Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2010, Article ID 657076, 9 pages
doi:10.1155/2010/657076

Research Article

Revisiting Sum of Residues Modular Multiplication

Yinan Kong[1] and Braden Phillips[2]

[1] *Department of Electronic Engineering, Macquarie University, North Ryde, NSW 2109, Australia*
[2] *Department of Electrical and Electronic Engineering, The University of Adelaide, North Terrace, SA 5005, Australia*

Correspondence should be addressed to Yinan Kong, ykong@science.mq.edu.au

Received 30 March 2010; Revised 10 August 2010; Accepted 15 September 2010

Academic Editor: Yong-Bin Kim

In the 1980s, when the introduction of public key cryptography spurred interest in modular multiplication, many implementations performed modular multiplication using a sum of residues. As the field matured, sum of residues modular multiplication lost favor to the extent that all recent surveys have either overlooked it or incorporated it within a larger class of reduction algorithms. In this paper, we present a new taxonomy of modular multiplication algorithms. We include sum of residues as one of four classes and argue why it should be considered different to the other, now more common, algorithms. We then apply techniques developed for other algorithms to reinvigorate sum of residues modular multiplication. We compare FPGA implementations of modular multiplication up to 24 bits wide. The Sum of Residues multipliers demonstrate reduced latency at nearly 50% compared to Montgomery architectures at the cost of nearly doubled circuit area. The new multipliers are useful for systems based on the Residue Number System (RNS).

1. Introduction

Modular multiplication is important for many applications including cryptography and image processing. Many different modular multiplication algorithms have been published [1–5] and have been deployed for public-key cryptography or digital signal processing. In this paper, we reinvigorate the sum of residues class of modular multipliers by describing a new modular multiplication algorithm and implementation.

Section 3 surveys the literature of modular multiplication to arrive at 4 classes of algorithm: *sum of residues*, *classical*, *Barrett* and *Montgomery*. The goal of this section is not a comprehensive survey all publications, but to support the claim that sum of residues is a distinct class that has been largely ignored. In Section 4, we apply optimizations, originally proposed for other classes of reduction, to breathe new life into sum of residues modular multiplication. Section 5 evaluates the reinvigorated sum of residues approach by comparing it with Montgomery multiplication on an FPGA.

2. Motivation: The Residue Number System

Our interest in modular multiplication at word length up to around 24-bits stems from its application to systems built using the Residue Number System (RNS). By representing integers in independent short-word length channels, residue number systems offer advantages for digital signal processing [6–8] and long word length arithmetic, especially for crypto-operations [9, 10].

A residue number system [11] is characterized by a set of N coprime moduli $\{m_1, m_2, \ldots, m_N\}$. In the RNS, a number X is uniquely represented in N channels: $X = \{x_1, x_2, \ldots, x_N\}$, where x_i is the residue of X with respect to m_i, that is, $x_i = \langle X \rangle_{m_i} = X \bmod m_i$.

If X, Y, and Z have RNS representations given by $X = \{x_1, x_2, \ldots, x_N\}$, $Y = \{y_1, y_2, \ldots, y_N\}$, and $Z = \{z_1, z_2, \ldots, z_N\}$, then denoting $*$ to represent the operations $+$, $-$, or $\times$, the RNS version of $Z = X * Y$ satisfies

$$Z = \{\langle x_1 * y_1 \rangle_{m_1}, \langle x_2 * y_2 \rangle_{m_2}, \ldots, \langle x_N * y_N \rangle_{m_N}\}. \quad (1)$$

Thus, addition, subtraction, and multiplication can be concurrently performed on the N residues within N parallel channels, and it is this high-speed parallel operation that makes the RNS attractive. The multipliers described in this paper are intended for the modular multiplications $\langle x_i \times y_i \rangle_{m_i}$ within RNS channels. These typically have a word length from 4 to 24 bits.

2.1. Contribution. This paper contributes to the literature of modular multiplication by

(1) identifying sum of residues as a separate class of modular reduction algorithm,

(2) presenting a new algorithm and implementation for sum of residues modular multiplication, and

(3) demonstrating that this approach can deliver performance benefits, particularly for channel-width modular multipliers for RNS systems on FPGA.

2.2. Notation. We consider the modular multiplication $C = A \times B \bmod M$, where A, B, and M are n-digit integers of the form $X = \sum_{i=0}^{n-1} x_i r^i$. The radix r is typically a positive power of 2.

Note that it is common for modular multiplication algorithms to produce a result $C > M$ such that a few subtractions of M are required to fully reduce the result. The usual approach is to design the algorithm so C can be fed back to the input without overflow, even if C is not fully reduced.

3. Classes of Modular Multiplication

Some early modular multipliers [4, 12–15] proceed by accumulating residues modulo M. Equation (2) is a typical starting point. The residues, typified by $(Br^i \bmod M)$ in (2), may be precomputed and retrieved from a table (e.g., [12]) or evaluated recursively during the modular multiplication (e.g., [13]). A typical algorithm will be shown in the next section, from which point the sum of residues algorithm will be analyzed and improved

$$C = \sum_{i=0}^{n-1} a_i \left(Br^i \bmod M \right). \tag{2}$$

Instead of accumulating residues modulo M, reduction can be performed by subtracting multiples of M. Papers that take this approach include [5, 16–19]. Algorithm 1 is typical. Reduction in this way can be understood as a division in which the quotient is discarded and the remainder retained. Development of modular multipliers along this line has, therefore, closely followed the development of division, especially SRT division (as originally in [20]).

