A Prediction-Based Scalable Design for Montgomery Modular Multiplication

De-Sheng Chen*, Huan-Teng Li, Yi-Wen Wang
Department of Information Engineering and Computer Science
Feng Chia University
Taichung, Taiwan, R.O.C.
email: dschen@fcu.edu.tw*

***Abstract*—Modular multiplication is a basic operation in public key cryptosystems, like RSA and elliptic curve cryptography (ECC). There are many algorithms to speed up its calculation. Among them, Montgomery algorithm is the most efficient method for avoiding expensive divisions. Recently, due to the increasing use of diverse embedded systems, variable precision modular multiplications with scalable architectures gain more and more attentions. In this paper, we propose a new word-based implementation of Montgomery modular multiplication. A predict policy is incorporated with a scalable architecture to reduce area cost and time latency. Compared with other scalable designs, our area-time product is the best among all, with little memory overhead.**

Keywords—Montgomery modular multiplication, word-based version of Montgomery algorithm, scalable architecture.

I. Introduction

Modular multiplication is regarded as a basic arithmetic operation widely used in many public key cryptosystems, such as RSA [1] and ECC [2]. These cryptographic protocols require the application of large modulus *M*, which is needed in repeated modular multiplications. Traditional algorithms use division by *M* to avoid calculation overflow, but it leads to low performance in hardware and software implementations. So far, the most efficient and popular modular multiplication method is Montgomery algorithm [3]. Its modular multiplication starts from the least significant position and uses division by a power of two, instead of division by *M*. As a result, the primary operations for Montgomery algorithm are just simple additions and shifts. For the sake of high performance, several designs [4], [5] had proposed the use of carry save adder (CSA) to prevent the long carry propagation delay.

To speed up the computation of Montgomery multiplication, various techniques had been reported, such as systolic architecture designs [6] [7] to get high throughput. However, most improved Montgomery modular multiplication methods were developed for fixed precision of operands, which are not applicable for variable precision multiplication. In [8], Tenca introduced a radix-2 word-based approach for Montgomery modular multiplication, as well as a scalable architecture design. But, due to the data dependency among words, Tenca's method requires the iteration process to take two more clock cycles to complete the multiplication. Several improved techniques, based on Tenca's radix-2 word-based Montgomery algorithm, had been proposed. In [9], Harris introduced left shifting *Y* and *M*, instead of right shifting *S*. In [10], Shieh defined a new math function for $S' = S - SM' / 2$ to solve the data dependency problem. In [11], Huang selected the result, from two possible values, based on the most significant bit to reduce the clock latency. Besides, a high radix word-based Montgomery algorithm was proposed to decrease the operation time, by using the Booth encoding technique [12]. Nevertheless, its computation logic requirement increases as well.

In this paper, we work on optimization design of hardware architecture of word-based Montgomery algorithm. We modify Huang's algorithm and incorporate a predict policy to the proposed scalable architecture to improve overall performance. This paper is organized as follows: Section II introduces Montgomery modular multiplication algorithms. Section III discusses a predict policy and the associated scalable architecture. Section IV shows experimental results, and Section V presents summary and conclusion.

II. Background

A. Montgomery Modular Multiplication

Montgomery modular multiplication algorithm replaces the trial division by simple addition and shift operations. Given an *n*-bit odd modulus *M* and a radix constant *R*, defined as r^n mod *M*. The radix number is $r = 2^d$, where *d* is radix parameter. Montgomery modular multiplication (MM) is defined as $S = X \times Y \times R^{-1}$ mod *M*, where *X* and *Y* are integers smaller than *M*. To use Montgomery multiplication in public key cryptosystems, we need to transform *X* and *Y* to $X \times R$ mod *M* and $Y \times R$ mod *M* through the use of constant value R^2 mod *M*. The computation $S = X \times Y \times R^{-1}$ mod *M* is repeatedly processed in Montgomery domain. A final step is needed to transform the result back to $S = X \times Y$ mod *M*. The radix-*r* MM algorithm is described as:

