Microelectronics Journal 55 (2016) 169–178

Contents lists available at ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

High-performance scalable architecture for modular multiplication using a new digit-serial computation

Abdalhossein Rezai [a,*], Parviz Keshavarzi [b]

[a] Academic Center for Education, Culture and Research (ACECR), Isfahan University of Technology (IUT) Branch, Isfahan 8415681167, Iran
[b] Electrical and Computer Engineering Faculty, Semnan University, Semnan, Iran

ARTICLE INFO

Article history:
Received 30 July 2015
Received in revised form
20 June 2016
Accepted 25 July 2016

Keywords:
Montgomery modular multiplication
Scalable architecture
Digit-serial computation
Public-key cryptography
Field-Programmable Gate Array (FPGA)
Application-Specified Integrated Circuit (ASIC)

ABSTRACT

Modular multiplication with a large modulus plays a vital role in many Public-Key Cryptosystems (PKCs) such as RSA and Elliptic Curve Cryptosystem (ECC). Montgomery modular multiplication algorithm is an efficient multiplication algorithm to simplify the quotient computation. The scalable architecture has been employed to perform the Montgomery modular multiplication with any precision of the modulus. This paper presents and evaluates a novel scalable modular multiplication algorithm/architecture with variable-radix. The proposed algorithm/architecture, which is based on a new digit-serial computation technique, parallelizes the data path to shorten the critical data path. It also reduces the complexity of the high-radix partial multiplications to binary partial multiplications. In this paper, we present implementation results on 0.18-μm ASIC technology, and on Xilinx Spartan 3, Virtex 2 and Virtex 6 FPGA. The results demonstrate that the proposed algorithm/architecture has area × time complexity and performance advantages compared to related algorithms/architectures.

1. Introduction

Modular multiplication with a large modulus has widely applications in cryptosystems such as Elliptic Curve Cryptosystem (ECC) [1,2], and RSA [3–7]. So, computational methods which simplify and accelerate the use of such operations are always of great value for system security [5,8].

Montgomery Modular Multiplication (M3) algorithm [9] is one of the most successful modular multiplication algorithms for computing the large modular multiplication. This algorithm replaces the division operation with a series of shift and addition operations, which are simple for hardware implementation [5,8,10,11]. The challenging issues in this algorithm are time-consuming carry propagations and using fixed-precision of the operands in hardware implementations [12,13].

Several designs and techniques have been proposed to improve the efficiency of the hardware implementation of the Montgomery modular multiplication which could be classified into three categories: (1) Carry Save Adder (CSA) designs [8,14], (2) high-radix designs [15–18], and (3) scalable design [7,12,13,19–36].

High-radix designs and CSA designs are well-known techniques to relax the problem of time-consuming carry propagations in hardware implementation of the Montgomery modular multiplication algorithm [8]. Using the high-radix design, the number of required clock cycles to complete the multiplication is reduced at the expense of the critical path overhead [8]. The total computation time in high-radix designs is also dependent on the impact of the more complex production, addition of partial products and the complexity of the control logic for selection of multiples of the multiplier operand [8].

To relax the problem of using fixed-precision of the operand, Tenca and Koc [27] introduced a scalable (word-based) Montgomery modular multiplication design, in which the modulus and multiplicand are processed word-by-word. The data path of scalable designs can perform Montgomery modular multiplication with any precision, and the precision of the operands is only limited by the memory capacity [12]. Several improvements have been presented in the literature, which show different accelerate performance of the scalable Montgomery modular multiplication [7,19–26].

Tenca et al. extended scalable Montgomery modular multiplication design [27] for several radixes [19,21,22]. They presented the implementation results for radix-8 [19], radix-2 [21], and radix-4 [22] scalable Montgomery modular multiplication. They concluded that the radix-4 scalable Montgomery always performed better than the radix-2 and radix-8 scalable Montgomery modular multiplications. Fan et al. [26] improved Tenca et al. design [19] to reduce the critical path using a parallel of computation technique. Harris et al. [20] improved [21] using left shift of partial products instead of right shift of multiplicand and modulus in computation of intermediate results of radix-2 scalable Montgomery modular multiplication. Kelly and Harris [25]

* Corresponding author.
E-mail addresses: rezaie@acecr.ac.ir, rezai560516@gmail.com (A. Rezai), pkeshavarzi@semnan.ac.ir (P. Keshavarzi).

http://dx.doi.org/10.1016/j.mejo.2016.07.012
0026-2692/

proposed very high-radix scalable Montgomery multiplier which reduces the latency. They also parallelized two multiplications within each Processing Element (PE). Jiang and Harris [23] improved Kelly and Harris design [25] using quotient pipelining. Pinckney and Harris [24] developed a parallelized radix-4 scalable Montgomery modular multiplication, which reduced PE cycle latency and critical path delay. Ibrahim et al. [7] proposed a systematic methodology for exploring possible processor array of scalable radix-4 Montgomery modular multiplication algorithm. They also presented the implementation results for scalable radix-8 architecture in GF(p) and GF(2^n) in [13].

