Circuits and Systems, 2016, 7, 4024-4035
http://www.scirp.org/journal/cs
ISSN Online: 2153-1293
ISSN Print: 2153-1285

Area Efficient Sparse Modulo $2^n - 3$ Adder

Ritesh Kumar Jaiswal, Chatla Naveen Kumar, Ram Awadh Mishra

Department of Electronics and Communication Engineering, Motilal Nehru National Institute of Technology, Allahabad, India
Email: rel1406@mnnit.ac.in, chnaveen43@gmail.com, ramishra@mnnit.ac.in

How to cite this paper: Jaiswal, R.K., Kumar, C.N. and Mishra, R.A. (2016) Area Efficient Sparse Modulo $2^n - 3$ Adder. *Circuits and Systems*, **7**, 4024-4035.
http://dx.doi.org/10.4236/cs.2016.712333

Received: April 23, 2016
Accepted: May 23, 2016
Published: October 31, 2016

Abstract

This paper presents area efficient architecture of modulo $2^n - 3$ adder. Modulo adder is one of the main components for the implementation of residue number system (RNS) based applications. The proposed modulo $2^n - 3$ adder is implemented effectively, which utilizes parallel prefix and sparse concepts. The carries of some bits are calculated with the help of sparse approach in $\log_2 n$ prefix levels. This scheme is implemented with the help of idempotency property of the parallel prefix carry operator and its consistency. Parallel prefix structure contributes to fast carry computation. This will reduce area as well as routing complexity efficiently. The presented adder has double representation of residues in {0, 1, and 2}. The proposed adder offers significant reduction in area as the number of bits increases.

Keywords

Residue Number System (RNS), Parallel Prefix Adder, End Around Carry (EAC), Sparse Adder

1. Introduction

Residue number system (RNS) is a classical and a non weighted number system [1]. RNS divides the given number into collection of small numbers, which significantly improves the speed of operation; the result is obtained by reverse conversion [2]. RNS has plenty of applications in different fields, e.g., digital signal processing (DSP) for filters, convolution, FFT transforms [3]-[7], cryptography [8], image processing for wavelet transforms [9]-[11], error detection and error correction [12], fault tolerance signal processing properties [13] and communication [14].

An RNS is specified by set of moduli $\{m_1, m_2, m_3, \cdots, m_k\}$, which are relatively prime to each other. An Integer A is converted into RNS as $A \xrightarrow{\text{RNS}} \langle a_1, a_2, a_3, \cdots, a_k \rangle$ here $a_k = (A) \bmod m_k$ *i.e.* the least non negative remainder of the division of A by m_k. The dynamic range is denoted by M, which is defined as a product of moduli set [1]. The residue number system also has a lot of applications in the field of arithmetic operations like addition, subtraction, multiplication [15]. The most widely used moduli set is $\{2^n - 1, 2^n, 2^n + 1\}$ [16]. To increase the dynamic range of RNS, the moduli set is increased further to $\{2^n \pm 1, 2^n \pm 3\}$ [17]. L. Kalampoukas in [18] has proposed a new design in the view of modularizing to generate and propagate a factor in place of conventional end around carry scheme (EAC). This adder has parallel prefix carry computation structure which reduces the number of stages, leading to optimize in the speed and area for $2^n - 1$ modulo addition. H. T. Vergos *et al.* [19] proposed a new architecture which eliminates double parallel-prefix computation problem and customizes modulo $2^n + 1$ addition. The design offers reduction in cell area, wiring complexity and power consumption in conjunction with high speed of operation with the concept of sparse modulo $2^n + 1$ adder which is based on the extension of eminent idempotency property of prefix operator. Latency compatible parallel prefix modulo $2^n - 3$ adder is presented in [20] to include extra modulus term. In this, design technique of [18] is extended and modified for the difficulties occurred in derivation of generate and propagates signals formula with variable-weight end around carries.

Main Contribution

Double representation for modulo ($2^n - 3$) *i.e.* (0, 1, and 2) is explained in [21] where ripple carry addition strategy is used. In this paper we propose a modulo $2^n - 3$ adder which uses the concept of parallel prefix sparse adder. Parallel prefix approach has better compatibility with modulo ($2^n - 1$). Sparse parallel prefix adder is endorsed for large word-lengths addition, curtails the wiring and area design without affecting the delay. The proposed adder has lesser area as compared to existing modulo $2^n - 3$ adder [20].

This paper is organized as follows: Section 2 describes basics of parallel prefix addition. In Section 3, modulo $2^n - 3$ adder is discussed. Section 4 explains about sparse concept for modulo $2^n - 3$ adder. Finally, unit gate area and unit gate delay are calculated in Section 5.

2. Basics of Parallel Prefix Adder

Parallel-Prefix adder (PPA) performs parallel addition which plays a key role in microprocessors, DSP, mobile devices and other high speed applications. Parallel-Prefix structure reduces logic complexity and delay thereby enhancing the performance in term of area and power dissipation. Let the two inputs are A, B described as $A = A_{n-1}A_{n-2}\cdots A_0$ and $B = B_{n-1}B_{n-2}B_{n-3}\cdots B_0$, addition of these two numbers are represented as $S = S_nS_{n-1}S_{n-2}S_{n-3}\cdots S_0$. The addition performed in PPA is computed in three steps. The first stage computes the carry generation (G_i), propagation (P_i) and half sum (H_i) bits given as.

$$\begin{cases} G_i = A_i \cdot B_i \\ P_i = A_i + B_i \\ H_i = A_i \oplus B_i \end{cases} \tag{1}$$

where $\cdot$, + and $\oplus$ symbols are used to represent the logical AND, OR, XOR operations. Second stage of network defines carry computation unit, where we use two different types of operators that are ● and ● The operation performed by these operators is as follows [22].

