I J C T A, 9(18) 2016, pp. 8935-8943

Low-Area Low-Power Parallel Prefix Adder Based on Modified Ling Equations

Rohan Pinto* **and Kumara Shama***

ABSTRACT

For the design and implementation of general purpose processors, addition plays an important role. Speed of an adder is one of the key factor that influences the performance of the system. Parallel prefix adders are one of the best solution for this, they are also suitable for VLSI implementation. Here in this paper parallel prefix adders based on modified ling equation have been proposed for 8-bit, 16-bit and 32-bit. Logic gates can be reduced using the proposed method that leads to reduction of area and power. The proposed adder is implemented using 90 nm and 180nm CMOS technology and compared with, other adopted adders. Synthesis results reveal that the proposed adder can achieve minimum area saving of 10% and power saving of 6%.

Keywords: Parallel prefix adder; Ling adder; VLSI implementation; Area-Power efficient.

I. INTRODUCTION

Data Path is the core element of every microprocessor and DSP. It is often the most critical circuit component when it comes to die area, power dissipation and delay. At the heart of the data path is the arithmetic unit which comprises of adders, multipliers and comparators. The fundamental operation found in the arithmetic unit is addition [1]. Besides two number addition, adders are also used in more complicated operations like division and multiplication. Therefore binary adder is the most important building block in Arithmetic and Logic Unit (ALU) and Address Generation Unit (AGU). It is likewise particularly imperative if implemented in hardware since it includes a costly carry propagation unit. Avital problem in VLSI design is efficiently implementing the binary adder in an integrated circuit.

Several adder structures have been presented in literature, with each one of them having its own advantages and disadvantages. They also have their own distinctive characteristics in terms of area, power and delay. For high speed addition parallel prefix adders are mostly preferred because they use parallel prefix trees to determing group generate and group propagate which inturn is used to compute carry and sum. Parallel prefix trees are very well suited for VLSI implementation since they depend on basic cells and keep up a regular connection between them.

Several parallel prefix trees have been proposed in the past.One of the earliest parallel prefix tree algorithm for adders was proposed by Sklansky [2]. Here a tree structure with minimum depth was developed for parallel, high speed addition. Kogge and Stone [3] parallel prefix tree had regular structure, minimal logic depth and uniform fan-out.This structure had shortcoming of larger gate count and complex interconnectionwhich led tohigh power dissipation. Ladner and Fischer [4]introduced a prefix graph with minimum depth. Brent and Kung [5] designed an area optimum prefix computation graph. It also had a regular structure, but slightly increased in latency as compared to other structures. Han and Carlson [6] proposed a hybrid adder by combining the structures proposed by Brent-Kung and Kogge-Stone. The design achieved a better trade-off between logic depths and interconnect complexity. Knowles [7] presented a

* Department of Electronics and Communication Engineering, Manipal Institute of Technology, Manipal, Karnataka, India, *E-mail: rohan.pinto@yahoo.co.in; shama.kumar@manipal.edu*

family of different adders with minimum logic depth. They had regular fan-out and were bounded at the extreme ends by Ladner-Fischer [4] graph and Kogge-Stone [5] graph. But, each structure had a different trade-offs between speed and area/power.

One of the most popular design for integer adders are carry look ahead adders (CLA). A variation to CLA is Ling adder [8] that simplifies carry computation and leads to faster structure. Parallel prefix structures for Ling carry computation exploit the straightforwardness of Ling equations. D Nikolos *et al.* [9] presented a method for computing Ling carries using parallel prefix tree network.As compared to conventional carry look ahead adder the parallel prefix tree structure required an extra OR gate. Dimitrakopoulos and Nikolos [10] spared one-logic level of execution which led to speedier implementation of the parallel-prefix addition. Juang [11] proposed a parallel prefix structure, which computed the real carries and saved one logic gate. Poornima and Kanchana [12] implemented a hybrid prefix adder based on modified Ling equations by combining the two tree networks. Ladner-Fischer structure was used for even-numbered bits and for odd-numbered bitsKogge-Stone structure was used.

In this paper an area-power efficient adder block based on the parallel-prefix computation technique has been proposed. Modified Ling equations have been used to compute intermediate generate and intermediate propagate signals. This structure computes the real carries based on Ling carries created by the lower order bits position thus saving a significant area. The proposed parallel prefix adder is implemented using 90nm and 180nm CMOS technology and compared with widely adopted prefix structures. To prove the efficiency of the proposed adder, area, power and delay parameters are computed for the proposed adder and few other adders and compared for 8-bit, 16-bit and 32-bit word sizes.

