Speed and Area Optimized Parallel Higher-Radix Modular Multipliers

Khalid Javeed, Xiaojun Wang

***Abstract*—Modular multiplication is the fundamental and compute-intense operation in many Public-Key crypto-systems. This paper presents two modular multipliers with their efficient architectures based on Booth encoding, higher-radix, and Montgomery powering ladder approaches. Montgomery powering ladder technique enables concurrent execution of main operations in the proposed designs, while higher-radix techniques have been adopted to reduce an iteration count which formally dictates a cycle count. It is also shown that by an adopting Booth encoding logic in the designs helps to reduce their area cost with a slight degradation in the maximum achievable frequencies. The proposed designs are implemented in Verilog HDL and synthesized targeting virtex-6 FPGA platform using Xilinx ISE 14.2 Design suite. The radix-4 multiplier computes a 256-bit modular multiplication in 0.93 μs, occupies 1.6K slices, at 137.87 MHz in a cycle count of $\lceil n/2 \rceil + 2$, whereas the radix-8 multiplier completes the operation in 0.69 μs, occupies 3.6K slices, achieves 123.43 MHz frequency in a cycle count of $\lceil n/3 \rceil + 4$. The implementation results reveals that the proposed designs consumes 18% lower FPGA slices without any significant performance degradation as compared to their best contemporary designs.**

***Index Terms*—Finite field, elliptic curve cryptography (ECC), interleaved multiplication, public key cryptography (PKC).**

I. Introuction

Crypto-systems based on public-key cryptography (PKC) [1], [2], [3], [4] are structured using finite field arithmetic primitives such as modular addition, subtraction, multiplication, and inversion. Among these primitives, modular multiplier is the one that is computationally intensive and therefore it must be carefully optimized in order to boost a performance of the associated system. Dedicated hardware architectures have been the most optimum choice for high-performance systems. Field Programmable Gate Array (FPGA) becomes a very popular implementation platform to accelerate many tedious operations because of its low cost, reconfigurability, short design cycle, and many others factors.

Several independent hardware architectures have been proposed to speed-up a modular multiplication primitive in order to reduce the computational complexities of an overall security infrastructure in many communication networks. Modular multiplication of operands x, y over a modulus M (denoted as $z = x \times y$ modulo M) is a two step process: integer multiplication and reduction modulo M. The reduction operation typically requires a division operation which is a very compute-intense operation, therefore many strategies have been proposed to lower the computational intensity of the reduction step. Generally these can be lined-up in three main categories: designs over standard primes [5], designs based on Montgomery multiplication method [6] and designs over interleaved multiplication method [7].

K. Javeed is with the School of Electronic Engineering, Dublin City University, Dublin, Ireland (email: khalid.javeed2@mail.dcu.ie)

X. Wang is with the School of Electronics Engineering, Dublin City University, Dublin, Ireland (email: xiaojun.wang@dcu.ie).

II. Background and Related Work

In order to lower a computational intensity of the reduction step the National Institute of Standard and Technology (NIST) has recommended five specialized primes M of size $(M_{192}, M_{224}, M_{256}, M_{384}, M_{512})$. These primes have a special structure (very close to the power of 2) i.e., $2^a \pm 2^b \pm 1$, and are called pseudo-Mersenne primes. Typically a modular multiplication operation when computed over this prime form results in high performance and lower computational cost. However, as a modulus value is pre-defined, which results in a very-rigid structure not able to opt for any other prime value, hence lacks flexibility. The designs reported in [8], [9] exploited the special structure of NIST recommended primes. The architecture in [9] is developed to support NIST primes M_{224} and M_{256} while [8] supports all of five NIST primes. It occupies 8340 slices and 259 dedicated DSPs blocks on Virtex-6 FPGA platform and performs the respective operation between 80-200 ns. These implementations have limited flexibility which may not be suited to many applications.

Montgomery multiplication method converts the required division operation into cheap shift and addition operations. However, to take leverage of the Montgomery method one needs to transform operands and a result from normal to Montgomery representation and vice versa before and after the original operation. The method is suitable where an overall back and forth conversion overhead is negligible as compared to the main operation cost e.g., in exponentiation algorithms. Montgomery multiplication algorithm based designs are reported in [10], [11]. Among these [11] is a based on radix-4 and [10] incorporated radix-16 techniques. Koç et al. in [12] discussed several possible strategies on basis of their performance and implementation cost.