$$\left(G_{out}, P_{out}\right) = \left(G_2, P_2\right) \bullet \left(G_1, P_1\right) = \left(G_2 + P_2 G_1, P_2 P_1\right) \tag{2}$$

$$\left(G_{out}, P_{out}\right) = \left(G_2, P_2\right) \bullet \left(G_1\right) = \left(G_2 + P_2 G_1\right) \tag{3}$$

The equations that are useful for generation of carry network [23] are:

$$\begin{cases} G_{i:i} = G_i,\ P_{i:i} = P_i \\ G_{i:j} = G_{i:k} + P_{i:k} \cdot G_{k-1:j} \\ P_{i:j} = P_{i:k} \cdot P_{k-1:j} \end{cases} \tag{4}$$

Or

$$\left(G_{i:j}, P_{i:j}\right) = \left(G_i, P_i\right) \bullet \left(G_{i-1}, P_{i-1}\right) \bullet \left(G_{i-2}, P_{i-2}\right) \cdots \left(G_j, P_j\right) \tag{5}$$

In the above expression $C_{i+1} = G_{i:j}$

The third stage is an "xor" operation of half sum bits and previous carry to get the final sum.

$$S_i = H_i \oplus C_i \tag{6}$$

Figure 1(a) and **Figure 1(b)** represent 8 bit Ladner Fischer and Kogge Stone structure of PPA respectively. **Figure 1(c)** represents the basic cells that are used in the construction of PPA.

For the design of large word length adders the concept of sparse is used [24]. In sparse PPA, instead of generating carry for every bit, it generates the carry for every k^{th} bit therefore it is called sparse-k parallel prefix adder. **Figure 2(a)** represents a simple 16-bit sparse-4 PPA as shown below.

Figure 2(b) shows carry select adder block which is used in sparse-4 PPA. This computes two sets of sum assuming carry equal to one and zero, select the resultant sum based on the carry which come from prefix network. By applying carry select adder in sparse PPA, routing problem is eliminated and area decreases effectively.

3. Modulo $2^n - 3$ Adder

The generalized formula for modulo $2^n - 3$ adder is described as [20]:

$$(A+B) \text{modulo } 2^n - 3 = \begin{cases} A+B & \text{if } A+B < 2^n \\ A+B+3 & \text{if } A+B \geq 2^n \end{cases} \tag{7}$$

The above expression for modulo $2^n - 3$ adder has double representation for {0, 1 and 2} with the last three numbers that are $2^n - 3$, $2^n - 2$, $2^n - 1$.

Unlike the modulo $2^n - 1$ adder, here we have to add the end around carry to the position 0 as well as position 1, this creates problem in implementation of the modulo PPA structure during addition. The two inputs and the EAC for position zero and position one [20] are taken as follows:

