Efficient Modulo Multiplier

Rekib Uddin Ahmed, Sheba Diamond Thabah, Mridul Haque, and Prabir Saha

***Abstract*—The paper presents the methodology to compute modulo multiplication with the moduli set 2^n, 2^n-1, 2^n+1. In addition to this, designs of the modulo multipliers, namely 2^n, 2^n-1, and 2^n+1 (with n = 4, 8, and 16), have been proposed which are based on half adders, full adders, 4:3 compressor, 7:3 compressor, and the multi-column compressor namely 5,5:4. The gate level design of 4:3 compressor is carried out by solving the truth table using the K-map reduction. To verify the functionalities we have implemented the proposed modulo multipliers using VHDL coding in Xilinx 14.2 design suite. Simulation using Virtex-6 device has been performed to estimate delay, power consumption, and power-delay product (PDP). Moreover, the modulo multipliers are simulated in Cadence RC compiler using 0.18 μm technology to estimate the area. One of the major contributions to the arts of this work is in the partial product reduction stage which utilizes the multi-column 5,5:4 compressor to reduce power and area. The modulo 2^n-1 multiplier of operand size 4-bit shows an improvement of 66.34% in terms of area over the best-reported paper. On the other hand, the modulo 2^n+1 multiplier of operand size 4-bit shows an improvement of 58.59% terms of in area and the same of operand size 8-bit shows an improvement of 22.72% over the best-reported paper. The proposed algorithms of moduli multiplication are applicable to Booth multiplication of signed numbers.**

***Index Terms*—Algorithm, modulo multiplier, multi-column compressor.**

Original Research Paper
DOI: 10.53314/ELS2327018A

I. Introduction

THE computation of modulo arithmetic is a fascinating problem as it leads to efficient combinational converters, which in turn improves the conversion in binary systems and have versatile applications like residue number system (RNS) [1], cryptography [2], digital signal processors [3], fault-tolerant computer system [4], etc. The computation using moduli set (2^n, 2^n-1, and 2^n+1) is a popular approach because of the existence of an efficient combinational converter from/to binary systems [5]. In addition to this, such moduli set have added benefits that includes efficient low-cost hardware for conversion, reduction and reverse conversion as well [6]. The moduli multipliers gained popularity as they offer efficient circuitry from the perspective of area $\times$time^2 product and efficient converters to and from the binary system. Fast and efficient modulo adders and multipliers are the prerequisites for higher-performance microchip. Numerous researches have been carried out in the literature in designing algorithms and hardware for individual modulo adder and multiplier, which are based on either (2^n-1) and/or (2^n+1) [5], [7]–[14]. It is a crucial arithmetic block required in applications involving pseudorandom number generation and cryptography. It can be observed that in (2^n-1) operator designs, an adder block is used either in parallel prefix adder or carry save adder, which can handle the carry stage at the cost of an extra stage required for the end around carry [15].Further in [7] the authors have shown the computation of the end around carry in the prefix level itself. However, the operation of the modulo (2^n-1) is still confined to a regular structure format despite incorporating the prefix added which results in routing delay penalty [7]. Modulo (2^n+1) multiplier is much more complex and has a higher hardware complexity than its counterpart design, i.e., modulo (2^n-1) [7]. Depending on the type of operands and outputs generated, the modulo multipliers can be divided into three categories: 1) result and both inputs use weighted representation; 2) result and both the inputs use diminished-one representation; 3) result and one of the inputs use weighted representation, while the other input uses diminished-one. Of the three major types of special modulo multipliers, the modulo 2^n+1 is the most critical in terms of hardware complexity because of irregular structure causing inefficiency as compared to other moduli set (2^n-1 and 2^n). This irregularity in the structure of the modulo 2^n+1 is due to the excessive computations introduced by an extra bit [18]. The modulo 2^n+1 multiplication plays an important role in Fermat number transform [10] and RNS. The diminished-one number system proposed by Leibowitz [16] has been utilized for efficient implementation of (2^n+1) design. However, the diminished-one number system adder hardware's complexity significantly increases as it is based on the end around carry, which requires parallel adders

The modulo multipliers require various ROM-based solutions using look-up tables (LUT) [9], [17] which have been proposed and compared in [11]. Sophisticated methods are used to reduce the LUT sizes by combining smaller LUTs with simple arithmetic operations such as additions. However, it has been observed that for word lengths larger than eight bits, these solutions require prohibitively larger ROMs and many clock cycles for computation. With the current advancement in VLSI technology, traditional algorithms and methodologies for

Manuscript received 9 November, 2022. Received in revised form 25 March, 2023. Accepted for publication 19 April, 2023.

This work was supported in part by the National Institute of Technology Meghalaya, TEQIP-III, and in part by Visvesvaraya PhD Scheme, Government of India.

Rekib Uddin Ahmed was with the Department of Electronics and Communication Engineering, National Institute of Technology Meghalaya, Shillong 793003, India. He is currently a Senior Research Scientist with the Department of Electrical Engineering, Indian Institute of Technology Bombay, Mumbai, India. (e-mail: rekib@nitm.ac.in; phone: +919436594902).

Sheba Diamond Thabah was with the Department of Electronics and Communication Engineering, National Institute of Technology Meghalaya, Shillong 793003, India. She is presently working in IBM Systems, Bangalore, India (e-mail: sheba.diamond99@gmail.com; phone: +918787311900).

Mridul Haque is with the Department of Computational and Data Science, Indian Institute of Science, Bangalore 560012, India (e-mail: mridulh7@gmail.com; phone: +918876727066).