Interleaved multiplication method is proposed by Blakley [7] in 1983. The method is based on iterative addition and reduction of partial products. Partial products accumulation and intermediate results reduction are integrated in a way to eliminate the final division. The idea is to reduce intermediate results below a modulus value in each iteration so that the final division can be omitted. The algorithm starts traversing a multiplier from most-significant-bit (MSB) to least-significant-bit (LSB). Several modifications and hardware architectures

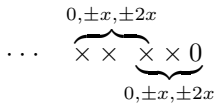

Fig. 1. BE radix-4 scheme

have been reported [13], [14], [15], [16], [17], [18]. In [17] a faster interleaved modular multiplier based on Montgomery and Barrett reduction techniques is reported. Its 130-nm ASIC implementation runs at a maximum frequency of 320 MHz and computes one 256-bit modular multiplication in 0.05 us. Ghosh et al in [16] reported a radix-2 parallel interleaved modular multiplier. Its Virtex-II Pro FPGA implementation consumes 3475 slices with a latency of 3.2 us and takes n clock cycles to a perform n-bit modular multiplication.

A. Contribution

The main objective of this work is to design efficient modular multipliers that are not only optimized for speed and area, but also have a flexibility to adopt any value for a prime M. The main contributions of this work are summarized below.

- A Booth encoded Montgomery powering ladder based radix-4 modular multiplier is presented. Radix-4 method is adopted to reduce iteration count (partial products), Montgomery powering ladder based strategy is incorporated to perform main operations concurrently, and Booth recoding is used to optimize the design space complexity.
- Then, the same idea is extended to a radix-8 Booth encoded Montgomery powering ladder based modular multiplier.
- The radix-4 version of the multipliers performs a n-bit modular multiplication in $\lceil n/2 \rceil + 2$ clock cycles, whereas the radix-8 version executes the operation in $\lceil n/3 \rceil + 4$ for the same bit length.
- In both of the proposed modular multipliers addition and subtraction is performed by the available fast carry chains (FCC) on FPGA platform.

The rest of this paper is structured as follows: Section III presents our proposed radix-4 and radix-8 Booth encoded Montgomery Powering ladder based modular multipliers. Hardware architectures of the multipliers are presented in section IV. Section V presents implementation results and performance evaluation to the other related designs, while the paper is concluded in section VI.

III. Proposed Modular Multipliers

Typical Higher-radix based multipliers produce faster results because of their lower iteration count as compared to their bit-level implementation. However, these techniques deteriorate the critical path, which limit their maximum achievable frequencies. Several techniques have been proposed to reduce inner complexities that shortened the critical path of higher-radix multipliers. We introduced modifications to the Interleaved Multiplication (IM) algorithm.

Input: $x, y, M : y := [1, y_{n-2}, ..., y_0]$
Output: $x^y \bmod M$
1 $r_1 := x$ and $r_2 := x^2$
2 **for** $i := n-2$ *downto* 0 **do**
3 **if** $(y_i := 1)$ **then**
4 $r_1 := r_1.r_2$ and $r_2 := (r_2)^2$
5 **else**
6 $r_2 := r_1.r_2$ and $r_1 := (r_2)^2$
7 **return** r_1

Algorithm 1: The Montgomery Powering Ladder

Input: $M, x, y : 0 \leq x, y \leq M$
Output: $z = x \times y \bmod M$
1 $z := 0, R_1 := x$, $R_2 := 2x \bmod M$
2 **for** $i := 0$ *downto* $n-1; i := i+2$ **do**
3 **if** $(y_{(i,i+1,i+2)}) := (\{000\}|\{111\})$ **then**
4 $v^{(i)} := 0$
5 **else if** $(y_{(i,i+1,i+2)}) := (\{001\}|\{010\}\{101\}|\{110\})$ **then**
6 $v^{(i)} := R_1^{(i)}$
7 **else**
8 $v^{(i)} := R_2^{(i)}$
9 $R_1^{(i+2)} := 4R_1^{(i)} \bmod M$
10 $R_2^{(i+2)} := 4R_2^{(i)} \bmod M$
11 **if** ($BE_{ctr} := 1$) **then**
12 $z^{(i+2)} := z^{(i)} - v^{(i)} \bmod M$
13 **else**
14 $z^{(i+2)} := z^{(i)} + v^{(i)} \bmod M$
15 **return** $z^{(i)}$

