Journal of Engg. Research ICAPIE Special Issue pp. 8-18 DOI:10.36909/jer.ICAPIE.15073

# Area Efficient Diminished $2^n$-1 Modulo Adder using Parallel Prefix Adder

Beerendra K. Patel*and Jitendra Kanungo

*Department of Electronics & Communication Engineering, Jaypee University of Engineering & Technology, Guna-473226, India.*
**Corresponding Author: beerendrapatel23@gmail.com*

## ABSTRACT

Residue Number System has to carry free operation and its various applications like digital signal processing, multimedia, security purpose, and medical perception. Removes of the redundant logic operation require the group carry selection logic which is dependent on Parallel Prefix Adder design. Therefore the logic operation of the pre-processing unit of PPA is simplified form to save logic resources. This modified parallel prefix adder consumes less area as compared to the existing design. In this paper, we propose the parallel architecture based on a parallel prefix tree is helpful for computation at higher speed operation. The reported design consumes 24.1% more area and 26.4% more power compare to THE proposed parallel prefix adder design. The proposed PPA design using modified carry computation algorithm and reported design used diminished-1 modulo 2n+1 adder structure is presented. A presented modulo adder design using the proposed parallel prefix adder and improved carry computation used in the previously proposed design. The proposed diminished-1 modulo ($2^n$ +1) adder design shows a 24.5% saving in area-delay-product (ADP).

**Keywords:** Parallel Prefix adder; Computer arithmetic; Diminished-1 representation.

## INTRODUCTION

In the field of digital computer arithmetic, the residue number system offers advantages in terms of conversion of higher arithmetic integers into a smaller arithmetic number via parallel operation. An integer is a set of residues called moduli. RNS is a highly scalable and fast technique that is suitable for high performance (R.P. Bent et al., 1982). In a portable device design, the scaling technique is the better option for low-power digital circuit design. RNS has performed carry-free operations for compact, high-speed implementation of circuits (R. Ladner et al.,1980, A. A. Hiasat,2002). RNS applications like digital image processing, widely used in medical perception, machine view, military target finding, multimedia, and security purpose requires low power consumption and high speed. RNS achieves high speed with a parallelism nature. RNS has been adopted in the design of Digital Signal Processors (DSP). Modulo $2^n$+1 multiplier is widely used in a wide range of applications including random number generator which has remarkable applications in cryptographic algorithms (Kogge P.,1973). Various cryptographic systems have been studied and implemented to ensure the security of information. In the designing of a normal modulo adder, it is essential to use continual computing stages. (Patel et al.,2007) have defined implementation architecture for modulo $2^n$ - ($2^{n-2}$ + 1) adder. This architecture consists of a carry-computation unit. Thereby, repetitive elements of carrying computation stages are suppressed. However, the structure is the same as the carry computation unit as for $2^n$ - ($2^{n-2}$ + 1) modulus. In (S. Knowles 1999), authors have described various algorithms to generate the fast carry bit.

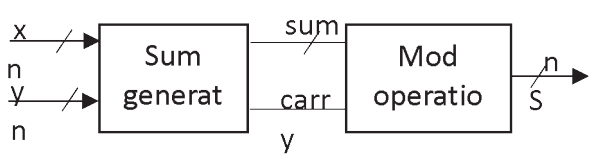


**Figure 1.** Basic diagram of modulo ($2^n$ +1) adder

Figure 1 shows the modulo adder has a two-block sum generation block and a selection of modulo block. The sum of two residue number perform sum reproduction block and modulo operation performs modulo group. The reversed form of carrying in sum reproduction unit restricted the modulo processed. A modulo adder two ripples carry adder (RCA). The single RCA has path delay of TA and two RCA has 2TA is the delay of RCA. To implement diminished-1 modulo (2n+1) adder has considered various types of modulo adder with less critical path delay. (S. Song et.al.,2002) proposed a design using carry select adder (CSLA) which reduces the half delay of modulo adder and to increases the combinational logic in an equal manner. Authors (R. Zimmerman,1999), using parallel prefix adder in diminished-1 modulo (2n+1) adder given possibility to reduce complexity. This adder using an extra carry increment stage in place of two CSLA (R. Zimmerman, 1999). Hence, mostly preferred parallel prefix adder in place of CSLA and generally used in design to modulo (2n+1) adder. In RNS the overhead is the main issue that can remove using carry rippling. For slash, the carry rippling inside the sum computation group is achieved by the CPD (Critical Path Delay) saving. For reducing carry rippling indoors the carry computation group of Parallel Prefix Adder remove overhead logic operation required different selection logic to formulated. To achieve the design efficiency of the parallel prefix adder the overhead logic operation should be reduced. To this end, study the review of parallel Prefix adder is presented in this section, presents an algorithm of parallel prefix adder has been done. Then, the modified logic formulation of parallel prefix adder has been described. Then proposed structure of diminished-one modulo 2n+1 adder has been performed. The goal is to save the area in a parallel prefix adder. Finally, the Synthesis results have been evaluated, the last section is the conclusion.

