Implementation of Montgomery Multiplier using Scalable Architecture

Satya Ranjan Das, Badri Narayan Sahoo

Abstract:- This paper describes the methodology and design of a scalable Montgomery multiplication module. This multiplier can manipulate any number of bits without any limitation. The size of a word depends upon the area which is available and also the performance which is required. After the general architecture is described, hardware organization is analyzed for implementing parallel computation and the discussions on design tradeoffs are done for recognising best configuration for hardware.

Keywords—montgomery multiplication, pipelining, processing element, modular multiplication..

I. INTRODUCTION

To perform multiplication efficiently, Montgomery multiplication[1] is chosen. It is used on general purpose systems such as microprocessors and signal processors. It performs multiplication by using some random modulus. To ease the computations as they are available in binary format, it performs division by a power of 2 instead by M. A variety[2][3][1] of algorithms are available on arithmetic or special coprocessors in order to obtain efficient software implementations. But our area of interest belongs to hardware implementations by Montgomery algorithm. It operates on perceptive representation of residue class modulo M[1].

A variety of hardware implementations and algorithms of Montgomery multiplication are available proposing limited accuracy of operands[1][4][3]. High-radix algorithms have been proposed in order to improve performance[3]. But they have a drawback of complexity and they occupy a notable area of chip. Not only this, the circuits developed by using high radix multiplication fail to ensure the desirable hike in speed. An investigatory case of a design in radix-4 montgomery multiplier for studying its tradeoffs is given below[5]. It can be seen that the use of digit multiplier increases with an increase in radix. Hence, leading to an increase in clock cycle times and design complexities. To overcome this drawback, low-radix designs are preferred to perform hardware implementation. To perform modular exponentiation[6][7] in RSA and Di e-Hellman PKC algorithms, Montgomery multiplication unit is a fundamental building unit[8][9].

For long integers the most interesting motive to perform inexpensive and fast modular multiplication is exponentiation as a series of modular multiplication in modular exponentiation chips. More emphasis is laid upon elliptic key cryptography [10] instead of discrete exponentiation over GF(2^k) and finitefield GF(2^k) [6]. A significant set of possibilities were witnessed in elliptic key cryptography by the introduction [6] of Montgomery multiplication in GF(2^k) .

To investigate design tradeoffs for limited chip area and various areas of design chip, this paper presents a scalable Montgomery multiplication architecture. First and foremost, a brief introduction of scalability requirement followed by a discussion on issues of Montgomery multiplier is given.

Later on, it is followed by explanation of word-based algorithm along with its step by step parallel evaluation. Based on this analysis, an architecture for modular multiplication is obtained and the design of module is presented. To give first order evaluation of performance of multiplier and area/time tradeoffs, simulations are performed.

II. SCALABILITY

An arithmetic unit said to be scalable if it is capable of being replicated or reused in such a way that long-precision results are produced independent of data path for which the actual design of unit was meant.

Consider an example of a system that needs 1,024 bits a multiplier that is designed for 768 bits cannot be immediately implemented. Hence, multiplier needs to be redesigned as the functions performed by low precision multipliers are not in compliance with those of large precision ones. Usually software and standard digit multipliers are implemented in order to perform hardware scalability. The algorithms that perform software computation of multiplication by Montgomery are given [6][2]. But radix-2 hardware implementation has much lower complexity in comparison with software-oriented algorithms. Given below, a hardware algorithm that offers good scalability and good performance design approach for Montgomery multiplication is presented.

III. MONTGOMERY MULTIPLICATION

Consider X and Y are two integers. The result obtained of required n bits after applying radix-2 Montgomery multiplication (MM) algorithm is as following:

$$Z = \mathrm{MM}(\bar{X}, Y) = XY\ r^{-l} \bmod \bar{M}; \quad (1)$$

Here M and $r=2^n$ denotes an integer falling between the range $2n-1 < M < 2n$ such that gcd(r,M) = 1. It is known that modulus M is an odd integer as r = 2n. To satisfy a condition of relatively prime, M is either itself a prime number or a product of two prime numbers in cryptographic applications.