Algorithm 2: BE radix-4 IMML Modular Multiplication

A. BE Radix-4 IMML Multiplier

A Booth encoded Montgomery powering ladder based radix-4 multiplier (BE radix-4 IMML) is a modification of the interleaved modular multiplication (IM) algorithm which is also known as binary double-and-add algorithm. Montgomery powering ladder (ML) [19] technique was initially proposed to speed-up a square and multiply method of a modular exponentiation. The ML method given in Alg. 1 eliminates conditional branch evaluation and enables parallel execution of a modular multiplication and modular squaring operations as mentioned in steps 4 and 6 of the Alg. 1. Both of these operations are performed at every iteration of the algorithm irrespective of an exponent bit y_i. Serial Booth encoded radix-4 and radix-8 IM multipliers are proposed in [13] and radix-4 and ML based parallel multiplier is reported in [18]. Booth encoding technique was first suggested by Booth [20] and later on modified by Macsorley [21]. The modified radix-4 Booth encoding technique is shown in Fig. 1, where it appends zero to the right of least significant bit (LSB) of a multiplier y. In the case of radix-4, it is operated on triplets with one bit overlap starting from $y-1$ and proceeds towards the most significant bit (MSB), Possible partial products in case of BE radix-4 due to scanning of a triplet of y are $\{0, \pm 1, \pm 2\}x$ for a multiplicand x. Our proposed modification to the IM algorithm on basis of BE, radix-4, and ML techniques is given in Alg.

TABLE I
BOOTH ENCODED RADIX-4 MONTGOMERY POWERING LADDERING BASED PARALLEL MULTIPLIER

	$z := 0$	$R_1 := x$	$R_2 := 2x \bmod M$
$i = 0$	$z^{(i+2)} := z^{(i)} \pm v^{(i)} \bmod M$	$R_1^{(i+2)} := 4R_1^{(i)} \bmod M$	$R_2^{(i+2)} := 4R_2^{(i)} \bmod M$
$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$
$i = \lceil n/2 \rceil$	$z^{(i+2)} := z^{(i)} \pm v^{(i)} \bmod M$	$R_1^{(i+2)} := 4R_1^{(i)} \bmod M$	$R_2^{(i+2)} := 4R_2^{(i)} \bmod M$

TABLE II
BOOTH ENCODED RADIX-8 MONTGOMERY POWERING LADDERING BASED PARALLEL MULTIPLIER

	$z := 0$	$R_1 := x$	$R_2 := 2x \bmod M$	$R_3 := 3x \bmod M$	$R_4 := 4x \bmod M$
$i = 0$	$z^{(i+3)} := z^{(i)} \pm v^{(i)} \bmod M$	$R_1^{(i+3)} := 8R_1^{(i)} \bmod M$	$R_2^{(i+3)} := 8R_2^{(i)} \bmod M$	$R_3^{(i+3)} := 8R_3^{(i)} \bmod M$	$R_4^{(i+3)} := 8R_4^{(i)} \bmod M$
$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$
$i = \lceil n/3 \rceil$	$z^{(i+3)} := z^{(i)} \pm v^{(i)} \bmod M$	$R_1^{(i+3)} := 8R_1^{(i)} \bmod M$	$R_2^{(i+3)} := 8R_2^{(i)} \bmod M$	$R_3^{(i+3)} := 8R_3^{(i)} \bmod M$	$R_4^{(i+3)} := 8R_4^{(i)} \bmod M$

Fig. 2. BE radix-8 scheme

2. The modified BE radix-4 IMML algorithm scans triplets of a multiplier y from LSB to MSB and instead of shifting an accumulator contents it shifts partial products in each iteration. The Alg. 2 is comprised of several independent steps such as 9, 10, 12, and 14. In each iteration possible partial products (R_1, R_2) are two-bit left shifted and are either modular added or subtracted from an accumulator (z) contents. Table I depicts an execution flow of the ALg. 2, note that a partial product $2x$ mod M needs to be pre-computed. Due to processing two bits of a multiplier in each iteration, the total number of iterations in the given algorithm is $\lceil n/2 \rceil$, where n is bit length of a modulus M i.e., $n = \log_2 M$.

