

A Radix-4 Design of a Scalable Modular Multiplier With Recoding Techniques

Lo'ai A. Tawalbeh, Alexandre F. Tenca and Ç. K. Koç

School of Electrical Engineering & Computer Science

Oregon State University

{tawalbeh,tenca,koc}@ece.orst.edu

Abstract

This paper presents an algorithm and architecture for a scalable radix-4 multiplier that makes use of two types of digit recoding in order to generate an efficient solution. Experimental results are shown to demonstrate that the proposed radix-4 Montgomery Multiplier design has better area/time tradeoff than previous radix-2 and radix-8 scalable designs.

I. INTRODUCTION

Modular multiplication is the most important arithmetic operation used in many cryptographic algorithms such as RSA and Diffie-Helman key exchange. The Montgomery Modular Multiplication algorithm (MM) [1] has enabled considerable advantage in speeding up cryptographic algorithms.

Previous scalable modular multiplier designs use radix-2 MM algorithm [2]. Multiplication is implemented by a series of multi-precision partial-product additions. To reduce the number of products and, therefore, the complexity of multi-operand addition, it is convenient to consider the multiplier in radix higher than 2, as done in [3] for radix 8. A radix-4 design generated by the use of the same techniques applied to radix 8 resulted in very poor area utilization and performance. In this paper we show that a more elaborate design using two types of digit recoding makes the radix-4 design the best solution for the implementation of this scalable multiplier. Experimental results are shown to support this claim.

II. PROPOSED RADIX-4 MM ALGORITHM

The notation used in the proposed multiple-word Radix-4 Montgomery Multiplication algorithm (R4MM) is shown below:

```

Step
1:    $S := 0$ 
       $x_{-1} := 0$ 
2:   FOR  $j := 0$  TO  $N - 1$  STEP 2
3:      $Z_j = \text{Recoding1}(x_{j+1..j-1})$ 
4:      $(Ca, S^{(0)}) := S^{(0)} + (Z_j * Y)^{(0)}$ 
5:      $q_{M_j} := S_{1..0}^{(0)} * (4 - M_{1..0}^{(0)^{-1}}) \bmod 4$ 
5a:     $q'_{M_j} := \text{Recoding2}(q_{M_j})$ 
6:      $(Cb, S^{(0)}) := S^{(0)} + (q'_{M_j} * M)^{(0)}$ 
7:     FOR  $i := 1$  TO  $e - 1$ 
8:        $(Ca, S^{(i)}) := Ca + S^{(i)} + (Z_j * Y)^{(i)}$ 
9:        $(Cb, S^{(i)}) := Cb + S^{(i)} + (q'_{M_j} * M)^{(i)}$ 
10:       $S^{(i-1)} := (S_{1..0}^{(i)}, S_{BPW-1..2}^{(i-1)})$ 
      END FOR;
11:     $Ca := Ca \text{ or } Cb$ 
12:     $S^{(e-1)} := \text{signext}(Ca, S_{BPW-1..2}^{(e-1)})$ 
  END FOR;

```

Fig. 1. Multiple-word Radix-4 Montgomery Multiplication (R4MM) Algorithm.

- X_j - a single radix-4 digit of X at position j ;
- q_{M_j} - quotient digit that determines a multiple of the modulus M to be added to the partial product S ;
- w - number of bits in a word of either Y , M or S ;
- $e = \lceil \frac{n+1}{w} \rceil$ - number of words in either Y , M or S ;
- NS - number of stages;
- C_a, C_b - carry bits;
- $(Y^{(e-1)}, \dots, Y^{(1)}, Y^{(0)})$ - operand Y represented as multiple words;
- $S_{k-1..0}^{(i)}$ - bits $k - 1$ to 0 of the i^{th} word of S .

Figure 1 shows the R4MM algorithm, which is an extension of the Multiple-Word High-Radix ($R2^k$) Montgomery Multiplication algorithm (MWR 2^k MM) presented and proved to be correct in [3].

There are two types of recoding applied in the R4MM. The first one (Recoding1) is Booth recoding [4] applied to the multiplier X . This recoding scheme translates conventional

radix-4 digits in the set $\{0, 1, 2, 3\}$ into the digit set $\{-2, -1, 0, 1, 2\}$. The recoded digit Z_i is obtained from the radix-4 multiplier digit $X_i = (x_{2i+1}, x_{2i})$ as:

$$Z_i = \text{Recoding1}(X_i, x_{2i-1}) = -2x_{2i+1} + x_{2i} + x_{2i-1}$$

where $i = 0, 1, 2, \dots, \frac{n}{2}-1$.