The Quotient Digit Selection function (QDS) has received a great deal of attention to: permit quotient digits (q_i) to be trivially estimated from only the most significant bits of the partial result C, allow the partial result to be stored in a redundant form, and move the QDS function from the critical path (e.g., [17, 18]).

Ensure: $C \equiv A \times B \bmod M$
  $C = 0$
for $i = n - 1$ downto 0 **do**
    $C = rC + a_i B$ {Partial product accumulation}
    $q_i = \mathrm{QDS}(C, M)$ {Quotient digit selection}
    $C = C - q_i M$ {Reduction step}
end for

Algorithm 1: A Typical Example of Classical Modular Multiplication.

Require: α, β {Pre-defined parameters}
Require: $K = \lfloor 2^{n+\alpha}/M \rfloor$ {A precomputed constant}
Ensure: $C \equiv A \times B \bmod M$
  $C = A \times B$
  $C1 = \lfloor C/2^{n+\beta} \rfloor$ {Right shift by $n + \beta$}
  $C2 = C1 \times K$
  $Q = \lfloor C/2^{\alpha-\beta} \rfloor$
  $C = C - Q \times M$

Algorithm 2: Improved Barrett modular multiplication.

The relationship between division and modular multiplication is made explicit in

$$A \times B \bmod M = (A \times B) - \left\lfloor \frac{A \times B}{M} \right\rfloor \times M. \tag{3}$$

This equation suggests an alternative mechanism: one may perform the division $\lfloor (A \times B)/M \rfloor$ by multiplying by M^{-1}. Papers that follow this line include [1, 21, 22]. Note that M^{-1} is a real number so that correct evaluation of (3) using fixed-point arithmetic requires careful design of the representation. A typical example is the improved Barrett algorithm (named after Barrett's reduction in [1]) described in [22] and shown in Algorithm 2.

Most recently, modular multipliers based on Montgomery's reduction algorithm [2] have been popular [3, 23, 24]. A typical form is shown in Algorithm 3. Note that the quotient digit selection step examines only the least significant digit of the partial result C. Also, note that Montgomery's method does not produce a fully reduced residue $C = A \times B \bmod M$ directly, but rather the Montgomery residue $C \times R^{-1} \bmod M$. Computation can proceed with Montgomery residues as an internal representation. An extra modular multiplication is required to convert the final result to a fully reduced residue.

We have, therefore, identified 4 different classes of modular multiplication algorithm according to the way in which they perform reduction.

(1) *Sum of Residues*: reduction is achieved by accumulating residues modulo M.

(2) *Classical*: multiples of the modulus $q_i M$ are subtracted according to a QDS function that examines the most significant digits of the partial result C.

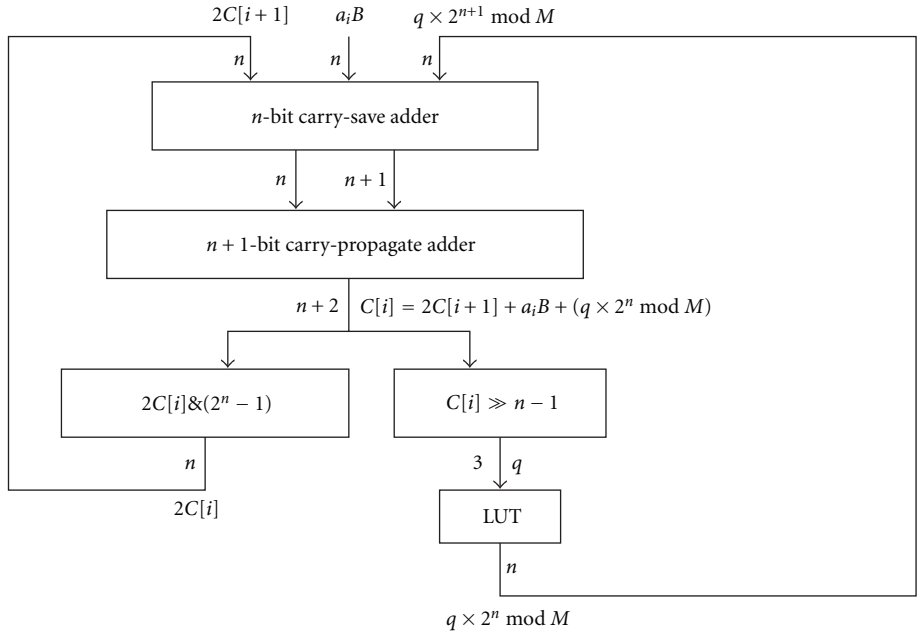

Figure 1: An architecture for Tomlinson's interleaved modular multiplication algorithm.

