PAPER Special Section on Selected Papers from the 20th Workshop on Circuits and Systems in Karuizawa

A High-Speed Design of Montgomery Multiplier

Yibo FAN[†a)], ***Nonmember***, **Takeshi IKENAGA**[†], ***Member***, ***and*** **Satoshi GOTO**[†], ***Fellow***

SUMMARY With the increase of key length used in public cryptographic algorithms such as RSA and ECC, the speed of Montgomery multiplication becomes a bottleneck. This paper proposes a high speed design of Montgomery multiplier. Firstly, a modified scalable high-radix Montgomery algorithm is proposed to reduce critical path. Secondly, a high-radix clock-saving dataflow is proposed to support high-radix operation and one clock cycle delay in dataflow. Finally, a hardware-reused architecture is proposed to reduce the hardware cost and a parallel radix-16 design of data path is proposed to accelerate the speed. By using HHNEC 0.25 μm standard cell library, the implementation results show that the total cost of Montgomery multiplier is 130 KGates, the clock frequency is 180 MHz and the throughput of 1024-bit RSA encryption is 352 kbps. This design is suitable to be used in high speed RSA or ECC encryption/decryption. As a scalable design, it supports any key-length encryption/decryption up to the size of on-chip memory.

key words: *Montgomery multiplier, high-speed, high-radix, scalable*

1. Introduction

Public key cryptography plays a very important role in modern information security. It not only can be used to encrypt/decrypt data like symmetric cryptography, but also can provide service such as confidentiality, authentication, data integrity check and non-repudiation. RSA algorithm [1], which is proposed by Rivest, Shamir and Adleman in 1976, is the most widely used public key cryptographic algorithm. ECC algorithm, which is introduced by Koblitz [2] and Victor S. Miller [3], is another very famous public cryptographic algorithm.

Both of RSA algorithm and ECC algorithm use modular multiplication as the primary operation. With the increase of the key-length used in these algorithms, the speed of modular multiplication becomes a bottleneck. A lot of papers have been published to accelerate the speed of modular multiplication. Till now, Montgomery modular multiplication algorithm [4] is considered as the most efficient algorithm. A lot of hardware implementations are based on this algorithm. Some of them focus on scalable design [5], [6], which makes the hardware implementation have ability to handle any key-length encryption/ decryption. Some focus on high-radix design in [7]–[9], [11], which can reduce total clock cycles for multiplication. Some focus on dataflow optimization [10], which can reduce the delay cycles in dataflow.

Manuscript received June 22, 2007.
Manuscript revised October 3, 2007.

[†]The authors are with IPS, Waseda University, Kitakyushu-shi, 808-0135 Japan.
a) E-mail: fanyibo@ruri.waseda.jp

DOI: 10.1093/ietfec/e91–a.4.971

In this paper, a high-speed design of Montgomery multiplier is presented. Firstly, by using the proposed modified scalable high-radix Montgomery algorithm, it can parallelize the data path and shorten the critical path. Secondly, by using the proposed high-radix clock-saving dataflow, it achieves high-radix design with one clock cycle delay in dataflow. Finally, a compact hardware design of Montgomery multiplier is proposed to reduce hardware cost and accelerate the speed.

The rest of the paper is organized as following: Montgomery algorithm and the proposed modified algorithm are introduced in Sect. 2. The proposed clock-saving dataflow is introduced in Sect. 3. The architecture design of Montgomery multiplier is presented in Sect. 4. The experimental results and analysis are presented in Sect. 5. Finally, conclusion is given in Sect. 6.

2. Algorithms

Table 1 shows the notations used in this paper.

2.1 Previous Algorithms

Algorithm 1: Montgomery Multiplication Algorithm

Input: X, Y, M
Output: $S = MM\ (\mathrm{X}, \mathrm{Y}) = \mathrm{XY}r^{-1} \bmod \mathrm{M}$
1. $S = 0$
2. For $i = 0$ to $N - 1$
3. $\quad S = \mathrm{S} + X_i \times \mathrm{Y}$

Table 1 Notations used in this paper.

Notation	Description	Notation	Description
M	Modulus (N bits)	X^i	i^{th} word of X
X	Multiplier operand (N bits)	X_i	i^{th} bit of X
Y	Multiplicand operand (N bits)	q_{Yj}, q_{Mj}	Coefficients used in high-radix algorithm
r	Constant, $r=2^N$	BPW	Bits-Per-Word
N	Bit length of operands	NS	Number of Stages in Dataflow
S	Product	NW	Number of Words in Y and M
(S, C)	Carry-save product	k	Radix = 2^k
(S^i, C^i)	i^{th} word of Carry-save product	~	Inverse Operation

4. $S = S + S_0 \times M$
5. $S = S/2$
6. End For
7. If $S >= M$ Then $S = S - M$
8. Return S

Algorithm 1 shows the original Montgomery algorithm. The advantage of this algorithm is that the division in modular operation is replaced by shift. In this way, this algorithm is very suitable to be implemented in hardware.

