How to Maximize the Potential of FPGA Resources for Modular Exponentiation

Daisuke Suzuki

Mitsubishi Electric Corporation, Information Technology R&D Center,
5-1-1 Ofuna Kamakura, Kanagawa, 247-8501, Japan
Suzuki.Daisuke@bx.MitsubishiElectric.co.jp

Abstract. This paper describes a modular exponentiation processing method and circuit architecture that can exhibit the maximum performance of FPGA resources. The modular exponentiation architecture proposed by us comprises three main techniques. The first technique is to improve the Montgomery multiplication algorithm in order to maximize the performance of the multiplication unit in FPGA. The second technique is to improve and balance the circuit delay. The third technique is to ensure and make fast the scalability of the effective FPGA resource. We propose a circuit architecture that can handle multiple data lengths using the same circuits. In addition, our architecture can perform fast operations using small-scale resources; in particular, it can complete 512-bit modular exponentiation in 0.26 ms by means of XC4VF12-10SF363, which is the minimum logic resources in the Virtex-4 Series FPGAs. Also, the number of SLICEs used is approx. 4000 to make a very compact design. Moreover, 1024-, 1536- and 2048-bit modular exponentiations can be processed in the same circuit with the scalability.

1 Introduction

The fast hardware implementation of public-key cryptosystems has been extensively researched thus far; in particular, a circuit architecture using Montgomery multiplication [1] has often been proposed [2–19]. There are two main arguments concerning these researches. The first refers to an efficient architecture that the standard complementary metal oxide semiconductor (CMOS) gates are supposed to form, and the second refers to an architecture limited to the specified devices such as a field programmable gate array (FPGA).

The latter argument originates from the fact that the FPGA architecture has advanced significantly over the last ten years. In current FPGAs, basic components such as a multiplexer (MUX), shift register and two-input adder, large-capacity dual-port memory, and multiplier are pre-mounted as hardware macros, along with the RAM-based lookup table (LUT) and flip-flop (FF) to construct the user logic. A circuit architecture that is efficient at the CMOS gate level is not necessarily efficient in an FPGA; therefore, the above mentioned architecture using pre-mounted hardware macros has been proposed.

In 2004, Xilinx (an FPGA vendor) introduced the Virtex-4 Series FPGAs [22]. These are equipped with a functional block, instead of a conventional multiplication unit, as a hardware macro, and they support dynamic changes in the multiple-pattern multiplicative summation (henceforth called the "digital signal processing (DSP) function"). Some applications of this DSP function have already been reported, such as the fast finite impulse response (FIR) filter and an image processing; however, we believe that no cryptographic algorithms using this function have yet been reported excluding a simple usage such as [19].

This paper describes a modular exponentiation processing method and circuit architecture that can derive the maximum performance from this DSP function. The modular exponentiation architecture proposed by us comprises three main techniques. The first technique is to improve the Montgomery multiplication algorithm in order to maximize the performance of the DSP function. The performance of this DSP function depends on its operating frequency and operation rate. In order to maximize its performance, it is necessary to improve the algorithm such that the DSP function works at the maximum operating frequency and consumes the least time. The second technique is to improve and balance the circuit delay. The operating frequency is specified by the circuit path having the maximum delay in the conventional synchronous circuit. This paper maximizes the performance of the DSP function by optimizing the division method of pipeline processing operations and the circuit layout taking into consideration the FPGA characteristics. The third technique is to ensure and improve the scalability of the effective FPGA resources. We propose a circuit architecture that can handle multiple data lengths using the same small-scale circuits. In addition, the architecture proposed by us can perform fast operations using small-scale resources; in particular, it can complete 512-bit modular exponentiation in 0.26 ms by using XC4VF12-10SF363, which is the minimum logic resources in the Virtex-4 Series FPGAs. Moreover, 1024-, 1536- and 2048-bit modular exponentiations can be processed in the same circuit with the scalability.

2 Features of Virtex-4 Series FPGAs

This section describes the architecture and performance of the Virtex-4 Series FPGAs that are described and used in this paper. The following descriptions are limited to only the relevant issues with regard to this paper. For more information, refer to [22–24].

2.1 Internal Configuration

First, we explain the architecture of the Virtex-4 Series FPGA. As shown at the top of Fig. 1, this FPGA comprises an 18 Kbit dual-port memory group called Block RAM (henceforth called "BRAM"), a hardware macro group called XtremeDSP (henceforth called "DSP48") to provide the above mentioned DSP function, and a configurable logic block (CLB) as a basic block for the implementation of user logic [22, 23]. The schematic representation of the CLB's internal

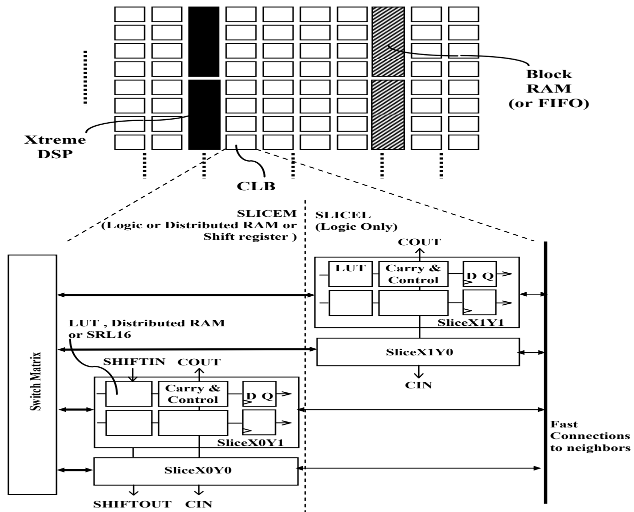


Fig. 1. Internal configuration of Virtex-4

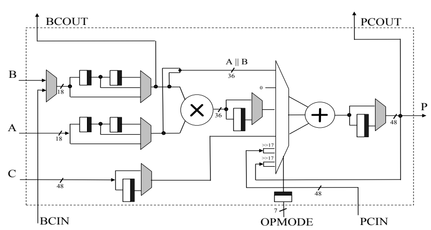
configuration is shown at the bottom of Fig. 1. The CLB comprises four blocks called SLICE. Each SLICE is divided into a pair of blocks, namely, SLICEL and SLICEM. The former comprises LUTs, FFs, MUXs, and carry logics for addition processing. The latter includes the SLICEL functions and it is also equipped with the operation mode for the 16×1-bit (maximum) single-port memory with the LUT function (henceforth called "distributed RAM") or the 16×1-bit (maximum) variable shift register (henceforth called "SRL16").