## REVIEW OF PARALLEL PREFIX ADDER

### SklanskyAdder

Sklansky prefix tree is an 8-bit adder. It is unpretentious among the prefix trees. Figure 2 shows the Sklansky tree. Sklansky adder uses fewer logic resources to compute the carries. It has reduced logic resource utilization as compared to Knowles and Kogge Stone prefix architecture (Kogge P.,1973). But it has a higher fan-out. The fan-out grows exponentially from input to output of such adder. This adder is a compact version of the Brent Kung adder where fan-out is increased and logic resources are reduced (H. T. Kung et al.,1982) (Kogge P.,1973).

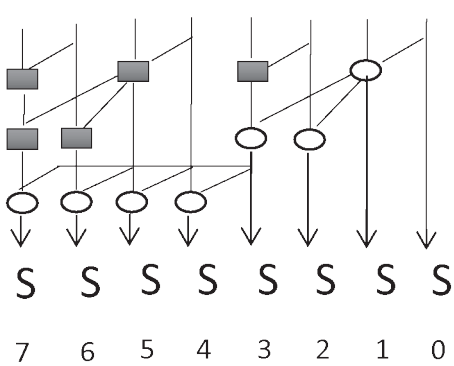


**Figure 2.** Sklansky adder [2]

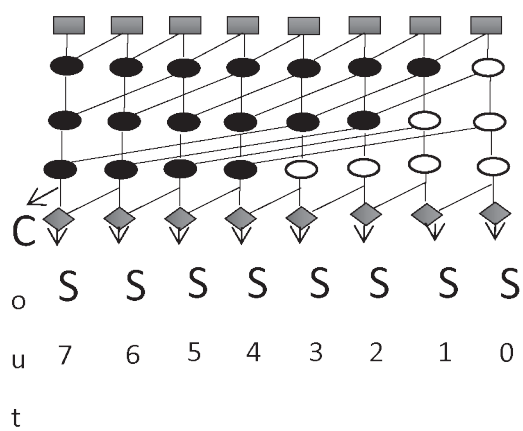


**Figure 3.** Kogge stone adder[4]

**Kogge Stone Adder**

The Kogge Stone adder is a parallel and regular structure. It is shown in Figure-3 for the 8-bit operand. This adder is extensively examined in the fastest adder design. It is generally used for high-performance adders. The Kogge Stone adder takes more logic resources to compute carries; it results in more area requirements than the Brent Kung. It has also a lower fan-out at each stage so it enhances the performance. Another drawback of the Kogge Stone adder is the wiring congestion.

**Brent Kung Adder**

Brent Kung adder has a large number of design levels so it reduces its computational speed. Figure 4 shows the Brent Kung prefix tree. It has an 8-bit operand. With a large number of inputs, it has the lowest area and delay.

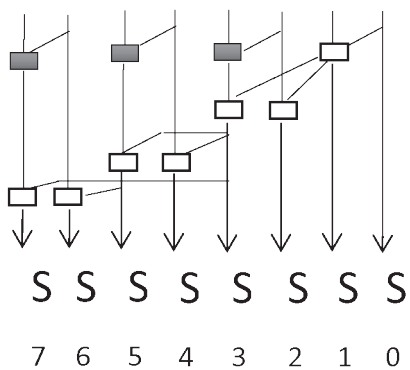


**Figure 4.** Brent Kung adder[2]

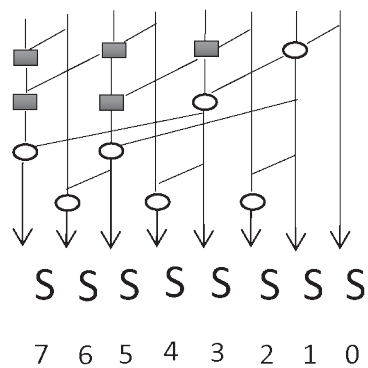


**Figure 5.** Han Carlson adder[3]

**Han Carlson Adder**

Han Carlson prefix adder as shown in Figure 5 is similar to the Kogge Stone adder as it has a maximum fan-out of 2.This adder uses fewer cells and wire tracks than the Kogge Stone. It has an 8-bit operand. It is observed as a sparse version of the Kogge Stone prefix tree because it requires extra logic levels.