Algorithm Montgomery Modular Multiplication

Inputs: *M* (odd modulus, *n* bits), *X* (multiplier, *n* bits), *Y* (multiplicand, *n* bits), and $X, Y < M$.
Output: $S \equiv X \times Y \times R^{-1} \bmod M$, $R = r^n$, $r = 2^d$, $0 \leq S < M$.

```
1: $S = 0$
2: for $i = 0$ to $n - 1$ do
3:     $S = S + x_i \times Y$
4:     $q = (-S \times M^{-1}) \bmod r$
5:     $S = (S + q \times M) / r$
6: end for
7: if ($S \geq M$) then $S = S - M$
8: end if
9: return $S$
```

Since the convergence range of S is 0 to $2M$, the final output subtraction of Montgomery modular multiplication can be directly removed to avoid the extra subtraction $S = S - M$ [13].

B. Word-Based Montgomery Modular Multiplication

Tenca [8] proposed a radix-2 word-based Montgomery modular multiplication algorithm (OR2WMM), which takes two more clock cycles to complete multiplication. Huang's improved radix-2 word-based Montgomery algorithm (IR2WMM) solves this problem, and reduces the time latency from $2n$ to n. The n-bit modulus M and the multiplicand Y are partitioned into $e = \lceil (n+1)/w \rceil$ words, where w indicates word size. The intermediate result $S = SM + SR$, where SM contains the most significant bit of each word of S, and SR is the remaining part. The data dependency implies two possibilities, 0 and 1. The intermediate value S between iteration i and $(i + 1)$ is mapped to two pre-computed values, which will decide the final result. The IR2WMM algorithm is shown as:

Algorithm IR2WMM

```
Input: $M$ (odd modulus, $n$ bits), $X$ (multiplier, $n$ bits),
       $Y$ (multiplicand, $n$ bits), and then $X, Y < M$.
Output: $S \equiv X \times Y \times 2^{-n} \bmod M,\ 0 \le S < 2M$.

 1: for $i = 0$ to $n - 1$ do
 2:     $q = (x_i$ and $Y_0^{(0)})$ xor $S_0^{(0)}$
 3:     $(CO^{(1)}, SO_{w-1}^{(0)}, S_{w-2..0}^{(0)}) = (1, S_{w-1..1}^{(0)}) + x_i \times Y^{(0)} + q \times M^{(0)}$
 4:     $(CE^{(1)}, SE_{w-1}^{(0)}, S_{w-2..0}^{(0)}) = (0, S_{w-1..1}^{(0)}) + x_i \times Y^{(0)} + q \times M^{(0)}$
 5:     if $(SM^{(0)} = 1)$ then
 6:         $C^{(1)} = CO^{(1)}$
 7:         $SR^{(0)} = (SO_{w-1}^{(0)}, S_{w-2..1}^{(0)})$
 8:     else
 9:         $C^{(1)} = CE^{(1)}$
10:         $SR^{(0)} = (SE_{w-1}^{(0)}, S_{w-2..1}^{(0)})$
11:     end if

12:     for $j = 1$ to $e - 1$ do
13:         $(CO^{(j+1)}, SO_{w-1}^{(j)}, S_{w-2..0}^{(j)}) = (1, S_{w-1..1}^{(j)}) + C^{(j)} + x_i \times Y^{(j)} + q \times M^{(j)}$
14:         $(CE^{(j+1)}, SE_{w-1}^{(j)}, S_{w-2..0}^{(j)}) = (0, S_{w-1..1}^{(j)}) + C^{(j)} + x_i \times Y^{(j)} + q \times M^{(j)}$
15:         if $(SM^{(j)} = 1)$ then
16:             $C^{(j+1)} = CO^{(j+1)}$
17:             $SR^{(j)} = (SO_{w-1}^{(j)}, S_{w-2..1}^{(j)})$
18:         else
19:             $C^{(j+1)} = CE^{(j+1)}$
20:             $SR^{(j)} = (SE_{w-1}^{(j)}, S_{w-2..1}^{(j)})$
21:         end if
22:     end for
23: end for
24: return $S$
```

III. PROPOSED WORD-BASED MONTGOMERY MODULAR MULTIPLICATIN ALGORITHM

A. Prediction-Based Method

In 2005, David proposed an efficient method by using a lookup table (LUT) to reduce the area cost [14]. Four possible operands are pre-computed and selectively added to the intermediate result in the loop of conventional Montgomery algorithm. Based on this idea, we modify Huang's design [11] and implement a prediction-based scalable architecture for Montgomery multiplication. A new word-based predict policy (PP) is developed and shown below. The notation $OPA^{(j)}$ indicates the j-th word operator for four possible values 0, $Y^{(j)}$, $M^{(j)}$, $(Y + M)^{(j)}$.

