Low Space Complexity, High performance Unified and Scalable Word-Based Radix 4 Architecture for Montgomery Modular Multiplication in GF(P) and GF(2^k)

ATEF IBRAHIM
Salman Bin-Abulaziz University & Electronics Research Institute
Department of Computer Engineering & Department of Microelectronics
SAUDI ARABIA & EGYPT
atef@ece.uvic.ca

Abstract: This paper presents a novel low space complexity and high performance (low power and high speed) unified and scalable word-based radix 4 architecture for Montgomery modular multiplication in GF(P) and GF(2^k). In this architecture, the multiplicand and the modulus words are allocated to each processing element rather than pipelined between the processing elements as in the previous architectures extracted by L. Tawalbeh, and also the multiplier bits are fed serially to the first processing element of the processor array every odd clock cycle. To reduce multiplier area and accelerate its operation, the hardware architecture employs 3-to-2 carry save adders instead of 4-to-2 carry save adder, as used in conventional designs, to avoid carry propagation at each addition operation of the add-shift loop. To reduce power consumption, glitch blockers are employed at the outputs of some circuit modules to reduce the spurious transitions and the expected switching activities of high fan-out signals. Moreover, the architecture was modified to reduce more power by replacing the dual field conventional 3-to-2 carry save adder (CSA) by modified low power dual field 3-to-2 CSA that has internal logic structure with balanced delays in SUM and CARRY outputs to reduce the chance of glitches occurrence. An ASIC Implementation of the proposed architecture shows that it can perform 1024-bit modular multiplication (for word size $w = 32$) in about 4.81 μs. Also, the results show that it has smaller $Area \times Time$ values compared to existing competing designs by ratios ranging from 13.1% to 77.2% which makes it suitable for implementation where both area and performance are of concern. Also, it has higher throughput over them by ratios ranging from 2.6% to 82.9%. In addition, it achieves a decrease in power consumption compared to these designs by ratios ranging from 25.3% to 70.4%.

Key–Words: Montgomery Modular Multiplier, Cryptography, Systolic Arrays, ASIC Design, Hardware Security

1 Introduction

Modular addition, Modular multiplication and modular inversion in prime field GF(p) and binary extension filed GF(2^k) are the essential math operations that have numerous applications in cryptosystems, such as DiffieHellman key exchange algorithm [1], Elliptic Curve Digital Signature Algorithm [2], decipherment operation of RSA algorithm [3], and elliptic curve cryptography [4, 5]. The most critical of these math operations is the modular multiplication operation since it is the center operation in numerous cryptographic functions. Therefore, the performance of any cryptography application depends to a large extend on the efficient implementation of the modular multiplication operation. For resource constraint applications, it is necessary to develop efficient implementation for this operation that takes into account savings in space and at the same time achieve high performance (i.e., high speed and low power consumption).

A famous approach for computing the modular multiplication operation in hardware is based on the Montgomery modular multiplication algorithm [6, 7]. This algorithm replaces the trial division by modulus with a series of addition and shifting operations which are easy to implement in hardware. Therefore, the critical operation is basically the a three operand addition inside an iteration loop. Unfortunately, The long carry chains of the carry ripple adders when used to add long operands seriously influence the performance of cryptography system. Accordingly several approaches to keeping away from long carry chains during the addition operation have been proposed to achieve a significant speed up of Montgomery modular multiplication.

The most important approach is the implementation of the modular multiplier architectures using Carry Save Adders (CSAs) (e.g., [9]-[12]), but these implementations are not desirable for resource constrained applications that they have a significant increase in hardware complexity and power consumption [13, 14]. Moreover, most of the previously published CSA-based scalable Montgomery multipliers did not pay enough attention to the power consumption issue. Consequently, this paper focuses on reducing the power consumption with only a slight area overhead. Previous published works such as [15, 16] have developed techniques to reduce the power consumption of Montgomery multipliers. The work in [15] designed a low-power Montgomery multiplier architecture that composed of ripple-carry adders by employing the custom CMOS design of several basic building blocks, including logic gates, full adder, and D flip-flop. In [16], some latches named glitch blockers are located at the outputs of some circuit modules to reduce the spurious transitions and the expected switching activities of high fan-out signals in the radix-4 scalable Montgomery multiplier. In [17], the authors modified the CSA-based radix 2 Montgomery algorithm to bypass the iterations that perform superfluous carry-save addition and register write operations in the add-shift loop. In this paper, we attempt to reduce the power consumption of CSAs in the CSA-based unified and scalable radix 4 Montgomery multiplier using technique that is different from [15, 16, 17]. The goal is achieved by modifying the conventional dual field 3-to-2 CSA to have internal logic structure, based on the multiplexing of the Boolean functions XOR/XNOR and AND/AND, to obtain balanced delays in sum and carry outputs that reduce the chance of glitches occurrence. There are several papers [18], [19] used the balanced delay technique to reduce the chance of glitches occurrence. In [18], the authors modified the Wallace Tree multiplier architecture to reduce spurious activity further by introducing technique that combines transmission gates with level-restoring static CMOS gates. This combination suppresses glitches via RC low pass filtering. In [19], the authors presented a carry skip adder with balanced critical paths. To achieve balanced delay, they grouped the input bits into variable sized carry skip blocks. This grouping reduces dynamic power by minimizing extraneous glitches. These techniques are totally different from the design technique used in this paper in that they applied to architectures that have carry ripple chain, but our technique is applied to architecture that has unified carry save adders. Moreover, the technique used in the design of [18] was applied at the transistor level, but our technique was applied at the logic level.