Let, the parallel prefix adder adds two n-bit numbers $X = X_{n-1}, X_{n-2}, X_{n-3}, X_{n-4}, ....., X_0$ and $Y = Y_{n-1}, Y_{n-2}, Y_{n-3}, Y_{n-4}, ....., Y_0$ along with a carry input bit $C_{in}$, and produces n-bit sum bit and $S = S_1, S_2, S_3, S_4, ....., S_{n-1}$ output carry $C_{out}$. The input operands receive pre-processing unit x and y and compute three operands namely carry propagate bit p, carry generate bit g, and half sum bit h which size are n-bit are given as:

$$g_i = x_i, y_i, p_i = x_i + y_i, h_i = x_i \oplus y_i \qquad (1)$$

where symbol{.;+} denote logical OR, AND, and Ex-OR operations.

The pre-processing unit generates two terms, generate terms(g) and propagate terms(p), and these generate and propagate bit using carry computation unit as input and count the n-bit carry called Cout. The sum propagation unit using input Cout and half sum bit. The input Cout using (n-1) least significant bits(LSBs) from the carry estimation group and half sum bit produce from the pre-processing block. This paper presents Algorithm-1 (Kogge P, Stone H, 1973), the parallel prefix operators use in the carry computation unit.

1. **Algorithm-1 of carry computation**

Loop $= 0 \quad n - 1$
$(Gi,0\ ,\ Pi,0) = (gi\ ,\ pi)$
**end** loop;
Loop1 $j = 1$ **to** $J$ **do**
Loop2 $= 0$ **to** $n - 1$ **do**
**if** $(i < 2\ j - 1\ )$ **then**
$(Gi,j\ ,\ Pi,j\ ) = (Gi,-1\ ,\ Pi,j-1)$
**else** $(Gi,j\ ,\ Pi,j\ ) = (Gi,j-1\ ,\ Pi,j-1) \circ (Gi-2\ j-1\ ,j-1\ ,\ Pi-2\ j-1\ ,j-1\ )$
**end if**;
**end** loop2;
**end** loop1;
loop $i = 0$ **to** $n - 1$ **do**
$ci = Gi,J$
$c_{out} = c_{n-1}$
end loop;

$$(G,P) \circ (G_1.P_1) = (G + P_1G,\ PP_1) \tag{2}$$

Where $\circ$ operator defines the parallel prefix operator.

In the n-bit carry bit, the most significant bit(MSB) Cn-1 represent output carry bit Cout and the remaining bit called (n-1) Least Significant Bits of n-bit carry are forwarded to the final sum group, there are used to compute the n-bit sum S using half sum bit and the input carry. The half sum bit is generated by pre-processing block. The logic expression has given as-

$$c_{-1} = c_{in}$$
$$S_i = h_i \oplus c_{i-1} \tag{3}$$

**2. Algorithm-2 for Parallel Prefix Adder**

The carry computation algorithm (first algorithm) is represented in a modified algorithm by the authors (S. Mathew et al.,2003), it introduces at the time that the group carries computation algorithm (GCCA). The group generates bit and group propagate bit($GG_{k,l}$, $PP_{k,l}$) are generated. The GCCA is given as

**Algorithm of Group carry computation**
loop $k = 0$ **to** $K - 1$ **do**
$(GGk,0\ ,\ PPk,0) = \{g4k+3\ ,\ p4k+3 \circ g4k+2\ ,\ p4k+2 \circ g4k+1\ ,\ p4k+1 \circ g4k\ ,\ p4k\ \}$
**end** loop;
loop $l = 1$ **to** $L$ **do**
loop $k = 0$ **to** $K - 1$ **do**
**if** $(k < 2\ l-1\ )$ **then**
$(GGk,l\ ,\ PPk,l) = (GGk,-1\ ,\ PPk,l-1)$
**else**
$(GGk,l\ ,\ PPk,l) = (GGk,-1\ ,\ PPk,l-1) \circ (GGk-2\ l-1\ ,l-1\ ,\ PPk-2\ l-1\ ,l-1\ )$
**end if**;
**end** loop;
**end** loop ;
loop
$k = 0$ **to** $K - 1$ **do**
$Ck = GGk,\ cout = CK-1$
**end** loop;

Here, prefix level $\{s_{4k}, s_{4k+1}, s_{4k+2}, s_{4k+3}\}$ and group index is $k = (n/4)$

The carry computation unit depends on group carry computation algorithm which are generate K group carry bits $C_k$, for $0 \le k \le k-1$, where K=(n/4). All group carry bit computes and generate four intermediate carry signal . Using the logic expression

$$c_{4k-1} = c_{k-1} \tag{4.1}$$
$$c_{4k} = g_{4k} + p_{4k}.c_{4k-1} \tag{4.2}$$
$$c_{4k+1} = g_{4k+1} + p_{4k+1}.c_{4k} \tag{4.3}$$
$$c_{4k+2} = g_{4k+2} + p_{4k+2}.c_{4k+1} \tag{4.4}$$
$$c_{-1} = c_{in} \qquad c_{out} = c_{k-1} \tag{4.5}$$