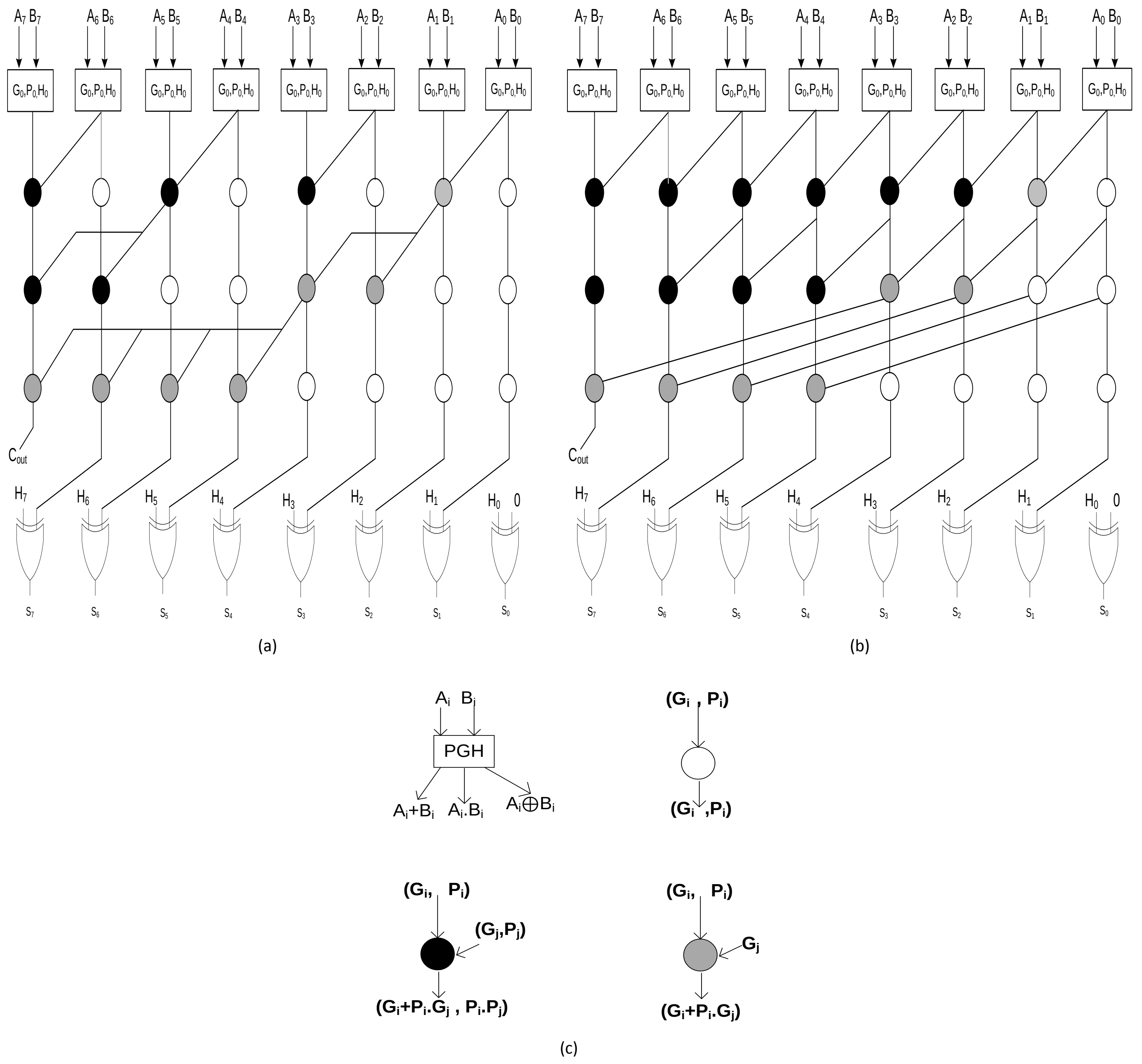
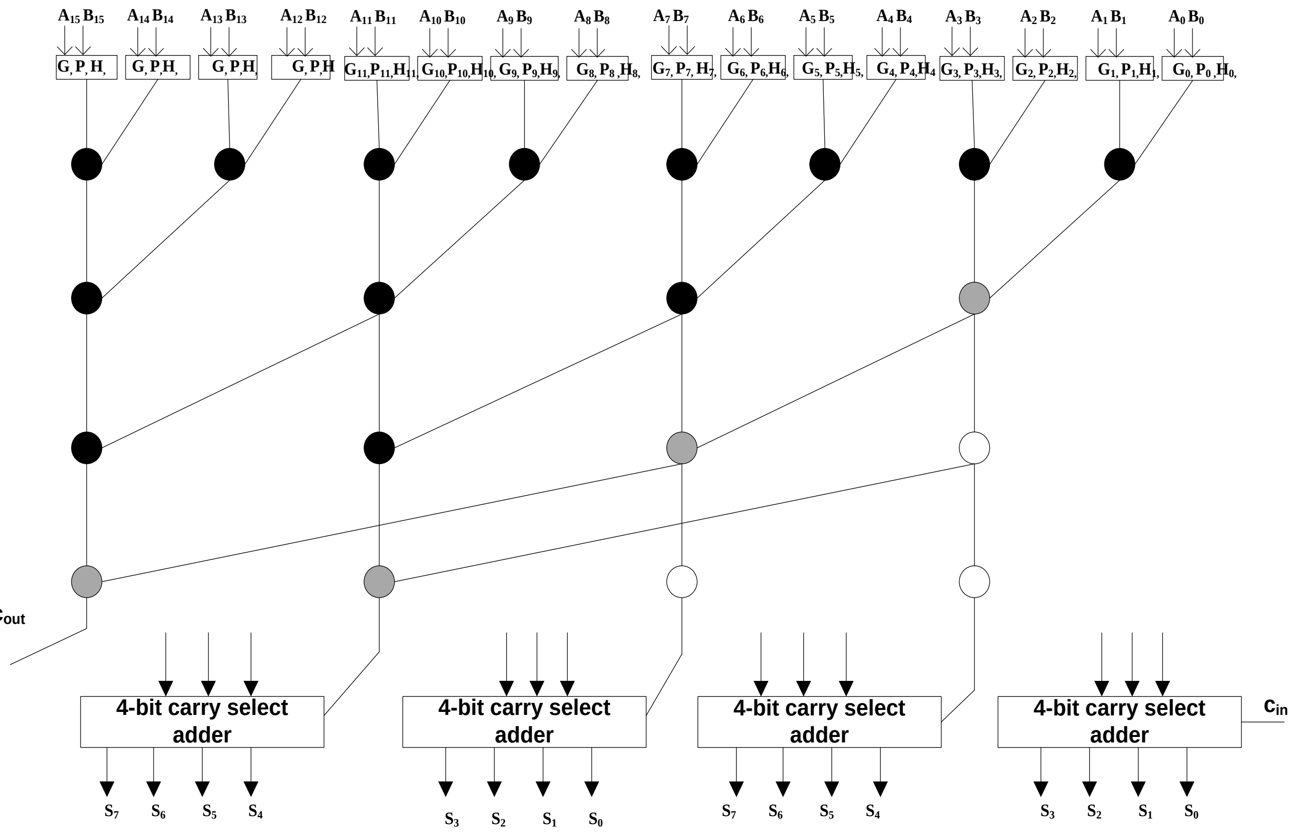


Figure 1. 8-bit parallel prefix adder. (a) Ladner FISCHER [23], (b) Kogge Stone [22], (c) The basic cells used in PPA.

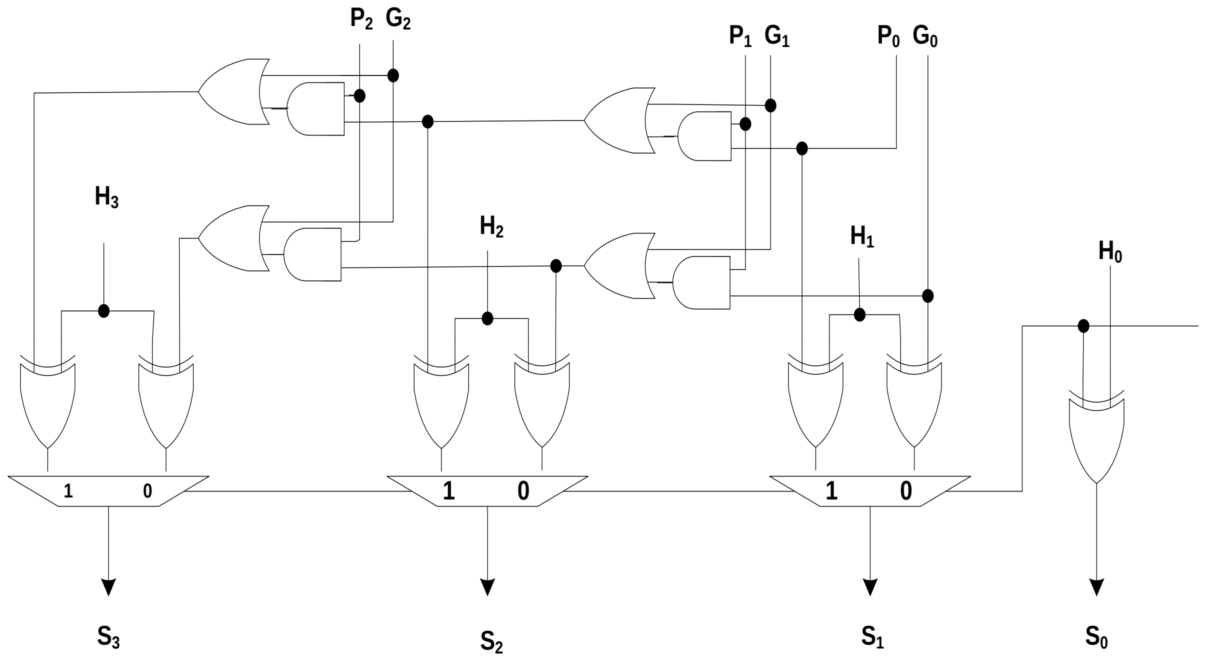
$$\begin{array}{cccccc} A_{n-1} & A_{n-2} & \cdots & A_2 & A_1 & A_0 \\ B_{n-1} & B_{n-2} & \cdots & B_2 & B_1 & B_0 \\ & & & & e & e \end{array}$$