Revised Manuscript Received on September 10, 2019.

Satya Ranjan Das, Dept. of Computer Sc. Eng, Siksha O Anusandhan Deemed to be University, Odisha, India.
(E-mail: satyadas@soa.ac.in)
Badri Narayan Sahoo, Dept. of Electronics & Communication Eng, Siksha O Anusandhan Deemed to be University, Odisha, India.
(E-mail: badrinarayansahu@soa.ac.in)

When an integer is transformed into another integer falling in the same range i.e., [0,M-1] by Montgomery algorithm, then it is known as image or M-residue of integer. $a=ar \bmod M$ is known as M-residue or image of a. The image c=MM$(a;b)$ is the image obtained by images a and b after performing Montgomery multiplication over them. It is to be noted that integer $c=ab \bmod M$[2]. The transformation between integer set and image can be accomplished by using MM as following:

- From the integer value to the M-residue: a = MM(a; $r2$) = $ar2r-1 \bmod M = ar \bmod M$.
- From the M-residue to the integer value: a = MM(a; 1) = $arr-1 \bmod M = a \bmod M$.

Since $r(\bmod M)$ and $r2(\bmod M)$ are saved after pre computing, either of these transformations are computed by a single MM.

The tradeoff of MM algorithm has lower complexity in comparison with conventional multiplication as latter needs a divisional operation.

In modular exponentiation, when several MMs are calculated over M –residues before translating back the result to its actual integer set, it offers as an advantage of MM. For m-bit operands, radix-2 MM algorithm $X=(x_{m-1};;;;;;;x_1;x_0)$, Y and M are as follows:

The Radix-2 Algorithm:
$S0 = 0$
for $i = 0$ to $m - 1$
if $(Si + xiY)$ is even
then $Si+1 := (Si + xiY)=2$
else $Si+1 := (Si + xiY + M)=2$
if $Sm \_ M$ then $Sm := Sm - M$ .....final correction step

A bit-shift multiplication is performed by this algorithm and addition, word-by-bit multiplication and bit-shift (division by 2) are also done. This algorithm is sufficient for doing hardware implementation because it consists of simple operations. Even condition is tested by to decide whether addition by M is required. For that, checking of least significant bit of partial sum(Si) is done. A hardware which is designed for m number of bits will not work for more bits.

IV. A MULTIPLE-WORD RADIX-2 MONTGOMERY MULTIPLICATION ALGORITHM

In circuit implementation of high-fan out signals, broadcast problem tends to boost the propagation delay. It can be reduced by using short precision words. The scalable hardware units of MM are supported by a word oriented algorithm. Hence we need to choose that algorithm that produces word-level output after performing bit-level computation.

We propose an algorithm in which the operand Y (multiplicand) is scanned word-by-word, and the operand X (multiplier) is scanned bit-by-bit. This decision enables us to obtain an efficient hardware implementation. We call it

Multiple Word Radix-2 Montgomery Multiplication algorithm (MWR2MM). We make use of the following vectors:

$$M = (M^{(e-1)}; \ldots; M^{(1)}; M^{(0)});$$
$$Y = (Y^{(e-1)}; \ldots; Y^{(1)}; Y^{(0)});$$
$$X = (x_{(m-1)}; \ldots; x_1; x_0);$$

where the words are marked with superscripts and the bits are marked with subscripts. The concatenation of vectors a and b is represented as (a,b). A particular range of bits in a vector a from position i to position j, $j > i$ is represented as $a_{j..i}$. The bit position i of the kth word of a is represented as $a_{(i)}^{k}$.