Fig. 2 shows a schematic representation of the internal configuration of DSP48. The DSP48 is designed to support dynamic changes in a 42-pattern multiplicative summation by switching the control signals (OPMODE) [23]. Controlling the ash-colored MUXs in Fig. 2 during the configuration operation allows us to change the latency of the signal conductors. The maximum operating frequency of the DSP48 depends on the speed grade of the FPGA and the latency set above, and the operation is valid at a maximum frequency of 400 MHz in the lowest speed grade (-10) [24] [1]. A detailed description is provided in the next section.

2.2 Characteristics of Basic Functions

We first examine the performance of the FPGA functions before examining the Montgomery multiplication, modular exponentiation processing method, and all the circuits. The multiple circuit architectures are generally supposed to perform

[1] The maximum operating frequency of the digital clock manager (DCM) in an FPGA is also 400 MHz, which is the threshold operating frequency in the speed grade of FPGA.



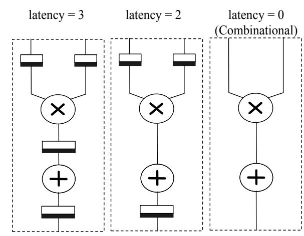


Fig. 2. Internal configuration of DSP48 **Fig. 3.** Examples of the latency in DSP48

Table 1. Delay time of adders composed by carry logics in SLICEMs

Functions of adder	No. of LUTs used	Circuit delay
8-bit 2-input addition	8 LUTs	2.201 ns
16-bit 2-input addition	16 LUTs	2.734 ns
32-bit 2-input addition	32 LUTs	3.564 ns
8-bit 3-input addition	14 LUTs	4.044 ns
16-bit 3-input addition	29 LUTs	4.407 ns
32-bit 3-input addition	65 LUTs	5.188 ns

a specific processing operation; currently, these are being used to determine which circuit architecture is advantageous to form the circuit in the FPGA. Otherwise, it is important to check if the examined circuit architecture is actually within available constraints.

First, we describe the performance of DSP48, which is important with regard to this paper. When three circuit architectures with different latencies are compared as shown in Fig. 3, their maximum operating frequencies from left to right are observed to be 400 MHz, 253 MHz, and 226 MHz (4.41ns) or less according to [24]. The value of the third circuit in this figure is described with “or less” because it is combined with the DSP48 and does not include the FF setup time and hold time necessary to actually operate within 4.41 ns and the wiring delay.

Therefore, in order to maximize the performance of the DSP48, we need to optimize the hardware architecture under the conditions that the clock frequency of DSP48 is 400MHz and the latency is 3 or more cycles.

Next, we describe the performance of the addition processing that is required for performing the Montgomery multiplication and modular exponentiation. Table 1 lists the results for certain adders evaluated using different parameters: the number of LUTs used and their circuit delay. These adders are composed by using the carry logics in SLICEs. The number of LUTs used increases in proportion to the bit length and the number of inputs. On the contrary, the circuit delay does not increase in proportion to the number of LUTs. This is because the carry propagation delay of the carry logic is very small (approximately 0.09 ns), while the wiring delay (approximately 1-2 ns) between the LUTs and the FF setup time (approximately 0.5-1.4 ns) are significantly greater. Therefore, the circuit delay tends to increase significantly in the 3-input addition that utilizes a greater number of LUTs than the 2-input addition.

Based on the results in Table 1, it is assumed that the addition limit operable at the maximum operating frequency of 400 MHz may be approximately 8-bit 2-input addition. Another interpretation of the results in Table 1 is that 32-bit 2-input addition is operable at approximately 250 MHz.

Based on the above descriptions, the partial circuit structured as a hardware macro has a potentially higher processing performance. However, it is verified that it is difficult to structure the user logic using the LUT in order to operate it at the maximum operating frequency. This trade-off is a design problem.

3 Proposed Architecture

This section describes the method for structuring the modular exponentiation circuits by using our proposed DSP functions.

3.1 Design Policy

Based on the characteristics of the basic functions of the Virtex-4 Series FPGAs described in Section 2, we evaluated the circuit architecture to satisfy the following requirements as the overall design policy.

(1) To allow the DSP48 to operate at a maximum operating frequency of 400 MHz.
(2) To design the circuits such that the DSP48 operation does not stall during the Montgomery multiplication.
(3) To enable multiple bit lengths such as 512 bits and 1024 bits to be processed using the same circuits for Montgomery multiplication.
(4) To set the bus width of the input/output signals to less than 36 bits in order to simplify the control of the operation results.
(5) To implement the circuits even on the minimum device of Virtex-4 Series.