Carry bits called a group of four intermediate signal $\{C_{sk-1}, C_{4k}, C_{4k+1}, C_{4k+2}\}$ ,a group of half sum bits $\{h_{4k}, h_{4k+1}, h_{4k+2}, h_{4k+3}\}$are used as a input to compute a group of intermediate signal called sum bits $\{S_{4k}, S_{4k+1}, S_{4k+2}, S_{4k+3}\}$ using the logic expression

$$s_{4k} = h_{4k} \oplus c_{4k-1} \tag{5.1}$$
$$c_{4k+1} = h_{4k+1} \oplus c_{4k} \tag{5.2}$$
$$c_{4k+2} = h_{4k+2} \oplus c_{4k+1} \tag{5.3}$$
$$c_{4k+3} = h_{4k+3} \oplus c_{4k+2} \tag{5.4}$$

When inside carrying bits are calculated to the group carry bit, there is a carry rippling. The region carries rippling, the PPA-based sum generation of algorithm-2 requires 6 extra gates delay as compared to the algorithm-1. Generally, Algorithm-2 actions in carrying computation unit save area on the other hand raise data addiction in final sum block.

Algorithm-2 offers selection logic of final sum computation unit, which are reduce the data dependency. Depend on this selection logic, two head of sum bits $\{S^0_{4k}, S^0_{4k+1}, S^0_{4k+2}, S^0_{4k+3}\}$ and $\{S^1_{4k}, S^1_{4k+1}, S^1_{4k+2}, S^1_{4k+3}\}$ , comparable to the certain group carry bit (Ck = 0) and (Ck = 1) are generated. In the group carry bit $C_k$ is used by selection logic. The selection logic to prefer one head available of the two heads for the final sum bits $\{S_{4k}, S_{4k+1}, S_{4k+2}, S_{4k+3}\}$ . The logical interpretation of certain selection logic and sum bits are given as

$$S_{4k} = S^0_{4k}\overline{C}_{k-1} + S^1_{4k}C_{k-1} \tag{6.1}$$
$$S_{4k+1} = S^0_{4k+1}\overline{C}_{k-1} + S^1_{4k+1}C_{k-1} \tag{6.2}$$
$$S_{4k+2} = S^0_{4k+2}\overline{C}_{k-1} + S^1_{4k+2}C_{k-1} \tag{6.3}$$
$$S_{4k+3} = S^0_{4k+3}\overline{C}_{k-1} + S^1_{4k+3}C_{k-1} \tag{6.4}$$

Where,

$$s^0_{4k} = h_{4k} \tag{7.1}$$
$$s^0_{4k+1} = h_{4k+1} \oplus g_{4k} \qquad s^1_{4k+1} = h_{4k+1} \oplus p_{4k} \tag{7.2}$$
$$s^0_{4k+2} = h_{4k+2} \oplus gg_{4k+1} \qquad s^1_{4k+2} = h_{4k+2} \oplus pp_{4k+1} \tag{7.3}$$
$$s^0_{4k+3} = h_{4k+3} \oplus gg_{4k+2} \qquad s^1_{4k+3} = h_{4k+3} \oplus pp_{4k+2} \tag{7.4}$$

and

$$gg_{4k+1} = g_{4k+1} + p_{4k+1}.g_{4k} \tag{8.1}$$
$$pp_{4k+1} = g_{4k+1} + p_{4k+1}.p_{4k} \tag{8.2}$$
$$gg_{4k+2} = g_{4k+2} + p_{4k+2}.gg_{4k+1} \tag{8.3}$$
$$pp_{4k+2} = g_{4k+2} + p_{4k+2}.pp_{4k+1} \tag{8.4}$$

## MODIFIED LOGIC FORMULATION OF PARALLEL PREFIX ADDER

To decrease data addiction of sum generation block depends on the group sum selection unit which is based on modified parallel prefix adder algorithm however requires some additional logic expressions. The group-sum-selection logic in the sum generation unit reduces by 6 gates critical path. The overhead complication of (3n/2) AND, (3n/4) NOT, (3n/4) OR, and (3n/4) XOR gates as n-bit parallel prefix adder. In group sum selection logic the overhead complexity comes out larger than the extent of delay saved.

## MODIFIED STRUCTURE OF PARALLEL PRE-PROCESSING BLOCK

The parallel preprocessing unit consists of n logic cells. These logic cells depend on two-input XOR gates, two inputs, AND gate, two OR gates. The hardware dependency of n-bit pre-processing of 6n gate costs, whereas one gate cost is equivalent to four transistors. Every logic cell can be implemented by using the logic diagram of the pre-processing unit, which is shown in Figure 6. Figure-6 shown the modified pre-processing block, which is based on the costs of the gate is 5n. So one gate saving in every logic cell when implemented pre-processing stage using this structure.

