RSA Cryptosystem Based on Early Word Based Montgomery Modular Multiplication

Rupali Verma[1](✉), Maitreyee Dutta[2], and Renu Vig[3]

[1] Punjab Engineering College, Chandigarh, India
rupali@pec.ac.in
[2] NITTTR, Chandigarh, India
[3] UIET, Panjab University, Chandigarh, India

Abstract. RSA is a public key cryptosystem in which encryption and decryption are modular exponentiation functions. Modular exponentiation is achieved by repeated modular multiplications. Montgomery modular multiplication is an efficient algorithm, hence is widely used for RSA public key cryptosystem. Performance of RSA depends on throughput of Montgomery modular multiplication. This paper presents RSA with Early Word based Montgomery modular multiplication. Early word based approach is scalable and Radix 4 Early Word Based Common Multiplicand Montgomery is proposed. RSA cryptosystem is implemented on virtex 5 FPGAs. The processing elements in Early Word based Montgomery use target device resources DSP48E for addition of operands. Two factors: algorithmic approach and use of target device resources have improved the performance of RSA on FPGAs.

Keywords: Early word based · FPGA · Montgomery · RSA

1 Introduction

RSA, a popular public key cryptosystem [1] provides two major functions: digital signatures and encryption, required for secure data communication. Digital signatures and encryption/decryption in RSA are mathematical expressions in terms of modular exponentiation. Binary, m-ary, addition chain, etc., are exponentiation techniques discussed by Koc [2]. All these techniques have modular multiplication as basic function to achieve modular exponentiation. In 1985 Montgomery [3] proposed Montgomery modular multiplication which is efficient and suitable for hardware and software implementations. It computes modular multiplication by addition and right shift rather than division by modulus.

For strengthening the security of cryptosystems, the size of modulus in RSA is 1024 bits or more which results in addition of long operands in modular multiplication. To avoid long carry propagation during addition, Montgomery modular multiplication designs are either word based architectures or carry save designs. A scalable word based Montgomery modular multiplication was proposed by Tenca and Koc [4] in which the multiplicand, modulus and intermediate results were processed word by word whereas the multiplier was taken bit by bit. Each iteration for multiplier bit requires a right shift of intermediate result, therefore in word based architectures a

A. Yang et al. (Eds.): SERVICES 2018, LNCS 10975, pp. 33–47, 2018.
https://doi.org/10.1007/978-3-319-94472-2_3

complete (j)th word is formed when (j + 1)th word for same iteration is computed. This dependency results in delay of two clock cycles between successive iterations for multiplier bits and latency of scalable Montgomery design [4] to 2k (where k is size of modulus) which is almost double that of carry save designs. Many word based designs exist in literature to reduce the clock cycle latency to one between successive iterations of multiplier bits. Harris et al. [5] proposed left shift of the multiplicand and modulus rather than right shift of intermediate result which reduced the latency to k cycles for k bit modular multiplication. A low latency scalable architecture was presented by Shieh and Lin [6] which deferred the accumulation of MSB of each word from (i + 1)th iteration to (i + 2)th iteration. Lin et al. [7] proposed word based Montgomery with low memory bandwidth by relaxing the dependency in word based architectures. A processing element interleaves operation of jth word from (i + 1)th to (i + w − 2)th iteration where w is word size. Huang et al. [8] proposed word based radix 2 architecture with the idea of computing partial results for MSB value 0 and 1. Though, the latency was reduced to k cycles but it doubled the area requirements. Chen et al. [9] incorporated prediction policy in Huang's design [8] to reduce area cost and time latency. The authors [8, 9] have used carry save adders for word based addition.

McIvor et al. [10] computed RSA modular exponentiation with carry save based Montgomery modular multiplication. Contrary to word based designs, the carry save Montgomery [10] takes the multiplicand, modulus and intermediate results in full precision. It completes k bit modular multiplication in k clock cycles but the area requirements are large as carry save adders require more resources when implemented on FPGAs.

This paper is an extension to our previous work [11]. Early word based radix 2 Montgomery modular multiplication (EWBR2MMM) and Early Word based radix 2 Common Multiplicand Montgomery modular multiplication (EWBR2CMMM) were proposed in our previous work [11]. Their comparative analysis with related work was also presented. This paper proposes early word based radix 4 Common multiplicand Montgomery modular multiplication (EWBR4CMMM). RSA serial and parallel modular exponentiation with EWBR2MMM and Montgomery Powering Ladder with EWBR2CMMM and EWBR4CMMM are implemented on FPGAs. Target device resources DSP48E are used for word based addition (word size 48 bits) on Virtex 5 FPGAs.

This paper is organized as follows. Section 2 briefly discuss RSA exponentiation. Section 3 presents Radix 4 Common Multiplicand Montgomery modular multiplication. Section 4 presents Early Word based Radix 4 common Multiplicand Montgomery and its dependency graph. Section 5 presents the implementation results of early word based Montgomery and RSA exponentiation on FPGAs and their comparison with related work. Section 6 presents conclusion.

2 RSA Exponentiation

Modular exponentiation is central to digital signatures, encryption and decryption in RSA public key cryptosystem. Many techniques for modular exponentiation are discussed in [2]. They differ in the number of modular multiplications to achieve modular exponentiation. Comparative analysis of exponentiation techniques [12] show m-ary and constant length sliding window are suitable for software based solutions whereas binary MSB (H) and LSB methods (L) are suitable for hardware implementations.