The direct hardware implementation of Montgomery algorithm can't support variable key-length operation. To make Montgomery algorithm scalable to variable key-length and improve the speed of this algorithm, Tenca and Koç proposed a scalable high-radix Montgomery multiplication algorithm:

Algorithm 2: Scalable High-radix Montgomery Multiplication Algorithm

Input: X, Y, M
Output: $S = MM(X, Y) = XYr^{-1} \bmod M$
1. $S = 0$, C = 0, $X_{-1} = 0$
2. For $j = 0$ to $N - 1$ Step k
3. $q_{Yj} = Booth(X_{j+k-1\ldots j-1})$
4. $(C^0, S^0) = C^0 + S^0 + q_{Yj} \times Y^0$
5. $q_{Mj} = (S^0_{k-1\ldots 0} + C^0_{k-1\ldots 0}) \times (2^k - (M^0)^{-1}_{k-1\ldots 0}) \bmod 2^k$
6. $(C^0, S^0) = C^0 + S^0 + q_{Mj} \times M^0$
7. For $i = 1$ to $NW - 1$
8. $(C^i, S^i) = C^i + S^i + q_{Yj} \times Y^i + q_{Mj} \times M^i$
9. $S^{i-1} = (S^i_{k-1\ldots 0}, S^{i-1}_{BPW-1\ldots k})$
10. $C^{i-1} = (C^i_{k-1\ldots 0}, C^{i-1}_{BPW-1\ldots k})$
11. End For
12. $S^{NW-1} = sign\ ext(S^{NW-1}_{BPW-1\ldots k})$
13. $C^{NW-1} = sign\ ext(S^{NW-1}_{BPW-1\ldots k})$
14. End For
15. $P = S + C$
16. If $P >= M$ Then $P = P - M$
17. Return P

The *sign ext* in step 12 and 13 is sign extending operation. The Booth function in step 3 is used to support high-radix operation. The detail of Booth function is,

$$\text{Booth}\,(X_{j+k-1\ldots j-1}) = -2^k X_{j+k-1} + 2^{k-1} X_{j+k-2} \cdots + 2^0 X_{j-1} \quad (1)$$

Algorithm 2 provides some advantages compared to algorithm 1. Firstly, word-based operation makes the multiplier be scalable to variable key-length. Secondly, high-radix design processes multiple bits of X in every loop (Step 3, Algorithm 2). It can reduce the clock cycles used in multiplication. Thirdly, carry-save adder is introduced to reduce critical path.

However, there are some disadvantages in Algorithm 2. Firstly, high radix design makes the calculation of q_{Yj} and q_{Mj} very complex, and the path delay will be increased very quickly when using higher radix. This problem can be solved by using our proposed algorithm (Sect. 2.2) and data path architecture (Sect. 4). Secondly, scalable design makes the data be dependent in pipeline, which causes two clock cycles delay in pipeline. This problem also can be solved by using our proposed high-radix clock-saving dataflow in Sect. 3.

2.2 Proposed Algorithm

Algorithm 3: Modified scalable high-radix Montgomery multiplication algorithm

Input: X, Y, M
Output: $S = MM(X, Y) = XYr^{-1} \bmod M$
1. $S = 0$, $C = 0$, $X_{-1} = 0$, $Y = Y \times 2^k$
2. For $j = 0$ to $N + k - 1$ Step k
3. $q_{Yj} = Booth(X_{j+k-1\ldots j-1})$
4. $q_{Mj} = (S^0_{k-1\ldots 0} + C^0_{k-1\ldots 0}) \times (2^k - (M^0)^{-1}_{k-1\ldots 0}) \bmod 2^k$
5. $(C^0, S^0) = S^0 + C^0 + q_{Yj} \times Y^0 + q_{Mj} \times M^0$
6. For $i = 1$ to $NW - 1$
7. $(C^i, S^i) = S^i + C^i + q_{Yj} \times Y^i + q_{Mj} \times M^i$
8. $S^{i-1} = (S^i_{k-1\ldots 0}, S^{i-1}_{BPW-1\ldots k})$
9. $C^{i-1} = (C^i_{k-1\ldots 0}, C^{i-1}_{BPW-1\ldots k})$
10. End For
11. $S^{NW-1} = sign\ ext(S^{NW-1}_{BPW-1\ldots k})$
12. $C^{NW-1} = sign\ ext(S^{NW-1}_{BPW-1\ldots k})$
13. End For
14. $P = S + C$
15. If $P >= M$ Then $P = P - M$
16. Return P