The main idea of the architecture proposed in this paper is based on the observation that the Montgomery multiplication algorithm for both fields GF(p) and GF(2^k) are essentially the same algorithm. The proposed unified architecture performs the Montgomery multiplication in the field GF(p) generated by an arbitrary prime p and in the field GF(2^k) generated by an arbitrary irreducible polynomial p(x). This proposed unified architecture provides a significant savings in area and power when both types of modular multipliers are needed.

In this paper, the first author modifies his earlier architecture [37] by proposing a unified Montgomery algorithm and carrying out a hardware implementation based on replacing the conventional 3-to-2 CSA by modified dual field 3-to-2 CSA taking the advantage of processing two operand words by the same processing element (PE) of the processor array. The internal logic structure of the modified dual field 3-to-2 CSA will result in reduced power consumption with only a slight area overhead. A unified word-based and scalable radix 4 architecture is proposed that can handle operands of any precision.

This paper is organized as follows. Section 2 presents the unified and scalable Multiple-Word Radix-4 Montgomery Multiplication (MWR4MM) algorithm with recoding [29]. Section 3 describes the proposed processor array architecture. Section 4 describes several techniques to decrease power consumption. Section compares the ASIC implementation of proposed processor array architecture with the previous existing competing architectures. Finally Section 6 concludes the paper.

2 MWR4MM algorithm

The notation used in this paper is as follows:

- $Fsel$: Field select bit, selects between GF(p) or GF(2^k)
- M : modulus.
- m_j : a single bit of M at position j.
- A : multiplier operand.
- a_j: a single bit of A at position j.
- B : multiplicand operand.
- n : operand size (in number of bits).
- R : a constant (called a Montgomery parameter), $R = 2^n$.
- qa_j : coefficient determines the multiples of the multiplicand B ($qa_j \times B$).
- qm_j : coefficient determines the multiples of the modulus M ($qm_j \times M$).

- S : intermediate partial product , or final result of modular multiplication.
- w : word size (in number of bits) of either B, M or S.
- $e = \lceil \frac{n}{w} \rceil$: number of words in either B, M or S.
- C_a, C_b : carry bits.
- $(B^{(e-1)}, ..., B^{(1)}, B^{(0)})$: word vector of B.
- $(M^{(e-1)}, ..., M^{(1)}, M^{(0)})$: word vector of M.
- $(S^{(e-1)}, ..., S^{(1)}, S^{(0)})$: word vector of S.
- $S^{(i)}_{(k-1...0)}$: bits $k-1$ to 0 from the i^{th} word of S.

Algorithm 1 shows the steps of the unified and scalable MWR4MM algorithm. This algorithm is a modified version of the Multiple-Word High-Radix (Radix = 2^k) Montgomery Multiplication (MWR2^kMM) algorithm presented in [22], [28]. It differs from it in that it uses a recoding scheme to recode qm_j [29], it works for both fields GF(p) and GF(2^k) and it is represented in a regular iterative form. The inputs of this algorithm are A, B, M and $Fsel$. The output is the partial product S that is represented in a carry save form (SS, SC). For multiplication in GF(2^k) the carry vector SC and carry bits Ca and Cb are always kept zero. Since prime field GF(p) uses carry-save arithmetic and binary extension field GF(2^k) is free from carry, a modified carry-save adder, named as Dual Field Adder (DFA) is used to force the carry to zero. The DFA is a conventional full adder with additional $Fsel$ signal. When $Fsel$ is '1' the carry C is computed, whereas when $Fsel$ is '0' it is always zero which is the desired property of GF(2^k) addition. In steps 3 to 8 of Algorithm 1, the output word S initialized to zero. Through steps 11, 13, 15 and 16, The word vectors of S is computed by performing the addition $(Fsel \times C, S^{(i)}) = S^{(i)} + X$, where X is either $(qa_j \times B)$ or $(qm_j \times M)$. In steps 17 and 20, the word vector S is shifted right by two bits to implement the division by four operation included in this algorithm.

we will use PP and MM to represent a partial product $qa_j \times B$ and a multiple of modulus $qm_j \times M$, respectively.

In the case of radix-4, qa_j , qm_j are 2-bit numbers. Thus the value sets of PP and MM are as follows:

$$\text{PP} \in \{0, A, 2A, 3A\}, \quad \text{MM} \in \{0, M, 2M, 3M\}$$

to calculate $3A$ and $3M$ on the fly, we need two extra adders. To remove the burden of calculating $3A$ in the PP's value set, a modified Booth recoding scheme is popularly used [29], see Table 1.

Algorithm 1 Regular iterative Unified and Scalable MWR4MM Algorithm.

```
 1: Inputs: $A, B, M, Fsel$
 2: Outputs: $S = (SC, SS)$
 3: for $i = 0$ to $e$ do  "Initialization"
 4:     $S^{(i)}_{-1} = 0$
 5: end for
 6: for $j = 0$ to $n-2$ do
 7:     $S^{(e)}_{j} = 0$
 8: end for
 9: for $j = 0$ to $\lceil n/2 \rceil - 1$ do
10:     $qa_j = Booth(a_{2j+1 \cdots 2j-1}) = Booth(A_j)$
11:     $(Fsel \times C_a, S^{(0)}_j) = S^{(0)}_{j-1} + (qa_j \times B)^{(0)}$
12:     $qm_j = Montg(S^{(0)}_{j(1 \cdots 0)} \times (4 - (M^{(0)}_{j(1 \cdots 0)})^{-1}) \bmod 4)$
13:     $(Fsel \times C_b, S^{(0)}_j) = S^{(0)}_j + (qm_j \times M)^{(0)}$
14:     for $i = 1$ to $e-1$ do
15:         $(Fsel \times C_a, S^{(i)}_j) = S^{(i)}_{j-1} + (qa_j \times B)^{(i)}$
16:         $(Fsel \times C_b, S^{(i)}_j) = S^{(i)}_j + (qm_j \times M)^{(i)}$
17:         $S^{(i-1)}_j = (S^{(i)}_{j(1 \cdots 0)}, S^{(i-1)}_{j(w-1 \cdots 2)})$
18:     end for;
19:     $C_a = C_a \text{ or } C_b$
20:     $S^{(e-1)}_j = signext(C_a, S^{(e-1)}_{j(w-1 \cdots 2)})$
21: end for;
```