Sequential binary modular exponentiation (SBME) also called H algorithm or binary MSB [13] process the exponent bits from most significant to least significant. Parallel Binary Modular Exponentiation (PBME) [10] also called LSB or L method process exponent bits from least significant to most significant. Each exponentiation iteration in SBME and PBME require two operations: modular squaring and modular multiplication (if the exponentiation bit is set). In SBME, modular squaring and modular multiplication occur serially as data dependency exists between these operations whereas in PBME modular squaring and modular multiplication are computed in parallel. Therefore, SBME requires one Montgomery modular multiplication unit while PBME requires two Montgomery units that work in parallel.

Whether the exponent bit is set or unset, Montgomery Powering Ladder (MPL) computes both modular squaring and modular multiplication in parallel. This gives a regular structure and prevents simple fault and power attacks [14]. Also, Montgomery Powering Ladder gives the advantage of being parallelized and share a common multiplicand between modular squaring and modular multiplication [14]. Montgomery Powering ladder (MPLCMMM) computes modular exponentiation [15] with Common Multiplicand Montgomery modular Multiplication (CMMM). Montgomery Powering Ladder with Carry save Common Multiplicand modular multiplication was implemented in [16]. Section 3 presents Early Word based Radix 4 Common Multiplicand Montgomery Modular Multiplication (EWBR4CMMM).

3 Montgomery Modular Multiplication

Montgomery modular multiplication computes modular product S defined by S = A × B × r^{-1} mod n (where $r = 2^k$, k is bit length of operands A, B and n). It replaces the complex operation of trial division by modulus n with simple operations of addition and right shift within an iterative loop. After k iterations, if the result $S \geq n$ then a final subtraction S = S − n is required. But this requires comparison and then subtraction which is not a suitable operation on hardware. Walter suggested to increase the bit width of operands and value of r to 2^{k+2} [17] which increased the multiplier iterations from k to k + 2. As a result the final subtraction by modulus was removed. Within the iterative loop of Montgomery modular multiplication there are two operations: quotient computation and addition of operands followed by right shift. The authors [18] suggested simple quotient computation by making it independent of partial product. The dependency of quotient on partial product was removed by left shifting multiplicand and making LSB of multiplicand as zero.

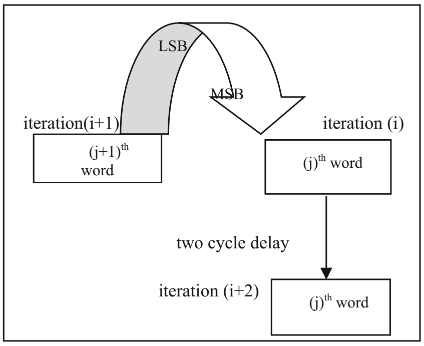

Fig. 1. Two cycle delay

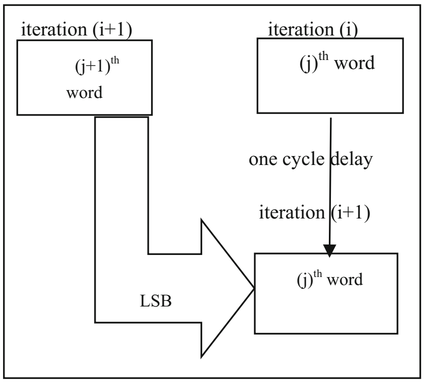

Fig. 2. Feed forward mechanism

The addition of operands in Montgomery design can be done either word by word or taking the complete operand k bits in size and adding using carry save adders. The word based Montgomery designs pipeline the computations which result in dependency on successor word and two cycle delay [4] between words of successive iteration as shown in Fig. 1.

RSA modular exponentiation requires modular squaring and modular multiplication. Common multiplicand Montgomery modular multiplication computes modular squaring and modular multiplication in parallel. Let P, R and n be k bit operands. Common multiplicand Montgomery reduces the common multiplicand P and computes two parallel accumulations [15] as shown in (1), (2) and (3).

