Hardware Implementation of Montgomery Modular Multiplication Algorithm Using Iterative Architecture

Antonius P. Renardy, Nur Ahmadi, Ashbir A. Fadila, Naufal Shidqi, Trio Adiono
Department of Electrical Engineering, School of Electrical Engineering and Informatics
Bandung Institute of Technology, Jl. Ganesha No. 10 Bandung, 40132, Indonesia
Email: antoniusperdana@students.itb.ac.id, tadiono@stei.itb.ac.id

***Abstract*—Modular multiplication is an integral part of RSA cryptosystems and its performance heavily determines the performance of the encryption hardware. This paper provides a hardware implementation of Montgomery's modular multiplication algorithm using iterative architecture. The propsed design is implemented in Verilog HDL and simulated functionally using ModelSim Altera 10.1E. The synthesis is performed using Altera Quartus II 9.1 with target FPGA board Altera DE2-70. The proposed design consumes 17540 logic elements with 15480 LUT and takes 2048 clock cycles to perform multiplication process. Based on trade-off parameter AT^2 measure, the proposed design offers the best performance among other designs.**

***Keywords*—RSA Cryptosystem, Modular Multiplication, Montgomery's Algorithm, Iterative Architecture, FPGA.**

I. Introduction

RSA is one of the first practicable public-key cryptosystems and is widely used for secure data transmission. RSA was invented by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977. In such a cryptosystem, the encryption key is public and contrasts from the decryption key which is kept secret.

The performance of public-key cryptosystems is mainly determined by the efficiency of the modular multiplication and exponentiation as the operands (the plain text of a message or the cipher or possibly a partially ciphered text) are usually large, i.e. 1024 bits or even more than 2048 bits. In order to improve time requirements of the encryption/decryption operations, it is necessary to minimize the number of modular multiplications performed and to reduce the time requirement of a single modular multiplication. There are various algorithms that implement modular multiplication such as Brickells [1] and Kochanki's algorithm [2], Booth's methods [3], and Walters's algorithm [4]. In this paper, we focus on the Montgomery's algorithm [5] as it is considered the most popular and efficient.

The main idea of Montgomery's algorithm is to simply use addition, subtraction, and shift operations to evade trial division. This way, multiplication can be performed faster than the naïve approach. This paper provides a prototype for implementing Montgomery's algorithm with iterative architecture.

The rest of this paper is organized as follows: Section II covers basic Montgomery's algorithm and its iterative algorithm; Section III describes the proposed iterative architecture of Montgomery's algorithm for hardware implementation; Section IV shows functional simulation result using ModelSim Altera 10.1E, synthesis result in Quartus II 9.1 with target board Altera DE2-70, and also the performance analysis based on the result obtained; finally, conclusions are drawn in Section V and some directions for future work are provided.

II. Montgomery's Algorithm

Generally, algorithms that formalize modular multiplication of $P = A \times B \,(\mathrm{mod}\, M)$ comprises two steps: the first is generating the product $P = A \times B$ and the second is reducing the product P to achieve modulo of M.

Implementing a multiplication can be straightway made up by using iterative adder-accumulator for the generated partial products. Nevertheless, this way is pretty slow as the final result is only accessible after n clock cycles, where n is the size of the operands.

Adding some partial products at once from iterative multiplier can make it faster. This could be attained by expanding the iterative multiplier and producing a combinatorial circuit that comprises several partial product generators together with some adders that work in parallel.

One of the broadly used algorithms for efficient modular multiplication is Montgomerys algorithm [5] [6] [7]. This algorithm calculates the product of two integers modulo a third one without implementing any trial divisions by M. It produces the reduced product using sequences of additions.

Let A, B, and M be the multiplicand, the multiplier and the modulus, respectively, and let n be the number of digits in their binary representations. Therefore,we express A, B, and M as

$$A = \sum_{i=0}^{n-1} a_i \times 2^i \tag{1}$$

$$B = \sum_{i=0}^{n-1} b_i \times 2^i \tag{2}$$

$$M = \sum_{i=0}^{n-1} m_i \times 2^i \tag{3}$$

The prerequisite of the Montgomery's algorithm are as follows.

