Hybrid Parallel Prefix Adder for High Performance Computing

P.Ramanathan[1] P.T.Vanathi[2] S.Geetha[3] S.Kanimozhi[4] and G.Sindhu Ganga[5]

Abstract

The design of a n-bit binary parallel adder for a VLSI circuit is trade-off between speed of operation and hardware complexity (chip area). When speed is not the concern, Ripple Carry Adder (RCA) is the best choice, because it occupies less area and has a regular structure. The delay is directly proportional to the number of bits of the adder. Today performance is much more important than chip area; the Carry Look Ahead (CLA) adder may be the right choice in which the carry bits are predicted well in advance to speed up the computation. But when the number of bits increases, the fan-in and fan-out of the CLA increases and hence speed starts to detoriate. In order to compensate this drawback parallel prefix adders are preferred. Parallel prefix adders are slight variation of CLA in which the carry bits are generated parallely without increasing the fan-in and fan-out of the computation nodes to a larger extent. The proposed PPA structure has given better optimization in power consumption, delay and power delay product. A comparative study is made between the proposed structure and existing PPA algorithms. All the circuits were simulated using Tanner EDA in 180nm technology.

[1]Senior Lecturer, ECE Department, PSG College of Technology, Coimbatore. E-mail: pramanathan_2000 @yahoo.com

[2]Assistant Professor, ECE Department, PSG College of Technology, Coimbatore.

[3,4 & 5] 3rd Year students, ECE Department, PSG College of Technology, Coimbatore.

Keywords : Parallel Prefix Adder, Dot Operator,Semi-Dot Operator.

1. Introduction

VLSI Integer adders find applications in Arithmetic and Logic units (ALU's), microprocessors and memory addressing units. Speed of the adder often decides the minimum clock cycle time in a microprocessor. The need for a Parallel Prefix adder (PPA) is that it is primarily fast when compared with a ripple carry adder. PPA is family of adders derived from the commonly known carry look ahead adders. These adders are suited for adders with wider word lengths. PPA circuits use a tree network to reduce the latency to $O(\log_2 n)$ where 'n' represents the number of bits. This chapter discusses the design proposal of new prefix adder architecture for 8-bit, 16-bit and 32-bit addition. The proposed architectures have the least number of computation nodes when compared with its peer existing one's. This reduction in hardware of the proposed architectures helps to reap a benefit in the form of least power and least power delay product.

2. Existing Parallel Prefix Adders

The structure of the prefix network specifies the type of the PPA. The Prefix network described by Haiku Zhu, Chung-Kuan Cheng and Ronald Graham [1], has the minimal depth for a given 'n' bit adder. Optimal logarithmic adder structures with a fan-out of two for minimizing the area-delay product is presented by Matthew Ziegler and Mircea Stan [2]. The Sklansky adder [3] presents a minimum depth prefix network at the cost of increased fan-out for certain computation nodes. In Sklansky adder, the fan-out from the inputs to outputs along the critical path increases drastically which

introduces latency in the circuit. The algorithm invented by Kogge-Stone [4] has both optimal depth and low fan-out but produces massively complex circuit realizations and also account for large number of interconnects. Kogge-Stone adder possesses a regular layout and is preferred for high performance applications. Brent-Kung adder [5] has the merit of minimal number of computation nodes, which yields in reduced area but structure has maximum depth which yields slight increase in latency when compared with other structures. Brent Kung adder is oriented towards simpler tree structure with a fewer computation nodes. The Han-Carlson adder [6] combines Brent-Kung and Kogge-Stone structures to achieve a balance between logic depth and interconnect count. Han-Carlson adder the reduces the hardware complexity when compared to that of a Kogge-Stone adder but at the cost of introducing a additional stage to its carry merge path. Ladner and Fischer [7] proposed a general method to construct a prefix network with slightly higher depth when compared with Sklansky topology but achieved some merit by reducing the maximum fan-out for computation nodes in the critical path. -Fischer adder is an improved version of Sklansky adder, where the maximum fan-out is reduced. Taxonomy of classical Prefix Parallel Adders based on fan-out, interconnect count and depth characteristics has been presented by Harris [8]. In this paper, a novel hybrid prefix adder structure for 8-bit, 16-bit and 32-bit has been proposed. The proposed structures have the least power delay product amongst all its peer one's.

