CHAPTER 4

PARALLEL PREFIX ADDER

4.1 INTRODUCTION

VLSI Integer adders find applications in Arithmetic and Logic Units (ALUs), 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 a family of adders derived from the commonly known carry look ahead adders. These adders are suited for additions 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 deals with the design proposal and implementation of new prefix adder architecture for 8-bit, 16-bit, 32-bit and 64-bit addition. The proposed architectures have the least number of computation nodes when compared with existing one’s. This reduction in hardware of the proposed architectures helps to reap a benefit in the form of reduced power and power-delay product. The proposed architectures are realized using three schemes namely Scheme I, Scheme II and Scheme III. Scheme III Consumes least power when compared to Scheme I and Scheme II. Scheme I performs computation at a faster rate when compared with other schemes.

4.2 RELATED BACKGROUND

A variety of prefix adders are discussed in the literature to achieve area and performance optimization. Conditional sum addition logic for prefix addition proposed by Sklansky (1960) offers a minimum depth prefix network at the cost of increased fan-out for certain computation nodes. The algorithm invented by Kogge and Stone (1973) has both optimal depth and low fan-out but produces massively complex circuit realizations and also accounts for large number of interconnects. The algorithm proposed by Brent and Kung (1982) uses less computation nodes but possesses maximal depth which accounts for increased latency. A general method to construct a prefix network with slightly higher depth when compared with Sklansky topology is proposed by Ladner and Fischer (1980). This method reduces the maximum fan-out for computation nodes in the critical path. The prefix adder proposed by Han and Carlson (1987) combines Brent-Kung and Kogge-Stone adders in-order to achieve a trade-off between logic depth, interconnect count and number of computation nodes. Reto Zimmermann (1996) proposed a heuristic approach for prefix adder optimization using depth controlled compression and expansion. Matthew Ziegler and Stan (2001) proposed prefix adder structures with a maximum fan-out of two for minimizing the area-delay product. Knowles (2001) presented a class of logarithmic adders with minimum depth by allowing the fan-out to grow. Algorithm for generating prefix carry trees proposed by Andrew Beaumont-Smith and Cheng-Chew Lim (2001) uses higher valency prefix cells in the initial stages of the architecture to accomadate less number of prefix cells in the critical path and to achieve less interconnect length. An algorithmic approach to generate irregular PPA proposed by Jianhua Liu et al (2003) achieves minimal delay for a given profile of input signals. The use of higher valency prefix cells for standard prefix architectures such as Brent-Kung, Sklansky, Ladner Fischer, Kogge-Stone and Han-Carlson was proposed by Harris (2004). This leads to reduced number of logic levels at the expense of greater fan-in at each level.

A zero deficient PPA with minimal depth for a given width was proposed by Haikun Zhu et al (2005). The parallel prefix Ling adder proposed by Giorgos Dimitrakopoulos and Dimitris Nikolos (2005) save one logic level of implementation and reduce fan-out requirements of the design by modifying the prefix equations. Zhanpen Jin et al (2005) proposed a modified 64-bit Kogge-Stone based PPA using clocked domino logic to improve the performance. Sparse tree binary adder proposed by Yan Sun et al (2006) combines the benefits of prefix adder and carry select adder to yield a smaller power-delay product. An integer linear programming method to build minimal power prefix adders within the given timing and area constraints is proposed by Jianhua Liu et al (2007). The performance of standard prefix adder architectures namely Brent-Kung, Sklansky, Ladner Fischer, Kogge-Stone and Han-Carlson implemented with Field Programmable Gate Array (FPGA) technology was investigated by Konstantinos Vitoroulis and Al-Khalili (2007). A scheme to enhance parallel prefix addition by incorporating carry save notation was proposed by Chen and Stine (2008). A new technique for exploiting energy savings offered by Data Driven Dynamic Logic (D3L) was proposed for Kogge-Stone adder by Fabio Frustaci et al (2009). Most of the PPA adders discussed in the literature to speed up the binary addition, use large number of computation nodes to attain parallelism. This accounts for increased power consumption.

4.3 STANDARD PREFIX ADDERS

Let $A = a_{n-1}...a_1a_0$ and $B = b_{n-1}.....b_1b_0$ be the n-bit augend and n-bit addend respectively, then binary addition is defined by the equations (4.1) and (4.2)

$$S_i = a_i \oplus b_i \oplus c_{i-1} \tag{4.1}$$

$$c_i = a_ib_i + a_ic_{i-1} + b_ic_{i-1} \tag{4.2}$$

There are two methods for prefix addition.

Method 1:

In this method prefix addition is carried out in three steps.

Step 1: Pre-processing

Pre-processing, involves creation of generate and propagate signals. According to prefix computation g_i (generate) and p_i (propagate) signals are defined by the equations (4.3) and (4.4) respectively.

$$g_i = a_i b_i \tag{4.3}$$

$$p_i = a_i \oplus b_i \tag{4.4}$$

Step 2: Prefix Computation

PPA construction depends on the notion of group carry propagate and group generate signals. Group generate and group propagate signals are defined by the equations (4.5) and (4.6) respectively.

$$G_{[i:k]} = \left\{ \begin{matrix} g_i, & if \;\; i = k \\ G_{[i:j]} + P_{[i:j]}.G_{[j-1:k]}, & otherwise \end{matrix} \right\} \tag{4.5}$$

$$P_{[i:k]} = \left\{ \begin{matrix} p_i, & if \;\; i = k \\ P_{[i:j]}.P_{[j-1:k]}, & otherwise \end{matrix} \right\} \tag{4.6}$$

To simplify the representation of G and P, an operator called as dot operator represented by $'*'$ is introduced to create group generate and group propagate, and is defined by the equation (4.7) as

$$(G,P)_{[i:k]} = (G,P)_{[i:j]} * (G,P)_{[j-1:k]} \tag{4.7}$$

Step 3: Post-processing

Post-processing step involves formation of carry and sum bits for each individual operand bit. The equations for c_i and S_i, are defined as per equations (4.8) and (4.9) respectively.

$$c_i = G_{[i:0]} \tag{4.8}$$

$$S_i = p_i \oplus c_{i-1} \tag{4.9}$$

Method 2:

In this method also, prefix addition is carried out in three steps.

Step 1: Pre-processing

Pre-processing, involves creation of generate (g_i), propagate (p_i) and kill (k_i) signals for each bit according to the relations given below in equations (4.10) and (4.11) and (4.12) respectively.

$$g_i = a_i b_i \tag{4.10}$$

$$p_i = a_i \oplus b_i \tag{4.11}$$

$$k_i = \overline{a_i + b_i} \tag{4.12}$$

Step 2: Prefix Computation

Prefix computation involves creation of group generate and group kill bar signals. Group generate and group kill bar signals are defined according to the equations (4.13) and (4.14) respectively.

$$G_{[i:k]} = \begin{cases} g_i, & if\ i = k \\ G_{[i:j]} + \overline{K_{[i:j]}}.G_{[j-1:k]}, & otherwise \end{cases} \tag{4.13}$$

$$\overline{K}_{[i:k]} = \begin{cases} \overline{k_i}, & if\ i = k \\ \overline{K}_{[i:j]}.\overline{K}_{[j-1:k]}, & otherwise \end{cases} \tag{4.14}$$

To simplify the representation of G and $\overline{K}$, an operator called as dot operator represented by $'*'$ is introduced to create group generate and group kill bar, and is defined by the equation (4.15) as

$$(G,\overline{K})_{[i:k]} = (G,\overline{K})_{[i:j]} * (G,\overline{K})_{[j-1:k]} \tag{4.15}$$

Step 3: Post-processing

Post-processing is a step in which c_i and S_i are computed as per equations (4.16) and (4.17) respectively.

$$c_i = G_{[i:0]} \tag{4.16}$$

$$S_i = p_i \oplus c_{i-1} \tag{4.17}$$

In both the methods, the first and last stages are intrinsically fast because they involve only simple operations on signals local to each bit position. The intermediate stage embodies long-distance propagation of carries. So the performance of the adder depends on the intermediate stage.

The prefix operator has two essential properties namely associative property and idempotent property which allow for greater parallelism.

1) Associative property can be explained as given in equations (4.18) and (4.19)

$$\left(G,P\right)_{[h:j]} * \left(G,P\right)_{[j:k]} = \left(G,P\right)_{[h:i]} * \left(G,P\right)_{[i:k]} \tag{4.18}$$

$$\left(G,\overline{K}\right)_{[h:j]} * \left(G,\overline{K}\right)_{[j:k]} = \left(G,\overline{K}\right)_{[h:i]} * \left(G,\overline{K}\right)_{[i:k]} \tag{4.19}$$

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

2) Idempotent property can be explained as given in equations (4.20) and (4.21)

$$(G,P)_{[h:j]} * (G,P)_{[i:k]} = (G,P)_{[h:k]} \tag{4.20}$$

$$\left(G,\overline{K}\right)_{[h:j]} * \left(G,\overline{K}\right)_{[i:k]} = \left(G,\overline{K}\right)_{[h:k]} \tag{4.21}$$

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

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.

This section discusses some of the important prefix adder algorithms for 16-bit word length. The algorithms will be visualized using Directed Acyclic Graphs (DAG's) with the edges standing for signals or signal pairs. The white nodes represent the input nodes. The black circles indicate the dot operator. The gray circles represent the semi-dot operator.

4.3.1 Sklansky Adder

Figure 4.1 shows a 16-bit Sklansky adder (Sklansky, 1960). It has minimum depth prefix graph. The longest lateral fanning wires go from a node to $\frac{n}{2}$ other nodes.