- The modulus M must be relatively prime to the radix, such as there is no common divisor for M and the radix;

978-1-4799-7711-6/15/$31.00 © 2015 IEEE

- The multiplicand and the multiplier need to be smaller than M.

Since binary representation are used for the operands, then the modulus M needs to be odd to fulfill the first prerequisite. The Montgomery's algorithm uses the least significant bit (LSB) of the accumulating modular partial product to decide the multiple of M to subtract. The common multiplication order is reversed by selecting multiplier bits from least to most significant and shifting right. If R is the recent modular partial product, then q is selected so that $R + q \times M$ is a multiple of the radix r, and it means it is right-shifted by 1 position, such as divided by r for use in the next iteration. Thus, after n iterations, the result attained is $R = A \times B \times r^{-n} (\mathrm{mod}\, M)$. Montgomery's modular multiplication algorithm is described in Figure 1.

Input: A, B, M
Require: $A, B < M$ and M is odd
Output: $R \leftarrow A \times B \times r^{-n} \bmod M$
1: $R \leftarrow 0$
2: **for** $i = 0 \rightarrow n - 1$ **do**
3: $R \leftarrow R + a_i \times B$
4: **if** $r_0 = 0$ **then**
5: $R \leftarrow R$ div 2
6: **else**
7: $R \leftarrow (R + M)$ div 2
8: **end if**
9: **end for**
10: **return** R

Fig. 1. Montgomery's modular multiplication algorithm

In order to produce the desired result, we need an extra Montgomery modular multiplication by the constant $2^n (\mathrm{mod}\, M)$. However as the main goal of the use of Montgomery modular multiplication algorithm is to calculate exponentiations, it is better to Montgomery pre-multiply the operands by 2^n and Montgomery post-multiply the result by 1 to remove of the 2^{-n} factor. Often this procedure is not necessary since RSA hardware will be working on Montgomery domain.

III. Iterative Architecture for Montgomery's Modular Multiplication

The block diagram of the proposed architecture is shown on Figure 2. Serial-In-Parallel-Out (SIPO) and Parallel-In-Serial-Out (PISO) modules are used to overcome I/O limitation on DE2-70 board. The board has only 475 I/O pin while the modular multiplication are designed for 2048-bits datapath. Therefore, the inputs to the top level of the design are partitioned by 64-bits and delivered in serial with the least-significant bits are sent first. The output of the SIPO modules are the amalgamation of the partitions (64-bits wide) of inputs. The PISO modules perform the exact opposite of the SIPO modules.

The inputs are processed in iterative way inside the Montgomery Core. The core, shown in Figure 3, consists of Montgomery Cell which are made of adders and multiplexers as shown in Figure 4. The cell are the hardware equivalent of the algorithm inside the iterative loop in Figure 1.