The remaining part of the paper is arranged as follows: Section 2 gives the fundamentals of parallel-prefix addition. Section 3 explains about Ling adders. Section 4 introduces the proposed area-power efficient parallel prefix adder. In section 5, simulation and synthesis results are discussed. Finally, conclusion is drawn in section 6.

II. BACKGROUND

Parallel prefix addition is done in three stages: pre-processing, prefix calculation and post-processing. Consider two numbers each of width m,$U = u_{m-1}u_{m-2} \ldots\ldots u_0$; $V = v_{m-1}v_{m-2} \ldots\ldots v_0$ to be added and $S = s_{m-1}s_{m-2} \ldots\ldots s_0$ be the sum [10].

2.1. Pre-Processing stage

In the first stage generate bits g_b, propagate bits p_b and half sum bits d_b are computed for all the bits $0 \le b \le m-1$ using the following equations,

$$g_b = u_b \; AND \; v_b$$
$$p_b = u_b \; OR \; v_b$$
$$d_b = u_b \; XOR \; v_b$$

2.2. Prefix calculation stage

Carry signal c_b is computed in this stage using generate bits g_b and propagate bits p_b. Any one type of prefix tree structure is used to calculate the carry. Parallel prefix structure with m inputs $u_1, u_2, \ldots\ldots u_m$ computes m outputs $v_1, v_2, \ldots\ldots v_m$ in parallel using an associative operator as follows:The operator $\odot$ associates generate and propagate bits as follows: $(g,p)\odot(g',p') = (g + p.g', p.p')$. The first output term $g + p.g'$ is group generate term and second term. p' is group propagate term. In a series of consecutive association of generate and propagate pairs (g, p), group generate and group propagate are computed for the bits spanning from i to j, that is, $(G_{i,j}, P_{i,j}) = (g_i, p_i) \odot (g_{i-1}, p_{i-1}), \ldots\ldots\ldots\ldots (g_{j+1}, p_{j+1}) \odot (g_j, p_j)$. The prefix operator $\odot$ is associative and idempotent, that is, $(g,p)\odot(g,p) = (g,p)$.

2.3. Post-Processing stage

This is the final stage where sum is computed using XOR operation which is given as

$$Sum = d_b \oplus c_{b-1}$$

III. LING ADDER

Ling [8] proposed a modification to carry look ahead equation to achieve a significant hardware saving. The technique is to remove one series transistor from the path of critical group generate at the cost of XOR gate in the sum computation [13]. The technique depends on calculating pseudo carry H_b instead of conventional carry c_b. The pseudo carry H_b can be computed quicker than the conventional carry c_b because it uses simpler boolean function [14]. H_b isexpressed as

$$H_b = g_b + g_{b-1} + p_{b-1}.g_{b-2} + p_{b-1}.p_{b-2}.g_{b-3} + p_{b-1}.p_{b-2} \cdots \cdots \cdots \cdots .p_1.g_0$$

Consider for example the carry for the 4th bit

$$c_{04} = g_{04} + p_{04}.g_{03} + p_{04}.p_{03}.g_{02} + p_{04}.p_{03}.p_{02}.g_{01} + p_{04}.p_{03}.p_{02}.p_{01}.g_{00}$$
$$H_{04} = g_{04} + g_{03} + p_{03}.g_{02} + p_{03}.p_{02}.g_{01} + p_{03}.p_{02}.p_{01}.g_{00}$$

The last term in c_{04} has 5 logic levels, whereas, H_{04} has only 4 logic level compared to c_{04}. Although the computation of pseudo carry is simpler compared to conventional carry, but the sum calculation is complicated. The sum bit for each bit position is given as $S_b = d_b\ XOR\ c_{b-1}$. Since, $c_{b-1} = p_{b-1}.H_{b-1}$, then $S_b = d_b\ XOR\ (p_{b-1}.H_{b-1})$. The computation of S_b bits can be transformed [15] into $S_b = \overline{H_{b-1}}.d_{b-1} + H_{b-1}(d_b \odot p_{b-1})$. Based on the value of H_{b-1} either d_{b-1} or $(d_b \odot p_{b-1})$ can be computed using a 2:1 multiplexer. The calculation time of the sum bits is decreased because of lessened complexity of the Ling carries.