Require: $R = r^n$
Require: m_0^{-1} s.t. $m_0 \times m_0^{-1} \equiv 1 \bmod r$
Ensure: $C \equiv A \times B \times R^{-1} \bmod M$
  $C = 0$
  for $i = 0$ to $n - 1$ **do**
    $C = C + a_iB$ {Partial product accumulation}
    $q_i = -c_0 \times m_0^{-1} \bmod M$ {Quotient digit selection}
    $C = (C + q_iM)/r$ {Reduction step}
  end for

Algorithm 3: Montgomery modular multiplication.

(3) *Barrett*: multiplication by M^{-1} is used to reduce modulo M.

(4) *Montgomery*: multiples of the modulus q_iM are accumulated according to a QDS function that examines the least significant digits of the partial result C.

We note here that each of the 4 classes permits separated and interleaved implementations. In a separated implementation, $A \times B$ is evaluated before being reduced modulo M. The alternative is to interleave the multiplication and reduction steps. This has the benefit of keeping intermediate values to the approximate magnitude of M but does not take advantage of any preexisting nonmodular multiplier. In this paper, we are largely concerned with interleaved implementations.

Surveys of modular reduction offer different classifications. References [25–27] identify three classes: classical, Barrett, and Montgomery. References [22, 28] include the Barrett algorithms with other classical algorithms and therefore divide the field into only two classes: classical and Montgomery. None of these surveys cover sum of residues papers. Other publications, [4, 29, 30], have made note of the sum of residues technique but categorize it along with other classical algorithms.

4. Reinvigorating Sum of Residues

4.1. Tomlinson's Algorithm. We take the modular multiplier developed by Tomlinson in [4] as our starting point for further development. Tomlinson's algorithm is shown in Algorithm 4.

At first sight, this may look like a classical algorithm as the quotient digit q is selected from the most significant bits of the partial result. The difference is that a classical algorithm then performs reduction by subtracting the multiple of the modulus q_iM; Tomlinson's algorithm performs the reduction by setting the most significant bits to zero and accounting for this change by adding the precomputed residue $(q \times 2^{n+1} \bmod M)$.

An architecture for Tomlinson's algorithm is shown in Figure 1. Note that the intermediate result C at iteration i is denoted $C[i]$. As the loop is from $n - 1$ down to 0, $C[i + 1]$ denotes the value C in the previous iteration.

A Carry-Save Adder (CSA) is used to perform the three-term addition $C[i] = 2C[i+1] + a_iB + (q \times 2^{n+1} \bmod M)$. To make sure $2C[i + 1]$ is n bits long, the same as the other two addends, q is set to be the upper 3 bits instead of 2 bits of the current partial result.

Note that the carry-save representation is a type of redundant representation that has been applied to other modular multiplication algorithms [15, 19, 31]. We will keep using this technique to enhance the sum of residues modular multiplier.

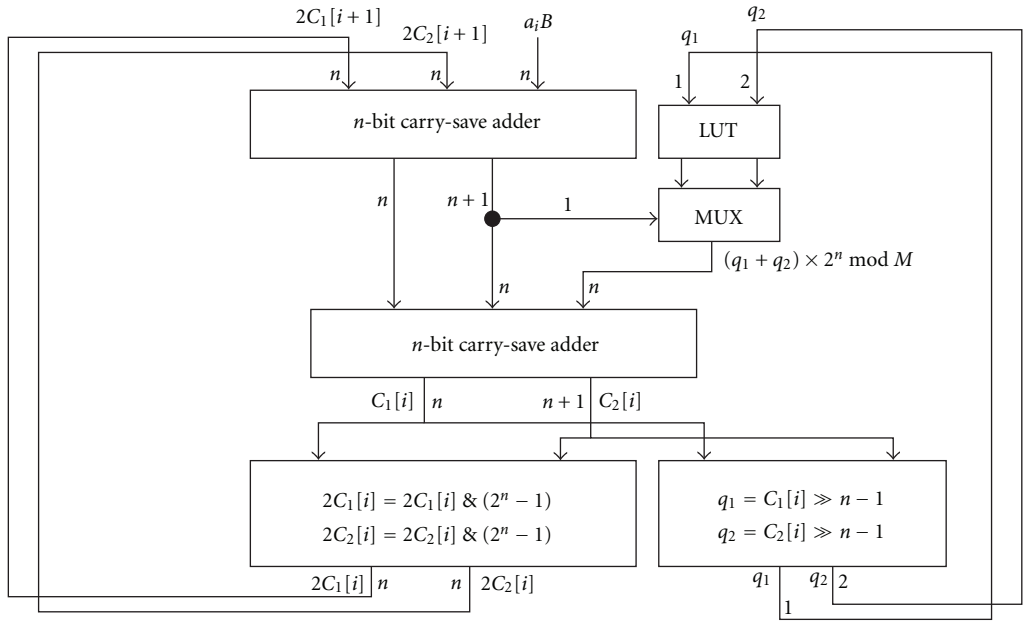

Figure 2: Modified sum of residues modular multiplier architecture.

Ensure: $C \equiv A \times B \bmod M$, $C < 2^{n+1}$
  $C = 0$
  $q = 0$
  for $i = n - 1$ downto 0 **do**
    $C = 2C + a_iB$
    $C = C + (q \times 2^{n+1} \bmod M)$ {The residue $(q \times 2^{n+1} \bmod M)$ is precomputed.}
    $q = \lfloor C/2^n \rfloor$ {q is the upper 2 bits of C}
    $C = C - q \times 2^n$ {Set the upper 2 bits of C to zero}
  end for
  $C = 2C + (q \times 2^{n+1} \bmod M)$

Algorithm 4: Tomlinson's sum of residues modular multiplication.