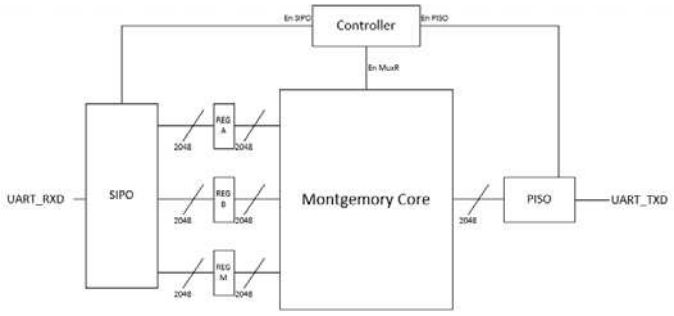


Fig. 2. Top level block diagram

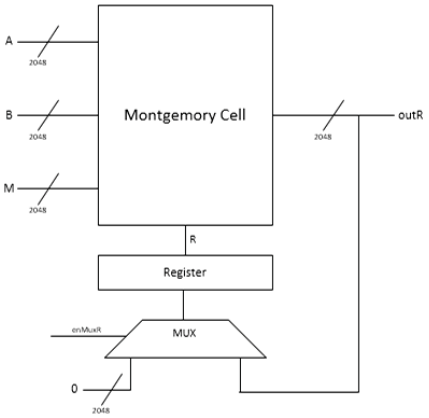


Fig. 3. Register Transfer Level (RTL) of Montgomery Core

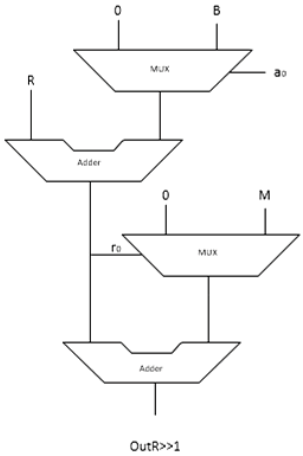


Fig. 4. Register Transfer Level (RTL) of Montgomery Cell

Iteration processes are controlled by Controller which generate appropriate control signals based on counter. The Controller also generates control signal to indicate the SIPO and PISO modules when to load the I/O.

IV. Synthesis and Simulation Results

The hardware is implemented using Verilog HDL. Functional simulation is performed using testbench file with predefined test stimuli in ModelSim Altera 10.1E. Figure 5 shows the functional simulation result of the hardware design. The hardware implementation requires 2048 clock cycles to

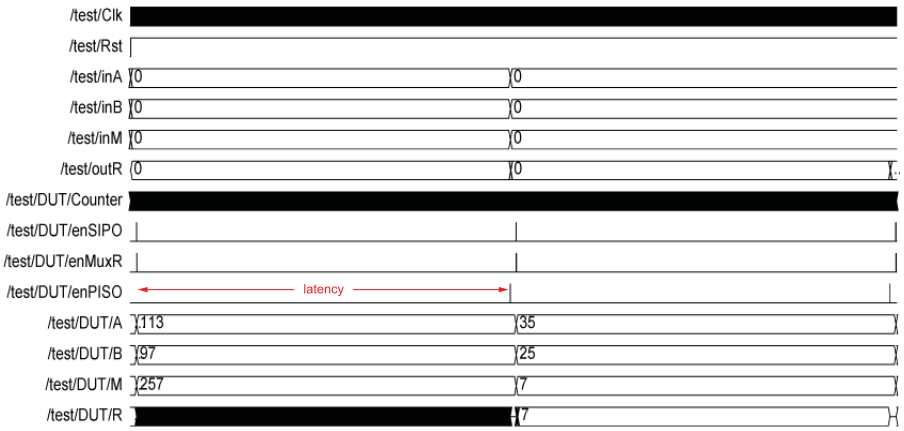
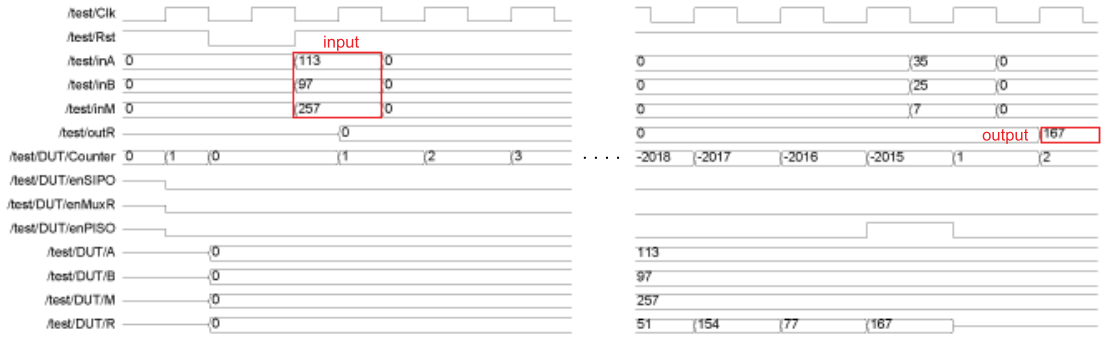
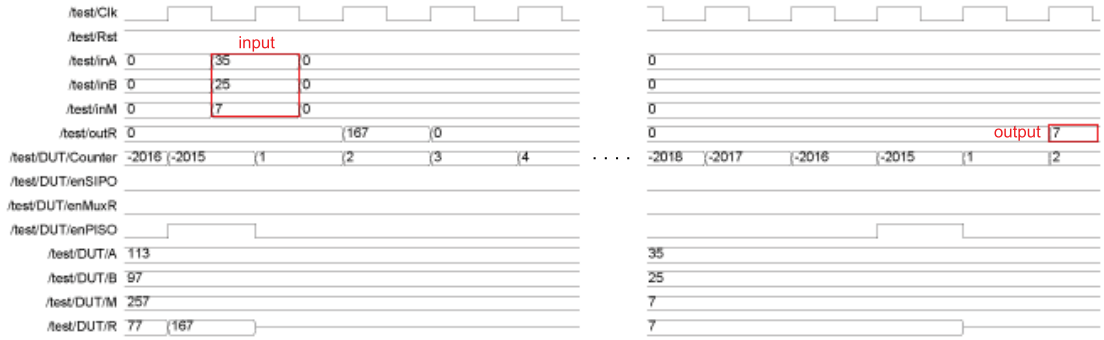