Items (1) and (2) are essential from the viewpoint of realizing the maximum performance of DSP48. Item (3) ensures scalability. Since the goal is to form the FPGA, the circuits may be reconfigured according to the bit length in order to achieve scalability. However, it is known that the FPGA circuits have a reconfiguration time of some milliseconds; therefore, this reconfiguration cannot be carried out based on the operating system. In addition, scalability must be ensured in the same circuit even when using functions that support dynamic changes in the operation patterns of the DSP48. Item (4) ensures the effective use of the FPGA resources. Assuming that the intermediate values such as the pre-operation results of modular exponentiation and the operation results of Montgomery multiplication are controlled within the FPGA, an effective circuit architecture may be created by employing a large memory capacity BRAM. Data can be processed at up to 36 bits per BRAM. Thus, many BRAMs are required to structure the system that data of large bus width is stored as it is. On the contrary, data can be stored in up to 512 depth per BRAM for 36-bit input/output operations. Therefore, the BRAM characteristics can be applied when the operation results are controlled as the stream data in the direction of depth with the narrow bus width. Further, the circuit having large bus width may always reduce its final performance from the viewpoint of the circuit location and wiring. The above viewpoints pertain to Item (4). With regard to Item (5), we believe that it is not necessary to use the large-scale FPGA and most of its resources only for cipher operations. On the other hand, it is difficult to quantitatively indicate which detailed circuit scale is generally permitted. Finally, we determined that it is possible to form the circuit with the minimum number of logics in the Virtex-4 Series FPGAs. In this case, the device name is XC4VF12, the number of SLICEs is 5472, the number of DSP48s is 32, and the number of BRAMs is 36.

3.2 Processing Method

This section describes the detailed processing method for Montgomery multiplication and modular exponentiation.

Montgomery Multiplication For the DSP48 to be operable at the maximum operating frequency under the conditions specified in the previous section, it must have some latency during the operations. Therefore, the processing method for Montgomery multiplication was improved on the basis of the Montgomery multiplication algorithm for pipeline processing operations in [3, 4]. Algorithm 1 shown below explains the Montgomery multiplication algorithm, as specified in [4].

Next, we describe the method for improving Algorithm 1 considering the features of Virtex-4. The processing method for Montgomery multiplication proposed in this paper is a combination of Algorithm 1 and the Multiple Word

---

Algorithm 1 Modular Multiplication with Quotient Pipelining [4]

Setting: radix : 2^k; delay parameter : d; no. of blocks : n; multiplicand : A; multiplier : B; modulus : M, $M > 2$, $\gcd(M, 2) = 1$, $(-MM' \bmod 2^{k(d+1)}) = 1$, $\tilde{M} = (M' \bmod 2^{k(d+1)})M$, $4\tilde{M} < 2^{kn} = R$, $M'' = (\tilde{M}+1)/2^{k(d+1)}$, $0 \le A, B \le 2\tilde{M}$, $B = \sum_{i=0}^{n+d}(2^k)^i b_i$, $b_i \in \{0, 1, \cdots, 2^k - 1\}$, for $i \ge n$ and $b_i = 0$
Input: A, B, M''
Output: $\mathrm{MM}(A, B) = S_{n+d+2} \equiv ABR^{-1} \bmod M$, $0 \le S_{n+d+2} \le 2\tilde{M}$
1: $S_0 := 0; q_{-d} := 0; \cdots; q_{-1} := 0;$
2: **for** $i = 0$ to $n + d$ **do**
3: $\quad q_i := S_i \bmod 2^k;$
4: $\quad S_{i+1} := S_i/2^k + q_{i-d}M'' + b_iA;$
5: **end for**
6: $S_{n+d+2} := 2^{kd}S_{n+d+1} + \sum_{j=0}^{d-1} q_{n+j+1}2^{kj};$
7: **return** $S_{n+d+2};$

Radix-2 Montgomery Multiplication (MWR2MM); the latter is a processing method for Montgomery multiplication explained in [7], and is the method for which the processing unit and flow are optimized for the Virtex-4.

The Montgomery multiplication algorithm proposed in this paper is described below as Algorithm 2. First, the settings of Algorithm 2 are explained. Since the DSP48 has a 17-bit shift function, the radix is set to $2^k = 2^{17}$. Next, the delay parameter must be determined by the required cycle before settling q_{i+1}; the smaller the value of the delay parameter, the lesser is the number of cycles required for the total Montgomery multiplication. In Algorithm 2, it is assumed that α-piece DSP48s are used for data processing. Here, the bit length of M is set to h and the bit length of A and B is set to h'. At this stage, Algorithm 1 provides the relational expression of $h' = h+k(d+1)+1$. The number of words n is defined as $n = \lceil h'/k \rceil$. Note that the bit length of one word is $k = 17$. Also, the number of words r processed by one DSP48 is defined as $r = 2\lceil (\lceil n/\alpha \rceil)/2 \rceil$. This implies that one DSP48 is applied to process only r words from the total number of words n. Note that the number of words r is set to an even number. The number of words processed by α-piece DSP48s is αr and the words over n are processed after the dataset by zero padding. The parameter (for example, $\alpha = 17$) specified in the parentheses in Algorithm 2 is a setting in the Montgomery multiplication circuits that will be explained in detail in the following section.

Next, we explain the correspondence between Algorithms 1 and 2. Here, $||$ in Algorithm 2 indicates a bit concatenation. In Algorithm 2, the multiple-length multiplication of $b_i A$ in Algorithm 1 is first calculated using the DSP48 (MUL_AB). This operation requires n multiplications. Here, it is assumed that one DSP48 performs r multiplications, and following which another DSP48 receives a carry to continue the subsequent multiplications. Therefore, this implies that α-piece DSP48s perform the required minimum number of n multiplications by dividing them into r multiplications in common per unit. The DSP48, which provides a carry, begins performing the multiple-length multiplication (MUL_MQ) corresponding to $q_{i-d}M''$ in the next step of Algorithm 1. In the manner as MUL_AB, this DSP48 performs r multiplications, following which another DSP48 receives a carry to continue the subsequent multiplications.