$$T[i] = P.2^{-i}\text{mod n} \tag{1}$$

$$X = \sum\nolimits_{i=1}^{k} r_i.P.2^{-i}\text{mod n} \tag{2}$$

$$Y = \sum\nolimits_{i=1}^{k} p_i.P.2^{-i}\text{mod n} \tag{3}$$

Radix 2 and Radix 4 Common Multiplicand Montgomery in [15] are based on feed forward mechanism as shown in Fig. 2. Algorithm R4CMMM is radix 4 word based common multiplicand Montgomery modular multiplication [15]. The multiplier bits of P and R are taken two at time. The quotient is computed using Karnaugh map and is discussed in [15]. The quotient q[i] and variable $d[i]_j$ are both 2 bits. The intermediate result is calculated taking w − 2 bits of $T[i-1]^j$. The design is based on feed forward mechanism. Hence, two bits are added in next cycle as shown in step 9. The accumulation units compute X and Y word by word in pipeline to reduction.

```
Algorithm R4CMMM [15]
//Input: P and R are both (k+g) bit numbers, $g = 1 + \lceil \log_2(k+1) \rceil$, k=2h, g=2t
//$P = \sum_{i=0}^{h+t-1} p_i . 4^i$ and $R = \sum_{i=0}^{h+t-1} r_i . 4^i$   $p_i \in \{0, 1, 2, 3\}$ $r_i \in \{0,1,2,3\}$
//m=⌈ (k+g)/w⌉ operands partitioned into m words, n is the modulus with $2^{k-1}$<n<$2^k$
// gcd (n, 2) =1
//Output: $X = R.P.4^{-(h+2t)}\ mod\ n$   $Y = P.P.4^{-(h+2t)}\ mod\ n$
//$0 \le X < 2^{k+g}, 0 \le Y < 2^{k+g}$
1.  X=0, Y=0;
2.  T=P;
3.  for i=1 to h+2t do
4.  $q[i]_0 = T_0^0$ , $q[i]_1 = \overline{n_1}.\overline{T_1}.T_0 + T_1.(\overline{T_0} + n_1)$;
5.   $S[0] = 0, D[0] = 0, c_0 = 0$;
6.   for j=0 to m-1 do
7.     $S[i]^j = T[i-1]_{w-3...0}^j + q[i].n^j + c_j$ ;
8.     $d[i]_j = S[i]_{10}^j$ ;
9.     $c_{j+1}.2^w + T[i]^j = S[i]_{w-1......2}^j + d[i-1]_{j+1}.2^{w-2}$ ;
10.  end for;
11.    if t+1 ≤ i ≤ h+2t then
12.      for j=0 to m-1 do
13.        $(c_{r1} + c_{r2}).2^w + X^j = X^j + r_{k+2g-i}.T^j + c_{r1} + c_{r2}$;
14.        $(c_{p1} + c_{p2}).2^w + Y^j = Y^j + p_{k+2g-i}.T^j + c_{p1} + c_{p2}$;
15.      end for;
16.    end if;
17.   end for;
18.  Return X,Y
```

Early word based approach for Montgomery modular multiplication is presented in Sect. 4. Figure 3 presents the early word based approach which removes the dependency on successor word of same iteration (formed in next iteration) and computes the least significant bit of successor word with simple bit computations. Our previous work [11] presents early word based radix 2 Montgomery and early word based radix 2 Common Multiplicand Montgomery; and their comparative analysis with related work in literature. Early word based radix 4 Common Multiplicand Montgomery is proposed in next section.

4 Early Word Based Montgomery Modular Multiplication

4.1 Early Word Based Approach

Montgomery modular multiplication computes modular product iteratively. Each iteration requires a right shift. Due to right shift, word based Montgomery architectures depend on next word for its most significant bit. The compute early word based approach eliminates this dependency and maintains one cycle delay between same words for consecutive iterations.

In radix t designs, t bits of multiplier are scanned each time. The operands: multiplicand, modulus and intermediate result are taken word by word. Each iteration requires a right shift by t bits. Compute early word based approach computes t bits of next word (j + 1) which forms the t most significant bits for word (j). The variable $bt_{j(t-1....0)}$ which is t bits in size denotes t bits to be added to the least significant t bits of

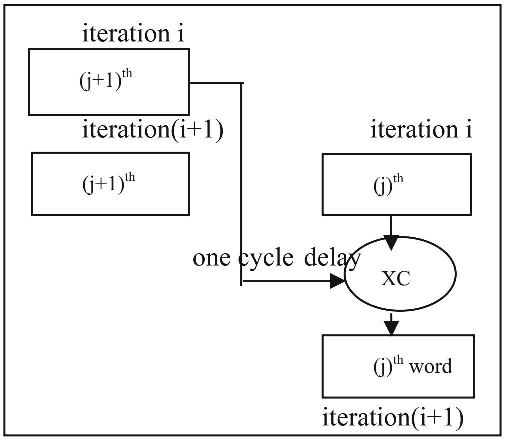

Fig. 3. Early word approach

Table 1. Notations in early word based Montgomery

Symbol	Meaning
A_i	ith multiplier bit
B_0^{j+1}	Least significant bit of word (j + 1) of B
BN_0^{j+1}	Least significant bit of word (j + 1) of BN (BN is sum of multiplicand B and modulus n)
BN^l	l^{th} word of BN
bt_j	Bit to be added to least significant bit of word (j + 1) of predecessor iteration
c_{j+1}	One bit carry for (j + 1) word generated by (j)th word
d_j	Computed LSB of word (j + 1)
q[i]	Quotient for ith iteration
n_0^{j+1}	Least significant bit of word (j + 1) of n
$PP[i]^j$	jth word of partial product for ith iteration
$S[i]_0^0$	Least significant bit (0) of word 0 of S[i]
$SP[i]^j$	jth word of partial result for iteration i (requires right shift)

word (j + 1) of predecessor iteration along with the carry generated by computation of word j. This requires t bit addition. The value of bt_j depends on multiplier bits and quotient bit and it is computed when the partial product for word (j) is decided. Early word based approach is simple and for radix t architectures it requires only t bit addition operations to compute the MSB for a word. Table 1 presents the notations used in Early Word Based Montgomery modular multiplication.

