How to Maximize the Potential of FPGA-Based DSPs for Modular Exponentiation*

Daisuke SUZUKI†, †† and Tsutomu MATSUMOTO†††, Members

SUMMARY 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 one is to improve the Montgomery multiplication algorithm in order to maximize the performance of the multiplication unit in an FPGA. The second one is to balance and improve the circuit delay. The third one is to ensure scalability of the circuit. Our architecture can perform fast operations using small-scale resources; in particular, it can complete a 512-bit modular exponentiation as fast as in 0.26 ms with the smallest Virtex-4 FPGA, XC4VF12-10SF363. In fact the number of SLICEs used is approx. 4200, which proves the compactness of our design. Moreover, the scalability of our design also allows 1024-, 1536-, and 2048-bit modular exponentiations to be processed in the same circuit.

key words: hardware architecture, modular exponentiation, Montgomery multiplication, FPGA, DSP

1. Introduction

In the research field on cryptographic implementation, fast implementation of public-key cryptography is most intensely worked on and there have long been many research works on it, both on software and hardware platforms [2]–[25]. Examples of the software implementation include implementation evaluation of public-key cryptography on widely-used general purpose CPUs by eBATS [23], and public-key implementation using graphics processing units (GPUs) as a fairly new topic [24], [25]. There also are diverse studies as to the arithmetic of public-key cryptography: modular exponentiation of multiple-length integers typical for RSA and DSA, point multiplication on elliptic curves, and pairing arithmetic, which is a fairly recent topic.

This paper deals with the implementation of modular exponentiation on FPGA for the following motivations. In the first place, we aim for embedded systems such as VPN router or tamper-resistant board [27] that need cryptographic operations. An increasing number of embedded devices are connected to networks in recent years, which gives rise to cases where public-key operations or key management such as in IPsec and SSL are processed within the devices. Microprocessors used in such devices, however, have lower performance than the CPUs used in PCs and servers, and thus some kinds of acceleration are needed. As means of acceleration, it is possible to use high-end CPUs or GPUs to run high-speed software based on the studies [23] and [24]. However, high-end CPUs and GPUs are generally not suitable for embedded use because of the large number of component parts, the power supply and the heat dissipation needed to make use of them, before considering per-chip prices. In fact, inside chassis of recent PCs, we can find motherboards and graphics cards with a huge heat sink and fan, and a high capacity power supply unit. As a more concrete example, Ref. [26] shows that high-end GPUs and CPUs require one digit higher power consumption than FPGAs and ASICs.

In general, high-speed processing in embedded systems is realized using hardware platforms such as ASIC and FPGA to integrate the functions and communication interfaces into system-on-a-chip (SoC). In particular FPGAs offer a cost-effective and high-performance hardware alternative to ASICs in low- and mid-volume products. These days embedded systems are often shipped with FPGAs installed regardless of whether cryptographic operations will actually be implemented, which makes easy the implementation of cryptographic operations by FPGA on existing platforms.

The second reason is that we need to work on algorithms that are for the present moment most demanded in the market and thus implemented most often. Many of the recent implementation studies such as [21] and [25] specialize in processing elliptic curve cryptography (ECC). ECC is able to be made high-speed compared to multiple-length modular exponentiation that involves 1024 to 4096 bits of data, needed for conventional public-key cryptography like RSA and DSA. It also does not require very heavy operations for key generation such as prime number generation. Therefore, ECC is very efficient for the implementation of whole security systems. On the other hand, many standards that have long adopted conventional cryptographic algorithms still require support for them, as can be told from the specifications of IPsec and SSL. It is then highly likely that they will not substitute ECC for the conventional ones in the next few years but allow parallel use of them. In those standards, the priority is to make high-speed RSA, DSA or prime number generation, all based on modular exponentiation, rather than to optimize ECC simply because it is easier.

Those two reasons lead us to weigh heavily the avail-
ability in industry. This paper discusses the efficient implementation architecture for modular exponentiation on FPGA, which is suitable for embedded devices, which we focus on. From here, we first review the structure of FPGA and its progress, and next some studies on the hardware implementation of public-key cryptography. The emphasis is on modular exponentiation or its fast implementation, and Montgomery multiplication useful for it.

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 a RAM-based lookup table (LUT) and flip-flop (FF), to construct user logic. In 2004, Xilinx (an FPGA vendor) introduced the Virtex-4 Series FPGAs. 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 “digital signal processing (DSP) function”).

