Systematic design of high-radix Montgomery multipliers for RSA processors

Atsushi Miyamoto, Naofumi Homma and Takafumi Aoki
Graduate School of Information Sciences, Tohoku University, Japan
{miyamoto, homma}@aoki.ecei.tohoku.ac.jp

Akashi Satoh
Research Center for Information Security, AIST, Japan
akashi.satoh@aist.go.jp

***Abstract*— The present paper proposes a systematic design approach to provide the optimal high-radix Montgomery multipliers for an RSA processor satisfying user requirements. We introduces three multiplier-based architectures using different intermediate-data forms ((i) single form, (ii) semi carry-save form, and (iii) carry-save form), and combined them with a wide variety of arithmetic components. Their radices are also parameterized from 2^8 to 2^{64}. A total of 202 designs for 1,024-bit RSA processors were obtained for each radix, and were synthesized using a 90-nm CMOS standard cell library. The smallest design of 0.9 Kgates with 137.8 ms/RSA to the fastest design of 1.8 ms/RSA at 74.7 Kgates were then obtained. In addition, the optimal design to meet the user requirements can be easily obtained from all the combinations. In addition to choosing the datapath architecture, the arithmetic component, and the radix parameters, the proposed systematic approach can also adopt other process technologies.**

I. Introduction

Modular exponentiation is the most important arithmetic operation for public-key cryptosystems, such as the RSA scheme, the ElGamal scheme, and the Diffie-Hellman key agreement protocol. Modular exponentiation is performed by repeating modular multiplication and squaring operations with large operands (1,024∼4,096 bits), and thus optimization of modular multiplication is essential in order to achieve high-performance public-key cryptosystem designs. The Montgomery multiplication algorithm [1] is widely used for practical hardware and software implementations because of its high-speed capability.

Many computation techniques and hardware architectures have been proposed for Montgomery multiplication [2]–[8]. Among them, the radix-2 algorithms proposed in [7] are primarily implemented with long k-bit adders to scan the k-bit operand bit-by-bit in a straightforward manner. Hardware architectures have large fan-out signals and large wire delays for long operands. These drawbacks can be reduced by systolic array architectures [6][8] with multiple operation units. However, these architectures are usually tailored for fixed-precision computations and cannot respond flexibly to changes in operand size. To deal with variable-length data, a radix-2 architecture was proposed [3][5] in which a k-bit operand is divided into mr-bit word blocks, and k-bit addition is performed by repeating r-bit addition m times. These radix-2 architectures are quite simple, but have difficulty in improving the performances of circuit area and efficiency. A high-radix architecture using a 64-bit × 64-bit multiplier was proposed in [2] to achieve higher circuit efficiency. The performance of such a multiplier-based architecture depends heavily on the datapath structure, and varies with the structure of the arithmetic components, but previous papers focused on the designs of their own architectures. These architectures are optimal for some design parameters such as size and speed, but the best design point in practical use varies with the application and the user requirements. Therefore, in order to provide the design that best satisfies these requirements, a systematic study from the datapath-architecture level to the arithmetic-component level is indispensable from a practical standpoint.

The present paper proposes a systematic design of high-performance high-radix Montgomery multipliers, in which three types of datapath architectures are combined with a wide variety of arithmetic components with a parameterized radix. Three datapath architectures that employ three intermediate data forms ((i) single form, (ii) semi carry-save form, and (iii) carry-save form) are newly introduced for architecture-level design. To demonstrate the capability of the proposed approach, 202 Montgomery multipliers are exhaustively generated, and 1,024-bit RSA processors with all of the multipliers are synthesized using a 90-nm CMOS standard cell library. Their size and speed features are then displayed graphically and analyzed so that a user can easily choose the best combination of datapath architecture, arithmetic unit, and radix.

II. High-radix Montgomery multiplier

A. Montgomery multiplication algorithm

Given two large integers X and Y, the Montgomery multiplication algorithm performs the following operation:

$$Z = XYR^{-1} \bmod N, \tag{1}$$

where $R = 2^k$ and the modulus N is an integer in the range $2^{n-1} < N < 2^n$ such that $\gcd(R,N) = 1$. For cryptographic applications, N is usually a prime number or a product of primes, and thus satisfies the condition easily. In addition, the k-bit integers X, Y, R, and N satisfy the following condition:

$$0 \leq X, Y < N < 2^k = R. \tag{2}$$

ALGORITHM 1 shows the original Montgomery multiplication algorithm [1], which replaces a modular division-by-N with a k-bit right shift operation. Equation (1) can