3. Comparative Study On Existing Prefix Adders

Table 1 summarizes the data regarding the requirement of number of computation nodes and logic depth for various existing Parallel Prefix adders. Let 'n' be the word-length of the adder in terms of bits

Table 1 : Structural Comparison Of N-bit Parallel Prefix Adders

Adder Type	Number of Computation Nodes	Logic Depth
Brent-Kung	$[2*n-2-\log_2 n]$	$[(2*\log_2 n)-2]$
Kogge-Stone	$[(n*\log_2 n)-n+1]$	$\log_2 n$
Han-Carlson	$[\frac{n}{2}*(\log_2 n)]$	$[(\log_2 n)+1]$
Ladner-Fischer	$[\frac{n}{2}*(\log_2 n)]$	$[(\log_2 n)+1]$
Sklansky	$[\frac{n}{2}*(\log_2 n)]$	$\log_2 n$

4. Proposed Hybrid Prefix Adder Architecture

The Proposed 32-bit Parallel Prefix adder architectures is shown in Figure.1. The architectures employ the associative property of the PPA to keep the number of computation nodes at a minimum, by eliminating the massive overlap between the prefix sub-terms being computed.

The Proposed adder structures are implemented using three schemes namely

1) Scheme I
2) Scheme II
3) Scheme III

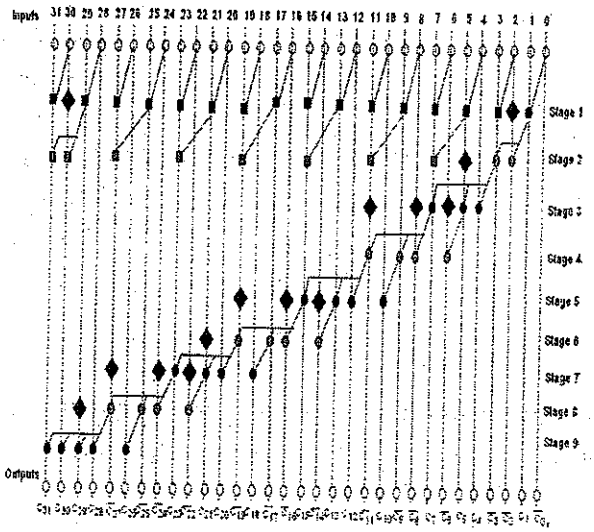

Figure1 : Prefix Graph of the Proposed 32-bit Adder

4.1. Properties of Dot Operator

The prefix operator (•) has two essential properties which allow for greater parallelism.

1) Associative property is listed in equation (1) and (2) given below

$$(G,P)_{[h:j]} \bullet (G,P)_{[j:k]} = (G,P)_{[h:i]} \bullet (G,P)_{[i:k]} \quad (1)$$

$$\left(G,\overline{K}\right)_{[h:j]} \bullet \left(G,\overline{K}\right)_{[j:k]} = \left(G,\overline{K}\right)_{[h:i]} \bullet \left(G,\overline{K}\right)_{[i:k]} \quad (2)$$

Where $h > i \geq j \geq k$.

2) Idempotent property is listed in equation (3) and (4) respectively

$$(G,P)_{[h:j]} \bullet (G,P)_{[i:k]} = (G,P)_{[h:k]} \quad (3)$$

$$\left(G,\overline{K}\right)_{[h:j]} \bullet \left(G,\overline{K}\right)_{[i:k]} = \left(G,\overline{K}\right)_{[h:k]} \quad (4)$$

Associativity allows pre-computation of sub-terms of the prefix equations. This indicates that a serial iteration implied by the above prefix operation can be parallelized. Idempontency allows these sub-terms to overlap, which provides some useful flexibility in the parallelization.

4.2. Scheme I

The first stage of the computation is called as pre-processing. The first stage in the architectures of the proposed 32-bit prefix adder involve the creation of complementary generate and propagate signals for individual operand bits in active low format. The equations (5) and (6) represent the functionality of the first stage.

$$\overline{G_i} = (\overline{a_i \cdot b_i}) \quad (5)$$

$$\overline{P_i} = \overline{a_i \oplus b_i} = a_i \; e \; b_i \quad (6)$$

In the above equations, a_i, b_i represent input operand bits for the adder, where 'i' varies from 0 to 31. The second stage in the Prefix addition is termed as prefix computation. This stage is responsible for creation of group generates and groups propagate signals. For deriving the carry signals in the second stage, this architecture introduces four different computation nodes for achieving improved performance. There are two cells designed for the dot operator. First cell for the dot operator named odd-dot represented by a '■', works with active low inputs and generates active high outputs.