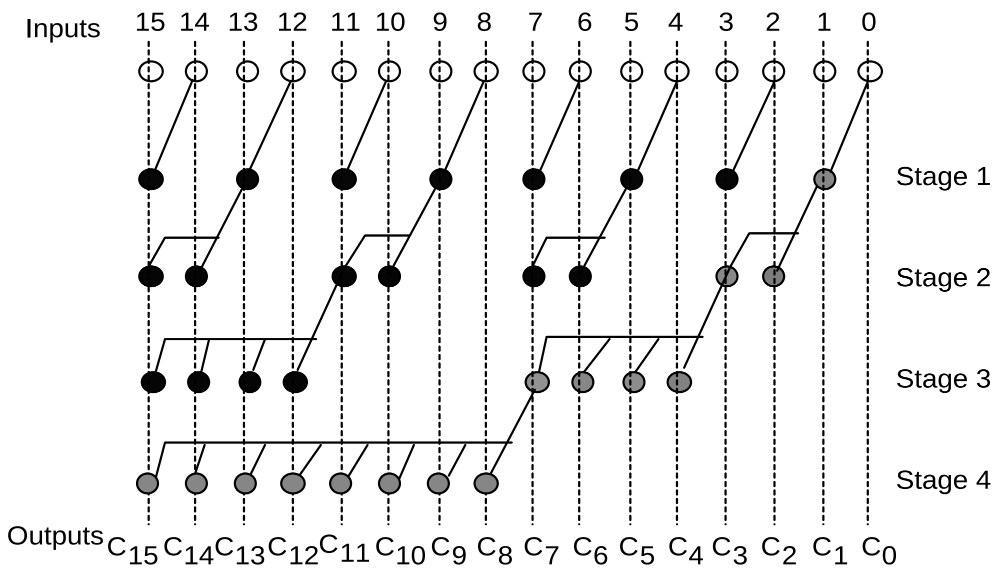

Figure 4.1 Prefix Graph of a 16-bit Sklansky Adder

The structure possesses an optimal depth given by $\log_2 n$ and the number of computational nodes is given by $[\frac{n}{2}(\log_2 n)]$. The fan-out of the Sklansky's adder increases drastically from the inputs to outputs along the critical path, which accounts for large amount of latency. This degrades the performance of the structure when the number of bits of the adder becomes large.

4.3.2 Kogge-Stone Adder

Figure 4.2 shows the prefix graph for a 16-bit Kogge-Stone adder (Kogge and Stone, 1973). Adders implemented using this technique possess regular layout and a controlled fan-out of two.

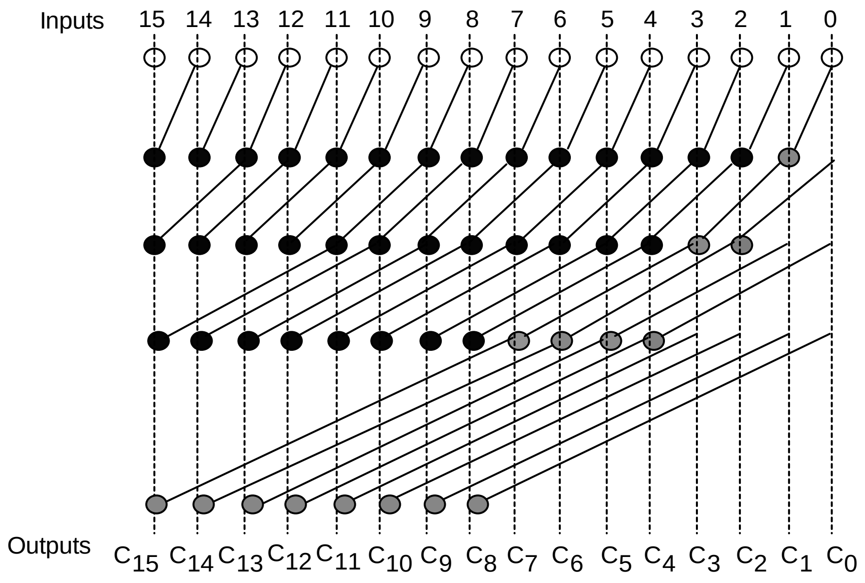

Figure 4.2 Prefix Graph of a 16-bit Kogge-Stone Adder

Kogge-Stone structure is very attractive for high-speed applications. However, it comes at the cost of area and power. The delay of the structure is given by $\log_2 n$. This structure possesses $[(n)(\log_2 n) - n + 1]$ computation nodes. The Kogge-Stone scheme addresses the problem of fan-out by introducing a recursive doubling algorithm. It uses idempotency property to limit the lateral fan-out, but at the cost of a dramatic increase in the number of lateral wires at each stage. This is because there, is a massive overlap between the prefix sub-terms being pre-computed.

4.3.3 Brent-Kung Adder

Figure 4.3 shows the prefix graph of a 16-bit Brent-Kung adder (Brent and Kung, 1980). A simpler tree structure could be formed, if only the carry at every power of two positions is computed as proposed by Brent and Kung. An inverse carry tree is added to compute intermediate carries. Its wire complexity is much less than that of a Kogge-Stone adder. The delay of the structure is given by $[(2)(\log_2 n) - 2]$ and the number of computation nodes is given by $[2(n) - 2 - \log_2 n]$.

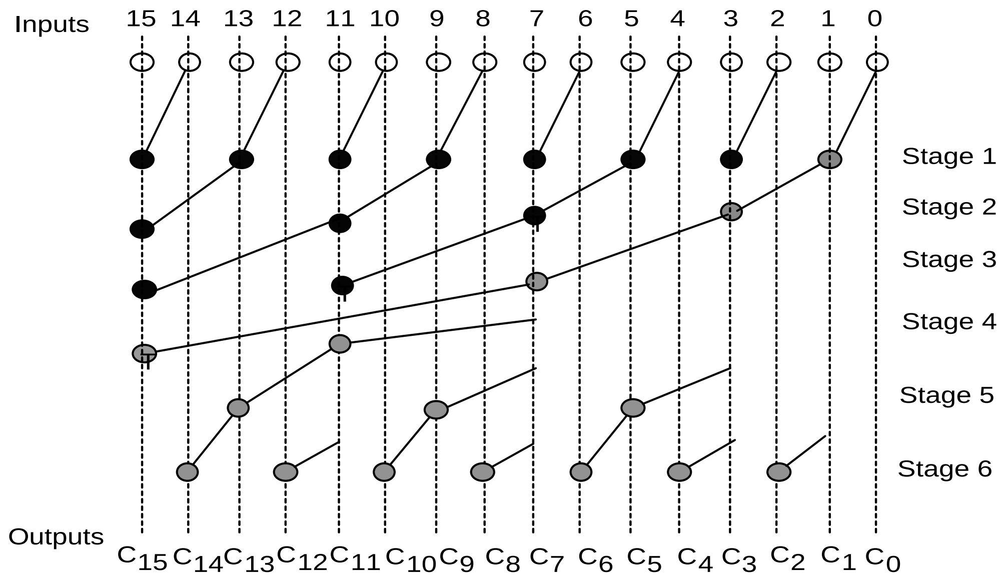
Inputs
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Stage 1
Stage 2
Stage 3
Stage 4
Stage 5
Stage 6
Outputs
C_{15} C_{14} C_{13} C_{12} C_{11} C_{10} C_9 C_8 C_7 C_6 C_5 C_4 C_3 C_2 C_1 C_0

Figure 4.3 Prefix Graph of a 16-bit Brent-Kung Adder

4.3.4 Han-Carlson Adder

Figure 4.4 shows the prefix graph for a 16-bit Han-Carlson adder (Han and Carlson, 1987). It is a hybrid design combining stages from Brent-Kung and Kogge-Stone. It has five stages, the first stage resembles Brent-Kung adder and the middle three stages resemble Kogge-Stone adder. It possess wires with shorter span than Kogge-Stone. The dot operator was placed in the odd bit positions in the initial stages, but the dot operator was placed in the even bit positions in the final stage. The delay in this structure is given by $[(\log_2 n)+1]$, while the computation hardware complexity is $[\frac{n}{2}(\log_2 n)]$. The hardware complexity is reduced compared to Kogge-Stone adder, but at the cost of introducing an additional stage to its carry merge path. This structure again trades off an increase in logical depth for a reduction in fan-out.

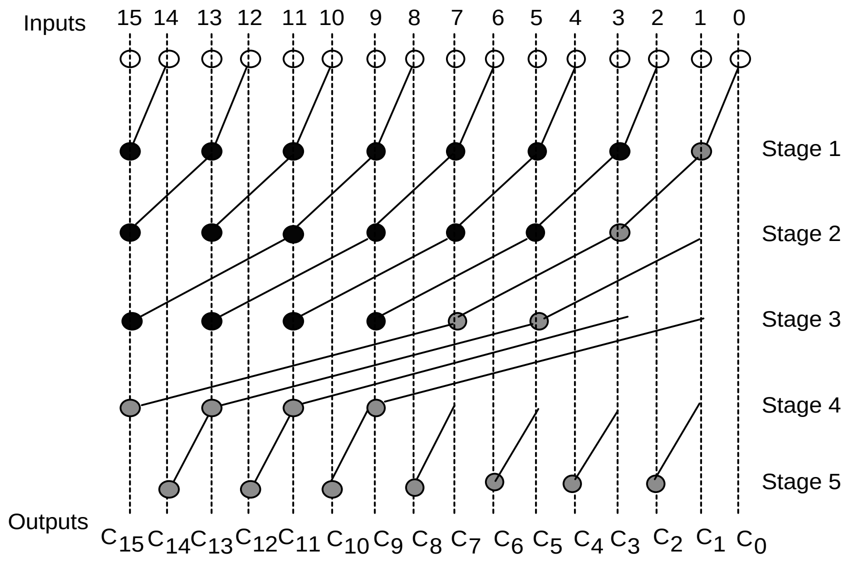

Figure 4.4 Prefix Graph of a 16-bit Han-Carlson Adder

4.3.5 Ladner-Fischer Adder

The Figure 4.5 shows the prefix graph structure for a 16-bit Ladner-Fischer adder (Ladner and Fischer, 1980).