Recent works include implementation methods of public-key cryptography that make use of DSP. The work of [20] is an implementation method of a basic Montgomery multiplication that uses a single DSP function. [21] proposes an ECC co-processor that uses DSP functions, which is dedicated to the elliptic curve over the prime field defined by NIST. It is, however, difficult to expand the architecture of [21] to fit our target modular multiplication because its high-speed processor of modular operations aims for sparse primes.

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 one 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 one 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 one 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 resource in the Virtex-4 Series FPGAs. Moreover, the scalability of our design also allows 1024-, 1536-, and 2048-bit modular exponentiations to be processed in the same circuit.

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 those issues that are relevant with regard to this paper. For more information, refer to [30]–[32].

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 RAMs (henceforth called “BRAMs”), a hardware macro group called XtremeDSP (henceforth called “DSP48”) to provide the above-mentioned DSP function, and a configurable logic blocks (CLBs) that serve as basic blocks for the implementation of user logic [30], [31]. The schematic representation of the CLB’s internal configuration is shown at the bottom of Fig. 1. The CLB comprises four blocks called SLICEs. 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 operation mode for the 16×1-bit (maximum) single-port memory with the LUT function (henceforth called “distributed RAM”) or a 16×1-bit (maximum) variable shift register (henceforth called “SRL16”).

Figure 2 shows the schematic representation of the internal configuration of DSP48. DSP48 is designed to support dynamic changes in a 42-pattern multiplicative summation by switching the control signals (OPMODE) [31]. Controlling the light-colored MUXs in Fig. 2 during the FPGA device configuration allows us to change the latency of the signal conductors. The maximum operating frequency of

![Fig. 1 Internal configuration of Virtex-4.](image-url)
DSP48 depends on the speed grade of the FPGA and the latency set above, and the operation is valid at the maximum frequency of 400 MHz in the lowest speed grade (−10) [32]. 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. There are generally several circuit architectures which perform a specific processing operation. It is important to determine which one is advantageous to form the circuit in the FPGA. It also is important to check if the examined architecture is actually within the 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 (4.41 ns) or less according to [32]. Let us note that the third circuit is a combinational one, and no FF setup, FF hold nor wiring delay time is taken into account, which is why the figure is “226 MHz or less” instead of just “226 MHz.”

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

Next, we describe the performance of the addition 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, listed in Table 1, it is assumed that the largest adder that is operable at the maximum operating frequency of 400 MHz is 8-bit 2-input addition. Another interpretation of the results shown in Table 1 is that 32-bit 2-input addition is operable at approximately 250 MHz.

Based on the above descriptions, a 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 our modular exponentiator by using DSP functions.

3.1 Design Policy

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

(1) To allow DSP48 to operate at the 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-bit and 1024-bit to be processed using the same circuits for Montgomery multiplication.
(4) To set the bus width of the input/output signals to less

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

<table>
<thead>
<tr>
<th>Functions of adder</th>
<th>No. of LUTs used</th>
<th>Circuit delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit 2-input addition</td>
<td>8 LUTs</td>
<td>2.201 ns</td>
</tr>
<tr>
<td>16-bit 2-input addition</td>
<td>16 LUTs</td>
<td>2.734 ns</td>
</tr>
<tr>
<td>32-bit 2-input addition</td>
<td>32 LUTs</td>
<td>3.564 ns</td>
</tr>
<tr>
<td>8-bit 3-input addition</td>
<td>14 LUTs</td>
<td>4.044 ns</td>
</tr>
<tr>
<td>16-bit 3-input addition</td>
<td>29 LUTs</td>
<td>4.407 ns</td>
</tr>
<tr>
<td>32-bit 3-input addition</td>
<td>65 LUTs</td>
<td>5.188 ns</td>
</tr>
</tbody>
</table>

†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.
than 36 bits in order to simplify the control of the operation results.

(5) To implement the circuits even on the smallest device of Virtex-4 Series.

Items (1) and (2) together mean to fully exploit the maximum performance of DSP48. Only raising the frequency does not necessarily improve throughput if there occur stalls as a result of doing that. That is why the zero stalling condition is required as well as the maximum frequency condition.