The second cell for the dot operator named even-dot represented by a '■', works with active high inputs and generates active low outputs. Similarly, there are two cells designed for the semi-dot operator. First cell for the semi-dot operator named odd-semi-dot represented by a '•', works with active low inputs and generates active high inputs. The second cell for the semi-dot operator named even-semi-dot represented by a '•', works with active high inputs and generates active low outputs. The last computation node in each column of the architecture is a semi-dot operator. The stages with odd indexes use odd-dot and odd-semi-dot cells where as the stages with even indexes use even-dot and even-semi-dot cells. The equations (7) and (8) represent the functionality of odd-dot and even-dot cells respectively.

$$\begin{aligned}(G,P)_{[i:k]} &= (\overline{G},\overline{P})_{[i:j]} \blacksquare (\overline{G},\overline{P})_{[j-1:k]} \\ &= (\overline{(\overline{G}_{[i:j]}.(\overline{P}_{[i:j]} + \overline{G}_{[j-1:k]}))}, \overline{\overline{P_{[i:j]}} + \overline{P_{[j-1:k]}}}) \\ &= (G_{[i:j]} + P_{[i:j]}.G_{[j-1:k]}, P_{[i:j]}.P_{[j-1:k]})\end{aligned} \quad (7)$$

$$\begin{aligned}(\overline{G},\overline{P})_{[i:k]} &= (G,P)_{[i:j]} \blacksquare (G,P)_{[j-1:k]} \\ &= (\overline{G_{[i:j]} + P_{[i:j]}.G_{[j-1:k]}}, \overline{P_{[i:j]}.P_{[j-1:k]}})\end{aligned} \quad (8)$$

The equations (9) and (10) represent the functionality of odd-semi-dot and even-semi-dot cells respectively.

$$(G)_{[i:0]} = (\overline{G},\overline{P})_{[i:j]} \bullet (\overline{G},\overline{P})_{[j-1:0]}$$
$$= \overline{((\overline{G}_{[i:j]}.(\overline{P}_{[i:j]} + \overline{G}_{[j-1:0]}))} \quad (9)$$
$$= (G_{[i:j]} + P_{[i:j]}.G_{[j-1:0]}) = c_i$$

$$(\overline{G})_{[i:0]} = (G,P)_{[i:j]} \bullet (G,P)_{[j-1:0]}$$
$$= \overline{(G_{[i:j]} + P_{[i:j]}.G_{[j-1:0]})} = \overline{c_i} \quad (10)$$

CMOS logic family will implement only inverting functions. The inverting property of CMOS logic is employed. If both the inputs of a computation node in stage 'i' are from stage '$i-(2*j-1)$', then an alternative cascade of odd computation cell and even computation cell derives the benefit of elimination of two pairs of inverters between successive stages for each computation node in stage 'i'. If a dot or a semi-dot computation node in a stage 'i' receives any of its inputs from stage '$i-(2*j)$, then it is essential to introduce a pair of inverters in a path. From the prefix graph of the proposed structure shown in Fig.1, it is observed that there are only few edges with a pair of inverters, to make (G,P) as $(\overline{G},\overline{P})$ or to make $(\overline{G},\overline{P})$ as (G,P) respectively. The pair of inverters in a path is represented by a '◆' in the prefix graph. Thus by introducing two cells for dot operator and two cells for semi-dot operator, we have eliminated a large number of inverters. Due to inverter elimination in paths, the propagation delay in those paths would have reduced. Further it accounts for a benefit in power reduction, since these inverters if not eliminated, would have contributed to significant amount of power dissipation due to switching.

The output of the odd-semi-dot cells gives the value of the carry signal in that corresponding bit position. The output of the even-semi-dot cell gives the complemented value of carry signal in that corresponding bit position.

The final stage in the prefix addition is termed as post-processing. The final stage involves generation of sum bits from the active low Propagate signals of the individual operand bits and the carry bits generated in true form or complement form. The proposed 32-bit structure has a maximum fan-out of 6 and a logic depth of 9. The lateral fan-out slightly increases, but we get an advantage of limited interconnect lengths since the prefix graph grows along the main diagonal.