In order to make the two least-significant bits of the partial product S all zeros, a multiple of the modulus M , namely $q_{M_j}M$, is added to the partial product. This step is required to make sure that there are no significant bits lost in the right shift operation performed in step 10. To compute the digit q_{M_j} we need to examine the two least-significant bits of the partial product S generated in step 4 of the R4MM algorithm [1].

It is shown in [3] that q_M , as computed in step 5, satisfies the relation $q_M * M \equiv -S \pmod{4}$, which can be rewritten as: $S_{1..0} + q_M * M_{1..0} \equiv 0 \pmod{4}$ and represents the fact that the last 2 bits of S are zeros before the right shift is done in step 10.

It is easy to show from Booth encoding properties that the multiplier X is represented by digits of Z_j [5]. However, it is still necessary to show that *Recoding2* (step 5a of R4MM) generates an equivalent result. In order to do that we need to show that $q'_{M_j} \equiv q_{M_j} \pmod{4}$. It is well known that conventional quotient digits q_{M_j} are in the set $\{0, 1, 2, 3\}$. Applying a recoding function (*Recoding2*) we convert q_{M_j} to another digit set $\{-1, 0, 1, 2\}$. The recoding scheme consists in replacing $q_{M_j} = 3$ by the recoded value $q'_{M_j} = -1$. It makes the generation of multiples of M less complex. Based on the fact that $-1 \equiv 3 \pmod{4}$ it is possible to conclude that $q'_{M_j} \equiv q_{M_j} \pmod{4} \rightarrow q'_{M_j} * M \equiv q_{M_j} * M \pmod{4}$. and therefore the application of *Recoding2* generates a result that is congruent to the correct result, modulo M . In fact steps 5 and 5a of the R4MM algorithm can be executed in a single step. Two steps were shown for clarity only.

III. OVERALL ORGANIZATION

The architecture of the modular multiplier that implements the R4MM consists of 3 main blocks; *Datapath (or Kernel)*, *IO & Memory*, and the *Control block*. The computation shown in the R4MM algorithm takes place in the *kernel*. The complete design is presented and discussed in details in[6].

The *kernel* is organized as a pipeline of Processing Elements (PE), separated by registers. Each PE implements one iteration of the R4MM algorithm (steps 3 to 12).

A. Radix-4 Processing Element

The radix-4 PE is organized as shown in Figure 2. The main functional blocks in the PE are: *booth recoding*, *multiple generation (Mult Gen)*, *multi-precision Carry-save adders (MPCSA)*, q'_{M_j} *table*, and *registers (shaded boxes)*. The PE operates on w -bit words and

for this reason the *Mult Gen* and *MPCSA* modules are capable of storing and transferring carry bits from one word to the next. Shifting and word alignment is done by proper combination of signals and registers at the output of the last MPCSA. The design uses a re-timing technique explained in [3]. More details about these modules and their operation can be found in [6]

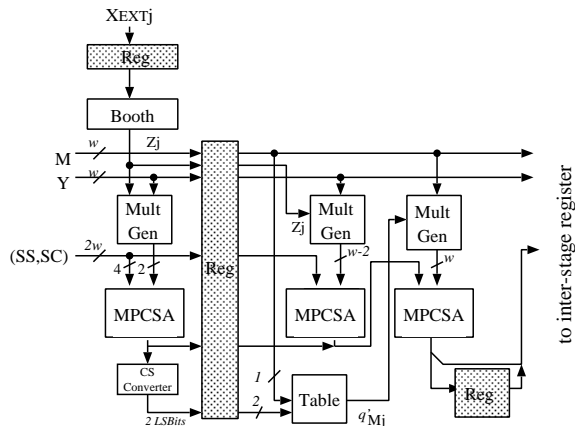


Fig. 2. PE Organization.

The Processing Element (PE) is divided in two sections. The first section (before the register) computes only the two least-significant (LS) bits of each word of $S + Z_i Y$. One can observe that q'_{M_j} depends on two LS bits of the data coming from the preceding PE in the pipeline: $(S_{1.0}^{(0)})$ and $Y^{(0)}$, and the recoded digit Z_j . The word size for S needs to be at least 4 bits in order to have the two LS bits of S generated as early as possible for the next PE.