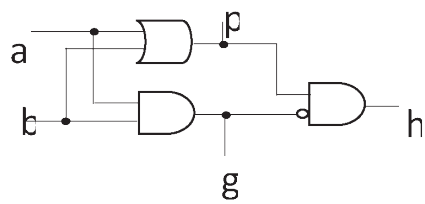


**Figure 6.** Modified logic diagram of the pre-processing unit.

In equation (6.2), putting k=1

$$S_{4k+1} = S^0_{4k+1} C_{k-1} + S^1_{4k+1} C_{k-1}$$

$$\begin{aligned} S_5 = S^0_5 \overline{C}_0 + S^1_5 C_0 &= (h_5 \oplus g_4).\overline{C_0} + (h_5 \oplus p_4).C_0 \\ &= (h_5.\overline{g_4} + \overline{h_5}.g_4).\overline{C_0} + (h_5.\overline{p_4} + \overline{h_5}.p_4).C_0 \\ &= \overline{h_5}.(g_4.\overline{C_0} + p_4 C_0) + h_5(\overline{g_4}.\overline{C_0} + \overline{p_4} C_0) \\ &= \overline{h_5}.(g_4.\overline{C_0} + p_4 C_0) + h_5(\overline{g_4.\overline{C_0} + p_4 C_0}) \\ &= h_5 \oplus (g_4.\overline{C_0} + p_4 C_0) \end{aligned} \quad (9)$$

Lemma 1:- $\overline{A.C} + \overline{B}.C = \overline{A.\overline{C} + B.C}$

Proof:-

$$C + \overline{C} = 1, C.\overline{C} = 1, A + 1 = 1, \overline{A + B} = \overline{A}.\overline{B}$$

$$\overline{A.C} + \overline{B}.C = \overline{\overline{A.C} + \overline{B}.C} = \overline{\overline{(A.C)}(\overline{B}.C)} = \overline{(\overline{\overline{A}} + \overline{C})(\overline{B} + \overline{C})}$$

$$= \overline{(A + C)(B + \overline{C})} = \overline{A.B + A\overline{C} + BC + C\overline{C}}$$

$$= \overline{A.B.1 + A\overline{C} + BC + 0} = \overline{A.B.(C + \overline{C}) + A\overline{C} + BC}$$

$$= \overline{A.B.C + AB\overline{C} + BC + A\overline{C}}$$

$$= \overline{B.C(A + 1) + A\overline{C}(B + 1)}$$

$$= \overline{B.C.1 + A\overline{C}.1}$$

$$= \overline{A.\overline{C} + B.C}$$

$$S_{4k+1} = h_{4k+1} \oplus (C^0_{4k} \overline{C}_{k-1} + C^1_{4k} C_{k-1}) \quad (10)$$

Equation (9) is rewritten as

Where $C^0_{4k} = g_{4k}$ and $C^1_{4k} = p_{4k}$, $C^0_{4k}$ and $C^1_{4k}$ show the certain carry bits corresponding to the group carry bit. The final carry bit $C_k = 0$, $C_k = 1$ can be chosen against the specified( $C^0_{4k}, C^1_{4k}$) using selection logic.

Equation (10) can be expressed as

$$S_{4k+1} = h_{4k+1} \oplus (g_{4k} + p_{4k} C_{k-1}) \tag{11}$$

Lemma 2:- $A\overline{C} + B.C = A + B.C$

Proof:- $C + \overline{C} = 1, A + 1 = 1, A = AB$ and $A\overline{C} + C = (A + C).(C + \overline{C})$ then:

$$A\overline{C} + BC = AB\overline{C} + BC = B(A\overline{C} + C)$$
$$= B.(A + C).(C + \overline{C})$$
$$= B.(A + C).1 = B.(A + C)$$
$$= AB + BC = A + BC$$

This structure of parallel prefix adder gives, a fast processing speed and regular structure. The proposed work is based on a carry computation unit. It has also less area as compared to the Kogge Stone adder (B. K. Patel et al.,2018). Our proposed architecture modified the PPA modulo (2n +1) adder shown in Figure-7. The processing speed is improved. The area is less in comparison to the Kogge Stone adder (H.T. Vergos et al., 2012). The proposed design focuses on the carry computation unit. In the carry-computation unit, the generate carry terms use to generate and propagate signals. The proposed work reduces the generation and propagation terms as compared to the Kogge Stone adder.