Prabir Saha is with the Department of Electronics and Communication Engineering, National Institute of Technology Meghalaya, Shillong 793003, India (e-mail: sahaprabir1@gmail.com; phone: +919485177005; fax: +91364-2501113).

RNS-based architectures should be re-evaluated as the VLSI technology has added new dimensions to the implementation of RNS-based architectures. The multiplier comprises two stages: in the first stage, the partial product is generated using an array of AND gates. In the next stage, the addition of the partial products (reduction) is added using the adders and compressors. The traditional weighted number system suffers from the carry propagation from low to high significant digits due to a slowdown in arithmetic operations like multiplication and addition. The carry can be speedup using special techniques but increases the hardware complexity. The works [1]–[18] reported in the literature so far on modulo multipliers discussed the VLSI implementation of the circuitry. The algorithm that seems impossible to implement circuity now has interesting implementation possibilities for the future. Thereby, the amalgamation of conventional and unconventional computer arithmetic methods will pave a new way for the investigation of new designs in the near future.

This paper presents algorithms to compute the modulo multiplications, viz. 2^n, 2^n-1, 2^n+1 moduli set. In addition to this, digital design circuity for implementation of the algorithm has also been discussed. The multi-column compressor has been used in the column reduction stage of the multiplication process with the aim of reducing computation time. The universal modulo multiplication deal with the diminished-one representation of the numbers. To do the modulo multiplication through the universal design hardware, the complexity of the circuit overhead goes increase, where the computational delay and power go increase, thereby efficiency of the VLSI design may go down. However, through the proposed designs, the same multiplier can be utilized for all the multiplication cases, thereby data re-usability can be possible for the fixed number value of 'n'.

II. Methodology

This section describes the proposed methodology to carry out the modulo multiplier computation. Firstly, the components used in the design to compute the operations are being discussed. Next, the required Boolean expressions of the components. Finally, the method to compute the modulo multiplication has been discussed. Fig. 1 shows the steps involved in the algorithms by pictorial depictions.The methodology begins

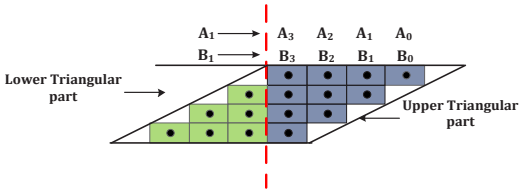

Fig. 1: Separation of the partial products into two halves. The vertical red line is the cutset of separation.

with the separation of the partial products obtained through multiplication of two operands A_n and B_n. The vertical red line in the Fig. 1 separates the partial products into two halves, namely, the upper triangular part and lower triangular part. The columns in the two halves are reduced (or added) separately to compute the result of the modulo multiplication. Fig. 2 shows the column addition scheme used in the proposed method to compute the modulo 2^n multiplication, where the components including half adders, full adders, and 4:3 compressor have been used.The Boolean expressions for the output of 4:3

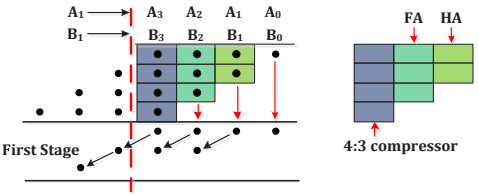

Fig. 2: Components used in the addition of partial product in the modulo multiplier designs. (The vertical dashed red line is used to divide the upper and lower triangular parts of the partial products).

compressor are:

$$Q_0 = (A \oplus B) \oplus (C \oplus D)\,, \tag{1}$$

$$Q_1 = A\,(C \oplus D) + B\,(A \oplus C) + D\,(B \oplus C)\,, \tag{2}$$

$$Q_2 = ABCD. \tag{3}$$

Assigning $A \oplus B = S_1$, $C \oplus D = S_2$, $A \oplus C = S_3$, and $B \oplus C = S_4$, the expressions (1)-(2) is rewritten as:

$$Q_0 = S_1 \oplus S_2, \tag{4}$$

$$Q_1 = AS_2 + BS_3 + DS_4. \tag{5}$$

A. Modulo 2^n Multiplier

This subsection discusses the method of the modulo 2^n multiplier operation. Fig. 3 shows a pictorial representation of the steps involved in 2^n modulo multiplication by taking an example of the 15×15 Mod 2^n. Appendix A shows the algorithm

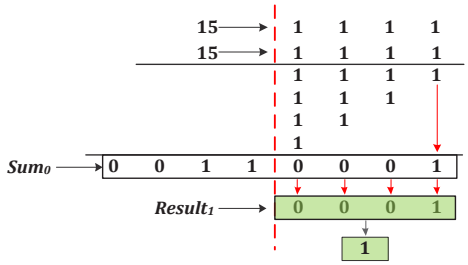

Fig. 3: Pictorial representation showing the steps to compute the modulo 2^n of 15×15.

of the procedural steps involved in computation of 2^n modulo multiplication. In the computation of 15×15 Mod 2^n modulo multiplication [Fig. 3], only the upper triangular portion of the partial products is considered. The partial products in the upper triangular portion upon performing the column reduction yields Sum_0 = "00110001". The last four bits of the register Sum_0 is considered as the final result of the operation i.e. $Result_0$ = "0001" which is = 1.