4.2 Proposed Early Word Based Radix 4 Common Multiplicand Montgomery Modular Multiplication

The proposed early word based radix 4 common multiplicand Montgomery modular multiplication is presented as algorithm EWBR4CMMM. The inputs P and R are represented in radix 4 notation. The bits p_i and r_i take value in set {0, 1, 2, 3}. The values twn and thn are precomputed so that each reduction iteration requires w bit addition. The quotient for each iteration is 2 bits in size. The detailed discussion of how quotient it is computed is given in R4CMMM [15]. For loop of step 3 runs for h + 2t iterations and reduces T word by word. For loop of step 6 computes intermediate results word by word. Based on quotient value, $PP[i]^j$ and $bt_{j(10)}$ are decided. Step 16 takes carry as input and computes w bit addition. In radix 4 designs each iteration requires a right shift by 2 bits. In early word based approach two least significant bits of (j + 1) word are computed which form the 2 most significant bits of j word. The variable $d_{j(10)}$ is 2 bits in size and it requires 2 bits addition with carry input. For loop of step 24 computes accumulation and is in pipeline to reduction. It runs from (t + 2) to (h + 2t + 1) iterations whereas the inner for loop run for words. Based on multiplier bits, the operands X and Y are accumulated with one of the values given below.

(i) 0 and 0 (ii) 0 and T (iii) T and T (iv) T and 2T.

Steps 26 and 27 perform carry save addition to get (w + 1) bits carry and w bits sum. The (+1) bit of carry is used as carry input for next word. The carry and sum are then added to w bit result with a carry output which forms the carry input for successor word. The idea of carry save addition and then w bit addition and handling two carry bits is in [15].

```
Algorithm EWBR4CMMM
//Input: P and R are both (k+g) bit numbers $g = 1 + \lceil \log_2(k+1) \rceil$, k=2h, g=2t
// $P = \sum_{i=0}^{h+t-1} p_i \, . \, 4^i$ and $R = \sum_{i=0}^{h+t-1} r_i \, . \, 4^i$  $p_i \in \{0,1,2,3\}$, $r_i \in \{0,1,2,3\}$
//Number of words m=⌈(k+g/w)⌉, n is the modulus $2^{k-1} < n < 2^k$, gcd (n, 2) = 1
//twn=2n , thn=3n  twn is (k+1) bits, thn is (k+2) bits
//Output: $X = P.R.4^{-(h+2t)} mod\ n$   $Y = P.P.4^{-(h+2t)}\ mod\ n$ $0 \le X < 2^{k+g}$,
//$0 \le Y < 2^{k+g}$
     1.  X=0, Y=0;
     2.  T[0]=P, S[0]=P, $c_0 = 0$;
     3.  for i=1 to (h+2t) do
     4.      $q[i]_0 = T[i-1]_0$;
     5.      $q[i]_1 = \overline{n_1} \, . \, \overline{T[i-1]_1} \, . \, T[i-1]_0 + T[i-1]_1 . (\overline{T[i-1]_0} + n_1)$;
     6.      for j=0 to m-1 do
     7.         if($q[i]_{10} = 00$) then
     8.             $PP[i]^j = 0$;  $bt_{j(10)} = 00$;
     9.         elsif($q[i]_{10} = 01$) then
    10.             $PP[i]^j = n^j$;  $bt_{j(10)} = n_{10}^{j+1}$;
    11.         elseif($q[i]_{10} = 10$) then
    12.             $PP[i]^j = twn^j$;  $bt_{j(10)} = twn_{10}^{j+1}$;
    13.         elseif($q[i]_{10} = 11$) then
    14.             $PP[i]^j = thn^j$;  $bt_{j(10)} = thn_{10}^{j+1}$;
    15.         end if;
    16.         $c_{j+1}, S[i]^j = T[i-1]^j + PP[i]^j + c_j$ ;
    17.         $d_{j(10)} = S[i-1]_{32}^{j+1} + bt_{j(10)} + c_{j+1}$ ;
    18.  //when i=1,$d_{j(10)} = S[i-1]_{10}^{j+1} + bt_{j(10)} + c_{j+1}$
    19.         $T[i]^j = (d_{j(10)}, S[i]_{w-1 \ldots 2}^j)$;
    20.      end for;
    21.  end for; // modular reduction of T
    22.  //parallel computation (3-21) and (24-29)
    23.  //hence pipelined
    24.  for i=t+2 to (h+2t+1) then
    25.     for j=0 to m-1 do
    26.         $(cx1_{j+1} + cx2_{j+1})$, $X^j = X^j + r_{k/2+g-i+1} . T[i-1]^j + cx1_j + cx2_j$ ;
    27.         $(cy1_{j+1} + cy2_{j+1})$, $Y^j = Y^j + p_{k/2+g-i+1} . T[i-1]^j + cy1_j + cy2_j$ ;
    28.     end for;
    29.  end for;
    30.  Return X, Y.
```

Figure 4 shows the first processing element PE0 for common multiplicand reduction. Unlike other PEs, it has additional task of quotient calculation. In radix 4 designs quotient is 2 bits. The two 4:1 MUX take 4 different inputs and the quotient bits select