978-1-4244-2658-4/08/$25.00 ©2008 IEEE

ALGORITHM 1
MONTGOMERY MULTIPLICATION

Input:	$X,Y,N,R(=2^k)$, $W=-N^{-1} \bmod R$
Output:	$Z=XYR^{-1} \bmod N$

```
1 :  t := XY·W mod R;
2 :  Z := (XY + tN)/R;
3 :  if (Z > N) then Z := Z − N;
```

ALGORITHM 2
HIGH-RADIX MONTGOMERY MULTIPLICATION

Input:	$X=(x_{m-1},...,x_1,x_0)_{2^r}$, $Y=(y_{m-1},...,y_1,y_0)_{2^r}$, $N=(n_{m-1},...,n_1,n_0)_{2^r}$, $w=-N^{-1} \bmod 2^r$
Output:	$Z=XY2^{-r\cdot m} \bmod N$

```
1 :  Z := 0;
2 :  for i = 0 to m − 1                      – Loop1
3 :     c := 0;
4 :     t_i := (z_0 + x_i y_0)w mod 2^r;
5 :     for j = 0 to m − 1                   – Loop2
6 :        Q := z_j + x_i y_j + t_i n_j + c;
7 :        if (j ≠ 0) then z_{j−1} := Q mod 2^r;
8 :        c := Q/2^r;
9 :     end for
10:     z_{m−1} := c;
11:  end for
12:  if (Z > N) then Z := Z − N;
```

be calculated by one multiplication and a k-bit right shift operation if the lowest k bits of XY are equal to 0. For this purpose, a multiple of N is added to XY in this algorithm. The final result is not changed by the addition because (1) is in modulo N arithmetic. In addition, the coefficient t is generated in advance using a pre-computed number W.

Public-key cryptosystems, such as the RSA scheme, use a very long key-length k of over 1,000 bits. In the high-radix Montgomery algorithm [2], a k-bit operand is divided into m blocks ($k=r\cdot m$) in order to use a normal r-bit $\times$ r-bit multiplier. The k-bit operand X can be represented by r-bit words x_i ($0\leq i\leq m-1$) as follows:

$$X = x_{m-1}2^{r(m-1)} + ... + x_1 2^r + x_0. \quad (3)$$

For simplification, we use the following notation.

$$X = (x_{m-1},...,x_1,x_0)_{2^r}. \quad (4)$$

ALGORITHM 2 shows the high-radix Montgomery multiplication algorithm in which each operand is divided into smaller words, and are processed in nested loops (Loop1 for x_i and Loop2 for y_j, n_j), respectively. The size of temporary variable Q is $2r$ bits and its upper r bits and lower r bits are stored separately in the word z_j and the intermediate carry c, respectively. Finally, the stored value $Z=(z_{m-1},...,z_1,z_0)_{2^r}$ is the output of this algorithm.

In the above algorithm, the most critical operation for circuit delay and area is the multiply accumulation at Line 6, which consists of two multiplication operations and three addition operations. In order to improve operation efficiency, the multiply accumulation is divided into two three-term operations [9], and each operation is usually performed by a multiplier in the datapath. As a result, the design of

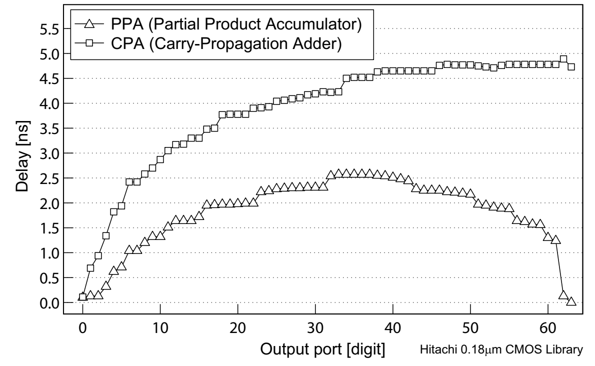


Fig. 1. Output arrival profile of conventional array multiplier.

such multiplier is of major importance for the hardware implementation of the high-radix Montgomery multiplier.