IV. MODIFIED LING EQUATION BASED PARALLEL PREFIX ADDER

Consider the pseudo carry equation for 3rd, 4th, 5th and 6th bit positions.

$$H_{03} = g_{03} + g_{02} + p_{02}.g_{01} + p_{02}.p_{01}.g_{00} \quad (1)$$

$$H_{04} = g_{04} + g_{03} + p_{03}.g_{02} + p_{03}.p_{02}.g_{01} + p_{03}.p_{02}.p_{01}.g_{00} \quad (2)$$

$$H_{05} = g_{05} + g_{04} + p_{04}.g_{03} + p_{04}.p_{03}.g_{02} + p_{04}.p_{03}.p_{02}.g_{01} + p_{04}.p_{03}.p_{02}.p_{01}.g_{00} \quad (3)$$

$$H_{06} = g_{06} + g_{05} + p_{05}.g_{04} + p_{05}.p_{04}.g_{03} + p_{05}.p_{04}.p_{03}.g_{02} + p_{05}.p_{04}.p_{03}.p_{02}.g_{01} + p_{05}.p_{04}.p_{03}.p_{02}.p_{01}.g_{00} \quad (4)$$

Since, $g_b = g_b.p_b$, Eq. 1, 2, 3 and 4 can be rewritten as

$$H_{03} = (g_{03} + g_{02}) + p_{02}.p_{01}.(g_{01} + g_{00})$$

Let, G_b^* and P_b^* be the intermediate generate and intermediate propagate bits respectively [10] given as:

$$G_b^* = g_b + g_{b+1} \text{and } P_b^* = p_b.p_{b-1},\ 0 \le b \le m-1.$$
With $g_{-1} = p_{-1} = 0$and $G_k^* = P_k^* = 0$, for $k < 0$,

$$\begin{aligned} H_{03} &= G_{03:02} + P_{02:01}.G_{01:00} \\ &= (G_{03:02}, P_{02:01}) \odot (G_{01:00}, P_{00:-1}) \\ &= (G_{03}^*, P_{02}^*) \odot (G_{01}^*, P_{00}^*) \end{aligned} \quad (5)$$

$$\begin{aligned} H_{04} &= (g_{04} + g_{03}) + p_{03}.p_{02}.(g_{02} + g_{01}) + p_{03}.p_{02}.p_{01}.p_{00}.g_{00} \\ &= G_{04:03} + P_{03:02}.G_{02:01} + P_{03:02}.P_{01:00}.G_{00:-1} \\ &= (G_{04:03}, P_{03:02}) \odot (G_{02:01}, P_{01:00}) \odot (G_{00:-1}, P_{-1:-2}) \\ &= (G_{04}^*, P_{03}^*) \odot (G_{02}^*, P_{01}^*) \odot (G_{00}^*, P_{-1}^*) \end{aligned} \quad (6)$$

$$
\begin{aligned}
H_{05} &= (g_{05} + g_{04}) + p_{04}.g_{03}(g_{02} + g_{01}) + p_{04}.p_{03}.p_{02}.p_{01}.(g_{01} + g_{00}) \\
&= G_{05:04} + P_{04:03}.G_{02:01} + P_{04:03}.P_{02:01}.G_{01:00} \\
&= (G_{05:04}, P_{04:03}) \odot (G_{03:02}, P_{02:01}) \odot (G_{01:00}, P_{00:-1}) \\
&= (G^*_{05}, P^*_{04}) \odot (G^*_{03}, P^*_{02}) \odot (G^*_{01}, P^*_{00})
\end{aligned} \tag{7}
$$

$$
\begin{aligned}
H_{06} &= (g_{06} + g_{05}) + p_{05}.p_{04}.(g_{04} + g_{03}) + p_{05}.p_{04}.p_{03}.p_{02}.(g_{02} + g_{01}) + p_{05}.p_{04}.p_{03}.p_{02}.p_{01}.g_{00} \\
&= G_{06:05} + P_{05:04}.G_{04:03} + P_{05:04}.P_{03:02}.G_{02:01} + P_{05:04}.P_{03:02}.P_{01:00}.G_{00:-1} \\
&= (G^*_{06}, P^*_{05}) \odot (G^*_{04}, P^*_{03}) \odot (G^*_{02}, P^*_{01}) \odot (G^*_{00}, P^*_{-1})
\end{aligned} \tag{8}
$$