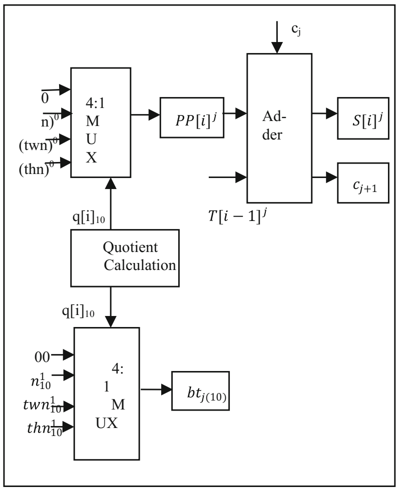

Fig. 4. PE0, the first processing element

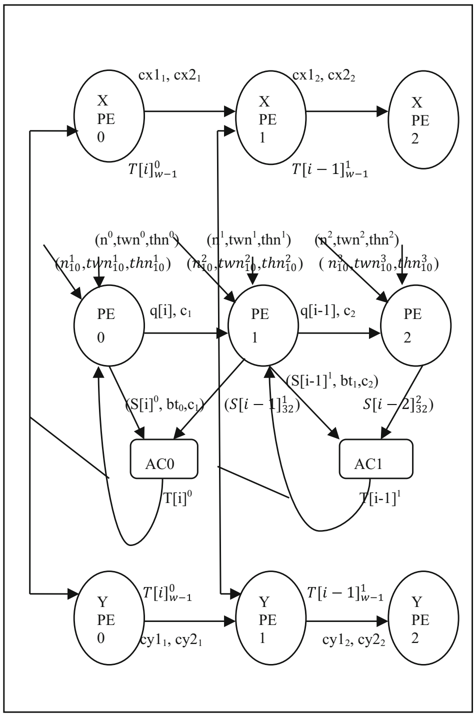

Fig. 5. Dependency graph of PE

the values for $PP[i]^j$ and $bt_{j(10)}$. The values $PP[i]^j$ and $T[i-1]^j$ are added by w bit adder. The adder takes carry input c_j and gives output $S[i]^j$ and carry bit c_{j+1}. The PEs from PE1 to PE(m-1) receive the quotient bits. These (m-1) PEs have two 4:1 MUX and w bit adder. Besides reduction PEs there are accumulation processing elements. Each accumulation processing element has 4:1 MUX, carry save adder and w bit adder. It receives 2 bits of multiplier which become selection inputs to the multiplexer to decide for operands to be added to X and Y.

Figure 5 shows the dependency graph of processing elements of EWBR4CMMM. The design has 3 m PEs: m PEs for modular reduction of T, m PEs for X accumulation and m PEs are for Y accumulation. The value of m depends on word size chosen and it gives us the number of words for P and R. The central PEs: PE0, PE1 and PE2 perform modular reduction of common multiplicand for words j = 0, 1 and 2 respectively. The inputs to each jth PE is (n^j, twn^j, thn^j) and (n_{10}^{j+1}, twn_{10}^{j+1} and thn_{10}^{j+1}). Each (j + 1)th PE receives quotient and carry bit from jth PE. Each jth PE is also dependent on (j + 1)th PE.

The AC_j represents the ADD and CONCATENATE unit. The (j + 1) PE gives $S[i-1]_{32}^{j+1}$ to AC_j. These bits are added with bt_j and carry bit c_{j+1} received from jth PE to form two least significant bits for (j + 1)th word which finally form the two most significant bits for jth word. The XPE and YPE are accumulating PEs. The XPEj and YPEj are dependent on central PEs for $T[i]^j$. The XPE (j + 1) and YPE (j + 1) are dependent on left neighbours XPEj and YPEj respectively for carry bits. Also, XPEj and YPEj pass the bit $T[i]^j_{w-1}$ to right neighbour XPE (j + 1) and YPE (j + 1) respectively. It is concatenated with left shifted $T[i]^{j+1}_{w-2--0}$ to form $(T[i]^{j+1}_{w-2...0}$ & $T[i]^j_{w-1})$. This value is one of the operands for addition when multiplier bits are 11. Such ideas for bit transfer have been presented in [15].

5 Comparison and Implementation Results

This section compares the early word based designs with related work in literature. Design efficiency is determined by two factors: time and area. Cycle count and critical path delay determine the design time. Cycle count is the number of cycles taken to get the desired output. Critical path delay is the longest path in design which determines the cycle time for a design. Table 2 gives the cycle count and critical path delay of Montgomery modular multiplication. Here k refers to bit length of modulus, m for number of words and g is for common multiplicand designs.

Table 2. Time complexity of Montgomery modular multiplication

Design	Cycle count	Critical path delay
[15] Radix 4	k/2 + k/w + g/2	2:1 MUX + (w − 2) bit Full addition + 2 bit full addition + 4:1 MUX + 2 AND + 3 OR
(Proposed) EWBR4CMMM Radix 4	k/2 + g + m + 1	4:1 MUX + w bit addition + 2 bit addition + 2 OR + 1 NOT + 1 AND

The EWBR4CMMM is radix 4 design where 2 bit quotient is computed as presented in [15]. To reduce the path delay in proposed design, basic operations like NOT of n_1, $T[i-1]_1$ and $T[i-1]_0$ are computed in parallel. The radix 4 design [15] takes input carry bit and add operands of length (w − 2) and w bit which result in w bit addition due to propagation of ripple carry. If the worst case is taken during addition it will generate a carry bit which needs to be handled during two bit feed forward addition in [15]. This overhead will add to complexity in radix 4 design [15].