The above mentioned processing operations obtain the output values p_j and u_j in Algorithm 2 from the α-piece DSP48s. It is necessary to perform the two types of multiple-length addition operations (ADD_PU and ADD_VS), as described in Algorithm 2, in order to obtain individual outputs. These processing operations are performed by an adder implemented with the LUT outside the DSP48. At this time, as shown in Algorithm 2, it is supposed that one loop of each addition completes 2 words (34 bits) to require the number of loops $\alpha r/2$ that are equivalent to half a multiple-length multiplication above. Note that the value r is an even number in the setting above. In other words, the DSP48 carries out the "single word multiplication" at the maximum operating frequency and the adder with the LUT performs the "double word addition" at half the maximum operating frequency, thus maintaining the total throughput. This operation is henceforth called the "SMDA." The advantage of SMDA is that the user logic can

Algorithm 2 Modified Algorithm 1 for Virtex-4

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

Input: A, B, M''

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

$S_0 := 0; q_{-1} := 0;$
for $i = 0$ to $n + 1$ **do**
  carry := $17'$b0; cv := $1'$b0; cs := $1'$b0;
  /* Multiple-length multiplication: MUL_AB */
  for $j = 0$ to $\alpha r - 1$ **do**
    carry|| $p_j := b_i a_j +$ carry;
  end for
  /* Multiple-length multiplication: MUL_MQ */
  for $j = 0$ to $\alpha r - 1$ **do**
    if $j = 0$ **then**
      carry|| $v_0 := q_{i-d} m_j + p_0$;
    else
      carry|| $u_i := q_{i-d} m_j +$ carry;
    end if
  end for
  /* Calculation q_i: ADD_V0S1 */
  $q_{i+1} := v_0 + s_{(i,1)}$;
  /* Multiple-length addition: ADD_PU */
  for $j = 0$ to $\alpha r/2 - 1$ **do**
    if $j = 0$ **then**
      cv$||v_1||v_0 := (p_1||17'b0) + (u_1||v_0)$;
    else
      cv$||v_{2j+1}||v_{2j} := (p_{2j+1}||p_{2j}) + (u_{2j+1}||u_{2j}) +$ cv;
    end if
  end for
  /* Multiple-length addition: ADD_VS */
  for $j = 0$ to $\alpha r/2 - 1$ **do**
    cs$||s_{(i+1,2j+1)}||s_{(i+1,2j)} := (v_{2j+1}||v_{2j}) + (s_{(i,2j+2)}||s_{(i,2j+1)}) +$ cs;
  end for
end for
$S_{n+3} := S_{n+2}||s_{(n+1,0)}$;
return S_{n+3};

be designed under the actual constraints while deriving the maximum potential performance of DSP48. As described in Table 1, approximately 32-bit 2-input addition can operate at 200 MHz (5 ns), which is half the operating frequency of 400 MHz. However, Table 1 indicates that it is difficult to perform 3-input addition at 200 MHz. Therefore, it is assumed that Algorithm 2 uses the pipeline processing operation to divide the two multiple-length addition operations after every 2-input addition.

Next, we explain the branch operation in Algorithm 2. The branch operation is introduced in the case where $j = 0$ in MUL_MQ and ADD_PU in order to reduce the necessary latency until q_{i+1} is settled. The addition for p_0, which was calculated in MUL_AB, is performed simultaneously with the multiplication for the least significant word in MUL_MQ. Since the multiplication for the least significant word does not require the addition with a carry, this operation can be performed only by modifying the operation mode of DSP48. Next, v_0 is settled at the output of MUL_MQ. Therefore, the operation required to settle q_{i+1} is an addition with $s_{(i,1)}$, such that q_{i+1} is settled with a smaller latency than that for a calculation of v_0 in MUL_MQ. The latency required to settle q_{i+1} affects the delay parameter in Algorithm 2. The Montgomery multiplication circuits described in the following section are operable with $d = 1$.

Sliding-window Exponentiation The sliding window [21] is one of the fast modular exponentiation algorithms in which the processing operation of multiple-bit exponentiations is performed; it is an improved m-ary exponentiation algorithm. The modular exponentiation is described below with the sliding window exponentiation as Algorithm 3. Generally, the hardware modular exponentiation is often carried out using the binary exponentiation [20]. However, since the Virtex-4 Series to be formed in this case has several large-capacity memory blocks as hardware macros, we attempted to form the Virtex-4 Series with the sliding window such that the resources were effectively utilized. All modular exponentiations in Algorithm 3 are based on the assumption that they are applied to the Montgomery multiplication described in Algorithm 2. The memory capacity required to store X_{2i+1} from Algorithm 2 is $2^{w-1} \times n \times k$ bits.

The modular exponentiation circuit explained in this paper was configured with the window size set to $w = 5$. This is because the maximum processing time is the least in 512-bit modular exponentiation. The Montgomery multiplication circuits described in this paper are designed to be operable in the same circuits for the maximum 2048-bit modulus. In this case, at least 2 BRAMs are necessary to store X_{2i+1}.

3.3 Hardware Architecture

This section describes the detailed circuit architecture required to process Algorithms 2 and 3.

Montgomery Multiplier First, we explain the circuit architecture required to process the Montgomery multiplication in Algorithm 2; the basic circuit is

Algorithm 3 Modular exponentiation with sliding-window exponentiation [21]

Input: $M'', X, R_R = R^2 \bmod M$, $E = (e_t, e_{t-1}, \cdots, e_1, e_0)_2$
Output: $Y \equiv X^E \bmod M$