The problem of calculating $3M$ cannot be solved by the Booth recoding scheme. L. Tawalbeh [29] adopted a specialized recoding method to transform the original value set of MM into the one that also has easily obtainable elements only. We will give this method the name "Montgomery recoding scheme". Let $(sp_{0,1}, sp_{0,0})_2$ be the 2 bits in the Least Significant Digit(LSD) of SP = S + PP and $(m_{0,1}, m_{0,0})_2$ be the 2 bits in the LSD of M. According to the input condition that M has to be odd [22][29], $m_{0,0}$ is always '1'. Then, Montgomery recoding scheme takes a bit stream $(sp_{0,1}, sp_{0,0}, m_{0,1})_2$ as input and generates a recoded MM according to Table 2, where qm_j is the recoded quotient digit for a MM at the j^{th} iteration.

Montgomery recoding scheme transforms the value set of MM into $\{-M, 0, +M, +2M\}$. The proof that the algorithm is still correct after applying Montgomery recoding scheme comes from the fact that $3 \equiv -1 mod 4$ [29].

Table 1: Booth recoding scheme.

Three input bits			Recoded quotient for PP	Recoded PP
$a_{j,1}$	$a_{j,0}$	$a_{j,-1}$	qa_j	PP
0	0	0	0	0
0	0	1	+1	+A
0	1	0	+1	+A
0	1	1	+2	+2A
1	0	0	-2	-2A
1	0	1	-1	-A
1	1	0	-1	-A
1	1	1	0	0

Table 2: Montgomery recoding scheme.

Three input bits			Recoded quotient for MM	Recoded MM
$sp_{0,1}$	$sp_{0,0}$	$m_{0,1}$	qm_j	MM
0	0	0	0	0
0	0	1	0	0
0	1	0	-1	-M
0	1	1	+1	+M
1	0	0	+2	+2M
1	0	1	+2	+2M
1	1	0	+1	+M
1	1	1	-1	-M

3 Hardware Implementation

The author of this paper applied a mapping methodology to the MWR4MM algorithm in prime filed GF(P) in a previous publication [37] and explored four different processor array configurations. Two of these processor arrays are suitable for efficient hardware implementation (Design1 and Design2) and the other two (Design3 and Design4) are complex designs and are not suitable for hardware implementation. The processor array architecture of Design1 was not reported before in the literature, while the processor array of Design2 was reported by L. Tawalbeh [29]. These designs perform multiplication in the prime field GF(p) only. In this paper, we will modify the architecture of Design1 to be used in both fields, prime field GF(P) and binary extension field GF(2^k). Figure 1 shows the proposed unified and scalable processor array architecture that implements the unified MWR4MM algorithm. The processor array consists of $z = \lceil (e+1)/2 \rceil$ Processing Elements (PEs). Input words $B^{(2i)}$, $B^{(2i+1)}$, $M^{(2i)}$, $M^{(2i+1)}$ are allocated to processor PEi and Input a_j is allocated to processor PE0. qa_j, qm_j are generated inside PE0 and pipelined to the next PEs with higher indices. The intermediate output words $S_j^{(i)}$ of each PE are pipelined between adjacent PEs. A tristate buffer at the output of each PE ensures that it is the only output fed to the output bus. The $Fsel$ signal is broadcast to each PE. This signal makes the field multiplier operates in either prime field GF(p) or binary extension field GF(2^n). If $Fsel = 1$ it works in GF(p) mode else in GF(2^k) mode

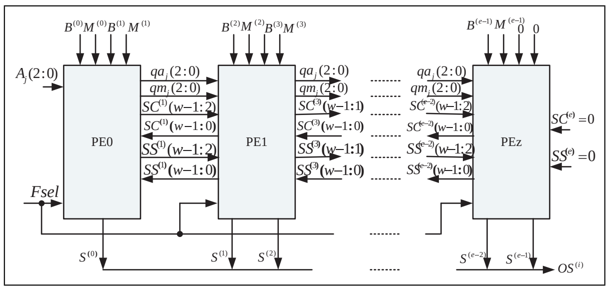

Figure 1: Processor array for MWR4MM algorithm.

3.1 First PE architecture

Figure 2 shows the block diagram of the first PE (PE0). The block diagram of this PE is divided into two portions. The first portion computes only the first 2 Least Significant Bits (LSBs) of the words of $S_{j-1}^{(0)} + qa_j \times B^{(0)}$ and $S_{j-1}^{(1)} + qa_j \times B^{(1)}$ during odd clock cycles (odd_cycl = 1) and even clock cycles (odd_cycl = 0), respectively. The second portion completes the computation of the word bits of $S_{j-1}^{(0)} + qa_j \times B^{(0)}$ and $S_{j-1}^{(1)} + qa_j \times B^{(1)}$ during the even clock cycles (odd_cycl_delay = 1) and the odd clock cycles (odd_cycl_delay = 0), respectively. Also, it performs the addition of full words of $S_j^{(0)} + qm_j \times M^{(0)}$ and $S_j^{(1)} + qm_j \times M^{(1)}$ during even clock cycles (odd_cycl_delay = 1) and odd clock cycles (odd_cycl_delay = 0), respectively. The main functional blocks are the Carry Save Adders (CSAs) which perform steps 4, 6, 8, and 9 in the MWR4MM algorithm, Algorithm 1. So, the intermediate partial product S is represented in a carry save form as two bit vectors SS (sum vector) and SC (carry vector). So, while the leftmost adder (CSA0) works on the two LSBs of a word of $S_{j-1}^{(0)} + qa_j \times B^{(0)}$ or $S_{j-1}^{(1)} + qa_j \times B^{(1)}$, the topmost adder (after the inter-stage register) works on the other bits of $S_{j-1}^{(1)} + qa_j \times B^{(1)}$