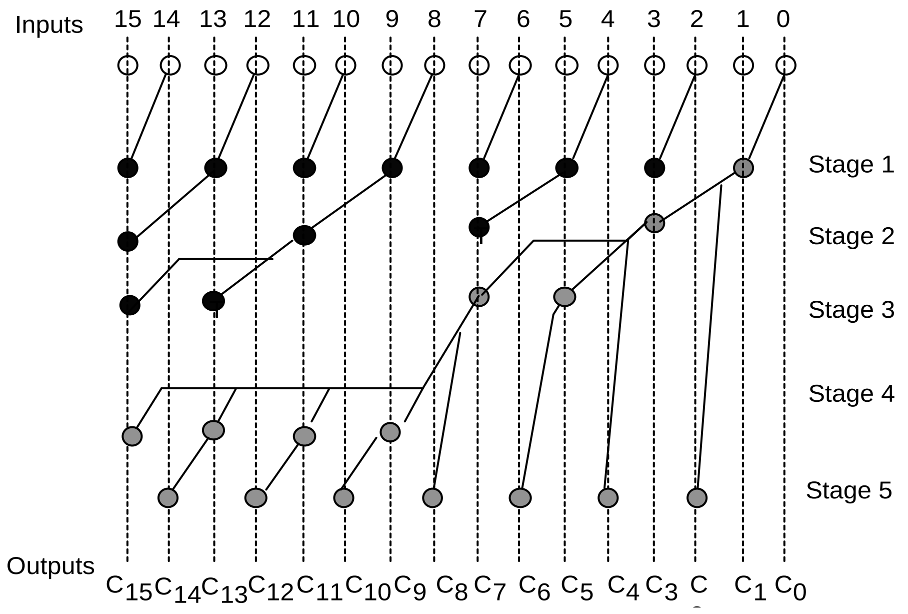

Figure 4.5 Prefix Graph of a 16-bit Ladner-Fischer Adder

This structure is a modified version of Sklansky's adder. In a 16-bit Ladner-Fischer adder, the longest lateral fanning wires go from a node to $\frac{n}{4}$ other nodes. The delay of the structure is given by $\log_2 n+1$. The number of computational nodes is given by $[\frac{n}{2}(\log_2 n)]$. Table 4.1 summarizes the structural details of the existing PPAs.

Table 4.1 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	$[\frac{n}{2}(\log_2 n)]$	$\log_2 n$

4.4 PROPOSED HYBRID PREFIX ADDER ARCHITECTURES

The Proposed 8-bit, 16-bit, 32-bit and 64-bit PPA architectures are shown in Figures 4.6 to 4.9 respectively. 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

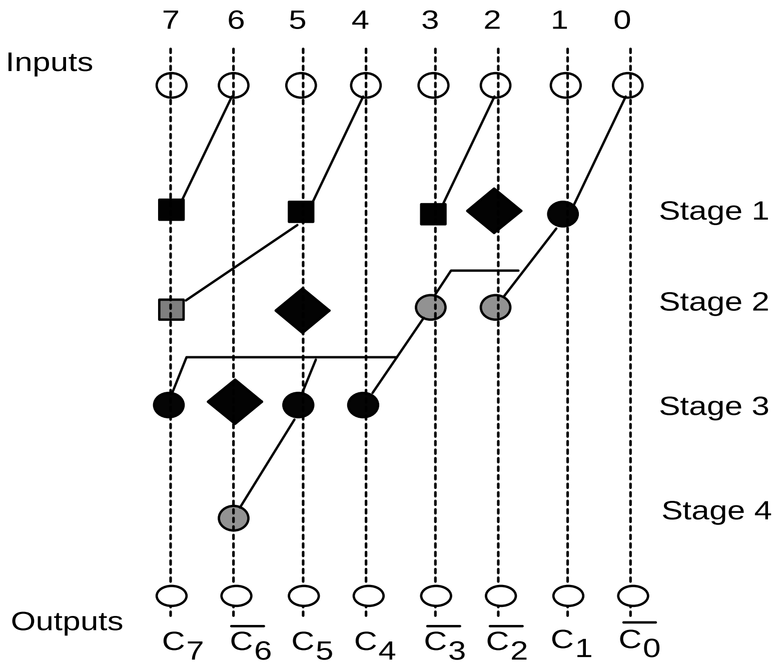

Figure 4.6 Prefix Graph of the Proposed 8-bit Adder

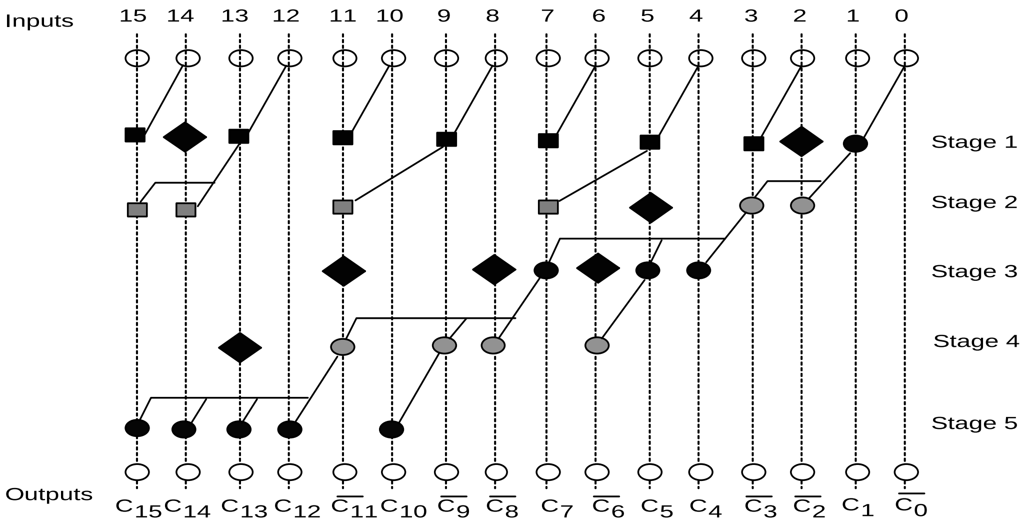
Inputs
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Stage 1
Stage 2
Stage 3
Stage 4
Stage 5
Outputs
C_{15} C_{14} C_{13} C_{12} $\overline{C_{11}}$ C_{10} $\overline{C_9}$ $\overline{C_8}$ C_7 $\overline{C_6}$ C_5 C_4 $\overline{C_3}$ $\overline{C_2}$ C_1 $\overline{C_0}$

Figure 4.7 Prefix Graph of the Proposed 16-bit Adder

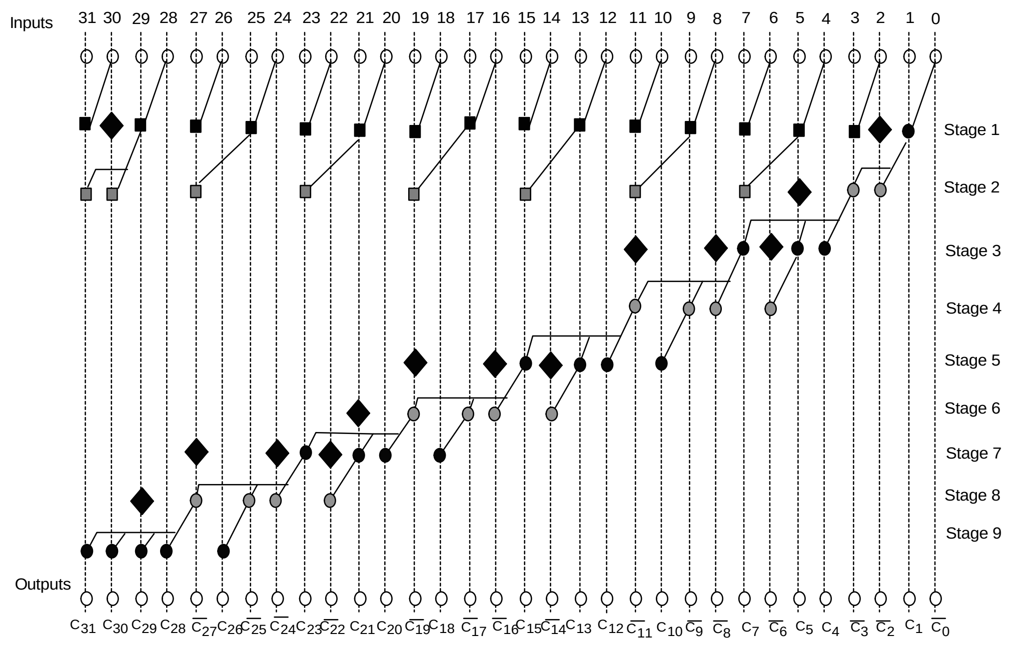
Inputs
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Stage 1
Stage 2
Stage 3
Stage 4
Stage 5
Stage 6
Stage 7
Stage 8
Stage 9
Outputs
c_{31} c_{30} c_{29} c_{28} $\overline{c}_{27}$ c_{26} $\overline{c}_{25}$ $\overline{c}_{24}$ c_{23} $\overline{c}_{22}$ c_{21} c_{20} $\overline{c}_{19}$ c_{18} $\overline{c}_{17}$ $\overline{c}_{16}$ c_{15} $\overline{c}_{14}$ c_{13} c_{12} $\overline{c}_{11}$ c_{10} $\overline{c}_{9}$ $\overline{c}_{8}$ c_{7} $\overline{c}_{6}$ c_{5} c_{4} $\overline{c}_{3}$ $\overline{c}_{2}$ c_{1} $\overline{c}_{0}$

Figure 4.8 Prefix Graph of the Proposed 32-bit Adder

Prefix computation employed for deriving the carry signals in the proposed architectures shown in Figures 4.6 to 4.9, introduce four different computation nodes for achieving improved performance. The proposed architectures use two cells for the dot operator and two cells for the semi-dot operator in the prefix-computation step. First cell for the dot operator named odd-dot is represented by a '■', and second cell for the dot operator named even-dot is represented by a '▨'. First cell for the semi-dot operator named odd-semi-dot is represented by a '●', and the second cell for the semi-dot operator named even-semi-dot is represented by a '●'. The last computation node in each column of the proposed architecture is a semi-dot operator. The stages with odd indices use odd-dot and odd-semi-dot cells where as the stages with even indices use even-dot and even-semi-dot cells. The lateral fan-out slightly increases in the proposed architectures, but we get an advantage of limited interconnect lengths since the prefix graph grows only along the main diagonal.