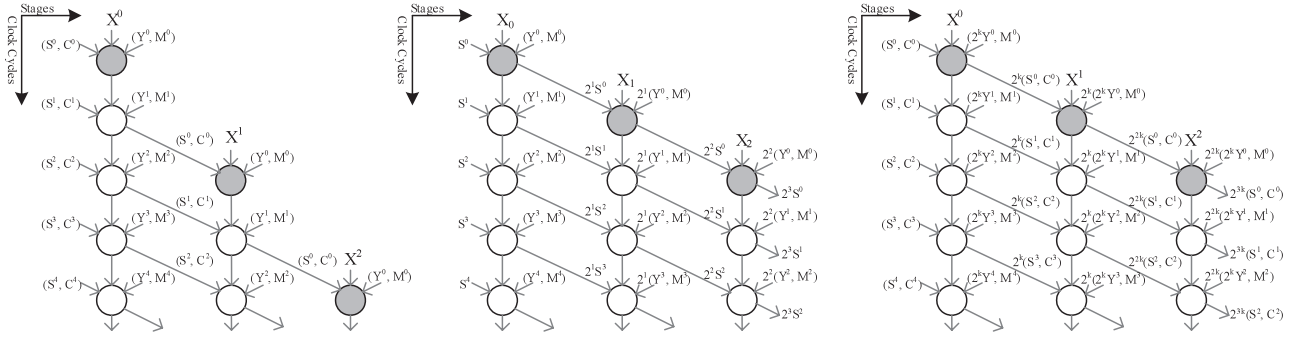
Algorithm 3 is different from algorithm 2. Firstly, in step 1, Y is multiplied by 2^k. In this way, all of the k-LSB of Y is zero. The result of $(S^0_{k-1\ldots 0}, C^0_{k-1\ldots 0})$ is not changed by adding $q_{Yj} \times Y^0$. The calculation of q_{Mj} is independent of q_{Yj}. Secondly, the calculation of q_{Yj} and q_{Mj} are performed in parallel (Step 3, 4), while these two calculations are calculated in serial in algorithm 2 (Step 3, 5).

For high-radix design, the path delay of q_{Yj} and q_{Mj} is large. In this way, our proposed algorithm can support parallel calculation of q_{Yj} and q_{Mj}, and it achieves much shorter critical path than algorithm 2. Our proposed algorithm is much more suitable for high-speed hardware implementation of Montgomery algorithm.

3. Data Flows

3.1 Previous Dataflows

The most frequently used dataflow for scalable high-radix Montgomery algorithm is shown in Fig. 1(a), which is proposed by Tenca and Koç in [7]. This data flow is a pipeline data flow of algorithm 2. Due to the data dependence in algorithm 2 (In Step 9, 10, the output (S^{i-1}, C^{i-1}) needs both of (S^i, C^i) and (S^{i-1}, C^{i-1})), there are two clock cycles delay in this dataflow. As a result, this dataflow needs more clock cycles to complete one time multiplication.



(a) Tenca-Koç Dataflow Two clock cycles delay Used for Algorithm 2 with Radix 2^k

(b) Herris Dataflow One clock cycles delay Used for Algorithm 2 with Radix 2

(c) Proposed Dataflow One clock cycles delay Used for Algorithm 3 with Radix 2^k

Fig. 1 Dataflow.

In order to deal with this problem, Herris proposed a new radix-2 dataflow in [10], which is shown in Fig. 1(b). This dataflow achieves one clock cycle delay by removing data dependence in algorithm 2. As shown in algorithm 2, the right-shifting (Step 9, 10.) causes data dependence. In Herris' dataflow, the right-shifting of product S is removed. As a result, the product S is equivalently be multiplied by 2 in every pipeline stage, so the input data (Y, M) of next stage need to be multiplied by 2 too.

Tenca's dataflow uses carry-save result and support high-radix design. Herris' dataflow achieves one clock cycle delay while using radix-2. However, both of Tenca's dataflow and Herris dataflow base on algorithm 2, they can't be used in this paper.

3.2 Proposed High-Radix Clock-Saving Dataflow

The proposed dataflow is shown in Fig. 1(c). This dataflow bases on proposed algorithm 3. It achieves both of one clock cycle delay and high-radix design.