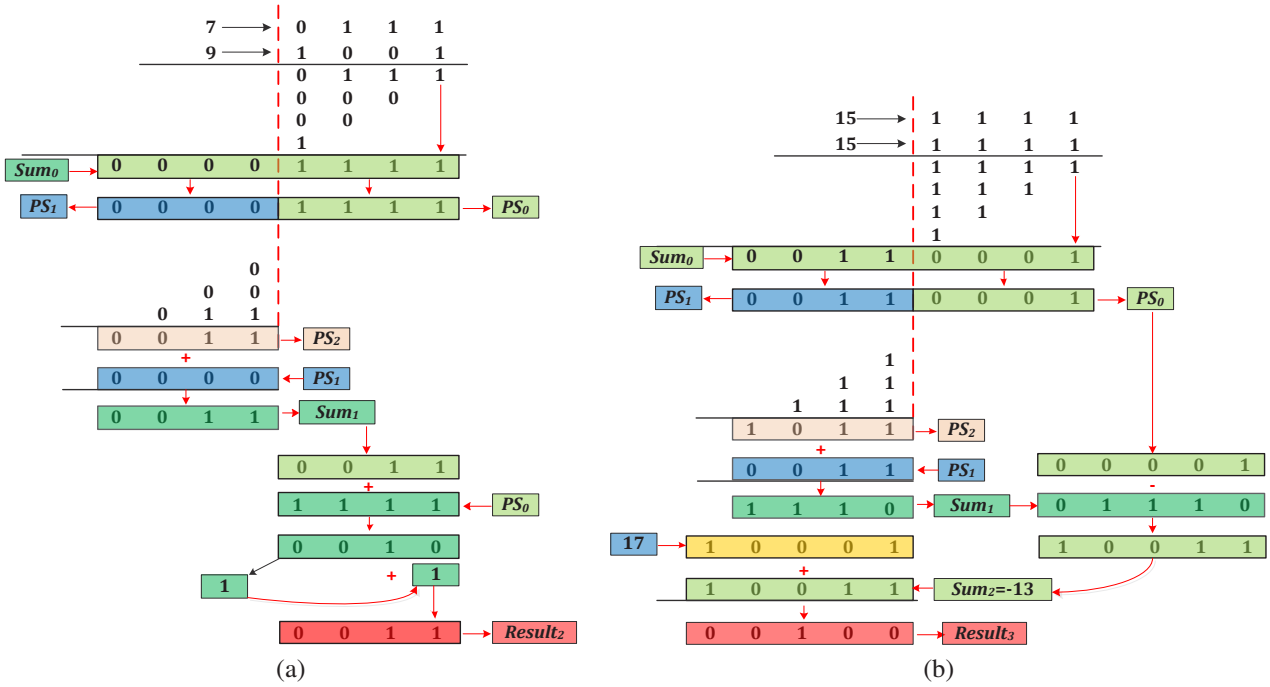

Fig. 4: Pictorial representation showing the steps to compute the (a) modulo 2^n-1 of 7×9 (b) modulo 2^n+1 of 15×15.

B. Modulo 2^n-1 Multiplier

In this subsection, we describe the modulo $2n-1$ multiplier. The procedural steps given in Appendix B represents the computation method of the modulo 2^n-1 multiplication. This multiplication process is elaborated by considering the example 7×9 Mod 2^n-1 as shown in Fig. 4(a). Unlike the modulo 2^n multiplication operation, the moduli $2^n\pm1$ multiplication operation considers both the upper and lower triangular portions of the partial products and their column reductions are performed separately. The 8-bits generated from the column reduction of upper triangular portion stored in the register Sum_0 are divided in two parts each consisting of four bits in the registers PS_0 and PS_1. The other 4-bits generated from column reduction of lower triangular part are stored in the register PS_2. The content of the registers PS_2 and PS_1 are added by a 4-bit ripple carry adder (RCA) [19] and stored in the register Sum_1 ("0011"). The content of Sum_1 is then added with that of PS_0 which result "10010". Since there is a carry bit '1', the carry bit is again added to the content of Sum_1 which upon addition gives the final result ($Result_1$ = "0011") of the operation.

C. Modulo 2^n+1 Multiplier

In this subsection, the operation of the modulo 2^n+1 multiplier by taking the example 15×15 Mod 2^n+1 has been discussed along with the operational procedure given in Appendix C. As stated earlier that there is a requirement of an extra bit (i.e. 5 bits in case of $n = 4$) in the modulo 2^n+1 operation, so the content of registers PS_0 and Sum_1 are padded with '0' (refer AppendixC). In the modulo 2^n+1 multiplication operation, the content of Sum_1 is subtracted (using 4 bit subtractor [19]) from PS_0 and the resultant is stored in the register Sum_2 as shown in Fig. 4(b). If the value of the content of the Sum_2 is negative i.e. $Sum_2(4)$ = '1', then the Sum_2 is added with the value 17 (= 2^4+1) to get the final result ($Result_3$ = "00100"). Otherwise if the content of Sum_2 is positive i.e. $Sum_2(4)$ = '0', then the final result ($Result_3 \leftarrow Sum_3$).