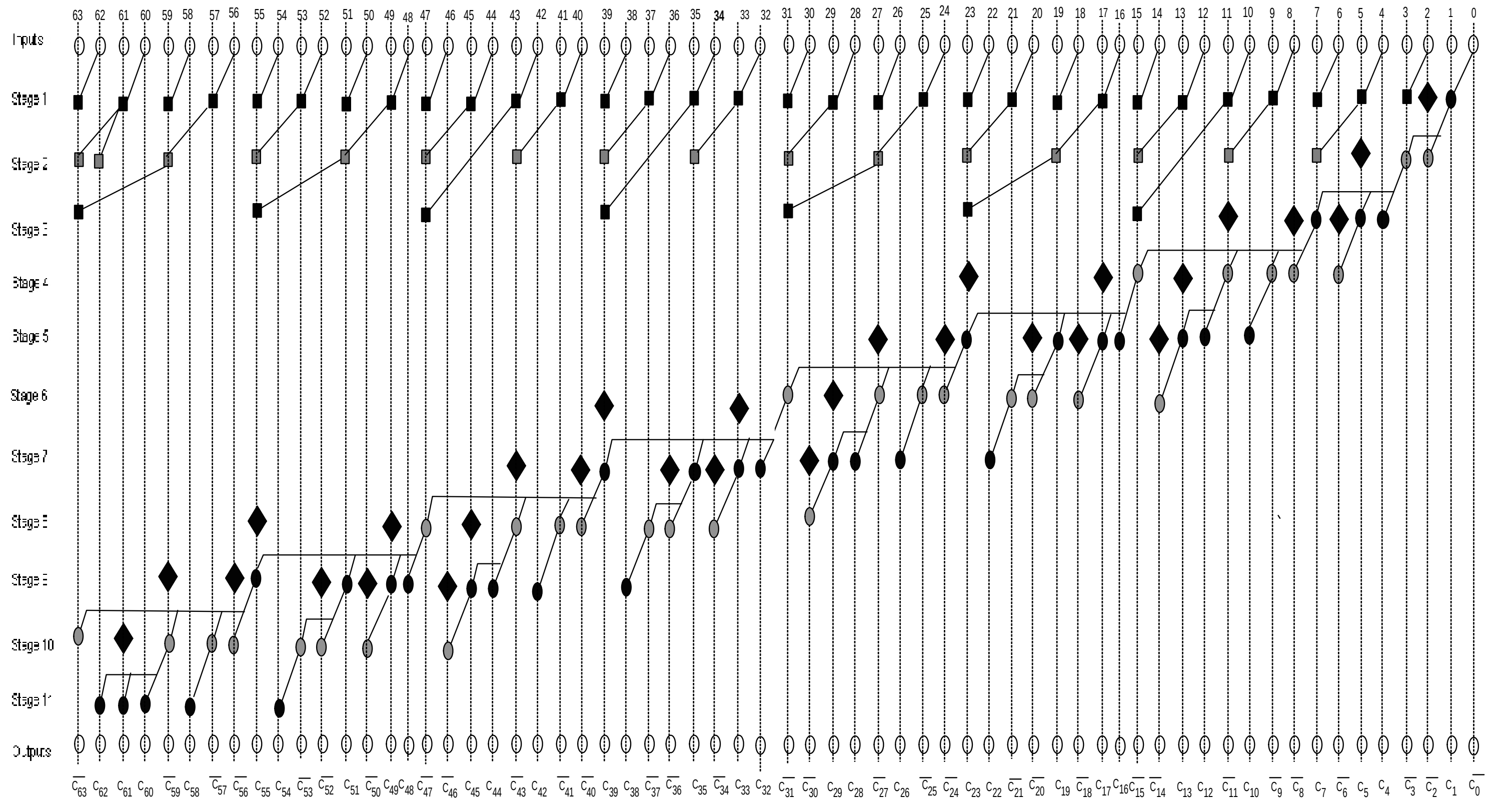
63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Inputs
Stage 1
Stage 2
Stage 3
Stage 4
Stage 5
Stage 6
Stage 7
Stage 8
Stage 9
Stage 10
Stage 11
Outputs
$\overline{c_{63}}$ c_{62} c_{61} c_{60} $\overline{c_{59}}$ c_{58} $\overline{c_{57}}$ $\overline{c_{56}}$ c_{55} c_{54} $\overline{c_{53}}$ $\overline{c_{52}}$ c_{51} $\overline{c_{50}}$ c_{49} c_{48} $\overline{c_{47}}$ $\overline{c_{46}}$ c_{45} c_{44} $\overline{c_{43}}$ c_{42} $\overline{c_{41}}$ $\overline{c_{40}}$ c_{39} c_{38} $\overline{c_{37}}$ $\overline{c_{36}}$ c_{35} $\overline{c_{34}}$ c_{33} c_{32} $\overline{c_{31}}$ $\overline{c_{30}}$ c_{29} c_{28} $\overline{c_{27}}$ c_{26} $\overline{c_{25}}$ $\overline{c_{24}}$ c_{23} c_{22} $\overline{c_{21}}$ $\overline{c_{20}}$ c_{19} $\overline{c_{18}}$ c_{17} c_{16} $\overline{c_{15}}$ $\overline{c_{14}}$ c_{13} c_{12} $\overline{c_{11}}$ c_{10} $\overline{c_9}$ $\overline{c_8}$ c_7 $\overline{c_6}$ c_5 c_4 $\overline{c_3}$ $\overline{c_2}$ c_1 $\overline{c_0}$

Figure 4.9 Prefix Graph of the Proposed 64-bit Adder

CMOS logic family may be used to implement only inverting functions. The inverting property of CMOS logic is employed, by alternatively cascading odd computation cells and even computation cells. An alternative cascade of odd computation cell and even computation cell provides the benefit of elimination of two pairs of inverters between successive stages. This benefit is achieved, if both the inputs of a computation node in stage $'i'$ are from stage $'i-(2*j-1)'$ where $j \geq 0$. A pair of inverters is introduced in the path, if a dot or a semi-dot computation node in a stage $'i'$ receives any of its inputs from stage $'i-(2*j)'$. The pair of inverters in a path is represented by a '◆' in the prefix graph. From the prefix graph of the proposed structure shown in Figure 4.6, Figure 4.7, Figure 4.8 and Figure 4.9, it is observed that there are only few edges with a pair of inverters. Thus by introducing two cells for dot operator and two cells for semi-dot operator, a large number of inverters are eliminated. Due to inverter elimination in paths, the propagation delay in those paths will be 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.

4.4.1 Scheme I

The Pre-processing stage in the proposed prefix adder architectures, involve the creation of complementary generate and propagate signals for individual operand bits. The equations (4.22) and (4.23) represent the functionality of the first stage.

$$\overline{g_i} = (\overline{a_i \cdot b_i}) \tag{4.22}$$

$$\overline{p_i} = \overline{a_i \oplus b_i} = a_i \odot b_i \tag{4.23}$$

In the above equations, a_i, b_i represent input operand bits for the adder. In this scheme, the prefix computation stage is responsible for formation of group generate and group propagate signals. The odd-dot operator and odd-semi-dot operator work with active low inputs and generate active high outputs. The even-dot operator and even-semi-dot operator work with active high inputs and generate active low outputs. The equations (4.24) and (4.25) represent the functionality of odd-dot and even-dot cells respectively.

$$\begin{aligned}(G,P)_{[i:k]} &= (\overline{G},\overline{P})_{[i:j]} \blacksquare (\overline{G},\overline{P})_{[j-1:k]} \\ &= (\overline{(\overline{G}_{[i:j]}.(\overline{P}_{[i:j]}+\overline{G}_{[j-1:k]}))}, \overline{\overline{P_{[i:j]}}+\overline{P_{[j-1:k]}}}) \\ &= (G_{[i:j]}+P_{[i:j]}.G_{[j-1:k]}, P_{[i:j]}.P_{[j-1:k]})\end{aligned} \tag{4.24}$$

$$\begin{aligned}(\overline{G},\overline{P})_{[i:k]} &= (G,P)_{[i:j]} \blacksquare (G,P)_{[j-1:k]} \\ &= (\overline{G_{[i:j]}+P_{[i:j]}.G_{[j-1:k]}}, \overline{P_{[i:j]}.P_{[j-1:k]}})\end{aligned} \tag{4.25}$$

The equations (4.26) and (4.27) represent the functionality of odd-semi-dot and even-semi-dot cells respectively.

$$\begin{aligned}(G)_{[i:0]} &= (\overline{G},\overline{P})_{[i:j]} \bullet (\overline{G},\overline{P})_{[j-1:0]} \\ &= (\overline{(\overline{G}_{[i:j]}.(\overline{P}_{[i:j]}+\overline{G}_{[j-1:0]}))}) \\ &= (G_{[i:j]}+P_{[i:j]}.G_{[j-1:0]}) = c_i\end{aligned} \tag{4.26}$$

$$\begin{aligned}(\overline{G})_{[i:0]} &= (G,P)_{[i:j]} \bullet (G,P)_{[j-1:0]} \\ &= (\overline{G_{[i:j]}+P_{[i:j]}.G_{[j-1:0]}}) = \overline{c_i}\end{aligned} \tag{4.27}$$

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.

4.4.2 Scheme II

The first stage in the architectures of the proposed prefix adder structures involves the creation of kill, propagate and complementary generate signals for individual operand bits using the equations (4.28) to (4.30) respectively.

$$k_i = \overline{a_i + b_i} = \overline{a}_i . \overline{b_i} \tag{4.28}$$

$$\overline{g}_i = (\overline{a_i \cdot b_i}) \tag{4.29}$$

$$p_i = (\overline{a_i \odot b_i}) = (a_i \oplus b_i) \tag{4.30}$$

In the above equations, a_i, b_i represent input operand bits for the adder. The prefix computation in this scheme is responsible for creating group generate and group kill signals. The odd-dot operator and the odd-semi-dot operator use active low group generate and active high group kill inputs to produce active high group generate and active low group kill outputs. The even-dot operator and even-semi-dot operator use active high group generate and active low group kill as inputs to yield active low group generate and active high group kill outputs.