As shown in Fig. 1(c), operand Y is initially multiplied by 2^k as specified in algorithm 3, so the input data (Y, M) for each stage becomes (2^kY, M). In order to achieve one clock cycle delay in dataflow and support high-radix design, the input data (2^kY, M) needs to be multiplied by 2^k accumulatively in each stage (except the first stage).

Compare with others' dataflow, the proposed dataflow has some advantages. Firstly, high-radix and one clock cycle delay make this dataflow need very few clock cycles to do multiplication. Secondly, algorithm 3 used in this dataflow can achieve much shorter critical path than other's design. In this way, our dataflow is much more suitable for high-speed design of Montgomery multiplier.

4. Proposed Hardware Architecture

4.1 Hardware-Reused Multiplier Architecture

The proposed Montgomery multiplier is shown in Fig. 2(a). The multiplier's data path contains NS MM Cells. The MM Cell is the basic processing element in the pipeline. There are two coefficient processing elements, q_{Yj} PE and q_{Mj} PE. They can be shared to all of the MM Cells in pipeline. From Fig. 1(c), it can be seen that the calculation of q_{Yj} and q_{Mj} are done just in the first cycle of each stage (Grey cycle shown in Fig. 1(c)). All of the remained cycles (White cycles) don't need to calculate q_{Yj} and q_{Mj}. This property provides possibility to reuse the q_{Yj} PE and q_{Mj} PE in the data path.

The FIFO in Fig. 2(a) is used to avoid data overflow in pipeline. When the NW (Number of words of operands) is larger than NS (number of stages in dataflow), data overflow will happen in pipeline. The FIFO can be used to store overflowed data temporarily.

4.2 Parallel Radix-16 MM-Cell Design

The design of MM Cell is shown in Fig. 2(b). This is a high-radix design. In this paper, we implement radix-16 ($k = 4$) design of MM Cell.

As shown in algorithm 3, the function of MM Cell is:

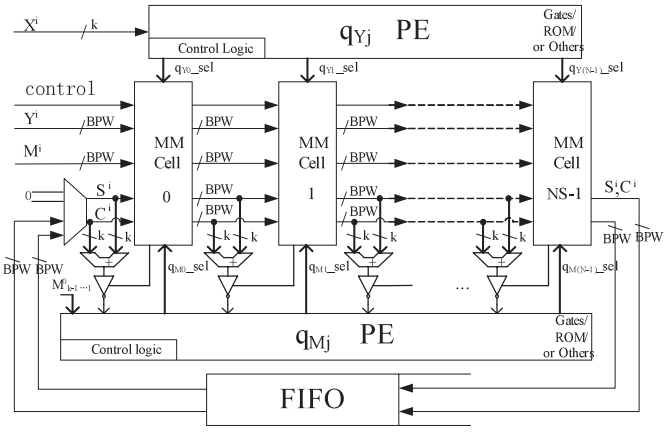
$$(C^i, S^i) = S^i + C^i + q_{Yj} \times Y^i + q_{Mj} \times M^i \tag{2}$$

While using proposed dataflow in Fig. 1(c), the input data (S^i, C^i, Y^i, M^i) becomes ($2^{jk}S^i$, $2^{jk}C^i$, $2^{(j+1)k}Y^i$, $2^{jk}M^i$) in the jth pipeline stage. The input q_{Yj}_sel and q_{Mj}_sel are the select signal for multiplexer, which is generated from q_{Yj} and q_{Mj} by q_{Yj} PE and q_{Mj} PE.

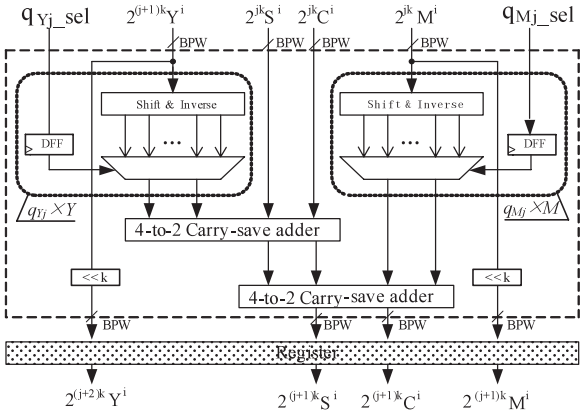
The implementation of $q_{Yj} \times Y^i$ is as following: Firstly, splitting q_{Yj} into two numbers which is power of 2. Secondly, shifting Y^i to get two components of $q_{Yj} \times Y^i$ based on these two numbers. Finally, adding these two components with (S^i, C^i) by using 4-to-2 carry-save adder. For example, while $q_{Yj} = 6$, q_{Yj} can be split into 2 and 4. Then, $6Y^i$ can be represented as ($2Y^i + 4Y^i$). The inputs of 4-to-2 carry-save adder are ($2Y^i$, $4Y^i$, S^i, C^i). Because $2Y^i$ and $4Y^i$ can be easily generated by left-shifting of Y^i, the multiplication of $6 \times Y^i$ can be avoided. $q_{Mj} \times M^i$ is implemented as same as $q_{Yj} \times Y^i$. The *shift & Inverse* modules in Fig. 2(b) are used for this purpose.