As seen from Eq. 5, 6, 7 and 8 the pseudo carries of odd indexed bit position and even indexed bit position are completely different, independent tree topologies can also be used for odd bits and even bits to optimize the delay [12]. By making use of intermediate generate and propagate pair (G^*_b, P^*_{b-1}) of the even-indexed and the odd-indexed bit positions, the final pseudo carries of remaining bits can be obtained. The pseudo carries of even indexed bit position and oddindexed bit position H_{b+1} are given as

$$
\begin{aligned}
H_b &= (G^*_b, P^*_{b-1}) \odot (G^*_{b-2}, P^*_{b-3}) \odot \ldots\ldots\ldots\ldots \odot (G^*_0, P^*_{-1}) \\
H_{b+1} &= (G^*_{b+1}, P^*_b) \odot (G^*_{b-1}, P^*_{b-2}) \odot \ldots\ldots\ldots\ldots \odot (G^*_1, P^*_0)
\end{aligned}
$$

The proposed parallel prefix adder built on modified ling equation for 8-bit is shown in Figure 1. The computation of real carries is done is four stages, which eliminates the use of multiplexer for calculating sum if ling carries were generated. This saves considerable amount of area. The sum is calculated in the 5th stage by XORing d_b and c_{b-1}. The white circle is a buffer. The white square is a node which computes generate (g_b), propagate (p_b) and half sum (d_b). Black square computes intermediate generate and propagate bits. White square and black square in Figure 1 are elaborated in Figure 2(a) and 2(b) respectively.Black circle that compute group generate and propagate bits are given in Figure 3. The cells which compute the real carries are shown in Figure 4. The black hexagon in Figure 1 is the proposed prefix cell which computes the real carry is presented in Figure 4(a). Since, only intermediate generate term forms the pseudo carry, it eliminates the computation of intermediate propagate term thereby saving two gates.

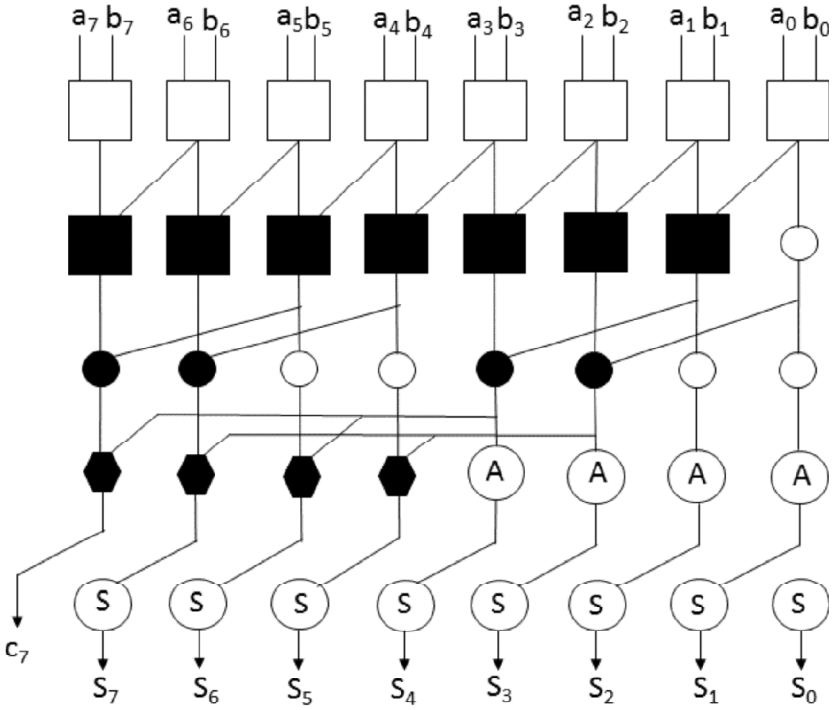

Figure 1: Proposed 8-bit parallel prefix adder based on modified ling equation.

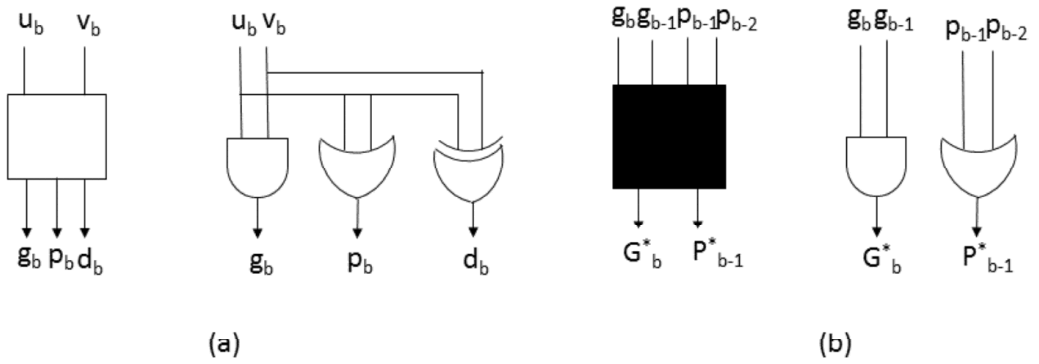

Figure 2: (a) Generate, propagate and half sum computing node. (b) Intermediate generate and propagate computing node [10].