4.3. Scheme II

The first stage in the architectures of the proposed prefix adder structures involves the creation of kill and complementary generate and propagate for individual operand bits using the equations (11), (12) and (13) respectively

$$K_i = \overline{a_i + b_i} = \overline{a_i}\,\overline{b_i} \quad (11)$$

$$\overline{G_i} = (\overline{a_i \cdot b_i}) \quad (12)$$

$$\overline{P_i} = (\overline{a_i \oplus b_i}) \quad (13)$$

In the above equations, a_i, b_i represent input operand bits for the adder, where 'i' varies from 0 to 7 for 8-bit, 0 to 15 for 16-bit and 0 to 31 for 32-bit adders respectively. For deriving the carry signals in the second stage, this architecture introduces four different computation nodes for achieving improved performance. There are two cells designed for the dot operator. First cell for the dot operator named odd-dot represented by '■', is defined by the equation (14)

$$(G,\overline{K})_{[i:k]} = (\overline{G},K)_{[i:j]} \blacksquare (\overline{G},K)_{[j-1:k]}$$
$$= \overline{((\overline{G}_{[i:j]}.(K_{[i:j]} + \overline{G}_{[j-1:k]}))}, \overline{K_{[i:j]} + K_{[j-1:k]}}) \quad (14)$$
$$= (G_{[i:j]} + \overline{K}_{[i:j]}.G_{[j-1:k]}, \overline{K_{[i:j]}}.\overline{K_{[j-1:k]}})$$

The second cell for the dot operator named even-dot represented by a '■', is defined by the equation (15)

$$(\overline{G},K)_{[i:k]} = (G,\overline{K})_{[i:j]} \blacksquare (G,\overline{K})_{[j-1:k]}$$
$$= \overline{(G_{[i:j]} + \overline{K}_{[i:j]}.G_{[j-1:k]} \;,\; \overline{K}_{[i:j]}.\overline{K}_{[j-1:k]})} \quad (15)$$

Similarly, there are two cells designed for the semi-dot operator. First cell for the semi-dot operator named odd-semi-dot represented by a '•', the second cell for the semi-dot operator named even-semi-dot represented by a '•', works are defined using equations (16) and (17) respectively.

$$(G)_{[i:k]} = (\overline{G},K)_{[i:j]} \bullet (\overline{G},K)_{[j-1:k]}$$
$$= \overline{((\overline{G}_{[i:j]}.(K_{[i:j]} + \overline{G}_{[j-1:k]}))}$$
$$= (G_{[i:j]} + \overline{K}_{[i:j]}.G_{[j-1:k]}) = c_i \quad (16)$$

$$(\overline{G})_{[i:k]} = (G,\overline{K})_{[i:j]} \bullet (G,\overline{K})_{[j-1:k]}$$
$$= \overline{(G_{[i:j]} + \overline{K}_{[i:j]}.G_{[j-1:k]})} = \overline{c_i} \quad (17)$$

The stages with odd indexes use odd-dot and odd-semi-dot cells where as the stages with even indexes use even-dot and even-semi-dot cells. CMOS logic family will implement only inverting functions. Thus cascading odd cells and even cells alternatively gives the benefit of elimination of two inverters between them, if a dot or a semi-dot computation node in an odd stage receives both of its input edges from any of the even stages and vice-versa. But it is essential to introduce two inverters in a path, if a dot or a semi-dot computation node in an even stage receives any of its edges from any of the even stages and vice-versa. From the prefix graph of the proposed structures, we observe that there are only few edges with a pair of inverters, to make $(\overline{G}, K)$ as $(G, \overline{K})$ or to make $(\overline{G}, K)$ $(G, \overline{K})$ respectively. The pair of inverters in a path is represented by a '◆' in the prefix graph. By introducing two cells for dot operator and two cells for semi-dot operator, we have eliminated a large number of inverters. Due to inverter elimination in paths, the propagation delay in these paths has reduced. Further we achieve a benefit in power reduction, since these inverters if not eliminated, would have contributed to significant amount of power dissipation due to switching. The output of the odd-semi-dot cells gives the value of the carry signal in that corresponding bit position. The output of the even-semi-dot cell gives the complemented value of carry signal in that corresponding bit position. The final stage involves generation of sum bits from the Propagate signals of the individual operand bits and the carry bits generated in true form or complement form.