Fig. 5. Functional simulation showing latency



(a) Input (left) and output (right) of first input



(b) Input (left) and output (right) of second input

Fig. 6. Functional simulation showing input and output

generate the parallel output from its parallel input, thus the latency of the design is 2048 cycles. The following inputs are processed after 2048 clock cycles since the previous input, hence the throughput has the same number of cycles as the latency.

Figure 6a and 6b present functional simulation results of our proposed design for the first and second input, respectively. The summary of this results can be seen in Table I. The hardware computes the result in Montgomery domain. This way, the results can immediately be used for further calculation since RSA employs modular exponentiation for encryption. To transform the results back from Montgomery domain to real numbers domain, we can denote that

$$R_{real} \equiv R_{montgomery} \times 2^n (\mathrm{mod}\ m) \tag{4}$$

with n represents the number of bits in the datapath and m the modulus.

TABLE I. SUMMARY OF SIMULATION WAVEFORMS FOR $N = 2048$

Input 1	Input 2	Modulus	Result		
			MD*	RD**	Expected
113	97	257	167	167	167
35	25	7	7	0	0

* Montgomery Domain
** Real Domain

From the results in Table I, comparison between results

in real domain to the expected results yield no difference. Hence, it can be inferred that the hardware designed are able to perform modular multiplication with accurate results.

Synthesis is performed using Altera Quartus II 11.0 and the summary of the compilation report is shown in Table II. The proposed design consumes 17540 logic elements, which utilizes 25.63% of total logic elements in Altera DE2-70.

Area-Time Square (AT^2) measure is used in this paper to compare the performance of the proposed design and other existing designs. This AT^2 measure was originally proposed by Thompson [8] and used in several literatures such as [9]–[11]. The parameter A denotes the chip area while T denotes the time required to perform computation (latency).

The area of the design is represented by Look-up Table (LUT) obtained from synthesizing process. The proposed design consumes smaller area of 15840 LUT compared to [6] [7] [12] and [13]. Our proposed design only needs 2048 clock cycles to finish multiplication computation. However, in term of area consumption, Tenca & Koc's design produces the best result.

Based on trade-off parameter (AT^2), our proposed design provides best performance among other existing designs as shown in Table III. Therefore, the proposed design is suitable for implementation in the target FPGA Altera DE2-70.