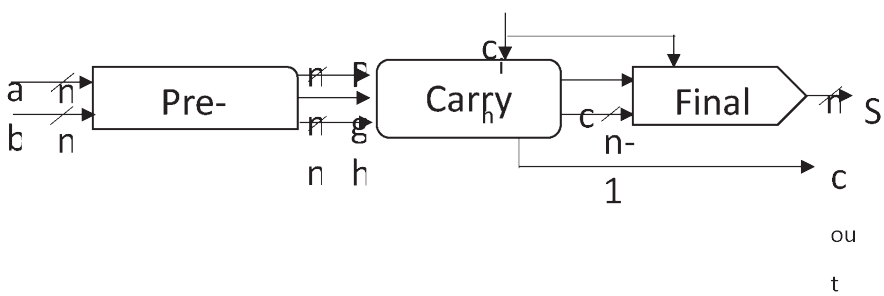


**Figure 7.** Modified parallel prefix structure of modulo $2^n$+1 adder.

The set of partial products is obtained by using the partial product reduction scheme as given in the Wallace tree method[2]. In this architecture, the Inverted End Around carrying (IEAC) represents the complement form of carrying output. It shows that the derived tree multiplier has a regular structure. Figure-8 represents the modulo (2n +1) multiplier. In this multiplier addition of diminished-1 number use modulo (2n +1) adder. The block 0,1,2,3,…..12 represent full adder.

## PROPOSED STRUCTURE OF DIMINISHED-ONE MODULO $2^n$+1 ADDER

The proposed modulo adder design is more efficient than the reported design (H.T. Vergos et al.,2012). The proposed Parallel Prefix adder design can be improved in a successive way than the reported modulo adder. Still, we find that the sparse carry computation algorithm of diminished-one adder can be expressed in a modified version to bring more regular compare to existing. The proposed modulo (2n +1) adder design is shown in figure-8 for n=16. The structure of solid circle represents prefix operation and it contains 2 AND gates and 1 OR gate. Hence the proposed structure

consists of n number of two-input XOR gates, number of two-input AND gates, number of two-input OR gates, and n number of NOT gates. The proposed carry computation block involves {K(L+3)} nodes, where K represents costs of NOT gates. Here k = n/4 and L=log2(n/4). The existing parallel prefix design based on algorithm-I and algorithm-II consumes extra n and (7n/4) XOR cells compare to the proposed parallel prefix adder design. The proposed parallel prefix design and existing PPA design are based on algorithm-I and algorithm-II are calculated in terms of the number of gates and number of transistor costs. Whereas one gate cost is equivalent to four transistors. The theoretical comparison of the area-based table is shown in Table-I. Table-I represents the proposed design area saving increases when increasing the bit size. It achieves in terms of transistor costs approximate 24% and 22% less area compared to the existing design (size n=32) which is based on Algorithm-I and Algorithm-II (H.T.Vergos et al.,2018, J. Peng et al.,2018, S. Mathew et al., 2003, B. Patel et al.,2018).