V. SCALABLE ARCHITECTURE DESIGNING

Following figure shows a pipeline which has two units for computation. The design of register file is featured in this organization. Register that work as circular shift register (rotator) store Y and M and others work as shifters because kernel receives word-serially data. The received digit is relayed to next unit in pipeline by processing elements. Except xi input, which is 1 bit wide all other paths are w bits wide. p-shift register yields the value of xi. In a pipeline, number of processing elements is denoted by p. A shift register is used for S because its contents are not used. The number of stages (n) and number of words (e) determines the length (L) of a shift register (S) in a pipeline as follows:

$$L = e + 2 - 2n \text{ if } (e + 2) > 2n$$
$$0 \text{ otherwise}$$

One memory element is connected to another in a looping or a chaining manner to implement these registers, without impacting clock cycle of entire system.

Multiplexers(MUXes) are connected between memory elements forming a chain to fulfill our need of loading capability for rotator. At last pipeline stage of algorithm, M and Y are loaded serially to eliminate implementation of too many MUXes. Thus, no multiplexer is needed between all memory elements but only between two memory elements. The delay of MUxes will not lead to formation of critical path in external circuit. For the sake of simplicity, the block of global control block is not included in this figure. Flip-flops are represented by shaded boxes.

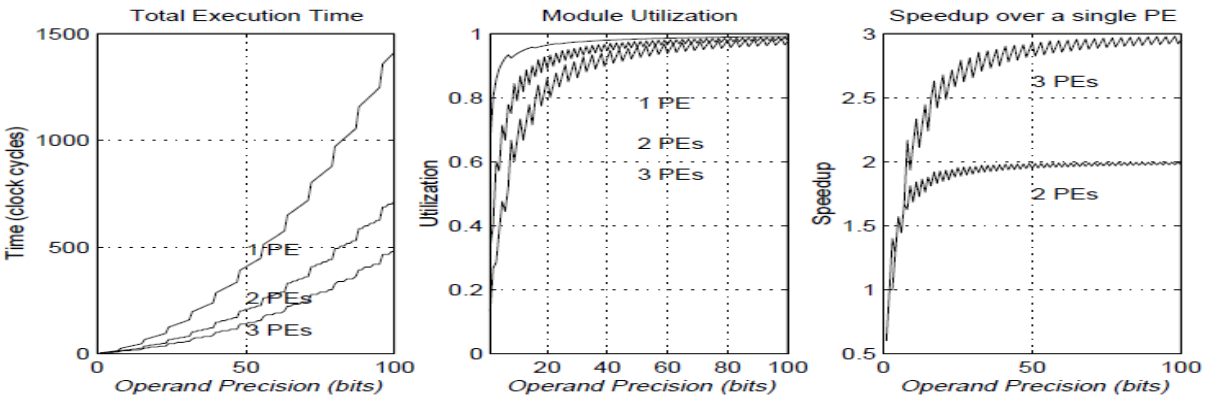


Fig. 1. The performance for multiple units with w = 8 bits.