Reference [15] gives an similar algorithm but sets q only two bits long. This means that the partial result $C[i]$ may be $n + 1$ bits long. To bound it within n bits, a subtracter is used to constantly subtract M until $C[i]$ has only n bits. This redundant step greatly increases the latency of the algorithm. In the following sections, we describe new enhancements to improve the performance of this algorithm.

4.2. Eliminating the Carry-Propagate Adder. There two obvious demerits of the architecture in Figure 1. Firstly, a Carry-Propagate Adder (CPA) is used to transform the redundant representation of $C[i]$ to its nonredundant form. This is required because the upper 3 bits of $C[i]$ have to be known to look up $q \times 2^n \bmod M$ before the next iteration. The CPA delay contributes significantly to the latency of the implementation. The second problem is that the lookup of $q \times 2^n \bmod M$ is on the critical path.

Both of these problems can be solved by keeping the intermediate result in a redundant carry-save form. The CPA of Figure 1 is eliminated so that the calculation of the partial result becomes $C_1[i]+C_2[i] = C_1[i+1]+C_2[i+1]+a_iB+((q_1+q_2) \times 2^n \bmod M)$, where $C_1[i]$ and $C_2[i]$ are the redundant representation of $C[i]$ as sum and carry terms, respectively. A modified architecture is shown in Figure 2. The CPA is replaced by a second CSA.

The precomputed residue $(q_1 + q_2) \times 2^n \bmod M$, which must be retrieved from a lookup table (LUT), can be sent to the second CSA rather than the first. All three addends to the first CSA are available at the beginning of each iteration and the table lookup step can be performed in parallel with the first CSA.

In Figure 1, it can be seen that the carry output of the first CSA is $n + 1$ bits wide. This cannot be input directly to the second CSA which is only n-bits wide. Thus, in Figure 2,

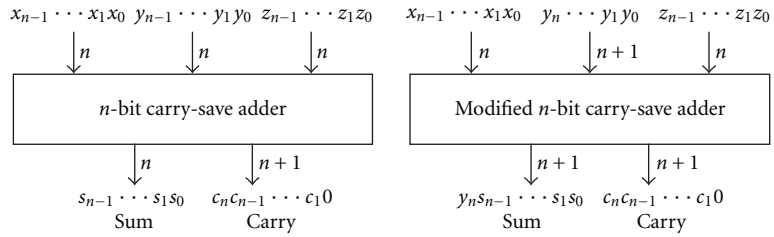

Figure 3: n-bit carry-save adders.

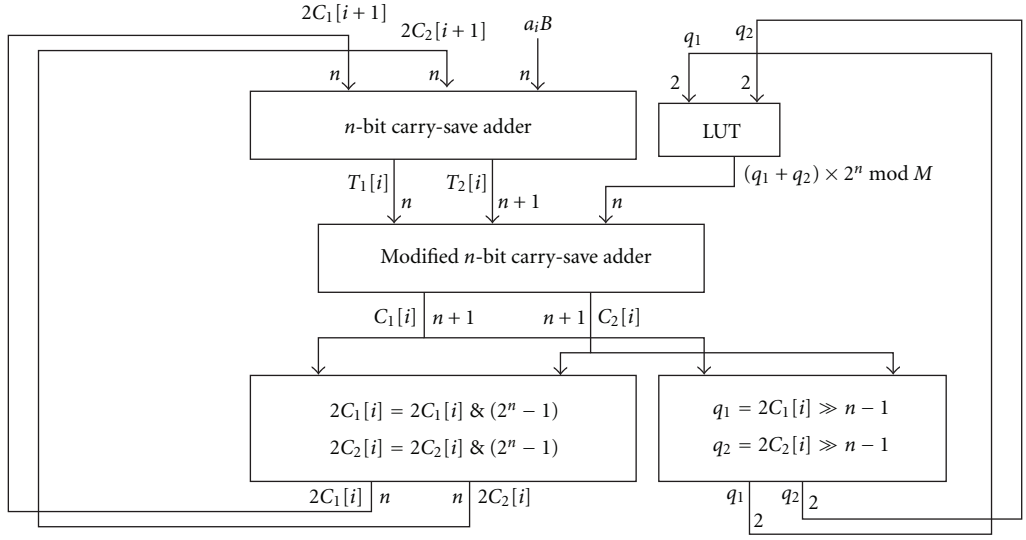

Figure 4: New sum of residues modular multiplier architecture.

the MSB of the $(n + 1)$-bit carry is sent to the LUT circuit instead. The LUT retrieves two possible values of $(q_1 + q_2) \times 2^n \bmod M$ corresponding to the case of either a 0 or 1 in the MSB of the carry output from the first CSA. An MUX selects the appropriate value of $(q_1 + q_2) \times 2^n \bmod M$ once the MSB is available. Thus, although the LUT executes in parallel with the first CSA, an additional MUX appears on the critical path.