Input: $M, x, y : 0 \leq x, y \leq M$
Output: $z = x \times y \bmod M$
1 $z := 0, R_1 := x, R_2 := 2x \bmod M$
2 $R_3 := 3x \bmod M$, $R_4 := 4x \bmod M$
3 **for** $i := 0$ *downto* $n-1; i := i+3$ **do**
4 **if** $(y_{(i,i+1,i+2,i+3)}) := (\{0000\}|\{1111\})$ **then**
5 $v^{(i)} := 0$
6 **else if** $(y_{(i,i+1,i+2,i+3)}) := (\{0001\}|\{0010\}\{1101\}|\{1110\})$ **then**
7 $v^{(i)} := R_1^{(i)}$
8 **else if** $(y_{(i,i+1,i+2,i+3)}) := (\{0011\}|\{0100\}\{1011\}|\{1100\})$ **then**
9 $v^{(i)} := R_2^{(i)}$
10 **else if** $(y_{(i,i+1,i+2,i+3)}) := (\{0101\}|\{0110\}\{1001\}|\{1010\})$ **then**
11 $v^{(i)} := R_3^{(i)}$
12 **else**
13 $v^{(i)} := R_4^{(i)}$
14 $R_1^{(i+3)} := 8R_1^{(i)} \bmod M$
15 $R_2^{(i+3)} := 8R_2^{(i)} \bmod M$
16 $R_3^{(i+3)} := 8R_3^{(i)} \bmod M$
17 $R_4^{(i+3)} := 8R_4^{(i)} \bmod M$
18 **if** ($BE_{ctr} := 1$) **then**
19 $z^{(i+3)} := z^{(i)} - v^{(i)} \bmod M$
20 **else**
21 $z^{(i+3)} := z^{(i)} + v^{(i)} \bmod M$
22 **return** $z^{(i)}$

Algorithm 3: BE radix-8 IMML Modular Multiplication

B. BE Radix-8 IMML Multiplier

A modified BE radix-8 IMML algorithm is given in Alg. 3. A BE radix-8 technique is shown in Fig. 2, where it scans a quadruplet of a multiplier y with a single bit overlap between adjacent quadruplets. Possible partial products in this case are $\{0, \pm 1, \pm 2 \pm 3, \pm 4\}x$. The generation of $\pm 3x, \pm 4x$ is a major difference to the BE radix-4 IMML algorithm.

The Alg. 3 is comprised of six main steps i.e., 14, 15, 16, 17, 19, and 20. In the steps 14, 15, 16, and 17 three-bit left-shift modulo M operation is performed on their respective partial products, while in the steps 15 and 17 modular addition or subtraction of a respective partial product is performed from an accumulator z as detailed in Table II. Note that due to the ML method, there is hardly any data dependency among all these operations, therefore these can be executed concurrently. In the case of BE radix-8 IMML method, the iteration count is further reduced to $\lceil n/3 \rceil$, however it requires more design space due to several parallel units to execute the above mentioned steps which would be discussed in the next section.

IV. Hardware Architectures

Hardware architectures of the presented BE radix-4 IMML and BE radix-8 IMML algorithms mentioned in Alg. 2, Alg. 3 are shown in Fig. 3(a) and Fig. 4(a), respectively.

The BE radix-4 IMML architecture consists of three main blocks EU_1, EU_2 and A/S. In addition to these main blocks there is a BE block, three n-bit data registers R_1, R_2, Z and one shift register (SR). Blocks EU_1 and EU_2 are exactly identical, and perform two-bit left shift mod M operation i.e., $4R_i$ mod M. The A/S block is responsible for performing addition/subtraction mod M operation i.e., $x \pm y$ mod M. Internal architectures of the blocks EU_i, A/S and BE are shown in Fig. 3(c), Fig. 3(d) and Fig. 3(e), respectively. Each EU_i block consists of two identical smaller A_1 blocks cascaded in series, where each of the A_1 blocks performs single-bit left shift mod M operation i.e., $2x$ mod M operation. The BE block accepts three respective multiplier bits, y_i, y_{i+1}, y_{i+2} and outputs a single-bit control signal (BE_{ctrl}) for the A/S block. Initially, the SR register is loaded with multiplier y