or $S^{(0)}_{j-1} + qa_j \times B^{(0)}$, respectively. Therefore, there is one clock cycle difference between the two portions. So, The result word $S^{(0)}_{\lceil n/2 \rceil -1}$ will be available on the output bus through the tristate buffers after latency of $n + 1$ clock cycles (last_cycl_in = 1). In this case, two rows of tristate buffers will be added to the next PEs to prevent any two output words from driving the output bus at the same time, see Figures 5, 6, and 7. The input of the second PE (PE1) will be taken directly from the output of the CSA2 of the first PE (PE0), see Figure 2, to ensure that the input of the first PE (PE0) from the second PE (PE1) arrives at the proper clock cycles.

Figure 2 shows another function blocks. One of these blocks is the **Booth Recoder** block that is used to find the coefficient digit qa_j according to Table 1. The output of this block is the control signals for the **PP-Generator**[29] block, see Figure 3, used to generate the multiples of *B*. There is also the **Montgomery Recoder** Block that is used to find the coefficient digit qm_j according to Table 2. The output of the **Montgomery Recoder** is the control signals for the **MM-Generator** block, see Figure 4, used to generate the multiples of M.

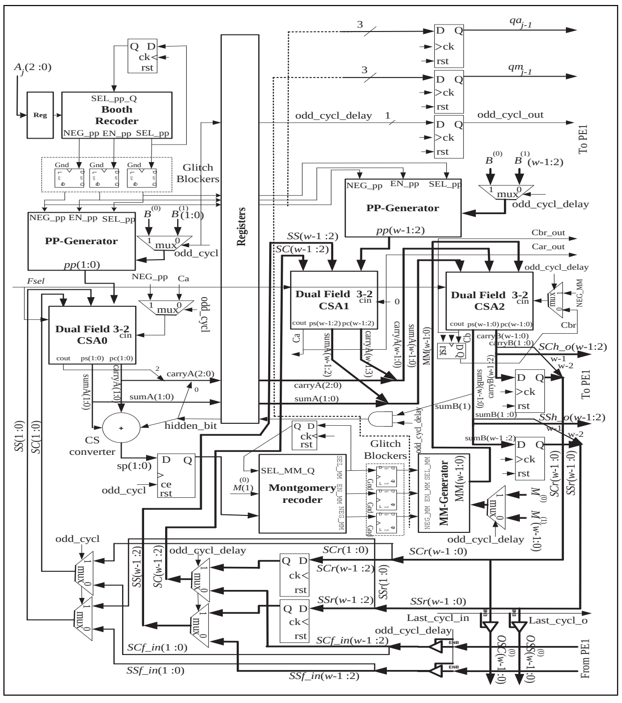

Figure 2: First PE architecture.

3.2 Intermediate PE architecture

Figure 5 shows the block diagram of the intermediate Processing Element (PEi). This PE computes the

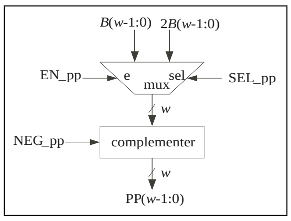

Figure 3: PP-Generator.

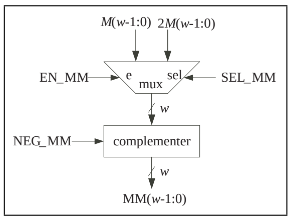

Figure 4: MM-Generator.

words $S^{(i)}_j$, (SSq, SCq), and $S^{(i+1)}_j$, (SSr, SCr), $1 \leq i \leq e - 3$ of the partial product S. The result words $S^{(i)}_{\lceil n/2 \rceil -1}$, $(OSS^{(i)}, OSC^{(i)})$, and $S^{(i+1)}_{\lceil n/2 \rceil -1}$, $(OSS^{(i+1)}, OSC^{(i+1)})$, will be available on the output bus through tristate buffers after latencies $(n+i+1)$ and $(n+i+2)$ clock cycles, respectively.

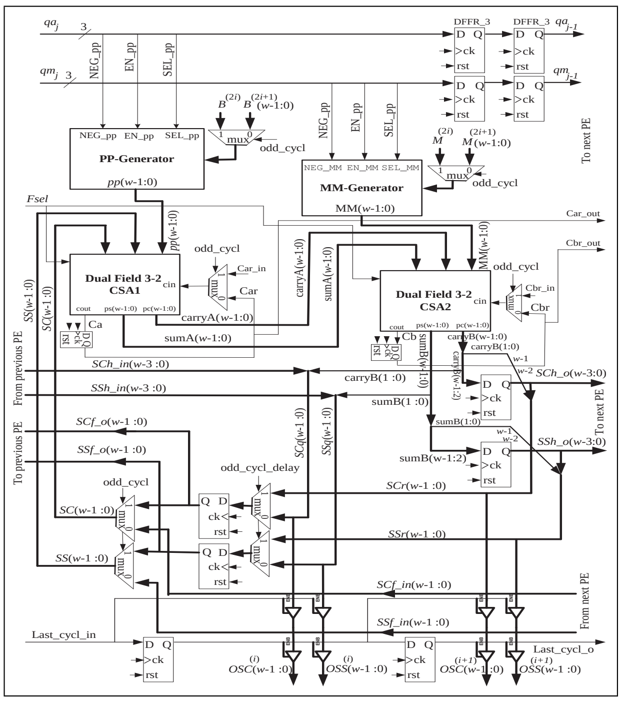

Figure 5: Intermediate PE architecture.