A stage consists of a PE and a register. At each clock cycle, one word of Y , M , SS , and SC is applied as inputs to a stage. The multiplier digits X_i are transferred to PEs at specific times. The newly computed words of SS and SC , together with words of Y and M , are propagated by each stage to the next stage. This way, small PEs work concurrently to perform several iterations of the R4MM algorithm.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

The experimental data were generated using Mentor Graphics CAD tools. The radix-4 design presented in this paper was described in VHDL and simulated in ModelSim for functional correctness. It was synthesized using Leonardo synthesis tool for *AMI05_fast auto* ($0.5 \mu m$ CMOS technology with hierarchy preserved) provided in the ASIC Design Kit (ADK) from the same company. The experimental data for radix-2 and radix-8 kernel implementations were taken from [5], where the *AMI05_slow flattened* (no-hierarchy) technology was used. The flattened designs were laid-out using ICStation.

Figure 3 shows a comparison of performance between the radix-4 design discussed in this

paper, the radix-2 design in [2], and the radix-8 design in [3]. The data shows the time to compute the modular multiplication for 256-bit operands as a function of the design area. When a small number of gates is used, the radix-2 design performs as well as the radix-8 design. For areas of 9,000 gates or more, the radix-8 design is better than radix-2 design. The radix-4 design always perform better than the other two designs for all area values. Since the data in [5] was obtained using a slightly more detailed synthesis process, the data for radices 2 and 8 were synthesized again using the same steps and technology applied to the radix-4 design. These data are shown in the lines labelled *radix-2 fast* and *radix-8 fast* in the Figure. The data are slightly better than the one presented in [5], but the radix-4 solution is still significantly better than the other two cases. The complete data analysis is presented in [6].

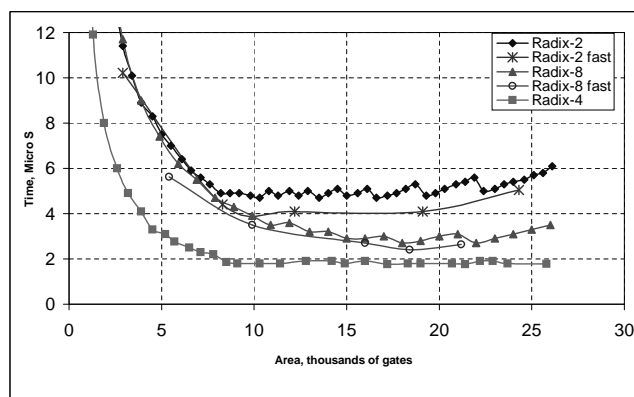


Fig. 3. Area \times time comparison between radix-2, radix-4, and radix-8 kernel, 256 operands.

V. CONCLUSION

We conclude that the proposed design made use of recoding schemes to reduce the complexity of multiple generation and multi-operand addition. The result was a design that is as slim as a radix-2 design, and can actually deliver twice the performance.

REFERENCES

- [1] P. L. Montgomery, “Modular multiplication without trial division,” *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, April 1985.
- [2] A. F. Tenca and C. K. Koc, “A word-based algorithm and scalable architecture for montgomery multiplication,” in *Cryptographic Hardware and Embedded Systems — CHES 1999*, Ç. K. Koç and C. Paar, Eds. 1999, Lecture Notes in Computer Science, No. 1717, pp. 94–108, Springer, Berlin, Germany.
- [3] A. F. Tenca, G. Todorov, and Ç. K. Koç, “High-radix design of a scalable modular multiplier,” in *Cryptographic Hardware and Embedded Systems — CHES 2001*, Ç. K. Koç and C. Paar, Eds. 2001, Lecture Notes in Computer Science, No. 1717, pp. 189–206, Springer, Berlin, Germany.
- [4] A. D. Booth, “A signed binary multiplication technique,” *Q. J. Mech. Appl. Math.*, vol. 4, no. 2, pp. 236–240, 1951.
- [5] G. Todorov, “ASIC design, implementation and analysis of a scalable high-radix Montgomery multiplier,” Master thesis, Oregon State University, USA, December 2000.
- [6] L. A. Tawalbeh, “Radix-4 ASIC Design of a Scalable Montgomery Modular Multiplier using Encoding Techniques,” M.S. thesis, Oregon State University, USA, October 2002.