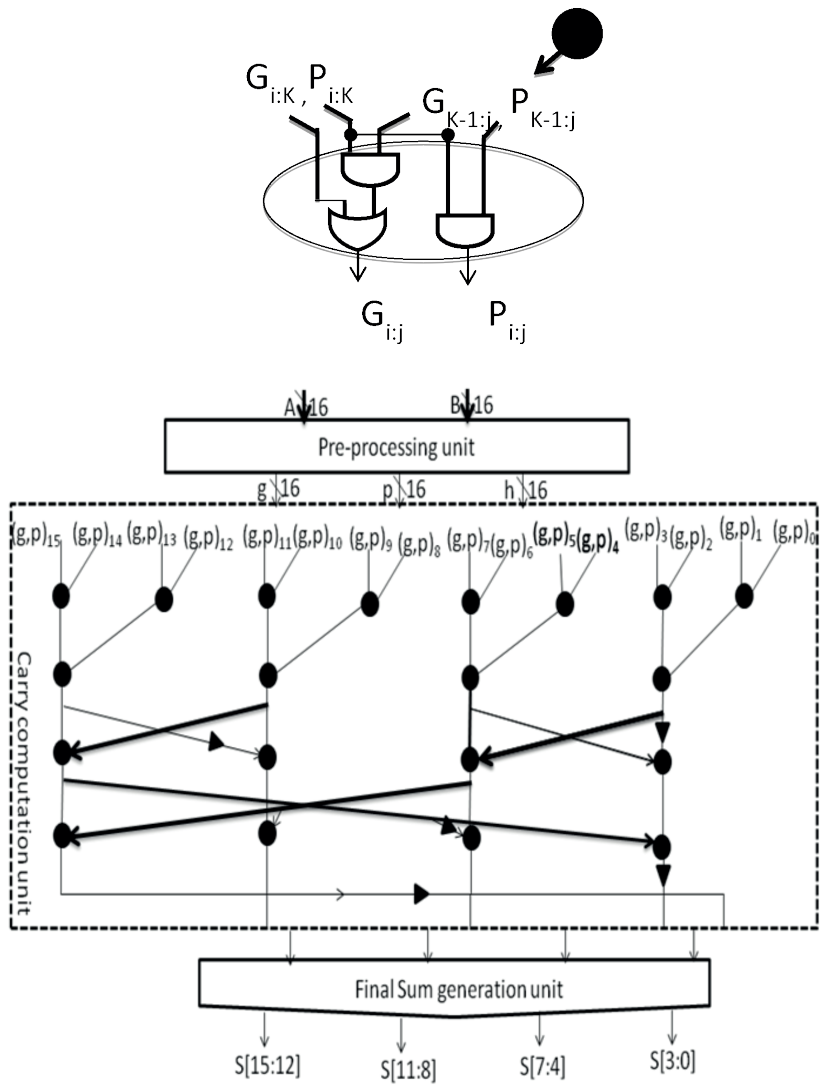


**Figure 8.** Proposed diminished-1 modulo 2n+1 adder for n=16.

**Table 1.** Hardware analysis in terms of gates and Transistors

| **Design** | **Based on Algorithm-I** | | | **Based on Algorithm-II** | | | **Based on Proposed Scheme** | | |
|---|---|---|---|---|---|---|---|---|---|
| **No. of bits** | 8 | 16 | 32 | 8 | 16 | 32 | 8 | 16 | 32 |
| **No.of gates** | 136 | 331 | 805 | 171 | 356 | 751 | 140 | 287 | 597 |
| **No. of Transisters** | 539 | 1326 | 3190 | 691 | 1435 | 2995 | 580 | 1190 | 2410 |

## SYNTHESIS RESULTS

Every VLSI design is coded using VHDL language and functional verification execute through test bench waveform perform the ISim simulator of Xillinx ISE 14.2 EDA tool. Each VLSI design is synthesized using Synopsys design compile TSMC 65nm CMOS to calculate the area, consumption of power(using virtual clock) at 0.88V supply voltage, and delay. Delay calculating in form of minimum clock period(MCP). The recorded value calculated by Design Compiler is shown in Table-II for comparison to the existing value. Table II represents the existing design result compare to the proposed design result consumes more area.

**Table 3.** Synthesis result of parallel prefix adder design

| Design | Based on Algorithm-I | | | Based on Algorithm-II | | | Based on Proposed Scheme | | |
|---|---|---|---|---|---|---|---|---|---|
| No. of bits | 8 | 16 | 32 | 8 | 16 | 32 | 8 | 16 | 32 |
| MCP(nsec.) | 0.36 | 0.46 | 0.54 | 0.36 | 0.46 | 0.54 | 0.38 | 0.48 | 0.55 |
| Area($\mu m^2$) | 197 | 436 | 1209 | 271 | 568 | 1201 | 201 | 422 | 892 |
| Power(μw) | 93 | 184 | 395 | 120 | 225 | 405 | 91 | 158 | 289 |

The proposed design for bit size 32 consumes 25.8% less area and 28.1% less power and hardly 3.4% more minimum clock period compare to the existing design based on algorithm-I. Equivalently, the existing PPA design based on algorithm-II (bit size 32) compare to the proposed Parallel Prefix Adder design consumes 25.8% more area, 28.2% more power, and hardly 3.4% less Minimum Clock Period. The proposed design save area and power provide mainly saving in XOR unit. The proposed design provides more area saving for large bit sizes. The proposed parallel prefix adder design to calculate efficiency for comparison in terms of ADP and EDP(ADP stands for area delay product and EDP stands for energy-delay product) from the simulation result shown in table-II. In figure-6, it is mentioned that the proposed Parallel Prefix Adder design requires less ADP and EDP compare to the reported Parallel Prefix Adder designs based on Algorithm-1 and Algorithm-2 for bit size 32. The proposed PPA design giving a 22.6% in ADP saving and 24.2% in EDP compared to the PPA designs based on Algorithm-2 (H.T.Vergos et al.,2018, Patel R.A. et al.,2007, S. Mathew et al., 2003, L.S. Didier et al.,2013, B. Patel et al.,2018) which is leading with the existing Parallel Prefix Adder designs.

## CONCLUSION

In this paper, a parallel prefix adder proposed which is depends on the group-carry selection logic. In the group sum selection logic by using carry rippling is used to avoid redundant logic operations in the carry computation unit. The pre-processing entity shows the reduced form to shorten logic resources. The proposed design represents an efficient PPA design. For bit size 32. The proposed PPA design consumes 24.8% less area and 26.4% less power. The proposed Parallel Prefix Adder design using reduced carry computation algorithm for diminished-1 modulo $2_n$+1 adder structure has been presented. The proposed diminished-1 modulo ($2_n$ +1) adder design saves 22.4% in ADP and 24.2% in EDP than the existing modulo adder structure. This design of the modulo adder can be applied in FIR filter design and cloud computing. The proposed work is further to be extended to analyze the diminished-1 and normal encoding impact

on an RNS application via FIR filters and discrete cosine transform. The proposed modulo adder design can be efficient implementation for the various modulo arithmetic circuits for the residue number system.