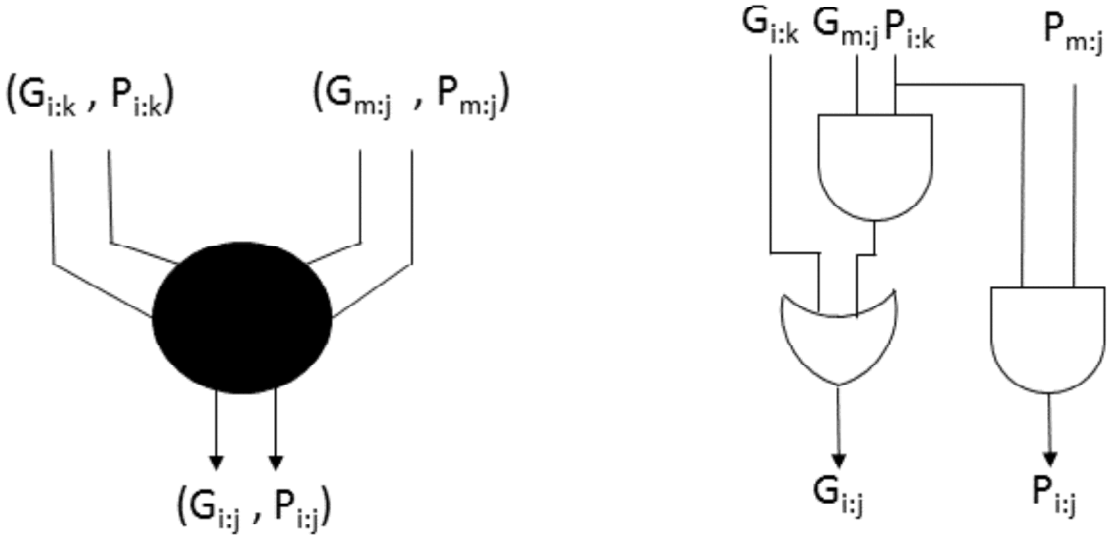

Figure 3: Group generate and propagate computing node [5].

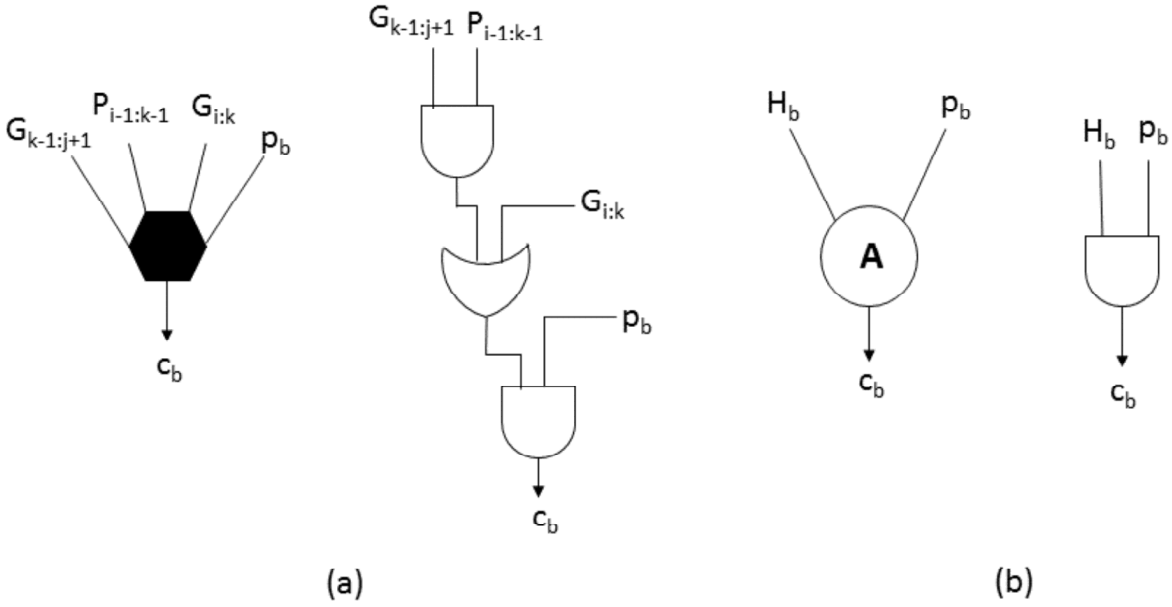

Figure 4: (a) Proposed prefix cell for computing real carries. (b) Prefix cell for computing real carry [11].

