

Hardware Implementation of Cryptographic Algorithm

Author: Daisuke Suzuki*

1. Introduction

Extensive research has been conducted on the hardware implementation of high-speed public key cryptosystems represented by RSA cryptography. In particular, various circuit architectures have been proposed for the processing of Montgomery multiplication⁽¹⁾. The Virtex-4 series and Spartan-3A DSP series released by Xilinx, Inc., a FPGA vendor, are equipped with a functional block (instead of a conventional multiplication unit) as a hardware macro, and they support dynamic changes in multiple-pattern multiplicative summation (henceforth called the "digital signal processing (DSP) function"). Some applications of this DSP function have already been reported, such as fast finite impulse response (FIR) filters and image processors; however, we believe that no cryptographic algorithms using this function have yet been reported, with the exception of the simple usage of such. This paper describes a modular exponentiation processing method and circuit architecture that can derive the maximum performance from this DSP function.

2. Processing Method

Montgomery multiplication is a high-speed processing method for modular exponentiation, which is heavily used in the field of public key cryptosystems. While Montgomery multiplication has many variations, we chose a processing method described in Ref. (2) and expanded its processing unit and flow. We also enhanced the FPGA hardware macros to provide more efficient usability. Figure 1 shows this processing method, which features the following three key improvements.

First improvement: For each multiple-length addition operation, additions for 2 blocks (34 bits) are performed in one loop, whereas each multiple-length multiplication operation is performed in a half loop count ($\alpha r/2$). This improvement, with the DSP48 being operated at the maximum frequency and other user logics at half frequency, eases the timing constraint on the circuits that are configured without using hardware macros, and allows the configuration of realistic circuits, while maintaining overall throughput. Second: By introducing branching control, Algorithm 1 can be processed in pipeline without any stalling in the process flow. Third: The number of DSP48 units used, α , is fixed in

this algorithm, and thus the same circuit can be used for input of different bit lengths.

Algorithm 1 Montgomery Multiplication for Virtex-4

Setting: radix: $2^k (= 2^{17})$, delay parameter : $d (= 1)$, no. of DSP48s : $\alpha (= 17)$, $2 < M < 2^h$ ($h \in \{512, 1024, 1536, 2048\}$), $\gcd(M, 2) = 1$, $(-MM' \bmod 2^{k(d+1)}) = 1$, $\bar{M} = (M' \bmod 2^{k(d+1)})M$, $0 \leq A, B \leq 2\bar{M}$, $h' = h + k(d+1) + 1$ no. of words at A and B: $n = \lceil h'/k \rceil$, no. of words processed by one DSP48 : $r = 2\lceil ([n/\alpha] / 2) \rceil$ ($r \in \{2, 4, 6, 8\}$), $A = \sum_{j=0}^{\alpha r - 1} (2^k)^j a_j$, $B = \sum_{j=0}^{n+d} (2^k)^j b_j$, $M'' = \sum_{j=0}^{\alpha r - 1} (2^k)^j m_j$, $S_i = \sum_{j=0}^{\alpha r - 1} (2^k)^j s_{(i,j)}$, $a_j, b_j, m_j, s_{(i,j)} \in \{0, 1, \dots, 2^k - 1\}$, $a_j = b_j = 0$ for $j \geq n$ and $m_j = 0$ for $j \geq \lceil h/k \rceil$.

Input: A, B, M''