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 FPGA circuits have a reconfiguration time of some milliseconds; therefore, this reconfiguration cannot be carried out on many operating systems. 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 from Montgomery multiplications are controlled within the FPGA, an efficient 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 and must be placed in rows to store large data in a parallel manner. On the contrary, data can be stored in up to 512 addresses (depth) per BRAM when the width is 36 bits. Therefore, the BRAM characteristics can be applied when the operation results are controlled as stream data in the direction of depth with narrow bus widths. Further, the performance of a circuit having a large bus width may always be reduced due to its location and wiring. The above viewpoints pertain to Item (4). With regard to Item (5), we believe that it is not necessary to use a large-scale FPGA and most of its resources only for cipher operations. On the other hand, it is difficult to quantitatively indicate what 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, 5472; the number of DSP48s, 32; and the number of BRAMs, 36.

3.2 Processing Method

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

Montgomery Multiplication

For 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 which are described in [4], [5]. Algorithm 1 shown below explains the Montgomery multiplication algorithm, as specified in [5].

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 Radix-2 Montgomery Multiplication (MWR2MM) [8].

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 cycles before setting $q_{i-1}$; the smaller the value of the delay parameter is, the less is the number of cycles required for the total Montgomery multiplication. In Algorithm 2, it is assumed that $\alpha$ 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 $n = [h'/k]$. Note that the bit length of one word is $k = 17$. Further, the number of words $r$ processed by one DSP48 is defined as $r = 2([n/\alpha])2/2$. This implies that one DSP48 is applied to process only $r$ words from a total number of $n$ words. Note that the number of words $r$ is set to an even number. The number of words processed by $\alpha$ DSP48s is $\alpha r$, which is equal to or greater than $n$. If the excess words, those other than the $n$ valid ones, are zero padded before processed. 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 $b_A$ in Algorithm 1 is first calculated using DSP48 (MUL_\text{AB}). This operation requires $n$ multiplications. Here, it is assumed that one DSP48 performs $r$ multiplications, following which another DSP48 receives a carry to continue the subsequent multiplications. DSP48, which provides a carry, begins performing the multiple-length multiplication (MUL_\text{MQ}) corresponding to $q_{i-1}M''$ in the next step of Al-
Algorithm 2 Montgomery Multiplication for Virtex-4

**Setting:** radix: \(2^8(=2^{11})\), delay parameter: \(d(=1)\), no. of DSP48s: \(a(=17), 2 < M < 2^8\) (in [512, 1024, 1536, 2048]), \(gcd(M, 2) = 1\), \((-MM') mod 2^{2(n+d+1)} = 1\), \(M = (M' mod 2^{2(n+d+1)})M\) if \(A, B \leq 2M\), \(h' = h + k\) \(d+1\) + 1, no. of words at \(A\) and \(B\): \(n = \lfloor h' / k \rfloor\); \(R = 2^{2n} > 4M\), no. of words processed by one DSP48: \(r = 22\lfloor (n/\alpha)/2 \rfloor + (r \in \{2, 4, 6, 8\})\). \(A = \sum_{i=0}^{2n}a_i\), \(B = \sum_{i=0}^{2n}b_i\), \(a_j, b_j, m_j, s_j, 3\) \(\in\{0, 1, \cdots, 2^i - 1\}\), \(a_j = b_j = 0\) for \(j \geq n\) and \(m_j = 0\) for \(j \geq \lfloor h/\alpha \rfloor\).

**Input:** \(A, B, M'\)

**Output:** \(\text{MMA}(A, B) = S_{n+3} \equiv ABR^{-1} \mod M, 0 \leq S_{n+3} < 2M\)

1: \(S_0 := 0, q_i := 0;\)
2: for \(i = 0\) to \(n + 1\) do
3: carry := \(17\) \(b_0, c := 1\) \(b_0; cs := 1\) \(b_0;\)
/* Multiple-length multiplication: MUL\_AB*/
4: for \(j = 0\) to \(ar - 1\) do
5: carry \(p_j := b_o a_j + carry;\)
6: end for
/* Multiple-length multiplication: MUL\_MQ*/
7: for \(j = 0\) to \(ar - 1\) do
8: if \(j = 0\) then
9: carry \(|i| : \equiv q_i m_j + p_i;\)
10: else
11: carry \(|j| : \equiv q_i a_j + carry;\)
12: end if
13: end for
/* Calculation \(q_i\): ADD\_V0S1*/
14: \(q_{i+1} := t_0 + s_{i+1};\)
/* Multiple-length addition: ADD\_PU*/
15: for \(j = 0\) to \(ar/2 - 1\) do
16: if \(j = 0\) then
17: \(\text{cv}(|i|, |i|) := (p_j || 17b_0) + (a_j || v_0);\)
18: else
19: \(\text{cv}(|j|, |j|) := (p_j || 17b_0) + (a_j || v_0) + (u_2j+1 || u_2j) + cv;\)
20: end if
21: end for
/* Multiple-length addition: ADD\_VS*/
22: for \(j = 0\) to \(ar/2 - 1\) do
23: \(\text{cs}(|j+1|, |j+1|) := (u_2j+1 || v_0) + (u_2j+1 || u_2j+1) + cs;\)
24: end for
25: end for
26: \(S_{n+3} := S_{n+2} || a_{n+1, 0};\)
27: return \(S_{n+3};\)