Table 3 present the results of Montgomery modular multiplication. Early word based designs: EWBR2MMM, EWBR2CMMM and EWBR4CMMM are coded in VHDL and synthesized in Xilinx ISE Design Suite 12.4. The word size chosen in early word based Montgomery designs is 48 bits (w = 48) because target device resources DSP48E are used for 48 bit addition. It adds two 48 bit operands with input carry and gives 48 bit sum with output carry. Our previous work [16] used DSP48E for addition (carry and sum were added to get binary result). The target device for early word based radix 2 Montgomery modular multiplication (bit size = 1024) is xc5vlx50t package ff665 speed grade −3. The design has 22 PEs: each PE use DSP48E for addition; and one DSP48E is used for precomputation. The synthesis results of EWBR2MMM for 1024 bits gives area in terms of 6624 slice registers, 5718 LUTs and 23 DSP48Es. The frequency is 390.83 MHz and it takes 2.68 μs to complete one 1024 bit modular multiplication.

Table 3. Results of Montgomery modular multiplication

Design	Bit size (bits)	FPGA tech/others	w	PEs	Area	Freq (MHz)	Clock cycles	T (μs)
[19] (Radix 32)	1024	XC5VLX110T	32	32	4588 Slices/7932 LUTs 63 DSP48E, 3(1024 bit) BRAM + 8(256 bit) BRAM	100	201	2.01
	2048	XC5VLX110T	32	32	4588 slices/7932 LUTs 63 DSP48E, 3 (2048 bit) BRAM + 8(512 bit) BRAM	100	820	8.2
[20] (Fully Systolic)	1024	Virtex 5	16	–	11652 slices, 67 DSP48Es	140.028	280	1.99
			32	–	11346 slices, 138 DSP48Es	92.649	140	1.51
[20] (Parallelized)	1024	Virtex 5	16	–	2242 Slices 18 DSP48Es	161.614	712	4.4
			32	–	3044 Slices 38 DSP48Es	95.913	356	3.71
[21] (Radix 16)	1024	Xilinx V5	–	–	14440 LUT 7826 Flip Flops 66 Multiplier	120	199	1.6
EWBR2MMM	1024	xc5vlx50t	48	22	6624 Slice Regs, 5718 LUTs 23 DSP48Es	390.83	1051	2.68
	2048	xc5vlx155t	48	43	12933 Slice Regs, 12500 LUTs, 44 DSP48Es	388.42	2096	5.38
EWBR2CMMM	1024	xc5vlx155t	48	66	12084 Slice Regs, 8644 LUTs 66 DSP48Es	418.04	1071	2.5
	2048	xc5vlx330t	48	129	23591 Slice Regs,16901 Slice LUTs 129 DSP48Es	391.09	2118	5.4
EWBR4CMMM	1024	xc5vlx155t	48	66	12134 Slice regs, 13956 LUTs 66 DSP48Es	357.45	547	1.52
	2048	xc5vlx330t	48	129	23781 slice registers, 27284 LUTs 129 DSP48Es	306.08	1081	3.52

EWBR2MMM (bit size = 1024) requires almost one-third DSP48E as compared to radix 32 Montgomery [19] and fully systolic Montgomery modular multiplication design [20] (word size 16 bits). When compared with fully systolic Montgomery [20] with word size 32 bits, EWBR2MMM has almost one sixth requirement of DSP48E. Area requirements of EWBR2MMM (1024 bits) are less compared to radix 32 Montgomery [19], fully systolic design [20] and radix 16 Montgomery [21] at the cost of more time to complete modular multiplication. But when compared to fully parallelized design [20] (word size 16 and 32 bits), EWBR2MMM takes less time with similar area requirements.

EWBR2MMM (bit size = 2048) is implemented on target device xc5vlx155t package ff1136 speed −3. RSA exponentiation (PBME) for 2048 bits require 88 DSP48E. Therefore, the component EWBR2MMM (2048 bits) is also implemented on xc5vlx155t. The device xc5vlx155t has 128 DSP48E whereas there are only 48 DSP48E available on xc5vlx50t. EWBR2MMM (2048 bits) has 43 PEs and requires almost double area and time as compared to EWBR2MMM (1024 bits). EWBR2MMM completes 2048 bit modular multiplication in 5.38 μs whereas radix 32 MMM [19] takes 8.2 μs which is 1.5 times that of former. Early word based radix 2 and radix 4 common multiplicand Montgomery modular multiplication (EWBR2CMMM and EWBR4CMMM) for 1024 bits is implemented on xc5vlx155t FPGA package ff1136 speed −3. Both the designs EWBR2CMMM and EWBR4CMMM have 66 PEs: 22 PEs for reduction, 22 PEs for X accumulation and 22 PEs for Y accumulation. The word size is 48 bits. Therefore each PE requires 48 bit addition. Hence, total number of DSP48E used are 66. EWBR2CMMM (1024) takes 2.5 **μ**s and EWBR4CMMM (1024) takes only 1.52 μs to complete modular multiplication. EWBR4CMMM takes less time as compared to EWBR2CMMM as number of cycles of former are reduced to half. EWBR4CMMM takes less time to complete modular multiplication compared to radix 32 Montgomery [19] and radix 16 Montgomery [21]. Radix 16 Montgomery [21] requires 66 hardcore multipliers, radix 32 Montgomery [19] requires BRAMS and 63 DSP48E whereas EWBR4CMMM (1024 bits) requires 66 DSP48E. All the above designs also require additional resources mentioned in Table 3. When compared with fully systolic design [20], EWBR4CMMM has less area requirements at the cost of 0.01 μs more compared to design [20] with word size 32 bits.

