# A Multiple Clock Domain Design of High-radix Montgomery Multiplication for Simplicity

Naoki Fujieda, Yusuke Ayuzawa, Masato Hongo, and Shuichi Ichikawa
Department of Electrical and Electronic Information Engineering,
Toyohashi University of Technology
fujieda@ee.tut.ac.jp (Naoki Fujieda),
ichikawa@ieee.org (Shuichi Ichikawa)

***Abstract*—This paper presents a case for intra-module multiple clock domain design in FPGAs using a high-radix Montgomery multiplier. Our design aims at simple hardware description from algorithm description, avoiding the decrease of clock rate by a long combinatorial path. According to our evaluation with 1024-bit modular exponentiation units, the calculation time reduced by 1.0% on average. The additional logic units for the proposed design was 2,556 LUTs and 1,043 flip-flops per multiplier at a maximum.**

## I. Introduction

Digital circuits often adopt multiple clock domain designs [3]. It is typically required by interfaces to external input/output devices (e.g. video signals) driven by different clocks. The clock frequency of a memory controller can be multiples of that of a computation circuit to leverage the high bandwidth of the external memory [2]. It is also used for reduction of power consumption because of the reduced number of circuit switching. There, voltage scaling is sometimes adopted in addition to frequency scaling [5]. More aggressively, in GALS (Globally Asynchronous, Locally Synchronous) designs [11], each circuit block is driven by a single clock that can be different from others. These blocks are connected by an asynchronous interconnect. One of the goals of multiple clock domain designs is to reduce the effect of long combinatorial paths. A path between registers with the longest combinatorial delay is called a critical path. When the whole circuit is driven by a single clock, a part of the circuit with only short paths might be slowed down by another part with long paths. This can be prevented by driving the short-path part by a faster clock. However, most of such designs suppose that multiple clocks are only used among modules. The use of multiple clocks within a module is considered as not recommended because of the problem of metastability and the difficulty of timing analysis [3].

In the early steps of computation circuit designs, a simple hardware description, which writes down its algorithm with a minimum modification, is useful. In addition to the ease of verification of itself, it can be used for the verification of an optimized circuit. It should be noted that the critical path might not reside in a part to be optimized. In this case, the effect of the optimization might not be correctly evaluated. Although it is a classic solution to first improve the part with the critical path, it is worth considering to leave it untouched for the present and drive it by a slower clock.

This study examines the use of multiple clocks within a module in FPGAs using high-radix Montgomery multiplication [4], [8]. It is often used for fast modular exponentiation, which is the basis of the RSA public-key cryptosystem [6]. A number of FPGA implementations of high-radix Montgomery multipliers have been proposed [1], [7], [9], [10]. Rather than such optimized circuits, we use a simple circuit written down from the algorithm. A timing problem between clocks is solved by generating phase-aligned clocks by a DCM (Digital Clock Manager) module of FPGA [3]. We evaluate the amount of hardware and the computation time of 1024-bit modular exponentiation circuits using the designed Montgomery multipliers.

## II. High-radix Montgomery Multiplication

### A. Brief Description of Algorithm

Montgomery multiplication [4] is an algorithm to efficiently compute modular multiplication. Its basic strategy is to transform a modulo operation by an odd number $p$ to a division and a modulo by $2^n$, which are easily performed by bit shift

1: $C \leftarrow 0$
2: **for** $i$ = 0 to $m-1$ **do**
3: $z \leftarrow 0$
4: $t_i \leftarrow (c_0 + a_i b_0) q \bmod 2^r$
5: **for** $j$ = 0 to $m-1$ **do**
6: $S \leftarrow c_j + a_i b_j + t_i p_j + z$
7: **if** $j \neq 0$ **then**
8: $c_{j-1} \leftarrow S \bmod 2^r$
9: **end if**
10: $z \leftarrow S/2^r$
11: **end for**
12: $c_{m-1} \leftarrow z$
13: **end for**
14: **if** $C > p$ **then**
15: $C \leftarrow C - p$
16: **end if**

Fig. 1. Algorithm of high-radix Montgomery multiplication [8], where the output $C$ is calculated by $C = \text{MM}(A, B, p, q) = AB2^{-n} \bmod p$.