In the same 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 \(a\) 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 assumed that one loop of each addition completes 2 words (34 bits) to have the number of loops \(ar/2\) 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, DSP48 carries out “single word multiplication” at the maximum operating frequency and the adder with the LUT performs “double word addition” at half the maximum operating frequency, thus maintaining the total throughput. This operation is henceforth called “SMDA.” The advantage of SMDA is that the user logic can 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, where it divides 3-operand multiple-length additions into two stages and thereby performs the additions virtually in one cycle without 3-input adders.

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 set. 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 of a carry, this operation can be performed only by modifying the operation mode of DSP48. Next, \(v_0\) is set at the output of MUL\_MQ. Therefore, the operation required to set \(q_{i+1}\) is an addition with \(s_{i+1}\), such that \(q_{i+1}\) is set with a smaller latency than that for a calculation of \(v_0\) in MUL\_MQ. The latency required to set \(q_{i+1}\) affects the delay parameter \(d\) in Algorithm 2. The Montgomery multiplication circuits described in the following section are operand with \(d = 1\).

**Sliding-window Exponentiation**

The sliding window [29] is one of the fastest 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, hardware modular exponentiation is often carried out using a binary method [28]. However, since the Virtex-4 Series have several large-capacity memory blocks as hardware macros, we attempted to form the modular exponentiator with a 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 \(X_{2j+1}\) from Algorithm 2 is \(2w^2 \times n \times k\) bits.

The modular exponentiation circuit explained in this paper was configured with the window size set to \(w = 5\). This is because we process modular exponentiation with the modulus being up to 2048-bit without needing extra memory. In addition, maximum processing time is shortest in 512-bit modular exponentiation, which is required in 1024-bit RSA operation with Chinese Remainder Theorem (CRT). In an application that attaches more importance to speed than efficiency, we should decide the window size that is most suitable for every size of modulus. The Montgomery multiplication circuits described in this paper are designed
Algorithm 3 Modular exponentiation with sliding-window exponentiation [29]

Setting: radix : \(2^d\); delay parameter: \(d\); no. of blocks: \(n\); modulus: \(M, M > 2, \text{gcd}(M, 2) = 1\), \((-MM' \mod 2^{d(i+1)}) = 1\), \(M = (M' \mod 2^{d(i+1)})M, 4M < 2^{2m} = R, M'' = (M + 1)/2^{d(i)}\).

Input: \(M'', X, R, R' = R^2 \mod M, E = (e_0, \ldots, e_1, e_0)\).

Output: \(Y = X^E \mod M, 0 \leq Y \leq 2^m\).

1: \(X_1 := \text{MM}(X, R), C_R := \text{MM}(1, R); X_2 := \text{MM}(X_1, X_1);
2: \text{for } i = 1 \text{ to } 2^{m-1} \text{ do}
3: \text{end for}
4: \text{end for}
5: \(S_R := C_R;
6: i := i - 1;
7: \text{while } i \geq 0 \text{ do}
8: \text{if } e_i = 0 \text{ then}
9: \(S_R := \text{MM}(S_R, S_R); i := i - 1;
10: \text{else}
11: \text{Searching maximum odd-number binary digit string } (e_i, e_{i-1}, \ldots, e_2) \text{ within window size, } i - l + 1 \leq w
12: \text{for } j = 0 \text{ to } l - 1 \text{ do}
13: \(S_X := \text{MM}(S_R, S_R);
14: \text{end for}
15: \(S_R := \text{MM}(X_{(i+1)}, X_i); l := l - 1;
16: \text{end if}
17: \text{end while}
18: Y := \text{MM}(1, S_R);
19: \text{return } Y;