EWBR2CMMM and EWBR4CMMM for 2048 bits are implemented on xc5vlx330t FPGA package ff1738 speed −2 as the designs require 129 DSP48E. The area requirements for 2048 bits are almost double the area requirements for 1024 bits. The time for 2048 bits is slightly more than twice the time for 1024 bits. This is because the chosen target device xc5vlx330t has speed grade −2. EWBR4CMMM for 2048 bits takes less than half the time taken by 2048 bit MMM [19]. When 2048 bits EWBR2CMMM is compared with MMM [19] the area requirements of former are more compared to latter but the latter takes almost 1.5 times the time taken by former to complete modular multiplication.

Table 4 present the results of RSA exponentiation. The value of e is 65537 and its binary representation has two bits with binary value 1. RSA parallel and serial binary exponentiation: PBME and SBME with EWBR2MMM for operand size 1024 bits is implemented on xc5vlx50t. The frequency of both the designs for 1024 bits is 390.83 MHz but the area requirement of PBME is approximately double that of SBME due to two Montgomery modular multipliers working in parallel: one computing modular multiplication and other computing modular squaring. RSA modular

Table 4. Results of RSA exponentiation (public exponent e = 65537)

	Bit size	FPGA tech	Area	Freq (MHz)/clock period	Clock cycles	Throughput (Mbps)
[22]	1024	xc5vlx50t	7158 slices	274	19494	14.5
PBME (EWBR2MMM)	1024	xc5vlx50t	19430slice regs, 13166 LUTs 46 DSP48Es	390.83	17867	22.3
	2048	xc5vlx155t	38153 slice regs 25925 LUTs 88 DSP48Es	387.31	35632	22.26
SBME (EWBR2MMM)	1024	xc5vlx50t	11806 Slice regs 6787 LUTs 23 DSP48Es	390.83	19969	20.04
	2048	xc5vlx155t	23218 slice regs 13516 LUTs 44 DSP48Es	387.31	39824	19.91

exponentiation: PBME and SBME for 2048 bits is implemented on xc5vlx155t because PBME requires 88 DSP48E; for comparison purposes SBME is also implemented on same target device. The frequency and throughput of PBME and SBME for 1024 and 2048 bits is more compared to the attack resistant exponentiation [22] where modular squaring and modular multiplication are computed in parallel.

Table 5 presents the time complexity of Montgomery Powering ladder designs. The number of cycles of MPLCMMM with radix 4 common multiplicand Montgomery [15] are less than of MPLCMMM with EWBR4CMMM respectively due to sharing of reduction cycles between modular multiplications which can also be extended to proposed word based design. MPLCMMM with proposed designs: EWBR2CMMM and EWBR4CMMM has major operation of common multiplicand modular multiplication with additional 2:1 MUX to decide whether P or R will be common multiplicand. This delay has been considered and added to critical path for MPLCMMM in our work.

Table 5. Time complexity of Montgomery Powering Ladder

	Critical path delay	Clock cycles
[15] Radix 4	$T_{wbit\ FA} + T_{MUX\ 2:1} +$ $T_{MUX\ 4:1} + 2\ T_{AND} + 3\ T_{OR}$	k (k/2 + k/w + g/2)
MPLCMMM (EWBR4CMMM)	2:1 MUX + 4:1 MUX + w bit addition + 2 bit addition + 2 OR + 1 NOT + 1 AND	(k + 2) (k/2 + g + m + 1)

Table 6 presents the results of Montgomery Powering Ladder. MPL is fault and simple power attack resistant (FA SPA). MPLCMMM with radix 2 and radix 4 designs: EWBR2CMMM and EWBR4CMMM are coded in VHDL and synthesized in Xilinx ISE 12.4 Design suite. The target device for 1024 bits is xc5vlx155t FPGA package ff1136 speed grade −3. MPLCMMM implemented with radix 2 and radix 4 designs has higher frequency than their counterparts [15]. Time for 1024 bit exponent cycles for 1024 bit modular exponentiation requires only 2.6 ms for MPL with radix 2 design and 1.58 ms with radix 4 design. MPL for 2048 bits is implemented on xc5vlx330t package ff1738 speed grade −2 due to requirement of 129 DSP48E. When compared with radix 32 design [19], MPLCMMM with EWBR2CMMM takes less time for 2048 bits. Radix 32 design [19] takes almost 1.5 times the time taken by MPLCMMM with EWBR4CMMM for 1024 bits and twice of MPLCMMM with EWBR4CMMM for 2048 bits.