3.3 Last PE architecture when $(e+1)$ even

Figure 6 shows the block diagram of last PE (PEz). It is the same as the intermediate PE (PEi) except that we added two bit muxs at the output of CSA2 to take sign extension into consideration during the last cycle of computation. The result words $S^{(e-2)}_{\lceil n/2\rceil-1}$, $(OSS^{(e-2)}, OSC^{(e-2)})$, and $S^{(e-1)}_{\lceil n/2\rceil-1}$, $(OSS^{(e-1)}, OSC^{(e-1)})$, will be available on the output bus through tristate buffers after latencies $(n+e-1)$ and $(n+e)$ clock cycles, respectively.

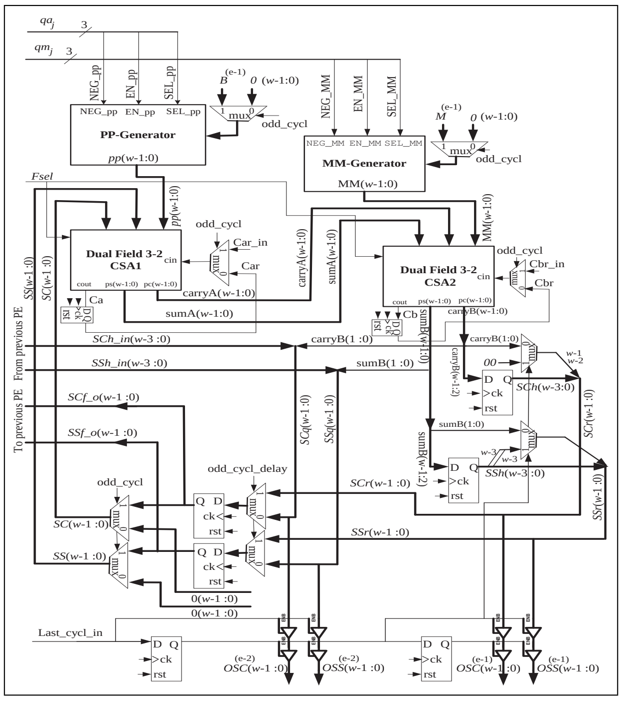

Figure 6: Last PE architecture when $(e+1)$ even.

3.4 Last PE architecture when $(e+1)$ odd

In this case the previous PE will operate only on the input words $B^{(e)} = 0$, $M^{(e)} = 0$, and $S^{(e)} = 0$. Since all of these inputs have a value of zero then the previous PE block diagram can be reduced to the PE block diagram shown in Figure 7. This PE is only used to propagate the last carries of Ca , Cb and to take sign extension into consideration during the last cycle of computation. The last result word $S^{(e-1)}_{\lceil n/2\rceil-1}$,$(OSS^{(e-1)}, OSC^{(e-1)})$, will be available on the output bus through tristate buffers after latency $(n+e)$ clock cycles.

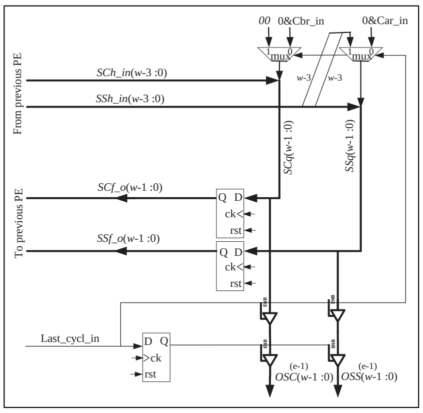

Figure 7: Last PE architecture when $(e+1)$ odd.

4 low power techniques

In this section we further improve the hardware to dissipate less power. The two circuit modules in charge of the **Booth** and **Montgomery recordings** comprise only combinational logic circuits according to Tables 1, 2. The outputs of these modules have unbalanced path delays and consequently introduce glitches which cause worthless dynamic power dissipation. Furthermore, the fan-outs of the glitchy signals are so big and this leads to the amount of dissipated power is significant.

To reduce the glitching power dissipation, we put in some latches and force the outputs to pass through latches. If all flip-flops and registers capture their inputs at the clock's rising edge, then the latches are transparent when the clock is in a low state. If the outputs of the two recoding modules can reach their stable values before the clock's falling edge, none of the glitches can propagate to the fan-out modules driven by the outputs. We name these latches "**glitch blockers**" [16], see Figure 2. The glitch blockers are also very effective for reducing the glitches appearing in the CSA since they synchronize the arrival of PP and MM at the CSA's inputs.

PP generator makes a PP by modifying a word of B according to the **Booth recoder**'s outputs, SEL_PP and EN_PP. Also, MM generator makes a MM by modifying a word of M according to the **Montgomery recoder**'s outputs, SEL_MM and EN_MM. When PP is zeroed by EN_PP, PP outputted from the PP generator does not depend on SEL_PP. Thus, keeping SEL_PP frozen at that time is effective for reducing power dissipation. Same reasoning also applies to SEL_MM. We place two 1-bit flip-flops [16], see Figure 2, and construct feedback loops for SEL_PP and SEL_MM to implement this idea.

The power consumption was reduced further by

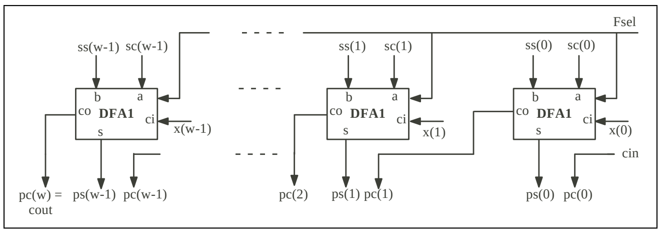

Figure 8: Conventional dual field 3-to-2 CSA.

