Design of Efficient VLSI Arithmetic Circuits

Thesis submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

in ELECTRONICS AND COMMUNICATION ENGINEERING

by

Sreehari Veeramachaneni
2006 42003
srihari@research.iiit.ac.in

International Institute of Information Technology, Hyderabad
(Deemed to be University)
Hyderabad – 500 032, INDIA
June 2015

International Institute of Information Technology

Hyderabad, India

CERTIFICATE

It is certified that the work contained in the thesis, titled “Design of Efficient VLSI Arithmetic Circuits" by Sreehari Veeramachaneni, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____

Date

Adviser: Prof. M. B. Srinivas

Acknowledgments

I am most grateful to my thesis advisor Prof. M. B. Srinivas for his thoughtful guidance and warm encouragement without whose help it would have been impossible for me to complete this thesis. With his enthusiasm, his inspiration, and his great efforts to explain things clearly and simply, he helped to make subject fun for me. Throughout my thesis-writing period, he provided encouragement, sound advice, good teaching, good company, and many good ideas. I would have been lost without him.

I wish to thank Prof. Rajeev Sangal, Director, IIT-BHU, Prof. P. J. Narayanan, Director, IIITH, Prof. Kamalakar Karlapalem, Dean (Academics), Prof. Vasudeva Varma, Dean (R & D), Prof. Govindarajulu and all faculty members for their support and encouragement throughout my research.

I am indebted to my friends and colleagues from IIIT Hyderabad and BITS-Pilani, Hyderabad Campus for providing a stimulating and fun environment in which to learn and grow.

Lastly, and most importantly, I wish to thank my parents. They raised me, supported me, taught me, and loved me. To them I dedicate this thesis.

Abstract

Arithmetic and Logic Unit (ALU) is a critical component of any CPU. In ALU, adders play a major role not only in addition but also in performing many other basic arithmetic operations like subtraction, multiplication, etc. Thus realizing an efficient adder is required for better performance of an ALU and therefore the processor. Research started in late 1950s on designing efficient adder algorithms and their hardware implementation. Many designs based on serial and parallel structures have been proposed to optimize different parameters from time to time.

The first contribution of this thesis is the development of an efficient adder architecture that addresses the problems for higher bit operand lengths like fan-out, wiring complexity, etc.

Another important element in an ALU after adder is a multiplier. In multipliers, for reducing partial products and computing final result, multi-operand adders and fast adders are required. A special structure known as counters/compressors are typically used for designing multi-operand adders. Counters are multi-input, multi-output combinational logic circuits, which determine the number of logic 1's in their input vectors, and generate a binary coded output vector that corresponds to this number. Large parallel counters like (15, 4), (32, 5), etc. can be constructed using this small counter and similar approach can be adopted in the case of compressors. The second contribution of the thesis is development of efficient counters and compressors for better performance of multiplier.

Apart from adders and multipliers in arithmetic units, elements like, incrementer/decrementer (INC/DEC) also play a major role in an ALU and also in address generation unit. A loop algorithm, for example, often needs a increment/decrement. These operations can be realized using adders but with a cost in terms of power and area. Therefore, standalone designs or unified designs for INC/DEC are required for low power applications. The third contribution of this thesis is the design of a multi-functional INC/DEC/2's complement/Priority encoder circuit. A design for binary INC/DECs is presented that is efficient in terms of speed without compromising on power.

The need to have hardware support for decimal arithmetic is increasing in recent years because of the growth in decimal data processing in commercial, financial and internet-based applications. To facilitate binary computations on the same hardware, a reconfigurable approach needs to be adopted. The fourth contribution of this thesis is the design of a new architecture for efficient Binary Coded Decimal (BCD) addition/subtraction that can be configured to perform binary addition/subtraction also. The architecture has been designed keeping in view the signed magnitude format where the adder logic itself detects the larger operand and carries out corresponding operations.

Finally, novel versions of two widely used arithmetic blocks i.e., multiplier and floating point adder, are designed. Efficient and proven basic functional units described above are used to implement these blocks. Simulations of these blocks have been carried out and comparisons made with existing designs that clearly demonstrate the efficiency of proposed units. Finally, a segment of a core of a processor is designed with incorporating all the above elements resulting in an efficient architecture.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

Contents

1.1 Motivation

In a microprocessor or a digital signal processor (DSP), data path plays a prominent role since performance metrics like the die-area, speed of operation, power dissipation etc., depend directly on the efficiency of data-path. As is known, core of the data path involves complex computations like addition, subtraction, multiplication and division, etc. Thus, realizing efficient hardware units for these computations, which directly affect the performance of data path, is of prime importance.

The most executed operation in the data path is addition, which requires a binary adder that adds two given numbers. Adders also play a vital role in more complex computations like multiplication, division and decimal operations. Hence, an efficient implementation of binary adder is crucial to an efficient data path. Relatively significant work has been done in proposing and realizing efficient adder circuits for binary addition as described in the next chapter. However, as the technology is scaling down new design issues like fan-out and wiring complexity are appearing in the front-line. These issues are addressed to some extent by new adder architectures known as sparse adders. As operand size increases, sparse adders also suffer from above design issues that are becoming vital as they have direct impact on the performance of an adder. Thus, there is an urgent need to develop alternative sparse adder architectures which can address these design issues.

The next most important block in data-path after adder is the multiplier, which is also very crucial in ASICs and DSPs. High speed multipliers reported in literature use parallel

multiplier architectures that employ counters/compressors along with adders as basic building blocks. Counters are multi-input, multi-output combinational logic circuits that determine the number of logic '1s' in their input vectors and generate a binary coded output vector that corresponds to this number. A counter differs from a compressor in that compressors have carry inputs and carry outputs in addition to the "normal" inputs and outputs which counters do not have. As these blocks lie directly within the critical path of a given design, thus dictating the overall circuit performance, there is an urgent need to design and validate new high speed/low power counters and compressors

Further, some of the recursive arithmetic operations that appear in processors/controllers other than addition and multiplication are increment and decrement operations. The increment and decrement operations count up or down by one step which can be performed by incrementer/decrementer (INC/DEC) block. This block also finds its application in address generation unit in processors and frequency dividers. The architectures of binary INC/DEC block are mainly based on adder/subtractor, counter or carry look-ahead adder.

Finally, despite the widespread use of binary arithmetic, decimal computation remains essential for many applications. Not only is it required whenever numbers are presented for human inspection, but is also often a necessity when fractions are involved. Decimal fractions are pervasive in human endeavors, yet most cannot be represented by binary fractions. Still, the major consideration while implementing Binary Coded Decimal (BCD) arithmetic will be to enhance its speed as much as possible while facilitating even binary applications on the same hardware. There are different architectures that support BCD as well as binary operations on the same hardware. However, when signed computations are required, the existing architectures, use 10's or 9's complement to implement subtraction in BCD. This introduces extra latency in the conversion process, which requires an architecture that can reduce/eliminate the correction latency in BCD adders.

1.2 Objectives of the thesis

With the above modifications, the following objectives are proposed to be addressed in this thesis

- Efficient realization of higher operand bit adders (for 32-bit and above).
- Realization of efficient counters/compressors for high speed parallel multiplication.
- Design of high performance stand-alone blocks like incrementer, decrementer etc.

- Implementation a unified Binary/BCD adder with improved performance.
- Demonstration of efficient arithmetic section of an ALU using the proposed basic units

1.3 Organization of the thesis

As the adder is the basic building block in designing most of the arithmetic circuits, chapter 2 discusses in detail various types of adder architectures and their realization in hardware. A high performance sparse adder architecture (for 32-bit and above) is proposed and studied in detail.

Multiplication finds a wide range of usage in signal processing hardware implementations. In Chapter 3, a special block known as counter/compressor is used in partial product reduction tree in multipliers and analyzed. An efficient counter/compressor block is proposed which makes use of signals and their complements available in CMOS implementation. A generalized n-bit counter is proposed which can be customized to any operand bit size.

The branching and interrupt instructions in any microprocessor need the help of special hardware blocks. In Chapter 4, dedicated hardware blocks like Incrementer/Decrementer, 2's complementer, priority encoder etc., are analyzed and a multi-functional block is proposed which can perform all the above operations using the same hardware.

The emphasis on error free arithmetic is increasing day by day and decimal arithmetic circuits are slowly taking the center stage. In Chapter 5, efficient decimal arithmetic hardware implementation that can perform both signed and unsigned arithmetic is proposed. The implementation reduces the hardware and thereby propagation delay with the proposed end around carry method of subtraction. The same hardware that implements the decimal arithmetic can be used for binary arithmetic without any degradation in performance.

The usage of the above-proposed arithmetic blocks in a multiplier and floating point adder is studied in Chapter 6 and a segment of a processor core is designed with the proposed arithmetic units in Chapter 7. Finally, the scope for further work is suggested in Chapter 8.

Chapter 2
Design and Implementation of Efficient Adders

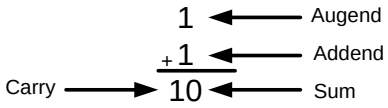
Contents

2.1 Introduction

Arithmetic and Logic Unit (ALU) is a critical element in any CPU. In ALU, adders play a major role not only for addition but also in performing many other basic arithmetic operations like subtraction, multiplication, increment / decrement etc. Thus, realizing an efficient adder is required for better performance of a processor in general and ALU in particular [1-6]. Research into design of efficient adder algorithms for hardware implementation of Very Large Scale Integrated (VLSI) arithmetic circuits started in late 1950s. Many designs based on serial and parallel structures have been proposed to optimize different parameters from time to time [5].

Binary addition consists of four possible elementary operations, which are

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 10$$



The first three operations produce only a ‘Sum’ whose length is one digit, but when both augend and addend bits are equal to 1, the binary sum consists of two digits. The higher significant bit of this result is called a ‘Carry’. A combinational circuit that performs the addition of two bits is called a half-adder while the one that performs the addition of three bits is known as a full-adder [5].

2.2 Review of Existing Adder Designs

Adders can be broadly classified into following four classes [5]:

- Ripple Carry Adder (RCA)
- Carry Select Adder (CSA)
- Carry Look-Ahead Adder (CLA)
- Parallel Prefix- based Adder (PPA)

2.2.1 Ripple Carry Adder

A full adder (FA) is a combinational circuit that takes two operand bits and a carry bit, say A,B and C_i respectively, as inputs and gives Sum (S) and Carry bit (C_o) as outputs. This output Carry bit C_o will serve as input Carry bit for the successive full adder. The combinational circuit follows the Boolean equations 2.1 and 2.2 mentioned below to implement a full adder and the gate level implementation of the same is shown in Fig 2.1. A simple implementation of higher operand adder for two operands A and B is carried out by cascading n of these basic full adder units and is known as a ripple carry adder.

A simple 4-bit ripple carry adder is shown in Fig 2.2. The design of this adder is simple and implementation is easy, but it suffers from serious delay issues. This is because the next stage full adder needs to wait for Carry bit from the previous stage FA. By inspecting the FA shown in Fig 2.1 it can be observed that the gate delay from C_{in} to C_o is 2 gates. Therefore, each full adder contributes to a 2-gate delay in the process of rippling the carry [1-6].

$$C_o = (A.B) + \left(C_i\ .(A \oplus B)\right) = (A.B) + (B.C_i) + (C_i.A) \quad (2.1)$$

$$S = (A \oplus B) \oplus C_i \quad (2.2)$$

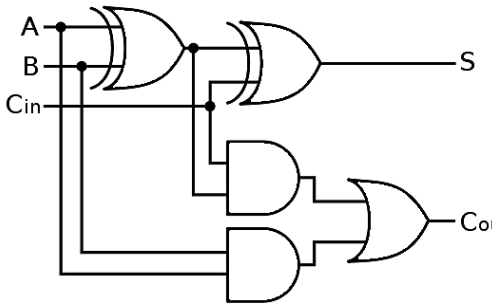


Figure 2.1 One - Bit Full Adder

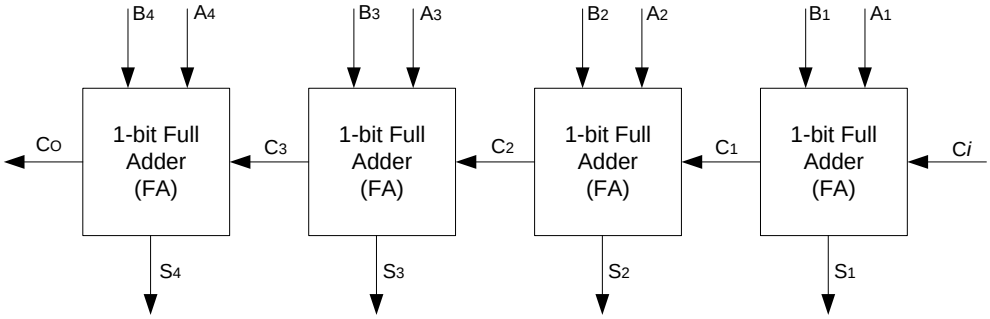


Figure 2.2 Four-Bit Ripple Carry Adder

2.2.2 Carry Select Adder (CSA)

Ripple carry adder waits for the input carry (Ci) and then computes the 'Sum' and the Carry out (Co) thus increasing its delay. In order to reduce the delay, carry select adder is introduced, which pre-computes the 'Sum' and 'Co' for the two possible cases i.e. Ci = 0 and Ci = 1. The calculated Sum is given to a multiplexer, which chooses the correct output depending upon the Ci coming from the previous stage. This pre-computation of Sum reduces the delay of rippling of Carry which is limited to only one multiplexer for each stage. Figure 2.3 below gives the gate level diagram of a 16- bit carry select adder. In this, each 4- bit adder is a bit ripple carry adder. Carry select adder uses more hardware even though it gives less delay compared to ripple carry adder. Thus, there is a tradeoff between area, power and delay between different adders[1-6].

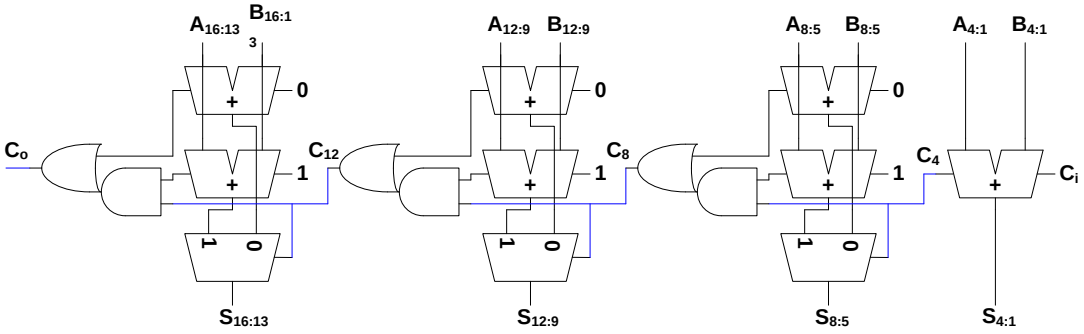


Figure 2.3 16-bit Carry Select Adder

2.2.3 Carry Look-Ahead Adder

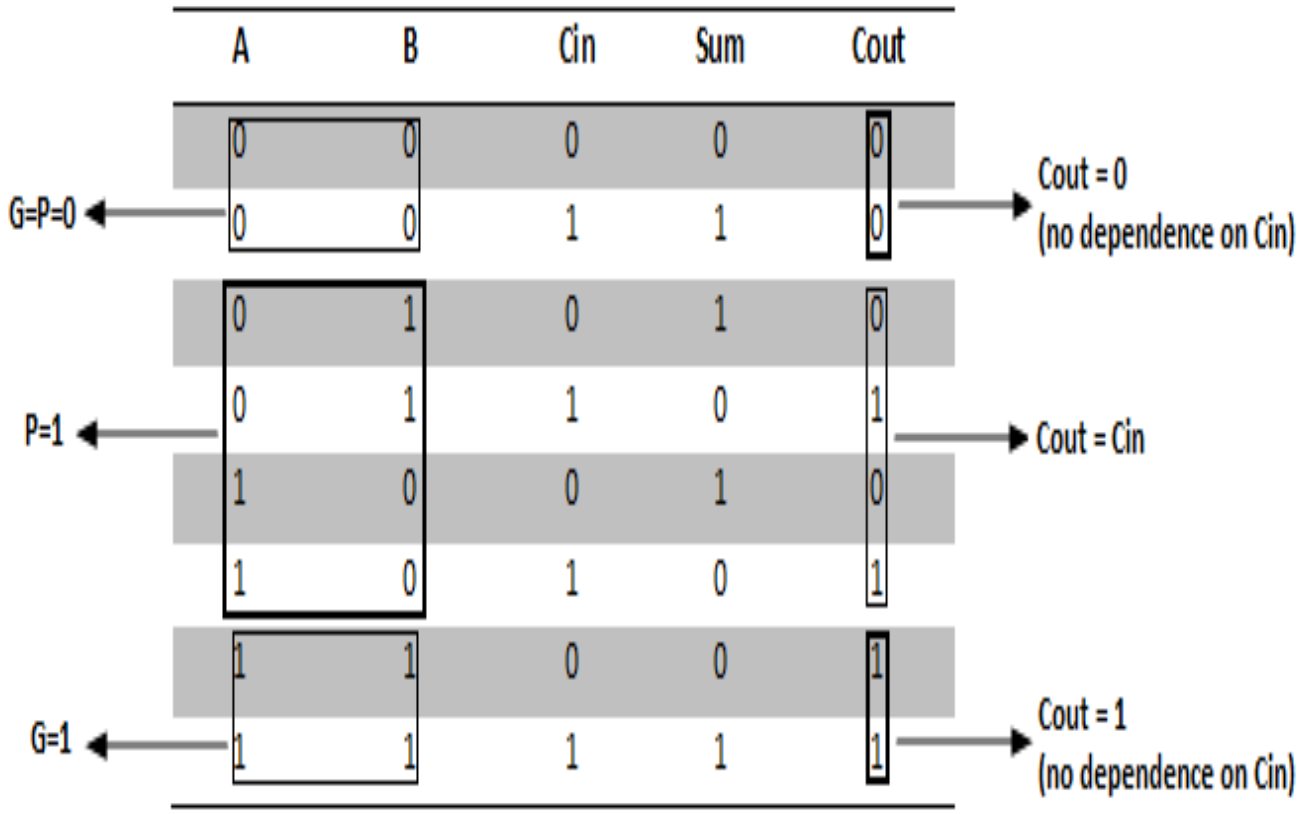
Various techniques have been proposed from time to time to decrease the overall delay in parallel addition [5]. One such technique is to derive the 'Sum' and 'Carry' outputs by using intermediate terms defined as 'Generate (G)' and 'Propagate (P)' terms [5-6]. Generate term produces a carry-out independent of the carry-in, i.e. no matter what the carry-in is, the carry-out is always '1', when both of its inputs A and B are '1' thus G = A.B. The Propagate term transfers the input Carry as output Carry when only one of the inputs is high and hence Propagate term is defined as P = A⊕B. Thus we have

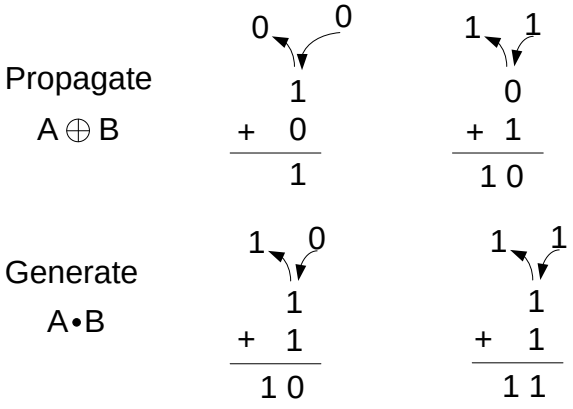
$$G(A,B) = A.B \tag{2.3}$$

$$P(A,B) = A \oplus B \tag{2.4}$$

Table 2.1 and the example shown below illustrate the concept of Propagate and Generate more clearly. In the Propagate case the 'Carry-out' depends on the 'Carry-in', i.e. when 'Carry-in' is 0 'Carry-out' is 0 and when 'Carry-in' is 1 'Carry-out' is 1 and in the case of Generate, no matter what the 'Carry-in' is 'Carry-out' is always 1.

Table 2.1 Truth Table of a Full Adder





The output 'Sum' and 'Carry' of the full adder in terms of P and G, can be observed form Table 2.1 to be,

$$S_i = P_i \oplus C_i \quad (2.5)$$

$$C_{i+1} = G_i + (P_i . C_i) \quad (2.6)$$

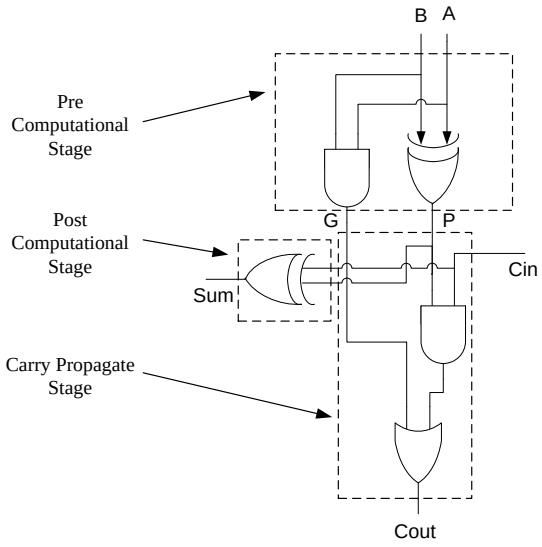


Figure 2.4 One-bit Full Adder with Carry Propagate and Generate

Figure 2.4 above illustrates the implementation of above equations (2.5) and (2.6) which is essentially same as Fig 2.1 but derived from Table 2.1. This logic is also called carry look-ahead logic. For each bit in a binary sequence to be added, the carry look-ahead logic will determine whether that bit pair will generate or propagate a Carry. This allows the circuit to "pre-process" two numbers being added to determine the carry ahead of time. Thus, when the actual addition is performed, there is no delay from waiting for the ripple carry effect (time it takes for the carry from the first full adder to be passed on to the last Full Adder) [5-6].

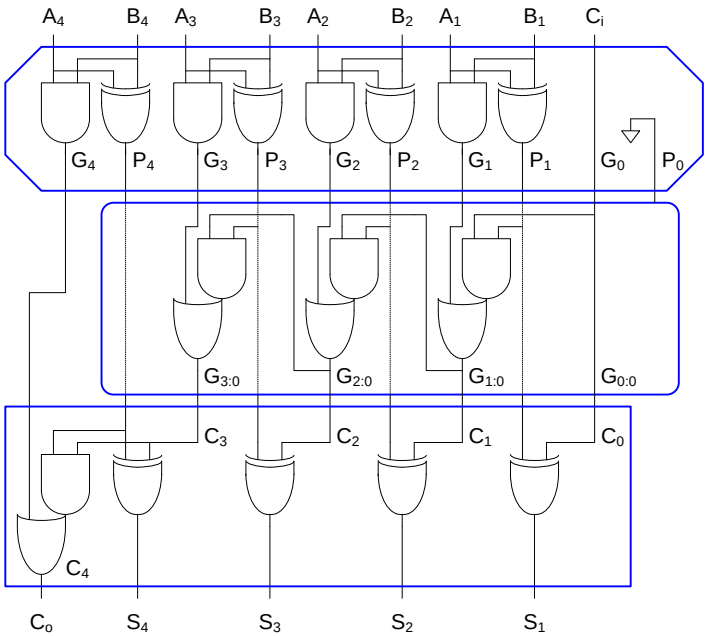


Figure 2.5 Ripple Carry Adder with Carry Propagate and Generate

The carry look-ahead type implementation of a ripple carry adder is shown in Fig 2.5. It can be seen from this figure that the carry propagation stage determines the critical path that determines the delay. To increase the speed of an adder, this stage has to be redesigned for fast carry propagation.

Keeping this in mind, Weinberger and Smith proposed a method for fast carry generation which states that the carry need not depend explicitly on the previous carry, but can be expressed as a function of only the relevant addend and augend digits and some lower order carry [7].

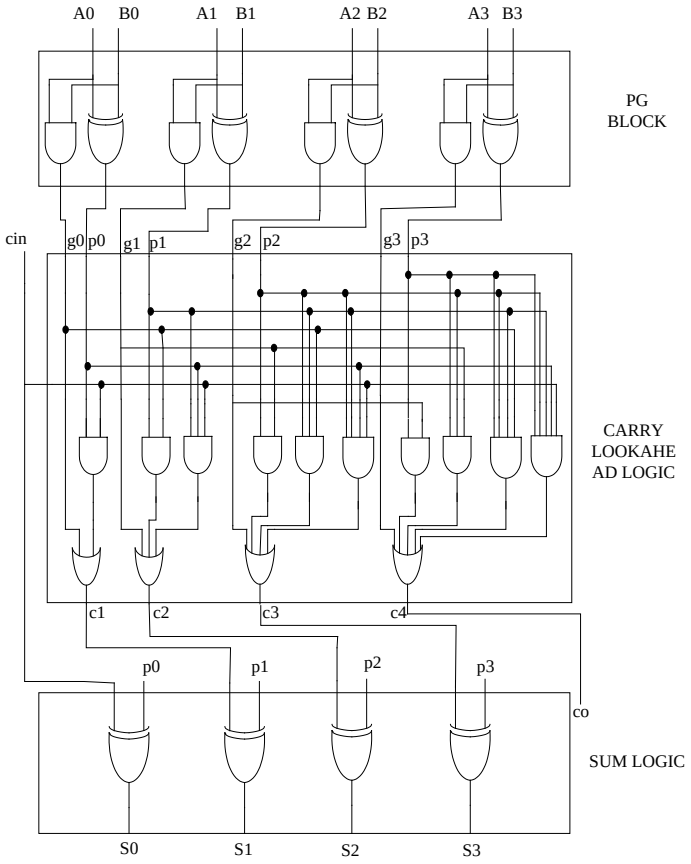


Figure 2.6 4-bit Weinberger-Smith CLA

The carry generation is done by first calculating Propagate (p_i) and Generate (g_i) terms.

$$g_i = A_i B_i \tag{2.7}$$
$$p_i = A_i \oplus B_i \tag{2.8}$$

After the parallel generation of Propagate and Generate terms, the carries can be generated using the equations below. In the following a 4-bit adder is considered as an example:

$$C_1 = g_0 + p_0 C_0 \tag{2.9}$$
$$C_2 = g_1 + p_1 g_0 + p_1 p_0 C_0 \tag{2.10}$$

$$C_3 = g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0C_0 \quad (2.11)$$
$$C_4 = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0C_0 \quad (2.12)$$

After the carries are generated, the sum is calculated using the equation

$$S_i = A_i \oplus B_i \oplus C_i \quad (2.13)$$

A typical 4-bit CLA implementing the above equations is shown in Fig 2.6. For wide adders where N > 16 (N is the input operand size), the delay of the carry look-ahead adders becomes dominated by the delay of passing the carry through the look-ahead stages and the implementation needs high fan-in gates [1, 5]. To overcome these problems, a new breed of networks has been designed that pass the carry through the look ahead stage in around log(N) stages. These networks are called Tree Networks and the adders that utilize these networks are called tree- adders or prefix- adders [5]. There are many ways to build the tree adders which offer tradeoffs among parameters like, the number of stages of logic, number of logic gates, the maximum fan-out of each gate and the amount of wiring between the stages.

2.2.4 Prefix Based Adders

A prefix adder consists of 3 stages i.e, pre-computation stage, prefix network stage and post-computation stage as shown in Fig 2.7 [5-11].

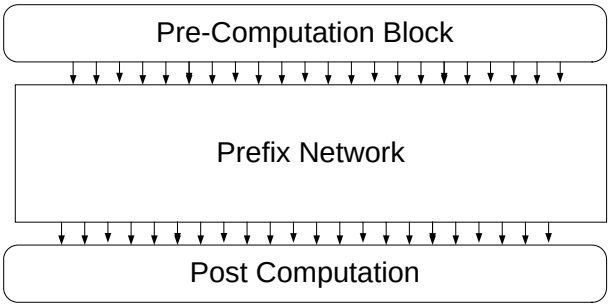


Figure 2.7 Block level diagram of a prefix adder

The pre-computation stage computes the carry ‘Propagate’ and carry ‘Generate’ bits for each input pair as given below.

$$\text{Generate, } g = ab \quad (2.14)$$
$$\text{Propagate, } p = a \oplus b \quad (2.15)$$

The prefix network stage computes the final carries from the carry ‘Propagate’ and carry ‘Generate’ bits. Carry computation can be transformed to a prefix problem [5-9] using the

associative operator '○', which associates pairs of 'Generate' and 'Propagate' bits as given below:

$$(g,p)\circ(g',p') = (g + p.g', p.p') \qquad (2.16)$$

where g and g' represent the 'Generate' terms and p and p' represent the 'Propagate' terms. Using the operator '○' consecutive 'Propagate' and 'Generate' pairs can be grouped to generate carry as follows:

$$C_i = (g_i, p_i) \circ (g_{i-1}, p_{i-1}) \circ \ldots\ldots (g_1, p_1) \circ (g_0, p_0) \qquad (2.17)$$

Representing the operator '○' as node ●, and signal pairs (g,p) as edges of a graph, parallel prefix carry computation can be represented as graphs. One of the prefix networks, Kogge-Stone [9] represented as a graph is shown in Fig. 2.8. The white color node in the graph represents a feed through node with no logic (generally realized with a buffer in hardware).

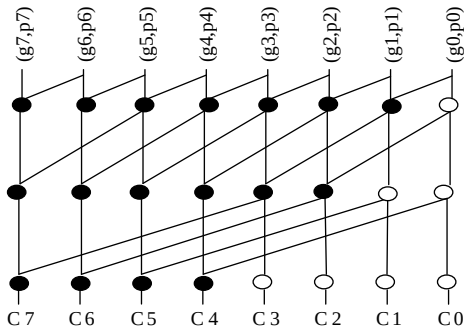


Figure 2.8 Example of a Kogge –Stone prefix adder

The final post computation stage computes the final Sum from carry generated in prefix network stage. These designs are very efficient in terms of delay and area when compared to carry-select and carry look-ahead adders.

As operand size increases (32-bits and above) these prefix adders suffer from complexity in prefix network due to an increase in number of logic cells and wiring [5, 9]. This problem can be addressed with a hybrid adder (also called as a sparse adder) which is a combination of prefix and carry-select adders [12-17]. These adders consist of two segments, one being carry generation block (CGB) that has prefix network and the other the conditional sum computation block (SCB) that has carry-select adders shown in Fig 2.9. As seen from the figure in CGB, where a Kogge-Stone network structure is used, all 'carry's are not computed as in prefix adders (shown in Fig. 2.8) but only a few (in this case C_3 and C_7) are computed depending on the degree

of sparseness where the degree of sparseness is the number of sums selected conditionally. For example, degree of sparseness 4 means that the carry will select four sum bits conditionally as shown in Fig 2.9. Hence, all the carries are not required in CGB.

The SCB in general is implemented by using a carry select adder. As seen from the Fig. 2.9, appropriate sums in SCB are selected by the 'carry's generated in CGB using multiplexers (MUX). Carry-select adders are better suited for sparse adders with low sparseness of 2-bit and 4-bit. Tyagi [18] proposed a reduced area scheme carry-select adder which can be used in a SCB. An optimized implementation of a sparse adder with carry-select adder in SCB can be found in [12, 17].

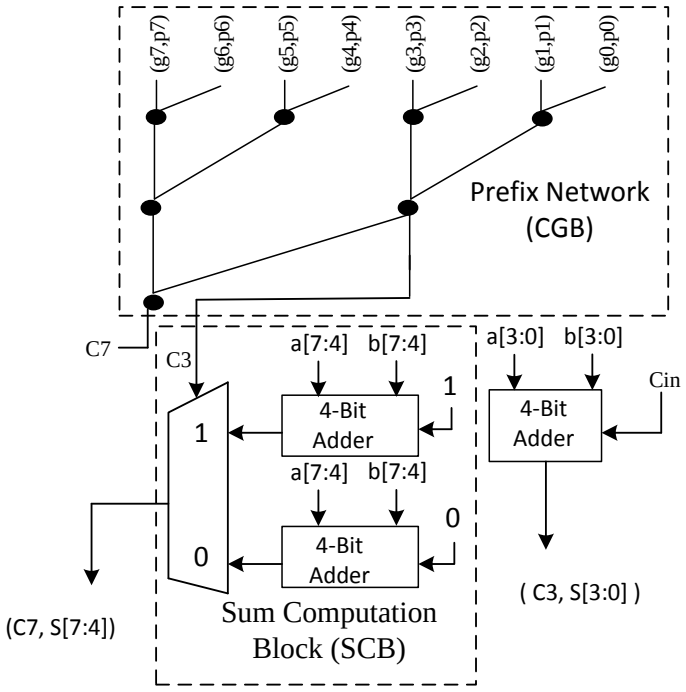


Figure 2.9 Example of a 8-bit Sparse adder with degree of sparsness as 4

It is clear that sparse adders have simple carry generation block. But as the operand length increases, sparse adders also suffer from high fan-out and lateral wiring complexity in carry generation network as in a prefix network. For instance, when the CGB of a 8-bit Kogge-Stone Sparse adder shown in Fig. 2.9 is extended for 64-bit Kogge-Stone Sparse adder, shown in Fig. 2.10, it can be seen that the wiring complexity (i.e., congestion between wires) to generate carry signals is increased. Similarly when a 64-bit Sklansky based prefix network is used in CGB shown in Fig. 2.11 it can be noticed from the figure that the fan-out on carry signal 'C31' is high (signal 'C31' is an input to compute the higher bit 'carry's).

