is transferred through interface circuit and registered in memory block by control circuit. During repeated multiplication steps, the intermediate data should be stored to memory in 1 clock cycle. But, the writing data to memory and the reading data from memory in every iteration steps cause many data transfer delay, hence, it reduce the whole system performance. To settle this difficulty and to increase operating speed, we use an additional shift register to store intermediate data. After reconverting step, the desired output data is stored in memory and the controller sets the end operation bit in control register.

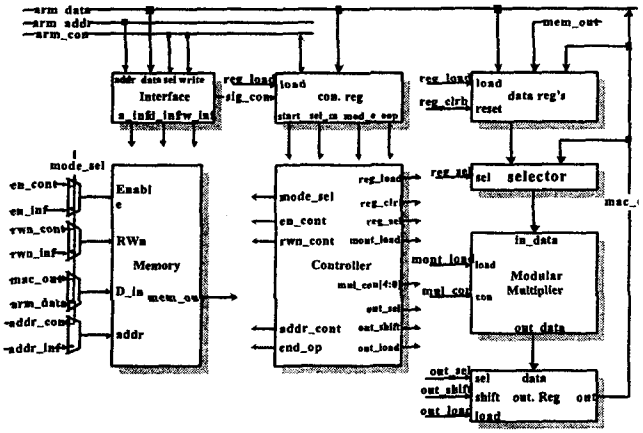

Figure 4. The architecture of RSA processor

For its computational efficiency, we designed the RSA processor could be used to compute both a modular multiplication and a modular exponentiation. The selection of these operations is chosen by the setting value of mode select bit of the control register.

The control register controls the operation of the RSA processor. It also contains the sequential information of exponent data used in encryption or decryption operation.

Table 1 shows the detailed bit information of control register.

Table 1. Bit information of control register

Bit	RSA processor operation	
0	Operation start signal	
1	Select the operation mode of processor	
	0 : multiplication $xy \bmod m$	1 : exponentiation $x^e \bmod m$
2	Select exponentiation mode	
	0 : encryption $c = x^e \bmod m$	1 : decryption $x = c^d \bmod m$
3	Initialize processor	
4	End operation notification	

To perform all these operation using 32-bit multiplier, adder and shift register, we need an efficient control block. The control block controls all data inputs and outputs of the modular multiplier, a path selection signal, an interrupt signal and a memory data access operations.

The designed RSA processor was implemented and verified using IESA system developed for smart card emu lating system. The IESA system mainly consists of ARM7 processor, interface circuit for smart card, smart card operating system, memory block, and designed RSA processor as shown in Figure 5.

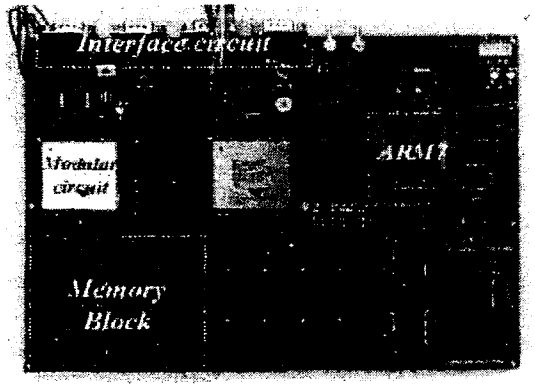

Figure 5: Photograph of the Implementation system.

5. Performance Analysis

The RSA processor using proposed architecture is designed by only use of 32-bit based elements. To control all input and output data of modular multiplier, there are some efficient control circuit. All elements are optimzed to reduce the hardware resources and to meet smart card's reasonable operating time.

For its verification, the proposed architecture is designed in 0.5 ㎛ Hynix technology using Active-HDL and the Synopsys FPGA Express is used as the logic synthesis tool. A Xininx Vertex series FPGA chip is used for the implementation of the RSA processor and the implement results are summarized in the Table 2.

Table 2. The feature of the 1024bit RSA processor

Technology	0.5 ㎛
Clock frequency	40 ㎒
Synthesized gate counts	2.7k gate
Modular Multiplication (1024bit)	356 ㎲
Encryption (16bit key)	6.4 ㎳
Decryption (1024bit key)	510 ㎳
RAM usages	≤ 4M bits

Because the smart card's tolerable response time according to a challnage signal is less than about 1 second, the implementation results indicate that the designed RSA processor is suitable for restricted system such as a smart card.

6. Conclusion

We proposed an efficient design scheme to implement an optimized 1024bit RSA cryptographic processor for restricted system such as smart card. The modified Montgomery algorithm made it possible to reduce circuit area and to meet reasonable operating time for smart card.

As a result, The designed modular multiplier circuit performs fast execution of modular multiplication and exponentiation with small size. From its high performance and small chip size, the designed RSA processor provides a good solution to pratical implementation for various security applications used in smart card or other stricted systems.

References

[1] P.L. Montgomery, "Modular Multiplication Without Trial Division," *Mathemat. of Computat.* , Vol. 44, pp.512-521, April 1985.

[2] S. R. Dusse and B. S. Kaliski, Jr., "A Cryptographic Library for the Motorola DSP56000," *Advances in Cryptology – Eurocrypt 90, Lecture Notes in Computer Science, No. 473*, Springer-Verlag, New York, 1990, pp. 230 ~244.

[3] Kai Hwang, *Computer Arithmetic*, John Wiley, 1997.

[4] Behrooz Parhami, *Computer Arithmetic*, Oxford University Press, 2000.

[5] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[6] Stephen E. Eldridge and Colin D. Walter, "Hardware Implementation of Montgomery's Modular Multiplication Algorithm," *IEEE Transactions On Computers*, Vol. 42, No. 6, June 1993, pp. 693 ~ 699.