B. Proposed Montgomery multiplication algorithms

In this section, we first discuss the delay profile of the output signals from a parallel multiplier to design high-Radix Montgomery multiplication algorithms for high-performance hardware designs. In the multiplier, a Carry Propagation Adder (CPA) follows a Partial Product Adder (PPA) to generate the final product in twos complement form from carry-save form. (See the following section for more details.) Fig.1 shows the delay profile of a 32-bit $\times$ 32-bit parallel multiplier, where the horizontal axis denotes the bit position from LSB to MSB, and the vertical axis shows the output signal delay time. The triangle and square symbols indicate the signal delay times for output bits of the PPA and CPA, respectively. As shown in Fig.1, the long carry-propagation chain of the CPA causes longer delays for higher bits. In contrast, the position of the slowest signal for the PPA is around the middle (32nd) bit, where the maximum number of operands exist. This delay profile suggests that it would be possible to improve the performance of the multiplier by optimizing the output data form to minimize the delay time corresponding to the word size of the CPA.

Based on the delay profile, we design three types of high-radix Montgomery multiplication algorithms for hardware implementations with different intermediate-data forms: (i) single form (Type-I), (ii) semi carry-save form (Type-II), and (iii) carry-save form (Type-III). These algorithms are variations of ALGORITHM 2 using Finely Integrated Operand Scanning (FIOS) method in [9]. ALGORITHMS 3∼5 show the proposed algorithms corresponding to the above types, where the tuple (c,z) indicates $c\cdot 2^r+z$, and v is the carry bit used in the i-th loop.

ALGORITHM 3 (Type-I) is based on a straightforward algorithm with a three-term multiply-addition operation such as $z+xy+c$. The arithmetic operation $Q:=z_j+x_iy_j+t_in_j+C$ in ALGORITHM 2 is divided into two steps: a multiplication step ($(c,z):=z_j+x_iy_j+C$) and a reduction step ($(c,z):=z_j+t_in_j+C$). In order to avoid increasing the number of variables, which are mapped to register arrays in hardware, this algorithm does not use a carry-save form for the intermediate-data. The multiply-addition result is a $2r$-bit value, and its upper half is the carry c fed to the

ALGORITHM 3 (TYPE-I)

1 : $Z := 0; \quad v := 0;$
2 : **for** $i = 0$ **to** $m-1$
4 : $\quad (c_a, z_0) := z_0 + x_i y_0;$
5 : $\quad t_i := z_0 w \bmod 2^r;$
6 : $\quad (c_b, z_0) := z_0 + t_i n_j;$
7 : $\quad$ **for** $j = 1$ **to** $m-1$
8 : $\quad\quad (c_a, z_j) := z_j + x_i y_j + c_a;$
9 : $\quad\quad (c_b, z_{j-1}) := z_j + t_i n_j + c_b;$
10: $\quad$ **end for**
11: $\quad (v, z_{m-1}) := c_a + c_b + v;$
12: **end for**
13: **if** $(Z > N)$ **then** $Z := Z - N;$

ALGORITHM 4 (TYPE-II)

1 : $Z := 0; \quad v := 0;$
2 : **for** $i = 0$ **to** $m-1$
4 : $\quad (cs1_a + cs2_a + ec_a, z_0) := z_0 + x_i y_0;$
5 : $\quad t_i := z_0 w \bmod 2^r;$
6 : $\quad (cs1_b + cs2_b + ec_b, z_0) := z_0 + t_i n_j;$
7 : $\quad$ **for** $j = 1$ **to** $m-1$
8 : $\quad\quad (cs1_a + cs2_a + ec_a, z_j) := z_j + x_i y_j + cs1_a + cs2_a + ec_a;$
9 : $\quad\quad (cs1_b + cs2_b + ec_b, z_{j-1}) := z_j + t_i n_j + cs1_b + cs2_b + ec_b;$
10: $\quad$ **end for**
11: $\quad (v, z_{m-1})$
$\quad\quad := cs1_a + cs2_a + ec_a + cs1_b + cs2_b + ec_b + v;$
12: **end for**
13: **if** $(Z > N)$ **then** $Z := Z - N;$

ALGORITHM 5 (TYPE-III)

1 : $Z := 0; \quad v := 0;$
2 : **for** $i = 0$ **to** $m-1$
3 : $\quad (cs1_a + cs2_a, zs1 + zs2) := z_0 + x_i y_0;$
4 : $\quad t_i := (zs1 + zs2) w \bmod 2^r;$
5 : $\quad (cs1_b + cs2_b, zs1 + zs2) := zs1 + zs2 + t_i n_j;$
6 : $\quad (ec, z_0) := zs1 + zs2;$
7 : $\quad$ **for** $j = 1$ **to** $m-1$
8 : $\quad\quad (cs1_a + cs2_a, zs1 + zs2) := z_j + x_i y_j + cs1_a + cs2_a;$
9 : $\quad\quad (cs1_b + cs2_b, zs1 + zs2) := zs1 + zs2 + t_i n_j + cs1_b + cs2_b + ec;$
10: $\quad\quad (ec, z_{j-1}) := zs1 + zs2;$
11: $\quad$ **end for**
12: $\quad (v, z_{m-1}) := cs1_a + cs2_a + cs1_b + cs2_b + ec + v;$
13: **end for**
14: **if** $(Z > N)$ **then** $Z := Z - N;$

multiply-addition in the next cycle. The lower half is used as an intermediate sum z.