Output: $MM(A, B) = S_{n+3} \equiv ABR^{-1} \bmod M$, $0 \leq S_{n+3} \leq 2\bar{M}$

```

1:  $S_0 := 0; q_{-1} := 0;$ 
2: for  $i = 0$  to  $n + 1$  do
3:    $\text{carry} := 17'b0; \text{cv} := 1'b0; \text{cs} := 1'b0;$ 
   /* Multiple-length multiplication: MUL_AB */
4:   for  $j = 0$  to  $\alpha r - 1$  do
5:      $\text{carry} || p_j := b_i a_j + \text{carry};$ 
6:   end for
   /* Multiple-length multiplication: MUL_MQ */
7:   for  $j = 0$  to  $\alpha r - 1$  do
8:     if  $j = 0$  then
9:        $\text{carry} || v_0 := q_{i-d} m_j + p_0;$ 
10:    else
11:       $\text{carry} || u_j := q_{i-d} m_j + \text{carry};$ 
12:    end if
13:  end for
   /* Calculation  $q_i$ : ADD_V0S1 */
14:   $q_{i+1} := v_0 + s_{(i,1)};$ 
   /* Multiple-length addition: ADD_PU */
15:  for  $j = 0$  to  $\alpha r/2 - 1$  do
16:    if  $j = 0$  then
17:       $\text{cv} || v_1 || v_0 := (p_1 || 17'b0) + (u_1 || v_0);$ 
18:    else
19:       $\text{cv} || v_{2j+1} || v_{2j} := (p_{2j+1} || p_{2j}) + (u_{2j+1} || u_{2j}) + \text{cv};$ 
20:    end if
21:  end for
   /* Multiple-length addition: ADD_VS */
22:  for  $j = 0$  to  $\alpha r/2 - 1$  do
23:     $\text{cs} || s_{(i+1, 2j+1)} || s_{(i+1, 2j)} := (v_{2j+1} || v_{2j}) +$ 
       $(s_{(i, 2j+2)} || s_{(i, 2j+1)}) + \text{cs};$ 
24:  end for
25: end for
26:  $S_{n+3} := S_{n+2} || s_{(n+1, 0)};$ 
27: return  $S_{n+3};$ 

```

Fig. 1 Montgomery multiplication algorithm for Virtex-4

3. Hardware Configuration

Figure 2 shows the basic circuit that performs each processing task of Algorithm 1. Input A and M'' are entered every 34 bits (2 blocks) at a time from left to right and stored in each specified DMEM. Only A is updated for each Montgomery multiplication. DMEM is implemented using a single port memory. Once a_j ($0 \leq j$

$\leq r-1$) is stored in the leftmost DMEM, the circuit connected to it begins processing according to Algorithm 1. The DSP48 that performs multiplication of the least significant block (MUL_AB and MUL_MQ) switches the calculation type depending on whether the carry bit is on or off.

ADD_PU is performed by the circuit consisting of the adders and LA1s (Latency Adjuster) shown at the center in Fig. 2. First, two blocks of the result from MUL_AB calculation are provided at the same time from the DSP48 and entered into the adder at the negative edge of the clock (clk1). At this time, by resetting all other inputs to zero, the calculation result from MUL_AB is stored in LA1 as it is. Then the calculation result from MUL_MQ is added to the calculation result from MUL_AB already stored in LA1. At this time, the difference between the input timing of MUL_AB and MUL_MQ is $r/2$ cycle. Carry propagation from the addition is one of two cases: propagation to the same adder or to the next adder. For the circuit shown in Fig. 2, a flip-flop to store the carry is implemented for each case to improve the timing.

The circuit shown at the bottom in Fig. 2 performs ADD_VS calculation. This circuit performs simultaneous addition of two blocks, $S_{(i, 2j+1)}$ and $S_{(i, 2j+2)}$, which are respectively provided from LA1 and LA2 at the timing when the calculation result is provided from ADD_PU. For more details about the hardware configuration, see Ref. (3).

4. Hardware Performance

This section describes the characteristics of a prototype circuit for modular exponentiation based on the Montgomery multiplication circuit shown in Fig. 2. In Table 1, a comparison is made between the results of the circuit placed and routed using XC4VFX12-10SF363 as the target device and the previous research results. To the best of our knowledge, this is the world's fastest processing performance for a modular exponentiation using FPGA. This circuit can be implemented on FPGA having the smallest logic scale among the Virtex-4 series. In addition, modular exponentiation from 512 to 2048 bits can be performed on the same circuit, providing high scalability.