4.3. Further Enhancements. If the second CSA in Figure 2 can be modified to accept an $(n + 1)$-bit input, the MUX can be eliminated. The left of Figure 3 shows a conventional n-bit CSA. Note that the output sum is only n bits wide. To accept an $(n + 1)$-bit input, we can just copy the MSB of the $(n + 1)$-bit input to the MSB of output sum. This is illustrated in the right of Figure 3. This modified CSA accepts 1 $(n + 1)$-bit input and 2 n-bit inputs and produces 2 $(n + 1)$-bit outputs.

Figure 4 shows the resulting modular multiplication architecture. The algorithm corresponding to this new architecture is given as Algorithm 5. The CPA has been eliminated from the iteration, and the residue lookup has been shifted from the critical path. Also, no subtraction is needed at the end of the algorithm to bound the output within $n+1$ bits. If $C_1[0]$ and $C_2[0]$ are simply summed using a CPA, the resulting output C could be $n + 2$ bits, which needs some further subtraction to be reduced. Therefore, the same technique as in the loop is applied. Both $C_1[0]$ and $C_2[0]$ are set to $n - 1$ bits and the n-bit residue corresponding to the 2 upper reset bits is retrieved from another LUT. The final sum yields an $(n + 1)$-bit output C.

The LUTs have a 4-bit input and an n-bit output so that a $(2^4 \times n)$-bit ROM can be used. Moreover, note that the sum of $(q_1 + q_2)$ is at most 110, which occurs when $(q_1$ and $q_2)$ are both 11. This implies that the possible sum of $(q_1 + q_2)$ is in the range from 000 to 110, which has 7 values only. Therefore, a ROM with a further reduced size of $(7 \times n)$ bits can be used for $(q_1 + q_2) \times 2^n \bmod M$. For example, a 128-bit modular multiplier only needs a 1K-bit ROM, which is reasonable for a RNS channel modular multiplier.

Figure 5 shows an example of the new algorithm for the case $r = 2$, $n = 4$, $A = 15 = (1111)_2$, $B = 11 = (1011)_2$, and $M = 9 = (1001)_2$. It is noted that at the last step, a second LUT of the same size is needed. Also, because the output C from the 4-bit CPA is at most 5 bits long, the final subtraction might not be necessary if a $n+1$-bit C is acceptable, as the case in quite a few other algorithms. Even if a 4-bit C is required, only one subtraction will do.