Module Word-based PP

```
Input: $Y^{(j)}, M^{(j)}, (Y + M)^{(j)}, q, x_i$
Output: $OPA^{(j)}$
1: Switch $(q, x_i)$ :
2:     $(0, 0) : OPA^{(j)} = 0$
3:     $(0, 1) : OPA^{(j)} = Y^{(j)}$
4:     $(1, 0) : OPA^{(j)} = M^{(j)}$
5:     $(1, 1) : OPA^{(j)} = (Y + M)^{(j)}$
```

According to the above word-based PP, a predict policy radix-2 word-based Montgomery modular multiplication algorithm (PP-R2WMM) is proposed. Given an n-bit modulus M and a multiplicand Y, they are partitioned to $e = \lceil (n+1)/w \rceil$ words. The dependency graph of PP-R2WMM is similar to Huang's graph, and is shown in Fig.1. The main difference is the replacement of two CSAs with one CSA and one LUT.

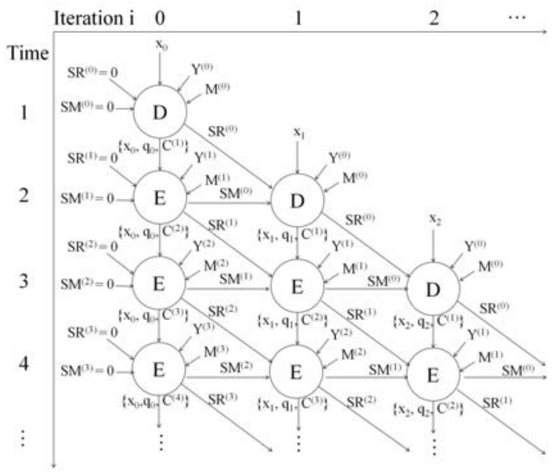

Fig.1. Data dependency graph of the PP-R2WMM algorithm.

The circle in the data dependency graph of Fig.1 represents a task of computation. The intermediate result $S = SM + SR$, where SM contains the most significant bit of each word of S and SR is the remaining part. The notation SR contains SRS and SRC in the carry save form. Task-D, for processing the first word, consists of three steps, the computation of q, the decision of $OPA^{(j)}$ of four possible values, and the calculation of two sets of possible results, where one of them will be selected as the final result by SM. Task-E processes the remaining words. The descriptions of Task-D and Task-E are given below,

Module Task-D

```
1: $q = SRS_0^{(0)}$ xor $SRC_0^{(0)}$ xor ($x_i$ and $Y_0^{(0)}$)
2: $OPA^{(0)}$ = Word-based PP ($Y^{(0)}, M^{(0)}, (Y+M)^{(0)}, q, x_i$)
3: $(CO^{(1)}, SO^{(0)}, SRS_{w\text{-}2..0}^{(0)}, SRC_{w\text{-}2..0}^{(0)}) = (1, SRS_{w\text{-}2..0}^{(0)}) + SRC_{w\text{-}1..0}^{(0)} + OPA^{(0)}$
4: $(CE^{(1)}, SE^{(0)}, SRS_{w\text{-}2..0}^{(0)}, SRC_{w\text{-}2..0}^{(0)}) = (0, SRS_{w\text{-}2..0}^{(0)}) + SRC_{w\text{-}1..0}^{(0)} + OPA^{(0)}$
5: if ($SM^{(0)} = 1$) then
6:     $C^{(1)} = CO^{(1)}$
7:     $(SRS_{w\text{-}2..0}^{(0)}, SRC_{w\text{-}1..0}^{(0)}) = (SO^{(0)}, SRS_{w\text{-}2..1}^{(0)}, SRC_{w\text{-}2..0}^{(0)})$
8: else
9:     $C^{(1)} = CE^{(1)}$
10:    $(SRS_{w\text{-}2..0}^{(0)}, SRC_{w\text{-}1..0}^{(0)}) = (SE^{(0)}, SRS_{w\text{-}2..1}^{(0)}, SRC_{w\text{-}2..0}^{(0)})$
11: end if
```