III. Results and Discussion

The modulo multipliers in gate level have been designed using VHDL coding in Xilinx ISE 14.2 design environment to verify the functionalities with operand sizes of 4, 8, and 16-bits. Keeping in mind the FPGA implementation of the proposed moduli-multipliers, the simulation is carried out using the Virtex-6 device. After performing extensive simulations, each design's delay and power metrics are reported from the synthesis and power reports. The number of LUTs in the Virtex-6 device is 474240 [20] and the static power dissipation is 4.447 W. The power consumption is calculated using the relation:

$$Power = \frac{Static\ Power}{Total\ number\ of\ LUTs} \times Number\ of\ LUTs\ used. \quad (6)$$

Fig 5 shows the column addition scheme of modulo multiplier of operand size 8 bits based in Wallace tree architecture. The components used in these large-sized moduli multipliers are half adders, full adders, 4:3 compressor [21], 7:3 compressor [22], and 5,5:4 compressor [23]. Table I shows the performance parameters obtained for the 4, 8, and 16-bit moduli-multipliers simulated by using Virtex-6 in Xilinx platform. On the other hand, to take care of the ASIC implementation, the moduli-multipliers are simulated in Cadence

TABLE I: SUMMARY OF POWER CONSUMPTION, DELAY, PDP, AND EDP OF THE MODULO MULTIPLIERS (n = 4, 8, AND 16-BITS) OBTAINED FROM SIMULATION IN THE DEVICE VIRTEX-6.

Designs	Number of LUTs used			Power (μW)			Delay (ns)			PDP (nJ)			EDP (nJ×ns)		
	n =4	n =8	n =16	n =4	n =8	n =16	n =4	n =8	n =16	n =4	n =8	n =16	n =4	n =8	n =16
2^n	8	56	300	75.01	525.12	2813.13	1.89	5.00	9.22	0.14	2.63	23.94	0.269	13.165	220.70
2^n-1	39	183	801	365.70	1716.01	7511.06	7.31	12.37	21.86	2.68	21.22	164.19	19.570	262.440	3589.23
2^n+1	35	181	784	328.20	1697.26	7351.65	4.69	12.07	17.24	1.54	20.45	126.74	7.211	247.221	2185.04

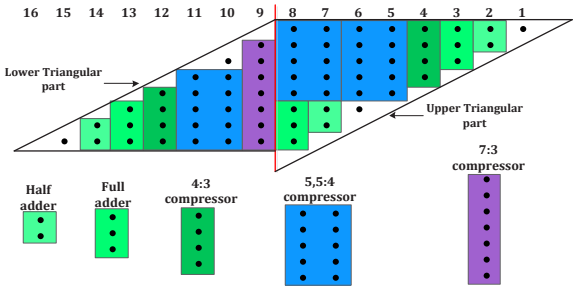

Fig. 5: Column addition scheme used in the Wallace tree architecture of moduli multipliers with operand size of 8 bits.

TABLE II: SUMMARY OF POWER CONSUMPTION, AREA, DELAY, PDP, AND EDP OF THE MODULO MULTIPLIERS (n = 4, 8, AND 16 BITS) OBTAINED FROM SIMULATION IN CADENCE RC COMPILER USING 0.18 μM TECHNOLOGY.

Designs	Power(μW)			Area(μm^2)		
	n =4	n =8	n =16	n =4	n =8	n =16
2^n	84.13	1105.34	3347.34	319.87	3151.75	10383.64
2^n-1	474.83	845.74	12036.79	1276.38	7197.45	42938.44
2^n+1	498.18	2548.21	13369.79	1335.94	10148.15	54918.43

RC compiler considering the Semi-Conductor Laboratory, India (SCL) 0.18 μm technology whose results are listed in Table II. The power metrics listed in Table II is the total power comprising both static and dynamic power. Figure 6 shows the area of moduli-multipliers at 0.18 μm technology extracted from the simulation. From Fig. 6, one can estimate the scalability of the moduli-multipliers with respect to the change in operand size. The increase in area deviates from linear relation (in log scale) as the operand size increases from $n = 8$ to $n = 16$.

A comparison of area of modulo 2^n-1 multiplier with the state of the art at 0.18 μm technology is listed in Table III. From Table III, it is observed that the proposed modulo 2^n-1 multiplier of 8-bit operand size shows an improvement of 50% in terms of the area over the reported work in [18]. A comparison of the modulo 2^n+1 multiplier with the state of the art is listed in Table IV. The proposed modulo 2^n+1 multiplier of 4-bit operand size shows 66.34% improvement in power consumption over that of reported in [24]. Figure 7 shows a pictorial representation of the comparison of power consumption of modulo 2^n+1 multiplier with the prior reported works. Similarly, Fig. 8 shows a pictorial representation of the comparison of area of the modulo 2^n+1 multiplier with prior reported works. The proposed modulo 2^n+1 multiplier

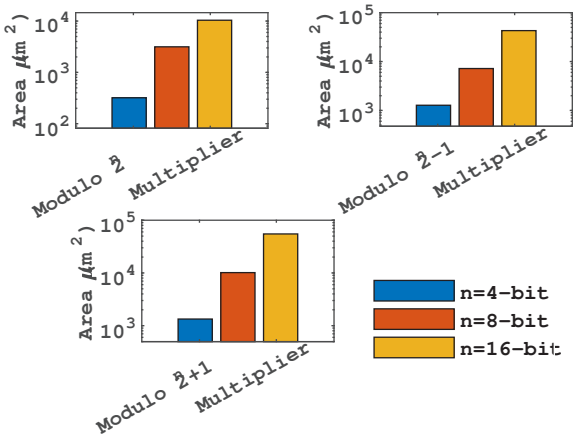

Fig. 6: Comparison of area of the moduli-multipliers with the increase in operand size.

TABLE III: COMPARISON OF PERFORMANCE PARAMETERS OF MODULO 2^n-1 MULTIPLIER AT 0.18 μM WITH PRIOR REPORTED WORK.