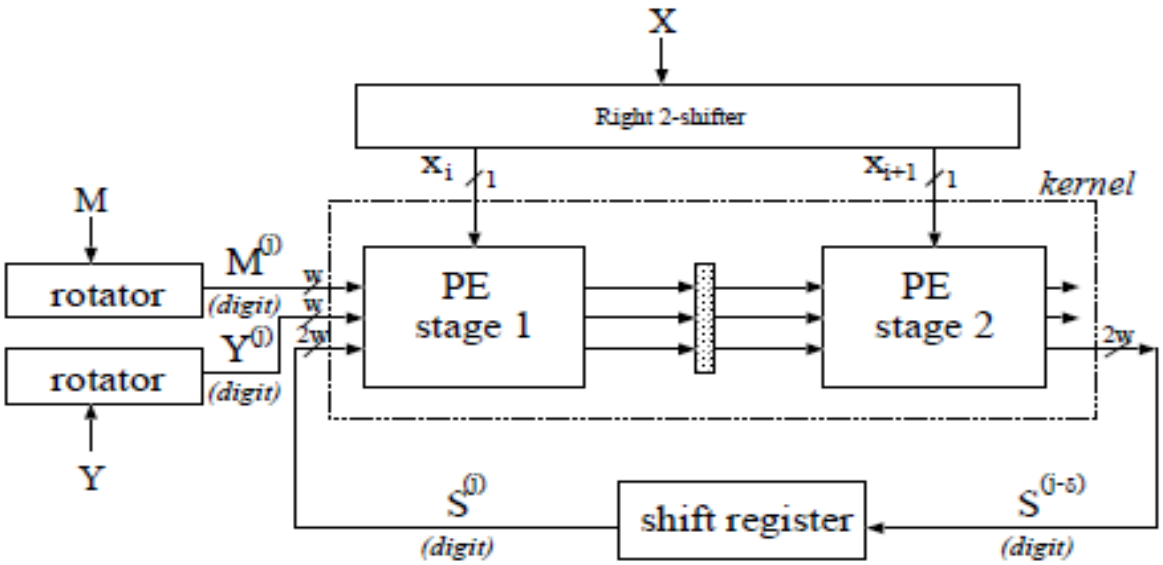


Fig. 2. Pipelined organization with 2 units

A. Processing Element

Figure 3 shows a block diagram of the processing element. Input from previous stage is received via data path and next S_{oj} is computed serially in a pipeline. Before sending input to next stage, one clock cycle delays the inputs. The non redundant form of *M,X* and *Y* are provided in order to reduce hardware and storage complexities.

The redundant carry-save form is produced after receiving internal sum S. Between units in each clock cycle, 2w bits per word are transferred in this case. Every pipeline cycle takes first ever computation step done by data path that is, computation S(j)+xiY(j information of least significant bit(t) is made available by data path. To manage addition of M(control signal c), only the value derived when least significant values of S and Y enter into the unit is utilised.

To relay some control signals to downstream modules and to store values during every pipeline cycle, local control is made responsible.

The customised ideology for least-significant-digit-first type of computation is followed up from [11]. Carry-save adders (CSA) are organised in two layers to form data path.

Considering a full accuracy structure for case w=1, as given in fig , a retiming process is proposed to obtain a design of serial circuit. Larger groups of adders are used if case for w>1 is considered. Not only because of arithmetic problem but also due to broadcast problem, a larger value of w may lead to higher cycle time. Xi and c change value for each pipeline cycle and are high fan out signals in design. The CSA structure must include bit-right-shift performed by data path.