The computation for odd-dot operator is defined by the equation (4.31)

$$\begin{aligned}(G,\overline{K})_{[i:k]} &= (\overline{G},K)_{[i:j]} \blacksquare (\overline{G},K)_{[j-1:k]} \\ &= (\overline{(\overline{G}_{[i:j]}.(K_{[i:j]}+\overline{G}_{[j-1:k]})}, \overline{K_{[i:j]}+K_{[j-1:k]}}) \\ &= (G_{[i:j]}+\overline{K}_{[i:j]}.G_{[j-1:k]},\ \overline{K_{[i:j]}}.\overline{K_{[j-1:k]}})\end{aligned} \quad (4.31)$$

The second cell for the dot operator named even-dot represented by a '■', is defined by the equation (4.32)

$$\begin{aligned}(\overline{G},K)_{[i:k]} &= (G,\overline{K})_{[i:j]} \blacksquare (G,\overline{K})_{[j-1:k]} \\ &= (\overline{G_{[i:j]}+\overline{K}_{[i:j]}.G_{[j-1:k]}},\ \overline{\overline{K}_{[i:j]}.\overline{K}_{[j-1:k]}})\end{aligned} \quad (4.32)$$

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 '●', work based on equations (4.33) and (4.34) respectively.

$$\begin{aligned}(G)_{[i:k]} &= (\overline{G},K)_{[i:j]} \bullet (\overline{G},K)_{[j-1:k]} \\ &= (\overline{(\overline{G}_{[i:j]}.(K_{[i:j]}+\overline{G}_{[j-1:k]}))}) \\ &= (G_{[i:j]}+\overline{K}_{[i:j]}.G_{[j-1:k]}) = c_i\end{aligned} \quad (4.33)$$

$$\begin{aligned}(\overline{G})_{[i:k]} &= (G,\overline{K})_{[i:j]} \bullet (G,\overline{K})_{[j-1:k]} \\ &= (\overline{G_{[i:j]}+\overline{K}_{[i:j]}.G_{[j-1:k]}}) = \overline{c_i}\end{aligned} \quad (4.34)$$

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.

4.4.3 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 complementary generate signal. The equation for the propagate signal is given in equation (4.35)

$$p_i = \overline{g_i + k_i} \qquad (4.35)$$

where g_i and k_i are generate and kill signals for the individual input operand bits a_i and b_i respectively. The rest of the architecture is similar to Scheme II. Since propagate is derived using NOR operation, further reduction in the switching activity is attained, which accounts for a considerable amount of power savings.

4.5 SIMULATION

Simulation for the PPA designs is done using Tanner EDA tool in 180 nm and 130 nm technologies. All the PPA structures are implemented using CMOS logic family. 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 is 1.8 V. For TSMC 130 nm technology, threshold voltages of NMOS and PMOS transistors are around 0.332 V and -0.3499 V respectively and the supply voltage is 1.3 V. The rise time and fall times of the inputs are set to 0.10 ns. The aspect ratio of the MOS transistor devices were chosen such that $\left(\frac{W}{L}\right)_p = 3\left(\frac{W}{L}\right)_n$. The input patterns are switched after every 10 ns. The parameters considered for comparison are power consumption, worst case delay and power-delay product. The various PPA structures are then compared with the number of computation nodes needed for circuit realizations.

4.6 SIMULATION RESULTS AND ANALYSIS

Tables 4.2 to 4.5 list out the structural characteristics of various 8-bit, 16-bit, 32-bit and 64-bit PPAs.

Table 4.2 Structural Comparison of 8-bit Parallel Prefix Adders

Adder Type	Number of Computation Nodes		Logic Depth
	Dot	Semi-Dot	
Brent-Kung	4	7	4
Kogge-Stone	10	7	3
Han-Carlson	5	7	4
Ladner-Fischer	5	7	3
Sklansky	5	7	4
Proposed	4	7	4

Table 4.3 Structural comparison of 16-bit Parallel Prefix Adders

Adder Type	Number of Computation Nodes		Logic Depth
	Dot	Semi-Dot	
Brent-Kung	11	15	6
Kogge-Stone	34	15	4
Han-Carlson	17	15	5
Ladner-Fischer	17	15	4
Sklansky	17	15	5
Proposed	11	15	5

Table 4.4 Structural Comparison of 32-bit Parallel Prefix Adders

Adder Type	**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

Table 4.5 Structural Comparison of 64-bit Parallel Prefix Adders

Adder Type	**Number of Computation Nodes**		**Logic Depth**
	Dot	**Semi-Dot**	
Brent-Kung	57	63	10
Kogge-Stone	316	63	6
Han-Carlson	129	63	7
Ladner-Fischer	129	63	7
Sklansky	129	63	6
Proposed	54	63	11

From the Tables 4.2 to 4.5, it is inferred that the number of dot computation nodes in the proposed architectures for various word lengths is minimal compared to other existing architectures. The logic depth has increased slightly to accomadate the reduction in computation nodes.

4.6.1 Simulation Results of Parallel Prefix Adders using Scheme I

Tables 4.6 to 4.13 list out the performance comparison of various 8-bit, 16-bit, 32-bit and 64-bit PPAs in Scheme I.

Table 4.6 Performance comparison of 8-bit Parallel Prefix Adders in Scheme I using 180 nm Technology

Adder Name	Average Power (µW)	Delay (ns)	Power-Delay Product (X 10^{-15} Joules)
Brent-Kung	51.74536	0.53	27.4250408
Kogge-Stone	64.43130	0.35	22.550955
Han-Carlson	54.65521	0.53	28.9672613
Sklansky	54.04176	0.35	18.914646
Ladner-Fischer	51.74552	0.52	26.9076704
Proposed	47.86712	0.34	16.2748208

Table 4.7 Performance comparison of 8-bit Parallel Prefix Adders in Scheme I using 130 nm Technology

Adder Name	Average Power (µW)	Delay (ns)	Power-Delay Product (X 10^{-15} Joules)
Brent-Kung	14.88472	0.5	7.44236
Kogge-Stone	18.76930	0.34	6.381562
Han-Carlson	15.82269	0.50	7.911345
Sklansky	15.72075	0.32	5.03064
Ladner-Fischer	14.88471	0.49	7.2935079
Proposed	13.43222	0.33	4.4326326

The proposed 8-bit PPA in Scheme I, attains a power saving of 7.49% and 9.75% in 180 nm and 130 nm technologies respectively when compared with Brent-Kung adder. This is due to the fact that a large number of inverters are eliminated in the proposed architecture. The results show that the proposed 8-bit PPA can achieve about 27% to 30% reduction in power-delay product when compared with Kogge-Stone adder. This is due to the fact that the total number of dot computation nodes have reduced by about 35.29%.

Table 4.8 Performance Comparison of 16-bit Parallel Prefix Adders in Scheme I using 180 nm Technology

Adder Name	**Average Power (µW)**	**Delay (ns)**	**Power-Delay Product (X 10^{-15} Joules)**
Brent-Kung	102.2749	0.88	90.001912
Kogge-Stone	140.9152	0.53	74.685056
Han-Carlson	111.5113	0.84	93.669492
Sklansky	110.0999	0.50	55.04995
Ladner-Fischer	101.8946	0.84	85.591464
Proposed	99.1801	0.51	50.581851

Table 4.9 Performance Comparison of 16-bit Parallel Prefix Adders in Scheme I using 130 nm Technology

Adder Name	**Average Power (µW)**	**Delay (ns)**	**Power-Delay Product (X 10^{-15} Joules)**
Brent-Kung	30.09975	0.68	20.46783
Kogge-Stone	41.62221	0.39	16.2326619
Han-Carlson	33.01376	0.68	22.4493568
Sklansky	32.5652	0.42	13.677384
Ladner-Fischer	30.06299	0.51	15.3321249
Proposed	29.70381	0.44	13.0696764

The proposed 16-bit PPA in Scheme I, offers about 2.66% power savings when compared with Ladner-Fischer adder in 180 nm technology. Further the proposed 16-bit PPA, achieves about 35% to 40% improvement in speed when compared to Han-Carlson adder. This benefit is attained because the Proposed PPA employs four computation nodes, which cause improvement in critical path delay.

Table 4.10 Performance Comparison of 32-bit Parallel Prefix Adders in Scheme I using 180 nm Technology

Adder Name	Average Power (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

Table 4.11 Performance Comparison of 32-bit Parallel Prefix Adders in Scheme I using 130 nm Technology

Adder Name	Average Power (µW)	Delay (ns)	Power-Delay Product (X 10^{-15} Joules)
Brent-Kung	62.23515	0.88	54.766932
Kogge-Stone	104.6994	0.62	64.913628
Han-Carlson	71.16859	0.75	53.3764425
Sklansky	81.28151	0.53	43.0792003
Ladner-Fischer	64.27141	0.67	43.0618447
Proposed	59.4531	0.65	38.644515

The proposed 32-bit PPA in Scheme I has reduced the critical path delay by about 6.59% and 2.98% in 180 nm and 130 nm technologies when compared with Ladner-Fischer adder. There is a reduction of 5.92% and 11.62% in power-delay product for the proposed 32-bit PPA compared to Sklansky adder in 180 nm and 130 nm technologies respectively. This merit is achieved due to reduction in the maximum fan-out of the computation nodes for the proposed PPA.

Table 4.12 Performance Comparison of 64-bit Parallel Prefix Adders in Scheme I using 180 nm Technology

Adder Name	**Average Power (µW)**	**Delay (ns)**	**Power-Delay Product (X 10^{-15} Joules)**
Brent-Kung	450.1228	1.14	513.139992
Kogge-Stone	980.0381	0.87	852.63314
Han-Carlson	524.17838	0.97	508.453028
Sklansky	679.462	0.96	652.28352
Ladner-Fischer	499.884	0.97	484.88748
Proposed	404.7353	0.94	380.451182

Table 4.13 Performance Comparison of 64-bit Parallel Prefix Adders in Scheme I using 130 nm Technology

Adder Name	**Average Power (µW)**	**Delay (ns)**	**Power-Delay Product (X 10^{-15} Joules)**
Brent-Kung	131.3161	0.97	127.37661
Kogge-Stone	296.2993	0.79	234.076447
Han-Carlson	155.14752	0.89	138.08129
Sklansky	219.8533	0.75	164.88998
Ladner-Fischer	143.9679	0.87	125.252073
Proposed	129.6077	0.84	108.870468

Tables 4.12 and 4.13 indicate that there is a reduction of 58.7% of power and 56.25% of power in the proposed 64-bit PPA compared to Kogge-Stone adder for 180 nm and 130 nm technologies respectively. This drastic improvement in power has occurred for the proposed PPA because of 83% reduction in dot computation hardware. Also results reveal that the proposed 64-bit PPA offers 3.1% and 3.4% reduction in critical path delay when compared with Ladner-Fischer adder.