Considering Eq. (1), while using radix-16, it becomes:

$$\begin{aligned} q_{Yj} &= Booth(X_{j+3\ldots j-1}) \\ &= -2^4 X_{j+3} + 2^3 X_{j+2} + 2^2 X_{j+1} + 2X_j + X_{j-1} \end{aligned} \tag{3}$$



(a) Hardware-reused Montgomery Multiplier Architecture



(b) Parallel Radix-2^k MM Cell Design

Fig. 2 Proposed hardware architecture.

Table 2 Mapping table of q_{Mj} [0, 15] to q_{Mj}' [−7, 8].

q_{Mj}	q_{Mj}'	q_{Mj}	q_{Mj}'	q_{Mj}	q_{Mj}'	q_{Mj}	q_{Mj}'
0	0	4	4	8	8	12	-4
1	1	5	5	9	-7	13	-3
2	2	6	6	10	-6	14	-2
3	3	7	7	11	-5	15	-1

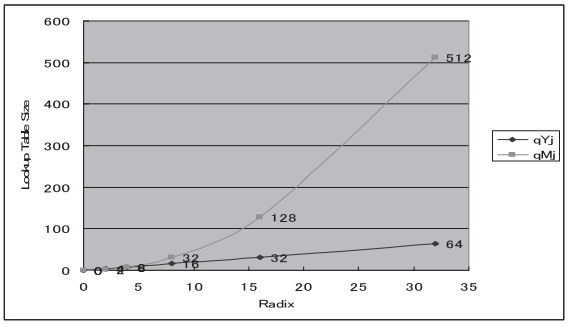


Fig. 3 Table size of q_{Yj} PE and q_{Mj} PE.

Table 3 $M^0_{3...0}$ to InvM.

$M^0_{3...0}$	InvM	$M^0_{3...0}$	InvM
0001	1111	1001	0111
0011	0101	1011	1101
0101	0011	1101	1011
0111	1001	1111	0001

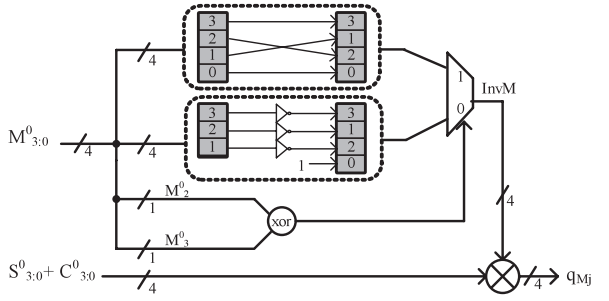


Fig. 4 Implementation of q_{Mj}.

The range of q_{Yj} is [−8, 8]. All of the number in this range can be split into two components, which is power of 2. For q_{Mj} under radix-16,

$$q_{Mj} = (S^0_{3...0} + C^0_{3...0}) \times (16 - (M^0)^{-1}_{3...0}) \bmod 16 \qquad (4)$$

The range of q_{Mj} is [0, 15]. Unfortunately, 11 and 13 can't satisfy the requirement. In order to deal with this problem, we propose a mapping table from [0, 15] to [−7, 8] in Table 2, which can be equivalently used for q_{Mj}.

4.3 Very Low Complex Implementation of q_{Mj}

As shown in Eqs. (3) and (4), q_{Mj} is much more complex than q_{Yj}. Normally, q_{Yj} PE and q_{Mj} PE are directly implemented by lookup table. The size of look up table is increased exponentially to radix number. This effect can be illustrated in Fig. 3. While using radix 16, the table size of q_{Mj} is 4 times of q_{Yj}. In order to reduce the cost of q_{Mj} calculation, we present a very low complex implementation of q_{Mj} in this paper.

Considering Eq. (4), we use Inv$M$ to present $(16 - (M^0)^{-1}_{3...0})$. Table 3 shows the mapping from $M^0_{3...0}$ to Inv$M$. As modulus M is an odd number, there is 8 different values of $M^0_{3...0}$.

All of these values can be divided into two groups: {(0001, 1111), (0111, 1001)} and {(0011, 0101), (1011, 1101)}. Here (A, B) means a pair of ($M^0_{3:0}$, InvM) or (InvM, $M^0_{3:0}$).