1: $Z \leftarrow \text{MM}(1, 2^{2n}, p, q)$
2: $Y \leftarrow \text{MM}(X, 2^{2n}, p, q)$
3: **for** $i = k-1$ downto 0 **do**
4: $Z \leftarrow \text{MM}(Z, Z, p, q)$
5: **if** $e_i = 1$ **then**
6: $Z \leftarrow \text{MM}(Z, Y, p, q)$
7: **end if**
8: **end for**
9: $Z \leftarrow \text{MM}(Z, 1, p, q)$

Fig. 2. Algorithm of left-to-right binary method with Montgomery multiplication to compute $Z = X^E \bmod p$.

and mask operations. It calculates $C = AB2^{-n} \bmod p$, where $0 \leq A, B < p < 2^n$ is met. To be divisible by $2^n$, a multiple of $p$, or $Tp$, is added to $AB$. Such $T$ is calculated by $T = ABq \bmod 2^n$, using $q$ that meets $pq \equiv -1 \pmod{2^n}$. As a result, the Montgomery multiplication actually calculates $C = (AB + Tp)2^{-n} \bmod p$. Since both $AB$ and $Tp$ is smaller than $2^n p$, the above modulo operation can be done by performing a subtraction of $p$ once or zero times.

In actual computation circuits, algorithms where an $n$-bit integer is divided into $m$ words of $r$-bit integer are usually used. They are called high-radix Montgomery multiplication with a radix of $2^r$. Figure 1 shows one of the high-radix Montgomery multiplication algorithms proposed by Satoh and Takano [8]. In this paper, we define it as a function $\text{MM}(A, B, p, q)$. There, words of $A$ are represented by $A_{m-1}, A_{m-2}, ...,$ and $A_0$ and $q$ is redefined as $q = -p^{-1} \bmod 2^r$.

1: $pw \leftarrow \text{MM}(1, 2^{2n}, p, q)$
2: $Z \leftarrow \text{MM}(X, 2^{2n}, p, q)$
3: **for** $i$ = 0 to $k-1$ **do**
4: $pw_{new} \leftarrow \text{MM}(pw, Z, p, q)$
5: $Z \leftarrow \text{MM}(Z, Z, p, q)$
6: **if** $e_i = 1$ **then**
7: $pw \leftarrow pw_{new}$
8: **end if**
9: **end for**
10: $Z \leftarrow \text{MM}(pw, 1, p, q)$

Fig. 3. Algorithm of right-to-left binary method with Montgomery multiplication.

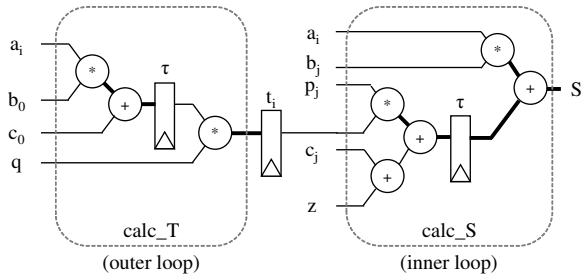


Fig. 4. Dataflow of the calculation of $t_i$ and $S$.

The Montgomery multiplication is applied to modular exponentiation by being combined with a binary method. Figures 2 and 3 describe algorithms where the Montgomery multiplication is combined with left-to-right and right-to-left binary methods, respectively. The $i$-th bit of exponent $E$ is represented by $e_i$ in the figures. When the Montgomery multiplication is applied to general modular multiplication, proper pre- and post-processes are required. They are multiplication and division by $2^n$ modulo $p$, which are the same as the Montgomery multiplication by $2^{2n}$ and 1, respectively. The right-to-left binary method is suitable for fast processing because two Montgomery multiplications (Lines 4 and 5) can be performed in parallel, while it requires larger amount of hardware.

### B. Hardware Implementations

There are a number of FPGA implementations of high-radix Montgomery multiplication algorithms. Blum and Paar [1] selected a radix of $2^4$ that fitted to LUT implementation and proposed a systolic array architecture for efficient computation. Song et al. [9] proposed an architecture with only one RAM block and one DSP block for minimum circuit area and high operating frequency. Suzuki and Matsumoto [10] optimized DSP blocks of a Xilinx FPGA for high-radix Montgomery multiplication by finely controlling the operations of the DSPs. The radix of the both implementations was set to $2^{17}$ by considering the input bit width of DSPs. San and At [7] applied the Karatsuba method to multiplication for the further reduction of the number of operations. They selected radices of $2^{16}, 2^{32}$, and $2^{64}$. Since we implement circuits simply described from the algorithm, we do not intend to compare our circuits with these optimized circuits. Although the idea of this study can be applied to more complicated circuits, it is left as a future work.