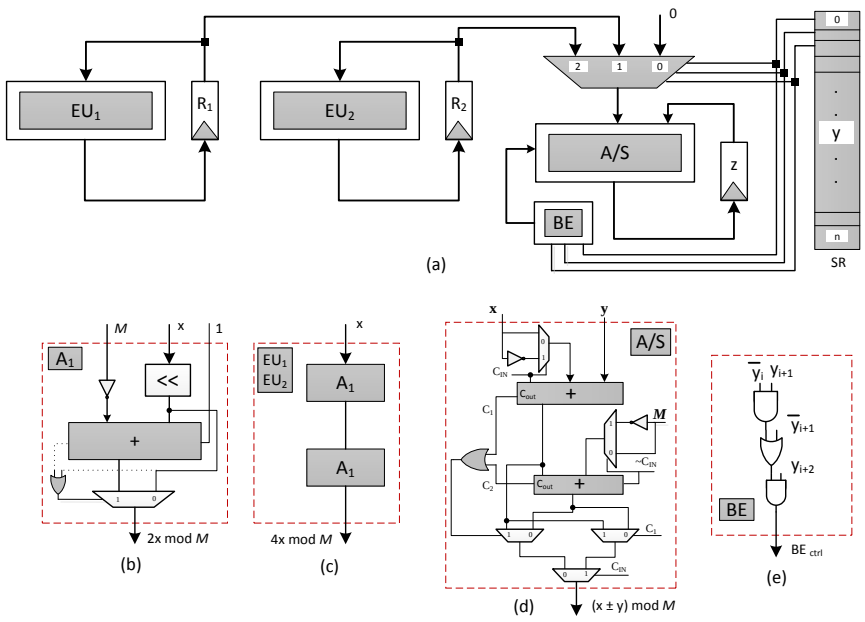

Fig. 3. BE radix-4 IMML multiplier hardware architecture

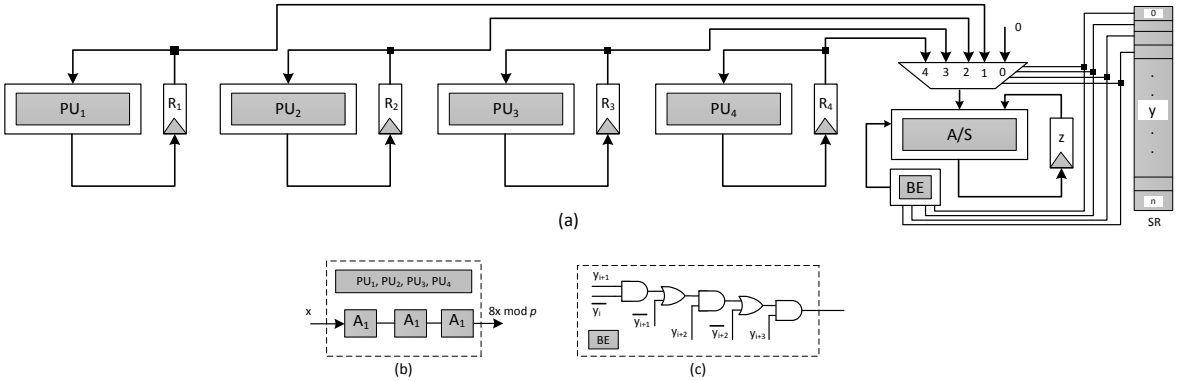

Fig. 4. BE radix-8 IMML multiplier hardware architecture

which performs two-bit right shift of y in each clock cycle. Steps 9, 10 are performed by the blocks EU_1 and EU_2 while steps 12 and 14 are performed by the A/S block. In each iteration of the algorithm all these steps are executed on their respective blocks concurrently. It is worth mentioning that at the start an operand x is loaded to register R_1, however register R_2 needs to be loaded with the $2x$ mod M value, which is done by a pre-computation process not shown in Fig. 3. In the pre-computation process any of the EU_i blocks can be configured to compute the $2x$ mod M value which is then saved in register R_2. This pre-computation process does not incur any additional combinational blocks and it only costs an extra two clock cycles overhead. As the total number of iterations in the BE radix-4 IMML algorithm in Alg. 2 is $\lceil n/2 \rceil$ and the proposed architecture performs each iteration in a single clock cycle, therefore the latency of the proposed design is $\lceil n/2 \rceil + 2$ clock cycles, where an extra two cycles are consumed in the pre-computation process. Note that the critical path of the BE radix-4 IMML is comprised of an A/S block and a multiplexer, overall which is comprised of four multiplexers and two n-bit adders.