The real carries are now obtained by directly ANDing the pseudo carries with respective propagate bit. This is expressed by:

$$c_0 = H_0.\, p_0$$
$$c_1 = H_1.p_1$$
$$c_2 = H_2.p_2$$
$$c_3 = H_3.p_3$$
$$c_4 = (G_{4:3} + P_{3:2}.G_{2:1})p_4$$
$$c_5 = (G_{5:4} + P_{4:3}.G_{3:0})p_5$$
$$c_6 = (G_{6:3} + P_{5:2}.G_{2:1})p_6$$
$$c_7 = (G_{7:4} + P_{6:3}.G_{3:0})p_7$$

Parallel prefix 16-bit adder based on modified ling equation is depicted in Figure 5. White circle with letter 'A' at the center, in level 4 of Figure 5 is an AND operation which is shown in Figure 4(b). This compute the real carries for the bit positioned from 0 to 3 and 6 to 7. This method of designing the adder can be extended to develop a 32-bit parallel prefix adder based on modified ling equation, which is depicted in Figure 6. The grey circle in Figure 6 computes only the group generate requiring two gates. Group propagate is not required in calculating the sum for the respective bits hence group propagate element can be neglected. This is shown in Figure 7. The proposed adder has an advantage of having a regular structure and reduced gate count.

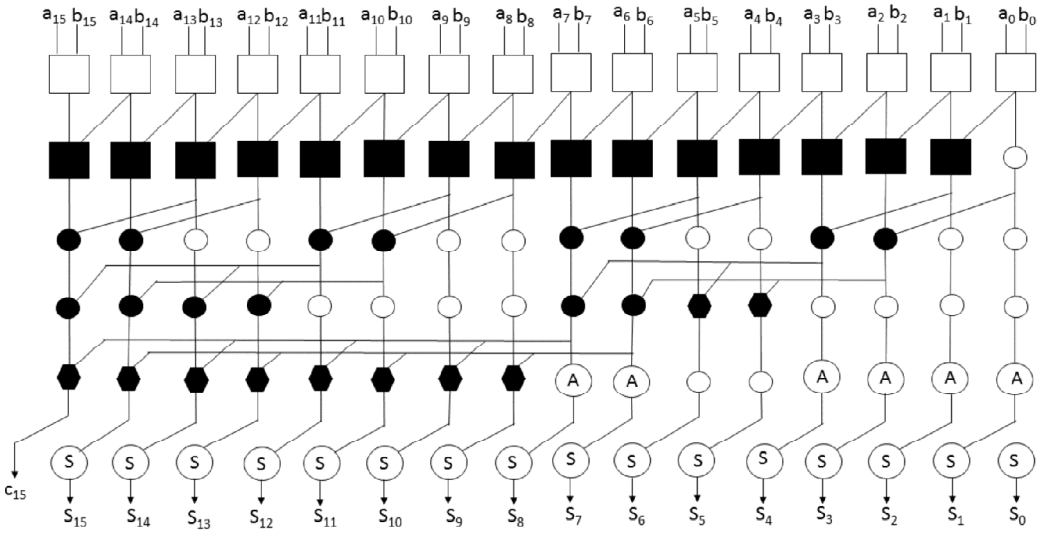

Figure 5: Proposed 16-bit parallel prefix adder based on modified ling equation.

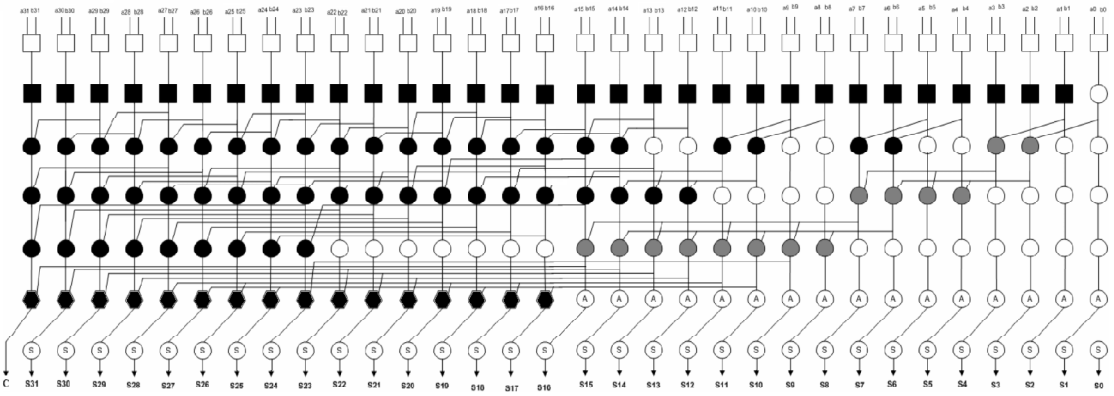

Figure 6: Proposed 32-bit parallel prefix adder based on modified ling equation.