Table 1 Characteristics of prototype circuit and comparison with previous research results

Architecture	Proposed method	Reference ⁽⁴⁾	Reference ⁽⁵⁾
Target device	XC4VFX12-10	XC2V3000-6	XC40250XV
Scalability	Y	N	N
512 bit MEX time	0.26 ms (max)	0.59 ms (avr)	2.93 ms (max)
512 bit MEX area	3937 slices + 17 DSP48	8235 slices + 32 multipliers	3413 slices
1024 bit MEX time	1.71 ms (max)	2.33 ms (avr)	11.95 ms (max)
1024 bit MEX area	3937 slices + 17 DSP48	14334 slices + 62 multipliers	6636 slices

* MEX: Modular exponentiation

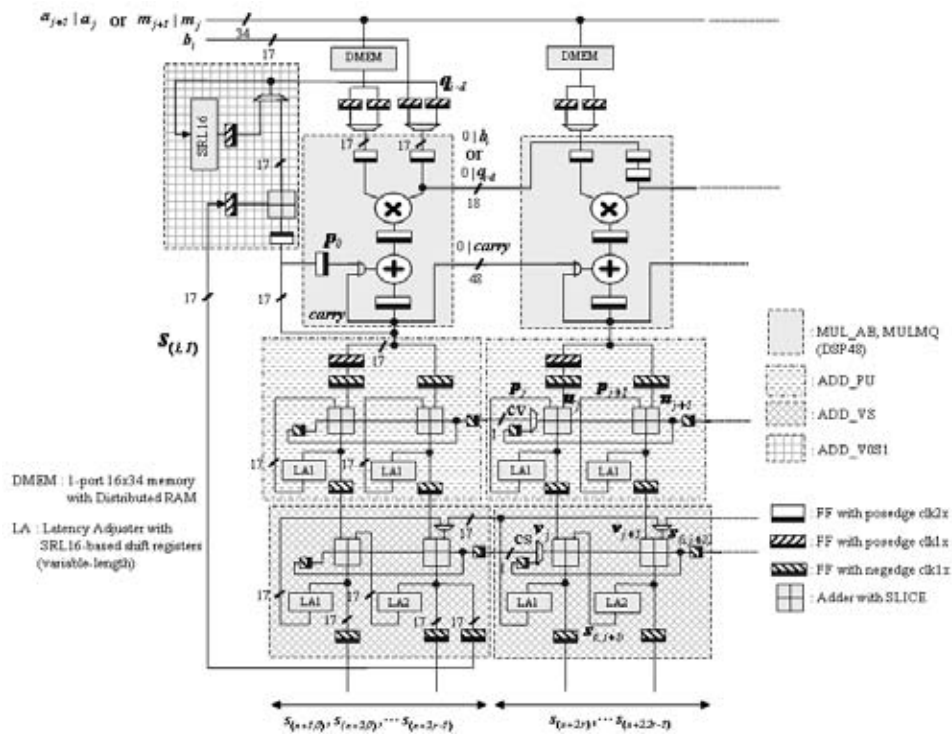


Fig. 2 Montgomery multiplication circuit

5. Conclusion

This paper presented hardware implementation technology for cryptographic algorithms using an example of high-speed hardware for public key cryptosystems. In addition to this example, we have also developed block cipher and stream cipher circuits, a hash function circuit, and other hardware setups required for the construction of security systems. We will strive to develop fully optimized applications using these hardware implementation technologies.

References

- (1) Montgomery, P.L.: Modular Multiplication without Trial Division. *Mathematics of Computation*, Vol. 43, No. 170, pp. 519-521, 1985.
- (2) Orup, H.: Simplifying quotient determination in high-radix modular multiplication, *Proc. of the 12th Symposium on Computer Arithmetic*, pp. 193-199, 1995.
- (3) Suzuki, D.: How to Maximize the Potential of FPGA Resources for Modular Exponentiation. *CHES 2007, LNCS*, Vol. 4727, pp. 272-288, 2007.
- (4) Blum, T., Paar, C.: High-Radix Montgomery Modular Exponentiation on Reconfigurable Hardware, *IEEE Transactions on Computers*, Vol. 50, No. 7, pp. 759-764, 2001.
- (5) Tang, S.H., Tusi, K.S., Leong P.H.W.: Modular Exponentiation using Parallel Multipliers, *FPT 2003*, pp. 52-59, 2003.