TABLE II. PERFORMANCE PARAMETER

Area (A)	Latency (T)	AT^2
15480 LUTs 2060 Reg 17540 LEs	2048 clock	7.35×10^{10}

TABLE III. DESIGN COMPARISON

Design [14]	Area (LUT)	Latency (Cycles)	AT^2
Tenca & Koc	12774	4224	22.7×10^{10}
Harris et al.	18455	2319	9.92×10^{10}
Huang	18535	2176	8.77×10^{10}
McIvor	20453	2049	8.58×10^{10}
Our Design	**15480**	**2048**	$\mathbf{6.57 \times 10^{10}}$

V. CONCLUSION

This paper presents hardware implementation of modular multiplier using iterative Montgomery modular multiplication algorithm for 2048-bit datapath. The design is successfully simulated using Verilog HDL in ModelSim Altera 10.1E. The advantage of this design lies on its relatively smaller area and least latency compared to other architectures in references since it only uses 1 cell as a Montgomery Core. This design also has Serial-In-Parallel-Out (SIPO) and Parallel-In-Serial-Out (PISO) modules which are used to overcome I/O limitation on DE2-70 board.

The synthesis is performed using Altera Quartus II 11.0. The proposed design consumes 17540 logic elements with 15480 LUT, which utilizes 25.63% of total logic elements in Altera DE2-70. The proposed design only takes 2048 clock cycles to perform multiplication. Based on AT^2 measure, the proposed design offers the best performance among other designs.

The results produced by the designed modular multiplier are in Montgomery domain. This will enable the results to be used in next calculation process of RSA algorithm.

REFERENCES

[1] E. F. Brickell, "A fast modular multiplication algorithm with application to two key cryptography," in *Advances in Cryptology*. Springer, 1983, pp. 51–60.

[2] M. Kochanski, "A new method of serial modular multiplication," Aug. 19, 2003. [Online]. Available: http://www.nugae.com/encryption/bin/design.pdf

[3] A. D. Booth, "A signed binary multiplication technique," *The Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, no. 2, pp. 236–240, 1951.

[4] C. D. Walter, "Systolic modular multiplication," *IEEE transactions on computers*, vol. 42, no. 3, pp. 376–378, 1993.

[5] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.

[6] N. Nedjah and L. de Macedo Mourelle, "Two hardware implementations for the montgomery modular multiplication: sequential versus parallel," in *Proceedings. 15th Symposium on Integrated Circuits and Systems Design, 2002.* IEEE, 2002, pp. 3–8.

[7] C. McIvor, M. McLoone, and J. V. McCanny, "Fast montgomery modular multiplication and RSA cryptographic processor architectures," in *Conference Record of the Thirty-Seventh Asilomar Conference on Signals, Systems and Computers, 2004.*, vol. 1. IEEE, 2003, pp. 379–384.

[8] C. D. Thompson, "Area-time complexity for vlsi," in *Proceedings of the eleventh annual ACM symposium on Theory of computing*. ACM, 1979, pp. 81–88.

[9] R. P. Brent and H. Kung, "The area-time complexity of binary multiplication," *Journal of the ACM (JACM)*, vol. 28, no. 3, pp. 521–534, 1981.

[10] H. Abelson and P. Andreae, "Information transfer and area-time tradeoffs for vlsi multiplication," *Communications of the ACM*, vol. 23, no. 1, pp. 20–23, 1980.

[11] R. E. Bryant, "On the complexity of vlsi implementations and graph representations of boolean functions with application to integer multiplication," *Computers, IEEE Transactions on*, vol. 40, no. 2, pp. 205–213, 1991.

[12] G. Perin, D. G. Mesquita, F. L. Herrmann, and J. B. Martins, "Montgomery modular multiplication on reconfigurable hardware: fully systolic array vs parallel implementation," *International Journal of Reconfigurable Computing*, pp. 6:1–6:10, 2011.

[13] A. E. Cohen and K. K. Parhi, "Architecture optimizations for the RSA public key cryptosystem: A tutorial," *IEEE Circuits and Systems Magazine*, vol. 11, no. 4, pp. 24–34, 2011.

[14] M. Huang, K. Gaj, and T. El-Ghazawi, "New hardware architectures for Montgomery modular multiplication algorithm," *IEEE Transactions on Computers*, vol. 60, no. 7, pp. 923–936, 2011.