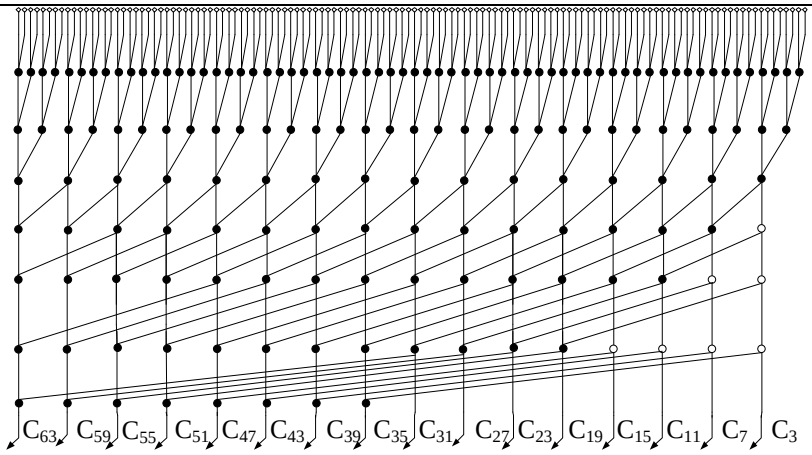


Figure 2.10 64 bit Kogge-Stone based CGB for a sparse 4 adder

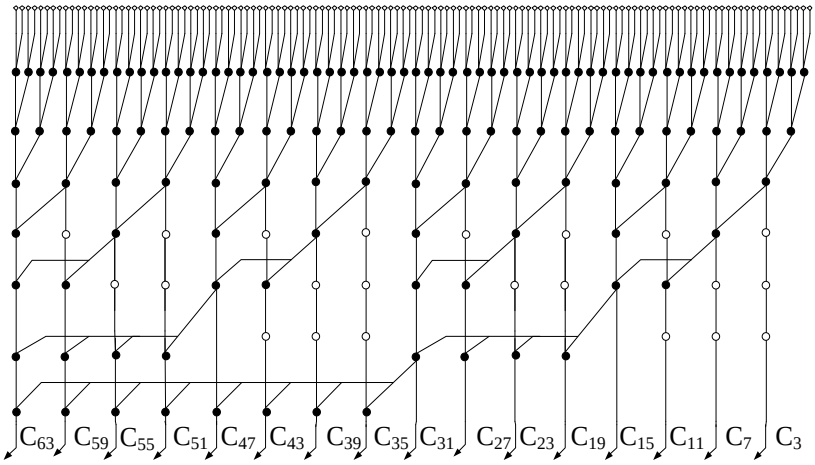


Figure 2.11 64 bit Sklansky based CGB for a sparse 4 adder

While these drawbacks can be overcome by increasing the degree of sparseness, the SCB complexity however will increase. Moreover, the loading on Carry signal will increase further as the number of Sum bits in SCB increases. For example in an 8-bit SCB, the Carry signal has to drive eight MUXes to select the Sum which consume a large amount of area and power, thus limiting the usage of direct higher bit carry-select adder as SCBs. In this work a modified SCB is proposed and analyzed to address these problems.

2.3 Design and Implementation of Efficient Sum Computation Block for Higher Bit Sparse Adders

As discussed in section 2.2.4 earlier, the power, area and fan-out overheads limit the usage of carry-select adder in SCB as the degree of sparseness increases. The area overhead can be reduced by using prefix structure with late Carry-in concept proposed by Sklansky [4]. This late

Carry-in concept or fastest input-carry processing is achieved by adding an extra row of node ● at the end of the prefix carry network as shown in Fig 2.12. This addition of extra node ● however increases the overall delay of the adder by one node stage. Any prefix structure can be preferred to implement the prefix carry network depending on the design requirement. However, the fan-out or loading on the Carry signals from the CGB is still a problem. In this work, the structure of prefix network and the late carry-in scheme are analyzed and a new structure is developed to address these problems.

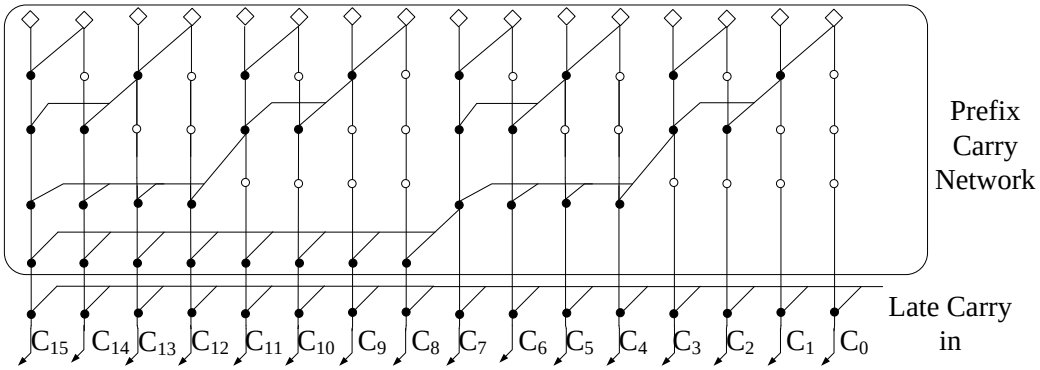
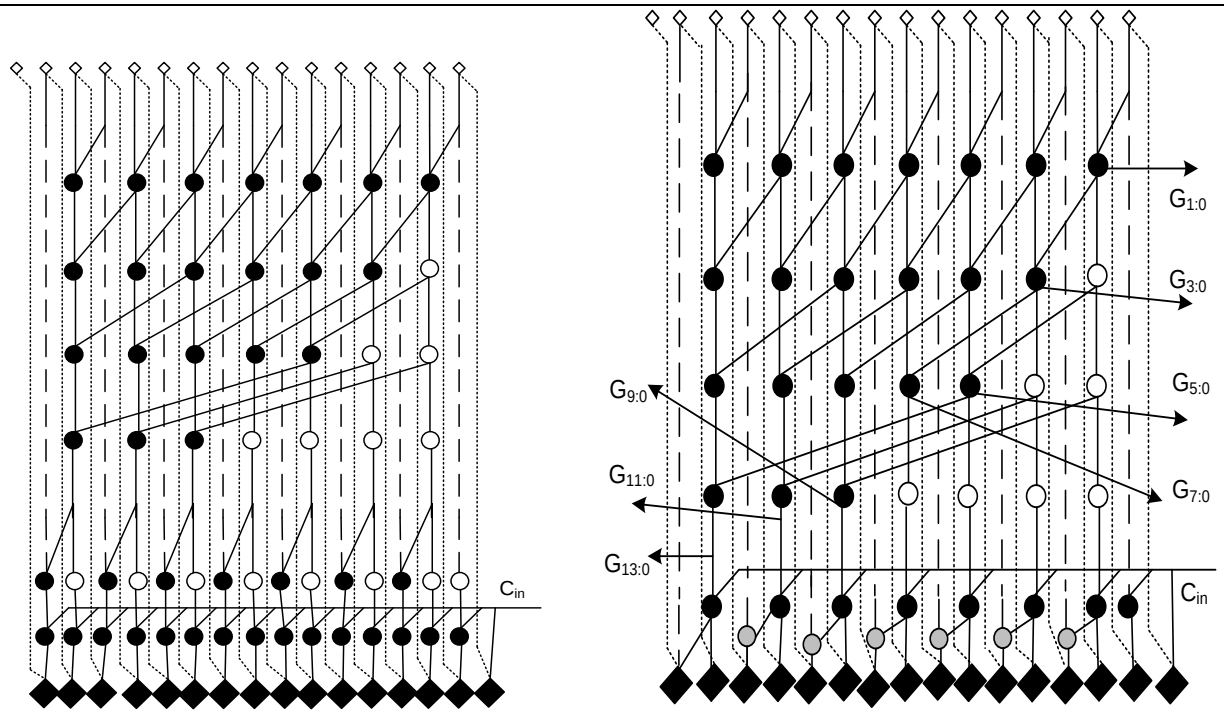


Figure 2.12 Sklansky parallel prefix adder with late Carry-in

The proposed approach is to reduce the fan-out or loading on the late carry-in signals it is achieved by feeding the late Carry-in signal only for a few ‘carry’s. The remaining ‘carry’s are to be computed with these few ‘carry’s to generate all the ‘carry’s required for sum computation. The proposed technique is illustrated through an example by taking Han-Carlson prefix structure. This prefix structure is chosen because of its uniformity in fan-in and fan-out requirements as well as reduced number of nodes when compared to other prefix structures [9-11].

A 16-bit traditional Han-Carlson adder with late carry-in is shown in Figs 2.13(a). From the figure, it can be seen that the loading on the C_{in} signal is 16. To reduce this loading, the proposed technique is applied to this structure wherein the late carry-in signal (C_{in}) is fed only to odd ‘carry’s i.e., $G_{1:0}$, $G_{3:0}$, etc… Even ‘carry’s are than computed from the odd ‘carry’s. The modified Han-Carlson with late carry-in using the proposed technique is shown in Fig. 2.13 (b). It can be observed that a 16-bit modified adder has a fan-out requirement of only 9 compared to the traditional late carry-in prefix structure that has a fan-out of 16. Thus, the proposed technique results in the reduction of fan-out on the late carry-in signal.



(a) (b)

Figure 2.13 (a) 16-Bit Han-Carlson Adder with late Carry-in. (b) Modified 16-Bit Han-Carlson Adder with late Carry-in.

The gate-level realization of the nodes of the above structures is shown in Fig 2.14.

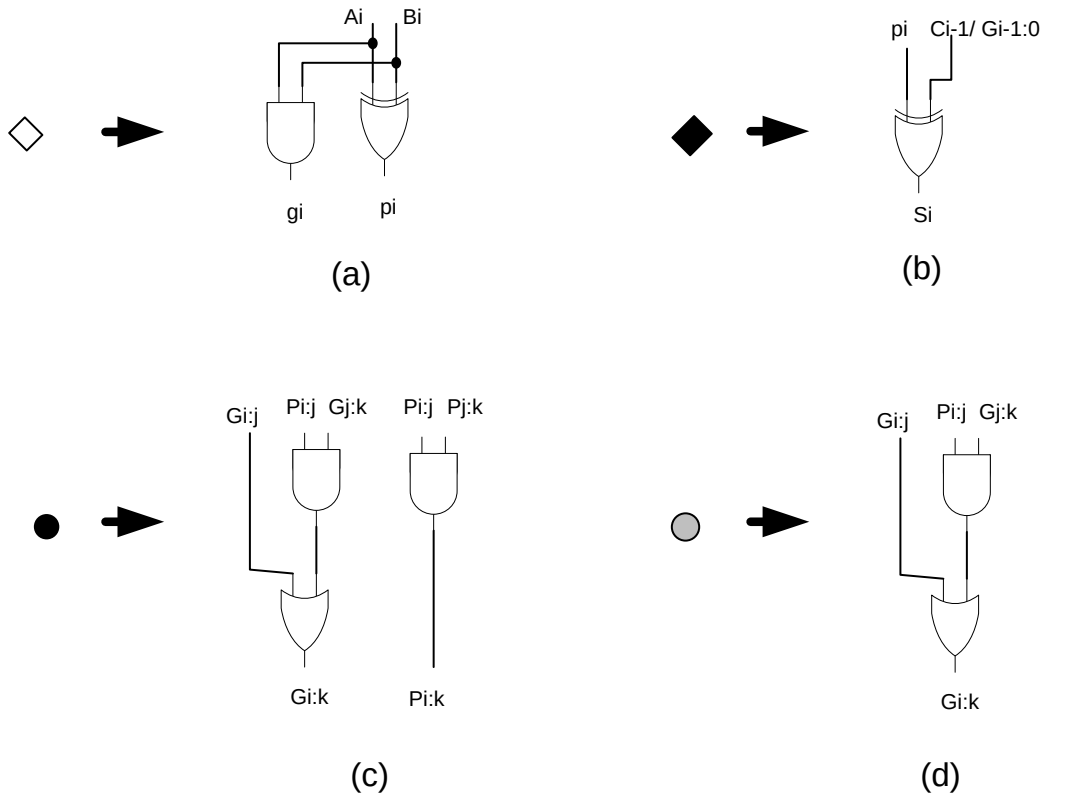


Figure 2.14 Gate level implementation of nodes in Fig. 2.13.

The main limitation of the proposed technique is the uneven arrival of even and odd Sums. For example, as seen from the Fig. 2.15, Sum S_2 will be computed when the Generate term $G'_{1:0}$ arrives. Further, Sum S_3 has to wait for generate term $G'_{2:0}$ which depends on $G'_{1:0}$. This not only results in extra delay but also leads to different arrival times of the digits of final Sum. This issues is addressed in this work as described below.

From Fig. 2.15, the equation to compute the Sum digits S_2 and S_3 are as follows:

$$S_2 = G'_{1:0} \oplus G_1 \quad (2.18)$$

$$S_3 = G'_{2:0} \oplus G_2 \quad (2.19)$$

The generate signal $G'_{2:0}$ equation given in terms of $G'_{1:0}$ is as follows

$$G'_{2:0} = G_2 + G'_{1:0} P_2 \quad (2.20)$$

From equations 2.19 and 2.20, S3 can be rewrite as

$$S_3 = (G_2 + G'_{1:0}P_2) \oplus P_3 = (G_2 \oplus P_3)\,(G'_{1:0})' + ((G_2 + P_2) \oplus P_3)G'_{1:0} \quad (2.21)$$

Equation 2.21 can be realized as shown in Fig. 2.16. Thus, the structure shown in Fig 2.13 can be further modified using this block as shown in Fig. 2.17. It can be seen that the overall delay is reduced as well as the varied arrival times of different Sum digits is addressed. This technique can also be adopted for the design of higher bit sparse tree adders [19-21] as explained below.

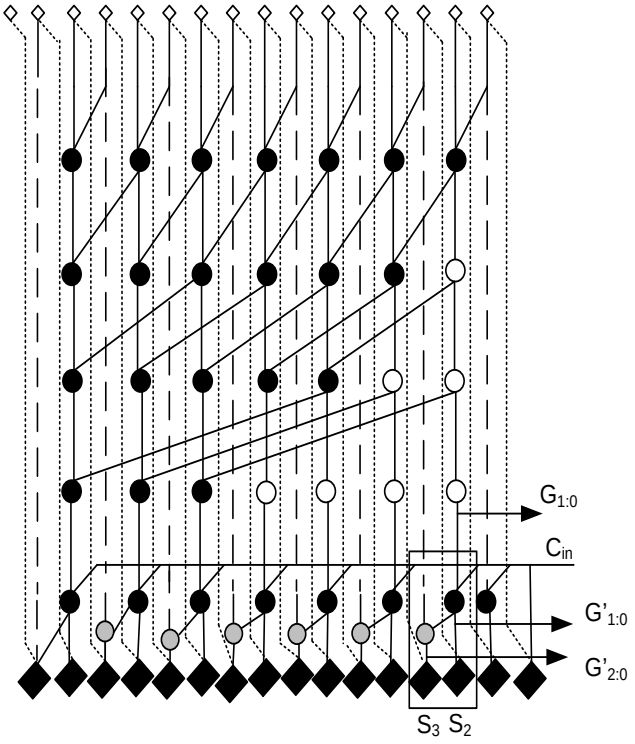


Figure 2.15 Modified 16-Bit Han-Carlson Adder with late Carry-in illustrating delay problem

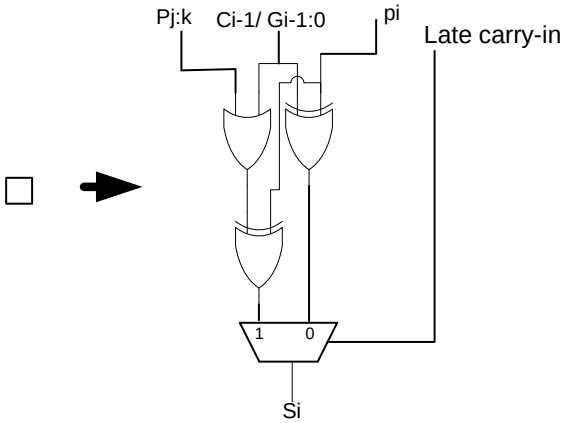


Figure 2.16 Gate level implementation of Equation 2.21

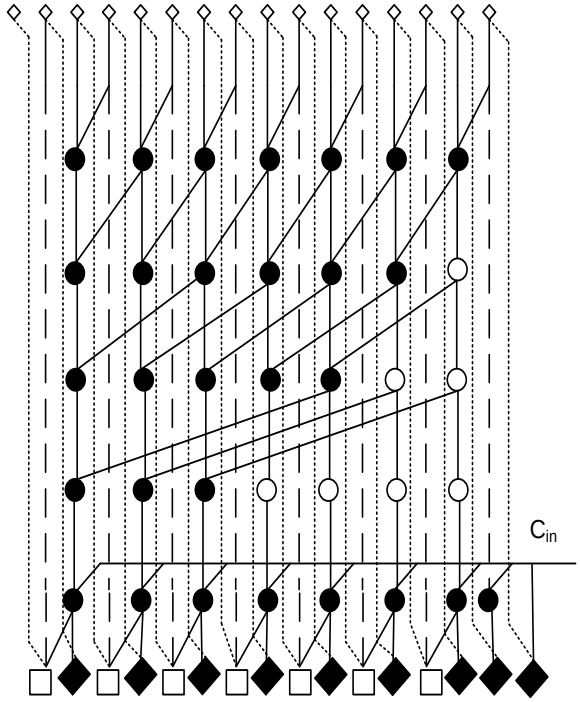


Figure 2.17 Proposed SCB block with reduced fan-out with reduced delay

2.4 Design and Implementation of Higher Bit Sparse Adder

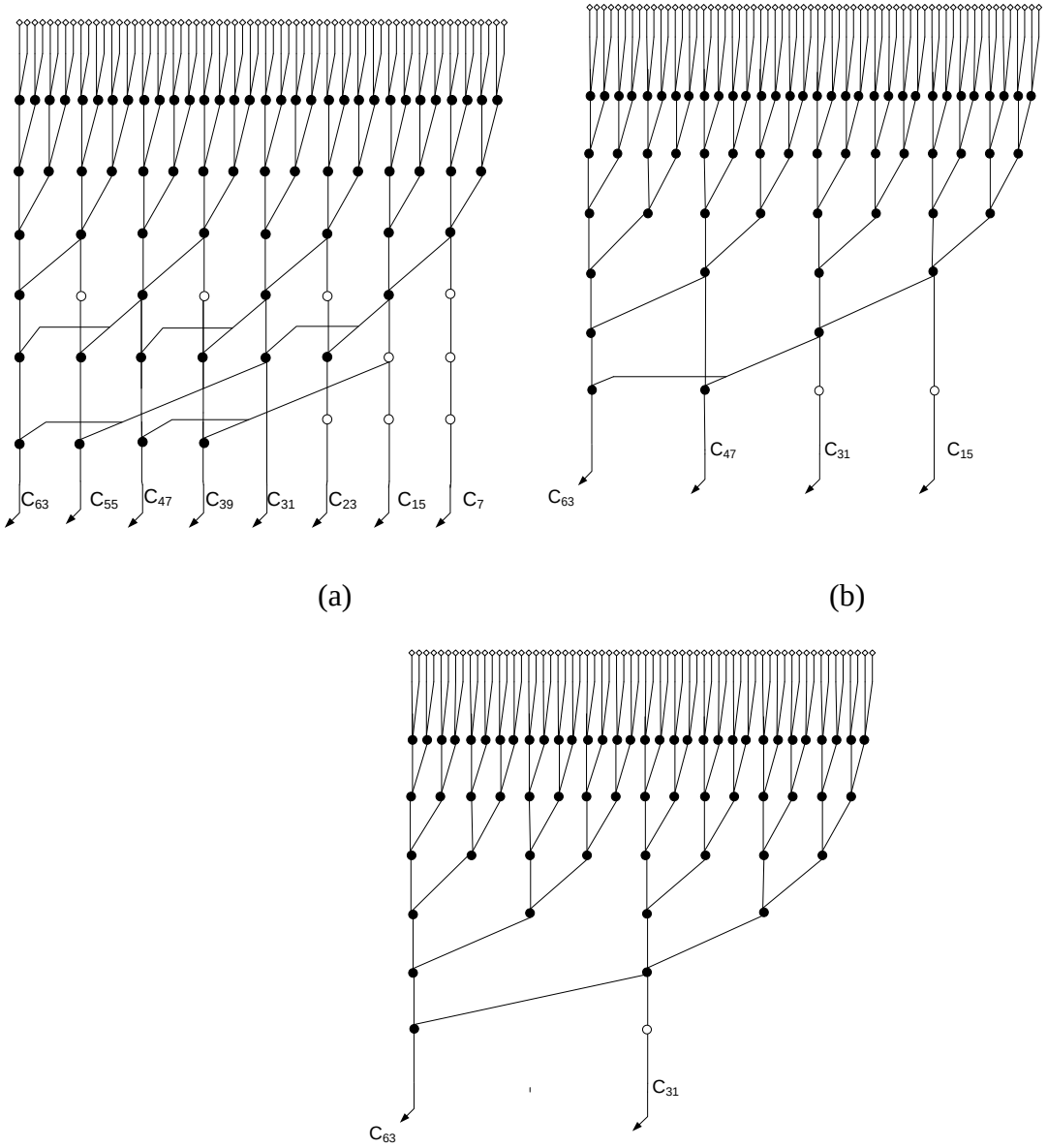
In Section 2.2.4, issues related to the implementations of higher bit sparse adders with degree of sparseness more than 4 such as increasing wiring complexity and loading on the carry-in signal have been explained. To address these problems a SCB has been proposed in Section 2.3. In this section, a 64-bit sparse adder is designed and implemented with a varying degree of sparseness of 8, 16 and 32–bit in order to verify the advantages of the proposed SCB structure stated earlier.

The CGB of 64-bit sparse adder with different degrees of sparseness mentioned above is shown in Fig 2.18 (a, b, c) [19-20]. From the figure, it can be observed that the CGB complexity has decreased as the degree of sparseness increases.

After the generation of 'carry's in CGB, the 'sum's are computed by using the SCB proposed in the previous section. The 16-bit SCB structure explained earlier can be used for a degree of sparseness 16. The same structure can also be extended for different bits to address different degrees of sparseness.

The SCB area can further be reduced by using some of the group 'Generate' and 'Propagate' terms that have already been computed in the CGB. If the intermediate Propagate and Generate terms generated at the end of the second stage of CGB, that is ($G_{[1:0]}$,$P_{[1:0]}$) ,

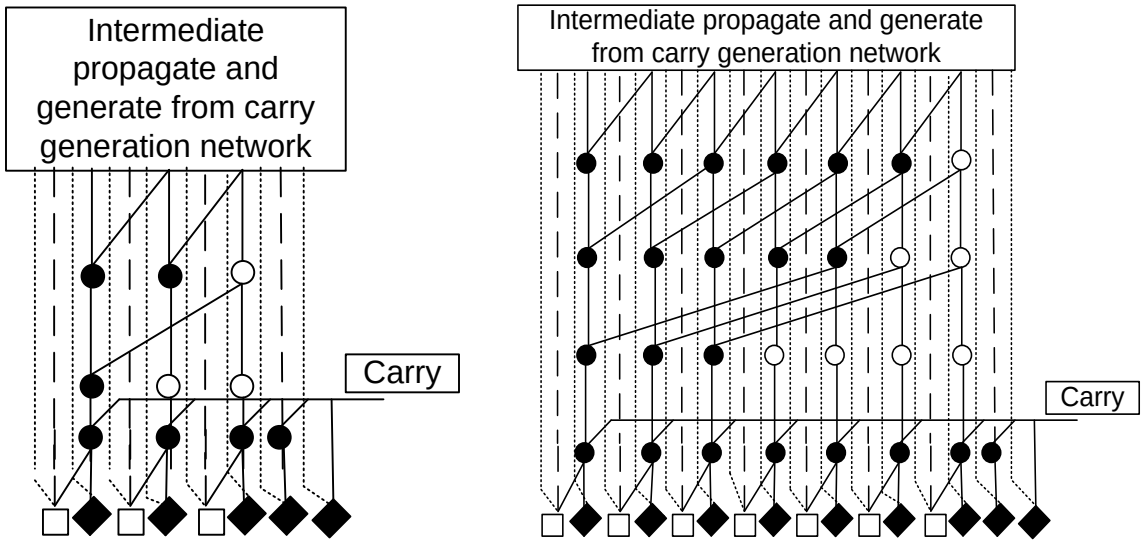
($G_{[3:2]}$,$P_{[3:2]}$)..etc., are used for Sum computation in SCB, it will result in power and area reduction when compared to the existing sparse implementations. The proposed SCB with reduced cells is shown in Fig 2.19(a) and Fig 2.19(b) for 8-bit and 16-bit respectively [19-20]. The same can also be extended to 32-bit SCB.



(a) (b)

(c)

Figure 2.18 Carry generation for 64-Bit (a) Sparse-8 (b) Sparse-16 and (c) Sparse-32 adders.



(a) (b)

Figure 2.19 Modified Han-Carlson adder with late Carry-in after reusing the second stage group Propagate and Generate terms from Carry generation stage (a) 8-bit (b) 16-bit

From Fig 2.18 it can be seen that the SCBs have progressively lesser wiring and logic cell complexity with increasing sparseness while the corresponding CGBs have increasing complexity as seen from Fig 2.19. Thus, it can be generalized that for a sparse tree adder, the complexity of SCB is inversely proportional to the complexity of the CGB. Since the sparse adders provide the flexibility to control the Carry signal, these adders have application in the design of multi-precision adders [22-24].

2.5 Simulation Results

All adders have been described in Verilog HDL and simulated using Cadence Incisive Unified Simulator (IUS) v6.1 and are mapped on to the Synopsys 90*nm* generic Technology library, using Cadence RTL Compiler v7.1. The derived netlist was then passed to Cadence First Encounter XL v7.1 for floor-planning and routing.

The modified Han-Carlson Sum computation block for 64-bit Sparse-8, -16 and -32 has been compared with 64-bit Sparse-8, -16 and -32 Han-Carlson late Carry-in adder and 64-bit sparse-4 with conditional Sum adder [19-20]. Table 2.2 presents performance parameters such as area, power, delay and power-delay product for all the three designs. Also, Fig. 2.20 provides a graphical comparison of these parameters.

Table 2.2 Area, power, delay and power-delay product for 64-bit adder with various sparseness

64-Bit adder	Sparse-4 with conditional sum	Sparse-8 with Han-Carlson late carry-in	Sparse-8 with Modified Han-Carlson late carry-in	Sparse-16 with Han-Carlson late carry-in	Sparse-16 with Modified Han-Carlson late carry-in	Sparse-32 with Han-Carlson late carry-in	Sparse-32 with Modified Han-Carlson late carry-in
Area (um^2)	4316 (100%)	4739 (109.8%)	3998 (92.63%)	5257 (121.80%)	4383 (101.55%)	5652 (130.95%)	4911 (113.79%)
Power(mW)	0.286 (100%)	0.281 (98.25%)	0.2471 (86.36%)	0.309 (108.04%)	0.2627 (91.60%)	0.3278 (114.62%)	0.293 (102.45%)
Delay(ns)	1.101 (100%)	0.956 (86.83%)	0.969 (88.01%)	1.05 (95.37%)	1.03 (93.55)	0.89 (80.84%)	0.89 (80.84%)
Power-Delay product (pJ)	0.3149 (100%)	0.2686 (85.29%)	0.2394 (75.87%)	0.3245 (103.05%)	0.2706 (86.03%)	0.2917 (92.63%)	0.2608 (83.17%)

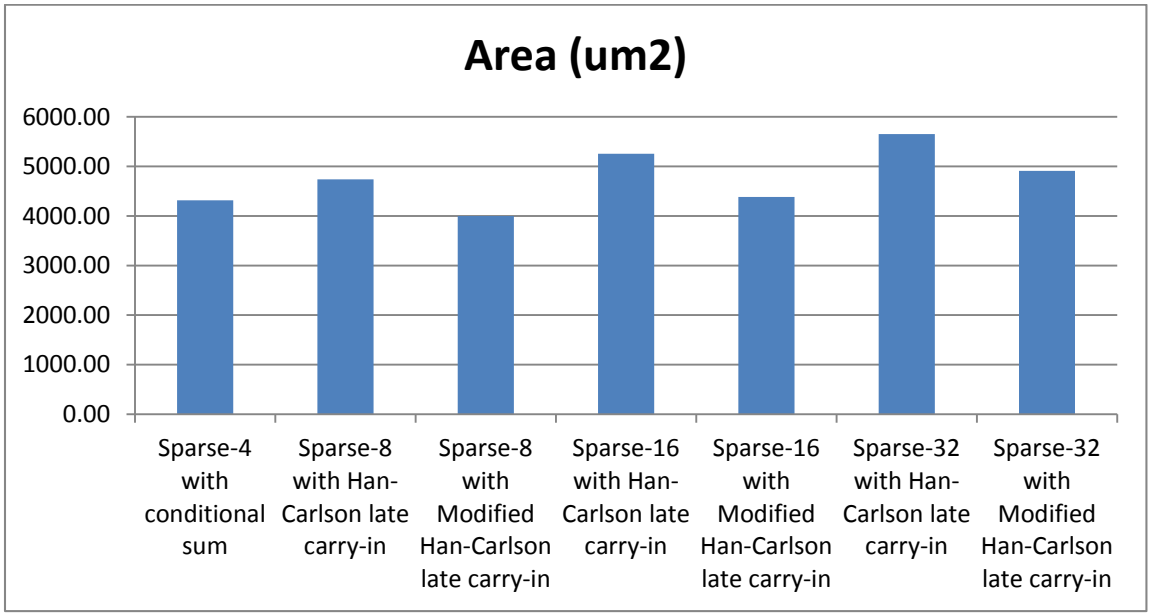


Figure 2.20 (a)

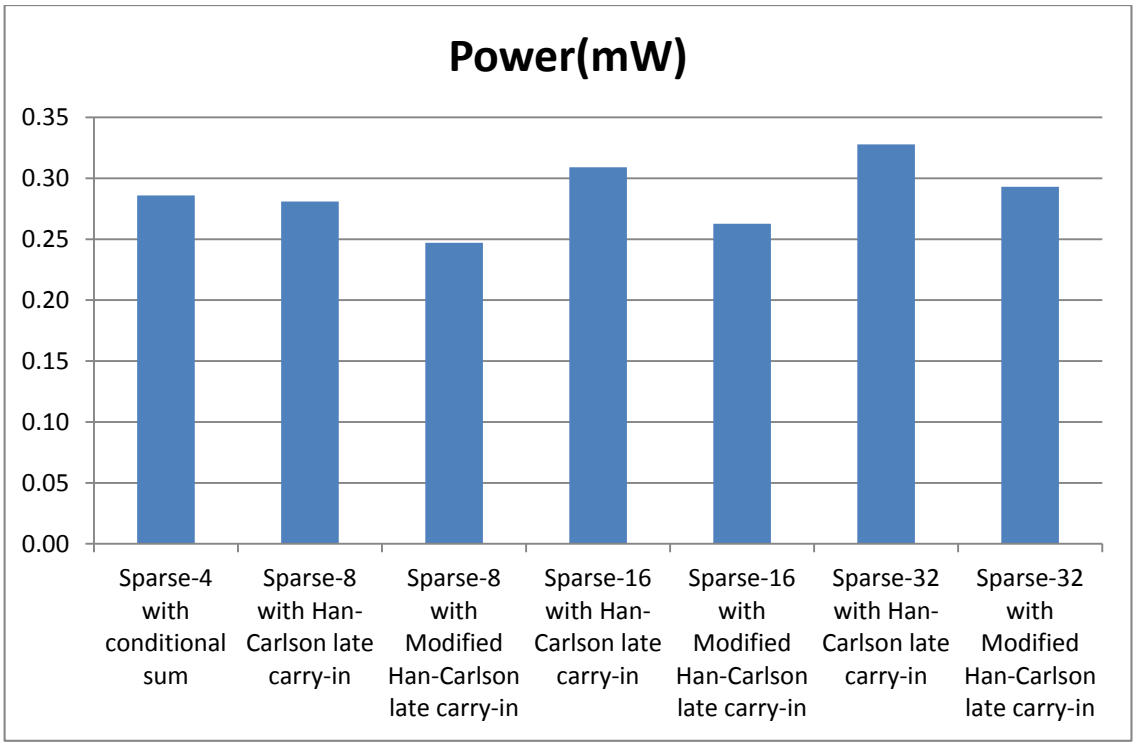


Figure 2.20 (b)

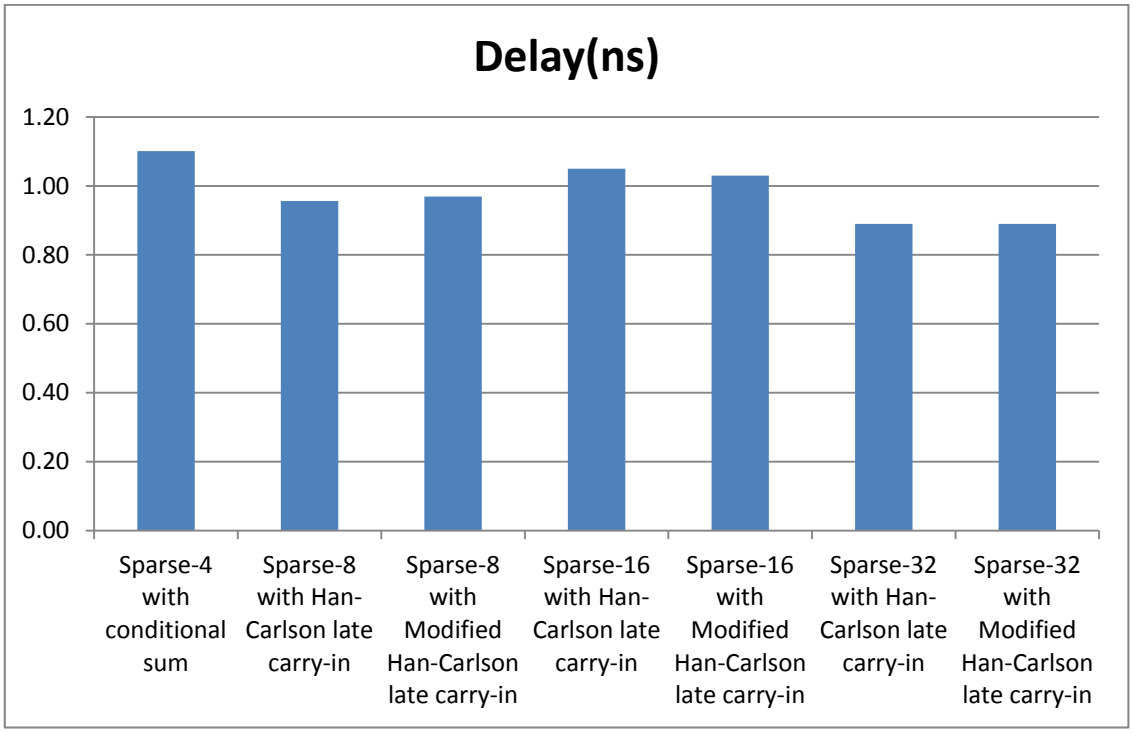


Figure 2.20 (c)

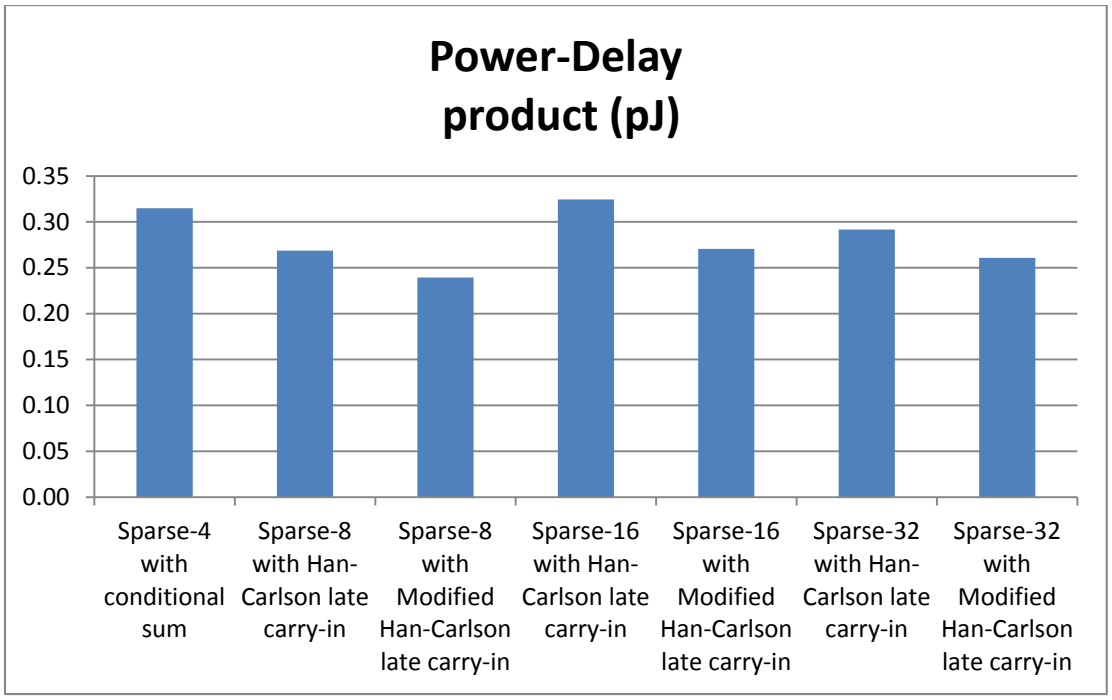


Figure 2.20 (d)

Figure 2.20 (a) Area (b) Delay (c) Power and (d) Power-Delay product analysis of various 64-bit adders with different degrees of sparsesness

Table 2.2 and Fig. 2.20 provide a comparison of various design parameters for 64-bit adder with different degrees of sparseness. As can be seen, the proposed 64-bit adder with a sparseness of 8 involving the modified Han-Carlson adder performs better than other designs in terms of power (a reduction of 14%) and delay (a reduction of 12%) resulting in a overall reduction of 25% in power-delay product. Further, there is also a reduction of 8% in area. But, if delay is the only parameter important, then the design with a degree of sparseness 32 with modified Han-Carlson late Carry-in adder that results in a 20% reduction in delay can be used.