In the first group:

$$\text{Inv}M = \sim M^0_{3:0} + 1 \qquad (5)$$

In the second group:

$$\text{Inv}M = \{M^0_3, M^0_1, M^0_2, M^0_0\} \qquad (6)$$

The difference of group 1 and group 2 is:

Group 1: M^0_2 xor $M^0_1 = 0$

Group 2: M^0_2 xor $M^0_1 = 1$

For example, (0001) is in group1, 0 xor 0 is equal to 0. (0011) is in group2, 0 xor 1 is equal to 1.

Based on above analysis, the q_{Mj} can be implemented as Fig. 4 shows. Because modulus M is an odd number, Eq. (5) can be further presented as:

$$\mathrm{Inv}M = \sim M^0_{3:0} + 1 = \{\sim M^0_{3,} \sim M^0_{2,} \sim M^0_{1,} 1\} \tag{7}$$

After q_{Mj} is calculated, the q_{Mj}_sel can be calculated by using a small size lookup table as same as q_{Yj}_sel.

5. Analysis and Experimental Results

Based on proposed dataflow in Fig. 1(c), the total number of clock cycles to do Montgomery multiplication is shown in Eq. (8),

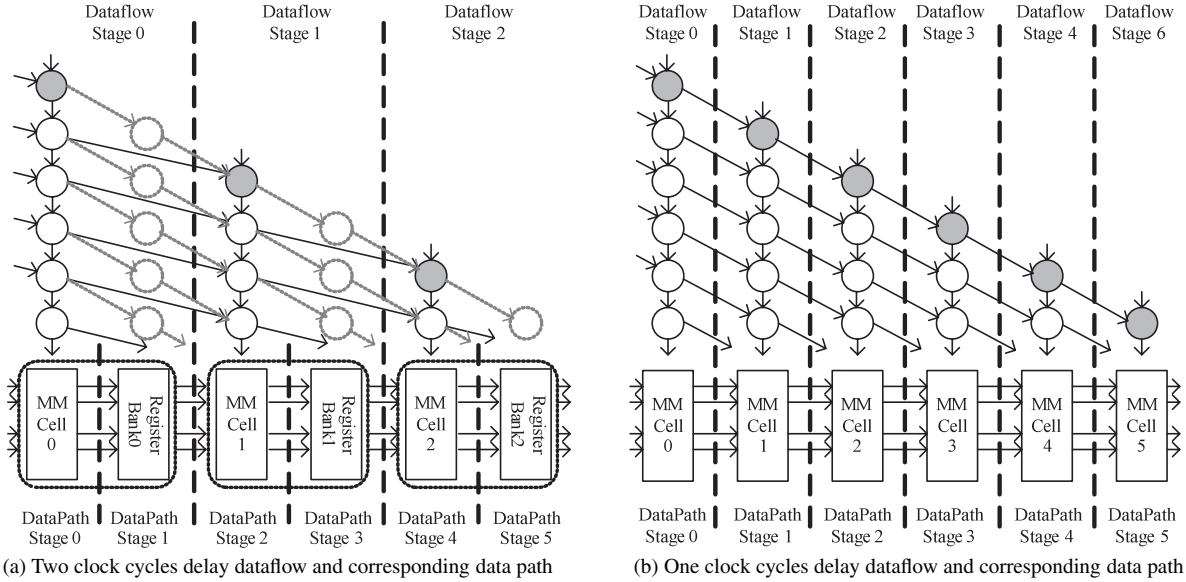
$$T_{CLKs} = \begin{cases} \left\lceil \frac{N}{k \cdot NS} \right\rceil \left(NS + \left\lceil \frac{k \cdot NS}{BPW} \right\rceil + 1\right) + NW + \left\lceil \frac{k \cdot NW}{BPW} \right\rceil + 1, NW \le NS \\ \left\lceil \frac{N}{k \cdot NS} \right\rceil \left(NW + \left\lceil \frac{k \cdot NS}{BPW} \right\rceil + 1\right) + NS + \left\lceil \frac{k \cdot NS}{BPW} \right\rceil, NW > NS \end{cases} \tag{8}$$

The meaning of notations in this equation can be found in Table 1. $\left\lceil \frac{N}{k \cdot NS} \right\rceil$ is the number of loops in pipeline. $\left\lceil \frac{k \cdot NS}{BPW} \right\rceil$ and $\left\lceil \frac{k \cdot NW}{BPW} \right\rceil$ are the extra clock cycles overhead for our clock-saving dataflow. As shown in Fig. 1(c), the product (S, C) is multiplied by 2^k in every stage. When it increases to 2^{BPW}(S, C), the LSW (Least-Significant-Word) of product is 0, so this all-zero LSW needs 1 extra clock cycle to be eliminated.