$X_1 := \mathrm{MM}(X, R_R)$; $C_R := \mathrm{MM}(1, R_R)$; $X_2 := \mathrm{MM}(X_1, X_1)$;
for $i = 1$ to $2^{w-1} - 1$ **do**
  $X_{2i+1} := \mathrm{MM}(X_{2i-1}, X_2)$;
end for
$S_R := C_R$;
for $i = t$ to 0 **do**
  if e_i=0 **then**
    S_R:= $\mathrm{MM}(S_R, S_R)$; i:= $i - 1$;
  else
    Searching maximum odd-number binary digit string $(e_i, e_{i-1}, \cdots, e_l)_2$ within window size, $i - l + 1 \le w$
    for $j = 0$ to $i - l$ **do**
      S_R:= $\mathrm{MM}(S_R, S_R)$;
    end for
    S_R:= $\mathrm{MM}(X_{(e_i, e_{i-1}, \cdots, e_l)_2}, S_R)$; $i := l - 1$;
  end if
end for
$Y := \mathrm{MM}(1, S_R)$;
return Y;

shown in Fig. 4. Input data A and M" are inputted from the left every 34-bits (two words) and are stored into the specified DMEMs. Data M" is only stored immediately after implementing the modular exponentiation. Therefore, only data A is updated after every Montgomery multiplication. The DMEM is implemented with a distributed RAM having the SLICE function and it is used as a single-port memory of 8 (depth) $\times$ 34 (bit width). In this case, the capacity of DMEN can correspond to the modulus size up to 2048 bit. When $a_j(0 \le j \le r-1)$ is stored into the leftmost DMEM, the lower connecting circuit performs the processing operations according to Algorithm 2. The leftmost DSP48 performs the first r of the αr multiplications in MUL_AB and MUL_MQ. This operation is performed by switching the OPMODE signal, which is shown in Fig. 2 to two patterns. Table 2 shows the sequence of r multiplications and their corresponding OPMODE values.

The second DSP48 from the left side switches the two patterns of the multiplicative summation to perform the next r multiplications in the same manner. Table 2 shows the sequence of these r multiplications and their corresponding OPMODE values. The third and following DSP48s perform the operation in the same sequence as those in the second DSP48.

The ADD_PU processing operation is performed in the circuits including the adders and LA1 (latency adjuster) shown at the center of Fig. 4. The two-step positive/negative FFs are placed on the left path of the circuits and the one-step negative FF is placed on the right path. This is because it is necessary to adjust the latency of lower-located words. This state allows two words as the

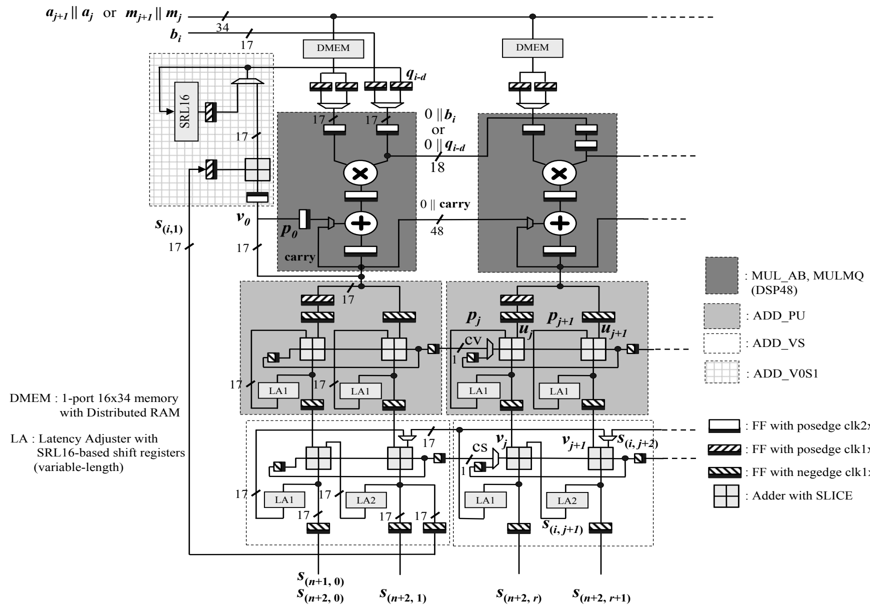


Fig. 4. Montgomery multiplier using DSP48

result of the MUL_AB operation transmitted from the DSP48 to be entered simultaneously into the adder with the negative clock (clk1x). Currently, the result of the MUL_AB operation is directly stored into the LA1 by resetting the LA1 output value to 0. Next, the result of the MUL_MQ operation is used to perform the addition with the result of the MUL_AB operation that has been pre-stored in LA1. The difference in the input time between the results of MUL_AB and MUL_MQ operations is a $r/2$ cycle depending on the modulus size.

The carry propagation in the addition must handle two cases: re-propagation to the same adder or propagation to the neighboring adder. The adders are located linearly due to the characteristics of the FPGA. When a carry FF is held in common, it is necessary to wire two adders to extend the circuit delay. In the circuits shown in Fig. 4, the different carry FFs are placed after every two cases in order to improve the circuit delay.

The lower circuits shown in Fig. 4 perform the ADD_VS processing operation. In the output timing of the result of the ADD_PU operation, the circuits perform simultaneous simultaneous additions for two words $s_{(i,2j+1)}$ and $s_{(i,2j+2)}$ that are transmitted from LA1 and LA2, respectively. At this stage, it should be ensured that $s_{(i,2j+2)}$ outputs data from LA1 at the right of the figure only in the first cycle, following which it outputs data from LA1 at the left. Among the lower FFs shown in Fig. 4, the FF connected to the output port is controlled to transmit 0