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 shown in Fig. 4. In addition, Fig. 5 shows relation of the pipeline processing in Algorithm 2 and Fig. 4. Data \(A\) and \(M''\) are inputted from the left, 34 bits (two words) at a time, and are stored into the specified DMEMs. \(M''\) is pre-calculated according to the modulus \(M\), and stored only once when executing modular exponentiation. Therefore, only \(A\) 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 \((\text{depth}) \times 34 \text{ (bit width)}\). In this case, the capacity of DMEM can correspond to the modulus size of up to 2048-bit. When \(a_i(0 \leq j \leq r - 1)\) 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 \(a_r\) multiplications in MUL\_AB and MUL\_MQ. This operation is performed by switching the OPMODE signal, which appears in Fig. 2, and Table 2 shows the sequence of the \(r\) multiplications and their corresponding OPMODE values. Note that our circuit utilizes three patterns of OPMODE: OPMODE = \(7'h35\) for multiplicative summation whose addend comes from port C in Fig. 2, OPMODE = \(7'h65\) for multiplicative summation whose addend is the carry of the multiplication within the same DSP48, and OPMODE = \(7'h55\) for multiplicative summation whose addend is the carry of the multiplication of the DSP48 to the left. The leftmost DSP48 uses two patterns out of the three.

The second DSP48 from the left performs the next \(r\) multiplications in the same manner switching the latter two patterns of multiplicative summation, shown also in Table 2. The third and subsequent 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 adders and LA1s (latency adjuster), shown at the center of Fig. 4. Two-step positive/negative FFs are placed on the left path of the circuits and a one-step negative FF is placed on the right path. This is because latencies of higher and lower words are different and it is necessary to adjust them to be the same. This state allows two words as the result of the MUL\_AB operation transmitted from the DSP48 to be entered simultaneously into the adders with the negative clock (clk1x). Currently, the result of the MUL\_AB operation is directly stored into the LA1s by resetting the LA1 output values to 0. Next, the result of the MUL\_MQ operation is used to perform addition with the result of the MUL\_AB operation that has been pre-stored in the left LA1s. The difference in the input time between the results of MUL\_AB and MUL\_MQ operations is \(r/2\) cycles 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, different carry FFs are placed for each case in order to improve the circuit delay.

The bottom-most 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 additions of 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 with the synchronous reset function until \(S_{n+3}\) is entered completely. This will be explained later.

In Fig. 4, LA1 and LA2 are shift registers whose latency is changeable from 1 to 4 and from 2 to 5, respectively. Further, they support a 0 resetting function. These units comprise variable-length shift registers based on SRL16. In this case, they 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 as shown in Fig. 6. 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.”

The ADD_V0S1 operation is performed in the upper left circuit shown in Fig. 4. This circuit has an FF of clock clk2x at the input port; however, the addition is performed according to the standard of clk1x.\(^\dagger\) 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 \(q_{j+1}\) latency and load signal to the DSP48 in the proper timing.

**Modular Exponentiator**

Figure 7 shows an overview of our modular exponentiator using Fig. 4. The modular exponentiator comprises the following components:

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

\(^\dagger\)This is the multi-cycle path for the FF output data with the clock “clk2x.”
(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_MEM in Item (c) can correspond to the modulus size of up to 2048-bit even if Algorithm 3 is processed with \( w = 5 \).

The output signal of MM_ENGINE is 578 bits; however, only 34 bits of them are valid at a time, representing a 34-bit part of \( S_{n+3} \), and all the rest 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 in B_MEM and A_MEM or X_MEM. When more than 34 bits of data are updated in A_MEM or X_MEM, MEX_CTL starts to read and transmit the data, which is required for DMEM of the Montgomery multiplication circuits. The number of cycles required from 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 modulus size supported, the modular exponentiator shown in Fig. 7 is designed with the win-

Table 2  Multiplication sequence of DSP48.