It can be observed from the above table and figure that the 64-bit adder with a sparseness of 16 and 32, while performing better than that with a sparseness of 4, do not perform as well as that with a sparseness of 8. Also the adder with sparseness 32 performs better than that with sparseness 16. This is because the 64-bit adder with a sparseness of 8 and 16, using either existing compound adder or late Carry- in adder, needs 6 Carry merge stages to compute Carry and has a fan-out of 8 and 16 respectively in the critical path. However, the same adder with a sparseness of 32 needs 5 carry merge stages to compute carry and has a fan-out of 32 in the critical path. Hence, there is an increase in delay for sparse-16 adder when compared to sparse-8

or sparse-32. This is also applicable to the sparse adders with modified sum computation block [19-20].

An extended analysis has also been done for a 128-bit adder using the proposed technique. Table 2.3 and Fig 2.21 present data related to area, power, delay and power-delay product for sparse-16 and sparse-32 adders with modified Han-Carlson sum computation block.

Table 2.3 Area, power, delay and power-delay product for 128-bit adder with sparseness of 16 and 32

128-Bit adder	**Sparse-16 with Modified Han-Carlson late Carry-in**	**Sparse-32 with Modified Han-Carlson late Carry-in**
Area(um^2)	8775	9846
Power(mW)	0.5354	0.603
Delay(ns)	1.367	1.349
Power-Delay Product(pJ)	0.7319	0.813

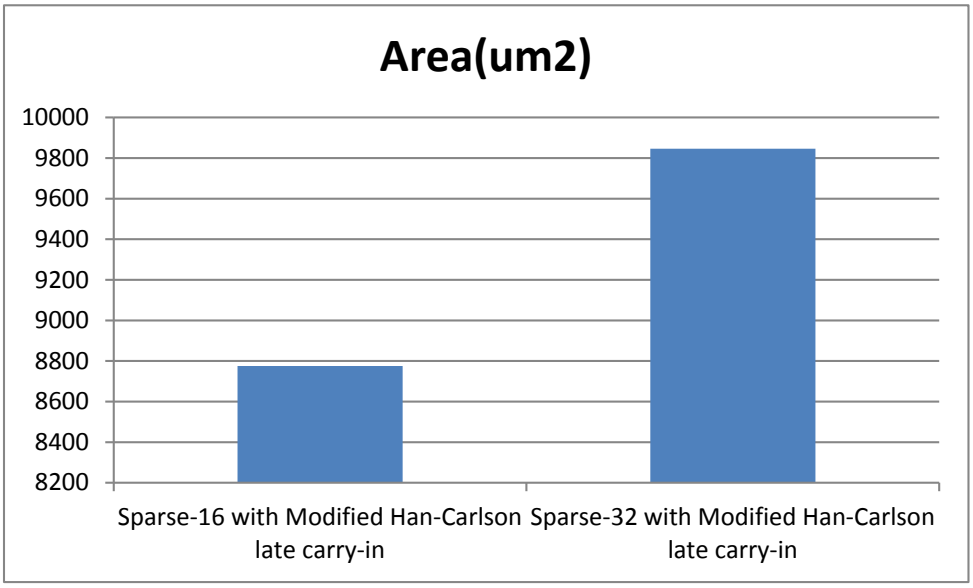


Figure 2.21 (a)

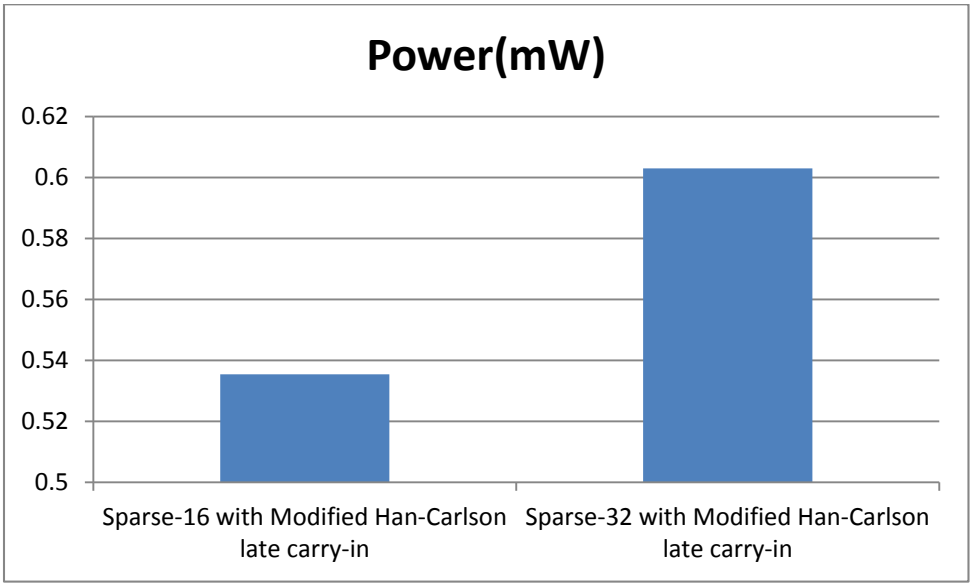


Figure 2.21 (b)

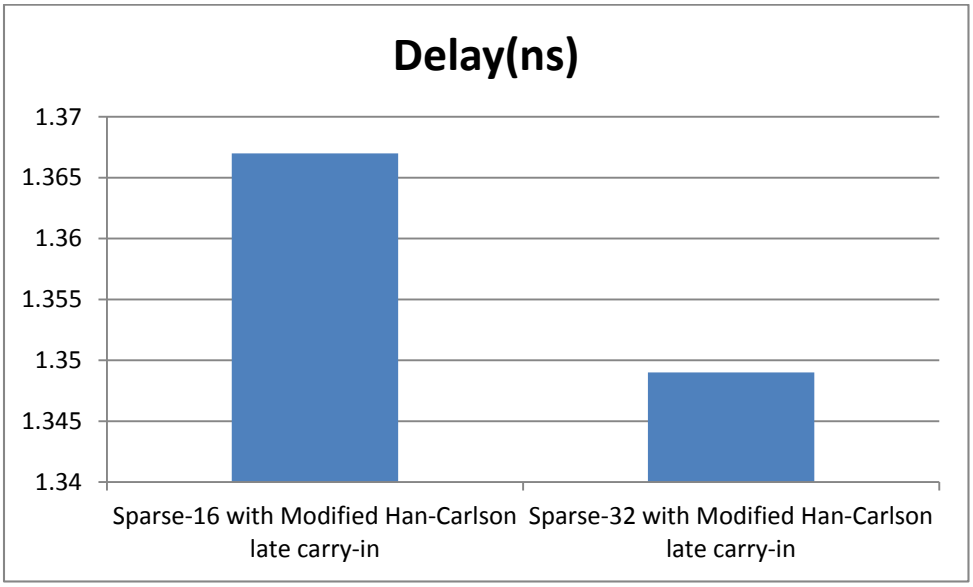


Figure 2.21 (c)

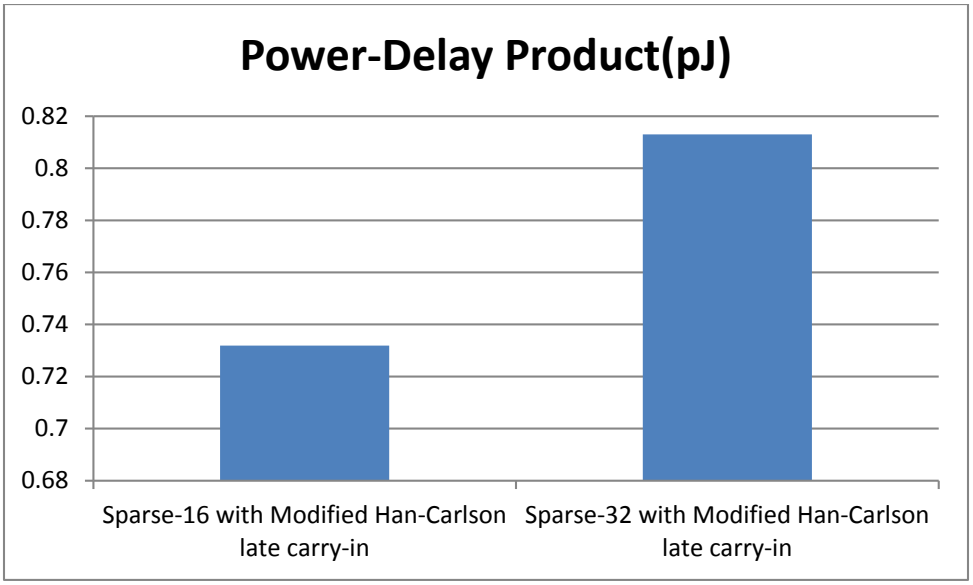


Figure 2.21 (d)

Figure 2.21 Extended analysis of 128-bit adder using the proposed technique in terms of (a) Area, (b) Delay (c) Power and Power-Delay Product

It can be seen from the above table and figure that the 128-bit adder with a sparseness of 16 performs better than the one with sparseness of 32.

2.6 Conclusions

In this chapter, novel designs for higher bit (64 & 128) sparse adders have been proposed. The increased complexity of the sum computation block at larger bit lengths has been compensated with alternate designs of carry generation block that results in reduced complexity. A detailed analysis of the 64 & 128-bit sparse adders with different degree of sparseness indicates that they perform better than the designs reported in literature.

Chapter 3
Design and Implementation of Efficient Compressors and Counters

Contents

3.1 Introduction

Multiplication is a basic arithmetic operation that is crucial in applications like digital signal processing which in turn rely on efficient implementation of generic arithmetic and logic units (ALU) and floating point units to execute dedicated operations like convolution and filtering. In the implementation of multipliers, the main phases include generation of partial products, reduction of partial products using CSA (Carry-Save Adder) [1-6] and Carry propagation for the computation of the final result as shown in Fig 3.1. The second phase i.e. reduction of the partial products contributes most to the overall delay, area and power.

In order to reduce partial products, multi-operand adders, which are different from conventional adders, are required and hence a different design methodology is needed for multi-operand adders [1-6]. A special structure known as counter/compressor is one strategy that can be adopted for multi-operand addition. Wallace and Dadda were the first ones who explained the usage of compressors and counters respectively for partial product reduction tree in multipliers [25-26]. Later different optimized structures for compressors and counter have been reported in literature [27-28].

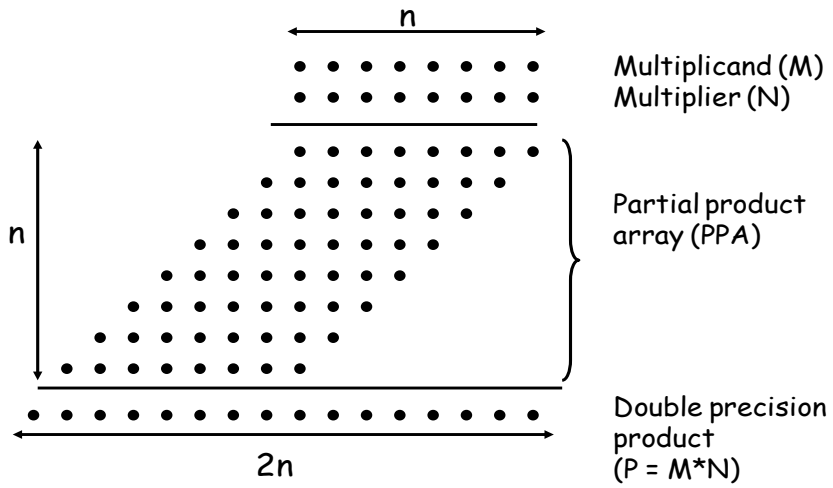


Figure 3.1 Steps involved in Multiplication

3.2 Compressors and Counters

A (N, 2) compressor is a logic circuit that takes N bits of same significance and generates a Sum bit and several Carry bits as the output. Though a compressor gives Sum and Carry, it is different from a conventional adder. For example, compressor adds N-bits of same precision whereas an adder adds 2 operands of N-bit numbers of different precision. Compressor operation can be shown logically as

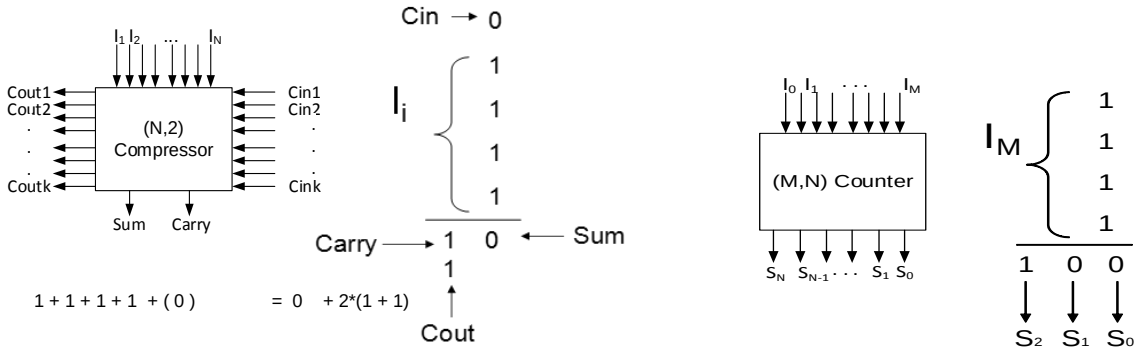
$$I_1 + I_2 + ... + I_N + (Cin_1 + Cin_2 + ... + Cin_k) = Sum + 2*(Carry + Cout_1 + .. + Cout_k)$$

Where I_1, I_2… and C_{in1}, C_{in2}…are inputs for compressor.

An (M, N) parallel counter is a circuit which provides an N-bit count of the number of the M-inputs that are logic ones. A counter differs from a compressor in that compressors have 'Carry-inputs' and 'Carry-outputs' in addition to the "normal" inputs and outputs, while counters do not have them. An (M, N) bit counter is defined as

$$I_0 + I_1 + ... + I_M = 2^0 * S_0 + 2^1 * S_1 + ... + 2^N * S_N$$

Figure 3.2 Illustrates the difference between a compressor and a counter. Fig 3.2 (a) and (b) explain the compressor and counter operations by taking an example of four same significant bits [27-28].



(a) (b)

Figure 3.2 (a) Compressor (b) Counter

3.3 Existing Compressor Designs

3.3.1 3-2 Compressor

A 3-2 compressor takes 3 inputs X1, X2, X3 and generates 2 outputs, the Sum bit S, and the Carry bit C as shown in Fig.3.3(a).

The compressor is governed by the basic equation

$$X1 + X2 + X3 = Sum + 2*Carry \quad (3.1)$$

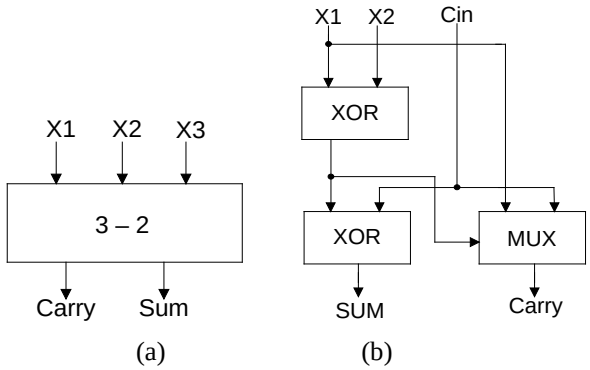


Figure 3.3 (a) 3-2 Compressor (b) Conventional Implementation of the 3-2 compressor

The 3-2 compressor can also be employed as a full adder cell when the third input is considered as the 'Carry-in' from the previous compressor block. Existing design shown in Fig 3.3(b) employs two XOR gates in the critical path [27-28].

3.3.2 4-2 Compressor

A 4-2 compressor has 4 inputs X1, X2, X3 and X4 and 2 outputs, Sum and Carry, along with a Carry-in (Cin) and a Carry-out (Cout) as shown in Fig 3.4. The input C_{in} is the output from the previous lower significant compressor. The C_{out} is the output to the compressor in the next significant stage.

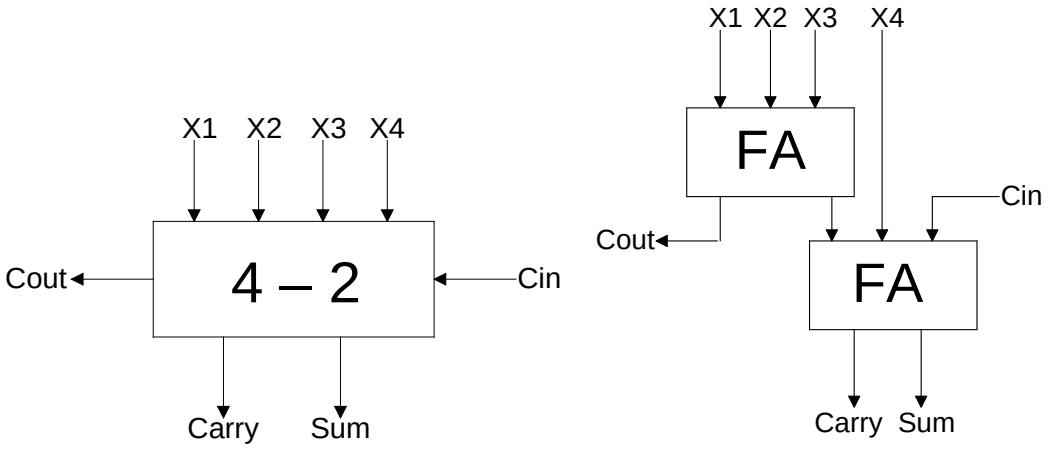


Figure 3.4 A 4-2 Compressor Block

Similar to the 3-2 compressor, a 4-2 compressor is governed by the basic equation

$$X1+X2+X3+X4+Cin = Sum + 2*(Carry + Cout) \quad (3.2)$$

The standard implementation of the 4-2 compressor can be done using 2 full Adder cells as shown in Fig 3.4 [1-3, 27,28].

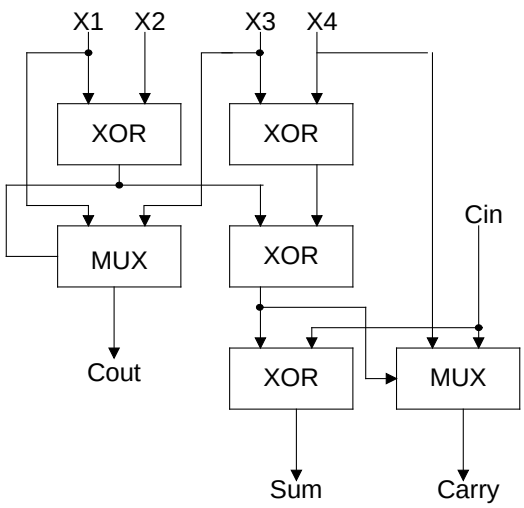


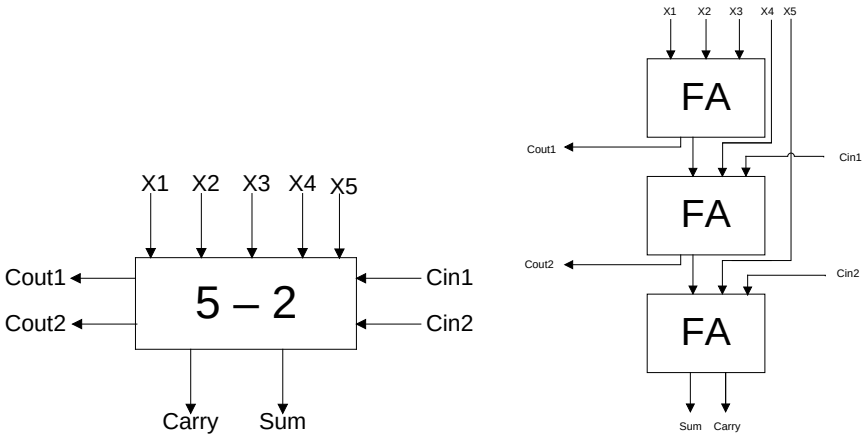
Figure 3.5 A 4-2 compressor implemented with full adders

When the individual full adders are broken into their constituent XOR blocks, it can be observed that the overall delay is equal to 4*Δ-XOR gates (where Δ refers to delay) as shown in Fig 3.4.

The block diagram in Fig 3.5 shows the existing architecture for the implementation of the 4-2 compressor with a delay of 3*Δ-XOR gates [1-3, 27-28]. But in this architecture, the fact that both the output and its complement are available at every stage was not taken into account [28].

3.3.3 5-2 Compressor

The 5-2 Compressor block has 5 inputs X1, X2, X3, X4 and X5 and 2 outputs, Sum and Carry, along with 2 input Carry bits (Cin1, Cin2) and 2 output Carry bits (Cout1,Cout2) as shown in Fig.3.6(a). Input Carry bits are the outputs from the previous lesser significant compressor block and the output Carry bits are passed on to the next higher significant compressor block.



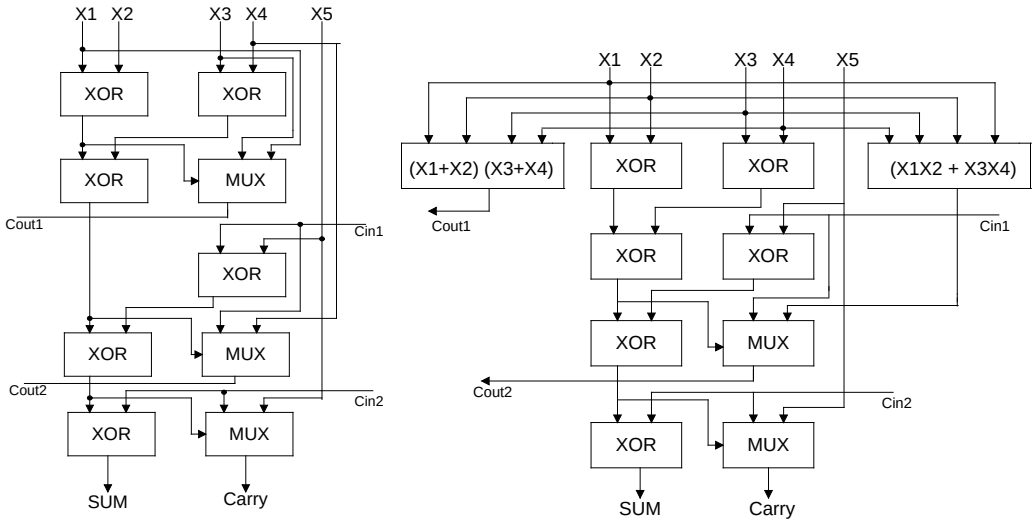
(a) (b)

Figure 3.6 (a) A 5-2 compressor block (b) Conventional implementation of a 5-2 compressor block

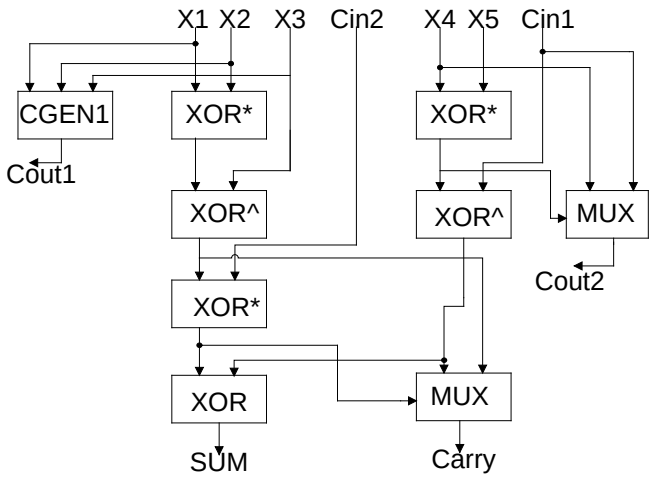
The basic equation that governs the function of a 5-2 compressor block is given below

$$X1+X2+X3+X4+X5+Cin1+Cin2=Sum+2*(Carry + Cout1 + Cout2) \qquad (3.3)$$

Conventional implementation of the compressor block is shown in Fig 3.6(b) where 3 cascaded full adder cells are used [27-28]. When these full adders are replaced with their constituent blocks of XOR gates, then it can be observed that the overall delay is equal to (6*Δ-XOR) for the Sum or Carry output.



(a) (b)



(c)

Figure 3.7 Existing architectures of a 5-2 compressor

Many designs of a 5:2 compressor have been proposed where the delay has been reduced to 5*Δ-XOR gates as shown in Fig 3.7(a) which have further been reduced to 4*Δ-XOR gates as shown in Fig 3.7 (b) & (c).

3.4 Design and Implementation of Efficient Compressors

3.4.1 3-2 Compressor

In CMOS implementation, the gates like OR and AND require implementation of NOR and NAND gates followed by an inverter. Thus, from OR and AND gates, we can obtain NOR and NAND outputs without any extra hardware. This technique is used to design a XOR-XNOR pair gate which is shown in Fig 3.8

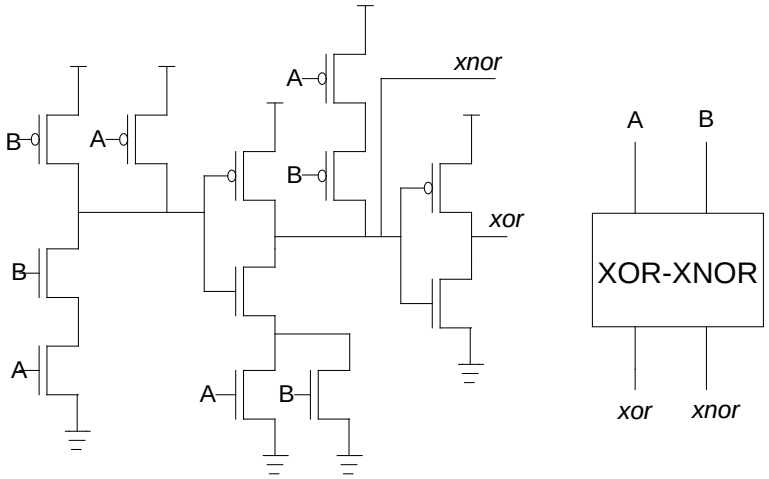


Figure 3.8 CMOS Implementation of XOR/XNOR Gate

A 3-2 compressor can be implemented by the following expressions.

$$Sum = X1 \oplus X2 \oplus X3 \tag{3.4}$$

$$Carry = (X1 \oplus X2) \cdot X3 + \overline{(X1 \oplus X2)} \cdot X1 \tag{3.5}$$

A gate-level implementation of these expressions has earlier been shown in Fig 3.3. In the existing design, the output of the first XOR gate and X3 are given as inputs to second stage XOR gate. This XOR gate can be replaced by a multiplexer which reduces the delay, as multiplexer has less delay compared to XOR gate [1-6, 29].

In the proposed design shown in Fig.3.9, the fact that both the XOR and XNOR outputs are computed, is efficiently used to reduce the delay by replacing the second XOR gate with a MUX. This is due to the availability of the select bit i.e. X3 at the MUX block before the inputs arrive. Thus, the time taken for the switching ON of the transistors is reduced in the critical path [30].

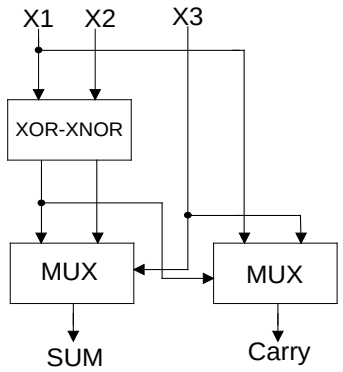


Figure 3.9 Proposed design of the 3-2 Compressor

The equations governing the proposed (3, 2) compressor outputs are shown below.

$$\text{Sum} = (X1 \oplus X2) \cdot \overline{X3} + \overline{(X1 \oplus X2)} \cdot X3 \tag{3.6}$$

$$\text{Carry} = (X1 \oplus X2) \cdot X3 + \overline{(X1 \oplus X2)} \cdot X1 \tag{3.7}$$

3.4.2 4-2 Compressor

In this design also, the fact that both the output and its complement are available at every stage is neglected [28]. Thus replacing some XOR gates with multiplexers results in a significant improvement in delay.

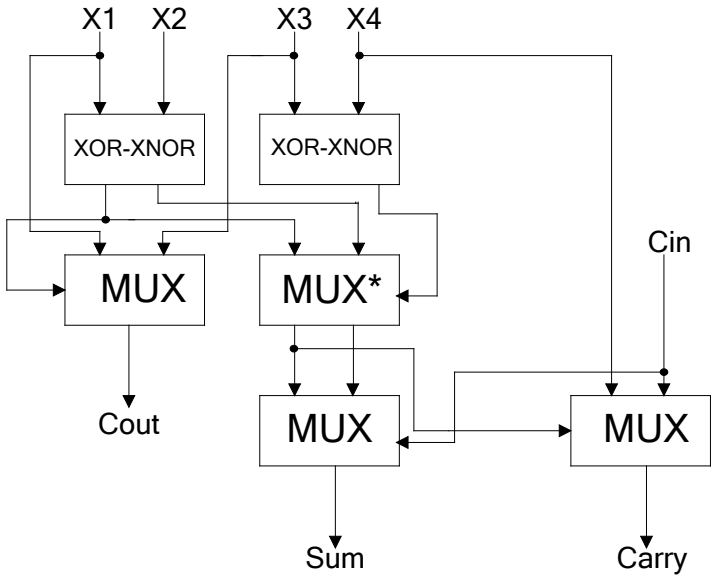


Figure 3.10 Proposed 4-2 Compressor Design

Like in previous case, the MUX block at the SUM output gets the select bit before the inputs arrive and thus the transistors are already switched ON by the time the inputs arrive. This minimizes the delay to a considerable extent [30] as shown in Fig 3.10.

The equations governing the outputs are shown below

$$Sum = X1 \oplus X2 \oplus X3 \oplus X4 \oplus Cin \tag{3.8}$$

$$Cout = (X1 \oplus X2) \bullet X3 + \overline{(X1 \oplus X2)} \bullet X1 \tag{3.9}$$

$$Carry = (X1 \oplus X2 \oplus X3 \oplus X4) \bullet Cin + \overline{(X1 \oplus X2 \oplus X3 \oplus X4)} \bullet X4 \tag{3.10}$$

The MUX* structure in Fig 3.10 is a multiplexer implemented using transmission gate logic style shown in Fig. 3.11. This design of the multiplexer is faster and also consumes lesser power than the CMOS design but requires buffers to enhance the driving capability. Therefore, these types of multiplexers can be used where there are a CMOS transistors at its input and output, because CMOS has good driving capability. Thus, transmission gate multiplexers are used in the intermediate stage, thereby increasing the performance.

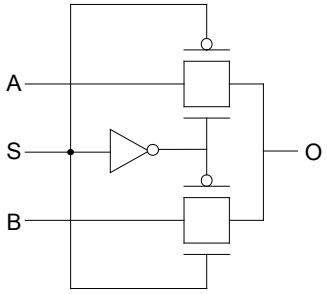


Figure 3.11 Transmission Gate Implementation of a multiplexer

3.4.3 5-2 Compressor

In the proposed design of the 5-2 compressor the most important change is to efficiently use the outputs generated at every stage. This is done by replacing some XOR blocks with MUX blocks as shown in Fig 3.12.