4.4 Scheme III

This scheme is a slight variation of Scheme II. The first stage involves creation of kill and complementary generate signal only for the individual operand bits. The Propagate signal is derived from the kill and the generate signal using a NOR operation. The rest of the architecture is similar to Scheme II.

5. Simulation Environment

Simulation for the Parallel Prefix adder designs was done using Tanner EDA in 180nm technology. All the Parallel Prefix Adder structures were implemented using CMOS logic family. The aspect ratio of the MOS transistor devices were chosen such that. $\left(\frac{W}{L}\right)_p = 3*\left(\frac{W}{L}\right)_n$.

For TSMC 180 nm technology, threshold voltages of NMOS and PMOS transistors are around 0.3694 V and -0.3944 V respectively and the supply voltage was kept at 1.8 V. The rise time and fall times of the input waveforms were set to 0.10 ns. The parameters considered for comparison are power consumption, worst case delay and power-delay product. The various PPA structures

were then compared with the number of computation nodes needed for circuit realizations. The input patterns were switched after every 10 ns.

6. Simulation Results and Analysis

Table 2 lists the structural characteristics of various 32-bit Parallel Prefix adders. From this table it is observed that the Proposed 32-bit Parallel Prefix adders have the least number of computation nodes amongst all other peer designs.

Table 2 : Structural Comparison Of 32-bit Parallel Prefix Adders

Adder Name	Number of Computation Nodes		Logic Depth
	Dot	Semi-Dot	
Brent-Kung	26	31	8
Kogge-Stone	98	31	5
Han-Carlson	33	31	6
Ladner-Fischer	33	31	6
Sklansky	33	31	5
Proposed	23	31	9

6.1 Simulation Results For Proposed Hybrid Parallel Prefix Adders Using Scheme I

Table 3 : Performance Comparison of 32-bit Parallel Prefix Adders using 180 nm Technology

Adder Name	Average ower (μW)	Delay (ns)	Power-Delay Product (X 10^{-15} Joules)
Brent-Kung	211.6303	1.05	222.211815
Kogge-Stone	352.5317	0.79	278.500043
Han-Carlson	238.2629	0.89	212.053981
Sklansky	272.2407	0.70	190.56849
Ladner-Fischer	217.3403	0.91	197.779673
Proposed	210.9441	0.85	179.302485

6.2. Simulation Results for Proposed Hybrid Parallel Prefix Adders using Scheme II

Table 4 : Performance Comparison Of 32-bit Parallel Prefix Adders Using 180 Nm Technology

Adder Name	Average Power (μW)	Delay (ns)	Power-Delay Product (X 10^{-15} Joules)
Brent-Kung	197.0751	1.39	273.934389
Kogge-Stone	350.2907	0.92	322.267444
Han-Carlson	233.8426	1.25	292.30325
Sklansky	263.7364	1.11	292.747404
Ladner-Fischer	211.6786	1.23	260.364678
Proposed	204.261	1.25	255.32625

6.3. Simulation Results for Proposed Hybrid Parallel Prefix Adders using Scheme III

Table 5 : Performance Comparison Of 32-bit Parallel Prefix Adders Using 180 Nm Technology

Adder Name	Average Power (μW)	Delay (ns)	Power-Delay Product (X 10^{-15} Joules)
Brent-Kung	196.4073	1.42	319.74495
Kogge-Stone	338.959	0.96	325.40064
Han-Carlson	220.71358	1.13	249.40635
Sklansky	244.175	1.07	261.26725
Ladner-Fischer	207.5353	1.25	259.419125
Proposed	187.0906	1.14	213.28328