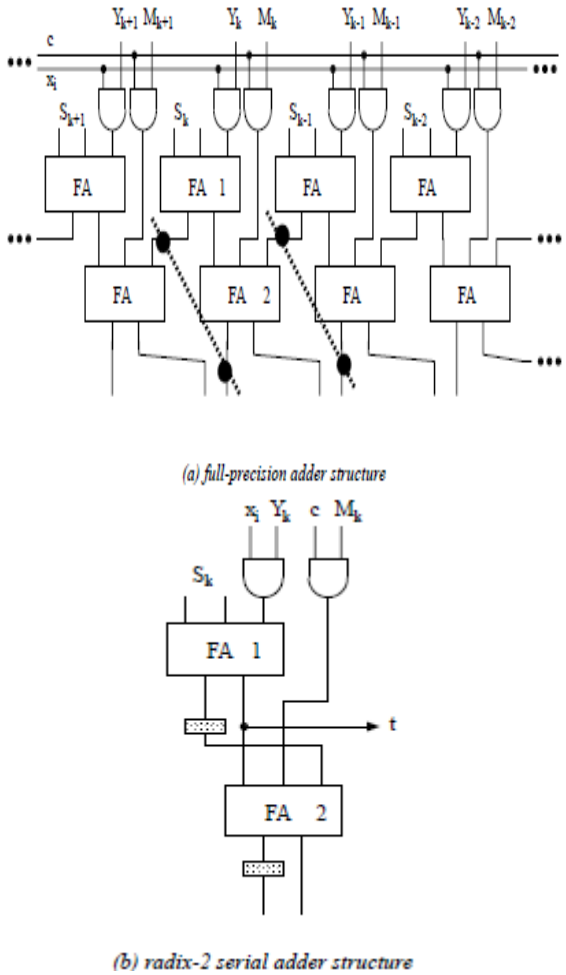
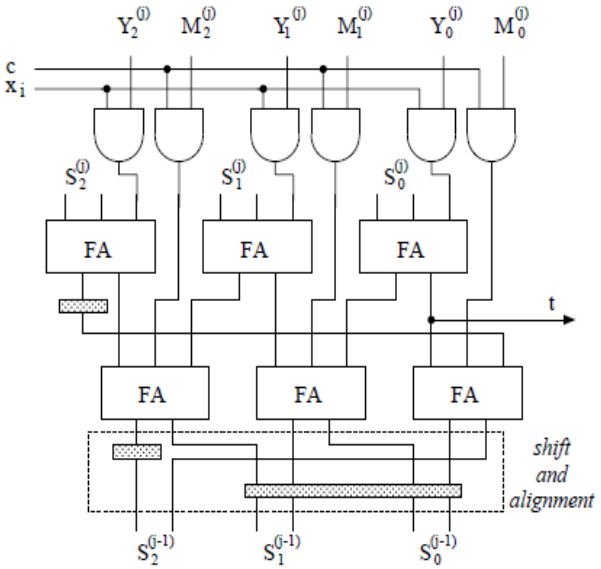


Fig. 3. Computation of the MM operations serially

Figure 4 shows the data path when case is w=3. The alignment and shift section are somewhat complicated to generate the next word S. The circuit produces most significant bit of $S_{(j-1)}$and w-1 bits of $S_{(j)}$. The bits of $S_{(j-1)}$ must be postponed and concatenated with MSB produced at step *j*.



V. AREA/TIME TRADEOFFS & RESULTS

For different word size *w*, operand precision *m* and pipeline organization, the area/time tradeoffs are different. Design constraint is given to area A. It is to be kept in mind that wiring area is not considered here. For small value of *w*, assume that propagation delay of PE is not dependent on *w* for approximation of first order. According to this hypothesis, comparison can be done between speed of different designs as clock cycle is same for every case. All registers utilise same area for modulus, operands and intermediate sum. To permit word-serial computation in a computational unit few extra cycles were added. As a result of which, case *w=m* has worst time of execution in the proposed scheme. Thus, we will think about the situation when we do not have enough chip area available for a full-precision conventional design.

Circuit is synthesized by 1.2 µm CMOS technology by VHDL tool on Mentor graphics. For a word size *w*, cell area is given by $A_{cell}(w)=47.2w$. The value 47.2 is the cost of area provided by the tool. Each inter-stage latch's area is significant during pipelined organization. It was calculated as $Alatch(w)=8.32w$. Thus, the area of pipeline having *n* units can be given as

$Apipe(n;w)=(n-1)Alatch(w)+nAcell(w)=55.52nw-8.32w.$

In a particular design, the maximum size of word that is utilised is found to be a function of number of pipeline stages *n* and that of available area *A*. It is given as:

$Apipe(n,w) \leq A$

$55.52nw - 8.32 \leq A$

$w \leq \frac{A}{55.52n-8.32}$

$wmax(A, n) = \frac{A}{55.52n-8.32}$

Total execution time for operands *m* is given as below:

$T(m,A,n)=\left\lceil\frac{m+1}{|n|}\right\rceil([(m+1)/wmax(A,n)] + 1)\text{-}1+2(n\text{-}1)$