ALGORITHM 4 (Type-II) uses the carry-save form only for the carry c, where $cs1$ and $cs2$ are the r-bit carry-save values generated by the PPA operation, and ec is the 1-bit value from the previous CPA operation. The carry is given as $c = cs1 + cs2 + es$.

ALGORITHM 5 (Type-III) uses the carry-save form for both intermediate sum z and carry c, where $cs1$ and $cs2$ are the carries, and $zs1$ and $zs2$ are the sums. No CPA operation is performed during the iteration cycles. Extra CPA operations are performed at the end of the inner loops at Lines 6 and 10. Type-II and Type-III algorithms using carry-save form can improve the operating frequency for hardware implementation, while they need extra registers to store the intermediate data in carry-save form.

C. Proposed datapath architectures

Fig.2 shows the proposed datapath architectures corresponding to ALGORITHMS 3∼5 (i.e., Type-I∼III). Each

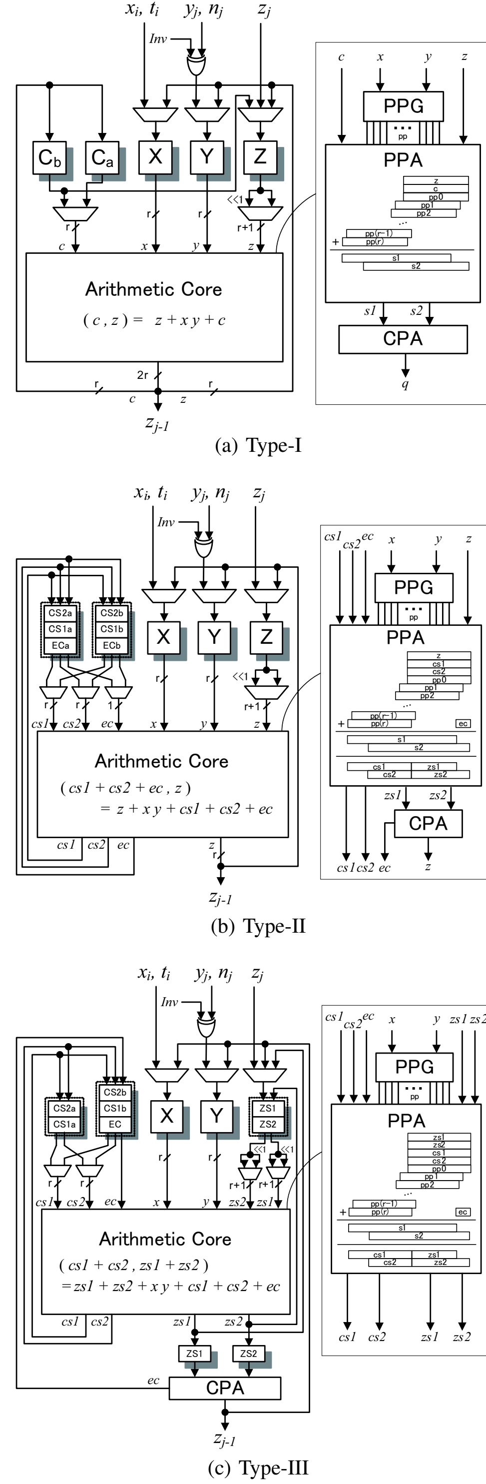


Fig. 2. Datapath architectures of Montgomery multiplication block.

datapath has an r-bit × r-bit multiply-accumulator called Arithmetic Core, which consists of three components: a Partial Product Generator (PPG), a Partial Product Accumulator (PPA) and a Carry Propagation Adder (CPA). The PPG stage first generates partial products from the multiplicand x and multiplier y in parallel. The PPA stage then performs multi-operand addition for all the generated partial products and other operands (c and z), and produces two outputs represented in carry-save form. Finally, the carry-save form

is converted to the corresponding binary output at CPA.