Figure 3 describes that the carry generated in position zero enters in to next bit that is position one which already contains EAC. In worst case the carry bypasses from position two to next position. This problem can be eliminated by using carry save preprocessing stage [20] as shown in Figure 4.

Where u_i is the half adder sum output of A_i and B_i, v_{i+1} is the half adder carry



(a)



(b)

Figure 2. (a) 16 bit sparse-4 parallel prefix adder, (b) carry select adder which is used in Figure 2(a).

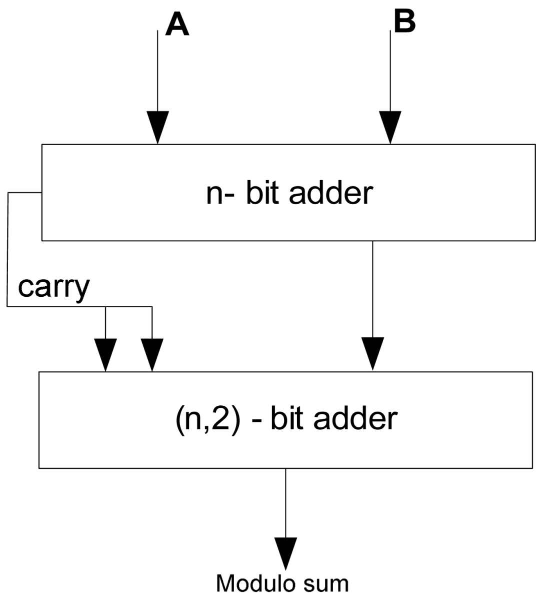
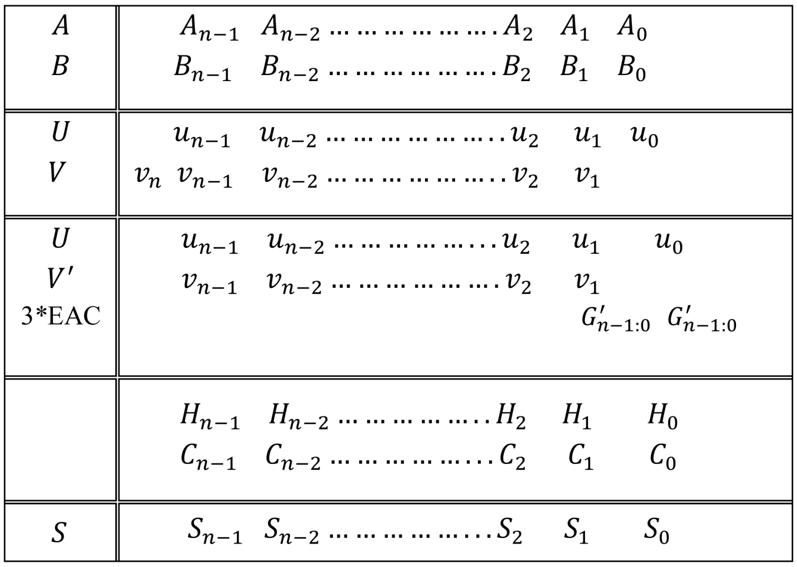


Figure 3. Two stage modulo $2^n - 3$ adder [21].



A		A_{n-1}	A_{n-2}	……………..	A_2	A_1	A_0
B		B_{n-1}	B_{n-2}	……………..	B_2	B_1	B_0
U		u_{n-1}	u_{n-2}	………………..	u_2	u_1	u_0
V	v_n	v_{n-1}	v_{n-2}	………………..	v_2	v_1	
U		u_{n-1}	u_{n-2}	………………..	u_2	u_1	u_0
V'		v_{n-1}	v_{n-2}	……………..	v_2	v_1	
3*EAC						$G'_{n-1:0}$	$G'_{n-1:0}$
		H_{n-1}	H_{n-2}	………………..	H_2	H_1	H_0
		C_{n-1}	C_{n-2}	………………..	C_2	C_1	C_0
S		S_{n-1}	S_{n-2}	………………..	S_2	S_1	S_0

Figure 4. Modulo $(2^n - 3)$ EAC addition using carry-save processing.

output of A_i and B_i. $G'_{n-1:0}$ represents end around carry for the next stage.