Also the select bits to the multiplexers in the critical path are made available much ahead of the inputs so that the critical path delay is minimized. For example, the Cout2 output from the previous lesser significant compressor block is utilized as the select bit after a stage it is produced so that the MUX block is already switched ON and the output is produced as soon as the inputs arrive. Also if the output of the multiplexer is used as select bit for another multiplexer, then it can be used efficiently in a similar manner because the negation of select bit is also required, as shown in Fig 3.7. Thus an extra stage to compute the negation can be saved. Similarly replacing the XOR block in the second stage with a MUX block reduces the delay because the select bit x3 is already available and the time taken for the transistor switching to take place happens in parallel with the computation of the inputs of the block [30].

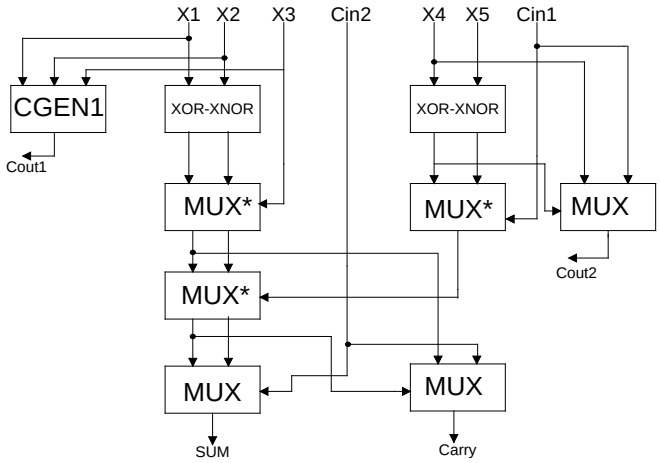


Figure 3.12 Proposed design of the 5-2 compressor

As mentioned before, in all the general implementations of the XOR or MUX block, in particular CMOS implementation, the output and its complement are generated. But in the existing design this advantage is not being utilized fully [27-28]. In the proposed design these outputs are utilized efficiently by using multiplexers at particular stages in the circuit. Also additional inverter stages are eliminated. This in turn contributes to the reduction of delay, power consumption and transistor count (area).

The equations governing the outputs are shown below:

$$Sum = X1 \oplus X2 \oplus X3 \oplus X4 \oplus X5 \oplus Cin1 \oplus Cin2 \tag{3.11}$$

$$Cout1 = (X1 + X2) \bullet X3 + X1 \bullet X2 \tag{3.12}$$

$$Cout2 = (X4 \oplus X5) \bullet Cin1 + \overline{(X4 \oplus X5)} \bullet X4 \tag{3.13}$$

$$Carry = ((X1 \oplus X2 \oplus X3) \oplus (X4 \oplus X5 \oplus Cin1)) \bullet Cin2 + \overline{((X1 \oplus X2 \oplus X3) \oplus (X4 \oplus X5 \oplus Cin1))} \bullet (X1 \oplus X2 \oplus X3) \tag{3.14}$$

In the carry generation module (CGEN1) shown in Fig.3.12, the above equation (3.12) is used to design the CMOS implementation of Cout1 as shown in Fig 3.13.

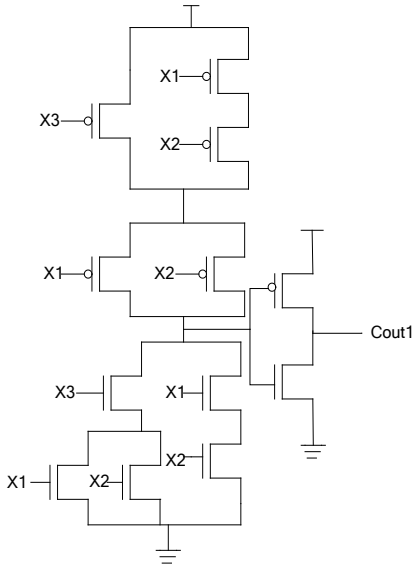


Figure 3.13 Carry Generation Module (CGEN1)

3.5 Designs of Existing Counters

A wide variety of parallel counters exist in literature which are been listed in [55-56]. The threshold gate counters proposed in [31-32] have been implemented with inverting threshold gates but have not been widely used due to difficulty in realizing large threshold gates with accurate thresholds. The switching tree counters proposed in [31-32] are implemented using

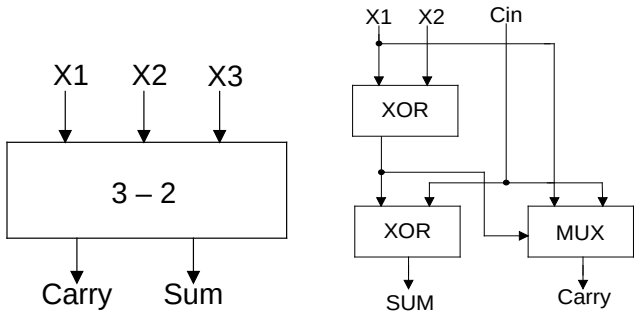
relay switches but the complexity of this approach grows as the square of number of inputs, making the realization of large counters prohibitively costly The quasi-digital counters [31-32] and residue-threshold based counters are also not widely used as they do not appear to be attractive with current technology.

The most popular and widely used existing counters are successive doubling counters [31-32] and synthesized counters [31-32]. In successive doubling approach, full adders are used to implement the counters. Typically, a (m, n) counter is implemented by using (n-m) full adders with a critical path delay of (2m-3) full adders. The synthesized counters are manually optimized [31-32] or obtained by computer aided design [31-32]. These counters have a lesser delay but a higher hardware complexity than the full adder based counters.

3.5.1 (3, 2) Counter

A (3, 2) counter takes 3 inputs X1, X2, X3 and generates 2 outputs, the Sum bit, and the Carry bit as shown in Fig 3.14 (a). The (3, 2) counter is governed by the basic equation,

$$X1 + X2 + X3 = Sum + 2*Carry \qquad (3.15)$$



(a) (b)

Figure 3.14 (a) 3-2 Counter (b) Conventional Implementation of the 3-2 Counter

Existing designs as shown in Fig 3.14(b) employs two XOR gates in the critical path.

3.5.2 (7, 3) Counter

A (7, 3) counter takes 7 inputs X1-X7 and generates 3 outputs, the Sum bit S, and the Carry bits C1 and C2 as shown in Fig.3.15. The outputs of the counter are represented as a binary number $\{C_2C_1S\}$. The basic equation of (7, 3) counter is:

$$X1 + X2 + X3 + X4 + X5 + X6 + X7 = S + 2*C1 + 4*C2 \qquad (3.16)$$

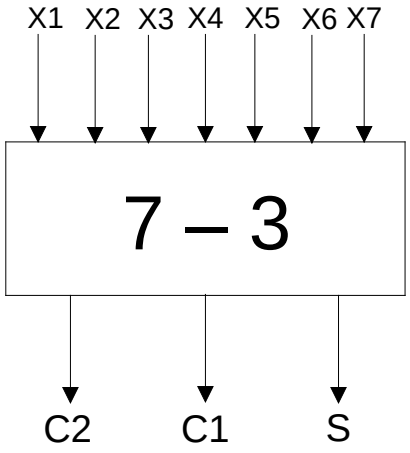
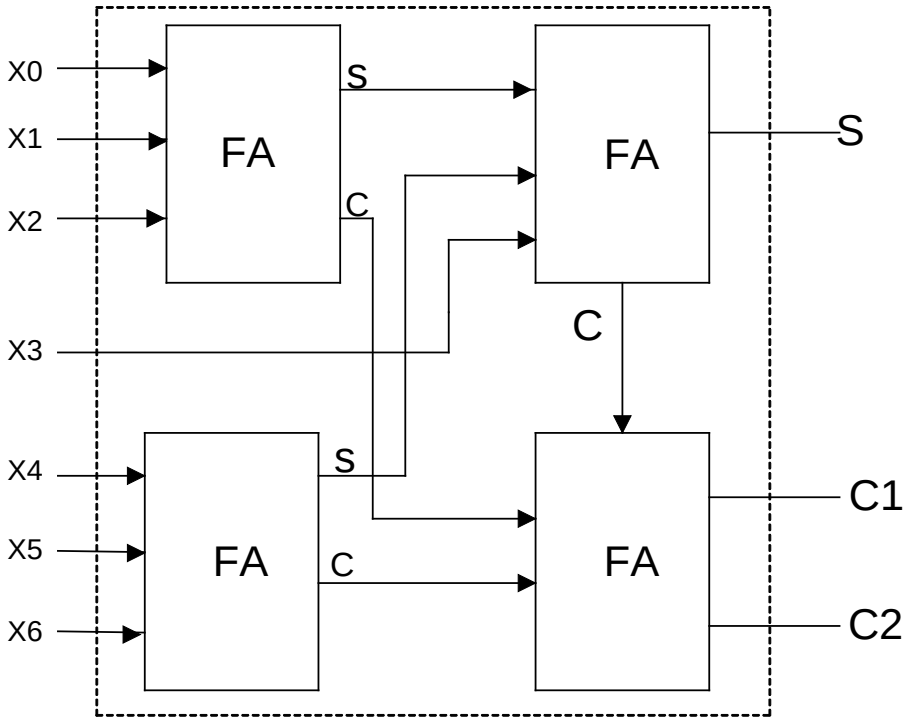
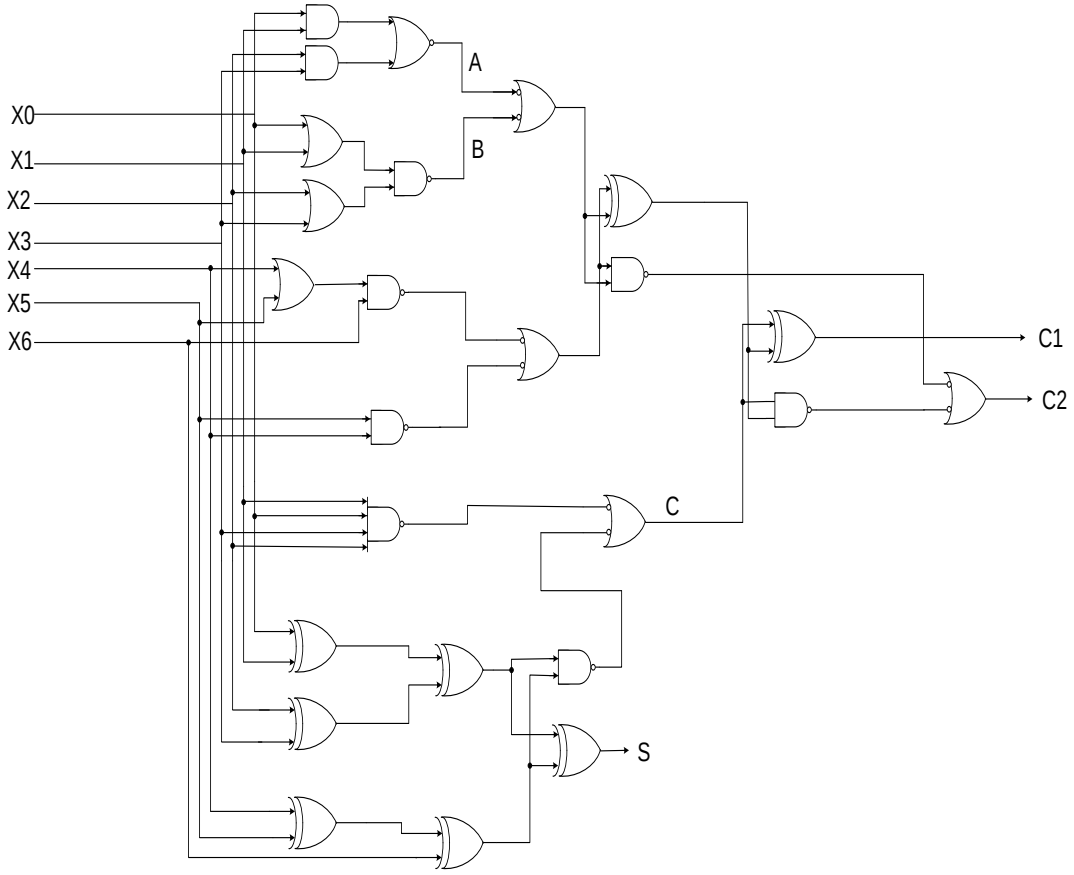


Figure 3.15 (7, 3) Counter block diagram.

The existing implementations of (7, 3) counter are shown in Fig 3.16 [31-32]. The full adder based counter circuit shown in Fig 3.16(a) has a delay of (6*Δ-XOR gates) while the synthesized counter circuit shown in Fig 3.16(b) has a delay of approximately (4*Δ-XOR gates) [31-32].



(a)



(b)

Figure 3.16 Existing (7, 3) Counter designs (a) adder based counter (b) synthesized counter

3.5.3 (15, 4) Counter

The (15, 4) counter takes 15 inputs (X0-X14) and generates 4 outputs S0, S1, S2 and C0 with weights of one, two, four and eight respectively. Thus, the output of the counter is represented as $\{S_0S_1S_2C_0\}$. The existing design of (15, 4) counter is shown in Fig 3.17 [31-32] with a critical path delay of (5*Δ-Full Adders) or (10*Δ-XOR gates).

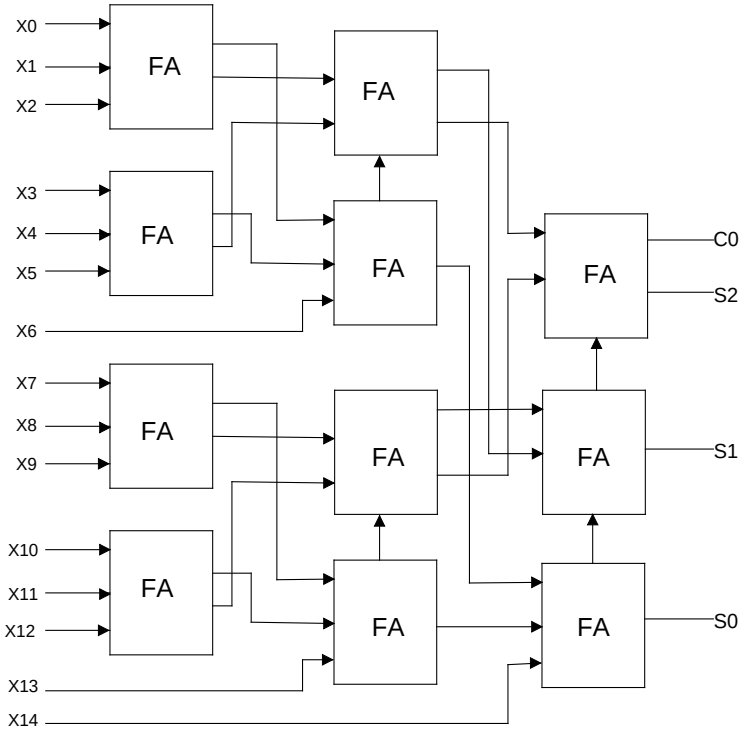


Figure 3.17 Exiting design for (15,4) counter.

3.5.4 (31, 5) Counter

Similarly, the (31, 5) counter has 31 inputs (X0-X30) and Generates 5 outputs S0, S1, S2, S3 and C0 with weights of one, two, four, eight and sixteen respectively. Thus, the output is represented as $\{S_0S_1S_2S_3C_0\}$. The existing design for a (31, 5) counter can be found in [31-32] which has a critical path delay of (7*Δ-Full Adders) .

3.6 Design and Implementation of Efficient Parallel Counters

3.6.1 (3, 2) Counter

In the proposed design as shown in Fig 3.18, the fact that both XOR and XNOR outputs values are computed is efficiently used to reduce delay by replacing the second XOR with a MUX as explained in the previous section. Also in the implementation of the second multiplexer which generates Carry, select bit and its complement are generated in the XOR-XNOR block in the previous stage, thus eliminating the need for additional inverters, thereby reducing the delay, area and power [30, 33].

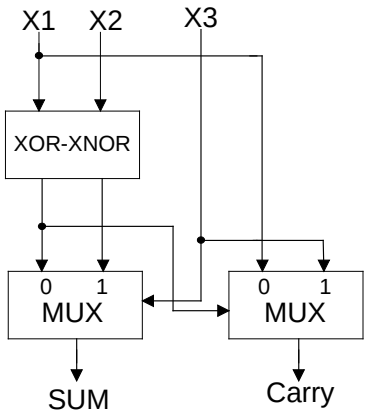


Figure 3.18 Proposed Implementation of the (3,2) Counter

Thus the proposed implementation shown in Fig.3.18 has a delay of (Δ-XOR +Δ-MUX) which is less when compared to existing design (Δ refers to delay).

3.6.2 (7, 3) Counter

In this design, outputs generated at every stage are efficiently made use of, as explained earlier in section 3.4. The proposed implementation of (7,3) counter is given in Fig 3.19. This counter has a delay of (Δ-XOR gate + 2*Δ-MUX) for S and (Δ-XOR gate + 3*Δ-MUX) for C1 and C2 and a comparison of this design with the exiting designs can be seen in Table 3.1 [33].

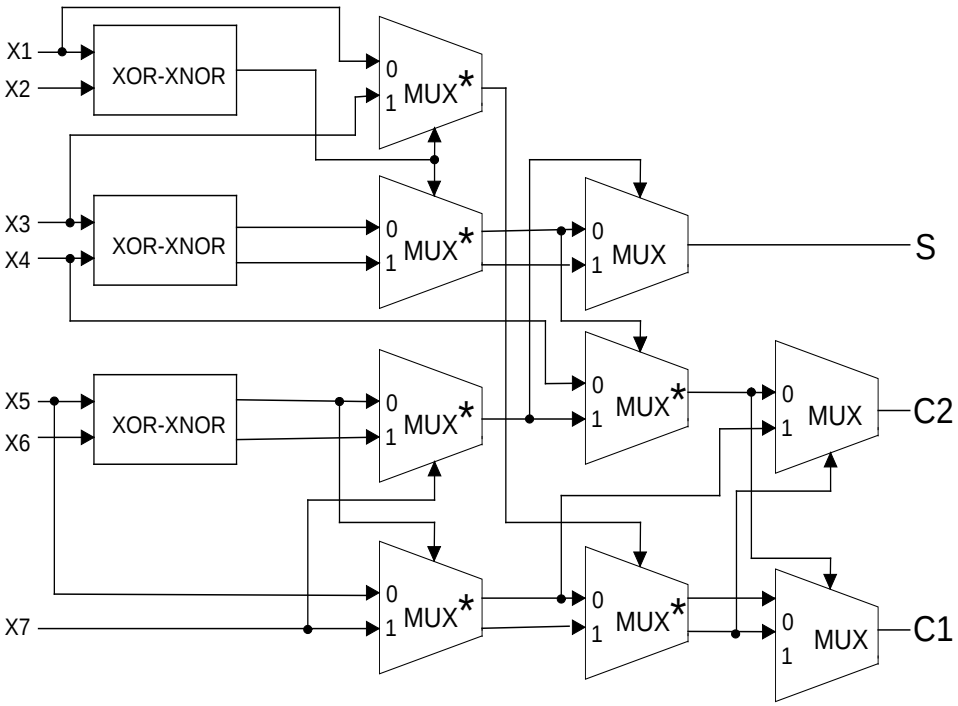


Figure 3.19 Proposed (7, 3) counter design.

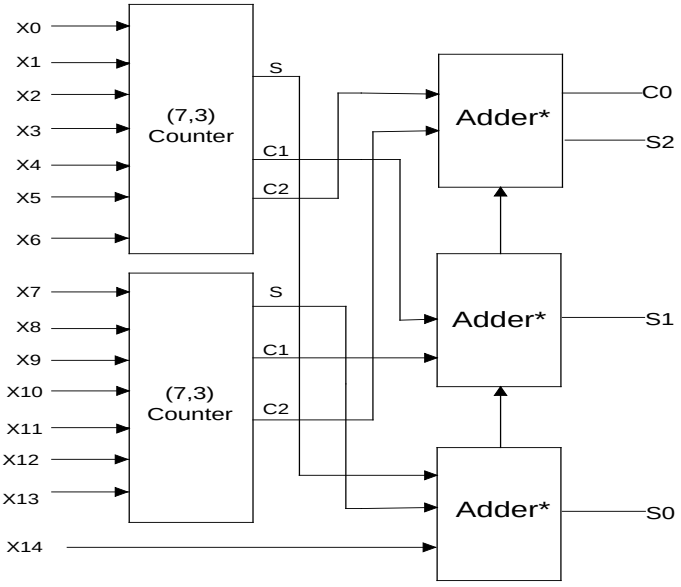
Table 3.1.Comparison of existing (7, 3) with proposed design

(7, 3)Counter	Delay	Gate Complexity
Full-Adder Based	6*Δ-XOR	8 XORs + 4 MUXes
Synthesized Counter [31]	4*Δ-XOR	7 XORs + 18 basic gates (OR, NAND, AND, NOR)
CAD synthesized counter [31]	Δ-MUX + 9 basic gates	2 MUXes + 32 basic gates
Proposed	Δ-XOR + 3*Δ-MUX	3 XORs + 9 MUXes

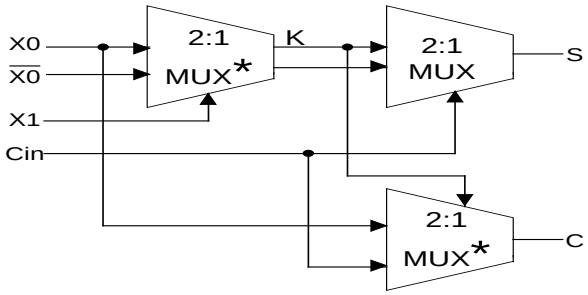
It can be seen from the above table that the proposed design results in reduced delay as compared to the existing design.

3.6.3 (15, 4) counter

The proposed design for (15, 4) counter is given in Fig 3.20(a) and the adder* module used in this design in given in Fig 3.20 (b). This design is based on (7, 3) counter proposed earlier. The delay of the proposed counter is (Δ-XOR gate + 4*Δ-MUX) for S0, (Δ-XOR gate+ 5*Δ-MUX) for S1 and (Δ-XOR gate + 6*Δ-MUX) for S2 and C0, thus having a critical path delay of (Δ-XOR gate + 6*Δ-MUX) or (Δ-(7, 3) counter + 3*Δ-MUX) as the S output of the (7, 3) counter is obtained one stage before the Carry bits as shown in Table 3.2 [33].



(a)



(b)

Figure 3.20 (a) Proposed design for (15, 4) counter (b) Adder* module used in the counter.

Table 3.2 Comparison of existing (15, 4) with proposed design

(15, 4)Counter	**Delay**	**Gate Complexity**
Full-Adder Based	10*Δ-XOR	22 XORs + 11 MUXes
CAD Synthesized Counter [31]	3*Δ-MUX + 3*Δ-XOR + 5 basic gates	10 MUXes + 49 basic gates
Proposed	Δ-XOR + 6*Δ-MUX	6 XORs + 27 MUXes

It can be seen from the above table that the proposed design results in reduced delay as compared to the existing design.

3.6.4 (31, 5) counter

The proposed design for the (31,5) counter is given in Fig 3.21. It has a critical path delay of (Δ-(15, 4) counter + 4*Δ-MUX) which is equal to (Δ-XOR gate + 8*Δ-MUX), while the critical path delay of existing design built with full adders is (7*Δ-Full Adder) or (14*Δ-XOR gate) as shown in Table 3.3.

Another advantage of the proposed counter design is that the outputs of the lower order counters ((15, 4) and (7, 3)) are obtained with a delay of one stage (one MUX) each. Hence in Adder* blocks, the output K (in Fig 3.20(b)) and the Cin are obtained almost at the same time, thereby reducing the speed of the circuit Also, if K=0, the Carry bit (Cout) doesn't depend on Cin and hence the delay of the next Adder* block is reduced considerably [33].

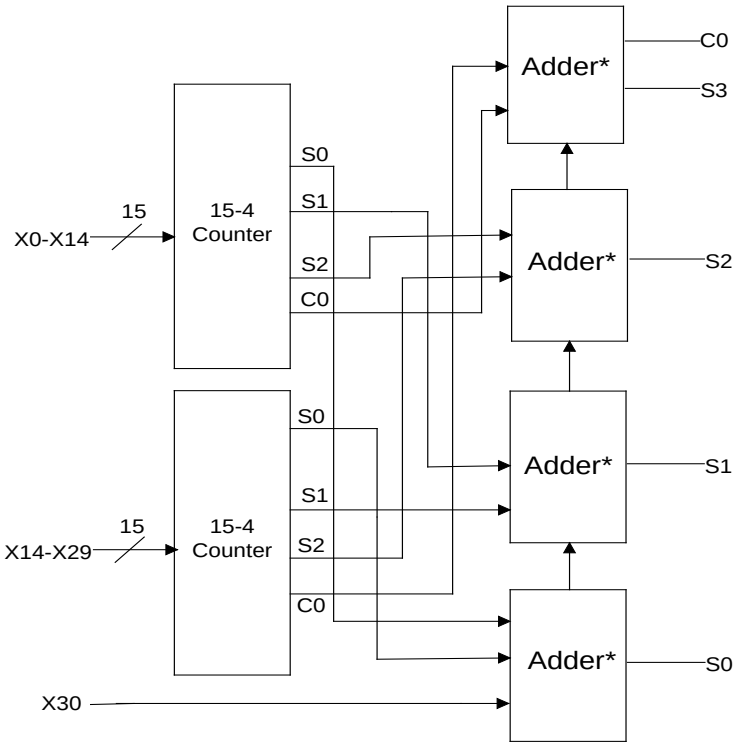


Figure 3.21 Proposed design for (31, 5) counter.

Table 3.3. Comparison of existing (31, 5) with proposed design

(31, 5)Counter	Delay	Gate Complexity
Full-Adder Based	14*Δ-XOR	52 XORs + 26 MUXes
Proposed	Δ-XOR + 8*Δ-MUX	12 XORs + 66 MUXes

It can be seen from the above table that the proposed design results in reduced delay as compared to the existing design.

3.6.5 (m, n) Parallel Counter

A general (m, n) parallel counter can be designed using (m-n) full adder blocks [33]. The output of a (m, n) counter is represented as {$S_0S_1S_2S_3….S_{n-2}C_0$}. Based on the proposed counter designs in section 3.6, any (m, n) parallel counter, for m>7, can be designed as shown in Fig 3.22. That is, every (m, n) counter can be designed by using [(m-1)/2, n-1] counters and Adder* blocks. Thus they can be implemented recursively by using lower order counters as the building blocks. Since the lower order counters are already minimized for maximum efficiency as described earlier, the (m, n) counter can be expected to have better performance characteristics [33]. The generalized delay of the (m, n) counter can be expressed as [Δ ((m-1)/2,n-1)counter + (n-1) *Δ-MUX)].

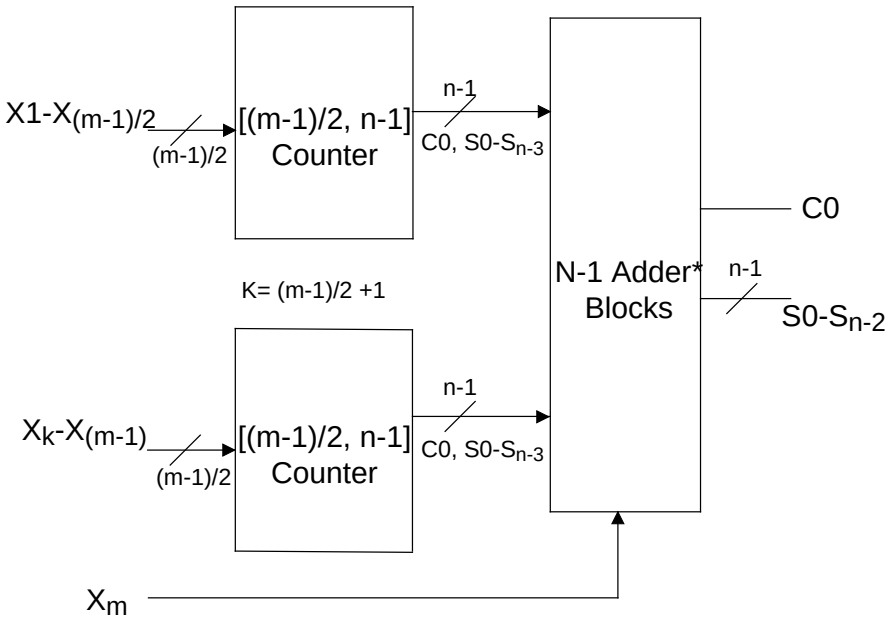


Figure 3.22 Proposed design for (m,n) counter.

3.7 Simulation Results and Analysis

All the simulations have been carried out using Cadence Tool Suite. The calculation of power and delay are carried out using the Virtual Analog Simulation Tool. All the schematics and layouts have been carried out using the CMOS 0.18-µm technology. The simulations are

performed under various voltages ranging from 0.9V to 3.3V and the inputs are toggled at a frequency of 100 MHz.

3.7.1 Compressor

The proposed compressor designs are compared with the existing ones and the performance parameters are detailed in Tables 3.4 to 3.9. Tables 3.4, 3.6, 3.8 and Fig. 3.23 (a) compare power consumption of the proposed compressors with the existing designs. Tables 3.5, 3.7, 3.9 and Fig. 3.23 (b) do the same for the delay.

Table 3.4 Power consumption for 3-2 compressor (nW)

	0.9V	**1.2V**	**1.8V**	**2.5V**	**3.3V**
Existing	1.74	4.3	9.65	15.9	27.5
Proposed	1.25	3.26	7.7	13.4	24.5

Table 3.5 Delay for 3-2 compressor (ns)

	0.9V	**1.2V**	**1.8V**	**2.5V**	**3.3V**
Existing	3.1	2.1	1.35	1.15	0.9
Proposed	2.94	1.92	1.25	1.08	0.86

Table 3.6 Power consumption for 4-2 compressor (nW)

	0.9V	**1.2V**	**1.8V**	**2.5V**	**3.3V**
Existing	4.5	11.8	20.2	31.5	53.5
Proposed	3.85	10.4	17.5	28.3	46.4

Table 3.7 Delay or 4-2 compressor (ns)

	0.9V	**1.2V**	**1.8V**	**2.5V**	**3.3V**
Existing	4.8	2.88	1.8	1.44	1.21
Proposed	4.4	2.5	1.5	1.28	1.1

Table 3.8 Power consumption for 5-2 compressor (nW)

	0.9V	**1.2V**	**1.8V**	**2.5V**	**3.3V**
Existing	25.25	44.95	101.25	240.5	355
Proposed	20.32	35.59	87.15	177.5	241.5

Table 3.9 Delay for 5-2 compressor (nS)

	0.9V	**1.2V**	**1.8V**	**2.5V**	**3.3V**
Existing	6.7	4.2	2.3	1.78	1.5
Proposed	6.1	3.75	1.7	1.41	1.28

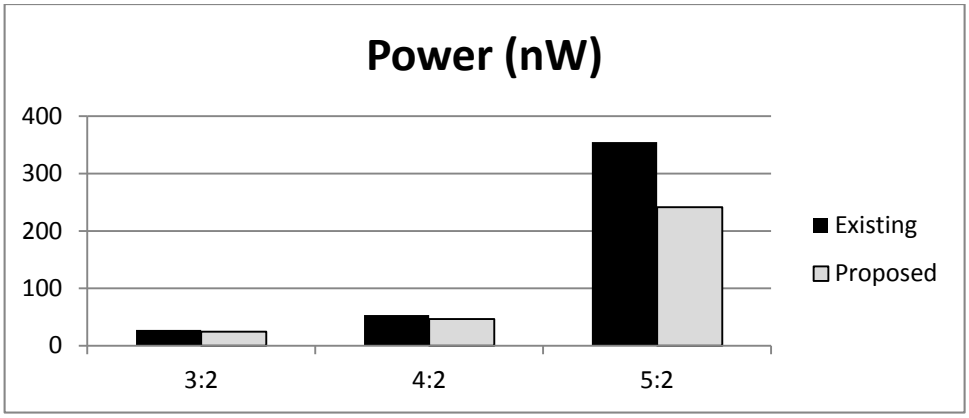


Figure 3.23 (a)

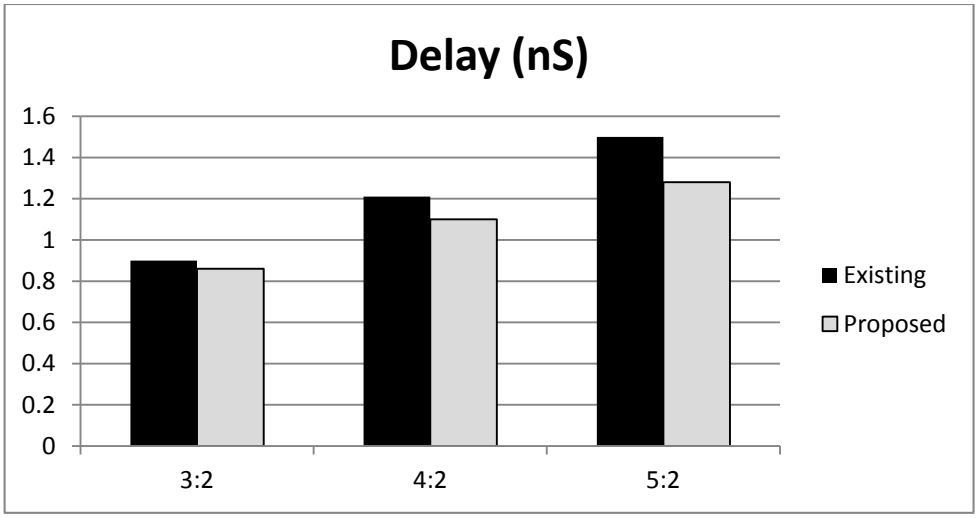


Figure 3.23 (b)

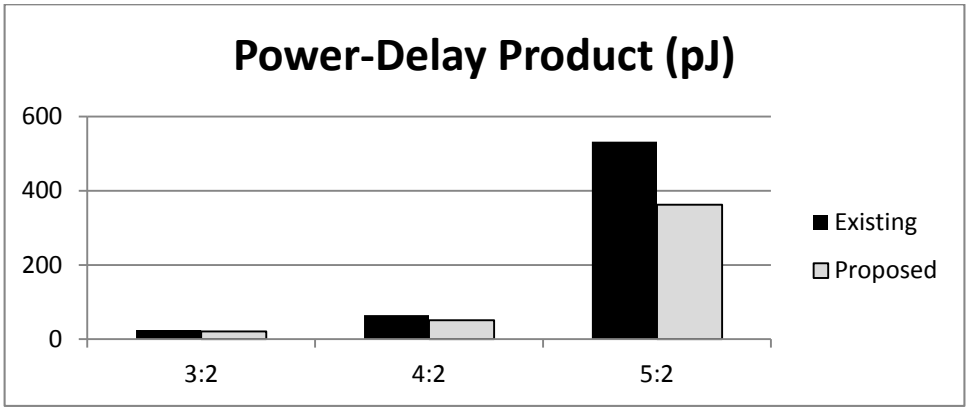


Figure 3.23 (c)

Figure 3.23 Comparison of proposed 3:2, 4:2 and 5:2 compressors with existing compressor in terms of (a) Power (b) Delay (c) Power-Delay Product

From the Tables and the figures, it can be observed that the proposed 3:2 compressor is 7% faster and consumes 10.2% lesser power than the existing designs. Also, the 4:2 compressor is 33.3% faster and consumes 15% lesser power than the existing ones. Further, the 5:2 compressor consumes 13.2% less power and is 26% faster than the existing ones when operating at 1.8V. The improvement in the power-delay product is 36.4%, 27.8% and 24% in the proposed 5-2 compressor, 4-2 compressor and 3-2 compressors respectively [30].