There are two cases of this equation. While $NW \le NS$, all of the words of Y, M can be loaded in the pipeline. The number of needed cycles is mainly decided by the NS. While $NW > NS$, the operand Y, M will be overflowed in the pipeline. In this case, the FIFO (shown in Fig. 2(a)) can be used to store overflowed data, and the number of cycles is mainly decided by the NW.

Table 4 Clock cycles comparison of different dataflows. (BPW = 32)

N	NS		NW	This Paper Clock Cycles (Radix-16)	Tenca-Koç Dataflow (Radix-16)		Herris Dataflow (Radix-2)	
	Two clock Cycles Delay Dataflow	One clock Cycle Delay Dataflow			Clock Cycles	Reduced Cycles (%)	Clock Cyels	Reduced Cycles (%)
512	4	8	16	297	552	46.2%	1111	73.3%
	8	16	16	171	289	40.8%	575	70.3%
	16	32	16	167	281	40.6%	559	70.1%
1024	8	16	32	578	1072	46.1%	2159	73.2%
	16	32	32	333	561	40.6%	1119	70.2%
	32	64	32	329	553	40.5%	1103	70.2%
2048	16	32	64	1140	2112	46.0%	4255	73.25%
	32	64	64	657	1105	40.5%	2207	70.2%
	64	128	64	653	1097	40.5%	2191	70.2%



(a) Two clock cycles delay dataflow and corresponding data path

(b) One clock cycles delay dataflow and corresponding data path

Fig. 5 Comparison of dataflow and corresponding data path.

Table 5 Performance comparison with other's work.

Ref.	Tech	Area (Gates/LUTs)	Frequency (MHz)	Scalable	Radix	Delay Cycles in Dataflow	1024-bit RSA Throughput (Kbps)
[8]	0.5 μm	28k Gates	64 MHz	Yes	8	2	22 Kbps
[9]	0.25 μm	100k Gates	125 MHz	Yes	16	2	143 Kbps
[10]	FPGA	5598 LUTs	144 MHz	Yes	2	1	62.5 Kbps
[11]	FPGA	2847 LUTs + 32 Mults + 5n RAM	102 MHz	Yes	2^{16}	4	152 Kbps
[12]	0.18 μm	150k	300 MHz	Yes	2	2	100 Kbps
This Paper	0.25 μm	130k	180 MHz	Yes	16	1	352 Kbps

Table 4 shows the clock cycles comparison of our dataflow with Tenca-Koç dataflow and Herris dataflow. It shows that our dataflow achieves much less clock cycles than their dataflow.

In this table, different key length and stages are used to calculate clock cycles for each dataflow. The BPW is equal to 32. Because our dataflow and Tenca-Koç dataflow are high-radix dataflow, we use radix-16 for these two dataflows in Table 4, and Herris dataflow uses radix-2.

In Table 4, the NS of one clock cycle delay dataflow is half of two clock cycles delay dataflow. The reason is illustrated in Fig. 5. Each *Dataflow Stage* of two clock cycles delay dataflow in Fig. 5(a) actually includes two *DataPath Stages*. One is *MM Cell* stage for operation of Eq. (2), the other is *Register Bank* stage for storing (Y^i, M^i, S^i, C^i). The dashed cycles in Fig. 5(a) represent the registering stages in the dataflow. For fairly comparing, the NS of one clock cycle delay dataflow should be two times of two clock cycles delay dataflow.

The ASIC implementation of this work uses NS = 32, BPW = 32, Radix = 16. We use HHNEC 0.25 μm CMOS standard cell library and Synopsys EDA tools to do ASIC design.

Table 5 shows the performance comparison of this paper's result with other's work. [8] is a radix 8 design proposed by Tenca-Koç, [9] is an improved radix 16 design. By using proposed algorithm and the parallel data path design, this design's frequency is higher than [9] under the same radix number. [10] is a FPGA implementation by using one clock cycle delay dataflow, [11] is a very high radix design (radix 2^{16}) using multiplier and RAM which is embedded in FPGA, [12] is a radix-2 scalable design which using 2 clock cycles delay dataflow.

Normally, radix-2^k design can achieve about k times of performance than radix-2 design. One clock cycle delay dataflow can achieve about 2 times of performance than two clock cycles delay dataflow. From the Table 5, our design uses radix-16 with one clock cycle delay dataflow. It achieves much higher performance than other's design.