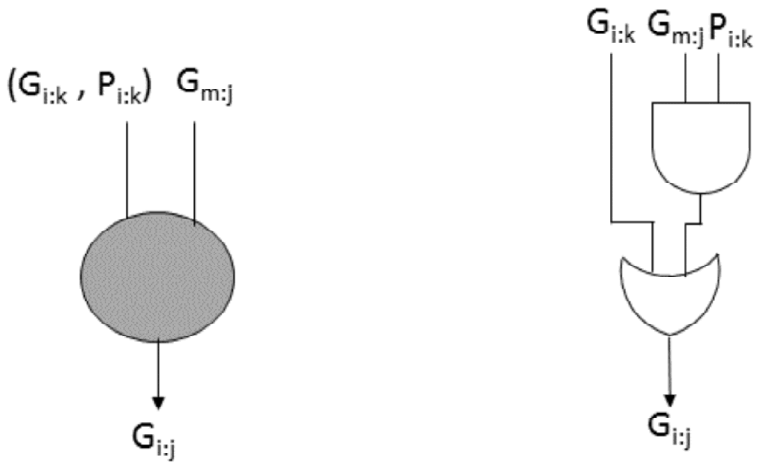

Figure 7: Group generate computing node.

V. SYNTHESIS RESULTS

The proposed adder was compared with structures proposed by Giorgos et al. [10], Juang et al. [11], Poornima et al. [12], Kogge-Stone parallel prefix ling adder structure presented by Giorgos et al. [10] and Juang's structure based on Kogge-Stone [11]. Each adder was coded using VHDL structural description and mapped on TSMC 90nm and TSMC 180nm CMOS technology using Cadence RTL Compiler. The derived netlist and design constraints were uploaded into Cadence Encounter tool to generate the layout. RC parasitic information was later extracted from the layout. Table 1, Table 2 and Table 3 shows the area, power and delay estimation of 8-bit, 16-bit and 32-bit adders respectively for 90nm technology. Table 4, Table 5 and Table 6 shows the area, power and delay estimation of 8-bit, 16-bit and 32-bit adders respectively for 180nm technology.Proposed adder clearly outperforms other adders in terms of area, due to one less logic level implementation. Power of proposed adder is also efficient than other traditional adders. Results show that proposed adder has reasonably good speed. In addition to this, the proposed adder contributes to reduced gate count leading to improvement in area cost factor as shown in Table 7.

Table 1
Comparison of 8-bit proposed adder with other adders for 90nm technology

Structure	*Area (μm^2)*	*Power (μW)*	*Delay (ns)*
Proposed	279	6.47	0.87
Giorgos [10]	269	6.49	0.97
Poornima [12]	303	6.98	0.85
Juang [11]	314	7.27	0.80
Kogge Ling [10]	**364**	**8.81**	**0.94**

Table 2
Comparison of 16-bit proposed adder with other adders for 90nm technology

Structure	*Area (µm2)*	*Power (µW)*	*Delay (ns)*
Proposed	651	15.46	1.14
Giorgos [10]	860	22.30	1.36
Poornima [12]	732	16.82	1.03
Juang [11]	785	18.23	1.04
Kogge Ling [10]	**895**	**21.64**	**1.20**

Table 3
Comparison of 32-bit proposed adder with other adders for 90nm technology

Structure	*Area (μm2)*	*Power (μW)*	*Delay (ns)*
Proposed	1661	39.74	1.48
Giorgos [10]	2205	62.80	1.44
Poornima [12]	**1717**	**40.79**	**1.43**

Table 4
Comparison of 8-bit proposed adder with other adders for 180nm technology

Structure	*Area (μm^2)*	*Power (μW)*	*Delay (ns)*
Proposed	1131	52.28	1.82
Giorgos [10]	1071	51.82	1.76
Poornima [12]	1224	55.60	1.79
Juang [11]	1264	57.61	1.65
Kogge Ling [10]	**1444**	**70.43**	**1.75**