4.6.2 Simulation Results of Parallel Prefix Adders using Scheme II

Tables 4.14 to 4.21 list out the performance comparison of various 8-bit, 16-bit, 32-bit and 64-bit PPAs in Scheme II.

Table 4.14 Performance Comparison of 8-bit Parallel Prefix Adders in Scheme II using 180 nm Technology

Adder Name	Average Power (µW)	Delay (ns)	Power-Delay Product (X 10^{-15} Joules)
Brent-Kung	49.8231	0.65	32.385015
Kogge-Stone	61.50808	0.44	27.0635552
Han-Carlson	51.78471	0.62	32.1065202
Sklansky	52.45249	0.49	25.7017201
Ladner-Fischer	50.2359	0.57	28.634463
Proposed	47.3812	0.51	24.164412

Table 4.15 Performance Comparison of 8-bit Parallel Prefix Adders in Scheme II using 130 nm Technology

Adder Name	**Average Power (µW)**	**Delay (ns)**	**Power-Delay Product (X 10^{-15} Joules)**
Brent-Kung	14.63704	0.59	8.6358536
Kogge-Stone	17.85775	0.36	6.42879
Han-Carlson	14.94024	0.52	7.7689248
Sklansky	15.3115	0.39	5.971485
Ladner-Fischer	14.2907	0.53	7.574071
Proposed	13.3921	0.42	5.624682

From Tables 4.14 and 4.15 it is inferred that the proposed 8-bit PPA in Scheme II, achieves 17.7% and 19.2% improvement in critical path delay compared to Han-Carlson adder for 180 nm and 130 nm technologies. This benefit is due to alternate cascading of odd and even indices computation cells in the proposed PPA which eliminate the inverters in the path.

Table 4.16 Performance Comparison of 16-bit Parallel Prefix Adders in Scheme II using 180 nm Technology

Adder Name	**Average Power (µW)**	**Delay (ns)**	**Power-Delay Product (X 10^{-15} Joules)**
Brent-Kung	100.3460	0.82	82.28372
Kogge-Stone	139.7753	0.58	81.069674
Han-Carlson	109.7427	0.91	99.865857
Sklansky	108.9231	0.79	86.049249
Ladner-Fischer	100.9433	0.97	97.915001
Proposed	97.5015	0.82	79.95123

Table 4.17 Performance Comparison of 16-bit Parallel Prefix Adders in Scheme II using 130 nm Technology

Adder Name	**Average Power (µW)**	**Delay (ns)**	**Power-Delay Product (X 10^{-15} Joules)**
Brent-Kung	29.10129	0.77	22.4079933
Kogge-Stone	41.31562	0.52	21.4841224
Han-Carlson	32.13299	0.74	23.7784126
Sklansky	31.68943	0.62	19.6474466
Ladner-Fischer	29.65798	0.71	21.0571658
Proposed	27.0368	0.66	17.844288

Comparing the results of the proposed 16-bit PPA in Scheme II with Ladner-Fischer adder, about 3.4% and 8.83% power savings can be achieved for 180 nm and 130 nm technologies respectively. Also the critical path delay for the proposed 16-bit PPA is reduced by 9.8% and 10.81% compared to Han-Carlson adder for 180 nm and 130 nm technologies.

Table 4.18 Performance Comparison of 32-bit Parallel Prefix Adders in Scheme II using 180 nm Technology

Adder Name	**Average Power (µW)**	**Delay (ns)**	**Power-Delay Product (X 10^{-15} Joules)**
Brent-Kung	197.0751	1.29	254.226879
Kogge-Stone	350.2907	0.89	311.758723
Han-Carlson	233.8426	1.07	250.211582
Sklansky	263.7364	0.92	242.637488
Ladner-Fischer	211.6786	1.13	239.196818
Proposed	194.261	1.09	211.74449

Table 4.19 Performance Comparison of 32-bit Parallel Prefix Adders in Scheme II using 130 nm Technology

Adder Name	Average Power (µW)	Delay (ns)	Power-Delay Product (X 10^{-15} Joules)
Brent-Kung	61.30189	0.96	58.8498144
Kogge-Stone	99.05527	0.69	68.3481363
Han-Carlson	68.65387	0.91	62.4750217
Sklansky	72.4006	0.82	59.368492
Ladner-Fischer	62.27339	0.93	57.9142527
Proposed	57.18413	0.85	48.6065105

Tables 4.18 and 4.19 reveal that, the proposed 32-bit PPA in Scheme II offers a benefit of 1.41% and 6.71% power reduction and 15.5% and 11.45% improvement in speed over Brent-Kung adder for 180 nm and 130 nm technologies respectively. The logic depth has increased by one for the proposed PPA compared to Brent-Kung structure. It is still possible to reap a benefit in speed for the proposed PPA due to the presence of four different computational cells. Further the power savings is due to 11% reduction in dot computation hardware.

Table 4.20 Performance Comparison of 64-bit Parallel Prefix Adders in Scheme II using 180 nm Technology

Adder Name	Average Power (µW)	Delay (ns)	Power-Delay Product (X 10^{-15} Joules)
Brent-Kung	433.6042	1.61	698.102762
Kogge-Stone	963.2994	0.96	924.767424
Han-Carlson	507.438442	1.35	685.041896
Sklansky	651.4289	1.12	729.60037
Ladner-Fischer	476.2768	1.48	704.889664
Proposed	376.035	1.36	511.4076

Table 4.21 Performance Comparison of 64-bit Parallel Prefix Adders in Scheme II using 130 nm Technology

Adder Name	Average Power (µW)	Delay (ns)	Power-Delay Product (X 10^{-15} Joules)
Brent-Kung	127.5177	1.25	159.3971
Kogge-Stone	257.4732	0.92	236.87534
Han-Carlson	148.9782	0.97	144.508854
Sklansky	174.4852	0.97	169.250644
Ladner-Fischer	133.855	1.03	137.87065
Proposed	125.478	0.97	121.71366

The circuit latency for the proposed 64-bit PPA in Scheme II has improved by 8.1% and 5.82% over Ladner-Fischer adder for 180 nm and 130 nm technologies respectively. Further, nearly 45% to 49% improvement in power-delay product can be achieved for the proposed 64-bit PPA compared to Kogge-Stone adder. The dot computation hardware has reduced by 83% nearly, hence it provides a significant amount of power reduction and a drastic improvement in power-delay product.

4.6.3 Simulation Results of Parallel Prefix Adders using Scheme III

Tables 4.22 to 4.29 list out the performance comparison of various 8-bit, 16-bit, 32-bit and 64-bit PPAs in Scheme III.

Table 4.22 Performance Comparison of 8-bit Parallel Prefix Adders in Scheme III using 180 nm Technology

Adder Name	Average Power (µW)	Delay (ns)	Power-Delay Product (X 10^{-15} Joules)
Brent-Kung	47.80145	0.72	34.41704
Kogge-Stone	60.12352	0.47	28.258054
Han-Carlson	49.08213	0.64	31.4125632
Sklansky	49.85631	0.57	28.41809
Ladner-Fischer	47.82076	0.64	30.605286
Proposed	46.18824	0.58	26.78917

Table 4.23 Performance Comparison of 8-bit Parallel Prefix Adders in Scheme III using 130 nm Technology

Adder Name	Average Power (µW)	Delay (ns)	Power-Delay Product (X 10^{-15} Joules)
Brent-Kung	13.60773	0.65	8.8450245
Kogge-Stone	17.11900	0.44	7.53236
Han-Carlson	13.93726	0.56	7.8048656
Sklansky	14.29659	0.52	7.4342268
Ladner-Fischer	13.84081	0.61	8.4428941
Proposed	13.23959	0.54	7.1493786

The critical path delay for the proposed 8-bit PPA in Scheme III has reduced by 9.37% and 5.26% compared to Han-Carlson adder for 180 nm and 130 nm technologies respectively. The proposed PPA eliminates several inverters along the critical path to achieve improvement in speed.

Table 4.24 Performance Comparison of 16-bit Parallel Prefix Adders in Scheme III using 180 nm Technology

Adder Name	Average Power (µW)	Delay (ns)	Power-Delay Product (X 10^{-15} Joules)
Brent-Kung	95.02580	0.96	91.22476
Kogge-Stone	136.5688	0.61	83.30696
Han-Carlson	103.6214	0.95	98.44033
Sklansky	107.5552	0.86	92.4974
Ladner-Fischer	96.27817	1.13	108.794332
Proposed	88.93544	0.90	80.041896

Table 4.25 Performance Comparison of 16-bit Parallel Prefix Adders in Scheme III using 130 nm Technology

Adder Name	Average Power (µW)	Delay (ns)	Power-Delay Product (X 10^{-15} Joules)
Brent-Kung	27.25826	0.91	24.80501
Kogge-Stone	39.65700	0.59	23.7976
Han-Carlson	30.27793	0.78	23.6167
Sklansky	30.90898	0.68	21.01810
Ladner-Fischer	27.82156	0.96	26.70869
Proposed	25.7084	0.75	18.92445

The circuit latency of the proposed 16-bit PPA in Scheme III has improved by about 20% in comparison with Ladner-Fischer adder. The power-delay product for the proposed 16-bit PPA in comparison with Brent Kung adder, reveal that there is an improvement of 12.25% and 23.70% for 180 nm and 130 nm technologies respectively. The delay for the proposed PPA has improved because the logic depth has reduced by one compared to Brent-Kung structure and a small amount in power reduction is due to elimination of inverters, due to alternate cascading of computation cells. Hence power-delay product has improved compared to Brent-Kung structure.