Area (μm^2)	Operand size (n)	
	n = 8	n = 16
This work	7197.45	42938.44
[18]	14516.40	57630.0
Relative improvement (%)	50.42	25.49

of 4-bit operand size shows an improvement of 58.59% in terms of area over the best-reported paper [26]. In case of 8-bit operand size, the proposed modulo 2^n+1 multiplier shows a 22.72% reduction in the area as compared to that reported in [26].

To do the direct conversion from decimal to RNS and vice versa the same proposed algorithms can be followed. For the RNS to decimal conversion, the Chinese Remainder Theorem or mixed-radix conversion algorithm has to be followed. The hardware design of such an algorithm is also a complex task. However, the algorithm for such conversion has been designed in the same computational platform and added the result in Table V. This presented work discusses the bit-by-bit modulo multiplication of unsigned numbers. For the signed multiplication, the proposed modulo multipliers can be worked in the following manner:

$$-z = \overline{z} = Mul(x, y)\ Mod\ M = \overline{xy}\ r^{-1}\ Mod\ M, \quad (7)$$

where r^{-1} is the inverse of $r\ Mod\ M$. The signed modulo

TABLE IV: COMPARISON OF PERFORMANCE PARAMETERS OF MODULO 2^n+1 MULTIPLIER AT 0.18 μM WITH PRIOR REPORTED WORKS.

Power (μW)	Operand size (n) $n=4$	$n=8$	$n=16$
This work	498.18	2548.21	13369.79
[24]	1480.0	2960.0	8930.0
[25]	246.80	1150.0	5024.0
[26]	296.7	1112.0	4613.0

Area (μm^2)	Operand size (n) $n=4$	$n=8$	$n=16$
This work	1335.94	10148.15	54918.43
[18]		20889.80	69555.17
[24]	4119.10	13354.20	49237.0
[25]	3768.0	15108.0	53155.0
[26]	3226.0	13132.0	43861.0

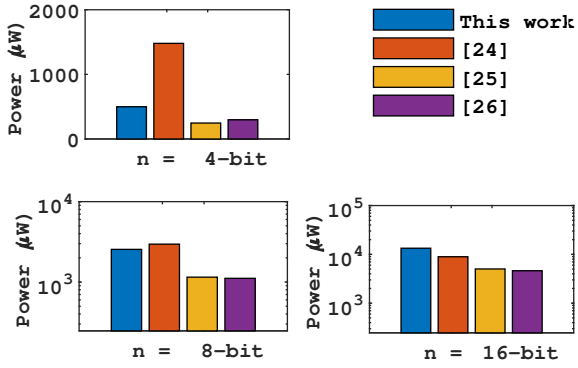

Fig. 7: Pictorial comparison of power consumption of modulo 2^n+1 multiplier with prior reported works.

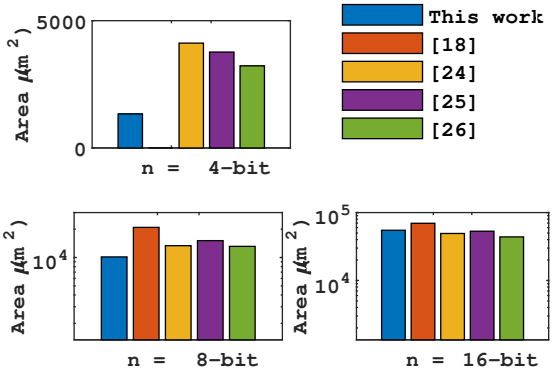

Fig. 8: Pictorial comparison of the area modulo 2^n+1 multiplier with prior reported works.

multiplication is also applicable to Booth multiplication.

IV. CONCLUSION

This paper presents a new methodology and algorithm for the implementation of modulo multiplies namely 2^n, 2^n-1, and 2^n+1. The simulation of the modulo multipliers has been carried out considering operand size of 4, 8, 16-bits. The modulo multipliers are designed using VHDL coding and this makes the design suitable for integrating with high-performance modulo arithmetic units. The expected outputs are verified in the Xilinx 14.2 tool. Moreover, power consumption, delay, PDP, and EDP are obtained using the Virtex-6 device. Towards the ASIC implementation the designs are simulated in Cadence RC compiler using SCL-0.18 μm technology node. The performance parameters of the proposed moduli-multipliers in 0.18 μm technology have been compared with the state of the art. The modulo 2^n-1 multiplier of operand size 4-bit shows an improvement of 66.34% in terms of area over the best-reported paper. On the other hand, the modulo 2^n+1 multiplier of operand size 4-bit shows an improvement of 58.59% in terms of the area and the same of operand size 8-bit shows an improvement of 22.72% over the best-reported paper. As future works, this presented work can be extended to support modulo multiplier for operands of higher order bits ($n = 32$).

ACKNOWLEDGMENT

This work was supported by the project grant with file no. NITMGH/TEQIP-III/MP/2019-20/252. The authors would like to thank Dr. Phrangboklang Lyngton Thangkhiew (Assistant professor, Indian Institute of Information Technology Guwahati, India), and Thulasiram Varma K. (Research scholar, National Institute of Technology Meghalaya, India) for their help and useful suggestions.

APPENDIX A
ALGORITHMIC REPRESENTATION OF MODULO 2^n MULTIPLICATION

Algorithm 1: Algorithm to compute 2^n modular multiplication, i.e. $(A \times B)\ Mod\ 2^n$ where $n = 4$.