As mentioned in section 3.4.2, the MUX* blocks in the proposed design can be implemented using transmission gate (CMOS+) design. This new implementation is compared with the CMOS implementation and the results are detailed in Tables 3.10 to 3.12 below.

Table 3.10 Delay for 5:2 compressor with MUX* in CMOS and CMOS+ design (nS)

	0.9V	**1.2V**	**1.8V**	**2.5V**	**3.3V**
MUX* AS CMOS	6.1	3.75	1.7	1.41	1.28
MUX* AS CMOS+	5.304348	3.26087	1.478261	1.226087	1.113043

Table 3.11 Power Consumption for 5:2 compressor with MUX* in CMOS and CMOS+ design (nW)

	0.9V	**1.2V**	**1.8V**	**2.5V**	**3.3V**
MUX* AS CMOS	20.32	35.59	87.15	177.5	241.5
MUX* AS CMOS+	19.35238	33.89524	83	169.0476	230

Table 3.12 Power Delay Product for 5:2 compressor with MUX* in CMOS and CMOS+ design (aJ)

	0.9V	**1.2V**	**1.8V**	**2.5V**	**3.3V**
MUX* AS CMOS	123.952	133.4625	148.155	250.275	309.12
MUX* AS CMOS+	102.6518	110.528	122.6957	207.2671	256

It is evident from the above tables that an improvement in the delay of 14.6%, power of 5.1% and power-delay product of 18.2% has been obtained when compared to the CMOS implementation [30].

3.7.2 Counter

The proposed counter designs are compared with the existing ones and the performance parameters are detailed in Tables 3.13 to 3.18. Tables 3.13, 3.15, 3.17 and Fig. 3.24 (a) compare power consumption of the proposed counter designs with the existing designs. Tables 3.14, 3.16, 3.18 and Fig. 3.24 (b) do the same for the delay.

Table 3.13 Power for 7,3 counter (nW)

	0.9V	**1.2V**	**1.8V**	**2.5V**	**3.3V**
Existing	6.96	17.2	38.6	63.6	110
Proposed	5.725	15.29	29.05	48.4	83.15

Table 3.14 Delay results for 7,3 counter (nS)

	0.9V	**1.2V**	**1.8V**	**2.5V**	**3.3V**
Existing	9.3	6.3	4.05	3.45	2.7
Proposed	5.87	3.46	2.125	1.82	1.53

Table 3.15 Power for 15,4 counter (nW)

	0.9V	**1.2V**	**1.8V**	**2.5V**	**3.3V**
Existing	19.14	47.3	106.15	174.9	302.5
Proposed	15.2	40.36	81.2	137	240.5

Table 3.16 Delay for 15,4 counter (nS)

	0.9V	**1.2V**	**1.8V**	**2.5V**	**3.3V**
Existing	15.5	10.5	6.75	5.75	4.5
Proposed	9.25	5.74	3.675	3.1	2.69

Table 3.17 Power for 31,5 counter (nW)

	0.9V	**1.2V**	**1.8V**	**2.5V**	**3.3V**
Existing	45.24	111.8	250.9	413.4	715
Proposed	35.4	93.76	193.2	327.6	579

Table 3.18 Delay for 31,5 counter (nW)

	0.9V	**1.2V**	**1.8V**	**2.5V**	**3.3V**
Existing	21.7	14.7	9.45	8.05	6.3
Proposed	15.13	9.58	6.175	5.26	4.41

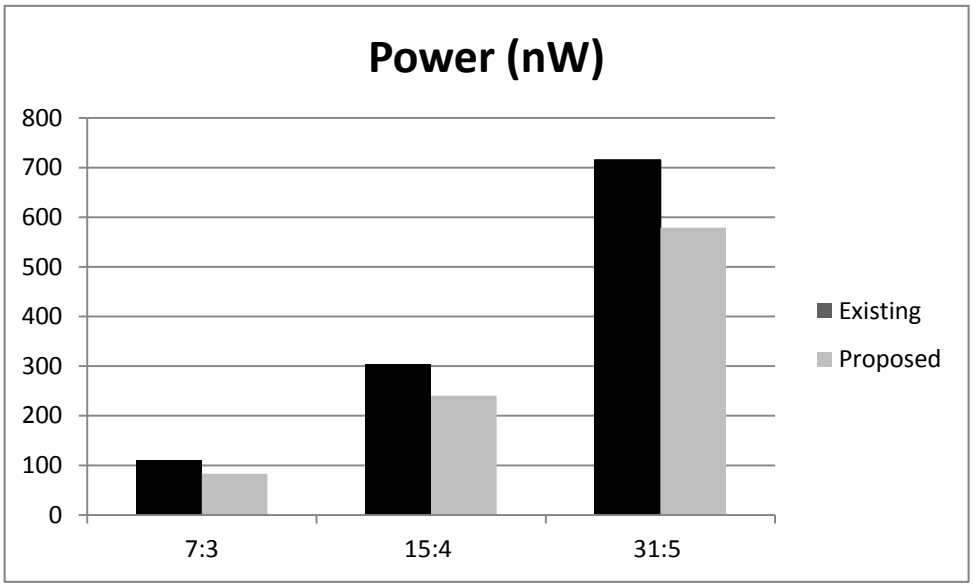


Figure 3.24 (a)

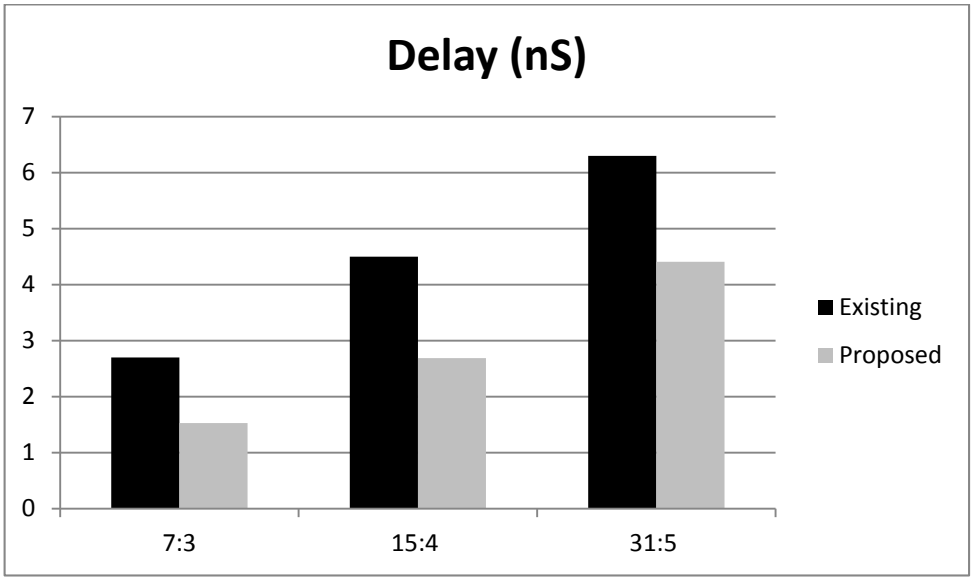


Figure 3.24 (b)

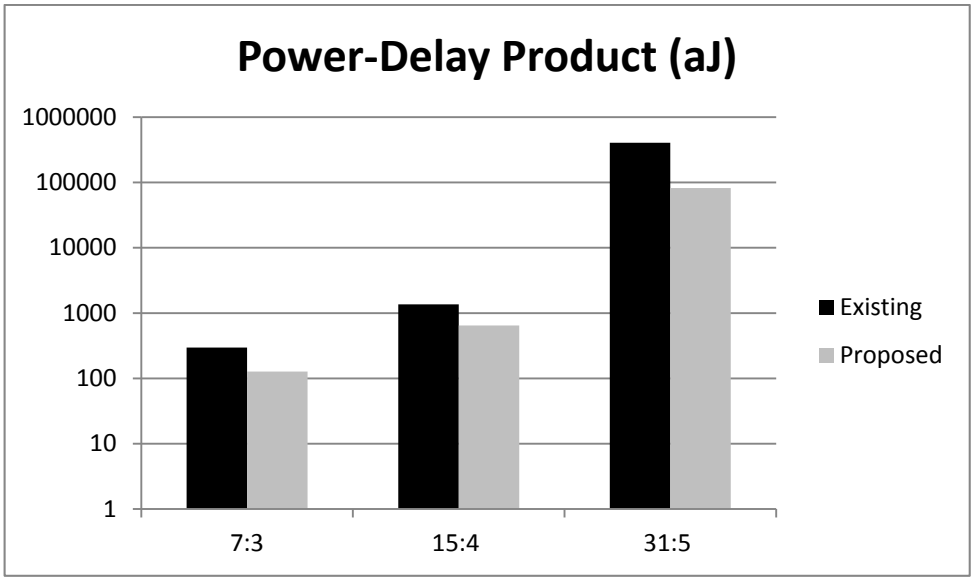


Figure 3.24 (c)

Figure 3.24 Comparison of proposed 7:3, 15:4 and 31:5 counters with existing counters in terms of (a) Power (b) Delay (c) Power-Delay Product

From the Tables and figures, it can be observed that the delay and power consumption of proposed counters are less when compared to existing counter designs.

As mentioned in section 3.4, the MUX* blocks in the proposed design can be implemented using Transmission gate (TG). This implementation has been compared with the CMOS implementation and the results are detailed in Table 3.19 and 3.20.

Table 3.19. Comparison of Average Power (nW) and Delay (nS) of the proposed counters with MUX* as CMOS and Transmission Gate(TG) logic

Counter	MUX* as CMOS		MUX* as TG	
	Delay	Power	Delay	Power
(3,2)	8.2	10.12	-	-
(7,3)	42.75	21.42	45.32	24.41
(15,4)	45.45	24.5	48.92	27.55
(31,5)	47.28	26.15	49.88	28.64

Table 3.20. Comparison of Average Power-Delay Product (aJ) of the proposed counters with MUX* as CMOS and Transmission Gate (TG) logic

Counter	MUX* as CMOS	MUX* as TG
	Power-Delay Prod.	Power-Delay Prod.
(3,2)	9.35	-
(7,3)	35.66	37.15
(15,4)	38.25	39.66
(31,5)	40.3	41.72

It is evident from the Table 3.19 that the proposed design for (7,3), (15,4) and (31,5) counters consume 20-30% lesser power and are 40-50% faster than the existing ones. From Table 3.20, it can be observed that there is an improvement of 35-40% in power-delay product. Thus the implementation of MUX* block with transmission gate (TG) logic found to be better in terms of power, delay and power-delay product when compared to the implementation with CMOS logic [33].

3.8 Conclusions

In this chapter, design of efficient counters and compressors are presented. The design for the 3-2, 4-2 and 5-2 compressor are analyzed using CMOS and CMOS+ implementations of XOR and the MUX blocks. New 3-2, 4-2 and 5-2 compressor designs have been proposed and compared with the existing ones. The proposed designs perform better than the existing ones in every aspect i.e., area, power, delay and power-delay product over the complete voltage range simulated. A set of efficient lower order counters like (3,2), (7,3), (15,4) has been proposed for

efficient higher order counter implementation. Further, a generalized (m, n) counter is proposed which uses lower order counters as basic building blocks resulting in an efficient counter design.

Chapter 4
Design of Function Specific Arithmetic Circuits

Contents

4.1 Introduction:

As mentioned earlier, the main purpose of this thesis is to develop efficient functional arithmetic circuits which when put together should result in an efficient Arithmetic and Logic Unit (ALU). In addition to design described in earlier chapter, circuits such as Incrementer/Decrementer circuit, 2's complement circuit and priority encoder circuit, which are

widely used in many digital systems [1-6, 34-42] are designed and analyzed in this chapter. A brief introduction to these circuit is given below:

4.1.1 Incrementer/Decrementer Circuit

The Increment/Decrement (INC/DEC) circuit is a digital module which can count up or down by one. It is a common building block in many digital systems like microprocessor, and microcontroller as a part of the address generation unit, program counter, etc…

4.1.2 2's Complement Circuit

The 2's complement circuit forms an important part of computer systems that are based on 2's complement number representation. An application of 2's complement circuit is in multipliers that need to find the 2's complement of multiplicand for the negative encoding in booth algorithm.

4.1.3 Priority Encoder Circuit

Priority encoder circuit is a circuit which makes input with highest priority active and all other inputs are inactive. This circuit is used for arbitrating among N units that are all requesting access to a shared resource and is used in interrupt controllers and conditional handlers.

4.2 Existing Designs

4.2.1 Increment/Decrement circuits

There are various designs for binary INC/DEC circuits in literature[34-36]. Many of these designs use adder to implement the increment/decrement operation. An adder/subtractor can be used for these operations by making one input as operand and other input as '1'. But usage of entire an adder for single bit addition increases delay and power when compared to a dedicated INC/DEC design.

The current designs of binary INC/DEC are mainly adder/subtractor-based, counter-based or Carry look-ahead adder-based as shown in Fig 4.1 (a) [34]. Recently, a MUX-based binary INC/DEC which is more efficient than the previous INC/DECs has been proposed in literature as shown in Fig 4.1 (b) [35]. This circuit has data-in MUX array, a decision block and data-out MUX array. While this design is efficient in terms of both speed and hardware complexity, when compared to adder based approaches, a series of (n-2) OR gates and a MUX in its critical path hampers the speed of the circuit to a certain extent.

An improvement to this circuit was proposed in [36], shown in Fig 4.2, which uses a look-ahead type approach resulting in reduced delay when compared to MUX-based INC/DEC. For example, an N-bit Carry look-ahead type results in (N/8) + 9 gates delay, when an 8-bit look-ahead block is used.

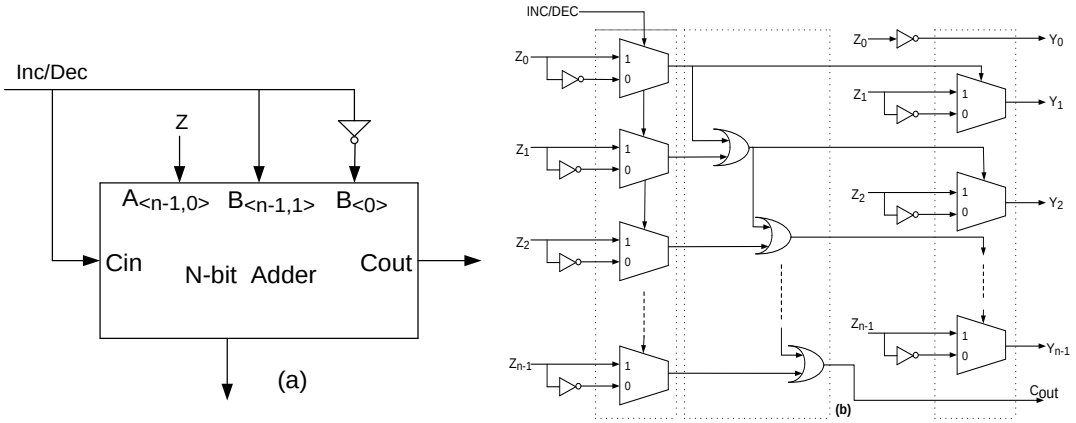


Figure 4.1 (a) Adder based (b) MUX based

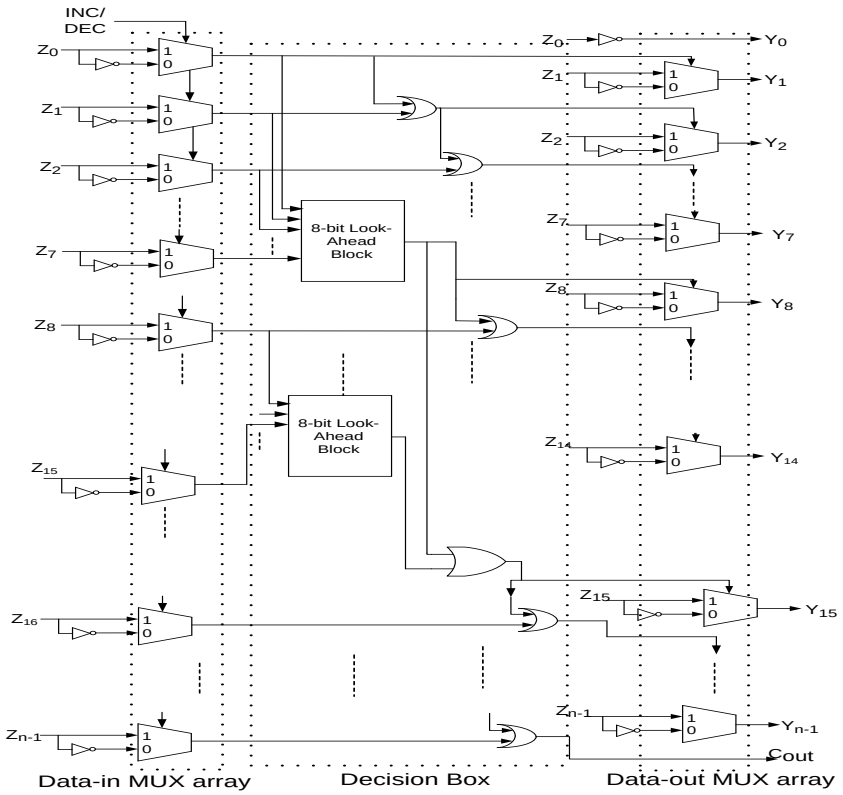


Figure 4.2 Hybrid Binary INC/DEC design (Lookahead based) [36]

4.2.2 2's Complement and Priority Encoder Circuits

Conventionally the 2's complement of a number is found by complementing each bit and adding 1 to the complemented number. When a normal adder circuit is used for this operation it results in large complexity in hardware and also large delay.

A logarithmic method has been proposed in [34]. Resulting in a ($\log_2 N$) +1 gate delay for implementing the N-bit 2's complement circuit.

A priority encoder is realized by using a prefix tree of AND gates. A similar ($\log_2 N$) +1 delay implementation of an N-bit priority encoder circuit can be found in [5].

4.3 Proposed Multi-functional INC/DEC/2's complement/Priority Encoder Circuit

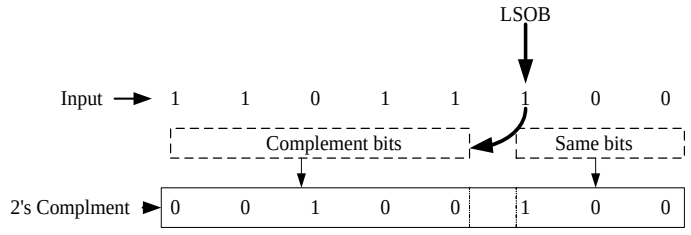
4.3.1 Motivation

The motivation in designing a unified multi-functional circuit, which performs increment, decrement, 2's complement and priority encoding, comes from two main observations explained below [37-38].

Observation 1: Similarity in implementation of 2's complement, priority encoder and decrementer circuits.

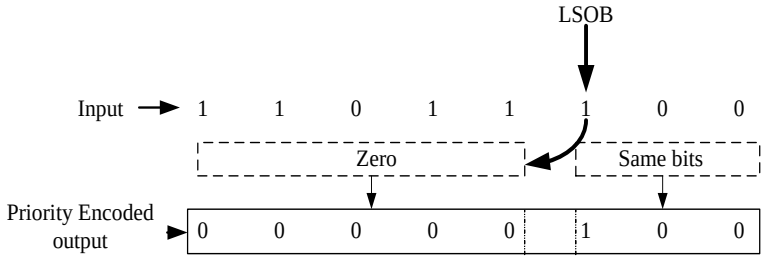
One of the paper and pen methods of finding a 2's complement number is to find least significant one bit (LSOB) and then complementing all the remaining input MSB bits while keeping the bits before the occurrence of LSOB same. This is explained with an example below.

Example 4.1 : To find the 2's complement of 11011100 we keep the bits same until the occurrence of LSOB. After the occurrence of LSOB all the remaining input bits are complemented resulting in 00100100



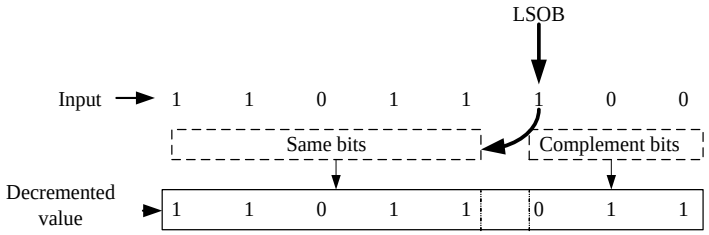
The priority encoder with LSB having highest priority also needs finding of the LSOB. Before the occurrence of LSOB the output bits remain the same as input bits. After the occurrence of LSOB, the remaining output bits are made zero.

Example 4.2 : To find the priority encoded output of 11011100 we keep the output bits same as input until the occurrence of LSOB. After the occurrence of LSOB all the remaining output bits are made zero. The result is thus 00000100.



Similar to 2's complement and priority encoder circuits the decrementer also needs finding of the LSOB. But unlike the other two circuits, the input bits are complemented till the occurrence of LSOB in decrementer. After the occurrence of the LSOB the output bits remain the same as input

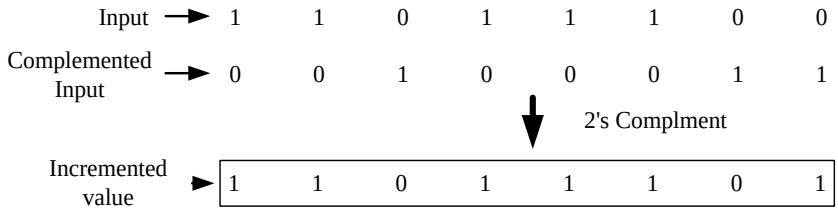
Example 4.3 : To find the decremented output of 11011100 we complement the input bits until the occurrence of LSOB. After the occurrence of LSOB all the remaining output bits remain same as input bits. The result is thus 11011011.



Observation 2: A 2's complement circuit can be used as incrementer by complementing the input.

The 2's complement of any number *A* is given by (*A*' + 1). Complementing the input *A* and taking 2's complement of the number results in (*A* +1), which is the incremented value of input *A.*

Example 4.4 : To find the incremented value of 11011100 we first complement the value resulting in 00100011 and then finding the 2's complement resulting in 11011101



As seen from above observations increment, decrement, 2's complement and priority encoder operations have a common operation of finding the LSOB. This forms the motivation for designing and implementing the multi-functional INC/DEC/2's complement/Priority encoder circuit, which is explained in detail in the following section.

4.4 Implementation

Based on the observations made in the previous section, the proposed multi-functional INC/DEC/2's complement/Priority encoder circuit [37-38] can be designed using the following blocks as shown in Fig. 4.3

1. Input Selection Block
2. Decision Block
3. Output Selection Block

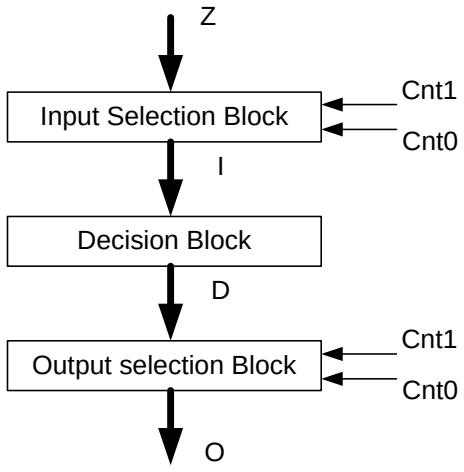


Figure 4.3 Basic Blocks of the Proposed Multi-functional circuit

The control signals 'Cnt1' and 'Cnt0' are used to select different operations as shown in Table 4.1.

Table 4.1 Control signals used to select different operations

Cnt1	Cnt0	Operation performed
0	0	Increment
0	1	Decrement
1	0	2's complement
1	1	Priority Encode

4.4.1 Input Selection Block

This block selects the normal input for decrement, 2's complement and priority encoding operations and complemented input for increment operation. Fig 4.4 shows the implementation of this block.

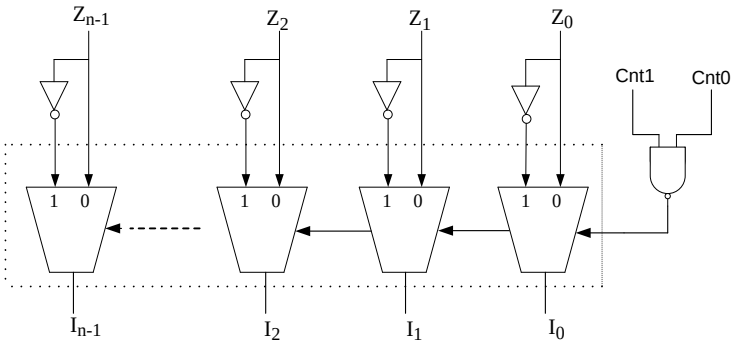


Figure 4.4 .Input Selection Block

In this figure Z represents input to the selection block and I represents output of the selection block. Since input Z is to be complemented only for increment operation i.e. when Cnt1 = 0 and Cnt0 = 0, a NAND gate is used to generate the selection signal for the array of input multiplexers. The outputs ($I_{n-1}I_{n-2}\ldots.I_1I_0$) of the input selection block now act as inputs to the decision block as shown in Fig 4.5.

4.4.2 Decision Block

This block finds the decision signals which have the information of least significant one bit (LSOB). Since increment, decrement, priority encoder and 2's complement operations need finding of LSOB, decision block is common to all the operations.

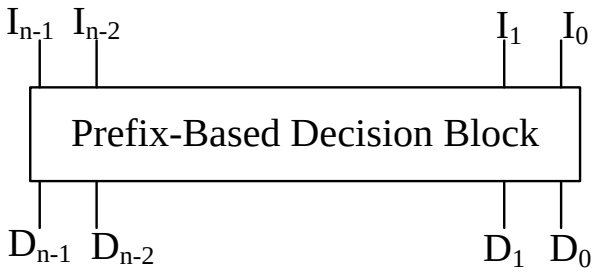


Figure 4.5. Decision Block

The MUX-based binary INC/DEC circuit mentioned earlier [35] in Fig 4.1(b) has a decision block with N-1 (OR) gate delay. An improved decision block is shown in Fig 4.2, which has a delay of (N/8+9) gates (when 8-bit look ahead is used). This delay can be further reduced by using prefix tree structure of OR gates resulting in $\log_2 N$ OR gate delay [5]. Different types of proposed 8-bit decision blocks are shown in the Fig 4.6 and Fig 4.7.

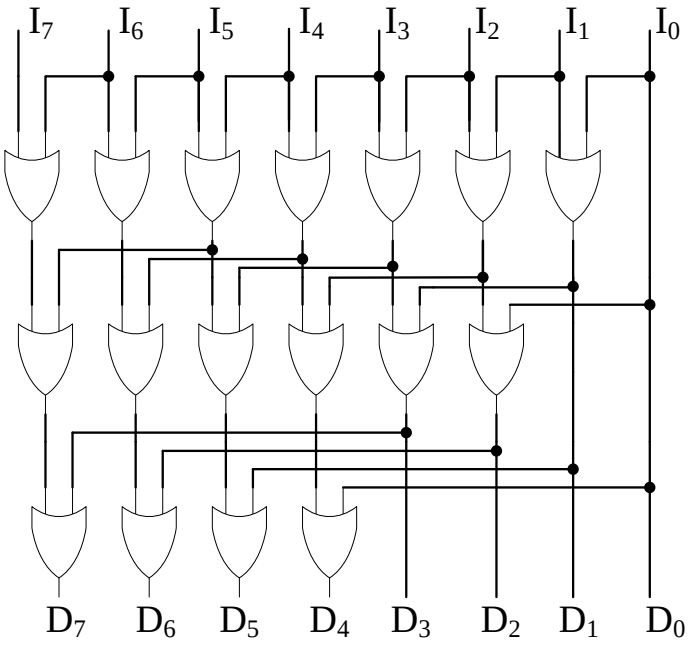


Figure 4.6 Prefix-Based Decision Block Type I

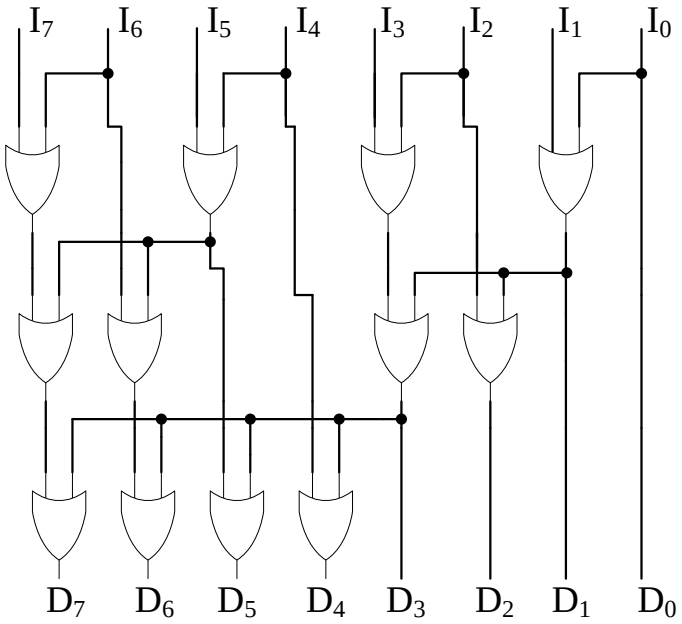


Figure 4.7 Prefix-Based Decision Block Type II

The above implementations of the decision blocks result in reduced delay, when compared with existing designs [35-36]. The delay and complexity of the proposed and existing implementations of the decision block are shown in Table 4.2.

Table 4.2 Delay and Number of Gates Required in Decision Blocks

Decision Block	**Number of Gates Required**	**Delay**
Mux-Based [41]	(N -1)OR	(N-1) t_{OR}
Hybrid [42]	(N-1)OR + N/8(4*NAND+2*NOR+ OR)	(N/8+9) t_{OR}
Proposed prefix-based Type I (Fig 4.6)	(N ($\log_2$ N)- N + 1) OR	($\log_2$ N) t_{OR}
Proposed prefix-based Type II (Fig 4.7)	N/2 ($\log_2$ N) OR	($\log_2$ N) t_{OR}

The structures shown in Fig 4.6 and 4.7 can be modified to result in less number of logic gates and with a delay of (($\log_2$N) +1) gates as shown in Fig 4.8 and 4.9.

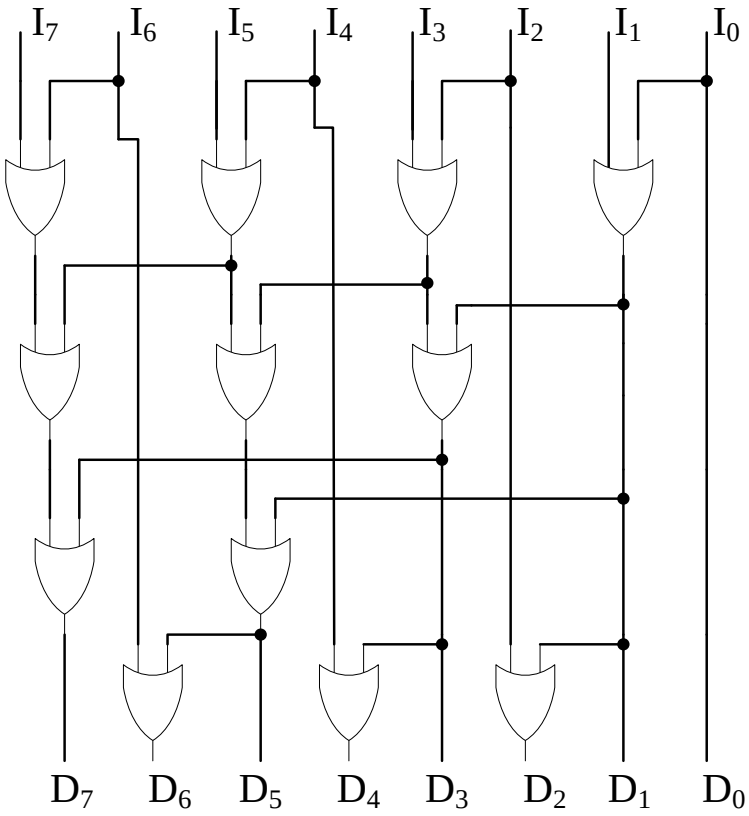


Figure 4.8 Area optimised version of Prefix-based Decision Block Type I

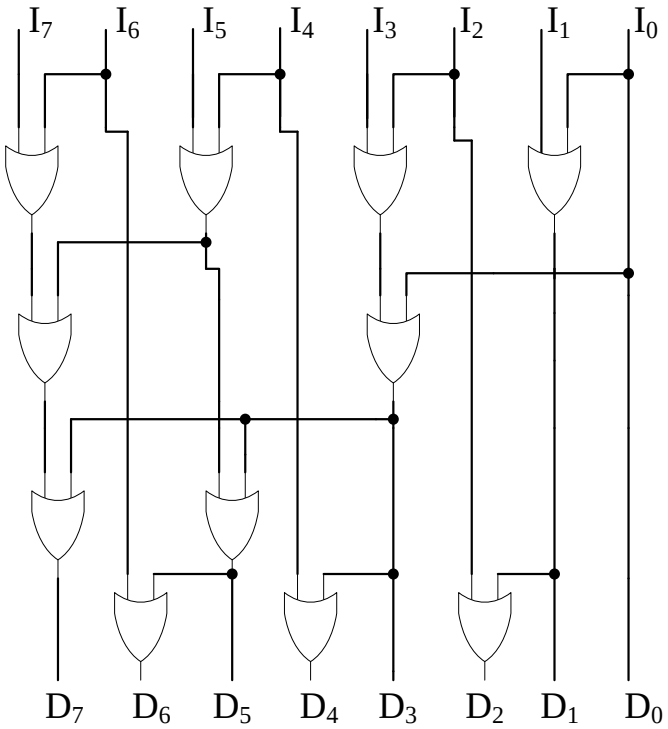


Figure 4.9 Area Optimised Version of Prefix-based Decsion Block Type II

For CMOS implementation, the structures shown in Figs 4.6-4.9 can be further modified by using NOR-NAND blocks in the first two stages. The modified version of the structure in Fig 4.6 is shown in Fig 4.10.