Table 6. Results of Montgomery Powering Ladder (MPL)

MPL design	Bit size	FPGA tech	Area	Freq (MHz)	Clock cycles	Time (ms)	FA SPA resistance
[15] Radix 2	1024	XC5VLX50T	3218 Slices	345	1097870	3.18	Yes
[15] Radix 4	1024	XC5VLX50T	5225 Slices	290	566431	1.95	Yes
[19] Radix 32	1024	XC5VLX110T	–	100	198656	2.5	Yes
	2048	XC5VLX110T	–	100	1679360	16.793	Yes
MPLCMMM (EWBR2CMMM)	1024	xc5vlx155t	17245 Slice Regs 11101 LUTs 66 DSP48Es	405.3	1098846	2.6	Yes
	2048	xc5vlx330t	33990 Slice Regs 21960 slice LUTs 129 DSP48Es	379.56	4341900	11.439	Yes
MPL CMMM (EWBR4CMMM)	1024	xc5vlx155t	18283 Slice Regs 16428 LUTs 66 DSP48Es	354.4	561222	1.58	Yes
	2048	xc5vlx330t	35987 slice regs 32554 slice LUTs 129 DSP48Es	300.91	2216050	7.36	Yes

6 Conclusion

This paper presented RSA implementation on FPGAs with early word based radix 2 and radix 4 designs. The early word based approach is simple and requires basic operations to compute most significant bit in word based architectures. The word size chosen was 48 bits due to DSP48Es which adds 48 bits and operates at high frequency. Addition is the critical operation in word based Montgomery designs and it determines the cycle time of Montgomery design. The use of DSP48Es for addition and early word based approach for bit determination has improved the performance of word based Montgomery and RSA on FPGAs.

References

1. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)
2. Koc, C.K.: High-Speed RSA Implementation, RSA Labs, Redwood City, CA, Technical report (1994)
3. Montgomery, P.L.: Modular multiplication without trial division. Math. Comput. **44**(170), 519–521 (1985)
4. Tenca, A.F., Koc, C.K.: A Scalable architecture for modular multiplication based on montgomery's algorithm. IEEE Trans. Comput. **52**(9), 1215–1221 (2003)
5. Harris, D., Krishnamurthy, R., Anders, M., Mathew, S., Hsu, S.: An improved unified scalable radix 2 Montgomery multiplier. In: Proceedings of the 17th IEEE Symposium on Computer Arithmetic (ARITH), pp. 172–178 (2005)
6. Shieh, M.D., Lin, W.C.: Word based Montgomery modular multiplication algorithm for low latency scalable architectures. IEEE Trans. Comput. **59**(8), 1145–1151 (2010)
7. Lin, W.C., Ye, J.H., Shieh, M.D.: Scalable Montgomery modular multiplication architecture with low-latency and low-memory bandwidth requirement. IEEE Trans. Comput. **63**(2), 475–483 (2014)
8. Huang, M., Gaj, K., Ghazawi, T.E.: New hardware architectures for Montgomery modular multiplication algorithm. IEEE Trans. Comput. **60**(7), 923–936 (2011)
9. Chen, D.S., Li, H.T., Wang, Y.W.: A prediction-based scalable design for Montgomery modular multiplication. In: Proceedings of 3rd International Conference on Electric and Electronics, pp. 46–50 (2013)
10. McIvor, C., McLoone, M., McCanny, J.V.: Modified Montgomery modular multiplication and RSA exponentiation techniques. IEE Proc. Comput. Digit. Techniques **151**(6), 402–408 (2004)
11. Verma, R., Dutta, M., Vig, R.: Early-word-based Montgomery modular multiplication algorithm. In: Proceedings of IEEE 2nd International conference on SPIN, pp. 595–600 (2015)
12. Sutter, G.D., Deschamps, J.-P., Imana, J.L.: Modular multiplication and exponentiation architectures for fast RSA cryptosystem based on digit serial computation. IEEE Trans. Ind. Electron. **58**(7), 3101–3109 (2011)
13. Shieh, M.D., Chen, J.H., Wu, H.H., Lin, W.C.: A new modular exponentiation architecture for efficient design of RSA cryptosystem. IEEE Trans. VLSI Syst. **16**(9), 1151–1161 (2008)

14. Joye, M., Yen, S-M.: The Montgomery powering ladder. In: Kaliski, B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 291–302. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_22
15. Wu, T., Li, S., Liu, L.: Fast, compact and symmetric modular exponentiation architecture by common multiplicand Montgomery modular multiplication. Integr. VLSI J. **46**, 323–332 (2013)
16. Verma, R., Dutta, M., Vig, R.: Carry save common multiplicand Montgomery for RSA cryptosystem. Am. J. Appl. Sci. **11**(5), 851–856 (2014)
17. Walter, C.D.: Montgomery exponentiation needs no final subtractions. Electron. Lett. **32**(21), 1831–1832 (1999)
18. Elridge, S.E., Walter, C.D.: Hardware implementation of Montgomery's modular multiplication algorithm. IEEE Trans. Comput. **42**(6), 693–699 (1993)
19. Mesquita, D., Perin, G., Herrmann, F.L., Martins, J.B.: An efficient implementation of montgomery powering ladder in reconfigurable hardware. In: Proceedings of 23rd Annual Symposium on Integrated Circuits and Systems Design SBCCI, pp. 121–126 (2010)
20. Perin, G., Mesquita, D.G., Herrmann, F.L., Martins, J.B.: Montgomery modular multiplication on reconfigurable hardware: fully systolic array vs parallel implementation. In: Proceedings of IEEE VI Southern Programmable Logic Conference, pp. 61–66 (2010)
21. Wang, P., Liu, Z., Wang, L., Gao, N.: High radix montgomery modular multiplier on modern FPGA. In: Proceedings of 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communication, pp. 1484–1489 (2013)
22. Fournaris, A.P.: Fault and simple power attack resistant RSA using Montgomery modular multiplication. In: Proceedings of 2010 IEEE International Symposium on Circuits and Systems, pp. 1875–1878 (2010)