This paper presents and evaluates an efficient scalable Montgomery modular multiplication algorithm/architecture. The main distinctive characteristics of our contribution are as follows:

1) Parallelize the data path and shorten the critical path in comparison with high-radix modular multiplication.
2) Achieve high-radix design with one clock cycle delay in data flow.
3) Process three operands (the zero chain multiplications, the required additions, and nonzero digit multiplication) in one clock cycle.
4) Support variable-precision and variable-radix modular multiplication feature.
5) Reduce the complexity of high-radix partial multiplication to binary partial multiplication.
6) Present the results on 0.18-μm ASIC technology, and on FPGA.

The results show that the proposed scalable Montgomery modular multiplication algorithm/architecture has the best performance in comparison with other scalable Montgomery modular multiplication algorithms/architectures and outperforms most of them in terms of area × time complexity.

The remaining of this paper is organized as follows. Section 2 briefly describes background of the scalable modular multiplication algorithm. Section 3 presents the proposed scalable modular multiplication. Section 4 provides in detailed hardware implementation of the developed scalable modular multiplication, and compares results to other architectures. Finally, paper is concluded in Section 5.

2. Background

Montgomery modular multiplication algorithm [9] is a common used modular multiplication algorithm in cryptography applications because it can replace the trial division with shift and addition operations [8,37–39]. The binary Montgomery modular multiplication for n-bit inputs multiplier X, multiplicand Y and modulus M is shown in Algorithm 1.

Algorithm 1. The binary Montgomery modular multiplication algorithm

Input: X, Y, M;
Output: S(n)=X.Y.R mod M;
1. S(0)=0;
2. For i=0 To n−1
3. $q_i=(S(i)+x_i.Y) \bmod 2$;
4. $S(i+1)=(S(i)+x_iY+q_i.M)/2$;
5. End For
6. IF S(n) ≥ M Then S(n)=S(n)−M;
7. Return S(n);

The output of this algorithm is S(n)=X.Y.R mod M where S(i) denotes S in the *i*th iteration, $x_i \in \{0, 1\}$ represents the *i*th bit of X, and $R=2^{-n}$ mod M.

The direct hardware implementation of the Montgomery modular multiplication can not support variable-precision multiplication feature [21,26]. Tenca and Koc [27] developed a scalable version of the Montgomery modular multiplication to support variable-precision multiplication feature. The binary scalable (word-based) version of the Montgomery modular multiplication is shown in Algorithm 2 [21].

Algorithm 2. The binary scalable Montgomery modular multiplication algorithm [21]

Input: X, Y, M;
Output: S=X.Y.R mod M;
1. S=0;
2. For i=0 To n−1
3. $(C_a, S^{(0)}) = x_i.Y^{(0)}+S^{(0)}$;
4. IF $S_0^{(0)} = 1$ Then;
5. $(C_b, S^{(0)})=S^{(0)}+M^{(0)}$;
6. For j=0 To e
7. $(C_a, S^{(j)})=C_a+x_i.Y^{(j)}+S^{(j)}$;
8. $(C_b, S^{(j)})=C_b+M^{(j)}+S^{(j)}$;
9. $S^{(j-1)}=(S_0^{(j)}, S_{w-1\ldots1}^{(j-1)})$;
10. Else
11. For j=0 To e
12. $(C_a, S^{(j)})=C_a+x_i.Y^{(j)}+S^{(j)}$;
13. $S^{(j-1)}=(S_0^{(j)}, S_{w-1\ldots1}^{(j-1)})$;
14. $S^{(e)}=0$
15. Return S;

In this algorithm, the n-bit multiplicand, modulus, and partial results are split into w-bit words as $Y = (0, Y^{(e-1)}, \ldots, Y^{(1)}, Y^{(0)})$, $M = (0, M^{(e-1)}, \ldots, M^{(1)}, M^{(0)})$, and $S = (0, S^{(e-1)}, \ldots, S^{(1)}, S^{(0)})$, where the words are marked with superscripts and $e = \left\lceil \frac{n+1}{w} \right\rceil$ denotes the number of words in each operand. The n-bit multiplier is shown as $X = (x_{n-1}, x_{n-2}, \ldots, x_1, x_0)_2$ where the bits are marked with subscripts. The total carry-out value generated per iteration corresponds to C_a+C_b, which is in the range [0, 2]. This algorithm computes a new partial results S for each bit of X, scanning the words of the present Y, M, and S. Once Y is completely processed, another bit of X is taken and the scan is repeated [21].

Hardware implementations of the binary scalable architectures are quite simple, but they have difficulty in improving the hardware area and efficiency. To achieve higher circuit efficiency, high-radix architectures have been proposed. The high-radix scalable version of the Montgomery modular multiplication is shown in Algorithm 3 [27].

In this algorithm, $M_{d\ldots0} = M \bmod 2^{d+1}$ and Booth recoding [40] is utilized to compute q_{Yi}. This recoding technique is utilized to reduce the complexity of the multiplication in the hardware implementation by reducing the Hamming weight of the multiplier [27]. This algorithm computes a new partial results S for each digit of X, scanning the words of the present Y, M and S. Once Y is completely processed, another digit of X is taken and the scan is repeated. In this algorithm, the number of required clock cycles is reduced from n-clock cycle to $\frac{n}{d}$-clock cycle for radix-2^d at the expense of the critical path overhead.