While it takes only 4 transistors to build NAND/NOR gate, 6 transistors are used for OR/AND gate (extra 2 transistors for inverter). Also, OR gate has an extra delay of inverter (1 transistor) compared to NOR/NAND gate. Hence the above implementation results in reduced area, power and delay. The area of the structure in Fig 4.10 is reduced by 12 inverters and the critical path delay is reduced by 2T (delay of 2 inverters), when compared to the original design in Fig 4.6. The structures in Figs 4.7-4.9 can also be designed using a NOR-NAND block in the first two stages.

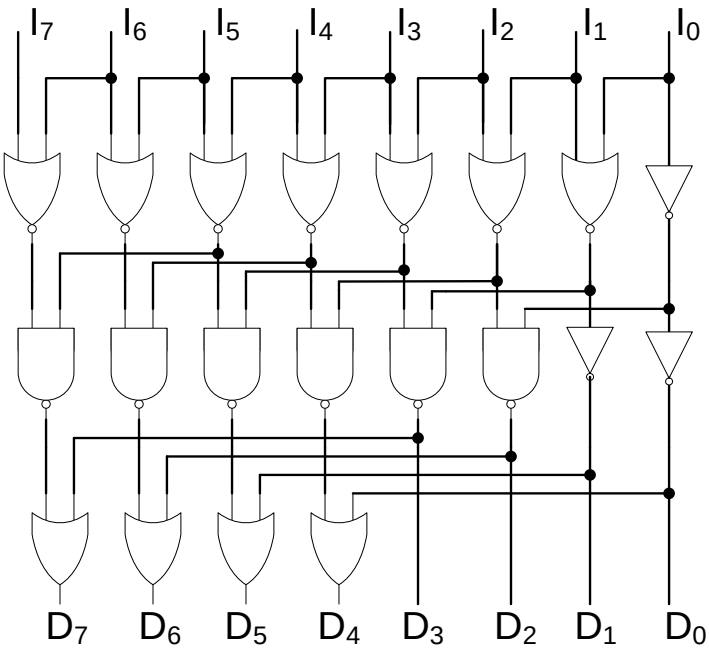


Figure 4.10 Prefix based decision block Type I with NOR-NAND

4.4.3 Output Selection Block

Output Selection Block is used to find the output, based on control signals Cnt1, Cnt0 and inputs ($Z_{n-1}…Z_1Z_0$). The Boolean equations for different operations can be derived from the observations made earlier. The outputs (O) for different operations based on decision signal (D) are given below.

Increment Operation: Cnt1 = 0, Cnt0 = 0

If $D_{n-1} = 0$, then output $O_n = Z_n$ (Z_n is n^{th} input signal)

If D_{n-1} = 1, then output $O_n = Z_n$' (Z_n' is complement of Z_n)

Decrement Operation: Cnt1 = 0, Cnt0 = 1

If D_{n-1} = 0, then output $O_n = Z_n$'

If D_{n-1} = 1, then output $O_n = Z_n$

2's complement Operation: Cnt1= 1, Cnt0 = 0

If D_{n-1} = 0, then output $O_n = Z_n$

If D_{n-1} = 1, then output $O_n = Z_n$'

Priority Encode Operation: Cnt1 = 1, Cnt0 = 1

If D_{n-1} = 0, then output $O_n = Z_n$

If D_{n-1} = 1, then output O_n= 0.

The implementation of output selection block based on the above equations is shown in Fig 4.11.

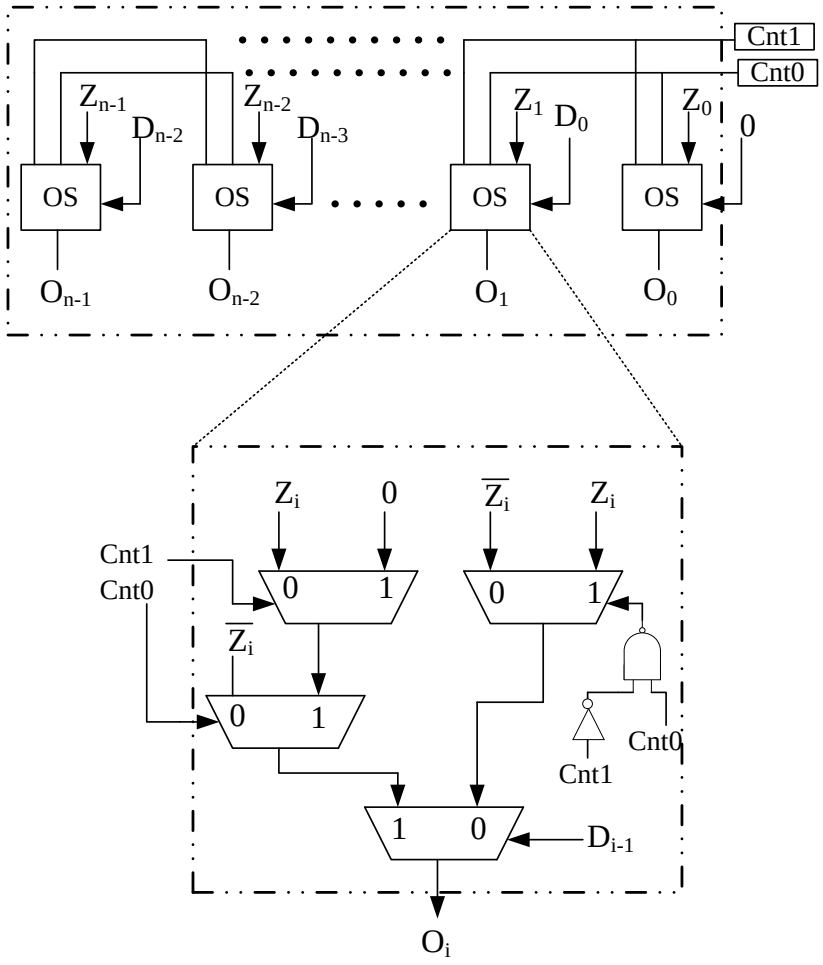


Figure 4.11 Output Selection Block

The output selection block is a multi-functional block which can be configured, using the control signals Cnt1 and Cnt0, to operate as a decrementer, incrementer, 2's complement or priority encoding circuit [37-38].

4.5 Simulation Results

4.5.1 Multi-functional INC/DEC/2's Complement/Priority Encoder:

All the designs have been structurally described using Verilog HDL and simulated using Cadence Incisive Unified Simulator (IUS) v6.1 covering all functional combinations. These adders were mapped on the TSMC 180*nm* Technology Typical library (operating conditions 1.8 V, 25ºC), using Cadence RTL Compiler v7.1. Inputs were set to have a toggle rate of 50% and a frequency of 1GHz for calculating dynamic power.

Initially, a comparison is carried out for INC/DEC circuits with proposed [37-38] and existing 32-bit decision blocks [35-36]. The prefix based Type I structures shown in Fig 4.6, Fig 4.8 and Fig 4.10 are chosen to implement the decision block of the proposed INC/DEC circuits and are compared with existing INC/DEC circuits shown in Fig 4.1 and Fig 4.2. Table 4.3 shows the comparison results.

Table 4.3 Simulation results for 32-bit INC/DEC Circuits

INC/DEC	**Delay (nS)**	**Power (mW)**	**Power-Delay Product (pJ)**	**Area (um^2)**
Mux-based [35] (Fig 4.1(b))	5.640 (100%)	0.336 (100%)	1.895 (100%)	854 (100%)
Hybrid [36] (Fig 4.2)	2.386 (42.30%)	0.418 (124.40%)	0.997 (52.61%)	933 (109.25%)
With Proposed Decision Block Type I (Fig 4.6)	1.436 (25.46%)	0.414 (123.21%)	0.594 (31.35%)	1200 (140.51%)
With Proposed Area Optimized Decision Block Type I (Fig 4.8)	1.589 (28.17%)	0.375 (111.61%)	0.596 (31.45%)	1027 (120.26%)
With Proposed Delay Optimized Decision Block Type I (Fig 4.10)	1.221 (21.65%)	0.407 (121.13%)	0.497 (26.23%)	1137 (133.14%)

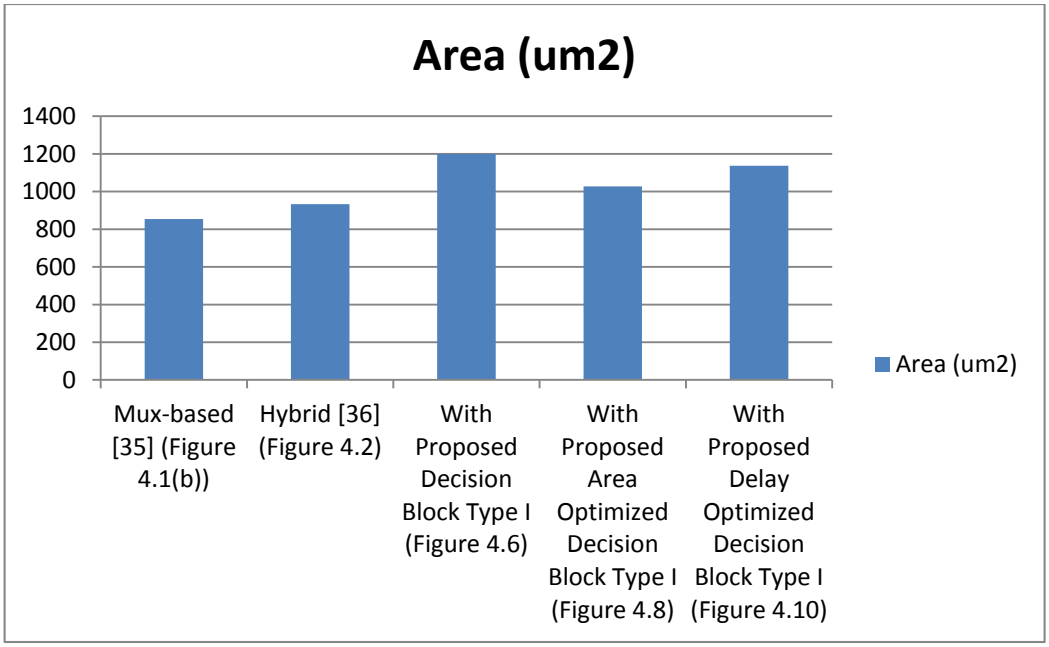


Figure 4.12 (a)

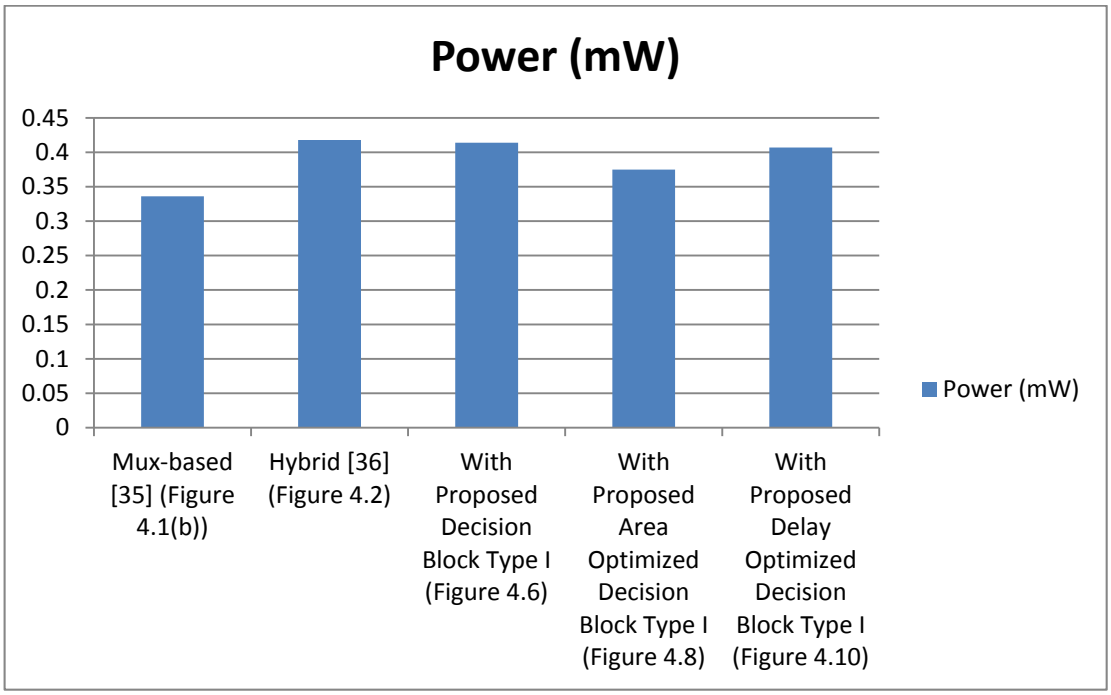


Figure 4.12 (b)

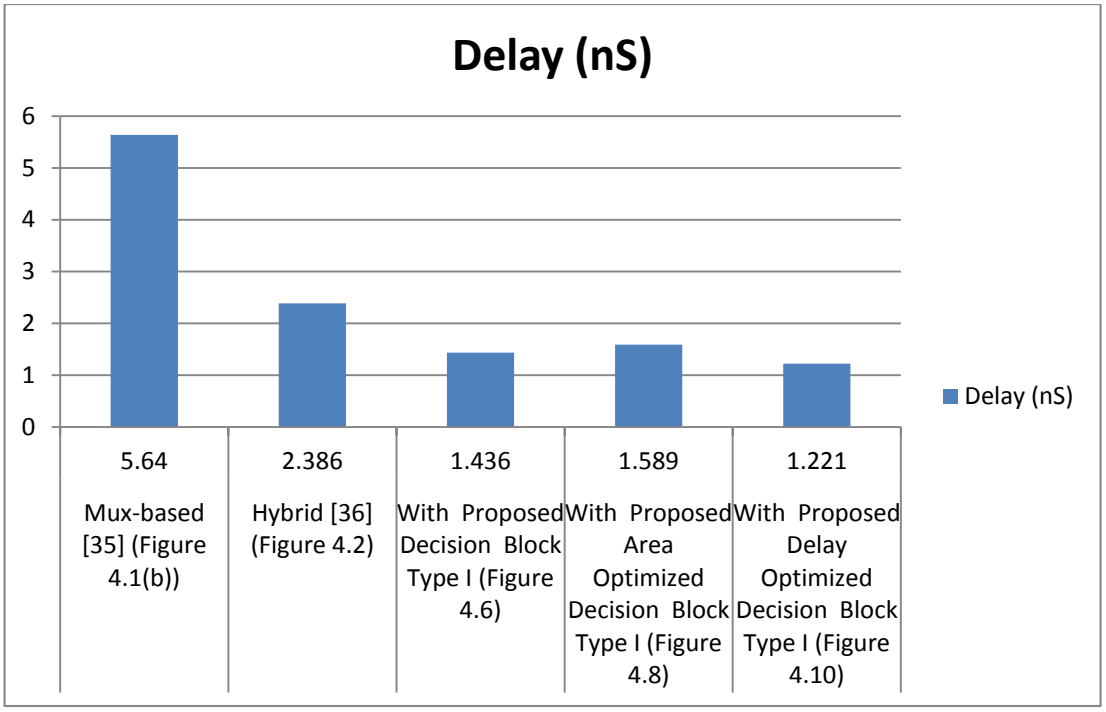


Figure 4.12 (c)

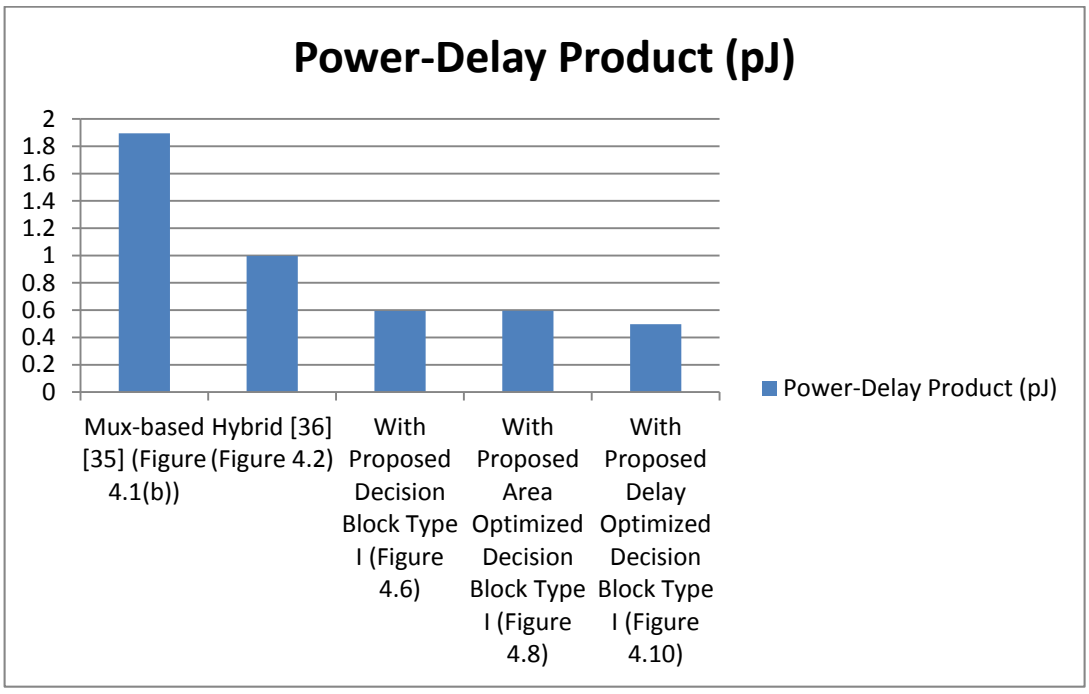


Figure 4.12 (d)

Figure 4.12 Comparisions of proposed designs with existing designs in terms of (a) Area (b)Power (c) Delay (d) Power-Delay Product

It is clear from the Table 4.3 and Fig 4.12 that the INC/DEC circuits with proposed decision blocks result in 33-48% reduction in delay and 30-50% reduction in power delay product, depending on the decision block used, when compared with the existing designs [36].

It can also be seen from the Table and the figure that INC/DEC circuit with delay optimized decision block of Type I results in 14% less delay when compared to a similar one with a simple prefix based decision block. Also, it has a 14% improvement in area.

Table 4.4. below details the results obtained for the multi-functional circuit that can be configured to perform increment, decrement, 2's complement or priority encode operations. This circuit has been implemented with both the existing and proposed decision blocks.

Table 4.4 Simulation results For 32-bit Multi-functional INC/DEC/2's complement/Priority Encoder Circuit

Multi-functional INC/DEC/2's complement/Priority Enoder Circuit	**Delay (nS)**	**Power (mW)**	**Power-Delay Product (pJ)**	**Area (um^2)**
With Decision Block of Mux-based [35]	6.745 (100%)	0.556 (100%)	3.75 (100%)	1354 (100%)
With Decision Block of Hybrid [36]	3.497 (51.85%)	0.619 (111.33%)	2.16 (57.60%)	1432 (105.76%)
With Proposed Decision Block Type I (Fig 4.6)	2.538 (37.63%)	0.630 (113.31%)	1.598 (42.51%)	1700 (125.55%)
With Proposed Area Optimized Decision Block Type I (Fig 4.8)	2.692 (39.91%)	0.593 (106.66%)	1.596 (42.56%)	1527 (112.78%)
With Proposed Delay Optimized Decision Block Type I (Fig 4.10)	2.333 (34.89%)	0.624 (112.23%)	1.455 (38.80%)	1637 (120.90%)

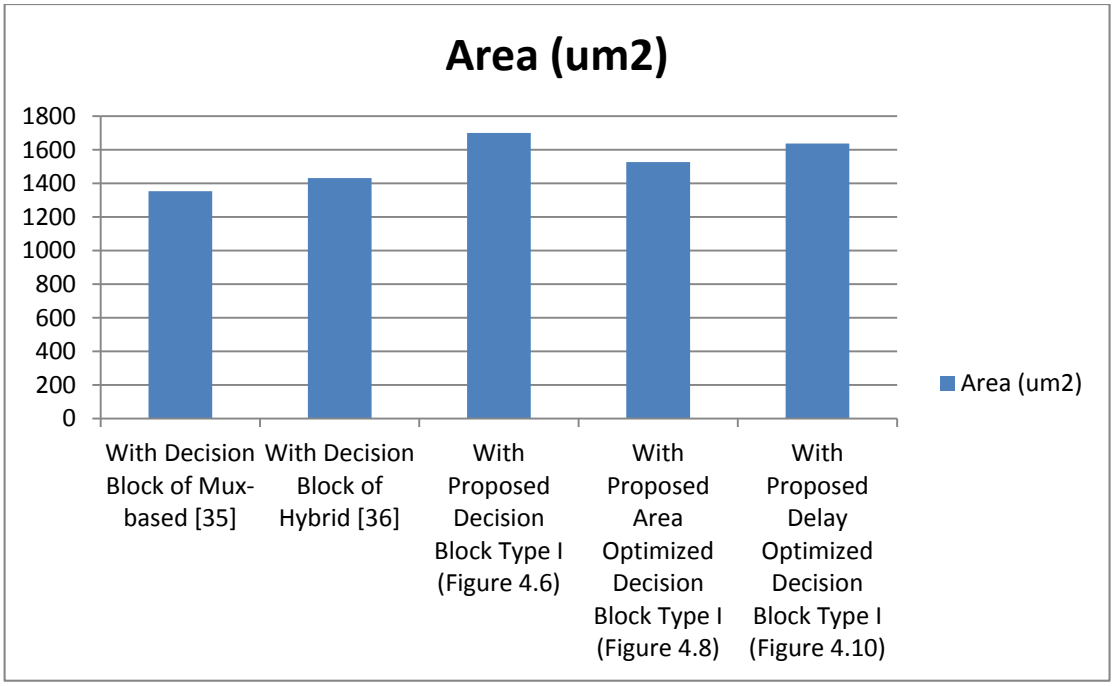


Figure 4.13 (a)

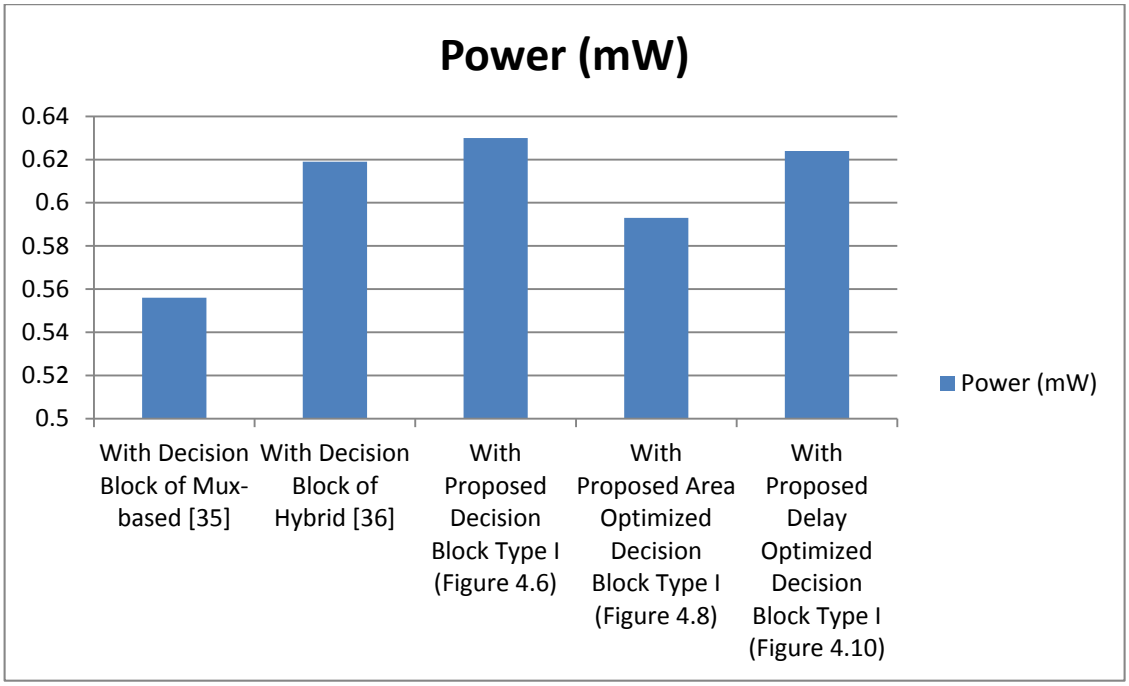


Figure 4.13 (b)

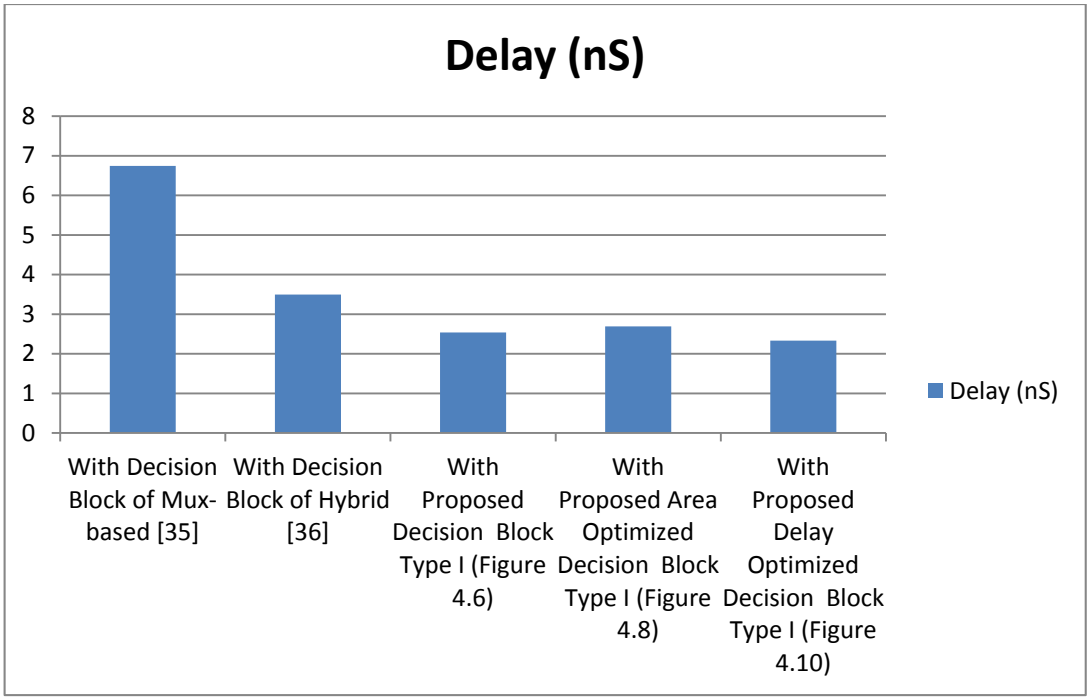


Figure 4.13 (c)

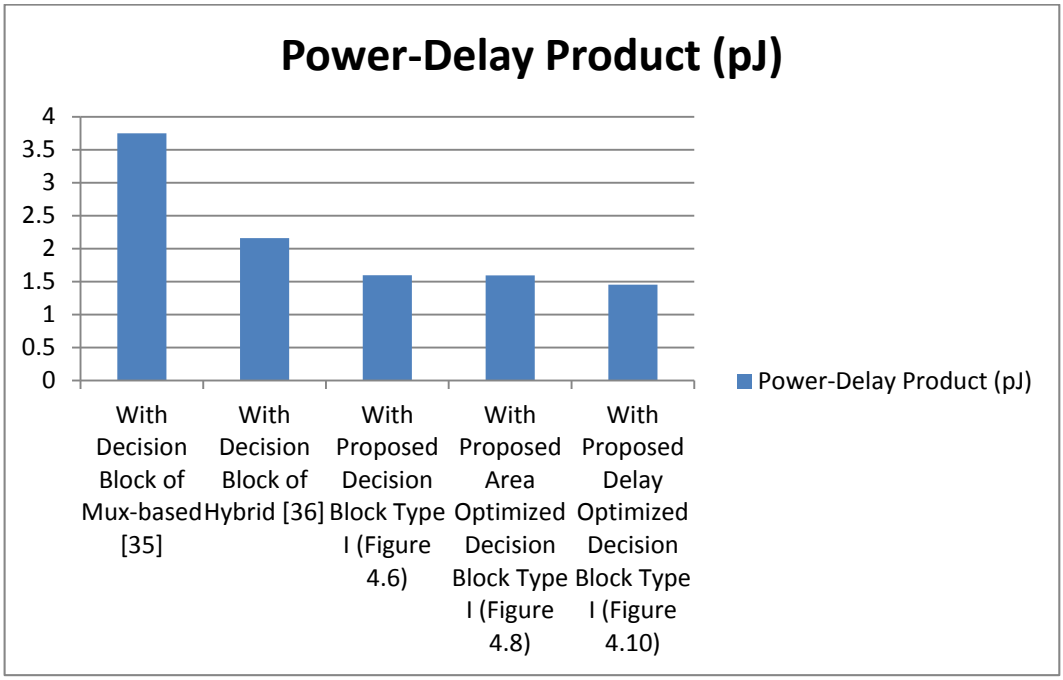


Figure 4.13 (d)

Figure 4.13 Comparisions of multi-functional circuit with proposed decision blocks and existing decision block in terms of (a) Area (b)Power (c) Delay (d) Power-Delay Product

It is clear from Table 4.4 and Fig 4.13 that the proposed multi-functional circuit with results in a 23-33% reduction in delay and 26-32% reduction in power delay product, depending on different prefix based decision blocks used, when compared with the similar circuit with the existing decision blocks [36].

4.6 Conclusion

In this chapter, some function specific arithmetic blocks such as incremeter/decrementer, priority encoder, etc.., that are frequently used in processors have been designed and analyzed. Further, a novel multi-functional circuit which can be configured to perform increment, decrement, 2's complement or priority encoder operations, has been proposed and implemented. The multi-functional circuit implementation using the novel decision blocks results in a reduction of up to 33% in delay and 32 % in power delay product when compared with the existing implementations. The decision block proposed in this work can also be used for the design of stand-alone INC/DEC block leading to a reduction of 48% in delay and 50% in power delay product.

Chapter 5
Design and Implementation of a Unified BCD/Binary Adder/Subtractor

Contents

5.1 Introduction

There is a growing importance of decimal arithmetic in commercial, financial and internet-based applications. These applications cannot tolerate errors that result from the conversion of binary format to decimal format. Thus, hardware support for decimal arithmetic is receiving considerable attention. Recently, specifications for decimal floating point arithmetic have been added to the draft revision of IEEE-754 standard for floating point arithmetic [43]. Despite the widespread use of binary arithmetic, decimal computation remains essential for many applications. Not only is it required whenever numbers are presented for human inspection, but is also often a necessity when fractions are involved. Decimal fractions are pervasive in human endeavors, yet most cannot be represented by binary fractions. The value 0.1 for example,

requires an infinitely recurring binary number. If a binary approximation is used instead of an exact decimal fraction, results can be incorrect even if subsequent arithmetic is correct.

As the IEEE standard for decimal floating point is approved, hardware support for decimal floating point arithmetic will be incorporated in processors for various applications. Still, a major consideration while implementing Binary Coded Decimal (BCD) arithmetic is to enhance its speed as much as possible.

BCD is a decimal representation of a number directly coded in binary, digit by digit. For example, the number $(9527)_{10}$ is represented as $(1001\ 0101\ 0010\ 0111)_{BCD}$. It can be seen that each digit of the decimal number is coded in binary and then concatenated to form BCD representation of the decimal number.

To use this representation all the arithmetic and logical operations need to be defined. As the decimal number system contains 10 digits, at least 4 bits are needed to represent a BCD digit. The BCD representation of digit A is $A_4A_3A_2A_1$ where all $A_k \in (0,1)$. The only point to note is that the maximum value that can be represented by a BCD digit is '9'. The representation of $(10)_{10}$ in BCD is (0001 0000).

Addition in BCD can be explained by considering two decimal digits A and B with BCD representations as $A_4A_3A_2A_1$ and $B_4B_3B_2B_1$ respectively. In the conventional approach , these two numbers are added using a 4-bit binary adder during which it is possible that the resultant Sum can exceed 9 resulting in an overflow. If the Sum is greater than 9, the binary equivalent of 6 is added to the resultant Sum to obtain the exact BCD representation. This can be illustrated with the following example

$$
\begin{array}{ll}
A & 0110 \quad (6) \\
B & \underline{0101} \quad (5) \\
Sum & 1011 \quad (11) \\
Add & \underline{0110} \quad (6) \\
BCD & 1\,0001 \quad (11\ in\ BCD) \\
Answer = (0001 & 0001)
\end{array}
$$

5.2 Review of Existing Techniques for BCD Addition/Subtraction

5.2.1 One-Digit BCD Full Adder

A BCD 1-digit adder is a circuit that adds two BCD digits in parallel and also produces the Sum digit in BCD along with the necessary correction logic. The conventional implementation of addition as mentioned above is shown in Fig 5.1[44]. It can be seen that a 4-bit binary adder is used initially to add two BCD digits (each digit expressed using 4 bits) with a carry-input. An overflow detection circuit is used (to check if the 'Sum' of the BCD digit has exceeded 9) which is designed using two 2-input AND gates and a 3-input OR gate. Finally, another 4-bit binary adder is used as a correction stage, which comes in the path of final Sum computation. Thus, the critical path in this circuit consists of a 4-bit binary adder, overflow logic and one more 4-bit binary adder. Assuming, in the best case, that the 4-bit binary adder is a carry look-ahead adder, a gate level analysis would indicate that it consists of 4-gates in the critical path. It can be observed from Fig 5.1 that the overflow detection circuit comes into picture only after the top-most 4-bit binary adder performs its operation and it consists of 2 gates in the critical path. Thus, a minimum of a 10-gate delay can be expected in conventional implementation [44]. The above design can however be optimized by removing those gates that are completely redundant in their operation. Such a modification is shown in Fig 5.2 which results in a smaller critical path. A faster carry prediction for this implementation is proposed in [45], which uses carry look-ahead logic to predict the carry in advance. These 1-digit full adders can be cascaded to realize higher digit BCD adders.