Module Task-E

```
1: $OPA^{(j)}$ = Word-based PP ($Y^{(j)}, M^{(j)}, (Y+M)^{(j)}, q, x_i$)
2: $(CO^{(j+1)}, SO^{(j)}, SRS_{w\text{-}2..0}^{(j)}, SRC_{w\text{-}2..0}^{(j)}) = (1, SRS_{w\text{-}2..0}^{(j)}) + C^{(j)} + SRC_{w\text{-}1..0}^{(j)} + OPA^{(j)}$
3: $(CE^{(j+1)}, SE^{(j)}, SRS_{w\text{-}2..0}^{(j)}, SRC_{w\text{-}2..0}^{(j)}) = (0, SRS_{w\text{-}2..0}^{(j)}) + C^{(j)} + SRC_{w\text{-}1..0}^{(j)} + OPA^{(j)}$
4: if ($SM^{(j)} = 1$) then
5:     $C^{(j+1)} = CO^{(j+1)}$
6:     $(SRS_{w\text{-}2..0}^{(j)}, SRC_{w\text{-}1..0}^{(j)}) = (SO^{(j)}, SRS_{w\text{-}2..1}^{(j)}, SRC_{w\text{-}2..0}^{(j)})$
7: else
8:     $C^{(j+1)} = CE^{(j+1)}$
9:     $(SRS_{w\text{-}2..0}^{(j)}, SRC_{w\text{-}1..0}^{(j)}) = (SE^{(j)}, SRS_{w\text{-}2..1}^{(j)}, SRC_{w\text{-}2..0}^{(j)})$
10: end if
```

The pre-computed operations produce two carry sets of $CO^{(j)}$ and $CE^{(j)}$ with 1 and 0 to merge with the final result ($SRS_{w\text{-}2..0}^{(j)}$, $SRC_{w\text{-}1..0}^{(j)}$) in carry save form. The PP-R2WMM algorithm that makes use of Task-D and Task-E modules is listed below:

Algorithm PP-R2WMM

```
Input:  $M$ (odd modulus, $n$ bits), $X$ (multiplier, $n$ bits),
        $Y$ (multiplicand, $n$ bits), and then $X, Y < M$,
        $Y + M$ (precomputed value, $n + 1$ bits).
Output: $S \equiv X \times Y \times 2^{-n} \bmod M, 0 \leq S < 2M$.
1: for $i = 0$ to $n - 1$ do Task-D Module
2:     for $j = 1$ to $e - 1$ do Task-E Module
3:     end for
4: end for
5: $Cc = 0$
6: for $j = 0$ to $e - 1$ do
7:     $(SS^{(j)}, SC^{(j)}) = SRS^{(j)} + SRC^{(j)} + SM^{(j)}$
8:     $(Cc, S^{(j)}) = SS^{(j)} + SC^{(j)} + Cc$
9: end for
10: return $S$
```

B. *Hardware Architecture of PP-R2WMM Algorithm*

A scalable architecture of PP-R2WMM algorithm is developed and shown in Fig.2. The architecture consists of six blocks: Kernel, LUT, *p*-shifter, Queue, CPA, and the reduction block. Basically, Kernel is the main computation part of PP-R2WMM algorithm. It consists of *p* PEs, and every PE is designed to execute Task-D and Task-E of PP-R2WMM. The LUT consists of a RAM and *p* PPs. Pre-computed values *Y*, *M*, *Y+M* are saved in RAM, and PPx passes the value $OPA^{(j)}$ to process element PEx of Kernel. The block diagrams of PP and PE are shown in Fig.3. Each PP consists of a simple four to two multiplexer and registers to decide $OPA^{(j)}$. Each PE contains a *q*-generator to generate *q* value and the data path is to compute the word-based modular multiplication. The details of *q*-generator and data path are shown in Fig.4.