<table>
<thead>
<tr>
<th>Count</th>
<th>Operation from leftmost DSP48</th>
<th>OPMODE</th>
<th>Remarks</th>
<th>Operation from 2nd DSP48</th>
<th>OPMODE</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( b_{0}a_{0} )</td>
<td>7'h35</td>
<td>Reset C</td>
<td>( q_{i-2}m_{2} + \text{carry} )</td>
<td>7'h55</td>
<td>Carry is received from leftmost DSP48</td>
</tr>
<tr>
<td>1</td>
<td>( b_{1}a_{1} + \text{carry} )</td>
<td>7'h65</td>
<td>-</td>
<td>( q_{i-2}m_{3} + \text{carry} )</td>
<td>7'h65</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>( q_{i-1}m_{0} + p_{0} )</td>
<td>7'h35</td>
<td>( p_{0} ) is stored into ( C )</td>
<td>( b_{i}a_{i} + \text{carry} )</td>
<td>7'h55</td>
<td>Carry is received from leftmost DSP48</td>
</tr>
<tr>
<td>3</td>
<td>( q_{i-1}m_{1} + \text{carry} )</td>
<td>7'h65</td>
<td>-</td>
<td>( b_{i}a_{i} + \text{carry} )</td>
<td>7'h65</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>( b_{i+1}a_{i} )</td>
<td>7'h35</td>
<td>Reset C</td>
<td>( q_{i-1}m_{2} + \text{carry} )</td>
<td>7'h55</td>
<td>Carry is received from leftmost DSP48</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>2048-bit mode ( (r = 8) )</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Fig. 7  Overview of our modular exponentiator.

(b) A_MEM, 2-port BRAM \((1024 \times 17) \times 2\), template memory;
(c) B_MEM, 2-port BRAM \((512 \times 34)\), template memory;
(d) X_MEM, 2-port BRAM \((1024 \times 17) \times 2\), \( X \) storage memory;
(e) E_MEM, 1-port BRAM \((2048 \times 5)\), exponent encode result storage memory;
(f) S_TRANS, circuits to convert the output signal of the Montgomery multiplication circuit into 34-bit stream data;
dow size \( w = 5 \) in Algorithm 3. 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 3.

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

4. Performance 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 carried out with Simplify Pro C-2009.06 and ISE 10.1.03 respectively.

The critical path of clk2x (400 MHz) is the selective signal of the MUX that is to be connected to the input ports A and B of the DSP48 shown in Fig. 2. 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 to be met than those of conventional logic. Further, the large fan-out of the selective signal causes a significantly increase in the circuit delay. Reference [33] 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 Sect. 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, which has the fewest 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 4200, 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.

I/O transfer, pre-calculation of \( M' \) and normalization of \( Y \) and \( R_8 \) in Algorithm 3 are not included in the operation time in Table 3. We assume that these processes are executed by a CPU inside or outside the FPGA. The connection example of a CPU inside the FPGA and our modular exponentiator targeting the Virtex-4 FX series is shown in Fig. 8. The BRAM controller in Fig. 8 is provided as free core by Xilinx, and CPU and our circuit can be connected easily via the controller.

We now compare our circuit designs with previously reported ones. The purpose of this comparison is not to discuss the advantages and disadvantages of the circuit process-
Table 4  Comparison with previous implementations.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Target device</th>
<th>Process</th>
<th>Additional FPGA function</th>
<th>Scalability</th>
<th>512-bit MEX time</th>
<th>512-bit MEX area</th>
<th>1024-bit MEX time</th>
<th>1024-bit MEX area</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[9]</td>
<td>[12]</td>
<td>This work</td>
<td></td>
<td>(Max.) 2.93 ms</td>
<td>3413 CLBs</td>
<td>(Max.) 11.95 ms</td>
<td>6633 CLBs</td>
</tr>
<tr>
<td></td>
<td>XC40250XV</td>
<td>XC2V3000-6</td>
<td>XC4VFX12-10</td>
<td>N</td>
<td>N</td>
<td>18235 SLICEs,</td>
<td>N</td>
<td>14334 SLICEs,</td>
</tr>
<tr>
<td></td>
<td>Process</td>
<td>0.35 μm</td>
<td>Basic function</td>
<td>N</td>
<td>(Avr.) 0.59 ms</td>
<td>8235 SLICEs,</td>
<td>(Avr.) 2.33 ms</td>
<td>4190 SLICEs,</td>
</tr>
<tr>
<td></td>
<td>Additional</td>
<td>0.12/0.15 μm</td>
<td>18x18 multiplier</td>
<td>Y</td>
<td>(Max.) 0.261 ms</td>
<td>32 Multipliers</td>
<td>(Max.) 1.71 ms</td>
<td>17 DSP48s</td>
</tr>
<tr>
<td></td>
<td>FPGA function</td>
<td>Distributed RAM</td>
<td></td>
<td></td>
<td>18Kbit BRAM,</td>
<td>18Kbit BRAM,</td>
<td>512-bit MEX time</td>
<td>512-bit MEX area</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18x18 multiplier</td>
<td>DSP48</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5  Performance comparison between devices with DSP functions.