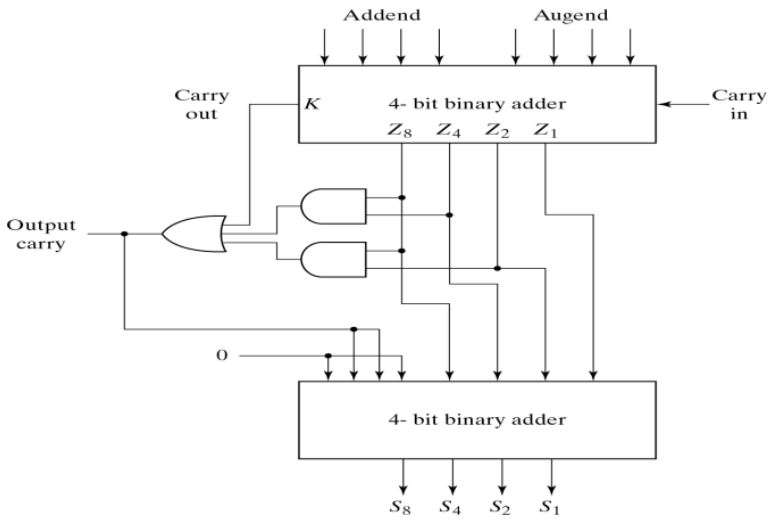


Figure 5.1 Block Diagram of Conventional 1-digit BCD Full Adder

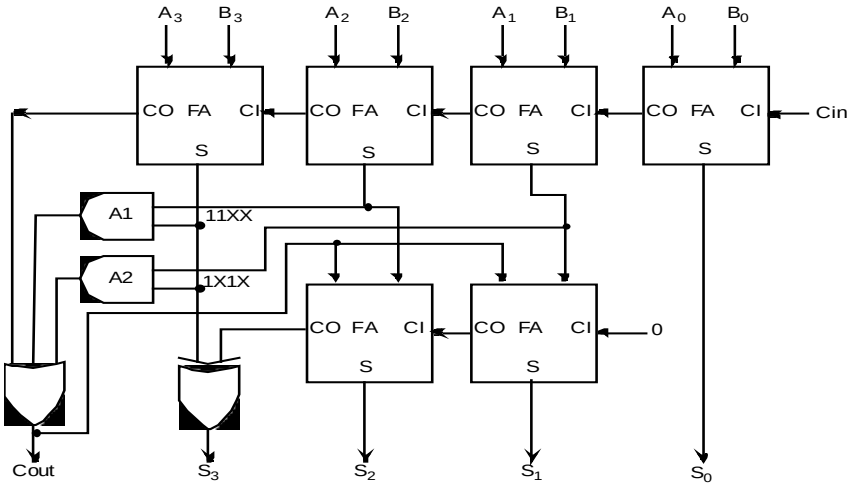


Figure 5.2 Block Diagram of Modified Conventional 1-digit BCD FA

5.2.2 Higher Bit BCD/Binary Adders/Subtractors

A unified BCD/Binary module is the one, which can perform both BCD as well as binary operation. There have been many contributions on decimal arithmetic especially on adders/subtractors [43-56]. Some of the initial contributions came from Schoomklar et al. [46] and Adiletta et al. [47]. The first BCD sign-magnitude adder/subtractor was designed by Grupe [48]. An area efficient sign-magnitude adder was later developed by Hwang [49]. In this approach two additional conversions were introduced before and after the binary addition.

A BCD adder similar to the carry select adder was presented in [50]. This design concurrently calculates two results, one assuming the presence of the input carry and the other its absence. It then selects the appropriate result as the carry is computed. Fischer et al. [51] later came up with an improved version of this design shown in Fig 5.3 where only a single adder was used to reduce the area overhead.

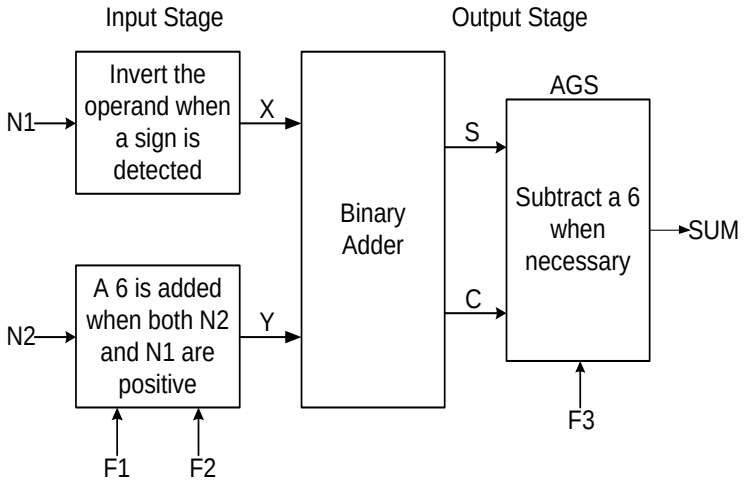


Figure 5.3 Fischer's Architecture [51]

BCD adder / subtractor architectures in many IBM processors are based on the work presented by Haller et al. in [52]. A generic architecture shown in Fig 5.4 operates in a single cycle, though requiring corrections in some cases. In case of subtraction, there is a need for computation of the complement to obtain correct result, thus increasing the latency.

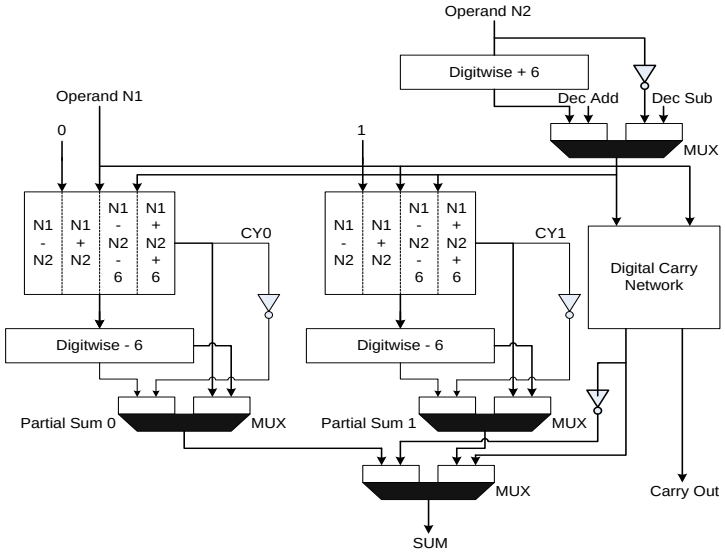


Figure 5.4 Haller's Architecture [52]

Humberto et al. [53] proposed a universal adder design shown in Fig 5.5 that uses effective addition / subtraction operations on unsigned/sign-magnitude and various complement representations. This design overcomes the limitations of previously reported approaches that produce some of the results in complement representation when operating on sign-magnitude numbers.

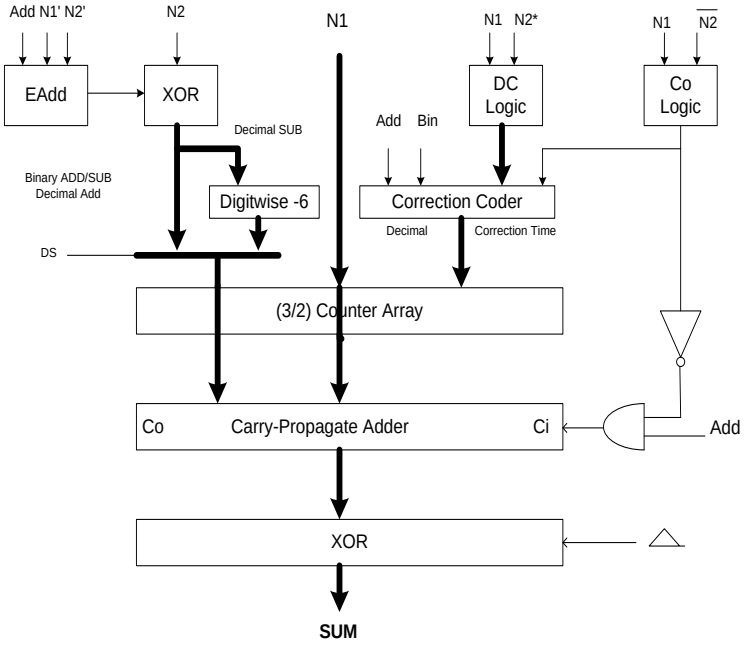


Figure 5.5 Humberto's Architecture [53]

In the existing architectures like Fischer's [51] and Haller's [52] , if a smaller operand is subtracted from larger operand, extra hardware for 2's complement or 10's complement is required to get the final unsigned number. This adds not only an area overhead but also affects the delay. In Humberto [53], a comparator is used in the pre-computational block to check which operand is smaller and necessary correction is incorporated in the pre-computation block thereby avoiding usage of extra complementary stage. However, in this architecture also usage of comparator creates hardware overhead and gives rise to delay in the critical path. In this thesis, a novel architecture is proposed that can perform BCD and binary addition / subtraction on both unsigned/signed numbers without any need of a comparator stage as well as 2's/10's complementary stage.

5.3 A Unified BCD/Binary Adder/Subtractor Architecture

In this architecture, the output carry signal is analyzed to determine which of the operands is greater unlike Humberto architecture that compares two numbers at the input stage itself. This approach eliminates the need for a comparator at the input stage which is not possible with Humberto architecture. Further, end-around carry technique is used to correct the 2's/10's complementary cases due to the which usage of 2's/10's complementary stage at the output can be avoided. Thus, the proposed architecture can be said to have advantage in terms of area and delay when compared to the existing architectures like Humberto's. In the following sub-section,

a detailed implementation of the proposed unified binary and BCD adder/subtractor is discussed. Initially, the conventional binary adder/subtractor is discussed followed by a modified binary adder/subtractor. This is followed by the existing design of BCD adder/subtractor leading to an improved version of the same. Finally, the binary and BCD adder/subtractor has been combined to realize a unified binary/BCD adder /subtractor that performs better than the exiting one.

5.3.1 Conventional Binary Adder /Subtractor

The subtraction operation between two operands say A and B is given as follows:

$$S = A + B' + 1.$$

Where B' represents the complement of B.

Thus, for implementing binary subtraction, one of the operands is inverted and given to an adder circuit with an input carry as '1'. In binary subtraction, two cases can arise, i.e., $A > B$ and $A \leq B$. When $A > B$, the result 'S' is in unsigned/signed magnitude form. When $A \leq B$, the result 'S' is in 2's complement form. In this case, 'S' needs to be corrected using 2's complementary stage to get correct result in signed magnitude form.

The conventional implementation of binary subtractor along with 2's complement correction is shown in Fig. 5.6. The final carry-out signal from the adder indicates whether $A > B$ or $A \leq B$. For example, when '4' is subtracted from '5' the carry-out will be '0'. If '5' is subtracted from '4', the carry-out of the binary adder is '1'. Thus, from final carry-out signal, the requirement of 2's complement correction can be decided.

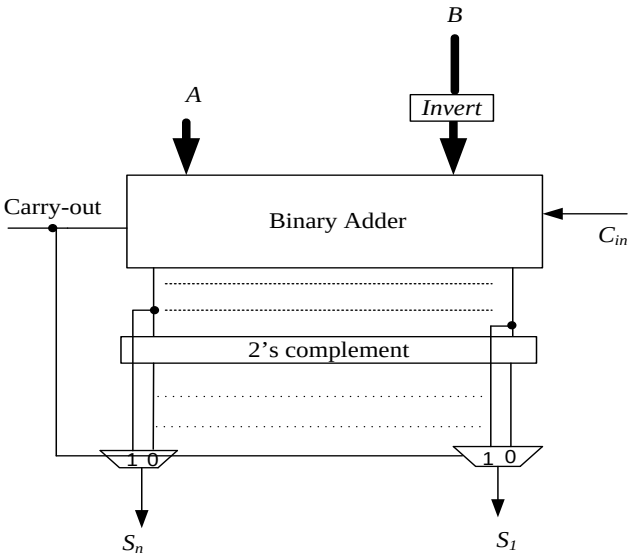


Figure 5.6 Conventional implementation of binary subtractor

In floating-point operations, where the operands are in signed magnitude form, there is an extra bit that indicates the sign of the operand. Addition/subtraction operation on this signed magnitude form not only depends on type of operation but also on the sign bit. For example, assume X and Y are two (n+1)-bit signed magnitude numbers such that $X = [X_nX_{n-1}X_{n-2}\ldots X_0]$ and $Y = [Y_nY_{n-1}\ Y_{n-2}\ldots\ldots Y_0]$, where X_n and Y_n are sign bits. The type of operation i.e., addition or subtraction is represented with 'Op'. (Where 'Op' is logic '1' the operation is subtraction and vice versa). The effective operation that depends on the type of operation as well as sign bits is given in Table 5.1.

Table 5.1 Effective Operation on signed magnitude numbers

X_n	Y_n	Op	Effective Operation (EOp)
0	0	0	Addition
0	0	1	Subtraction
0	1	0	Subtraction
0	1	1	Addition
1	0	0	Subtraction
1	0	1	Addition
1	1	0	Addition
1	1	1	Subtraction

From this Table, the 'Effective Operation' (*EOp)* is given by equation (5.1). When the 'EOp' is logic '1', the operation that needs to be performed is addition and when 'EOp' is logic '0' the operation is subtraction.

$$EOp = (X_n \odot Y_n) \oplus Op \qquad (5.1)$$

After the effective operation EOp is determined using equation 5.1, sign of the result is computed using the sign of the first operand X i.e. X_n and the Carry-out from the adder circuit. If the final effective operation is addition then the sign of the final result is equal to the sign of the first operand i.e. X. However, if the effective operation is subtraction the final sign depends on the sign of X and also the carry-out signal (indicates if X >Y or X≤ Y) of the adder circuit. The sign of the final result 'S_n' is given by

Final Sign $S_n = X_n$ if EOp = '1' i.e. addition

$S_n = X_n \oplus (C_{out})$' if EOp = '0' i.e. subtraction.

The sign of final result from the above equation can be obtained using the implementation shown in the Fig 5.7

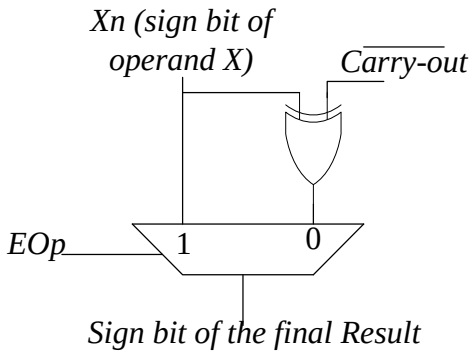


Figure 5.7 Final Sign 'S_n' computation logic

The design of binary adder/subtractor with conventional subtractor explained earlier and which supports the signed magnitude form is shown in Fig 5.8. From the figure, it can be observed that the extra 2's complementary stage increases delay and area when compared to the same used for addition operation.

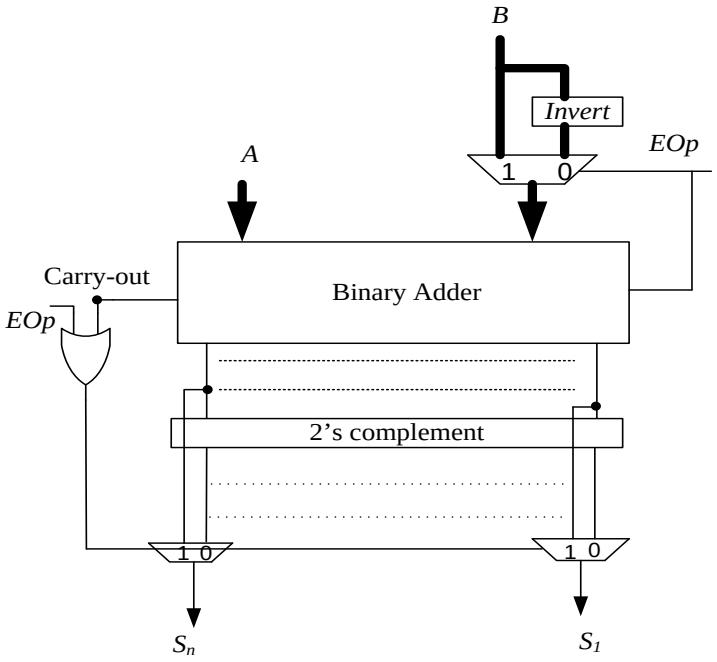


Figure 5.8 Conventional implementation of binary adder/subtractor with signed magnitude

5.3.2 A Modified Binary Adder/Subtractor

In this section, a binary adder/subtractor is proposed which uses end-around carry method to eliminate the complementary correction stage. For using end-around carry method, the adder

is implemented using the prefix network. In the proposed design, the subtraction operation (that depends on X > Y and X ≤ Y) is implemented by using the following equations.

If $X > Y$ (EOp = 0 and Carry-out =1) then result = $X + (Y)' + 1$ (5.2)

If $X \leq Y$ (EOp = 0 and Carry-out = 0) then result = $(X + (Y)' + 1)' + 1 = (X + (Y)')'$ (5.3)

When the effective operation is addition i.e., $EOp = 1$, both the operands X *and* Y are given directly to the prefix adder and the final result is $X+Y$. When the effective operation is subtraction i.e. $EOp = 0$, operand Y is inverted at the input side. The normal addition operation is carried out to result in $X + (Y)'$. The resulting 'Carry-out' of this addition indicates whether $X > Y$ *or* $X \leq Y$. Based on this and from the above equations, addition of '1' or inverting operation is decided to compute the final result. The optimized late carry-in adder proposed in section 2.3 is used in this design. This late carry-in is used for the addition of '1' when $X>Y$. When $X \leq Y$, *a* group of XOR gates carries out the inverting operation after the sum is computed. The proposed binary adder/subtractor design is shown in Fig 5.9.

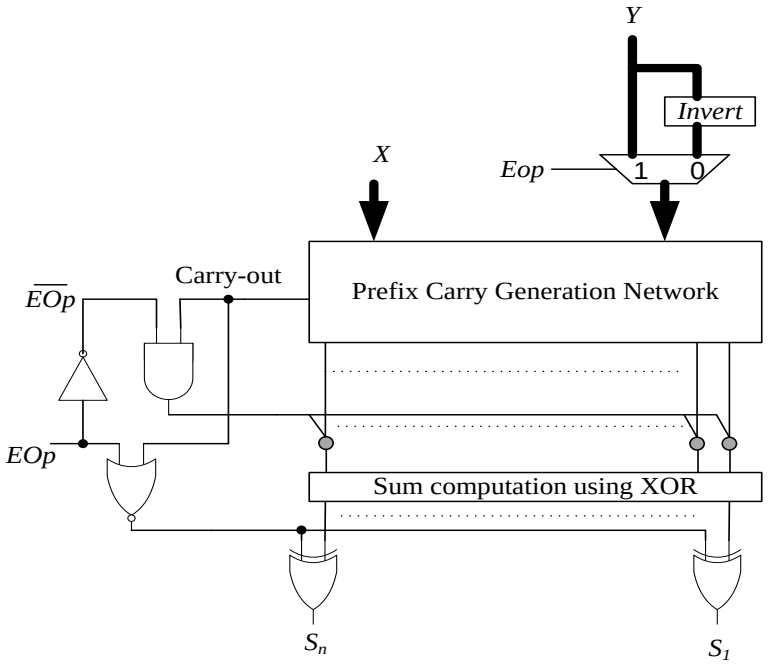


Figure 5.9 Implementation of the Proposed Binary Adder/subtractor Design

5.3.3 Modified BCD Adder/Subtractor

The existing BCD Adder/Subtractor architectures and the limitations of the same have been explained in section 5.2. In this section, a modified BCD adder/subtractor architecture is presented which overcomes these limitations. The proposed design is inspired from Fischer approach but eliminates the usage of complementary stage as well as supports unsigned and signed magnitude form.

In Fischer's approach, the BCD addition operation is performed by pre-correction block where digit-wise addition of '6' is carried out for one of the operands. After pre-correction the result is added to other operand by a binary adder. The post-correction block includes conditional subtraction of '6' depending up on the 'Carry-out' at each digit stage. This signal at each digit stage indicates whether the digit is greater than '9' or not. For example, if the 'Carry-out' is '1' the digit is less than or equal to '9' and hence no correction is required. If the 'Carry-out' is '0' the digit is greater than '9' and a correction by subtraction of '6' is needed. Since 2's complement of '6' is '10', subtraction of '6' i.e. $(0110)_2$ is accomplished by addition of '10' i.e., $(1010)_2$. The following example 5.1 and Fig 5.10 illustrate the above decimal addition operation.

Example 5.1:

Let X = 5 5 6

Y = 2 3 9

In BCD format: X = 0101 0101 0110

Y = 0010 0011 1001

Addition of digit-wise 6 i.e. $(0110)_2$ to X results in new X,

X = 0101 0101 0110

+6 0110 0110 0110

Hence, new X = 1011 1011 1100

Now the 'new X' is added to Y and correction is applied.

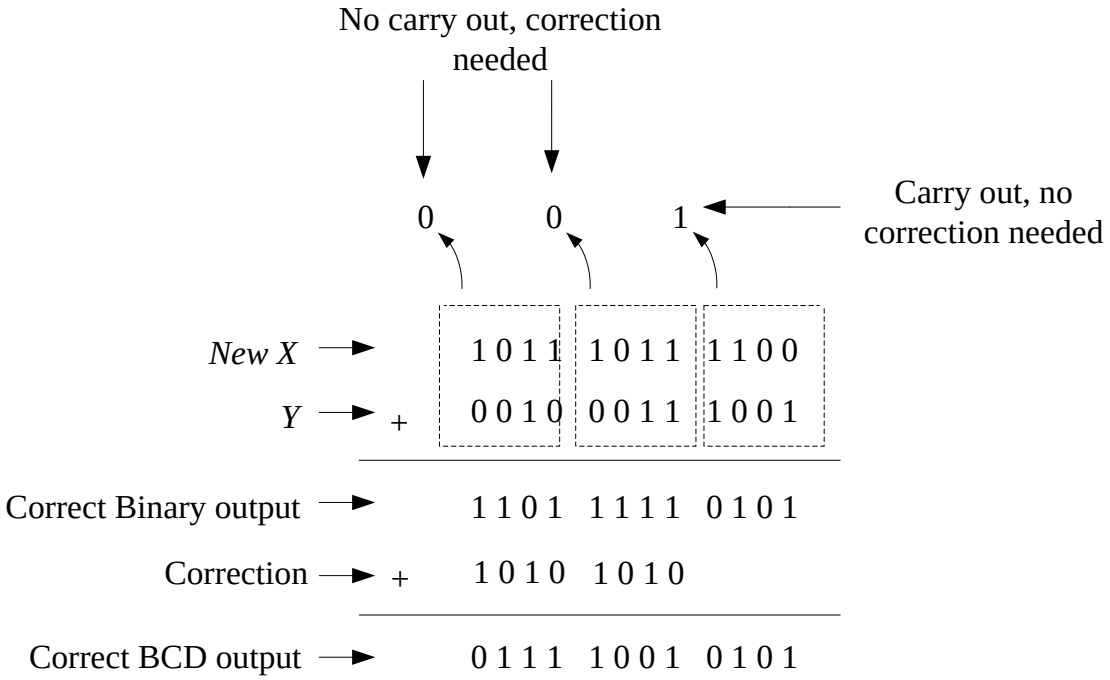


Figure 5.10 Illustration of BCD addition operation

The BCD subtraction is similar to binary subtraction with an extra post-correction stage like in BCD addition. In Fischer's approach, the BCD subtractor gives unsigned result when $X>Y$ where X and Y are minuend and subtrahend respectively. However, this approach requires a 10's complement, like 2's complement for binary, if $X \leq Y$. The design proposed in this work is aimed to eliminate the overhead related to this extra complementary stage. However, the post-correction stage needs to be modified to handle both conditions $X>Y$ and $X \leq Y$. The following examples 5.2 and 5.3, Fig 5.11 and Fig 5.12 illustrate the decimal subtraction operation for cases $X>Y$ and $X \leq Y$ respectively.

Example 5.2: Subtraction operation and X>Y

Let $X = 5\ 5\ 6$
$Y = 2\ 3\ 9$
In BCD format: $X = 0101\ 0101\ 0110$
$Y = 0010\ 0011\ 1001$

As explained in binary adder/subtractor, if X > Y, the result = X+Y'+1. Thus, taking 1's complement of Y (as in normal binary subtraction) results in 'New Y'. This 'New Y' is added to X. If the 'carry-out' signal is '1' that indicates X>Y, addition of '1' is carried out. Then post-correction is applied on this result to compute final BCD difference. The digit wise carry of '1'

indicates that the digit is less than or equal to '9' and hence no correction is required while the digit wise carry of '0' indicates the digit is greater than '9' and a correction by subtraction of '6' or addition of '10' is needed [57].

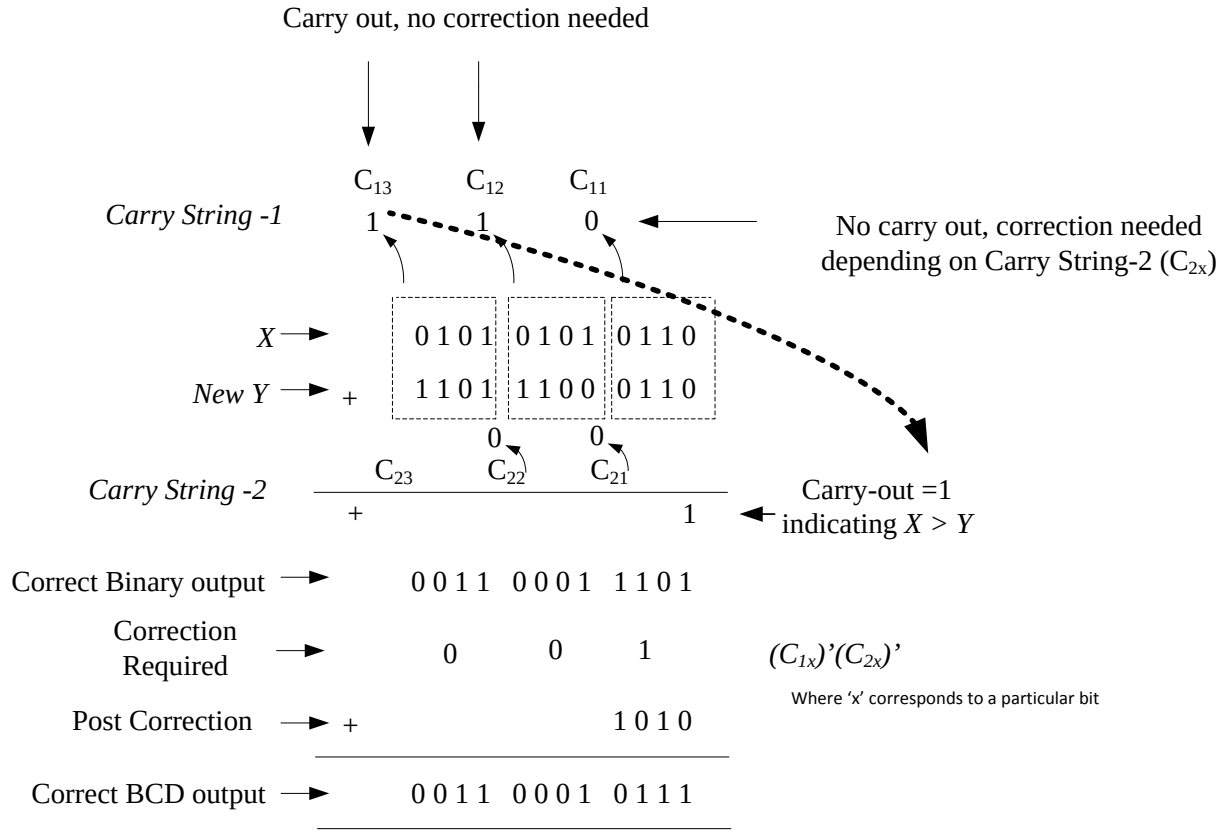


Figure 5.11 Illustration of BCD Subtraction operation when $X > Y$

Example 5.3: Subtraction operation and $X \leq Y$

Let $X = 2\ 3\ 9$

$Y = 5\ 5\ 6$

In BCD format: $X = 0010\ 0011\ 1001$

$Y = 0101\ 0101\ 0110$

As explained in binary adder/subtractor, if $X \leq Y$, the result = (X+Y')'. Thus, taking 1's complement of Y (as in normal binary subtraction) results in 'New Y'. This 'New Y' is added to X. If the 'carry-out' signal is '0' that indicates $X \leq Y$, the result is complemented. Then post-correction is applied on this result to compute final BCD difference. The digit carry of '0' indicates the digit is less than or equal to '9' and hence no correction is required while the digit

carry of ‘1’ indicates the digit is greater than ‘9’ and a correction by subtraction of ‘6’ or addition of ‘10’ is needed [57].

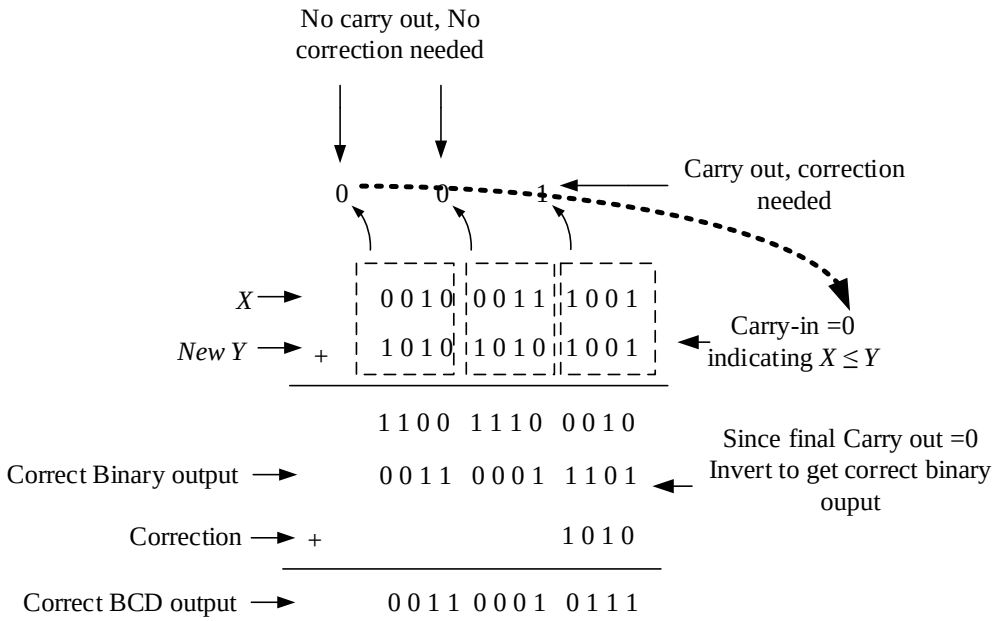
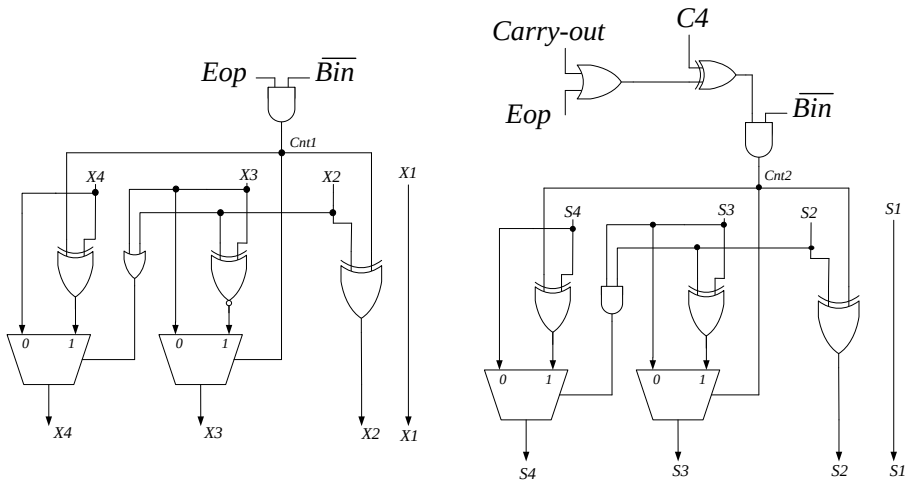


Figure 5.12 Illustration of BCD Subtraction operarion when $X \leq Y$

From the examples 5.2 and 5.3, the post-correction is needed only when $X > Y$, and digit wise carry-out is ‘0’ or when $X \leq Y$ and digit wise carry-out is ‘1’. In the proposed design, these conditions are incorporated in the post-correction stage [57].

As seen in the examples 5.1, 5.2 and 5.3, the correction is carried out by adding $(1010)_2$. The optimized implementation of the pre-correction block (which implements the +6 circuit) and the post correction block (which implements addition of $(1010)_2$) is shown in Fig 5.13 (a) and (b) respectively.



(a) (b)

Figure 5.13 (a) Pre-correction block (b) Post Correction block for BCD

The control signal for the pre-correction circuit is given as

$$Cnt1 = EOp.\ (Bin)'$$

which indicates that the addition of $(0110)_2$ is activated only for BCD addition operation. Similarly, the control signal for post correction block is given as

$$Cnt2 = ((Carry\text{-}out + EOp) \oplus C4).\ (Bin)'$$

$$Where\ C4 = (C_{1x})'(C_{2x})' as\ shown\ in\ the\ figure\ 5.11$$

which indicates the operation as BCD subtraction. It also takes into consideration both the cases of X >Y and X ≤ Y and also carry out at each digit stage [57].

5.3.4 A Modified Unified BCD/Binary Adder/Subtractor Architecture

The proposed unified BCD/Binary adder/subtractor architecture including pre-correction and post-correction stage is shown in Fig. 5.14. The 'Bin' signal indicates whether the operation is binary or BCD. If B_{in} =1 indicates binary operation and B_{in} = 0 indicates BCD operation [57].