The *p*-shifter is used to shift the multiplier *X* to the right by *p* bits. The Queue behaves as first-in and first-out (FIFO), if the intermediate values $SRS^{(j)}$, $SRC^{(j)}$, and $SM^{(j)}$ are hold, and some computed data are not finished. For the final result $S^{(j)}$, mergence and reduction by a word size CPA and the reduction block is required.

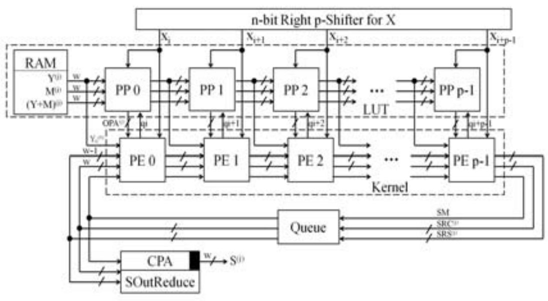

Fig.2. A proposed scalable architecture of the PP-R2WMM algorithm.

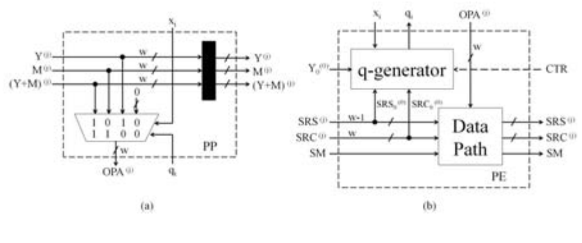

Fig.3. (a) Block diagram of PP. (b) Block diagram of PE.

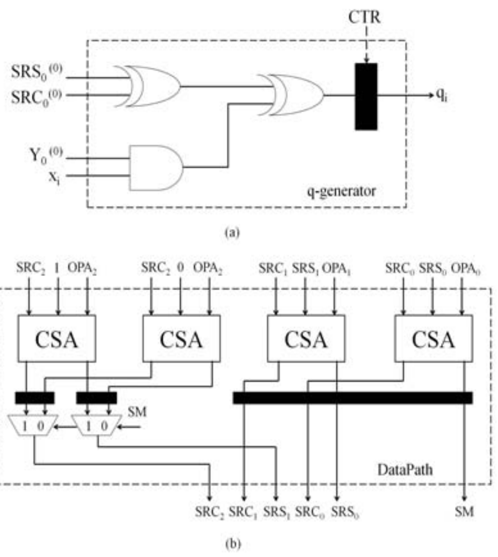

Fig.4. (a) Block diagram of *q*-generator. (b) Data path of PE (*w* = 3).

IV. Experimental Result

The proposed PP-R2WMM algorithm was coded in VHDL hardware description language and synthesized using the design platform of Altera foundation. The Altera FPGA EP2C70F896C6 is used as the benchmark test chip, and all experiments are conducted in Altera DE2-70 board. Our code was verified for word size w = 8-bit and tested for bit n = 512, 1024, and 2048. The performance and hardware area of three radix-2 scalable designs are shown in Table I. The IR2WMM [11] has less area × time than OR2WMM [8], mainly due to the reduced number of clock cycles by half, which also uses less registers than OR2WMM. Compare PP-R2WMM with IR2WMM, the former area is smaller than IR2WMM, because the kernel of PE is simplified to one CSA. But the extra cost is the memory usage as shown in Table II.

TABLE I Performance And HardwarE Area Of Three Radix-2 Scalable Architectures

Radix-2 Scalable Architectures				
Architecture Types	Resource and Performance	Precision Size		
		512-bit	1024-bit	2048-bit
Original Radix-2 Scalable Architecture [8] (w = 8-bit)	Max Frequency (MHz)	128.3		
	Number of PEs	33	65	129
	Min Clock Cycles (clks)	1,120	2,208	4,384
	Min Latency (us)	8.727	17.204	34.160
	Area (LEs)	2,259	4,436	8,788
	Min Latency x Area (us x LEs)	19,714	76,316	300,198
Improved Radix-2 Scalable Architecture[11] (w = 8-bit)	Max Frequency (MHz)	125.2		
	Number of PEs	65	129	257
	Min Clock Cycles (clks)	576	1,152	2,304
	Min Latency (us)	4.598	9.197	18.395
	Area (LEs)	3,194	6,330	12,602
	Min Latency x Area (us x LEs)	14,686	58,217	231,813
Proposed Predict Policy Radix-2 Scalable Architecture (w = 8-bit)	Max Frequency (MHz)	128.8		
	Number of PEs	65	129	257
	Min Clock Cycles (clks)	576	1,152	2,304
	Min Latency (us)	4.469	8.939	17.879
	Area (LEs)	2,732	5,420	10,796
	Min Latency x Area (us x LEs)	12,209	48,449	193,021