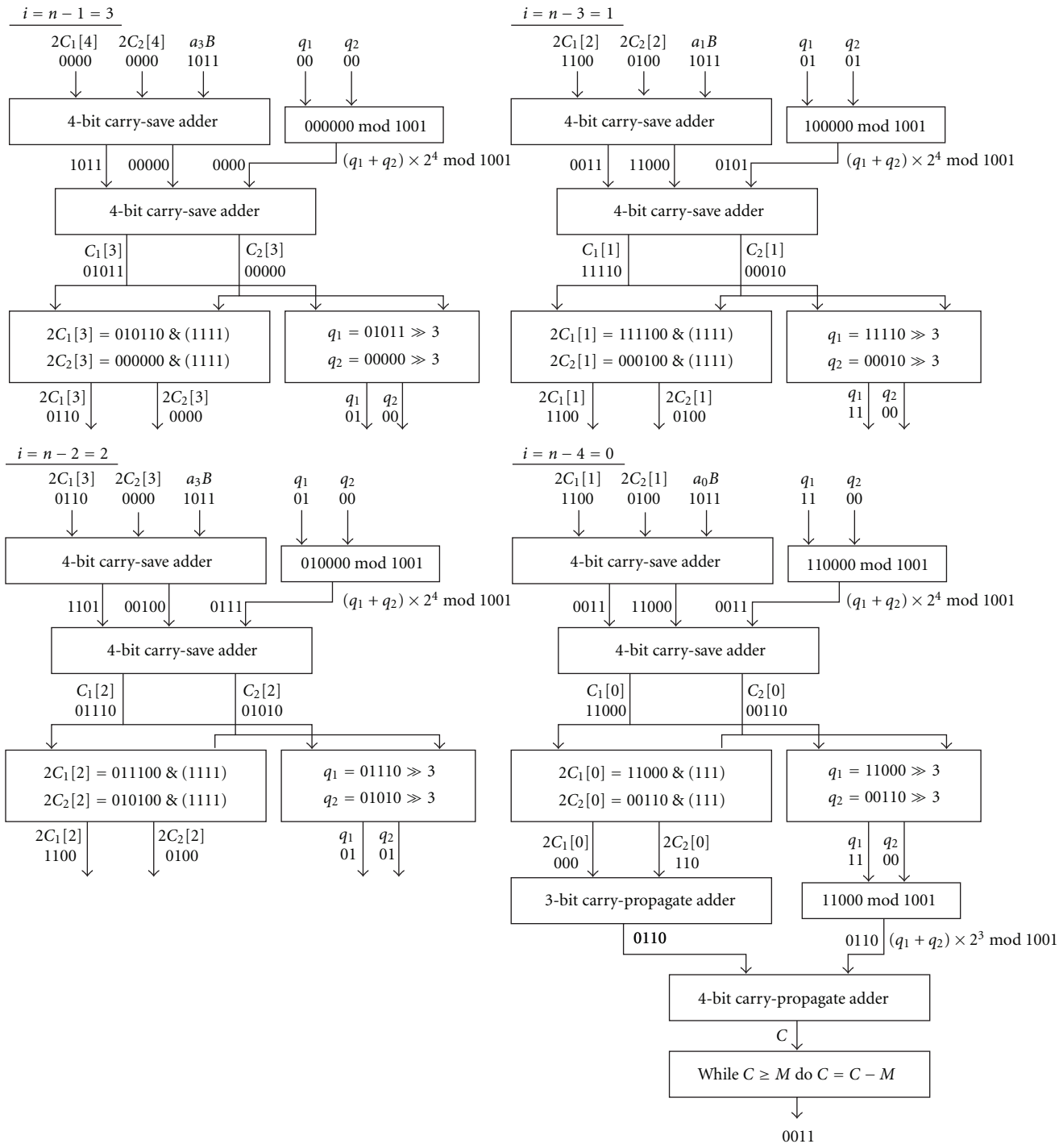

Figure 5: An example for the new algorithm $n = 4$, $A = (1111)_2$, $B = (1011)_2$, and $M = (1001)_2$.

4.4. Higher Radix. High radix is another popular technique that has been adopted by other algorithms [17, 24]. It also contributes to the new sum of residues algorithm. A radix-r version of the algorithm can be produced as in Figure 6. If $r = 2^k$, this version executes in n/k iterations; however, a larger LUT and $(n + k)$-bit CSAs are required.

5. Evaluation

5.1. Evaluation Environment. FPGA implementations have been prepared. A Xilinx Virtex2 FPGA was used as the implementation target. All the implementations have been performed using the Xilinx ISE environment using

Ensure: $C \equiv A \times B \bmod M$, $C < 2^{n+1}$
  $C_1[n] = C_2[n] = q_1 = q_2 = 0$
  for $i = n - 1$ downto 0 **do**
    $\{T_1[i], T_2[i]\} = 2C_1[i+1] + 2C_2[i+1] + a_iB$ {Carry save addition}
    $\{C_1[i], C_2[i]\} = T_1[i] + T_2[i] + ((q_1 + q_2) \times 2^n \bmod M)$
    {Carry save addition} {The residue $((q_1 + q_2) \times 2^n \bmod M)$ is precomputed}
    $\{q_1, q_2\} = \{C_1[i] \gg (n-1), C_2[i] \gg (n-1)\}$ {q_1 and q_2 are the upper 2 bits of $C_1[i]$ and $C_2[i]$ respectively.}
    $\{C_1[i], C_2[i]\} = \{2C_1[i] \,\&\, (2^n - 1), 2C_2[i] \,\&\, (2^n - 1)\}$
    {Set the upper 2 bits of $C_1[i]$ and $C_2[i]$ to zero}
  end for
  $\{C_1[0], C_2[0]\} = \{2C_1[0], 2C_2[0]\}$ {Right shift so they are both $n - 1$ bits}
  $C = C_1[0] + C_2[0] + ((q_1 + q_2) \times 2^{n-1} \bmod M)$

Algorithm 5: New sum of residues modular multiplication.

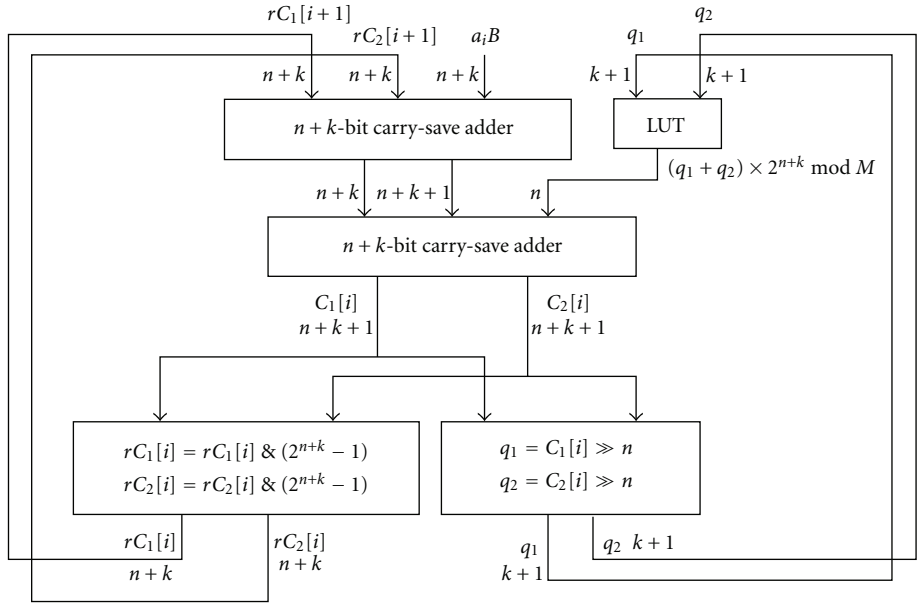

Figure 6: New higher-radix sum of residues modular multiplier architecture.

XST for synthesis and ISE standard tools for place and route with standard effort for all speed optimizations (see Table 2).