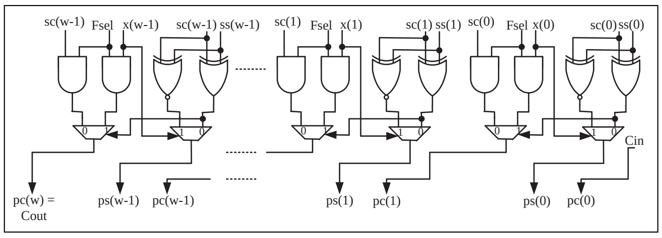
sc(w-1) Fsel x(w-1) sc(w-1) ss(w-1)
sc(1) Fsel x(1) sc(1) ss(1)
sc(0) Fsel x(0) sc(0) ss(0)
Cin
pc(w) = Cout
ps(w-1) pc(w-1)
ps(1) pc(1)
ps(0) pc(0)

Figure 10: Modified dual field 3-to-2 CSA.

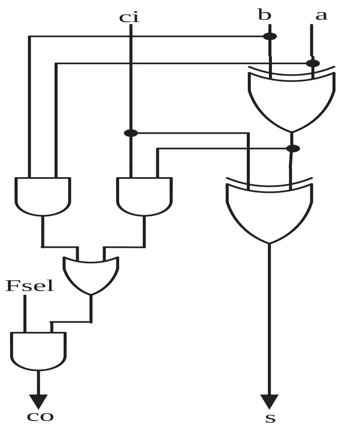

Figure 9: Dual Field adder.

modifying the structure of the conventional dual field 3-to-2 CSA shown in Figure 8. As shown in this figure, each cell of the conventional dual field 3-to-2 CSA consists of a dual field adder (DFA). The DFA is a conventional full adder with extra field select ($Fsel$) signal, as shown in Figure 9. From Figure 8, the sum (ps) and carry (sc) outputs of the DFA can be written as:

$$ps = \overline{x}.(ss \oplus sc) + x.\overline{(ss \oplus sc)} \tag{1}$$

$$\begin{aligned} pc &= x.(ss \oplus sc) + sc.ss \\ &= [x.(ss \oplus sc) + sc.\overline{(ss \oplus sc)}].Fsel \\ &= (x.Fsel).(ss \oplus sc) + (sc.Fsel).\overline{(ss \oplus sc)} \end{aligned} \tag{2}$$

where x represents the bit value of partial products PP or MM. sc and ss are bit values of the partial product vectors SC and SS, respectively.

Figure 10 shows the modified dual field 3-to-2 CSA. We notice from this figure that the internal logic structure is based on multiplexing of the Boolean functions XOR/XNOR and AND/AND, to obtain balanced delays in sum and carry outputs that reduce the chance of glitches occurrence. We did not realize the XOR/XNOR functions by synthesizing the XOR function and generating the XNOR function through an inverter. This type of design has the disadvantage of delaying the XNOR output, giving rise to skewed signal arrival time to the successive modules. This will increase the chance of producing spurious switching and glitches in these modules.

5 ASIC Implementation Comparison

We described the explored architectures and the existing unified/scalable architectures of Son [16], Savas [23], Harris [24], Todorov [28], Huang [35], Sudhakar [38], Ibrahim [39], Wang [40], Amberg [41], Ye [42], Sassaw [43] and Ibrahim [44] in VHDL at the register transfer level and synthesized them to the gate level for n = 1024 and word size w = 32 using a 0.18μm, 1.8V, standard-cell CMOS technology (In [38], the architecture is N bit-sliced architecture, where $N = zw$). We used the Synopsys synthesis tools package version 2005.09-SP2 for the logic synthesis and power analysis. All the synthesis results are obtained under typical operating conditions (1.8V, 25°C). Simulations were performed with Mentor Graphics ModelSim SE 6.0a. Table 3 compares the ASIC implementation of the different multipliers architectures. In this table, the column entitled "Area" represents the area of the multipliers as the number of gate equivalents and column entitled "Max. Multiplication delay" represents the maximum amount of time required by the multiplier to complete a single operation (in some designs multiplication delay depends on the operating fields GF(p) or GF(2^n)), while "Power" represents the estimated power consumed by the multiplier. The $Area \times Time$ metric and $throughput$ rate were calculated using the synthesis results in order to measure the degree of optimization achieved in each multiplier. From Table 3 we notice that, The proposed design has smaller $Area \times Time$ values compared to all designs by ratios ranging from 13.1% to 77.2%. Also, it has higher $throughput$ over them by ratios ranging from 2.6% to 82.9%. In addition, it achieves a decrease in power consumption compared to these designs by ratios ranging from 25.3% to 70.4%.

Table 3: ASIC Implementation Comparison of Different unified/scalable field Multipliers for $n = 1024$ and $w = 32$.

Field multiplier	Radix	Unified/ Scalable	Area (A) [kgates]	Max. mult. delay (T) [μs]	Area(A) $\times$ Times(T) [kgates.μs]	Throughput (n_{bits}/T) [Mbps]	Power [mW]
Savas [23]	2 for GF(p), 4 for GF(2^n)	yes/yes	22.822	28.220	644.044	36.29	87.50
Harris [24]	2	yes/yes	23.324	16.300	380.185	62.82	91.20
Sudhakar [38]	2	yes/yes	23.221	17.180	398.945	57.53	90.57
Ibrahim [39]	2	yes/yes	19.678	17.070	335.92	59.99	88.41
Son [16]	4	no/yes	35.212	10.122	365.416	101.17	53.23
Huang [35]	4	no/yes	33.231	8.123	269.935	126.06	101.22
Wang [40]	4	no/yes	35.212	4.952	174.370	206.79	99.44
Ye [42]	4	no/yes	34.231	4.942	169.170	207.20	77.77
Todorov [28]	8	no/yes	42.626	5.720	243.821	179.02	107.99
Amberg [41]	8	no/yes	38.412	4.960	190.524	206.54	106.88
Sassaw [43]	8	no/yes	43.423	6.460	280.513	158.51	109.22
Ibrahim [44]	8	yes/yes	39.279	5.450	214.071	187.89	43.23
proposed Design	4	yes/yes	30.528	4.812	146.901	212.80	32.31