Type-I architecture has the simple datapath with the Arithmetic Core, which receives three r-bit inputs (x, y, z) and an r-bit carry c, and outputs a $2r$-bit result. The upper r-bit value is fed back to the Arithmetic Core as the carry c in the next cycle, and the lower r-bit value z stored into a register or a memory outside the core. This architecture requires the least number of registers and intermediate wires among the three architectures, and thus is suitable to for compact implementation. However, the Arithmetic Core has the longest critical path because of the $2r$-bit CPA operation.

Type-II architecture enhances the hardware efficiency defined as a product of the circuit delay and area. The Arithmetic Core has an r-bit CPA for the lower r-bit output from the PPA stage, and produces four outputs $cs1$, $cs2$, ec and z. The carry signals $cs1$ and $cs2$ in carry-save form generated by the PPA are fed back to the Arithmetic Core. On the other hand, the sum signals $z1$ and $z2$ from the PPA are fed to the following CPA and are converted to an r-bit output z and 1-bit carry ec. The size of the r-bit CPA is approximately half of the $2r$-bit CPA in Type-I architecture, and thus the critical path is also halved. As a result, the entire critical path is shortened by 25%, while the number of registers is increased.

Type-III architecture has the fastest datapath without any carry-propagation in Arithmetic Core. Both the carry and sum signals by the PPA, that is, $cs1$, $cs2$, $zs1$ and $zs2$, are fed back into the core in carry-save form. The CPA is performed outside the core at the end of m iteration cycles. The critical path is approximately halved compared with the Type-I architecture, while the largest number of registers is required to handle two pairs of carry-save signals.

III. RSA PROCCESOR

The RSA cryptosystem employs modular exponentiation for encryption and decryption as $Z = X^E \bmod N$. Basically, there are two types of efficient exponentiation algorithms: binary methods and k-ary methods [10]. Among them, we focus on the left-to-right binary method that scans the bit pattern of the exponent E from left to right. The method is widely used in practical applications such as smartcards and embedded devices, because of its simplicity and low resource consumption. ALGORITHM 6 shows a left-to-right binary method with the Montgomery multiplication algorithm. Here, *MontMult* indicates the Montgomery multiplication $XYR^{-1} \bmod N$, and the operations shown in lines 1-2 ($W := -N^{-1} \bmod R$ and $Y := XR \bmod N$) indicate the precomputations for Montgomery multiplications, respectively. This algorithm always performs a squaring at line 3, regardless of the scanned bit value, but the multiply operation at line 5 is only executed if the scanned bit is 1.

Fig.3 shows a block diagram of our RSA processor that consists of five components: Multiplication Block, Sequencer Block, Memory, Data Counter, and Key Shift. Multiplication Block performs the multiply-addition operations repeatedly according to the exponent bits. Multiplication Block is implemented as one of the three multipliers in Fig.2. For example,

ALGORITHM 6
MODULAR EXPONENTIATION WITH *MontMult*

Input:	X, N $E = (e_{k-1}, \ldots, e_1, e_0)_2$,
Output:	$Z = X^E \bmod N$

1 : $W := -N^{-1} \bmod R$;
2 : $Y := XR \bmod N$;
3 : $Z := R \bmod N$;
4 : **for** $i = k-1$ **downto** 0
5 : $\quad Z :=$ *MontMult* (Z, Z, N, W); – squaring
6 : $\quad$ **if** (e_i = 1) **then**
7 : $\quad\quad Z :=$ *MontMult* (Z, Y, N, W); – multiplication
8 : $\quad$ **end if**
9 : **end for**
10: $Z :=$ *MontMult* (Z, 1, N, W);

Fig. 3. RSA processor airchitecture.

the number of clock cycles for Type-I is given as follows:

$$t1 = km + k + m + 2r + 2, \tag{5}$$

$$t2 = 2m^2 + 4m + 1, \tag{6}$$

$$t3 = t1 + (\frac{3}{2}k + 1) \cdot t2, \tag{7}$$

where $t1$, $t2$, and $t3$ indicate cycles for the precomputation at lines 1-2 of ALGORITHM 6, Montgomery multiplication, and modular exponentiation (RSA operation), respectively. If the key size is k = 1,024 bits and the data bus size is r = 32, then the number of partitioned blocks is m = 32. The Montgomery multiplication requires 2,177 cycles, and the total number of clock cycles for the modular exponentiation is approximately 3,400K. The number of cycles for the Type-II and Type-III architectures increase a few percent over the Type-I architecture for some extra addition operations.