Table 4.26 Performance Comparison of 32-bit Parallel Prefix Adders in Scheme III 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

Table 4.27 Performance Comparison of 32-bit Parallel Prefix Adders in Scheme III using 130 nm Technology

Adder Name	Average Power (μW)	Delay (ns)	Power-Delay Product (X 10^{-15} Joules)
Brent-Kung	57.03546	1.05	59.8872
Kogge-Stone	97.4262	0.74	72.0953
Han-Carlson	65.703108	0.97	63.702014
Sklansky	70.42346	0.96	67.6065
Ladner-Fischer	60.1703	1.07	64.38221
Proposed	54.18525	0.90	51.493226

The Proposed 32-bit PPA in Scheme III offers 16.79% and 22.85% power savings in comparison with Sklansky adder for 180 nm and 130 nm technologies respectively. This significant power savings is due to 30.3% reduction in dot computation hardware for the proposed PPA compared to Sklansky adder. Further in Sklansky adder, the maximum fan-out increases drastically along the main diagonal. This introduces latency in the Sklansky adder. Proposed PPA offers a controlled maximum fan-out of 5.

Table 4.28 Performance Comparison of 64-bit Parallel Prefix Adders in Scheme III using 180 nm Technology

Adder Name	Average Power (µW)	Delay (ns)	Power-Delay Product (X 10^{-15} Joules)
Brent-Kung	432.633	1.68	726.82344
Kogge-Stone	938.916	1.13	1060.97508
Han-Carlson	483.36274	1.52	734.71136
Sklansky	489.0781	1.55	758.07106
Ladner-Fischer	462.80	1.65	763.62
Proposed	346.7805	1.52	527.10636

Table 4.29 Performance Comparison of 64-bit Parallel Prefix Adders in Scheme III using 130 nm Technology

Adder Name	Average Power (µW)	Delay (ns)	Power-Delay Product (X 10^{-15} Joules)
Brent-Kung	125.0644	1.39	173.83951
Kogge-Stone	247.4625	0.97	240.038625
Han-Carlson	143.23277	1.12	160.420708
Sklansky	162.6781	1.16	188.706596
Ladner-Fischer	131.1712	1.25	163.964
Proposed	115.723	1.09	126.13807

The propagation delay is reduced by 9.52% and 21.58% for the proposed 64-bit adder in Scheme III relative to Brent-Kung adder for 180 nm and 130 nm technologies respectively. Comparison of the proposed 64-bit adder with Han-Carlson structure, reveal that there is 28.66% and 21.37% reduction in power-delay product for 180 nm and 130 nm technologies. The benefit in power reduction for the proposed PPA is due to elimination of inverters along the path and reduction in dot computation hardware. In the power comparison and power-delay product comparison bar charts shown below, BK represents Brent-Kung adder, KS represents Kogge-Stone adder, HC represents Han-Carlson adder, SK represents Sklansky adder and LF represents Ladner-Fischer adder.

Figures 4.10 and 4.11 show the power comparison of various 32-bit PPA structures in Scheme I, Scheme II and Scheme III respectively for 180 nm and 130 nm technologies respectively.

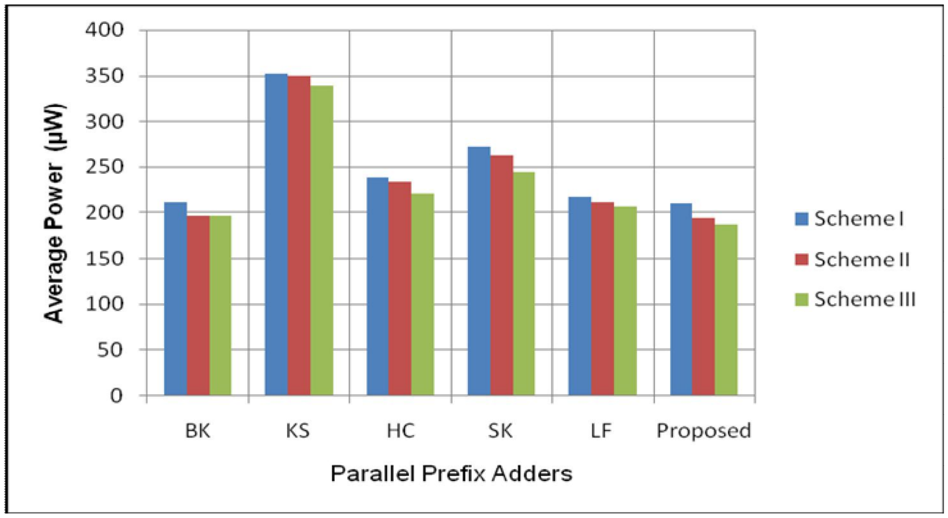

Figure 4.10 Power Comparison of the 32-bit Parallel Prefix Adders in Three Schemes for 180 nm Technology

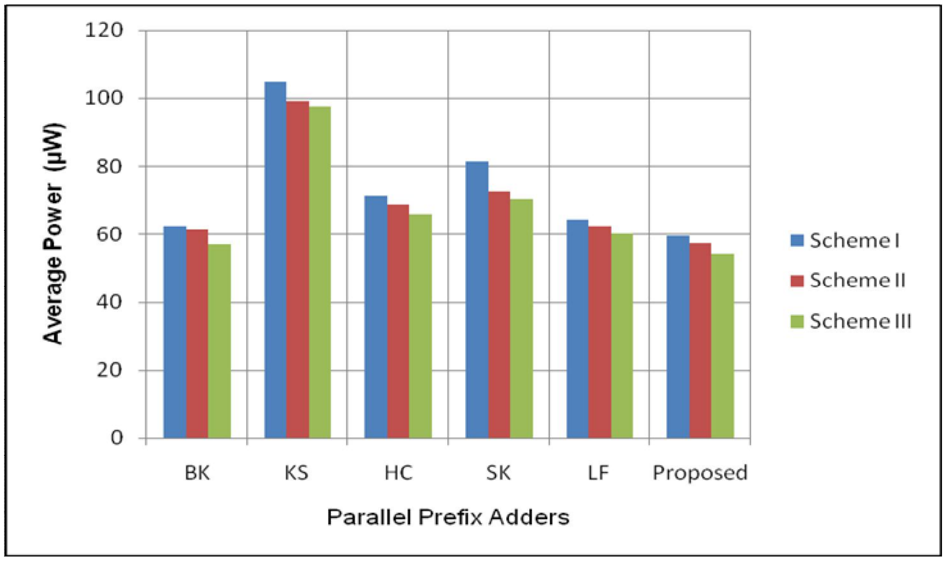

Figure 4.11 Power Comparison of the 32-bit Parallel Prefix Adders in Three Schemes for 130 nm Technology

Figures 4.12 and 4.13 show the power-delay product comparison of various 32-bit PPA structures in Scheme I, Scheme II and Scheme III respectively for 180 nm and 130 nm technologies respectively.

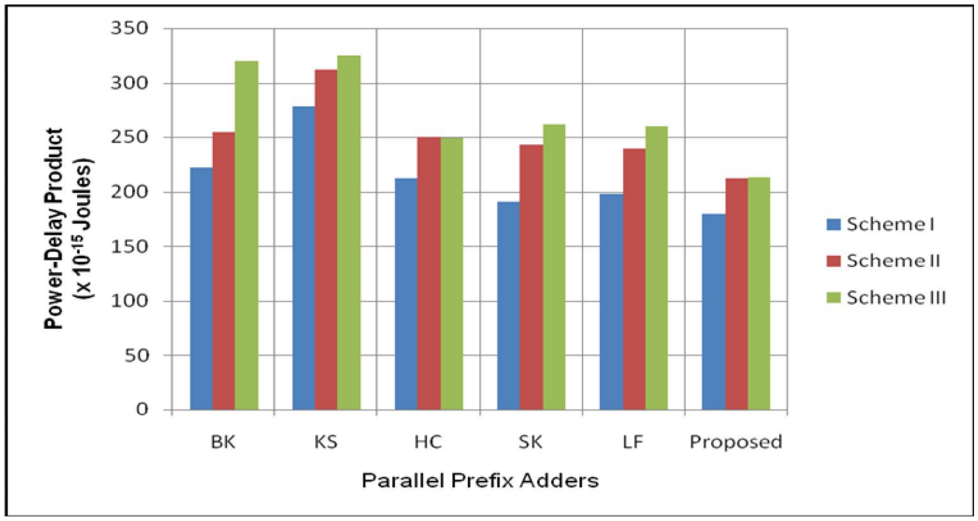

Figure 4.12 Power-Delay Product Comparison of the 32-bit Parallel Prefix Adders in three Schemes for 180 nm Technology

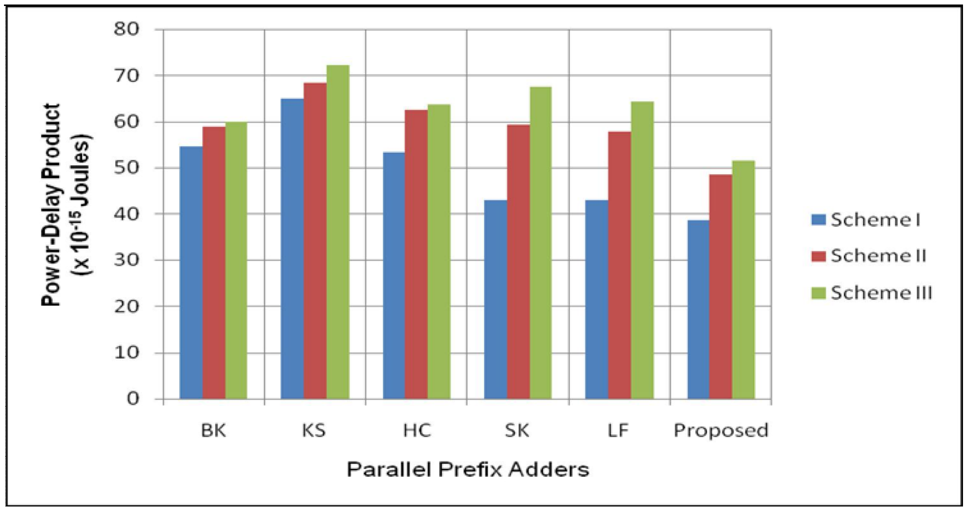

Figure 4.13 Power-Delay Product Comparison of the 32-bit Parallel Prefix Adders in three Schemes for 130 nm Technology

Figures 4.14 and 4.15 show the power comparison of various 64-bit PPA structures in Scheme I, Scheme II and Scheme III respectively for 180 nm and 130 nm technologies respectively.