Similarly, the BE radix-8 IMML architecture in Fig. 4(a) is comprised of four identical processing units PU_{1-4} and the same A/S block. In addition to these it also contains some data registers R_{1-3} and Z, a BE block and SR. Each PU_i unit performs three-bit left shift mod M operation i.e., $8x$ mod M, and consists of the three A_1 units cascaded in series fashion as shown in Fig. 4(b). The BE block as shown in Fig. 4(c) now operates on four respective multiplier bits i.e., $y_i, y_{i+1}, y_{i+2}, y_{i+3}$ and generates a control signal for the A/S block. Note that for the BE radix-4 IMML we need to have a pre-computed value ($2x$ mod M), however in the BE radix-8 IMML registers R_1, R_2, R_3, R_4 must be pre-loaded with an operand x and $2x$ mod M, $3x$ mod M, and $4x$ mod M, respectively. The values $2x$ mod M and $4x$ mod M are computed by the block PU_1 in a single clock cycle, then the registers R_2 and R_3 are updated with the results. The $3x$ mod M value is calculated in the A/S block by selecting appropriate inputs (R_1, R_3) and the result is stored in the register R_3 in the next clock cycle. The pre-computation process in the BE

TABLE III
MODULAR MULTIPLIERS IMPLEMENTATION COMPARISON IN FPGA

Design	Platform	M size	Area (slices)	LUTs[b]	S.reg[c]	Freq. (MHz)	Time (us)	TP[d] (**Mb/s**)	AT/b[e]
Radix-2 IM [7][♯]	Virtex 6	256	1012	2900	777	125	2.03	126	8.02
Radix-2 PIM [16]	Virtex II pro	256	3475	-	-	80	3.02	84.76	40.99
Radix-2 PIM [16][♯]	Virtex 6	256	1190	3207	1075	174.1	1.48	172.9	6.879
BE Radix-4 IM [13]	Virtex 6	256	1375	4630	-	86.6	1.49	171.8	7.42
BE Radix-8 IM [13]	Virtex 6	256	1739	5657	-	71	1.21	211	8.21
Radix-4 IMML [18]	Virtex 6	256	1985	6300	2187	166	0.78	328	6.04
		224	1745	5367	1883	167.6	0.68	329.4	5.29
		192	1519	4641	1625	168.7	0.57	336.84	4.52
Radix-8 IMML [18][♯]	Virtex 6	256	4428	13880	2756	124.4	0.69	79.2	11.93
		224	4014	12737	2436	127.6	0.59	379.66	10.57
		192	3631	10520	2116	132.6	0.48	400	9.07
BE Radix-4 IMML	Virtex 6	256	1631	4935	1382	137.87	0.93	275.26	5.9
		224	1496	4427	1221	142.7	0.79	283.54	5.27
		192	1395	3846	1057	145.7	0.66	290.9	4.8
		160	1042	3184	910	147	0.54	296.2	3.51
BE Radix-8 IMML	Virtex 6	256	3622	10284	1952	123.43	0.696	367.81	9.84
		224	3326	9115	1727	125.7	0.61	367.2	9.05
		192	2745	7728	1502	127	0.50	384	7.14
		160	2306	6334	1276	128.4	0.42	609.5	6.05

[♯]Our implementation, [b]Look-up-tables, [c]Slice registers, [d]Throughput, [e]Slice area times product per bit

radix-8 IMML multiplier is performed in 4 clock cycles. As the total number of iterations in the Alg. 3 is $\lceil n/3 \rceil$, therefore the proposed architecture computes n-bit modular multiplication operation in $\lceil n/2 \rceil + 4$ clock cycles.

V. IMPLEMENTATION AND RESULTS

The proposed BE radix-4 and BE radix-8 IMML multipliers have been coded in Verilog HDL and Xilinx ISE 14.2 Design Suite is used for synthesis, mapping, placement and routing purposes targeting Virtex-6 FPGA device XCV6LX550. Xilinx ISIM simulator has been used for behavioral simulation of the designs. Addition and subtraction is performed through in-built fast carry chains of the device.

Table IV lists PAR (post placement and routing) results of the designs against four different field sizes (160, 192, 224, 256), where the BE radix-4 IMML multiplier computes a 256-bit modular multiplication operation in 0.93 us, consuming 1985 slices (6300 LUTs) whilst running at 137.87 MHz maximum frequency. The BE radix-8 IMML multiplier completes the same bit length operation in 0.696 us, occupies 3622 slices (10284 LUTs). A throughput (TP) of our BE radix-4 IMML is 275.26 $\times 10^6$ bits per second (bps), whereas the BE radix-8 IMML has a TP of 367.8 $\times 10^6$ bps. Slice area $\times$ time per bit (AT/b) values of the BE radix-4 and BE radix-8 IMML designs are 5.9, 9.84 respectively.