The alternative approach has been presented for modulo adder using PPA structures [20]. It had given that ith carry expression in the case of modulo $2^n - 3$ adder is as follows:

$$C_i = G_{i-1:0} + P'_{i-1:0} G'_{n-1:i}, \quad 2 \le i \le n-1 \tag{8}$$

where,

$$P'_{i-1:0} = P_{i-1:2} \cdot P'_1 \cdot P'_0, \;\; P_{i-1:2} = P_{i-1} \cdot P_{i-2} \cdots P_2$$

$$G'_{n-1:i} = G_{n-1:i} + v_n$$

$$C_1 = \overline{u_0} \cdot G'_{n-1:0}$$

The sum expression for bit position one is

$$S_1 = H_1 \oplus C_1 = H_1 \oplus \overline{u_0} \cdot G'_{n-1:0} = H_1 \overline{G'_{n-1:0}} + \overline{H_1 \oplus u_0} \cdot G'_{n-1:0} \quad (9)$$

From above expression, the carries can be calculated by propagate and generate bits. Figure 5(a) shows modulo $2^8 - 3$ regular parallel prefix (RPP) adder structure [20]. The RPP is differing with modulo $2^8 - 1$ having half carry-save stage for preprocessing, one bit in "zero" position before enforcing the EAC and two carries enter into the position "one" after EAC enforcement. Figure 5(b) represents modulo $2^8 - 3$ total parallel prefix (TPP) adder structure [20]. TPP is same as RPP. The only difference is that we have $c_1 = G'_{n-1:0}\overline{u_0}$ one gate more delay than other carries. The sum S_1 is implemented with the help of multiplexer taking $G'_{n-1:0}$ as selection line shown in Figure 5(b). For the rest of bits the sum expression calculated using exclusive-OR gate.

The delay offered by RPP adder structure is more as compared to TPP adder structure due to extra prefix level. The TPP structure has a disadvantage of routing complexity as well as excessive area problem as the bit length of adder increases.

4. Sparse Modulo 2^n – 3 Adder

In this segment, we proposed modulo 2^n – 3 adder by utilizing the concept of integer

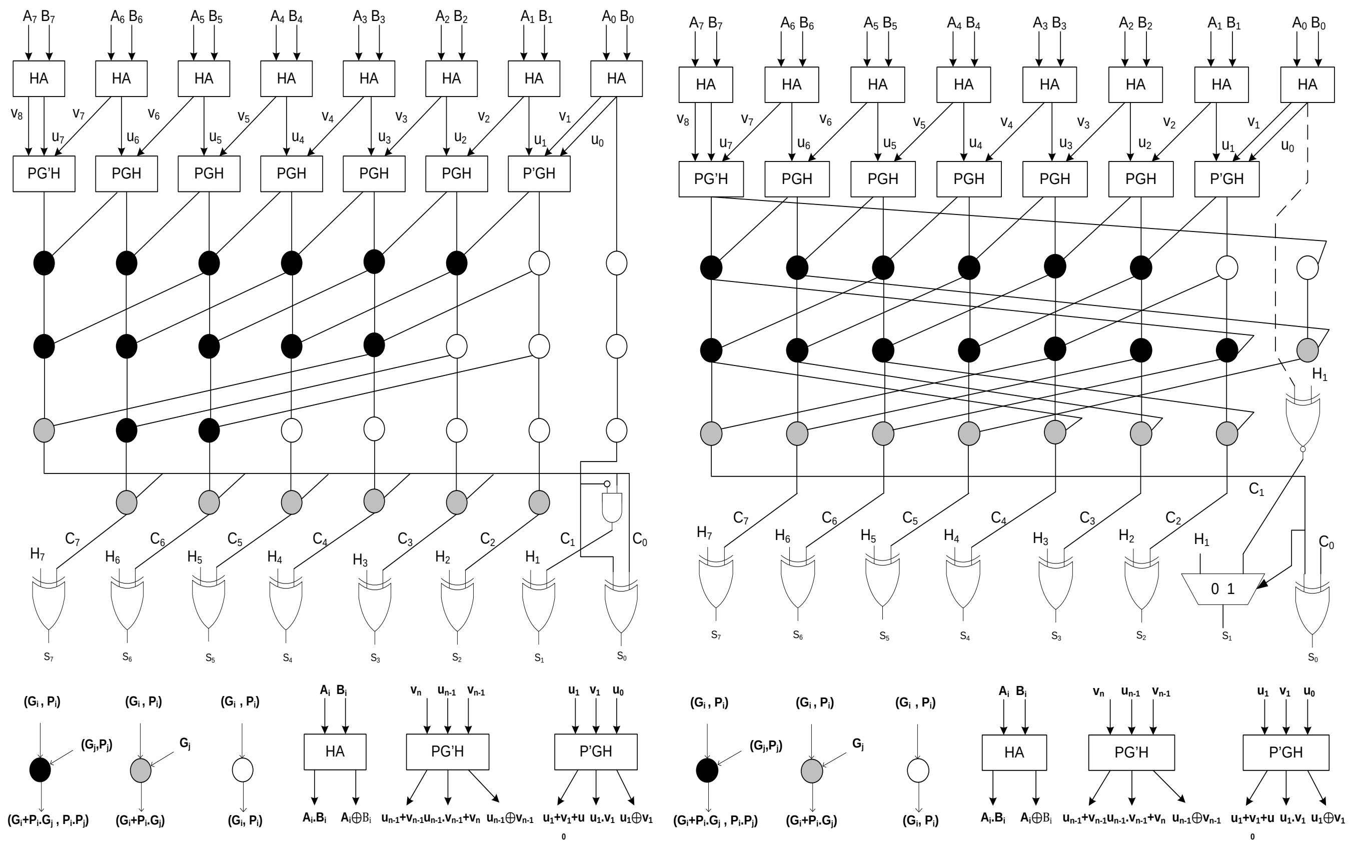


Figure 5. (a) Modulo 2^8-3 EAC adder, (b) recirculating EAC modulo 2^8-3 adder.

sparse-4 PPA in which the same carry select adder, used to implement sparse modulo $2^n - 3$ adder. In sparse-4, the carry is generated for every 4th bit. We are using carry select adder for modulo operation so we are required to show that the rest of carries are associated with available ones.