6 conclusion

This paper presented a new low space complexity, low-power and high-speed processor array architecture for the unified and scalable MWR4MM algorithm. In this architecture, the multiplicand and the modulus words are allocated to each processing element rather than pipelined between the processing elements as in the previous architectures extracted by L. Tawalbeh, and the multiplier bits are fed serially to the first processing element of the processor array every odd clock cycle. We applied several techniques to this architecture to decrease the power consumption. One of these techniques is conventional and uses glitch blockers at the outputs of some circuit modules to reduce the spurious transitions and the expected switching activities of high fan-out signals. Another new technique is to replace the dual field conventional 3-to-2 CSA by modified low power dual field 3-to-2 CSA that has internal logic structure, based on the multiplexing of the Boolean functions XOR/XNOR and AND/AND, to obtain balanced delays in sum and carry outputs that reduce the chance of glitches occurrence. An ASIC Implementation of the proposed architecture and all of the efficient existing ones shows that it can perform 1024-bit modular multiplication (for word size $w = 32$) in about 4.81 μs. Also, the results show that it has smaller $Area \times Time$ values compared to all compared designs by ratios ranging from 13.1% to 77.2% which makes it suitable for implementations where both area and performance are of concern. Also, it has higher $throughput$ over them by ratios ranging from 2.6% to 82.9%. In addition, it achieves a decrease in power consumption compared to these designs by ratios ranging from 25.3% to 70.4%.

Acknowledgements: This research was supported by Salman Bin AbdulAziz University, Alkharj, KSA and Electronics Research Institute, Cairo, Egypt. Also. It was supported by University of Victoria, Victoria, BC, Canada.

References:

[1] W. Diffie and M. Hellman, New directions in cryptography, *IEEE Transactions on Information Theory*. 22, 1976, pp. 644-654.

[2] National Institute for Standards and Technology, Digital Signature Standard (DSS), *FIPS PUB 186-2*,January 2000.

[3] J. Quisquater and C. Couvreur, Fast decipherment algorithm for RSA public-key cryptosystem, *Electronics Letters*. 18, 1982, pp.905-907.

[4] N. Koblitz, Elliptic curve cryptosystems, *Mathematics of Computation*. 48, 1987, pp. 203–209.

[5] A. Menezes, Elliptic Curve Public Key Cryptosystems, *Kluwer Academic Publishers,Boston*. MA,1993.

[6] P. Montgomery, Modular multiplication without trial division, *Mathematics of Computation*. 44, 1985, pp. 519–521.

[7] T. Yanik, E. Savas, and Ç. Koç, Incomplete reduction in modular arithmetic, *Mathematics of Computation*. 149, 2002, pp. 46–54 .

[8] C. Koç, T. Acar, Montgomery multiplication in GF(2k), *Designs, Codes and Cryptography*. 14, 1998, pp. 57–69.

[9] F. Gang, Design of modular multiplier based on improved Montgomery algorithm and systolic array, *in Proc. 1st Int. Multi-Symp. Comput. Comput. Sci*. 2006, pp. 356-359.

[10] S. Ghoreishi, M. Pourmina, H. Bozorgi and M. Dousti, High speed RSA implementation based on modified Booths technique and Montgomerys multiplication for FPGA platform, *in Proc. 2nd Int. Conf. Adv. Circuits, Electron. Micro-Electron*. 2009, pp. 86-93.

[11] G. Sassaw, C. Jimenez and M. Valencia, High radix implementation of Montgomery multipliers with CSA, *in Proc. Int. Conf. Microelectron*. 2010, pp. 315-318.

[12] J. Neto, A. Tenca, W. Ruggiero, A parallel k-partition method to perform Montgomery multiplication, *in Proc. IEEE Int. Conf. Appl.-Specif. Syst., Arch. Process*. 2011, pp. 251-254.

[13] A. Cilardo, A. Mazzeo, L. Romano and G. Saggese, Exploring the design-space for FPGA-based implementation of RSA, *Microprocess. Microsyst*. 28, 2004, PP. 183-190.

[14] D. Bayhan, S. Ors and G. Saldamli, Analyzing and comparing the Montgomery multiplication algorithms for their power consumption, *in Proc. Int. Conf. Comput. Eng. Syst*. 2010, pp. 257–261.

[15] X. Wang, P. Noel and T. Kwasniewski, Low power design techniques for a Montgomery modular multiplier, *in Proc. Int. Symp. Intell. Signal Process. Commun. Syst*. 2005, pp. 449-452.

[16] H. Son, S. Oh, Design and implementation of scalable low-power montgomery multiplier. *International Conference on Computer Design, ICCD 2004*. 2004, pp. 524–531.

[17] S. Kuang, J. Wang, K. Chang and H. Hsu, Energy-Efficient High-Throughput Montgomery modular multipliers for RSA cryptosystems, *IEEE Trans. on VLSI systems*. 11, 2013, pp. 1999–2009.

[18] F. Carbognani, F. Buergin, N. Felber and H. Kaeslin, Transmission Gates Combined With Level-Restoring CMOS Gates Reduce Glitches in Low-Power Low-Frequency Multipliers, *IEEE Trans. on VLSI Systems*. 16, 2008, pp. 830–836.