<table>
<thead>
<tr>
<th>DSP Type</th>
<th>DSP48</th>
<th>DSP48E</th>
<th>DSP48A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XC4VFX12-10</td>
<td>3237</td>
<td>4219</td>
<td>450 MHz</td>
</tr>
<tr>
<td>XC5VLX30T-1</td>
<td>7</td>
<td>5</td>
<td>250 MHz</td>
</tr>
<tr>
<td>XC3SD1800A-4</td>
<td>17</td>
<td>17</td>
<td>400 MHz</td>
</tr>
<tr>
<td>No. of SLCIEs used</td>
<td>4190</td>
<td>3237</td>
<td>250 MHz</td>
</tr>
<tr>
<td>No. of BRAMs used</td>
<td>5</td>
<td>5</td>
<td>520.8 MHz</td>
</tr>
<tr>
<td>No. of DSP48s used</td>
<td>7</td>
<td>7</td>
<td>0.535 ms</td>
</tr>
</tbody>
</table>

Table 6  Comparison: FPGA vs. ASIC, hardware vs. software, and elliptic curve vs. modular exponentiation.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Device</th>
<th>Logic</th>
<th>Clock</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This work: 1024-bit-MEX</td>
<td>XC5VLX30T-1</td>
<td>3237 SLICEs/7 DSPs/5 BRAMs</td>
<td>450 MHz</td>
<td>1.52 ms</td>
</tr>
<tr>
<td></td>
<td>XC3SD1800A-4</td>
<td>4219 SLICEs/7 DSPs/7 BRAMs</td>
<td>250 MHz</td>
<td>2.74 ms</td>
</tr>
<tr>
<td>1024-bit-MEX [22]</td>
<td>90-nm CMOS</td>
<td>74.8 Kgate</td>
<td>520.8 MHz</td>
<td>1.17 ms</td>
</tr>
<tr>
<td>GF(p), NIST-224 PointMul [21]</td>
<td>XC4VFX12-12</td>
<td>1580 SLICEs/26 DSPs</td>
<td>487 MHz</td>
<td>0.365 ms</td>
</tr>
<tr>
<td>Software</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1024-bit-RSA (with CRT) [23]</td>
<td>Intel Core2 Duo</td>
<td>-</td>
<td>2.13 GHz</td>
<td>1.38 ms</td>
</tr>
<tr>
<td>GF(p), NIST-224 PointMul [23]</td>
<td>Intel Core2 Duo</td>
<td>-</td>
<td>2.13 GHz</td>
<td>0.555 ms</td>
</tr>
<tr>
<td>1024-bit-MEX [24]</td>
<td>NVIDIA 8800GTS</td>
<td>-</td>
<td>1.20 GHz</td>
<td>1.25 ms</td>
</tr>
<tr>
<td>GF(p), 280-bit PointMul [25]</td>
<td>NVIDIA 8800GTS</td>
<td>-</td>
<td>1.20 GHz</td>
<td>0.708 ms</td>
</tr>
<tr>
<td></td>
<td>NVIDIA 8800GTS</td>
<td>-</td>
<td>1.20 GHz</td>
<td>0.414 ms</td>
</tr>
</tbody>
</table>

cause the expanded DSPs are compatible with DSP48, it is possible to implement the proposed circuit on Virtex-5 and Spartan-3A DSP with minor modifications to the code. We thus make necessary changes (such as port names) to the circuit in Sect. 4, whose performance we evaluate there, to fit each new device and evaluate the performances. The result is shown in the Table 5.

The performance gain from Virtex-4 to Virtex-5 is about 10%, which is due to the fact that the maximum operating frequency of DSP48E at the lowest speed grade is 450 MHz. The performance loss from Virtex-4 to Spartan-3A DSP is about 50%. However, the per-chip price of XC3SD1800A is about US$ 30 [40], which is about 1/3 that of XC5VLX30T, US$109 [41] and thus means high cost performance. Our modular exponentiator takes about 20% of the total resource of XC3SD1800A: there still remains large resource for symmetric-key cryptography, soft-macro CPUs, bus controllers and sequencers of other circuits, which enables us to develop a variety of purpose-specific SoCs. We thus conclude that the most practical platform of the three for embedded systems is XC3SD1800A.