Table 5
Comparison of 16-bit proposed adder with other adders for 180nm technology

Structure	*Area (μm2)*	*Power (μW)*	*Delay (ns)*
Proposed	2621	124.0	2.29
Giorgos [10]	3360	177.3	2.46
Poornima [12]	2927	132.2	2.09
Juang [11]	3127	142.0	2.07
Kogge Ling [10]	**3519**	**169.7**	**2.21**

Table 6
Comparison of 32-bit proposed adder with other adders for 180nm technology

Structure	*Area (μm2)*	*Power (μW)*	*Delay (ns)*
Proposed	6600	310.95	2.92
Giorgos [10]	8556	494.93	2.52
Poornima [12]	6812	318.17	2.77
Juang [11]	9826	-	1.67
Juang – KS [11]	**8262**	**-**	**1.87**

Table 7
Comparison of gate count of proposed adder with other adders

Structure	*8-bits*	*16-bits*	*32-bits*
Proposed	83	202	467
Giorgos [10]	162	344	681
Poornima [12]	115	288	496
Juang [11]	89	221	-
Kogge Ling [10]	**90**	**226**	**-**

VI. CONCLUSION

An organized approach for designing a parallel prefix adder using modified ling equation has been presented in this paper. Efficient structures of 8-bit, 16-bit and 32-bit addershave been proposed. These structures have regular interconnection pattern with reduced gate count. They compute the real carry instead of pseudo carry,thus making the sum calculation fast and area efficient. Comparative results demonstrate that the proposed adder is area proficient and performs better than other adders in terms of power, particularly 16-bit and 32-bit adders. This technique can be further extended to design 64-bit adder at the cost of larger fan-out and complex interconnects.Drawing the inference, the proposed adder offers a reduced area and power with a comparative speed. Modern DSP and microprocessors would be very much benefited by adopting the proposed adder.

REFERENCES

[1] Ahmet Sertba and R.Selami Özbey, "A performance analysis of classified binary adder architectures and the VHDL simulations", *J. Electrical & Electronics Engineering*, vol. 4, no. 1, pp. 1025-1030, 2004.

[2] J. Sklansky, "Conditional-sum addition logic", *IRE Tras. Electronic Computers*, vol. 9, pp. 226-231, June 1960.

[3] P.M. Kogge and H.S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations", *IEEE Trans. Computers*, vol. 22, no. 8, pp. 786-793, August 1973.

[4] R.E. Ladner and M.J. Fisher, "Parallel prefix Computation", *J. ACM*, vol. 27, no. 4, pp. 831-838, October 1980.

[5] R.P. Brent and H.T. Kung, "A regular layout for parallel adders", *IEEE Trans. Computers*, vol. 31, no. 3, pp. 260-264, March 1982.

[6] T. Han and D. Carlson, "Fast area-efficient VLSI adders", *Proc. Symp. Computer Arithmetic*, pp 49-56, 1987.

[7] Simon Knowles, "A family of adders", *Proc. 15th IEEE Symp. Computer Arithmetic*, pp. 277-281, 2001.

[8] Huey Ling, "High-speed binary adder", *IBM J. R&D*, vol. 25, no. 3, pp. 156-166, May1981.

[9] C. Efstathiou, H.T. Vergos and D. Nikolos, "Ling Adders in Standard CMOS Technologies*", Proc. IEEE Int. Conf. Electronics, Circuits and Systems (ICECS)*, vol. 2, pp. 485-48, 2002.

[10] Giorgos Dimitrakopoulos and Dimitris Nikolos, "High-Speed parallel-prefix VLSI ling adders" *IEEE Trans. Computers*, vol. 54, no. 2, pp. 225-231, February 2005.

[11] Tso-Bing Juang, Pramod Kumar Meher and Chung-Chun Kuan, "Area-efficient parallel-prefix ling adders" *Proc. IEEE Asia Pacific Conf. Circuits and Systems (APCCAS)*, Kuala Lumpur, pp. 736-739, 2010.

[12] Poornima N and V S Kanchana Bhaaskaran, "Area efficient hybrid parallel prefix adders", *Procedia Materials Science*, vol. 10, pp. 371-380, 2015.

[13] I. Koren, Computer Arithmetic Algorithms, 2nd Edition, Natrick, Massachusetts, 2002.

[14] B. Parhami, Computer Arithmetic-Algorithms and Hardware Designs, New York: Oxford Univ. Press; pp. 75-140, 2000.

[15] S. Vassiliadis, "Recursive Equations for Hardwired Binary Adders", J. Electronics, vol. 67, pp. 201-213, 1989.