From the general carry expression given in Equation (8)

Let $n = 32$ bit, the carry expression C_{14} can be derived by available C_{12} written as:

$$C_{12} = G_{11:0} + P'_{11:0}G'_{31:12} \tag{10}$$

$$C_{14} = G_{13:0} + P'_{13:0}G'_{31:14} \tag{11}$$

We can also write it as:

$$C_{14} \leftrightarrow \left(G_{13:0}, P'_{13:0}\right) \bullet \left(G'_{31:14}, P_{31:14}\right) \tag{12}$$

This can also be expanded as:

$$C_{14} \leftrightarrow \left(G_{13:12}, P_{13:12}\right) \bullet C_{14} \leftrightarrow \left(G_{13:12}, P_{13:12}\right) \bullet \left(G'_{31:14}, P_{31:14}\right) \tag{13}$$

By the formula of Rearraging the redundant terms given in [23].

$$C_{14} \leftrightarrow \left(G_{13:12}, P_{13:12}\right) \bullet \left(G_{11:0}, P'_{11:0}\right) \bullet \left(G'_{31:14}, P_{31:14}\right) \bullet \left(G_{13:12}, P_{13:12}\right) \tag{14}$$

Finally it is expressed by,

$$C_{14} \leftrightarrow \left(G_{13:12}, P_{13:12}\right) \bullet \left(G_{11:0}, P'_{11:0}\right) \bullet \left(G'_{31:12}, P_{31:12}\right) \tag{15}$$

So

$$C_{14} \leftrightarrow \left(G_{13:12}, P_{13:12}\right) \bullet \left(\underbrace{G_{11:0} + P'_{11:0} \cdot G'_{31:12}}_{C_{12}}, P'_{31:0}\right) \tag{16}$$

At last, the carry expression C_{14} in terms of C_{12} is written as:

$$C_{14} = G_{13:12} + P'_{13:12}C_{12} \tag{17}$$

From above expression we conclude that this relation is quite similar to integer adder. Therefore we can directly use carry select block Figure 2(b) of sparse integer adder for performing modulo operation. But the main problem is the carry expression given in Equation (8) which is defined for $2 \le i \le n-1$. The carry equation for C_1 is quite different so the modification of carry select block is needed for first four bits of modulo 2^n – 3 adder, it is based on carry C_1 given in Equation (9).

Figure 6 is similar to carry select block of Figure 2(b) except at sum position S_1. The Figure 6 is used only for first four bits of sparse-4 modulo $2^n - 3$ adder. The remaining bits uses carry select block of Figure 2(b) for implementation of sparse modulo $(2^n - 3)$ adder.

This sparse-4 modulo $2^n - 3$ adder has double representation for {0,1,2} with $2^n - 3$, $2^n - 2$, $2^n - 1$, so there are six pairs of combinations in which two pairs has tendency to produce wrong addition result. The solution for this problem is explained in [20] and [21]; these explanations still exist for proposed adder.

Figure 7 represents the proposed 32 bit sparse modulo $2^n - 3$ Adder having lesser area than previously reported modulo adder.

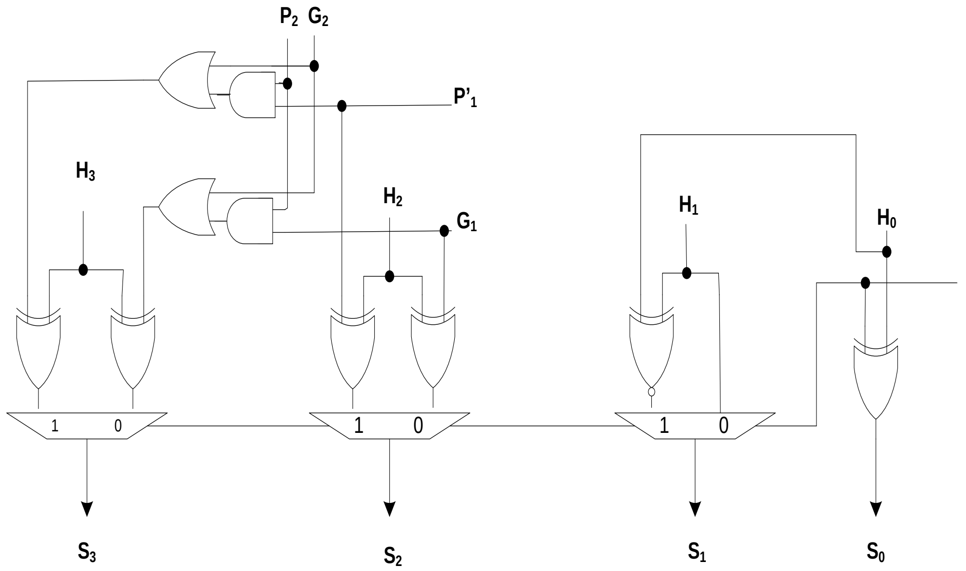


Figure 6. Carry select block for modulo $2^n - 3$ adder only for first 4 bits.