Next, we show in Table 6 performance comparison, which now includes devices other than FPGA, and figures of elliptic curve cryptography. Note that simple comparisons of the figures are meaningless because different platforms have different range of applicability or usage. Nonetheless, we still find the comparisons interesting in that we can see the tendency of each platform.
The ASIC implementation in [22] structures a modular exponentiation circuit based on 64-bit multiply-accumulators that fit the target CMOS process. The circuit, with the size of 74.8Kgate, can process a 1024-bit modular exponentiation in about 1 ms. Comparing the sizes of the multiply-accumulators, it is about the same size as ours, which uses 17 units of 18-bit multiply-accumulators (DSP48E). The emphasis must be put on the point that semi-custom design such as in ASIC allows us to decide the size of operators or latency freely, while in FPGA, if taking the same design method as ASIC, the delay of inter-LUT connections becomes dominant and hence the total performance greatly decreases compared to that in ASIC with the same design. Our architecture realizes the same level of performance as ASIC on FPGA, as is shown by the final performance.

Also, in the comparison with the software of [23] and [24], our architecture achieves the same performance. The same can be said about the cost performance because the market prices of Core2 Duo and 8800GTS are around US$100. We consider it significant that the same level of performance can be realized in embedded environments.

Table 6 implies that elliptic curve-based cryptography is more efficient in implementation than modular exponentation-based cryptography, regardless of the platform. The difference would probably become even larger as the key size increases. However, also highly worthwhile is RSA with up to 4096-bit keys realized on embedded platforms retaining practical cost and performance. Note that we assume that CRT is implemented on an embedded processor such as the PowerPC in Fig. 8, thereby realizing 4096-bit RSA with our 2048-bit modular exponentiator. That is the reason we do not implement 4096-bit modular exponentiation although it is estimated to be possible.

6. 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 4200 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.

References

[23] ECRYPT, eBATS: ECRYPT benchmarking of asymmetric systems,
Daisuke Suzuki received the B.E., and M.E. degrees from Tokyo University of Science, Japan, in 1999 and 2001, respectively. In 2001 he joined Mitsubishi Electric Corporation. His research interests include hardware implementation of cryptosystem. He was awarded the SCIS 2005 paper prize.

Tsutomu Matsumoto is a Professor and the head of Division of Social Environment and Informatics, Graduate School of Environment and Information Sciences, Yokohama National University. His current roles include an Associate Member of the Science Council of Japan, an Advisor of Research Center for Information Security, National Institute of Advanced Industrial Science and Technology, a Director of International Association for Cryptologic Research and a core member of CRYPTREC — the Cryptography Research and Evaluation Committees for governmental use of cryptographic technology. Starting from design and analysis of various cryptosystems and cryptographic protocols in the early 80's, he has opened up the field of security measuring for logical and physical security mechanisms including human-machine cryptography, information hiding, software security, biometric security, side-channel security, and artifact-metrics. He got Doctor of Engineering degree from the University of Tokyo in 1986. He received the Achievement Award from IEICE in 1995, DoCoMo Mobile Science Award in 2006, and the Culture of Security Award in 2008.

[31] Xilinx, “XtremeDSP for Virtex-4 FPGAs User Guide UG073 (v2.3).”
[32] Xilinx, “Virtex-4 Data Sheet: DC and Switching Characteristics DS302 (v2.0).”
[33] Xilinx, “Alpha Blending Two Data Streams Using a DSP48 DDR Technique XAPP706 (v1.0).”
[34] Xilinx, “Virtex-5 User Guide UG190 (v5.0).”
[36] Xilinx, “Virtex-5 FPGA Data Sheet: DC and Switching Characteristics DS202 (v5.2).”
[37] Xilinx, “Spartan-3 Generation FPGA User Guide UG331 (v1.5).”
[38] Xilinx, “XtremeDSP DSP48A for Spartan-3A DSP FPGAs User Guide UG431 (v1.3).”
[39] Xilinx, “Spartan-3A DSP FPGA Family: DC and Switching Characteristics DS610-3 (v2.2).”