Pure delays of the combinatorial circuit were measured excluding those between pads and pins. They were generated from the post-place and route static timing analyzer with a standard place and route effort level.

5.2. Results. Implementation results are listed in Table 1 for $n = 4$ to $n = 24$ at radix-2, the most popular word lengths of RNS channel modular multipliers [6, 9, 32]. The table includes results for the old architecture based on Tomlinson's algorithm (Figure 1) as well as for a carefully optimized Montgomery architecture.

The Montgomery architecture have incorporated various published techniques for optimization. For example, the techniques in [3, 23] have been applied to improve performance by making the quotient digit selection step trivial and moving it from the critical path. The cost of these techniques is that they impose limits on the possible values of the modulus M which may impact on other enhancements such as the use of higher radices. Consequently, the Montgomery architecture is interleaved, uses radix 2 and trivial quotient digit selection (as in [3, 33] described in Section 3) and was arrived at by varying these parameters to find the multiplier with lowest delay.

These multipliers are compared with the new binary sum of residues architecture of Figure 4. It can be seen that the new sum of residues modular multiplier is a competitive alternative implementation on FPGA. It demonstrates better timing performance than both Tomlinson's architecture and the Montgomery architecture although its hardware cost is the highest among the three.

6. Conclusions and Future Work

Sum of residues is a distinct class of modular multiplication that has been overlooked in recent years. We have shown

Table 1: Latency and space overhead of three interleaved modular multipliers.

Architecture	Figure	$n = 4$	$n = 8$	$n = 12$	$n = 16$	$n = 24$
New sum of residues	Figure 4	12.1 ns 35 slices	22.3 ns 142 slices	33.5 ns 346 slices	48.4 ns 616 slices	66.9 ns 1415 slices
Tomlinson	Figure 1	15.0 ns 17 slices	27.3 ns 62 slices	48.6 ns 158 slices	72.5 ns 275 slices	129.5 ns 607 slices
Montgomery		16.6 ns 24 slices	31.7 ns 84 slices	50.1 ns 176 slices	69.6 ns 300 slices	101.2 ns 644 slices

Table 2

Target FPGA	Virtex2 XC2V1000 with a −6 speed grade, 1M gates, 5120 slices and embedded 18 × 18 multipliers
Xilinx 6.1i	XST-synthesis ISE-place and route
Optimization goal	Speed
Language	VHDL

that techniques pioneered for other modular multiplication algorithms, such as the use of redundant representations and higher radices, can also be applied to sum of residues. By doing this, we have arrived at a new sum of residues modular multiplier that uses carry-save adders and redundant number representation to achieve a more parallel structure than previous versions. FPGA implementations of the new architecture demonstrate low latency relative to previous sum of residues and Montgomery architectures at the cost of increased space overhead.

Future work will be focused on reducing the space overhead. More advanced programmable logic devices will be attempted to utilize hardware resources more efficiently. ASIC design will be investigated, where a lot of the hardware redundancy expects to be reduced. For example, our new CSA shown in Figure 3 accepts one extra bit input without extra hardware resources needed. However, this benefit is hard to see in the FPGA implementation because splitting bits is impossible in the used configurable logic blocks (CLB) embedded on the FPGA. Nor do the embedded 18-bit multipliers. However, this gets easily illustrated in an ASIC implementation. Therefore, the new sum of residues modular multiplication algorithm is expected to be suitable for an ASIC application.

References

[1] P. Barrett, "Implementing the Rivest, Shamir and Adleman public-keyencryption algorithm on a standard digital signal processor," in *Proceedings of the Advancesin Cryptology (Crypto '86)*, vol. 263 of *Lecture Notes in Computer Science*, pp. 311–323, 1986.

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

[3] C. D. Walter, "Still faster modular multiplication," *Electronics Letters*, vol. 31, no. 4, pp. 263–264, 1995.

[4] A. Tomlinson, "Bit-serial modular multiplier," *Electronics Letters*, vol. 25, no. 24, p. 1664, 1989.

[5] E. F. Brickell, "A fast modular multiplication algorithm with application to two key cryptography," in *Proceedings of the Advancesin Cryptology (Crypto '82)*, Lecture Notes in Computer Science, pp. 51–60, 1982.

[6] M. A. Soderstrand, W. Jenkins, and G. Jullien, "Residue Number System Arithmetic: Modern Applications—Digital Signal Processing," 1986.

[7] M. Bhardwaj and B. Ljusanin, "The renaissance—a residue number system based vector co-processor," in *Proceedings of the 32nd Asilomar Conferenceon Signals, Systems & Computers*, vol. 2, pp. 202–207, 1998.

[8] G. Alia and E. Martinelli, "Optimal VLSI complexity design for high speed pipeline FFT using RNS," *Computers and Electrical Engineering*, vol. 24, no. 3-4, pp. 167–182, 1998.

[9] J.-C. Bajard and L. Imbert, "A full RNS implementation of RSA," *IEEE Transactions on Computers*, vol. 53, no. 6, pp. 769–774, 2004.

[10] J.-C. Bajard, L.-S. Didier, and P. Kornerup, "Modular multiplication and base extensions in residue number systems," in *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, vol. 2, pp. 59–65, 2001.