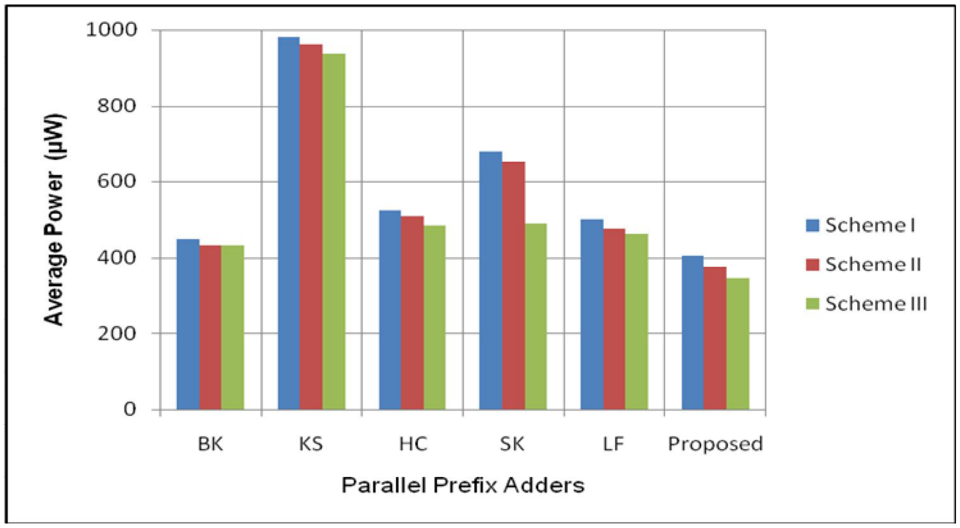

Figure 4.14 Power Comparison of the 64-bit Parallel Prefix Adders in Three Schemes for 180 nm Technology

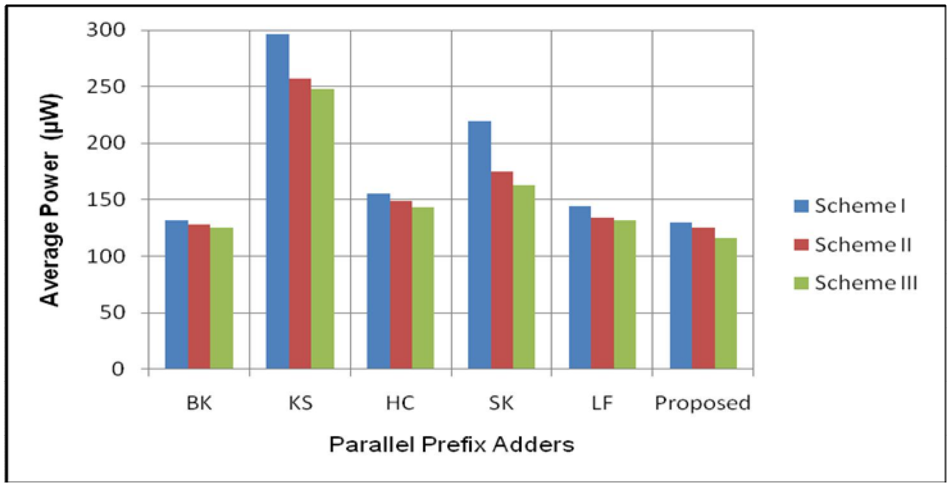

Figure 4.15 Power Comparison of the 64-bit Parallel Prefix Adders in Three Schemes for 130 nm Technology

Figures 4.16 and 4.17 show the power-delay product comparison of various 64-bit PPA structures in Scheme I, Scheme II and Scheme III respectively for 180 nm and 130 nm technologies respectively.

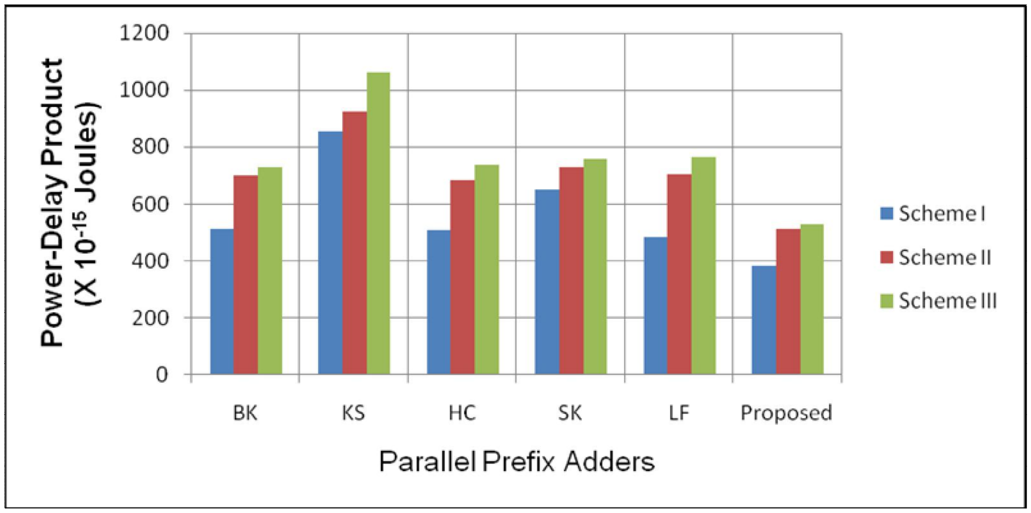

Figure 4.16 Power-Delay Product Comparison of the 64-bit Parallel Prefix Adders in Three Schemes for 180 nm Technology

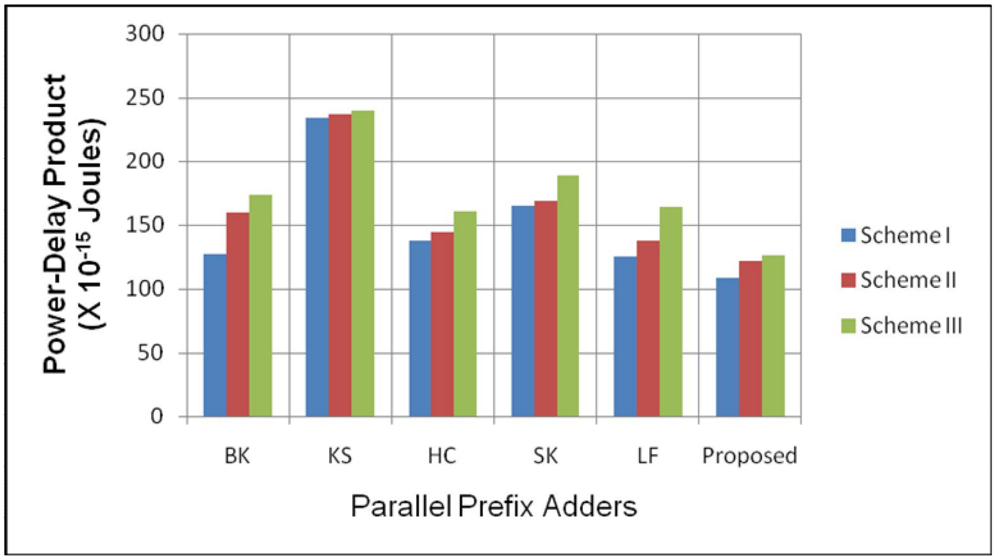

Figure 4.17 Power-Delay Product Comparison of the 64-bit Parallel Prefix Adders in Three Schemes for 130 nm Technology

The delay and the power-delay product are minimal in Scheme I. This is because the activity factor for propagate signal being at logic ‘1’ and at logic ‘0’ is 50% in the Pre-processing Stage. In the prefix computation stage, Scheme I employs computation of group generate and group propagate as opposed to computation of group generate and group kill bar for Scheme II and Scheme III.

It is inferred that the power consumption is minimal in Scheme III for all the PPA structures. This is because kill bar signal in the preprocessing remains at logic ‘1’ for 75% and at logic ‘0’ for 25%. Further in Scheme III, Propagate signal is derived using generate and kill signals of the individual operand bits. This accounts for reduction in the number of transistors count, which leads to further reduced power dissipation. Critical path delay in Scheme III is higher since propagate signal in pre-processing stage is derived from kill and generate signals using a NOR gate.

The amount of power saving gradually increases for the proposed architectures when the size of the prefix adder grows. It is also observed that the power-delay product of the proposed architecture is minimal when compared to other PPAs.

Table 4.30 Comparison of the Proposed 64-bit Parallel Prefix Adders with Previous Works

PPA	Technology / Supply Voltage	Power (mW)	Delay (ns)
Zhanpeng Jin et al (2005)	180 nm / 2.5 V	-	1.21
Yan Sun et al (2006)	180 nm / 1.8 V	13.8	0.59
Proposed Scheme I	180 nm / 1.8 V	0.404	0.94
Proposed Scheme II	180 nm / 1.8 V	0.376	1.36
Proposed Scheme III	180 nm / 1.8 V	0.346	1.52

4.7 CONCLUSION

Novel hybrid PPA architectures for 8-bit, 16-bit, 32-bit and 64-bits is proposed that offers minimal power dissipation and least power-delay product. The architectures are realized using three Schemes. Scheme I provides least delay and least power-delay product for all the PPA architectures. Scheme II offers moderate power and delay for all the PPA architectures. Scheme III provides least power dissipation in all the PPA architectures. The proposed PPA in Scheme I achieves 3% to 7% power savings and 15% to 35% improvement in speed compared with Brent Kung adder which has nearly same logic depth and number of computation nodes. The proposed PPA architecture in Scheme I attains 30% to 45% power savings at the expense of 5% to 10% delay penalty compared to Kogge-Stone adder which has maximum computation nodes.

The proposed PPA in Scheme II achieves 3% to 15% power savings and 15% to 28% improvement in performance when compared to Brent Kung adder. The proposed PPA architecture in Scheme II attains 35% to 52% power savings at the expense of 10% to 27% delay penalty compared to Kogge-Stone adder. The Proposed PPA in Scheme III offer 6% to 8% power savings and 10% to 25% improvement in critical path delay compared to a Brent-Kung adder. The proposed PPA architecture when compared to Kogge-Stone adder in Scheme III offers 45% to 55% power reduction at the expense of 17% to 25% delay penalty.