Table 2. Multiplication sequence of DSP48

512 bit mode ($r = 2$)						
Count	Leftmost DSP48			2nd DSP48 from left		
	Operation	OPMODE	Remarks	Operation	OPMODE	Remarks
0	$b_i a_0$	7'h35	Reset C	$q_{i-2}m_2$ + carry	7'h55	Carry is received from leftmost DSP48
1	$b_i a_1$ + carry	7'h65	-	$q_{i-2}m_3$ + carry	7'h65	-
2	$q_{i-1}m_0 + p_0$	7'h35	p_0 is stored into C	$b_i a_2$ + carry	7'h55	Carry is received from leftmost DSP48
3	$q_{i-1}m_1$ + carry	7'h65	-	$b_i a_3$ + carry	7'h65	
4	$b_{i+1}a_0$	7'h35	Reset C	$q_{i-1}m_2$ + carry	7'h55	Carry is received from leftmost DSP48
...	...	...	...	...	...	...
2048 bit mode ($r = 8$)						
Count	Leftmost DSP48			2nd DSP48 from left		
	Operation	OPMODE	Remarks	Operation	OPMODE	Remarks
0	$b_i a_0$	7'h35	Reset C	$q_{i-2}m_8$ + carry	7'h55	Carry is received from leftmost DSP48
1	$b_i a_1$ + carry	7'h65	-	$q_{i-2}m_9$ + carry	7'h65	-
...	...	...	...	...	...	...
6	$b_i a_6$ + carry	7'h65	-	$q_{i-2}m_{14}$ + carry	7'h65	-
7	$b_i a_7$ + carry	7'h65	-	$q_{i-2}m_{15}$ + carry	7'h65	-
8	$q_{i-1}m_0 + p_0$	7'h35	p_0 is stored into C	$b_i a_8$ + carry	7'h55	Carry is received from leftmost DSP48
9	$q_{i-1}m_1$ + carry	7'h65	-	$b_i a_9$ + carry	7'h65	
...	...	...	...	...	...	...
14	$q_{i-1}m_6$ + carry	7'h65	-	$b_i a_{14}$ + carry	7'h65	
15	$q_{i-1}m_7$ + carry	7'h65	-	$b_i a_{15}$ + carry	7'h65	
16	$b_{i+1}a_0$	7'h35	Reset C	$q_{i-1}m_8$ + carry	7'h55	Carry is received from leftmost DSP48
...	...	...	...	...	...	...

with the synchronous reset function until S_{n+3} is entered completely. This will be explained later.

In Fig. 4, LA1 and LA2 are the shift registers whose latency is changeable from 1 to 4 and from 2 to 5, respectively. Further, LA1 and LA2 support the 0 resetting function. These units comprise variable-length shift registers based on SRL16. In this case, LA1 and LA2 can correspond to the modulus size up to 2048 bit. The circuit delay of SRL16 is larger than that of the conventional LUT. In order to improve this circuit delay, the FF output data is used and the relative position constraint is set to the components (Fig. 5). Since the latency value is a constant when the modulus size is determined, the signal to control the latency can be set to "false path."

ADD_V0S1 operation is performed in the upper left circuit shown in Fig. 4. This circuit has the FF of clock clk2x at the input port; however, the addition is performed according to the standard of clk1x [2] . The data path of this circuit is 17-bit 2-input addition and 1-step 2-1 MUX. This circuit operates at 200 MHz. The SRL16 in this circuit is required for adjusting the q_{i+1} latency and load signal to the DSP48 in the proper timing.

Modular Exponentiator Fig. 6 shows the overview of our modular exponentiator using Fig. 4. The modular exponentiator comprises the following components:

(a) IF_MEN, 2-port BRAM (512 (depth) × 34 (bit width)), external interface memory;

[2] This is the multi-cycle path for the FF output data with the clock "clk2x".

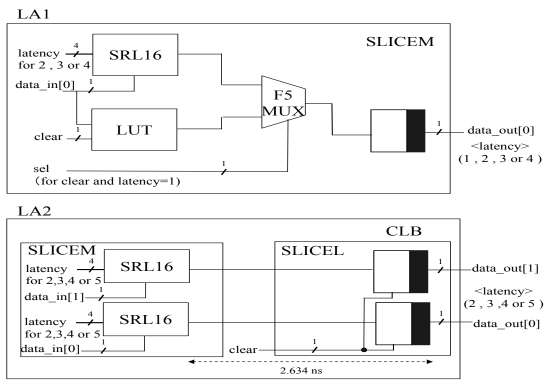


Fig. 5. Latency adjuster and relative position constraint

(b) A_MEN, 2-port BRAM (1024×17) × 2, template memory:
(c) B_MEN, 2-port BRAM (512×34), template memory;
(d) X_MEN, 2-port BRAM (1024×17) × 2, X_i storage memory ;
(e) E_MEN, 1-port BRAM (2048×5), exponent encode result storage memory;
(f) S_TRANS, circuits to convert the output signal of the Montgomery multiplication circuit into 34-bit stream data;
(g) MEX_CTL, control circuits for modular exponentiation circuits;
(h) MM_ENGINE, Montgomery multiplication circuits in Fig. 4 and their control circuits.

Item (a) facilitates the clock synchronization with the outside circuits such as CPU bus interface. The capacity of X_MEN in Item (c) can correspond to the modulus size up to 2048 bit even if Algorithm 3 is processed with $w = 5$.

The output signal of the MM_ENGINE is 578 bits; however, the effective output value is only 34 bits since S_{n+3} in a single cycle and others are controlled to be 0. Therefore, the output signal can be converted into 34-bit stream data by performing the XOR processing operation every 34 bits. This method can form the circuit more effectively than the method that selects data in the multiplexer and the circuit is operable at 200 MHz.

The output signal of S_TRANS is stored with B_MEM into A_MEM or X_MEM as necessary. When more than 34 bits of data are simultaneously updated in A_MEM or X_MEM, it starts to read and transmit data required for DMEM of the Montgomery multiplication circuits. The number of cycles required from the start of the output signal of S_TRANS to the start of the next Montgomery multiplication is $3r/2 + 3$ at the standard frequency of 200 MHz. Further, the processing time from the start of the Montgomery multiplication to the start of the output signal of S_TRANS is $(n + 1)r + 8$.