Input: Input operands A and B
Output: $Result_1 = (A \times B)\ Mod\ 2^n$, n is the number of bits.
Data: **Partial products** (A_iB_j). **Intermediate signals** ($X_i[0:1]$, $T_i[0:1]$, $U_i[0:1]$). **Register** (PS_0).
Define n_1 =3
for j *in 0 to 3* **do**
  for i *in 0 to* n_1 **do**
    $A_iB_j = A_i\ and\ B_j$
  end
  $n_1 = n_1 - 1$
end
$PS_0(0) \leftarrow A_0B_0$
Call ***half adder***$[A_1B_0,\ A_0B_1,\ X_0(0),\ X_0(1)]$
Call ***full adder***$[A_2B_0,\ A_1B_1,\ A_0B_2,\ X_1(0),\ X_1(1)]$
Call ***4:3 compressor***$[A_3B_0,\ A_2B_1,\ A_1B_2,\ A_0B_3,\ X_2(2),\ X_2(1),\ X_2(0)]$
$PS_0(1) \leftarrow X_0(0)$

TABLE V: SUMMARY OF POWER CONSUMPTION, DELAY, PDP, AND EDP FOR THE CONVERSION FROM DECIMAL TO RNS AND RNS TO DECIMAL (8-BIT) OBTAINED FROM SIMULATION IN THE DEVICE VIRTEX-6.

Designs	Number of LUTs used			Power (μW)			Delay (ns)			PDP (nJ)			EDP (nJ×ns)		
	$n=4$	$n=8$	$n=16$	$n=4$	$n=8$	$n=16$	$n=4$	$n=8$	$n=16$	$n=4$	$n=8$	$n=16$	$n=4$	$n=8$	$n=16$
2^n	31	69	156	250.87	603.89	1159.76	2.52	5.59	12.31	0.63	3.38	14.28	1.59	18.89	175.77
2^n-1	131	287	631	1257.39	2522.53	6017.98	8.75	17.9	36.53	11.00	45.15	219.83	96.25	808.18	8030.39
2^n+1	129	276	620	1199.58	2427.07	5091.39	8.23	17.2	35.27	9.87	41.75	179.57	81.23	718.10	6333.43

Algorithm 1: Algorithm 1 *Contd..*

Call ***half adder***[$X_1(0)$, $X_0(1)$, $T_0(0)$, $T_0(1)$]
Call ***half adder***[$X_2(0)$, $X_1(1)$, $T_1(0)$, $T_1(1)$]
$PS_0(2) \leftarrow T_0(0)$
Call ***half adder***[$T_1(0)$, $T_0(1)$, $U_0(0)$, $U_0(1)$]
Call ***half adder***[$X_2(1)$, $T_1(1)$, $U_1(0)$, $U_1(1)$]
$PS_0(3) \leftarrow U_0(0)$
$Result_0 \leftarrow PS_0$

APPENDIX B
ALGORITHMIC REPRESENTATION OF MODULO 2^n-1 MULTIPLICATION

Algorithm 2: Algorithm to compute 2^n-1 modular multiplication, i.e. $(A \times B)\ Mod\ 2^n-1$ where $n = 4$.

Input: Input operands A and B
Output: $Result_1 = (A \times B)\ Mod\ 2^n-1$ where n is the number of bits.
Data: Partial products (A_iB_j). **Intermediate signals** ($X_i[0:,]$, $T_i[0:1]$, $U_i[0:1]$, $V_i[0:1]$, $W_0[0:1]$, $Z_0[0:1]$, $I_i[0:1]$, $J_i[0:1]$, $K_0[0:1]$, $L_0[0:1]$, $C_{in1}[0:4]$, $C_{in2}[0:4]$, $C_{in3}[0:4]$). **Registers** ($PS_1[0:3]$, $PS_2[0:3]$, $Sum_1[0:4]$, $Res_1[0:3]$, $Res_2[0:3]$, $Result_1[0:3]$).
for *j in 0 to 3* **do**
  for *i in 0 to 3* **do**
    $A_iB_j = A_i\ and\ B_j$,
  end
end
Call ***half adder***[$U_1(1)$, $U_0(1)$, $V_0(0)$, $V_0(1)$]
Call ***half adder***[$X_2(2)$, $U_1(1)$, $V_1(0)$, $V_1(1)$]
$PS_1(0) \leftarrow V_0(0)$
Call ***half adder***[$V_1(0)$, $V_0(1)$, $W_0(0)$, $W_0(1)$]
$PS_1(1) \leftarrow W_0(0)$
Call ***half adder***[$V_1(1)$, $W_0(1)$, $Z_0(0)$, $Z_0(1)$]
$PS_1(1) \leftarrow W_0(0)$
$PS_1(2) \leftarrow Z_0(0)$
$PS_1(3) \leftarrow Z_0(1)$
Call ***full adder***[A_3B_1, A_2B_2, A_1B_3, $I_0(0)$, $I_0(1)$]
Call ***half adder***[A_3B_2, A_2B_3, $I_1(0)$, $I_1(1)$]
$PS_2(0) \leftarrow I0(0)$
Call ***half adder***[$I_1(0)$, $I_0(1)$, $J_0(0)$, $J_0(1)$]
$PS_2(1) \leftarrow J_0(0)$
Call ***half adder***[$J_1(0)$, $J_0(1)$, $K_0(0)$, $K_0(1)$]