IV. PERFORMANCE EVALUATION

The systematic design of RSA processors based on the above-mentioned Montgomery multiplier architectures is described in this section. The proposed approach can obtain a wide variety of Montgomery multipliers combining three datapath architectures with arithmetic algorithms from radix-2^8 to radix-2^{64}. Table I shows a set of arithmetic algorithms handled in the experimental system. For efficient PPA algorithms, we used a (4;2) compressor and (7,3) counter trees in addition to conventional algorithms based on (3,2) counters. In addition, we used three parallel-prefix adders (Kogge-Stone, Brent-Kung, and Han-Carlson adders) for high-speed

TABLE I

PPA AND CPA ALGORITHMS

PPA algorithms	CPA algorithms
(3:2) counter tree	Ripple carry adder
Array	Carry look-ahead adder
Wallace tree	Ripple-block CLA
Balanced delay tree	Block CLA
Overturned-stairs tree	Kogge-Stone adder
Dadda tree	Brent-Kung adder
(4;2) compressor tree	Han-Carlson adder
(7,3) counter tree	Conditional sum adder
	Fixed-block-size carry-skip adder
	Variable-block-size carry-skip adder

designs, a conditional-sum adder for balanced designs, and two carry-skip adders for compact designs, in addition to the conventional CPA algorithms such as the ripple carry and the carry look-ahead adder. In total, we synthesized and evaluated 202 RSA processors for each radix.

All the RSA processors were designed in the Verilog-HDL language and were synthesized by Synopsys Design Compiler with the STMicroelectronics 90-nm CMOS standard cell library (1.2-volt version) [11]. The circuit area of the datapath was evaluated based on a two-way NAND equivalent gate. (Memories and sequencer modules were not included.) The delay time was under the worst-case conditions.

Fig.4 shows the characteristics of 202 designs using radix-2^{32}, where the horizontal and vertical axes are the delay time and the gate count, respectively. The dots plotted lower and further to the left show better performance. As shown in this figure, the performances of obtained Montgomery multipliers are heavily dependent on the architecture and arithmetic components, and Type-I$\sim$-III architectures have significant advantages in size, balance, and delay, respectively. More precisely, a Type-I processor with a (7,3) counter tree and a variable-block-size carry-skip adder achieved the smallest gate count of 8.5 Kgates. A Type-II processor with a (4;2) compressor tree and the Brent-Kung adder then achieved the smallest area-time product of 29.9 ns·Kgates. In contrast, a Type-III processor with a (7,3) counter tree and a conditional-sum adder achieved the shortest critical path of 1.6 ns.

Table II shows the performances variation with the radices from 2^8 to 2^{64}, where Delay, Balance, and Area correspond to the arithmetic components shown in Fig.4, and the three rows in bold font indicate the best performances among all the designs in terms of circuit delay, hardware efficiency, and circuit area. The columns for MM time and RSA time indicate the computation times of Montgomery multiplication and RSA operation, respectively. Conventional designs [2]–[6][8] are also shown at the bottom of Table II. Our design approach provides a very wide variety from the smallest area of 0.9 Kgates with the Type-I radix-2^8 processor to the shortest RSA operating time of 1.8 ms at 520 MHz with the Type-III radix-2^{64} processor. When the Chinese Remainder Theorem (CRT) technique is applied, the RSA operating time can be reduced to 0.9 ms. The highest hardware efficiency of 92.8 ms·Kgates was achieved by the Type-II radix-2^{16} processor. Thus, the top performance obtained by the proposed system is higher as compared to the conventional designs. As shown above, the wide variety of performance data set obtained from the exhaustive synthesis can provide the best RSA processor design to the meet requirements of the target application. In this experiment, only a cell-based design with a 90-nm CMOS standard cell library was investigated, but the performance varies greatly depending on the process technology, the library, and the synthesis parameter. However, the proposed approach does not depend on these conditions, and just synthesizes and depends only on synthesis, and collects the performance data to adopt in the new environment. Each design layer of the proposed system can be optimized independently, and thus the proposed system also allows easy adoption of new architectures or arithmetic components.