The same Table also demonstrates performance metrics of several other related designs based on the IM algorithm. It is important to mention that to perform a fair and conclusive performance comparison, Table IV only demonstrates designs based on IM algorithm. Our radix-2 implementation of the IM algorithm [7] on the same platform takes 2.03 us to perform a 256-bit modular multiplication operation and occupies 1012 slices (2900 LUTs) whilst achieving a 125 MHz maximum frequency. It is 54% and 66% slower than the proposed BE radix-4 IMML and BE radix-8 IMML multipliers, respectively. The design in [16] is also based on radix-2 implementation of the algorithm on Virtex-II pro, where it computes the operation in 3.02 us and occupies 2475 slices. A direct comparison among our designs and [16] is not conclusive, therefore we implement the design on Virtex-6, where it takes 1.48 us, occupies 1190 slices and achieves 171.8 MHz maximum frequency. It is 1.59 and 2.12 times slower as compared to the proposed BE radix-4 and BE radix-8 IMML modular multipliers respectively.

The BE radix-4 and BE radix-8 based designs reported in [13] are not using ML method, hence, they execute the main operations of IM algorithm in a serial fashion. Though these designs consumes the same amount of clock cycles to perform a modular multiplication operation, however due to serial nature of the designs they are not able to achieve lower critical path delays as compared to the proposed designs. Our BE radix-4 IMML design is 1.6 times faster than the BE radix-4 IM design, while the BE radix-8 IMML design is 1.74 times faster than the BE radix-8 IM design on the same platform.

In [18] radix-4 IMML design is proposed and its implementation on Virtex-6 platform takes 0.77 us, occupies 1985 slices, and runs at a maximum frequency of 166 MHz. It is 0.16 times faster than our BE radix-4 IMML design, however it consumes 0.18 times more FPGA slices. If the same idea [18] is even extended to radix-8 then it requires 4428 slices, which is 1.2 times more than our BE radix-8 IMML design without any significant speed improvement. In Fig. 5 we present comparisons among the above mentioned designs on the basis of throughput versus slice area $\times$ time per bit value (AT/b) for a field size of 256-bit, which depicts that our BE radix-4 IMML design has the lowest area-delay product while the BE radix-8 IMML has the same throughput and much lower area-delay product as compared to the Radix-8 IMML [18][♯]. Therefore, by combining BE and Montgomery powering ladder approaches to include a parallelism in the IM algorithm result in speed and area optimized modular multipliers. Both of the presented architectures are highly flexible to perform modular

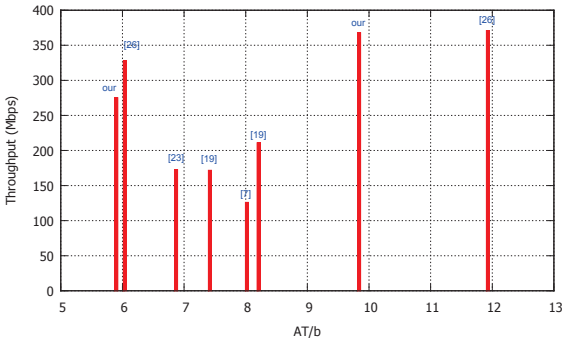

Fig. 5. Throughput vs AT/b for different designs

multiplication for any prime number M, which demonstrates that these designs are very much compatible to build many crypto-processors based on elliptic curve cryptography (ECC) using several different arbitrary curves [22].

VI. CONCLUSION

This paper has introduced Booth encoded radix-4 and radix-8 Montgomery powering ladder based modular multipliers. Higher-radix approaches have been used to decrease iteration count, which formally is the total number of required clock cycles to compute a modular multiplication operation. We noticed that using the Montgomery powering ladder approach reduces inner multiplier complexities and enables parallel execution of the main operations. Furthermore, we explored that incorporating Booth encoding logic helps to reduce design space complexity with a slight overhead in the critical path as compared to designs not using Booth recoding logic. Radix- 4 Booth encoded Montgomery powering ladder version computes a n-bit modular multiplication operation in $\lceil n/2 \rceil + 2$ clock cycles and achieves a maximum frequency of 137.87 MHz, while the radix-8 version takes $\lceil n/3 \rceil + 4$ clock cycles and achieves a maximum frequency of 123.43 MHz for a field size of 256-bit.