Algorithm 2: Algorithm 2 *Contd..*

$PS_2(2) \leftarrow K_0(0)$
Call ***half adder***[$J_1(1)$, $K_0(1)$, $L_0(0)$, $L_0(1)$]
$PS_2(3) \leftarrow L_0(0)$
Assign $C_{in1}(0) \leftarrow$'0'
Call ***full adder***[$PS_1(4)$, $PS_2(0)$, $C_{in1}(0)$, $Sum_1(0)$, $C_{in1}(1)$]
Call ***full adder***[$PS_1(5)$, $PS_2(1)$, $C_{in1}(1)$, $Sum_1(1)$, $C_{in1}(2)$]
Call ***full adder***[$PS_1(6)$, $PS_2(2)$, $C_{in1}(2)$, $Sum_1(2)$, $C_{in1}(3)$]
Call ***full adder***[$PS_1(7)$, $PS_2(3)$, $C_{in1}(3)$, $Sum_1(3)$, $C_{in1}(4)$]
Assign $C_{in2}(0) \leftarrow$'0'
Call ***full adder***[$Sum_1(0)$, $PS_0(0)$, $C_{in2}(0)$, $Res_1(0)$, $C_{in2}(1)$]; `// Refer Algorithm 1 for` PS_0
Call ***full adder***[$Sum_1(1)$, $PS_0(1)$, $C_{in2}(1)$, $Res_1(1)$, $C_{in2}(2)$]
Call ***full adder***[$Sum_1(2)$, $PS_0(2)$, $C_{in2}(2)$, $Res_1(2)$, $C_{in2}(3)$]
Call ***full adder***[$Sum_1(3)$, $PS_0(3)$, $C_{in2}(3)$, $Res_1(3)$, $C_{in2}(4)$]
Assign $C_{in3}(0) \leftarrow$'0'
Call ***full adder***[$C_{in2}(4)$, $Res_1(0)$, $C_{in3}(0)$, $Res_2(0)$, $C_{in3}(1)$]
Call ***full adder***[0, $Res_1(1)$, $C_{in3}(1)$, $Res_2(1)$, $C_{in3}(2)$]
Call ***full adder***[0, $Res_1(2)$, $C_{in3}(2)$, $Res_2(2)$, $C_{in3}(2)$]
Call ***full adder***[0, $Res_1(3)$, $C_{in3}(3)$, $Res_2(3)$, $C_{in3}(4)$]
while *There is a change in the value of* Res_2 **do**
  if Res_2 =*"1111"* **then**
    $Result_2 \leftarrow$"0000"
  else
    $Result_2 \leftarrow Res_2$
  end
end

APPENDIX C
ALGORITHMIC REPRESENTATION OF MODULO 2^n+1 MULTIPLICATION

Algorithm 3: Algorithm to compute 2^n+1 modular multiplication, i.e. $(A \times B)\ Mod\ 2^n+1$ where $n = 4$.

Input: Input operands A and B
Output: $Result_3 = (A \times B)\ Mod\ 2^n+1$, n is the number of bits.
Data: Partial products (A_iB_j). **Register** ($PS_3[0:4]$, $PS_4[0:4]$, $Sum_3[0:4]$, $Res_3[0:4]$).
$PS_3 \leftarrow$ '0'& PS_0 // Zero padding to PS_0 (PS_0 from Algorithm 1)
$Sum_3 \leftarrow$ '0'& Sum_1 // Zero padding to Sum_1 (Sum_1 from Algorithm 2)
$C_{in4}(0) \leftarrow$ '1'
$PS_4(0) \leftarrow PS_3(0)\ xor\ C_{in4}(0)$
$PS_4(1) \leftarrow PS_3(1)\ xor\ C_{in4}(0)$
$PS_4(2) \leftarrow PS_3(2)\ xor\ C_{in4}(0)$
$PS_4(3) \leftarrow PS_3(3)\ xor\ C_{in4}(0)$
$PS_4(3) \leftarrow PS_3(4)\ xor\ C_{in4}(0)$
Call ***full adder***[$Sum_3(0)$, $PS_4(0)$, $C_{in4}(0)$, $Res_3(0)$, $C_{in4}(1)$]
Call ***full adder***[$Sum_3(1)$, $PS_4(1)$, $C_{in4}(1)$, $Res_3(1)$, $C_{in4}(2)$]
Call ***full adder***[$Sum_3(2)$, $PS_4(2)$, $C_{in4}(2)$, $Res_3(2)$, $C_{in4}(3)$]
Call ***full adder***[$Sum_3(3)$, $PS_4(3)$, $C_{in4}(3)$, $Res_3(3)$, $C_{in4}(4)$]
Call ***full adder***[$Sum_3(4)$, $PS_4(4)$, $C_{in4}(4)$, $Res_3(4)$, $C_{in4}(5)$]
$f \leftarrow$"10001" // Assigning 17 to the Register f
while *There is a change in the value of* Res_3 **do**
  if $Res_3(4) =$'1' // i.e. Value in Res_3 is negative **then**
    $Result_3 \leftarrow f + Res_3$
  else
    $Result_3 \leftarrow Res_3$
  end
end

REFERENCES

[1] M.A. Soderstrand, W.K. Jenkins, G.A. Jullien, F.J. Taylor, *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*, New York: IEEE Press, 1986.

[2] X. Lai, J.L. Massey, "A Proposal for a New Block Encryption Standard," in *Advances in Cryptology – Lecture Notes in Computer Science*, vol. 473, Berlin, Germany: Springer-Verlag, 1990, pp. 389–404.

[3] E. Gholami, R. Farshidi, M. Hosseinzadeh, K. Navi, "High speed residue number system comparison for the moduli set (2^n-1,2^n,2^n+1)," *J. Commun. Comput.*, vol. 6, no. 3, pp. 40-46, Jan. 2009.