V. CONCLUSION

The present paper proposed a systematic approach to designing high-radix Montgomery multipliers for RSA processors. A number of RSA hardware architectures optimized for a few design parameters have been proposed, but it is not feasible to design all architectures independently to find the best design that meets the performance requirements for practical use. In contrast, the proposed approach provides the optimal Montgomery multiplier satisfying the requirements by combining three new datapath architectures using different intermediate-data forms ((i) single form, (ii) semi carry-save form, and (iii) carry-save form), a wide variety of arithmetic components, and radices ($2^8 \sim 2^{64}$). A wide variety of 1,024-bit RSA processors ranging from 0.9 Kgates@137.8 ms/RSA to the 74.8 Kgates@1.8 ms/RSA in a 90-nm CMOS standard cell library were obtained by exhaustive synthesis for all the combinations. Other than these two designs, a user can freely select the best design to fit their application from the combinations and can also choose other process technologies. In addition to the present approach from the datapath-architecture level to the arithmetic-component level, the performance of RSA processors can be improved at the cryptographic algorithm level, for example by the use of Chinese Remainder Theorem (CRT) and window methods. The Type-III processor with a 2^{64} radix can perform the RSA operation in less than 1.0 ms using the CRT. Further research to merge the algorithm level with the proposed system and to support other public-key cryptographic algorithms, such as elliptic curve cryptography, will be conducted.

REFERENCES

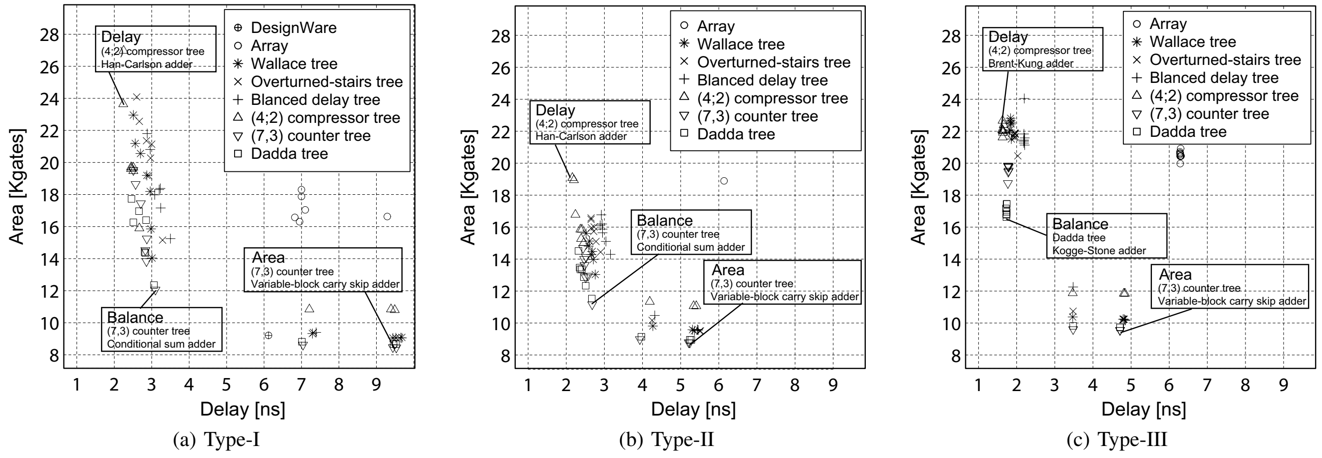
[1] P. L. Montgomery, "Modular multiplication without trial division," *Math. Comp.*, vol. 44, no. 170, pp. 519–521, 1985.

[2] A. Satoh and K. Takano, "A scalable dual-field elliptic curve cryptographic processor," *IEEE Trans. Comput.*, vol. 52, no. 4, pp. 449–460, 2003.

[3] 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, 2003.

[4] E. Savas, A. F. Tenca, and C. K. Koc, "A scalable and unified multiplier architecture for finite fields GF(p) and GF(2^m)," in *Proc. the 2nd Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. Springer-Verlag, 2000, pp. 277–292.

[5] D. Harris, R. Krishnamurthy, S. Mathew, and S. Hsu, "An improved unified scalable radix-2 Montgomery multiplier," in *Proc. the 17th IEEE Symp. Computer Arithmetic (ARITH)*. IEEE Computer Society, 2005, pp. 172–178.



(a) Type-I (b) Type-II (c) Type-III

Fig. 4. Synthesis results classified by PPA algorithms.