## REFERENCES

**R. P. Brent & H. T. Kung,1982.** A regular layout for parallel adders, IEEE trans, computers, Vol.C-31, pp. 260-264.

**R. Ladner & M. Fischer, 1980.** Parallel prefix computation, Journal of ACM.La.Jolla CA, Vol.27, pp.831-838.

**Kogge P, Stone H, 1973.** A parallel algorithm for the efficient solution of a general class Recurrence relation, IEEE Trans. Computers, Vol.C-22, pp 786-793.

**S. Knowles,1999.** A Family of Adders, Proc. 14th IEEE Symp. Computer Arithmetic, pp. 30-34.

**A. A. Hiasat, 1992.** A memoryless mod(2n+1) Residue Multiplier, Electronics Letters, vol.28,no.3,pp.314-315.

**R. Zimmerman,1999.** Efficient VLSI Implementation of Modulo (2n + 1) Addition and Multiplication, in Proc. 14th IEEE Symp. Computer Arithmetic, pp. 158-167.

**P. V. A. Mohan,2002.** Residue Number Systems: Algorithms and Architectures, Norwell, MA: Kluwer.

**H.T. Vergos, C. Efstathiou & D. Nikolos,2002.** Diminished-One Modulo (2n + 1) Adder Design, IEEE Trans. on Computers, vol. 51, no. 12, pp. 1389-1399.

**Y. Wang, S. Song, M. Aboulhamid and H. Shen,2002.** Adder Based Residue to Binary Number Converters for {(2n -1) 2n , (2n +1)}, IEEE Trans. Signal Processing, vol. 50, no. 7, pp. 1772- 1779.

**Hiasat A A.,2002.** High-speed and reduced-area modular adder structures for RNS. IEEE Trans on Computers, 51(1): 84–92.

**H.T. Vergos,2010.** A Family of Area-Time Efficient Modulo (2n + 1) Adder, IEEE Annual Symposium on VLSI, pp. 442-443.

**H.T. Vergos, C.Efstathiou,2009.**Efficient modulo 2n+1 adder architectures Integration the VLSI Journal, vol. 42 pp. 149-157, 2009.

**L.C.E.,H.T. Vergos et al.,2005.** Efficient Diminished-1 modulo 2n+1 Multipliers, IEEE Trans. On computers vol. 54 no. 4.

**H.T. Vergos et al.,2012.** On modulo (2n+1) Adder design, IEEE Trans on Computers, vol. 61, no.2 pp. 173-186.

**Shang Ma, Jian Hao Hu & Chen Hao,2013.** A novel modulo 2n – 2k – 1 adder for residue number system, IEEE Transactions On Circuits And Systems-I: Regular Papers. vol. 60, no. 11, pp. 2962–2972.

**S. Mathew et al., 2003.** A 4-GHz 130-nm Address Generation Unit with 32-bit Sparse Tree Adder Core, in the journal of Circuits System signal Processing, Springer, vol.36, no.3 pp. 1224-1246, March 2017.

**M. Bayoumi, G. Jullien, and W. Miller,1987.** A VLSI implementation of residue adders, IEEE Trans. Circuits Syst., vol. CAS-34, no.3, pp. 284–288.

**Beerendra K. Patel and J. Kanungo,2018.** Diminished-1 multiplier using modulo 2^n+1 adder, International journal of engineering and Technology, vol.4, no 4.20, DOI:10.14419/ijet.v7i4.20.22117.

**Patel R. A, Benaissa M, Boussakta S.2007.** Fast modulo addition: a new class of adder for RNS, IEEE Trans on Computers,56(4): 572-576.

**L.S. Didier and L. Jaulmes,2013.** Fast modulo (2n - 1) and (2n + 1) adder using carry-chain on FPGA, in Proc. IEEE Asilomar Symposium on signal, system, and computers, pp. 1155-1159.

**Patel R A, Benaissa M, Powel N, et al.,2004.** ELMMA: A New Low Power High-Speed Adder for RNS. Proc. In: IEEE Workshop on Signal Processing Systems. Austin:, 95-100.

**Patel, R. A. Benaissa, M. Powell, N, et al.,2007.** Novel Power-Delay-Area-Efficient Approach to Generic Modular Addition.IEEE Trans on Circuits and Systems I: Regular Papers, 54(6):1279-1292.

**M. Dugdale,1992.** VLSI implementation of residue adders based on binary adders, IEEE Trans. Circuits Syst. II: Analog Digit. Signal Process., vol. 39, no. 5, pp. 325–329.

**J. Peng, S. Sun, Vikram K. Narayana,Volker J. Sorger and Tarek el-Ghazawi,2018.** Residue number system arithmetic based on integrated nanophotonics, Optical Society of America(Optics Letters), Vol. 43, No. 9, DOI:10.1364/OL.43.002026.