## III. Design and Implementation

### A. Baseline Design

In our designs, the high-radix Montgomery multiplication algorithm shown in Fig. 1 is translated into a hardware description with minimum modification. We focus on arithmetic operations in Lines 4 and 6 and divide them with a temporary variable $\tau$ as follows:

1: $\tau \leftarrow (a_i b_0 + c_0) \bmod 2^r$

TABLE I
THE COMBINATORIAL DELAY OF COMPUTING ELEMENTS, ESTIMATED ON AN ARTIX-7 FPGA.

| Computing Elements | Delay [ns] |
|---|---|
| Multiply-add ($r = 16$) | 2.54 |
| Multiply-add ($r = 32$) | 9.20 |
| Multiply-add ($r = 64$) | 13.82 |
| Multiply-add ($r = 128$) | 23.55 |
| Subtraction ($n = 1024$) | 31.77 |

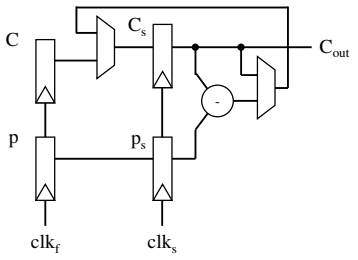


Fig. 5. Block diagram of the calculation of $C - p$ (if $C > p$), which corresponds to Lines 14–16 of Fig. 1.

2: $t_i \leftarrow \tau q \bmod 2^r$
3: $\tau \leftarrow t_i p_j + c_j + z$
4: $S \leftarrow a_i b_j + \tau$

It is clear that each of these operations can be performed by an $r$-bit, 2-input multiplier and a $2r$-bit, 3-input adder. They are combined into a 4-input multiply-add unit, shared by these operations clock-by-clock. Figure 4 shows the dataflow of these operations. Thin and thick lines represent $r$-bit and $2r$-bit signals, respectively. The upper $r$ bits are discarded in the calculation of $t_i$ because of the modulo operation.

Although the calculation within the loop are composed of $r$-bit or $2r$-bit operations, the final comparison and subtraction are $n$-bit operations. If they are directly described, they are likely to include the critical path. Table I summarizes the combinatorial delay of computing units estimated by our preliminary evaluation. The evaluation environment is the same as that shown in Section IV. When $n$ is set to 1024, the delay of subtraction is clearly longer than that of the multiply-add unit. To avoid this bottleneck, the subtraction can also be divided into $r$-bit words. We define the baseline design as a design where the subtraction is divided into words.

### B. Multiple Clock Domain Design

In this study, we examine another design where the subtraction is driven by a slower clock, rather than dividing it into words. Generally speaking, violation of timing constraints in multiple clock domain design causes metastability, which might result in an unpredictable error of the circuit [3]. One of the solutions in FPGAs is to generate phase-aligned clocks using a DCM (Digital Clock Manager) block. In this case, the ratio of clock frequencies should be as simple as they can (or $N$:1 if possible). Other solutions are preferred if the ratio is complicated, such as a double flop technique (using two flip-flops as a synchronizer) and the use of asynchronous FIFO based on a RAM block [3].

Figure 5 depicts the block diagram of the multiple clock domain design to calculate $C-p$. Note that all the data signals in this figure are $n$ bits wide as it corresponds to Lines 14–16 of Fig. 1. Fast and slow clocks are denoted by $clk_f$ and $clk_s$, respectively. The point of the design is that the operand calculated by a circuit of the fast clock has to be copied to registers driven by the slow clock. This ensures that the operation starts at the edge of the slow clock. The operation is performed using the copied values $C_s$ and $p_s$ and its result is written back to $C_s$. Although the output $C_s$ is also used in the fast clock domain, it is not problematic when the ratio of clock frequencies is $N$:1.

## IV. EVALUATION

### A. Methodology

In this section, the two designs of a high-radix Montgomery multiplier shown in Section III are implemented and evaluated on an FPGA. They are included in modular exponentiation circuits. In the evaluation, the baseline design described in Section III-A is defined as *Single* and the multiple clock domain design described in Section III-B is defined as *Multi*. As binary methods of modular exponentiation, both the left-to-right (LR) and right-to-left (RL) methods are implemented. Since a modular exponentiation circuit has thousands of bits of the input and output ports, it is rarely used alone. An interface circuit that communicates with external circuits via 16-bit interconnects is also implemented. The bit width of modular exponentiation is set to 1024, while the radix of Montgomery multiplication is set to $2^{16}, 2^{32}, 2^{64}$, or $2^{128}$ (i.e. $r = 16, 32, 64$, or 128). Based on the result of a preliminary evaluation such as Table I, the ratio of frequency of the fast clock and the slow clock is set to 6:1, 4:1, 3:1, or 2:1, respectively. The interface circuit is driven by the slow clock regardless of the design of Montgomery multiplier.