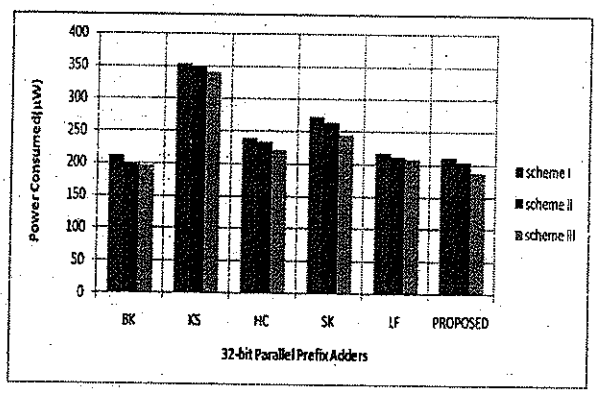

Figure 2 : Power Comparison Of 32-bit Adders In All Three Schemes

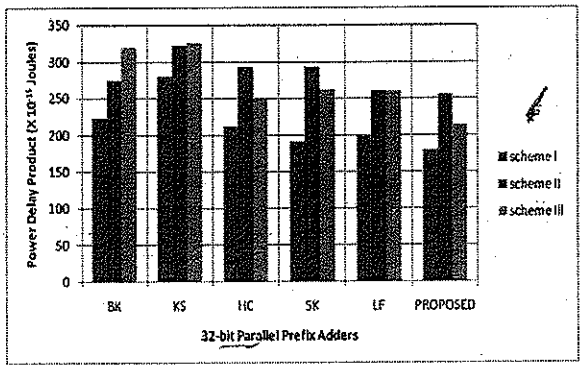

Figure 3 : Power Delay Product Comparison Of 32-bit Adders In All Three Schemes

7. Conclusion

Thus from the analysis it is clear that the power consumption and the power delay product of the proposed structure is found to be reduced. An improvement of 29% to 36% has been achieved in power delay product over Brent Kung Adder and 29% to 43% in power over Kogge Stone Adder. It is inferred that the power saving gradually increases for the proposed architecture implemented using Scheme III when the size of the prefix adder grows. It is also observed that the power delay product of the proposed architecture is minimal when the word length of the adder increases. The proposed architecture implemented using scheme III design consumes least power with very less delay penalty when compared with its peer existing prefix adders. The Proposed Adder has a distinct advantage not only in terms of speed of performance, but also regarding gate count when compared to other adders. Thus it can be used in DSP, memory addressing and microcontroller applications which demands fast addition.

Thus from the analysis it is clear that the power consumption and the power delay product of the proposed structure is found to be reduced. An improvement of 29% to 36% has been achieved in power delay product over Brent Kung Adder and 29% to 43% in power over Kogge Stone Adder. It is inferred that the power saving gradually increases for the proposed architecture implemented using Scheme III when the size of the prefix adder grows. It is also observed that the power delay product of the proposed architecture is minimal when the word length of the adder increases. The proposed architecture implemented using scheme III design consumes least power with very less delay penalty when compared with its peer existing prefix adders. The Proposed Adder has a distinct advantage not only in terms of speed of performance, but also regarding gate count when compared to other adders. Thus it can be used in DSP, memory addressing and microcontroller applications which demands fast addition.

References

[1] Haikun Zhu, Chung-Kuan Cheng and Ronald Graham, *"Constructing Zero Deficiency Parallel Prefix adder of Minimum Depth"*, Proceedings of 2005 Asia South Pacific Design Automation Conference, PP. 883–888, 2005.

[2] Matthew Ziegler, Mircea Stan, *"Optimal Logarithmic Adder structures with a fan-out of two for minimizing area delay product"*, IEEE 2001.

[3] J. Sklansky, *"Conditional Sum Addition Logic"*, IRE Transactions on Electronic computers, Vol. EC-9, PP. 226-231, 1960.

[4] P.Kogge and H.Stone, *"A Parallel Algorithm for the efficient solution of a general class of recurrence relations"*, IEEE Transactions on Computers, Vol. C-22, No.8, PP.786-793, August 1973.

[5] R.Brent and H.Kung, *"A Regular Layout for Parallel adders"*, IEEE Transaction on Computers, Vol. C-31, No.3, PP. 260-264, March 1982.

[6] T. Han and D. Carlson, *"Fast Area Efficient VLSI adders"*, Proceedings of the 8th Symposium on Computer Arithmetic, PP.49-56, September 1987.

[7] R. Ladner and M. Fischer, *"Parallel Prefix Computation"*, Journal of ACM, Vol.27, No.4, PP. 831-838, October 1980,

[8] David Harris, *"A Taxonomy of Parallel Prefix, Networks"*, Proceedings of the 37th Asilomar Conference on Signals, Systems and Computers, PP. 2213-2217, 2003.