[19] K. Chirca, M. Schulte, J. Glossner, H. Wang, X. Mamidi, P. Balzola, and S. Vassiliadis, A static low-power, high-performance 32-bit carry skip adder, *Euromicro Symposium on Digital System Design*. 2004, pp. 615–619.

[20] A. Tenca and Ç. Koç, A scalable architecture for modular multiplication based on montgomery's algorithm, *IEEE Trans. On Computers*. 52, 2003, pp. 1215–1221.

[21] A. Tenca, E. Savas and Ç. Koç, A design framework for scalable and unified architectures that perform multiplication in gf(p) and gf(2^m). *International Journal of Computer Research*. 13, 2004, pp. 68–83.

[22] T. Todorov, A. Tenca, and Ç. Koç, High-radix design of a scalable modular multiplier, *in Ç. Ko, D. Naccache, and C. Paar, editors, Cryptographic Hardware and Embedded Systems, Lecture Notes in Computer Science No. 2162, Springer Verlag, Berlin, Germany*. 2001, pp. 189–205.

[23] E. Savas, A. Tenca, M. Ciftcibasi and Ç. Koç, Multiplier architectures for gf(p) and gf(2^n), *IEE Proc. on Computers and Digital Techniques*. 151, 2004, pp. 147–160.

[24] D. Harris, R. Krishnamurthy, M. Andersm, S. Mathew and S. Hsu, An improved unified scalable radix-2 Montgomery multiplier, *IEEE (ARITH-17)*. 2005, pp. 172–178.

[25] J. Groschdl, A bit-serial unified multiplier architecture for finite fields GF(p) and GF(2^m), *Proc. CHES 2001*. 2001, pp. 202–218.

[26] E. Savas, A. Tenca and Ç. Koç , A scalable and unified multiplier architecture for finite fields GF(p) and GF(2^m), *Proceedings of CHES 2000*. 2000, pp. 281–296.

[27] A. Ibrahim, F. Gebali, H. Elsimary and A. Nassar, New Processor Array Architecture for Scalable Radix 8 Montgomery Modular Multiplication, *in proc. Canadian Conference on Electrical and Computer Engineering (CCECE)*. 2011, pp. 389–394.

[28] G. Todorov, A. Tenca, Asic design, implementation and analysis of a scalable high-radix montgomery multiplier, Master's thesis, Oregon State University, USA, 2000.

[29] L. Tawalbeh and A. Tenca, Radix-4 asic design of a scalable montgomery modular multiplier using encoding techniques, Master's thesis, Oregon State University, USA, 2002.

[30] C. Walter, Montgomery exponentiation needs no final subtractions, *Electronics Letters*. 35, 1999, pp. 1831–1832.

[31] H. Orup, Simplifying quotient determination in high-radix modular multiplication, *Proc. 12th IEEE Symp. Computer Arithmetic*. 1995, pp. 193–199.

[32] G. Hachez, J. Quisquater, Montgomery exponentiation with no final subtractions, Improved results. ¸C Koç, D. Naccache, and C. Paar, editors, Cryptographic Hardware and Embedded Systems, Lecture Notes in Computer Science No. 1965, Springer, Berlin, Germany. 2000, pp. 293–301.

[33] F. El-Guibaly, A. Tawfik, Mapping 3d iir digital filter onto systolic arrays, *Multidimensional Systems and Signal Processing*. 7, 1996, pp. 7–26.

[34] Gebali, F.: Algorithms and parallel computing, John Wiley. New York, 2011.

[35] M. Huang, K. Gaj, T. El-Ghazawi, New Hardware Architectures for Montgomery Modular Multiplication Algorithm, *IEEE Trans. on Computers*. 7, 2011, pp. 923–936.

[36] A. Ibrahim, F Gebali, H. El-Simary and A. Nassar, Design and VLSI Implementation of Fast Modular Multiplier Architectures for Cryptosystems, Phd's Dissertation, Department of Electronics and Electrical Communication, Cairo University, 2010.

[37] A. Ibrahim, F Gebali, H. El-Simary and A. Nassar, Processor Array Architectures for Scalable Radix 4 Montgomery Modular Multiplication Algorithm, *IEEE Trans. on Parallel & Distributed System*. 22, 2011, pp. 1142–1149.

[38] M. Sudhakar, R, Kamala and M. Srinivas, New and Improved Architectures for Montgomery Modular Multiplication, *Mobile Netw Appl*. 12, 2007, pp. 750–755.

[39] A. Ibrahim, F Gebali and H. El-Simary, New and improved word-based unified and scalable architecture for radix 2 Montgomery modular multiplication algorithm, *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. 2013, pp. 153–158.

[40] H. Wang, W. Lin, J. Ye and M. Shieh, Fast scalable radix-4 Montgomery modular multiplier, *2012 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2012, pp. 3049–3052.

[41] P. Amberg, N. Pinckney and M. Harris, Parallel High-Radix Montgomery Multipliers, *42nd Asilomar Conference on Signals, Systems and Computers*. 2008, pp. 772–776.

[42] J. Ye, T. Hung and M. Shieh, Energy-efficient Architecture for Word-based Montgomery Modular Multiplication Algorithm, *2013 International Symposium on VLSI Design, Automation, and Test (VLSI-DAT)*. 2013, pp. 1–4.

[43] G. Sassaw, J. Carlos and M. Valencia, High Radix Implementation of Montgomery Multipliers with CSA, *22nd International Conference on Microelectronics (ICM 2010)*. 2010, pp. 315–318.

[44] A. Ibrahim, F Gebali and H. El-Simary, Low-Power, High-Speed Unified and Scalable Word-Based Radix 8 Architecture for Montgomery Modular Multiplication in GF(P) and GF(2^n), *2013 Iaccepted for inclusion in future issue of Springer - Arabian Journal of science and Engineering*. May. 2014.