The circuits are synthesized and implemented by Vivado 2017.3 with the default settings. The target FPGA is Xilinx Artix-7 XC7A200T-2FBG676C. Due to constraints of an external clock and a DCM block, the frequency of the slow clock is set to a multiple of $25/48 (\sim 0.52)$ MHz. As measures of the amount of hardware, the number of slices, LUTs, flip-flops (FFs), and DSPs are evaluated. In addition to the whole circuits, the number of slices in the modular exponentiation circuit (without the interface circuit) is also collected. As measures of operation speed, the estimation of the maximum operating frequency of the fast clock, the number of cycles of the fast clock to complete the calculation, and the estimated execution time are evaluated. The maximum operating frequency of the fast clock $F_{max}$ is estimated by $1000/(t_{targ} - \text{WNS})$ MHz, where $t_{targ}$ [ns] is the shortest cycle time of the fast clock that meets all the timing constraints, and WNS [ns] is the worst negative slack of the fast clock under those constraints. The number of cycles is measured by simulation. To estimate the worst case, the exponent $E$ is set to $2^n - 1$ in the simulation.

The execution time is estimated by dividing the number of cycles by the maximum frequency.

### B. Results

Table II summarizes the evaluation results. The difference of *Multi* from *Single* is shown in parentheses. The absolute difference is shown in the evaluation of the amount of hardware, while the relative difference is shown in the evaluation of operation speed.

The operating frequency decreased by 0.8% on average, which ranged from a 4.4% decrease to a 2.7% increase. It can be considered within the range of the variation due to optimization. It did not affect the critical path to drive a part of the calculation by a slower clock. The number of cycles required for calculation reduced by 3.2% at a maximum, depending on the size of words $r$. This is because the proportion of the subtraction in the high-radix Montgomery multiplication becomes large when $r$ is large (or the number of division $m$ is small). As a result, the overall execution time reduced by 1.0% on average. The simplicity of the hardware description was achieved without degradation of execution time.

The increase of logic elements was, at a maximum, 2,556 LUTs and 1,043 flip-flops with the LR binary method or 3,555 LUTs and 2,023 flip-flops with the RL binary method. Taking into account the fact that the RL binary method includes two Montgomery multipliers, the increase per Montgomery multiplier maximized in the case of the LR binary method. The increase of slices of the whole circuits varied widely among them. However, according to the results without the interface circuits, the increase of slices per Montgomery multiplier can be estimated as about 600. Additional flip-flops came from the copy of register value shown in Fig. 5. Since the output register $C_{out}$ was also required in the baseline design, the register for $p_s$ contributed to the increase. One of the reasons for the increase of the LUTs was the increase of the bit width of the subtractor. However, the amount of increment was too large to account for due to a single reason. In respect of energy consumption, the increase with the multiple clock domain design might be smaller than that of logic elements because circuits driven by the slow clock became larger. Analyses on the essential effect of dividing clocks, including energy consumption, is left for a future work.

## V. Conclusion

A simple hardware description is useful in the early steps of computation circuit design. In this paper, we presented a case study for multiple clock domain design within a module to achieve simplicity. We designed a high-radix Montgomery multiplier and included it in modular exponentiation circuits. According to the evaluation, the overall execution time reduced by 1.0% on average and the simplicity of the hardware description was achieved. The increase of logic elements per Montgomery multiplier was 2,556 LUTs and 1,043 flip-flops at a maximum. Further analyses on the effect of dividing clocks, comparison with recent works, and application of the idea of multiple clocks to more optimized circuits are left for future studies.


## Acknowledgement

This study was partially supported by JSPS Grants-in-Aid for Scientific Research (KAKENHI) Grant Number 16K00072.


## References


[1] T. Blum and C. Paar, "High-Radix Montgomery Modular Exponentiation on Reconfigurable Hardware," *IEEE Transactions on Computers*, vol. 50, no. 7, pp. 759–765, 2001.

[2] G. K. Gultekin and A. Saranli, "An FPGA based high performance optical flow hardware design for computer vision applications," *Microprocessors and Microsystems*, vol. 37, no. 3, pp. 270–286, 2013.

[3] S. Kilts, *Advanced FPGA Design: Architecture, Implementation, and Optimization*. Wiley–IEEE Press, 2007.

[4] P. L. Montgomery, "Modular Multiplication Without Trial Division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985.

[5] J. Oliver *et al.*, "Synchroscalar: A Multiple Clock Domain, Power-Aware, Tile-Based Embedded Processor," in *Proc. 31st Annual International Symposium on Computer Architecture (ISCA)*, 2004, pp. 150–161.