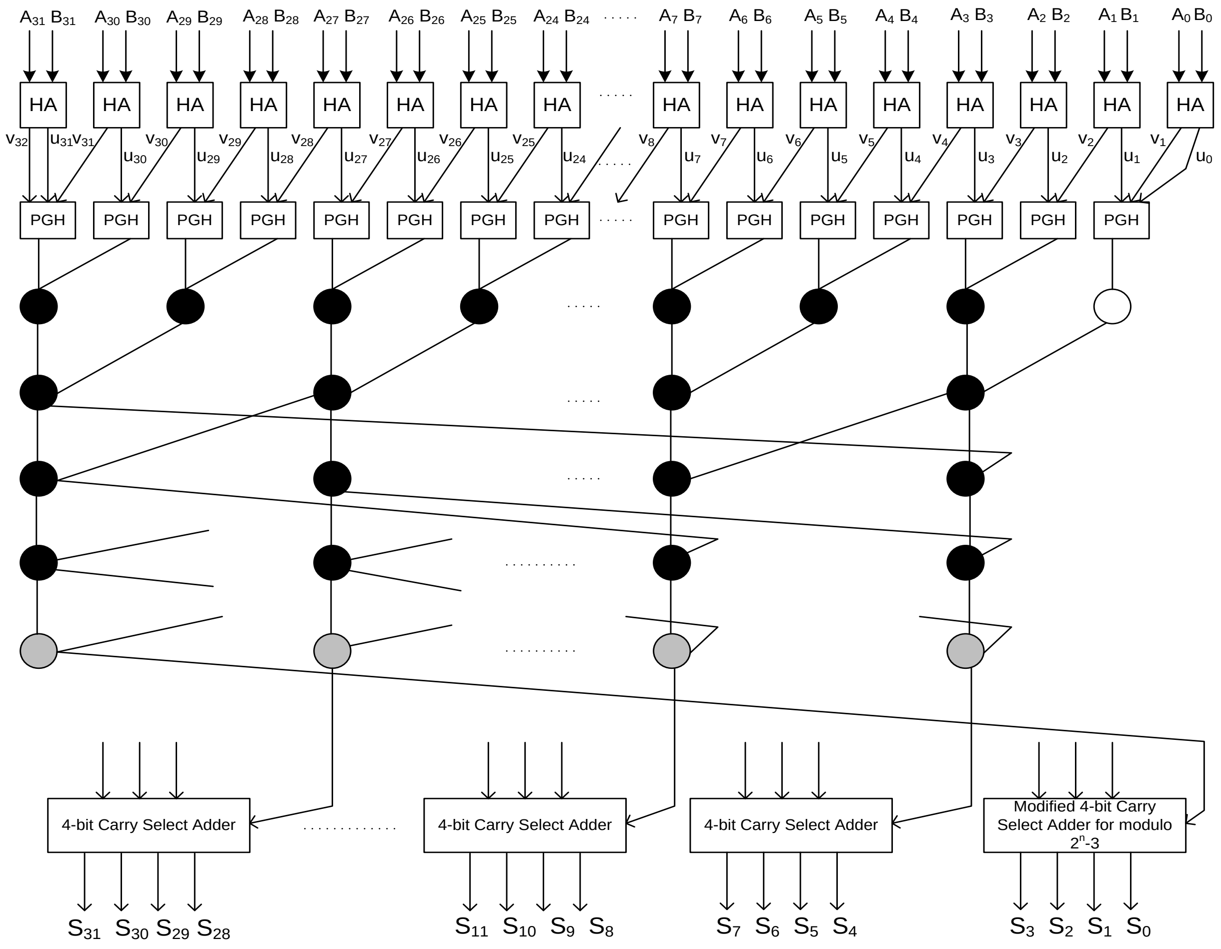


Figure 7. The proposed 32 bit sparse-4 modulo $2^n - 3$ parallel prefix adder using [17] architectures.

5. Performance Analysis and Comparison

The theoretical area and delay analysis is explained in terms of area (Δa) and delay (Δg) of basic 2-input gates. From the concept of unit gate model, basic 2-input AND, OR, NAND, NOR are assumed as single unit gate (Δa, Δg), whereas exclusive-OR & exclusive-NOR and assumed to be double unit gate (2Δa, 2Δg) [15]. The area and delay of Inverters and buffers are not taken into account in unit gate model.

The delay offered by proposed sparse modulo $2^n - 3$ adder is same as [20]. Table 1 shows the estimated gate delay and gate area of proposed adder as function of bit length n.

Table 2 shows the unit gate delays and unit gate areas for different values of n of proposed adder and also shows the percentage reduction in area in comparison with [20].

The percentage reduction in area increases as the number of bit length increases. We have also elaborated proposed work with HDL code written on Xilinx 14.7 and verified for correctness using simulation tests. Number of lookup table (LUTs) count is given in Table 3 for n = 8 which measures the area utilization for proposed adder.

6. Conclusion

In this paper, we have proposed an area efficient sparse modulo $2^n - 3$ adder which plays an important role in verity of computer applications. The efficiency in term of area of proposed adder is explained by using the concept of unit gate model. For different value of n (=8, 16, 32, 64), the percentage area reduction is (=2.3, 13.2, 21, 27.54)

Table 1. Adders unit gate area and delay estimations.

Adder	Delay (Δg)	Area (Δa)
[20]	$2\log(n)+4$	$3n\log(n)+8n-10$
Proposed	$2\log(n)+4$	$\frac{29n}{2}+\frac{3n\log(n)}{4}-11$

Table 2. Delay and area for different bit length.

Bits (n)	[20]		Proposed		Reduction %	
	Delay (Δg)	Area (Δa)	Delay (Δg)	Area (Δa)	Delay (Δg)	Area (Δa)
8	10	126	10	123	0	2.3
16	12	310	12	269	0	13.2
32	14	726	14	573	0	21.0
64	16	1654	16	1205	0	27.54

Table 3. LUT count for $n = 8$.

[20]	Proposed Sparse Adder	% Reduction in LUT Count
64	42	34

respectively with same delay. Simulation results show that the area of proposed adder has been reduced by 34% in term of LUT count for $n = 8$. Therefore, it is observed that, the presented modulo adder offers less area in performing the addition for larger word length input and also reduces the routing complexity in comparison with the previously reported adder.

References

[1] Ma, S., Hu, J.H., Zhang, L. and Xiang, L. (2008) An Efficient RNS Parity Checker for Moduli Set and Its Applications. *Science in China*, *Series F*: *Information Sciences*, **51**, 1563-1571.

[2] Garner, H.L. (1959) The Residue Number System IRE. *Transactions on Electronic Computers*, **8**, 140-147. http://dx.doi.org/10.1109/TEC.1959.5219515

[3] Garai, P. and Dutta, C.B. (2014) RNS Based Reconfigurable Processor for High Speed Signal Processing. *TENCON* 2014—2014 *IEEE Region* 10 *Conference*, Bangkok, 22-25 October 2014, 1-6.