[11] N. S. Szabo and R. H. Tanaka, *Residue Arithmetic and Its Applications to Computer Technology*, McGraw-Hill, New York, NY, USA, 1967.

[12] S.-I. Kawamura and K. Hirano, "A fast modular arithmetic algorithmusing a residue table," in *Proceedings of the Advances in Cryptology (Eurocrypt '88)*, vol. 330 of *Lecture Notes in Computer Science*, pp. 245–250, 1988.

[13] P. A. Findlay and B. A. Johnson, "Modular exponentiation using recursivesums of residues," in *Proceedings of the Advancesin Cryptology (Crypto '89)*, vol. 435 of *Lecture Notes in Computer Science*, pp. 371–386, 1989.

[14] F. F. Su and T. Hwang, "Comments on iterative modular multiplication without magnitude comparison," in *Proceedings of the 6th National Conference on Information Security*, pp. 21–22, Taichung, Taiwan, 1996.

[15] C.-Y. Chen and C.-C. Chang, "Fast modular multiplication algorithm for calculating the product *ab* modulo *n*," *Information Processing Letters*, vol. 72, no. 3, pp. 77–81, 1999.

[16] G. R. Blakley, "A computer algorithm for calculation the product *ab* modulo *m*," *IEEE Transactions on Computers*, vol. C-32, no. 5, pp. 497–500, 1983.

[17] H. Orup and P. Kornerup, "A high-radix hardware algorithm for calculating the exponential m^e modulo *n*," in *Proceedings of the 10th IEEE Symposium on Computer Arithmetic*, vol. 576, pp. 51–57, June 1991.

[18] C. D. Walter, "Faster multiplication by operand scaling," in *Proceedings of the Advancesin Cryptology (Crypto '91)*, vol. 576 of *Lecture Notes in Computer Science*, pp. 313–323, 1991.

[19] R. Modugu and M. Choi, "A fast low-power modulo 2^n + 1 multiplier design," in *Proceedings of the IEEE International Instrumentation and Measurement Technology Conference (I2MTC '09)*, 2009.

[20] J. E. Robertson, "A new class of digital division methods," *IRE Transactions on Electronic Computers*, vol. 7, pp. 218–222, 1958.

[21] C. D. Walter, "Logarithmic speed modular multiplication," *Electronics Letters*, vol. 30, no. 17, pp. 1397–1398, 1994.

[22] J.-F. Dhem, *Design of an efficient public-key cryptographic library forRISC based smart cards*, Ph.D. thesis, Université Catholique de Louvain, May 1998.

[23] M. Shand and J. Vuillemin, "Fast Implementations of RSA cryptography," in *Proceedings of the 11th IEEE Symposium on Computer Arithmetic*, pp. 252–259, July 1993.

[24] H. Orup, "Simplifying quotient determination in high-radix modular multiplication," in *Proceedings of the 12th IEEE Symposium on Computer Arithmetic*, pp. 193–199, July 1995.

[25] A. Bosselaers, R. Govaerts, and J. Vandewalle, "Comparisons of three modular reduction functions," in *Proceedings of the Advancesin Cryptology (Crypto '93)*, vol. 773 of *Lecture Notes in Computer Science*, pp. 175–186, 1993.

[26] A. Bosselaers, R. Govaerts, and J. Vandewalle, "A fast and flexible software library for large integer arithmetic," in *Proceedings of the 15th Symposiumon Information Theory in the Benelux, Louvain-La-Neuve(B)*, pp. 82–89, 1994.

[27] F. C. Chang and C. J. Wang, "Architectural tradeoff in implementing RSA processors," *ACM SIGARCH Computer Architecture News*, vol. 30, no. 1, pp. 5–11, 2002.

[28] C. D. Walter, "Montgomery's multiplication technique: how to make it smaller and faster," in *Proceedings of the 1st International Workshop on Cryptographic Hardware and Embedded Systems (CHES '99)*, C.-K. Koc and C. Paar, Eds., vol. 1717 of *Lecture Notes in Computer Science*, pp. 80–93, Worcester, Ma, USA, August 1999.

[29] C. W. Chiou and T. C. Yang, "Iterative modular multiplication algorithm without magnitude comparison," *Electronics Letters*, vol. 30, no. 24, pp. 2017–2018, 1994.

[30] C. H. Lim, H. S. Hwang, and P. J. Lee, "Fast modular reduction with precomputation," in *Proceedings of the Japan-Korea Joint Workshop on Information Security and Cryptology (JW-ISC '97)*, pp. 65–79, October 1997.

[31] C. Mclvor, M. McLoone, and J. V. McCanny, "Modified Montgomery modular multiplication and RSA exponentiation techniques," *IEE Proceedings: Computers and Digital Techniques*, vol. 151, no. 6, pp. 402–408, 2004.

[32] H. Nozaki, M. Motoyama, A. Shimbo, and S. Kawamura, "Implementation of RSA algorithm based on RNS montgomery multiplication," in *Proceedings of the Cryptographic Hardware and Embedded Systems (CHES '01)*, pp. 364–376, September 2001.

[33] Y. Kong and B. Phillips, "Montgomery modular multiplier," Tech. Rep., The University of Adelaide, Adelaide, Australia, 2005.