[6] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-key Cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[7] I. San and N. At, "Improving the Computational Efficiency of Modular Operations for Embedded Systems," *Journal of Systems Architecture*, vol. 60, no. 5, pp. 440–451, 2014.

[8] 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.

[9] B. Song, K. Kawakami, K. Nakano, and Y. Ito, "An RSA Encryption Hardware Algorithm Using a Single DSP Block and a Single Block RAM on the FPGA," in *Proc. 1st International Conference on Networking and Computing (ICNC)*, 2010, pp. 140–147.

[10] D. Suzuki and T. Matsumoto, "How to Maximize the Potential of FPGA-Based DSPs for Modular Exponentiation," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E94-A, no. 1, pp. 211–222, 2011.

[11] P. Teehan, M. Greenstreet, and G. Lemieux, "A Survey and Taxonomy of GALS Design Styles," *IEEE Design and Test of Computers*, vol. 24, no. 5, pp. 418–428, 2007.

TABLE II
EVALUATION RESULTS.

| Binary | $r$ | Clock | whole circuit | | | | w/o I/F | $F_{max}$ [MHz] | Cycle | Time[ms] |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Slice | LUT | FF | DSP | Slice | | | |
| LR | 16 | Single | 3,899 | 8,231 | 16,542 | 2 | 2,353 | 138.07 | 17,193,354 | 124.53 |
| | | Multi | 4,838 | 10,754 | 17,585 | 2 | 3,092 | 135.16 | 17,084,706 | 126.41 |
| | | | (+939) | (+2,523) | (+1,043) | | (+739) | (-2.1%) | (-0.6%) | (+1.5%) |
| LR | 32 | Single | 3,932 | 8,376 | 16,620 | 4 | 2,297 | 84.52 | 4,401,352 | 52.08 |
| | | Multi | 3,923 | 10,932 | 17,663 | 4 | 2,777 | 83.43 | 4,346,004 | 52.09 |
| | | | (-9) | (+2,556) | (+1,043) | | (+480) | (-1.3%) | (-1.3%) | (+0.0%) |
| LR | 64 | Single | 3,739 | 9,270 | 16,782 | 16 | 2,362 | 61.42 | 1,154,154 | 18.79 |
| | | Multi | 4,352 | 11,320 | 17,821 | 16 | 3,063 | 58.74 | 1,131,603 | 19.27 |
| | | | (+613) | (+2,050) | (+1,039) | | (+701) | (-4.4%) | (-2.0%) | (+2.5%) |
| LR | 128 | Single | 4,394 | 11,066 | 17,128 | 64 | 2,952 | 42.85 | 317,752 | 7.41 |
| | | Multi | 4,816 | 12,640 | 18,139 | 64 | 3,420 | 44.01 | 307,504 | 6.99 |
| | | | (+422) | (+1,574) | (+1,011) | | (+468) | (+2.7%) | (-3.2%) | (-5.8%) |
| RL | 16 | Single | 5,410 | 14,037 | 21,846 | 4 | 3,801 | 131.72 | 8,605,068 | 65.33 |
| | | Multi | 6,714 | 17,051 | 23,832 | 4 | 5,110 | 132.91 | 8,550,690 | 64.34 |
| | | | (+1,304) | (+3,014) | (+1,986) | | (+1,309) | (+0.9%) | (-0.6%) | (-1.5%) |
| RL | 32 | Single | 5,084 | 14,383 | 21,989 | 8 | 3,684 | 84.65 | 2,202,824 | 26.02 |
| | | Multi | 5,829 | 17,938 | 23,972 | 8 | 4,838 | 85.61 | 2,175,124 | 25.41 |
| | | | (+745) | (+3,555) | (+1,983) | | (+1,154) | (+1.1%) | (-1.3%) | (-2.4%) |
| RL | 64 | Single | 5,237 | 15,121 | 22,275 | 32 | 3,825 | 60.05 | 577,641 | 9.62 |
| | | Multi | 7,179 | 18,184 | 24,256 | 32 | 5,134 | 60.04 | 566,355 | 9.43 |
| | | | (+1,942) | (+3,063) | (+1,981) | | (+1,309) | (+0.0%) | (-2.0%) | (-1.9%) |
| RL | 128 | Single | 5,781 | 18,186 | 22,805 | 128 | 4,750 | 42.74 | 159,032 | 3.72 |
| | | Multi | 6,801 | 21,412 | 24,828 | 128 | 5,699 | 41.48 | 153,904 | 3.71 |
| | | | (+1,020) | (+3,226) | (+2,023) | | (+949) | (-3.0%) | (-3.2%) | (-0.3%) |