[4] T.R.N. Rao, E. Fujiwara, *Error Control Coding for Computer Systems*, New Jersey, USA: Prentice-Hall, 1989.

[5] H.T. Vergos, D. Bakalis, C. Efstathiou, "Efficient modulo 2^n+1 multi-operand adders," *Proc 15th IEEE Int. Conf. Electron, Circuits Syst*., St. Julien's, pp. 694-697, 2008.

[6] H.T. Vergos, G. Dimitrakopoulos, "On modulo 2^n+1 adder design," [7] L. Kalampoukas *et al.*, "High-speed parallel-prefix modulo 2^n-1 adders," *IEEE Trans. Comput.*, vol. 49, no. 7, pp. 673–680, Jul. 2000.

[8] C. Efstathiou, H.T. Vergos, D. Nikolos, "Modulo $2^n \pm 1$ adder design using select prefix blocks," *IEEE Trans. Comput.*, vol. 52, no. 11, pp. 1399–1406, Nov. 2003.

[9] A.V. Curiger, H. Bonnenberg, H. Kaeslin, "Regular VLSI architectures for multiplication modulo (2^n+ 1)," *IEEE J. Solid-State Circuits*, vol. 26, no. 7, pp. 990-994, Jul. 1991.

[10] Z. Wang, G.A. Jullien, W.C. Miller, "An efficient tree architecture for modulo 2n+1 multiplication," *J. VLSI Sign. Process. Syst. Sign. Image Video Technol.*, vol. 14, pp. 241–248, Dec. 1996.

[11] R. Zimmermann, "Efficient VLSI implementation of modulo ($2^n \pm 1$) addition and multiplication," *Proc. 14th IEEE Symp. Computer Arithmetic*, Adelaide, SA, Australia, pp. 158-167, 1999.

[12] A. Skavantzos, "Design of multioperand carry-save adders for arithmetic modulo (2^n+1)," *Electronics Lett.*, vol. 25, no. 17, pp. 1152-1153, 1989.

[13] G. Jaberipur, B. Parhami, "Unified approach to the design of modulo-($2^n \pm 1$) adders based on signed-LSB representation of residues," *Proc. 19th IEEE Symp. Computer Arithmetic*, Portland, pp. 57-64, 2009.

[14] S.-H. Lin, M.-H. Sheu, "VLSI Design of diminished-one modulo 2^n+1 adder using circular carry selection," *IEEE Trans. Circuits Syst. II: Express Briefs*, vol. 55, no. 9, pp. 897-901, May 2008.

[15] G. Dimitrakopoulos, D.G. Nikolos, H.T. Vergos, D. Nikolos,C. Efstathiou, "New architectures for modulo 2^N-1 adders," *Proc. 12th IEEE Int. Conf. Electron, Circuits Syst.*, Gammarth, pp. 1-4, 2005.

[16] L.M. Leibowitz, "A simplified binary arithmetic for the fermat number transform," *IEEE Trans. Acoustics Speech, Signal Process.*, vol. 24, no. 5, pp. 356-359, Oct. 1976.

[17] Y. Ma, "A simplified architecture for modulo (2^n+1) multiplication," *IEEE Trans. Comput.*, vol. 47, no. 3, pp. 333-337, Mar. 1998.

[18] S. Menon, C. Chang, "A reconfigurable multi-modulus modulo multiplier," *Proc. IEEE Asia-Pacific Conf. Circuits Syst.*, Singapore, pp. 1168-1171, 2006.

[19] M.M. Mano, M.D. Ciletti, *Digital Design*, New Jersey: Pearson Education, 2007.

[20] E. Stavinov, *100 Power Tips for FPGA Designers*, Paramount, CA: CreateSpace, 2011.

[21] P. Saha, R.U. Ahmed, S.D. Thabah, "Design and implementation of multi-operand 2^n-1, 2^n, and 2^n+1 modulo set adder," in *Advances in Communication, Devices and Networking. Lecture Notes in Electrical Engineering*, vol. 776, S. Dhar, S.C. Mukhopadhyay, S.N. Sur, C.M. Liu Ed., Singapore: Springer, 2022, pp. 1-8.

[22] R.U. Ahmed, P. Saha, "Power and delay comparison of 7:3 compressor designs based on different architectures of XOR gate," in *Inventive Communication and Computational Technologies. Lecture Notes in Networks and Systems*, vol. 89, G. Ranganathan, J. Chen, A. Rocha Ed, Singapore: Springer, 2020, pp. 461–469.

[23] R. U. Ahmed, S.D. Thabah, P. Saha, "Design of new multi-column 5,5:4 compressor circuit based on double-gate UTBSOI transistors," *Procedia Comput. Science*, vol. 165, pp. 532-540, 2019.

[24] J.W. Chen, R.H. Yao, and W.J. Wu, "Efficient 2^n+1 modulo multipliers," *IEEE Trans. VLSI*, vol. 19, no. 12, pp. 2149-2157, Dec. 2011.

[25] T.B. Juang, C.T. Kuo, G.L. Wu, and J.H. Huang, "Multifunction RNS modulo $2^n \pm 1$ multipliers," *J. Circuits Syst. Comput.*, vol. 21, no. 04, pp. 1250027(1-13), 2012.

[26] H. T. Vergos and C. Efstathiou, "Design of efficient modulo multipliers," *IET Comput. Digital Tech.*, vol. 1, no. 1, pp. 49–57, Jan. 2007.