6. Conclusion

This paper proposes a high speed design of Montgomery multiplier. By using proposed algorithm, it parallelizes the data path and shortens the critical path. By using proposed clock-saving dataflow, it reduces the total clock cycles of multiplication to a very small number. Finally, a very compact hardware architecture design is proposed to reduce hardware cost and improve the performance. The experimental results show that this design achieves very high performance with low hardware cost. This design is very suitable for high-speed RSA or ECC implementation.

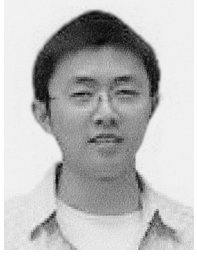
Acknowledgement

This research is supported by CREST, JST.

References

[1] R.L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public key crypto- systems," Commun. ACM, vol.21, no.2, pp.120–126, 1978.
[2] N. Koblitz, "Elliptic curve cryptosystems," Mathematics of Computation, vol.48, no.177, pp.203–209, 1987.
[3] V. Miller, "Use of elliptic curves in cryptography," Proc. CRYPTO 85, pp.417–426, 1985.
[4] P.L. Montgomery, "Modular multiplication without trial division," Mathematics of Computation, vol.44, no.170, pp.519–521, April 1985.
[5] C. Koc, T. Acar, and B. Kaliski, "Analyzing and comparing Montgomery multiplication algorithms," IEEE Micro, vol.16, no.3, pp.26–33, June 1996.
[6] A.F. Tenca and C.K. Koc, "A scalable architecture for modular multiplication based on Montgomery's algorithm," IEEE Trans. Comput., vol.52, no.9, pp.1215–1221, Sept. 2003.
[7] A.F. Tenca, G. Todorov, and C.K. Koc, "High-radix design of a scalable modular multiplier," Cryptographic Hardware and embedded Systems-CHES 2001, Lect. Notes Comput. Sci., no.2162, pp.189–205, May 2001.
[8] G. Todorov, ASIC design, implementation and analysis of a scalable high-radix Montgomery multiplier, M.S. Thesis, Oregon State University, June 2001.
[9] Y. Fan, X.Y. Zeng, Y. Yu, G. Wang, H. Deng, and Q.L. Zhang, "High speed radix-16 design of a scalable Montgomery multiplier," Proc. 6th International Conference on ASIC-ASICON 2005, vol.1, no.24-27, pp.153–157, Oct. 2005.

[10] D. Harris, R. Krishnamurthy, M. Anders, S. Mathew, and S. Hsu, "An improved unified scalable radix 2 Montgomery multiplier," Proc. 17th IEEE Symposium on Computer Arithmetic, pp.172–178, June 2005.
[11] K. Kelley and D. Harris, "Very high radix scalable Montgomery multipliers," Proc. Fifth International Workshop on System-on-Chip for Real-Time Applications, pp.400–404, July 2005.
[12] C.H. Wang, C.P. Su, C.T. Huang, and C.W. Wu, "A word-based RSA crypto-processor with enhanced pipeline performance," Proc. 2004 IEEE Asia-Pacific Conference on Advanced System Integrated Circuits, pp.218–221, Aug. 2004.



Yibo Fan received the B.E. degree in electronics and engineering from Zhejiang University, China in 2003 and M.S. degree in Microelectronics from Fudan University, China in 2006. Currently, he is a Ph.D. candidate in Graduate School of Information, Production and Systems, Waseda University, Japan. His research interesting includes information security, video coding and associated VLSI architecture.

Takeshi Ikenaga received his B.E. and M.E. degrees in electrical en- gineering and the Ph.D. degree in infor-mation & computer science from Waseda University, Tokyo, Japan, in 1988, 1990, and 2002, respectively. He joined LSI Laboratories, Nippon Telegraph and Telephone Corporation (NTT) in 1990, where he has been undertaking research on the design and test methodologies for high-performance ASICs, a real-time MPEG2 encoder chip set, and a highly parallel LSI & System design for image-understanding processing. He is presently an associate professor in the system LSI field of the Graduate School of Information, Production and Systems, Waseda University. His current interests are application SOC for image, security and network processing. Dr. Ikenaga is a member of the IPSJ and the IEEE. He received the IEICE Research Encouragement Award in 1992.

Satoshi Goto was born on January 3rd, 1945 in Hiroshima, Japan. He received the B.E. and M.E. degree in Electronics and Communication Engineering from Waseda University in 1968 and 1970, respectively. He also received the Dr. of Engineering from the same university in 1981. He is IEEE fellow, member of Academy Engineering Society of Japan and professor of Waseda University. His research interests include LSI system and Multimedia System.