Considering all the supporting modulus size, the modular exponentiator shown in Fig. 6 is designed with the window size $w = 5$ in Algorithm 2. At this stage, the maximum number of Montgomery multiplications required for the modular exponentiation is $t + \lceil (t + 1)/5 \rceil + 20$ according to Algorithm 2.

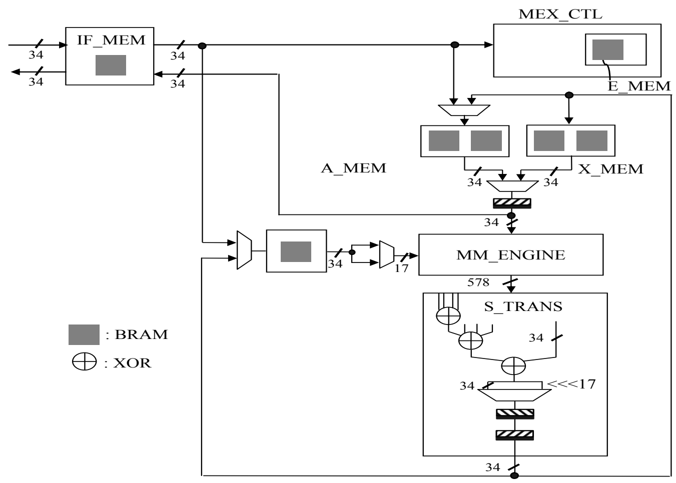


Fig. 6. Overview of our modular exponentiator

The exponent encoding operation in Algorithm 3 repeats the data search every bit. As a result, the encoding operation requires a number of cycles equivalent to the number of exponent bits. This processing operation is performed simultaneously with the calculation of X_{2i+1} in Algorithm 3. The calculation of X_{2i+1} requires more cycles than the exponent encoding operation. Therefore, the exponent encoding operation time does not affect the total operation time.

4 Performances Evaluation

The performances of the trial circuits are described below. Table 3 lists the results on XC4VFX12-10SF363 as a target device. The logic synthesis and the place-and-route are based on Simplify Pro 8.6.2 and ISE 8.1.03i, respectively.

The critical path of clk2x (400 MHz) is a selective signal of the MUX that is to be connected to the input ports A and B of the DSP48 shown in Fig. 4. The number of logic steps is one 2-1 MUX only. However, since the circuits are placed on a boundary with the hardware macros, their locating and wiring constraints are more difficult than those of conventional logic. Further, the large fan-out of the selective signal causes a significantly increase in the circuit delay. Ref. [25] describes a technique to improve the timing in such circuits; however, our trial circuits in this paper, which include this technique, are designed to make the fan-out of selective signal less than 4 in order to improve the timing.

The critical path of clk1x (200 MHz) is a path of the adders for ADD_PU and ADD_VS. This improves the timing by using some techniques described in Section 3.3.

It is revealed from Table 3 that our circuit designs allow 512-bit modular exponentiation to be performed in approximately 0.26 ms on XC4VF12-10SF363,

Table 3. Performances of our modular exponentiator

No. of SLICEs used	3937/5472
No. of BRAMs used	7/36
No. of DSP48s used	17/32
Critical path of 400-MHz operating circuits	2.493 ns
Critical path of 200-MHz operating circuits	4.988 ns
Max. operation time of 512-bit modular exponentiation	0.261 ms
Max. operation time of 1024-bit modular exponentiation	1.71 ms
Max. operation time of 1536-bit modular exponentiation	5.45 ms
Max. operation time of 2048-bit modular exponentiation	12.6 ms

Table 4. Comparison with Previous Implementations

Architecture	[8]	[11]	This work
Target device	XC40250XV	XC2V3000-6	XC4VFX12-10
Process	0.35 μm	0.12/0.15 μm	0.09 μm
Additional FPGA function	Basic function (: LUT, FF, Carry logics, Distributed RAM)	18Kbit BRAM, 18x18 multiplier	18Kbit BRAM, DSP48
Scalability	N	N	Y
512 bit MEX time	(Max.) 2.93 ms	(Avr.) 0.59 ms	(Max.) 0.261 ms
512 bit MEX area	3413 CLBs *1	8235 SLICEs, 32 Multipliers	3937 SLICEs, 17 DSP48s
1024 bit MEX time	(Max.) 11.95 ms	(Avr.) 2.33 ms	(Max.) 1.71 ms
1024 bit MEX area	6633 CLBs *1	14334 SLICEs, 62 Multipliers	3937 SLICEs, 17 DSP48s

*1 These CLBs are resources that correspond to SLICEs today.

which is the minimum logic resources of the Virtex-4 series. We believe that this is the fastest FPGA modular exponentiator. Further, the number of SLICEs used is approximately 4000, which leads to a very compact design. In addition, 1024-, 1536- and 2048-bit modular exponentiations can be processed in the same circuit due to its scalability.

We now compare our circuit designs with the previously reported ones. The purpose of this comparison is not to discuss the advantages and disadvantages of the circuit processing performance and circuit area since this is difficult to do so for circuits formed using different devices. This comparison is performed in order to observe the relation between the development of the FPGA architecture and the implementations of cipher circuits. Table 4 lists the performances of our circuit designs and two other designs. We selected these two designs since we determined that they were the most suitable to the FPGA architecture in each generation. The target FPGA described in [8] has functions such as the LUT, FF, adder logic, and distributed memory. Further, the target FPGA described in [11] has the multiplication function and BRAM as hardware macros along with the above mentioned functions. Our target FPGA has the DSP function instead of the multiplication function.

The improved performance of hardware macros contributes to faster cipher processing operations by designing the circuits other than the hardware macros in the form of SMDA, as explained in this paper. In addition, the operation patterns of DSP48 are useful to ensure scalability with the trade-off of circuits with few dynamically changeable functions. We conclude that the Virtex-4 architecture is at least effective for cipher processing operations due to the use of the modular exponentiator in comparison with the conventional FPGA architectures.