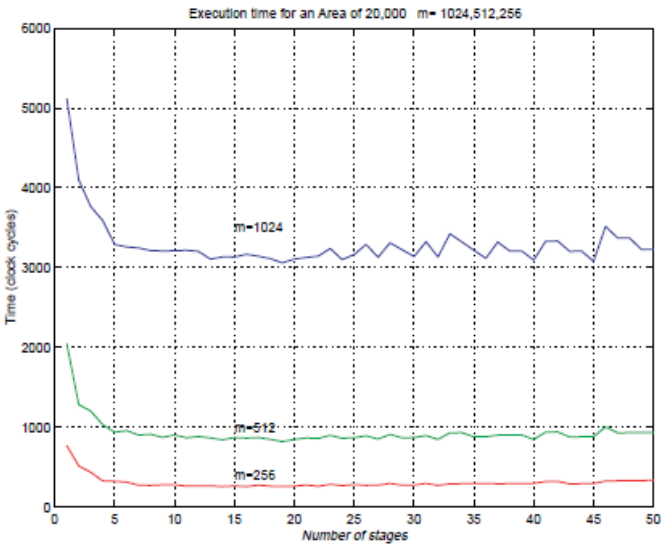


Fig. 5. The execution time of the MM hardware for various precision and configurations

With an increase in number of stages in a pipelined organization, the word size decreases when area is fixed as in table 1.

Table 1. The number of pipeline stages versus word size, for a fixed chip area

n (stages)	1	2	3	4	5	6	7	8	9	10
w (bits)	423	194	126	93	74	61	52	45	40	36

From the synthesis tools we also obtained a minimum clock cycle time of 11 ns (clock frequency of 90MHz). For the case m = 1024 bits, n = 10 stages, and w = 36 bits, the total execution time is 3107 * 11 = 34, 177 nanoseconds.

VI. CONCLUSION

The Montgomery multiplication is implemented by a new architecture. It can be accommodated to any chip area and is scalable according to operand of any size. It offers high flexibility and investigation of other design tradeoffs that are involved in performing Montgomery multiplication. It is clearly shown that pipelining into various units is better than s single unit working as large word length.

The usage of several number of units leads to the reduction in size of word, bandwidth and consequently the data paths. The circuit synthesized by a proposed data path's circuit is able to function upto frequencies of 90MHz. The chosen configuration of pipeline and availability of area defines the total time of computation by Montgomery multiplication.

VII. REFERENCES

1. P. L. Montgomery, "Modular Multiplication Without Trial Division," *Math. Comput.*, 2006.
2. Ç. K. Koç, T. Acar, and B. S. Kaliski, "Analyzing and comparing montgomery multiplication algorithms," *IEEE Micro*. 1996.
3. H. Orup, "Simplifying quotient determination in high-radix modular multiplication," 2002.
4. S. E. Eldridge and C. D. Walter, "Hardware Implementation of Montgomery's Modular Multiplication Algorithm," *IEEE Trans. Comput.*, 1993.
5. C. D. Walter, "Space/time trade-offs for higher radix modular multiplication using repeated addition," *IEEE Trans. Comput.*, 1997.
6. Ç. K. Koç and T. Acar, "Montgomery Multiplication in GF(2k)," *Des. Codes, Cryptogr.*, 1998.
7. T. Hamano, N. Takagi, S. Yajima, and F. P. Preparata, "O(n)-depth modular exponentiation circuit algorithm," *IEEE Trans. Comput.*, 1997.
8. R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, 1978.
9. W. Diffie, W. Diffie, and M. E. Hellman, "New Directions in Cryptography," *IEEE Trans. Inf. Theory*, 1976.
10. A. Menezes, *Elliptic Curve Public Key Cryptosystems*. 2011.
11. A. F. Tenca and M. D. Ercegovac, "A variable long-precision arithmetic unit design for reconfigurable coprocessor architectures," in *Proceedings - IEEE Symposium on FPGAs for Custom Computing Machines, FCCM 1998*, 1998.