[4] Di Claudio, E. D., Piazza, F. and Orlandi, G. (1995) Fast Combinatorial RNS Processors for DSP Applications. *IEEE Transactions on Computers*, **44**, 624-633. http://dx.doi.org/10.1109/12.381948

[5] Chang, C.H., Molahosseini, A.S., Zarandi, A.A.E. and Tay, T.F. (2015) Residue Number Systems: A New Paradigm to Datapath Optimization for Low-Power and High-Performance Digital Signal Processing Applications. *IEEE Circuits and Systems Magazine*, **15**, 26-44.

[6] Kurokawa, T., Payne, J. and Lee, S. (1980) Error Analysis of Recursive Digital Filters Implemented with Logarithmic Number Systems. *IEEE Transactions on Acoustics*, *Speech*, *and Signal Processing*, **28**, 706-715. http://dx.doi.org/10.1109/TASSP.1980.1163466

[7] Krishnan, R., Jullien, G. and Miller, W. (1985) Complex Digital Signal Processing Using Quadratic Residue Number Systems. *IEEE International Conference on Acoustics*, *Speech*, *and Signal Processing*, 764-767.

[8] Schinianakis, D.M., Kakarountas, A.P. and Stouraitis, T. (2006) A New Approach to Elliptic Curve Cryptography: An RNS Architecture. *MELECON* 2*006—*2*006 IEEE Mediterranean Electrotechnical Conference*, Malaga, 16-19 May 2006, 1241-1245.

[9] Safari, A., Niras, C.V. and Kong, Y. (2016) Power-Performance Enhancement of Two-Dimensional RNS-Based Dwt Image Processor Using Static Voltage Scaling. *Integration*, **53**, 145-156.

[10] Taleshmekaeil, D.K. and Mousavi, A. (2010) The Use of Residue Number System for Improving the Digital Image Processing. *IEEE* 10*th International Conference on Signal Processing Proceedings*, Beijing, 24-28 October 2010, 775-780.

[11] Taleshmekaeil, D.K., Mohamamdzadeh, H. and Mousavi, A. (2011) Using Residue Number System for Edge Detection in Digital Images Processing. 2011 *IEEE* 3*rd International Conference on Communication Software and Networks* (*ICCSN*), Xi'an, 27-29 May 2011, 249-253.

[12] Etzel, M. and Jenkins, W. (1980) Redundant Residue Number Systems for Error Detection and Correction in Digital Filters. *IEEE Transactions on Acoustics*, *Speech*, *and Signal Processing*, **28**, 538-545. http://dx.doi.org/10.1109/TASSP.1980.1163442

[13] Pontarelli, S., Cardarilli, G.C., Re, M. and Salsano, A. (2008) Totally Fault Tolerant RNS Based FIR Filters. 14*th IEEE International On-Line Testing Symposium*, Rhodes, 7-9 July

2008, 192-194. http://dx.doi.org/10.1109/iolts.2008.14

[14] Yang, L.-L. and Hanzo, L. (2002) A Residue Number System Based Parallel Communication Scheme Using Orthogonal Signaling. I. System Outline. *IEEE Transactions on Vehicular Technology*, **51**, 1534-1546. http://dx.doi.org/10.1109/TVT.2002.804850

[15] Zimmermann, R. (1999) Efficient VLSI Implementation of Modulo ($2^n \pm 1$) Addition and Multiplication. 14*th IEEE Symposium on Computer Arithmetic*, Adelaide, 14-16 April 1999, 158-167.

[16] Chang, C.H. and Low, J.Y.S. (2011) Simple Fast and Exact RNS Scaler for the Three-Moduli Set $2^n - 1$, 2^n, $2^n + 1$. *IEEE Transactions on Circuits and Systems I*: *Regular Papers*, **58**, 2686-2697. http://dx.doi.org/10.1109/TCSI.2011.2142950

[17] Ananda Mohan, P.V. (2008) New Reverse Converters for the Moduli Set $\{2^n - 3, 2^n - 1, 2^n + 1, 2^n + 3\}$. *International Journal of Electronics and Communications*, **62**, 643-658. http://dx.doi.org/10.1016/j.aeue.2007.08.008

[18] Kalampoukas, L., *et al*. (2000) High-Speed Parallel-Prefix Modulo $2^n - 1$ Adders. *IEEE Transactions on Computers*, **49**, 673-680.

[19] Vergos, H.T. and Dimitrakopoulos, G. (2012) On Modulo $2^n + 1$ Adder Design. *IEEE Transactions on Computers*, **61**, 173-186.

[20] Jaberipur, G. and Langroudi, S.H.F. (2015) ($4 + 2\log n)\Delta G$ Parallel Prefix Modulo-$2^n - 3$ Adder via Double Representation of Residues in [0, 2]. *IEEE Transactions on Circuits and Systems II*: *Express Briefs*, **62**, 583-587. http://dx.doi.org/10.1109/TCSII.2015.2407772

[21] Fatemi, H. and Jaberipur, G. (2014) Double {0, 1, 2} Representation Modulo-($2^n - 3$) Adders. *IWSSIP Proceedings*, Dubrovnik, 12-15 May 2014, 119-122.

[22] Kogge, P.M. and Stone, H.S. (1973) A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations. *IEEE Transactions on Computers*, **22**, 786-793. http://dx.doi.org/10.1109/TC.1973.5009159

[23] Ladner, R.E. and Fischer, M.J. (1980) Parallel Prefix Computation. *Journal of the ACM*, **27**, 831-838. http://dx.doi.org/10.1145/322217.322232

[24] Mathew, S., Anders, M., Krishnamurthy, R. and Borkar, S. (2002) A 4 GHz 130 nm Address Generation Unit with 32-Bit Sparse-Tree Adder Core. *VLSI Circuits Digest of Technical Papers*, Honolulu, 13-15 June 2002, 126-127.