VII. ACKNOWLEDGMENT

This work is funded by the Higher Education Authority (HEA) Ireland, under Telecommunication Graduate Initiative (TGI) scheme.

REFERENCES

[1] W. Diffie and M. E. Hellman, "New directions in cryptography," *Information Theory, IEEE Transactions on*, vol. 22, no. 6, pp. 644–654, 1976.

[2] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[3] V. S. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology-CRYPTO85 Proceedings*. Springer, 1986, pp. 417–426.

[4] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and Public-Key cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, 1978.

[5] P. FIPS, "186-2. digital signature standard (dss)," *National Institute of Standards and Technology (NIST)*, 2000.

[6] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.

[7] G. Blakely, "A computer algorithm for calculating the product AB modulo M," *IEEE Transactions on Computers*, vol. 32, no. 5, pp. 497–500, 1983.

[8] H. Alrimeih and D. Rakhmatov, "Pipelined modular multiplier supporting multiple standard prime fields," in *Application-specific Systems, Architectures and Processors (ASAP), 2014 IEEE 25th International Conference on*, June 2014, pp. 48–56.

[9] T. Güneysu and C. Paar, "Ultra high performance ECC over NIST primes on commercial FPGAs," in *Cryptographic Hardware and Embedded Systems–CHES 2008*. Springer, 2008, pp. 62–78.

[10] K. Kelley and D. Harris, "Very high radix scalable Montgomery multipliers," in *System-on-Chip for Real-Time Applications, 2005. Proceedings. Fifth International Workshop on*, July 2005, pp. 400–404.

[11] A. Tenca and L. Tawalbeh, "An efficient and scalable radix-4 modular multiplier design using recoding techniques," in *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*, vol. 2, Nov 2003, pp. 1445–1450 Vol.2.

[12] Ç. K. Koç, T. Acar, and B. S. Kaliski Jr, "Analyzing and comparing montgomery multiplication algorithms," *Micro, IEEE*, vol. 16, no. 3, pp. 26–33, Jun 1996.

[13] K. Javeed and X. Wang, "Radix-4 and radix-8 booth encoded interleaved modular multipliers over general Fp," in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, Sept 2014, pp. 1–6.

[14] D. Narh Amanor, C. Paar, J. Pelzl, V. Bunimov, and M. Schimmler, "Efficient hardware architectures for modular multiplication on FPGAs," in *Field Programmable Logic and Applications, 2005. International Conference on*, Aug 2005, pp. 539–542.

[15] S. Ghosh, D. Mukhopadhyay, and D. Roychowdhury, "Secure dual-core cryptoprocessor for pairings over barreto-naehrig curves on fpga platform," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 3, pp. 434–442, March 2013.

[16] S. Ghosh, D. Mukhopadhyay, and D. Chowdhury, "High speed Fp multipliers and adders on FPGA platform," in *Design and Architectures for Signal and Image Processing (DASIP), 2010 Conference on*, Oct 2010, pp. 21–26.

[17] M. Knezevic, F. Vercauteren, and I. Verbauwhede, "Faster interleaved modular multiplication based on Barrett and Montgomery reduction methods," *Computers, IEEE Transactions on*, vol. 59, no. 12, pp. 1715–1721, Dec 2010.

[18] K. Javeed, X. Wang, and M. Scott, "Serial and parallel interleaved modular multipliers on FPGA platform," in *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*, Sept 2015, p. in press.

[19] P. L. Montgomery, "Speeding the pollard and elliptic curve methods of factorization," *Mathematics of computation*, vol. 48, no. 177, pp. 243–264, 1987.

[20] A. D. Booth, "A signed binary multiplication technique," *The Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, no. 2, pp. 236–240, 1951.

[21] O. L. MacSorley, "High-speed arithmetic in binary computers," *Proceedings of the IRE*, vol. 49, no. 1, pp. 67–91, 1961.

[22] D. J. Bernstein and T. Lange, "Safecurves: choosing safe curves for elliptic-curve cryptography," *http://safecurves. cr. yp. to*, 2014.