TABLE II Memory Resource Requirement Of Three Scalable Architecture Designs

Memory Resource						
Architecture Types	Resource Requirement (w = 8-bit)			Area (LEs)		
	512-bit	1024-bit	2048-bit	512-bit	1024-bit	2048-bit
Original Radix-2 Scalable Architecture	2x66xw	2x130xw	2x258xw	1,318	2,589	5,109
Improved Radix-2 Scalable Architecture	2x65xw	2x129xw	2x257xw	1,301	2,572	5,092
Predict Policy Radix-2 Scalable Architecture	3x65xw	3x129xw	3x257xw	2,179	4,212	8,318

V. Conclusion

In this paper, we propose a predict policy with a new scalable architecture for word-based Montgomery modular multiplication. The experimental result shows our area-time product is the best among all compared methods, with little memory overhead. In the future, the proposed techniques will be extended to consider various design metrics, such as parallel, scalable, and high-radix design.

References

[1] R.L.Rivest, A.Shamir, and L.Adleman, “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems,” *Comm. ACM*, vol.21, no.2, pp.120-126, February 1978.

[2] N.Koblitz, “Elliptic Curve Cryptosystems,” *Math. Computation*, vol.48, pp.203-209, January 1987.

[3] P.L.Montgomery, “Modular Multiplication without Trial Division,” *Math. of Computation*, vol.44, pp.519-521, April 1985.

[4] Ming-Der Shieh, Jun-Hong Chen, Wen-Ching Lin, and Hao-Hsuan Wu, “A New Algorithm for High-Speed Modular Multiplication Design,” *IEEE Transactions on Circuits and Systems*, vol.56, no.9, pp.2009-2019, September 2009.

[5] Hu Zhengbing, R.M.Al Shboul, V.P.Shirochin, “An Efficient Architecture of 1024-bits Cryptoprocessor for RSA Cryptosystem Based on Modified Montgomery's Algorithm,” *IEEE Workshop on IDAACS*, pp.643-646, September 2007.

[6] C.D.Walter, “Systolic Modular Multiplication,” *IEEE Transactions on Computers*, vol.42, no.3, pp.376-378, March 1993.

[7] C.McIvor, M.McLoone, and J.V.McCanny, “High-Radix Systolic Modular Multiplication on Reconfigurable Hardware,” *IEEE International Conference on Field-Programmable Technology*, pp.13-18, December 2005.

[8] A.F.Tenca and C.K.Koc, “A Scalable Architecture for Modular Multiplication Based on Montgomery's Algorithm,” *IEEE Transactions on Computers*, vol.52, no.9, pp.1215-1221, September 2003.

[9] 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.

[10] Ming-Der Shieh and Wen-Ching Lin, “Word-Based Montgomery Modular Multiplication Algorithm for Low-Latency Scalable Architectures,” *IEEE Transactions on Computers*, vol.59, no.8, pp.1145-1151, August 2010.

[11] Miaoqing Huang, Kris Gaj, and Tarek El-Ghazawi, “New Hardware Architectures for Montgomery Modular Multiplication Algorithm,” *IEEE Transactions on Computers*, vol.60, no.7, pp.923-936, July 2011.

[12] A.F.Tenca, G.Todorov, and C.K.Koc, “High-Radix Design of a Scalable Modular Multiplier,” *Proc. Third Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES'01)*, pp.189-205, 2001.

[13] C.D.Walter, “Montgomery Exponentiation Needs No Final Subtractions,” *Electronics Letters*, vol.32, no.21, pp.1831-1832, October 1999.

[14] David Narh Amanor, "Efficient Hardware Architectures for Modular Multiplication," *University of Applied Sciences Offenburg*, February, 2005.