TABLE II

PERFORMANCE COMPARISON OF RSA PROCESSORS

Reference	Technology	Implementation Type	Radix	MAC design	Max. freq. [MHz]	Area [Kgates]	1,024bit MM time	1,024bit RSA time	RSA time × Area
This work	90 nm CMOS	I	2^8	Delay	704.2	2.426	47.25 μs	72.77 ms	176.62
				Area	**371.7**	**0.901**	**89.52 μs**	**137.86 ms**	**124.22**
				Balance	568.1	1.230	58.57 μs	90.20 ms	110.96
			2^{16}	Delay	546.4	7.096	15.46 μs	23.87 ms	169.41
				Area	203.6	2.523	41.48 μs	64.04 ms	161.63
				Balance	429.1	3.717	19.68 μs	30.39 ms	112.98
			2^{32}	Delay	446.4	23.640	4.87 μs	7.56 ms	178.86
				Area	104.9	8.453	20.74 μs	32.19 ms	272.10
				Balance	322.5	12.058	6.74 μs	10.47 ms	126.26
			2^{64}	Delay	364.9	78.447	1.58 μs	2.47 ms	194.27
				Area	55.2	30.837	10.44 μs	16.36 ms	504.75
				Balance	246.9	38.953	2.33 μs	3.66 ms	142.58
		II	2^8	Delay	751.8	1.755	44.43 μs	68.42 ms	120.09
				Area	581.3	1.054	57.46 μs	88.49 ms	93.32
				Balance	657.8	1.256	50.78 μs	78.20 ms	98.26
			2^{16}	Delay	549.4	5.092	15.49 μs	23.91 ms	121.81
				Area	337.8	2.689	25.19 μs	38.90 ms	104.61
				Balance	**505.0**	**3.568**	**16.85 μs**	**26.02 ms**	**92.84**
			2^{32}	Delay	462.9	19.083	4.77 μs	7.40 ms	141.25
				Area	190.8	8.777	11.57 μs	17.95 ms	157.62
				Balance	373.1	11.164	5.92 μs	9.18 ms	102.53
			2^{64}	Delay	398.4	68.690	1.48 μs	2.33 ms	160.06
				Area	103.5	31.420	5.72 μs	8.96 ms	281.80
				Balance	283.2	37.207	2.09 μs	3.27 ms	121.93
		III	2^8	Delay	900.9	2.317	37.22 μs	57.32 ms	132.84
				Area	735.2	1.491	45.61 μs	70.23 ms	104.77
				Balance	961.5	2.106	34.87 μs	53.71 ms	113.15
			2^{16}	Delay	729.9	7.377	11.75 μs	18.14 ms	133.84
				Area	390.6	3.103	21.95 μs	33.90 ms	105.22
				Balance	775.1	6.128	11.06 μs	17.08 ms	104.69
			2^{32}	Delay	617.2	22.106	3.63 μs	5.63 ms	124.54
				Area	212.3	9.561	10.55 μs	16.37 ms	156.60
				Balance	578.0	16.628	3.87 μs	6.01 ms	100.04
			2^{64}	**Delay**	**520.8**	**74.765**	**1.17 μs**	**1.83 ms**	**137.03**
				Area	107.6	32.963	5.66 μs	8.86 ms	292.33
				Balance	512.8	73.199	1.18 μs	1.86 ms	136.26
[2]	0.13 μm CMOS	Radix 2^{64}, 64-bit multiplier			137.7	96.224	4.57 μs	-	-
[3]	0.5 μm CMOS	Radix 2, 40PEs x 8bit			80	28	43 μs	88.2 ms	2469.6
[4]	1.2 μm CMOS	Radix 2, 7PEs x 32bit			80	-	61 μs	-	-
[5]	Xilinx Virtex Pro	Radix 2, 64PEs x 16bit			144	5598 LUTs	-	16 ms	-
[6]	Xilinx XC40150XV	Radix 2, Systolic array			52	4865 CLBs	-	40.5 ms	-
[8]	Xilinx XC40250XV	Radix 2^4, Systolic array			45	6633 CLBs	-	11.95 ms	-

[6] T. Blum and C. Paar, "Montgomery modular exponentiation on reconfigurable hardware," in *Proc. the 14th IEEE Symp. Computer Arithmetic (ARITH)*. IEEE Computer Society, 1999, pp. 70–78.

[7] A. Daly and W. Marnane, "Efficient architectures for implementing Montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic,"in *Proc. the 2002 ACM/SIGDA 10th Int. Symp. Field-Programmable Gate Arrays*. ACM Press, 2002, pp.40–49.

[8] T. Blum and C. Paar, "High-radix montgomery modular exponentiation on reconfigurable hardware," *IEEE Trans. Comput.*, vol. 50, no. 7, pp. 759–764, 2001.

[9] C. K. Koc, T. Acar, and B. S. Kaliski, "Analyzing and comparing Montgomery multiplication algorithms," *IEEE Micro*, vol. 16, no. 3, pp. 26–33, 1996.

[10] J. A. Menezes, C. P. Oorschot, and A. S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1997.

[11] Circuits Multi-Projets (CMP), CMOS 90nm (CMOS090) from STMicroelectronics, http://cmp.imag.fr/products/ic/?p=STCMOS090.