5 Conclusion

This paper describes the architecture of modular exponentiators, which effectively use typical hardware macros such as the DSP function of an FPGA, and we proposed the processing method and hardware architecture. Further, we evaluated the performances of the Virtex-4 series XC4VFX12-10SF363 as a target device and observed that the operation time of the 512-bit modular exponentiation is 0.261 ms. We believe that this is the fastest modular exponentiator available in FPGA. Further, the number of SLICEs used is approximately 4000 so that they can be formed even on the minimum logic FPGA in the Virtex-4 Series. In addition, 1024-, 1536-, and 2048-bit modular exponentiations can be processed in the same circuit.

As future studies, we enhance our modular exponentiator for the Virtex-5 and Spartan-3A Series, apply our Montgomery multiplier to elliptic curve cryptosystem, and evaluate modular exponentiation combined with a CPU integrated into an FPGA.

References

1. P. L. Montgomery,“Modular Multiplication without Trial Division,” Mathematics of Computation, Vol. 43, No. 170, pp. 519-521, 1985.
2. C. D. Walter, “Systolic Modular Multiplication,” IEEE Transactions on Computers, Vol. 42, No. 3, pp. 376-378, 1993.
3. S. E. Eldridge and C. D Walter “Hardware Implementation of Montgomery’s Modular Multiplication Algorithm,” IEEE Transactions on Computers, Vol. 42, No. 6, pp. 693-699, 1993.
4. H. Orup, “Simplifying Quotient Determination in High-Radix Modular Multiplication,” Proc. of the 12th IEEE Symposium on Computer Arithmetic, pp. 193-199, 1995.
5. T. Blum, and C. Paar, “Montgomery Modular Exponentiation on Reconfigurable Hardware, Proc. of the 14th IEEE Symposium on Computer Arithmetic, pp. 70-77, 1999.
6. C. D Walter, “Montgomery’s Multiplication Technique: How to Make It Smaller and Faster,” *CHES’99*, LNCS 1717, pp. 80-93, Springer-Verlag, 1999.
7. A. F. Tenca and Ç.K. Koç, “A Scalable Architecture for Montgomery Multiplication,” *CHES’99*, LNCS 1717, pp. 94-108, Springer-Verlag, 1999.

8. T. Blum and C. Paar, ‘High-Radix Montgomery Modular Exponentiation on Reconfigurable Hardware,” IEEE Transaction on Computers, Vol. 50, No. 7, pp. 759-764, 2001.
9. A. F. Tenca, G. Todorov, and Ç. K. Koç, “High-Radix Design of a Scalable Modular Multiplier,” *CHES 2001*, LNCS 2162, pp. 185-201, Springer-Verlag, 2001.
10. H. Nozaki, M. Motoyama, A. Shimbo, and S. Kawamura, “Implementation of RSA Algorithm Based on RNS Montgomery Multiplication,” *CHES 2001*, LNCS 2162, pp. 364-376, Springer-Verlag, 2001.
11. S. H. Tang, K. S. Tsui and P. H. W. Leong, “Modular Exponentiation using Parallel Multipliers,” Proc. of the 2003 IEEE International Conference on Field Programmable Technology (FPT 2003), pp. 52-59, 2003.
12. A. Satoh and K. Takano, “A Scalable Dual-Field Elliptic Curve Cryptographic Processor,” IEEE Transactions on Computers, Vol. 52, No. 4, pp.449-460, 2003.
13. C. McIvor, M. McLoone and J. V. McCanny, “FPGA Montgomery Multiplier Architectures - A Comparsion,” Proc. of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2004) pp. 279-282, 2004.
14. C. McIvor, M. McLoone and J. V. McCanny, “High-Radix Systolic Modular Multiplication on Reconfigurable Hardware,” Proc. of the 2005 IEEE International Conference on Field Programmable Technology (FPT 2005), pp. 13-18, 2005.
15. E. A. Michalski, D. A. Buell, “A Scalable Architecture for RSA Cryptography on Large FPGAs,” Proc. of the 16th IEEE International Conference on Field Programmable Logic and Applications (FPL 2006) pp. 145-152, 2006.
16. R. V. Kamala and M. B. Srinivas, “High-Throughput Montgomery Modular Multiplication,” Proc. of the 14th IFIP International Conference on Very Large Scale Integration (VLSI-SoC 2006), pp. 58-62, 2006.
17. K. Sakiyama, B. Preneel and I. Verbauwhede “A Fast Dual-Field Modular Arithmetic Logic Unit and Its Hardware Implementation,” Proc. of the 2006 IEEE International Symposium on Circuits and Systems (ISCAS 2006), pp. 787-790, 2006.
18. K. Sakiyama, E. De Mulder, B. Preneel and I. Verbauwhede “A Parallel Processing Hardware Architecture for Elliptic Curve Cryptosystems,” Proc. of the 2006 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2006), Vol. 3, pp. III-904-III-907, 2006.
19. The OpenCiphers Project, `http://opencipherssourceforge.net/oc/`, 2005.
20. D. E. Knuth, “The Art of Computer Programming, Volume 2, Seminumerical Algorithms, Third Edition,” Addison-Wesley, 1997.
21. Ç. K. Koç, “Analysis of Sliding Window Techniques for Exponentiation,” Computers and Mathematics with Applications, Vol. 30, No. 10, pp. 17-24, 1995.
22. Xilinx, “Virtex-4 User Guide UG070 (v1.6)”.
23. Xilinx, “XtremeDSP for Virtex-4 FPGAs User Guide UG073 (v2.3)”.
24. Xilinx, “Virtex-4 Data Sheet: DC and Switching Characteristics DS302 (v2.0)”.
25. Xilinx, “Alpha Blending Two Data Streams Using a DSP48 DDR Technique XAPP706 (v1.0)”.