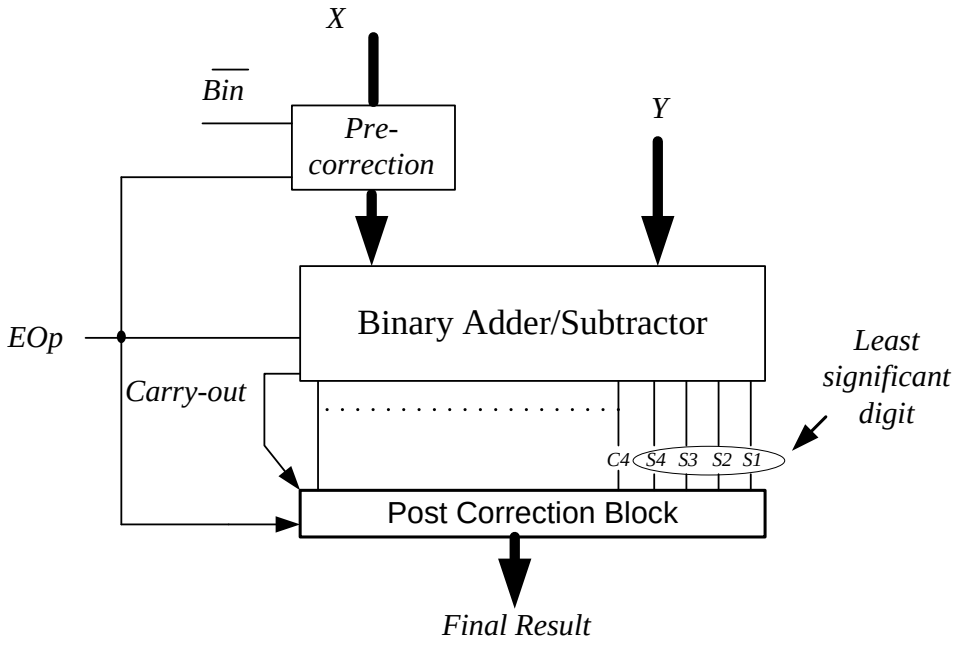


Figure 5.14 Architecture of unified BCD and binary adder / subtractor

The proposed architecture [57] eliminates the usage of complimentary stage unlike the Fischer's approach [51] and also the comparator stage unlike the Humberto's approach [53].

5.4 Simulations and Results

In this section, the proposed architecture is compared with Humberto architecture[53] but not with the architectures described in [47-52], as it is the only unified adder/subtractor architecture which supports 2's complement signed, unsigned, signed magnitude operands. Since, the Humberto architecture was implemented on an FPGA, in this thesis, both the proposed and the Humberto architectures have been implemented on an ASIC for a fair comparison.

The architectures have been structurally described using Verilog HDL and simulated using Cadence Incisive Unified Simulator (IUS) v6.1 covering all functional combinations. These architectures were mapped on to the TSMC 180*nm* technology typical library (operating conditions of 1.8 V, 25ºC), using Cadence RTL Compiler v7.1. Inputs were set to have a toggle rate of 50% and a frequency of 1GHz for calculating the dynamic power.

Table 5.2 and Fig. 5.15 provide a comparison of Humberto architecture with the proposed architecture. Since, the proposed design does not require the comparator and complex pre-computation stage, it results in a delay improvement of 13.6% and an area improvement of 14%.

The proposed approach can also be extended to higher operand lengths leading to efficient designs of unified BCD/Binary adder / subtractor architectures.

Table 5.2: Results for a 32-bit Unified BCD/Binary Adder/Subtractor

	Humberto [53]	**Proposed**
Delay (nS)	4.004 (100%)	3.460 (86.4%)
Power (mW)	14.5 (100%)	13.37 (92.2%)
Power-Delay (pJ)	58.06 (100%)	46.26 (79.7%)
Area (um^2)	12068 (100%)	10498 (87%)

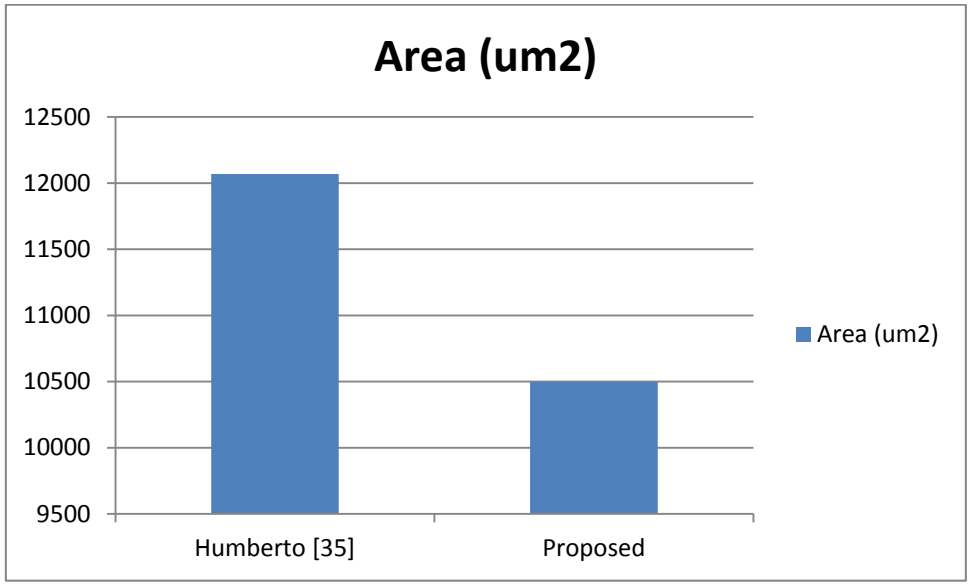


Figure 5.15 (a)

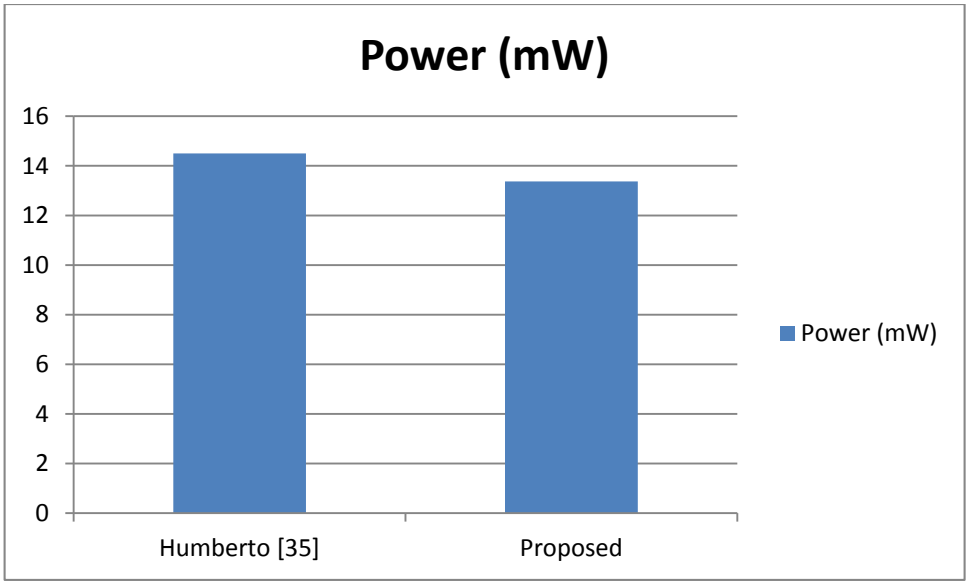


Figure 5.15 (b)

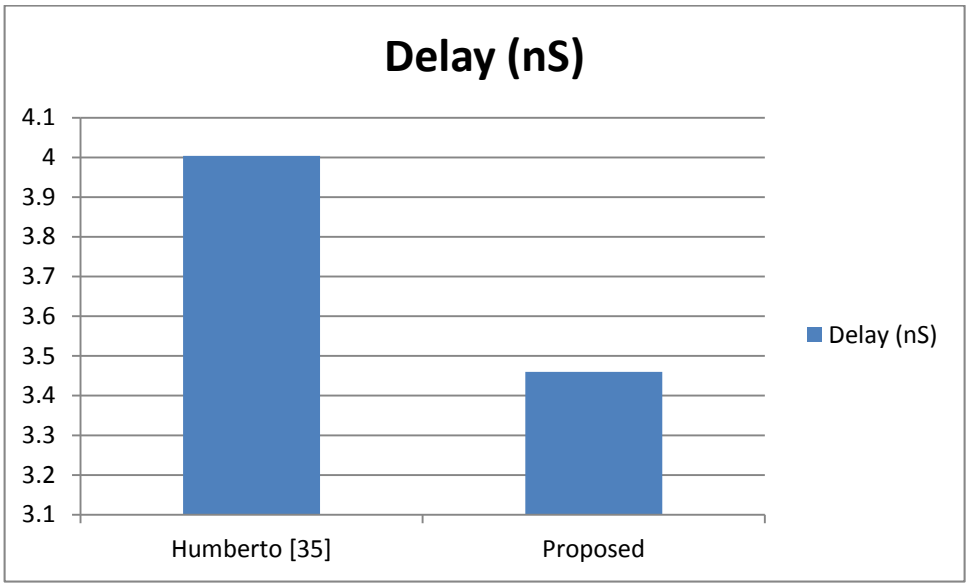


Figure 5.15 (c)

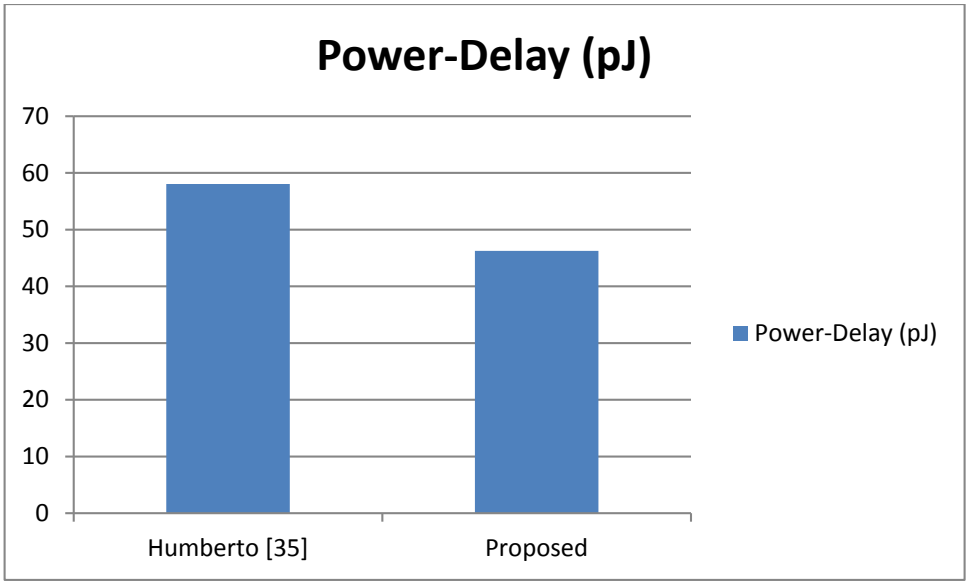


Figure 5.15 (d)

Figure 5.15 Comparision between proposed unified adder/subtractor with existing design in terms of (a) Area (b)Power (c) Delay (d) Power-Delay Product

5.5 Conclusions

In this chapter, efficient blocks for Binary and BCD arithmetic operations have been proposed. Also a unified BCD/Binary adder /subtractor which can handle both signed as well as unsigned numbers has been proposed and analyzed in detail. The results indicate that the proposed designs are efficient in terms of area, power and power- delay product when compared with those reported in literature.

Chapter 6
Design of Efficient Floating Point Adder and Multiplier

Contents

6.1 Floating Point Adder/Subtractor

6.1.1 Introduction

Floating point adder/subtractor units like fused floating point adder, triple path floating point adder, etc.., involve exponent comparison/subtraction, mantissa addition/subtraction and incrementing values while rounding, as basic operations. To realize these operations, efficient arithmetic units like comparators, adders, subtractors, incrementers are vital [1-6].

6.1.2 Design of Floating Point Units- General Implementation

Efficient floating-point unit are important in the design of arithmetic circuits for processors such as DSPs. A floating point adder/subtractor mainly consists of four different sections in the critical path viz swapping, shifting, addition/subtraction and normalization [58-61].

The operations in a typical 32-bit floating-point adder can be explained as follows: First the floating-point numbers are unpacked, i.e. sign bit, exponent and mantissa are isolated and the exponents are given as an input to a 8-bit comparator/subtractor. The mantissa with the smallest exponent is selected using a multiplexer and is right-shifted by a number of times equal to the difference of the exponents, making the exponents equal. The new mantissas are now added/ subtracted as per the control signal resulting in a difference that may be a positive or negative number. In case of a negative number, 2's complement of the number is taken to get the final result. Then, if required, the result is normalized and rounded. The architecture of the adder is shown in Fig 6.1.

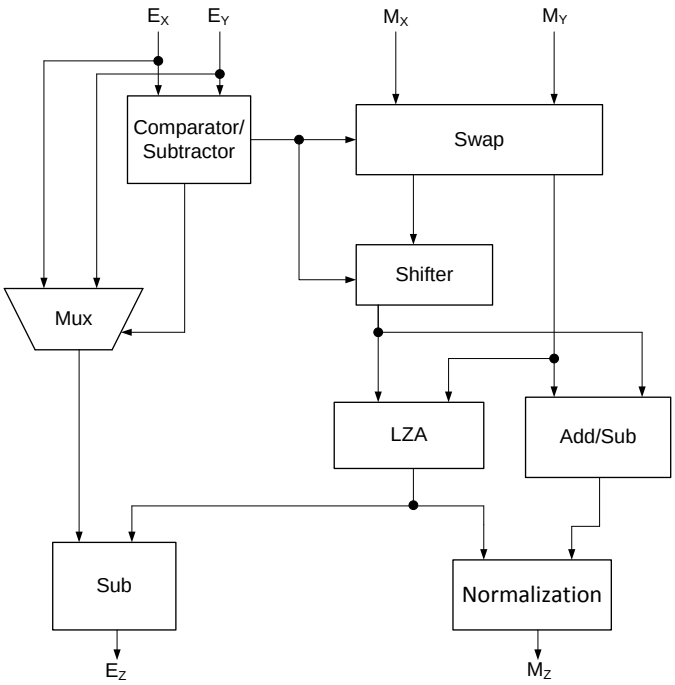


Figure 6.1 Architecture of a floating point adder/ subtractor

A comparator is required for exponent's comparison and these exponents are subtracted to initiate the operation of addition/subtraction. This can be taken care of by the comparator/subtractor block. Since these are the initial blocks which can't be avoided in the critical path, an optimized design is needed which can reduce the delay and power of these blocks.

Further, mantissas are added/subtracted according to a given operation by reusing an efficient adder/subtractor. Moreover a 2's complement block is required to correct the result during the subtraction. Some designs have been developed to eliminate the need of 2's

complement but that lead to circuit overhead. An efficient design which reduces the overheads is thus required.

Efficient and unified blocks have been developed as a part of this work and presented in the following.

6.1.3 Design of Efficient Binary Adder/Substractor

As explained earlier in section 5.3, a 2's complement block is required in adder/subtractor circuit for correcting the 2's complement sum when difference is negative. This correction however increases the circuit delay by log N and dissipates energy [58-61]. A design methodology has been proposed and discussed in Chapter 5 Section 5.3, which eliminates the requirement of 2's complement circuitry. In the proposed adder/subtractor this circuit is used to find out which of the operands is greater and the correspondingly a complement operation is carried out using the end around carry method. The binary adder/subtractor structure of the proposed design ,discussed in Chapter 5, section 5.3, is shown in Fig 6.2.

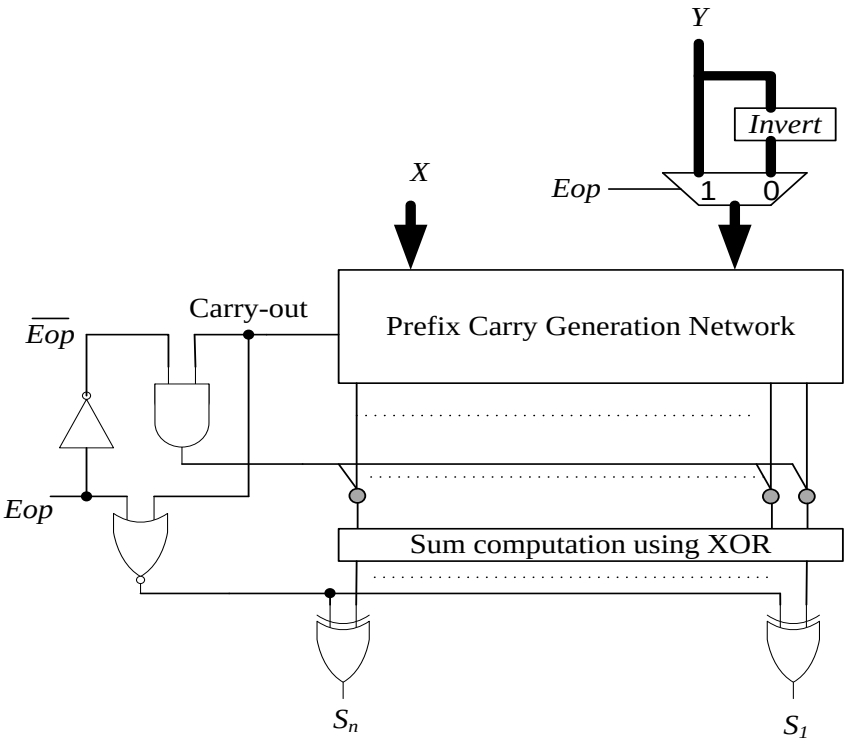


Figure 6.2 Implementation of Binary Adder/subtractor of Operands in signed magnitude form

6.1.3.1 Comparator for 32-bit floating point unit

A 32-bit single precision floating-point adder/subtractor unit requires an 8-bit comparator to check the exponents and an 8-bit subtractor to calculate the amount by which the mantissas are to be shifted. A subtractor can also be used as a comparator by analyzing the output Carry bit and hence extra comparator circuit is not required. But, if the difference of the subtractor is negative, a 2's complement circuitry is required to correct the final result. A 2's complement circuit requires an adder (or incrementer) to add '1' to correct the result which incurs a log N delay in the critical path and also contributes to overall area and power [58-61]. To overcome this problem, the design explained in section 5.3 is used to design an 8-bit subtractor cum comparator that avoids using 2's complement circuitry. This results in a significant improvement in critical path delay and also overall power and area.

Other than adder and comparator, floating point adder unit has leading zero anticipator (LZA) and a normalization unit (which includes rounding off). Many optimized designs have been developed for realizing these units [59] and hence these optimized circuits are used in this work along with the proposed comparator and adder designs for realizing an efficient floating point adder/subtractor unit.

6.1.4 Results and Comparison

All the units have been structurally described using Verilog HDL and simulated using Cadence Incisive Unified Simulator (IUS) v6.1 covering all functional combinations. These units were mapped on to the TSMC 180*nm* Technology typical library (operating conditions 1.8 V, 25ºC), using Cadence RTL Compiler v7.1. Inputs were set to have a toggle rate of 50% and a frequency of 1GHz for calculating dynamic power.

A complete 32-bit floating-point unit has been designed by integrating the designs presented in the previous chapters of this thesis. This proposed design [62] has been compared with the existing floating point units [58-61] and the results are given in Table 6.1.

Table 6.1 A Comparison of Performance of Floating Point Adder Units

	Area (um^2)	**Power (mW)**	**Delay (pS)**	**Power-Delay Product (pJ)**
Existing Design	20073.2 (100%)	95437.249 (100%)	10560 (100%)	1007.81 (100%)
Proposed Design	17759.102 (88.47%)	86070.872 (90.19%)	9823 (93.20%)	845.47 (83.89%)

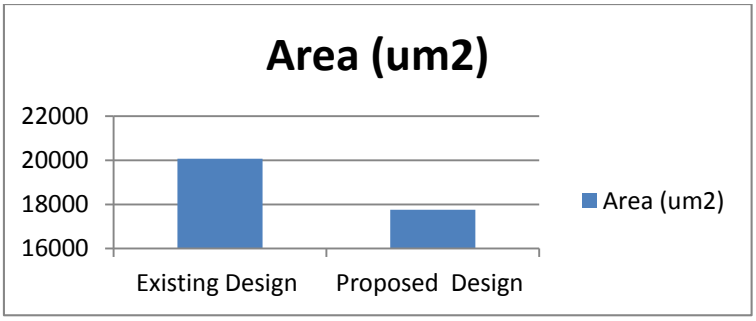


Figure 6.3 (a)

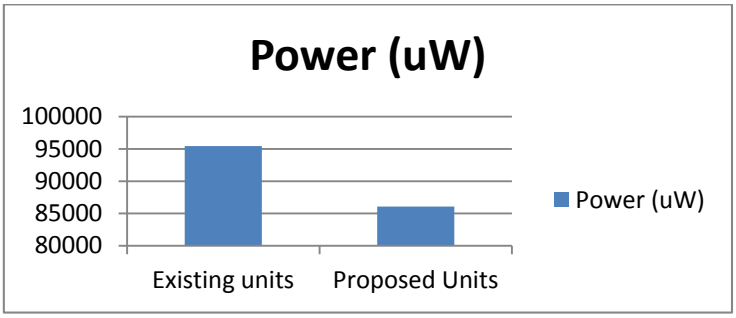


Figure 6.3 (b)

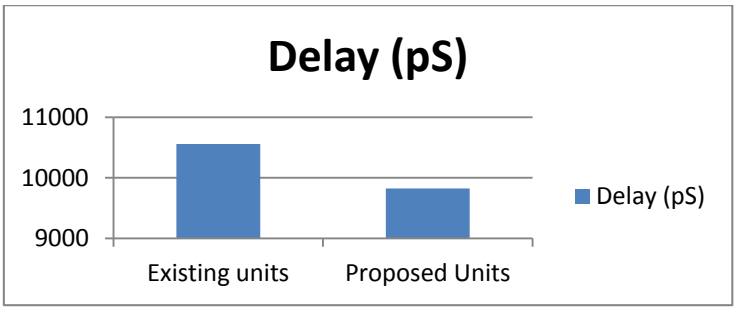


Figure 6.3 (c)

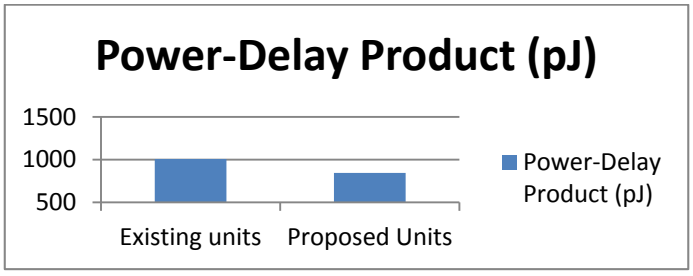


Figure 6.3 (d)

Figure 6.3 A Comparision of the proposed floating point adder unitd with the exiting design in terms of (a) Area (b)Power (c) Delay (d) Power-Delay Product

It is clear from Fig 6.3 that the proposed design has an improvement of 6.98% in terms of delay and around 16.11% improvement in power-delay product when compared with the existing floating point adder designs [58-61].

The overall delay is reduced because of the elimination of overhead related to 2's complement circuitry (thus the logN component of delay) in the exponent comparison block, mantissa subtraction block and reduction of large fan-out signals in the mantissa addition block.

6.2 Implementation of High Speed Multiplier

6.2.1 Introduction

A typical binary multiplier that multiplies two binary numbers is divided into three parts as shown in Fig 6.4. The first one is the partial product generation part . There are different methods for partial product generation i.e. Booths encoding scheme, AND gate logic, etc… These partial products are added in the second stage which is the partial product reduction tree. This stage can be realized by any one of the methodologies like using array structure of adders or compressors or counters. Finally in the third stage an adder is required to add the reduced partial products [1-6].

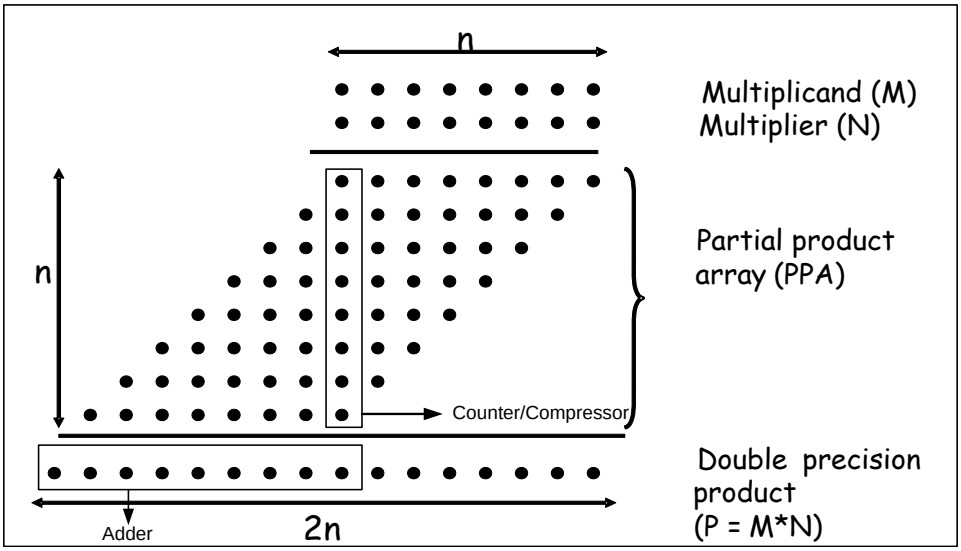


Figure 6.4 General Multiplier Structure

6.2.2 Design of Multipliers using Wallace and Dadda Algorithms

Wallace and Dadda are the first among those who designed and explained the usage of special structures called compressors and counters for partial product reduction tree in multipliers [25-26, 63-64]. Figure 6.5 and 6.6 show the design of a 16x16 bit multiplier partial

product reduction tree using compressors and counters respectively. These counters/compressors can also be used for multi-operand addition

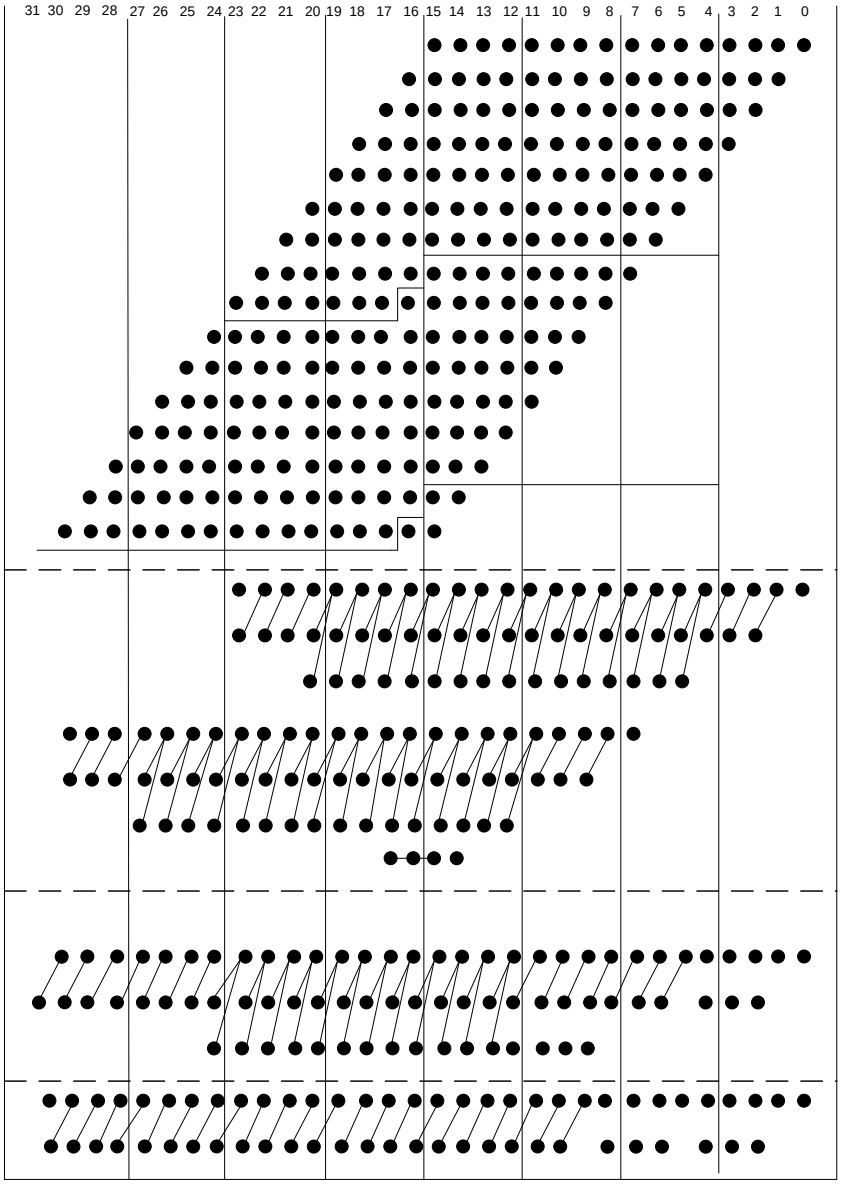


Figure 6.5 Wallace Algorithm for the design of a 16 bit Multiplier

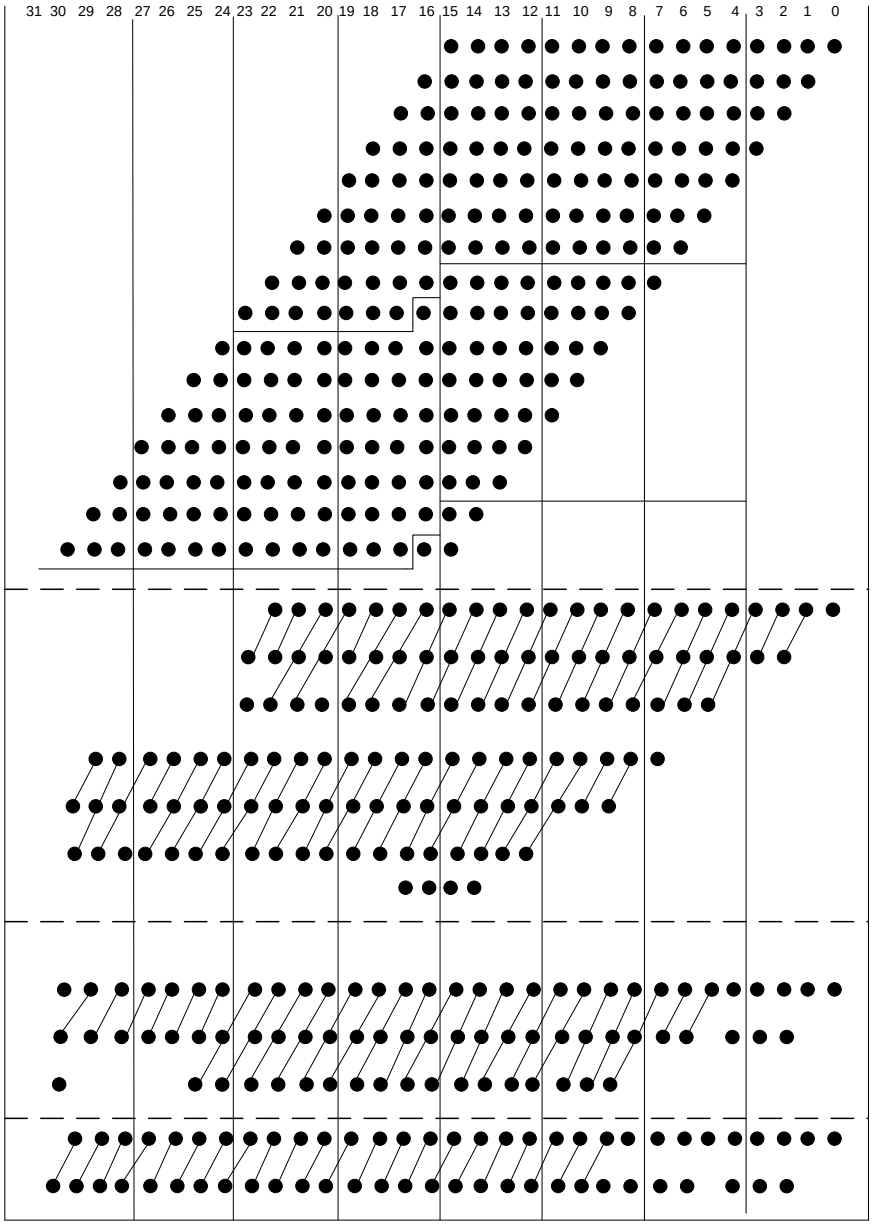


Figure 6.6 Dadda Algorithm for the design of a 16 bit Multiplier

Most of the multipliers consist of counters/compressors followed by a high speed adder. Critical path delay of multipliers thus includes not only counter/compressor delay in the partial product reduction tree but also adder delay in the final stage. Designing these units efficiently is the prime requirement for a high performance multiplier and therefore an ALU. A wide variety of parallel adders and counters/compressors exist in the literature and has been explained in detail in chapters 2 and 3.

From the results of different compressors and counters that have been presented, it is clear that these units are efficient. However, counter based design have proved to be more delay

efficient when compared to compressor based designs [63-64] and hence the multiplier in this section has been designed and implemented using counters. Further, the new adder explained in Chapter 2, Section 2.3 is used as a fast prefix adder in the final stage of the multiplier.

6.2.3 Simulation Results

All the units were structurally described using Verilog HDL and simulated using Cadence Incisive Unified Simulator (IUS) v6.1 covering all functional combinations. These units were mapped on to the TSMC 180*nm* Technology typical library (operating conditions 1.8 V, 25ºC), using Cadence RTL Compiler v7.1. Inputs were set to have a toggle rate of 50% and a frequency of 1GHz for calculating dynamic power.

Table 6.2, provides a comparison of existing and newly designed 32-bit multiplier. As the adder and counters used in this design are optimized in terms of power and delay, they result in improved performance of overall circuit. From Fig 6.7, it can be observed that there is a significant improvement in power-delay product (around 14.85%) when compared to the exiting implementation.

Table 6.2 Simulation Results of a 32-bit Multiplier

	Area(um2)	**Power (mW)**	**Delay(pS)**	**Power-Delay Product (pJ)**
Existing Design	25771.33 (100%)	145.628 (100%)	11740 (100%)	1709.67 (100%)
Proposed Design	23461.61 (91.04%)	136.302 (93.60%)	10680 (90.97%)	1455.71 (85.15%)

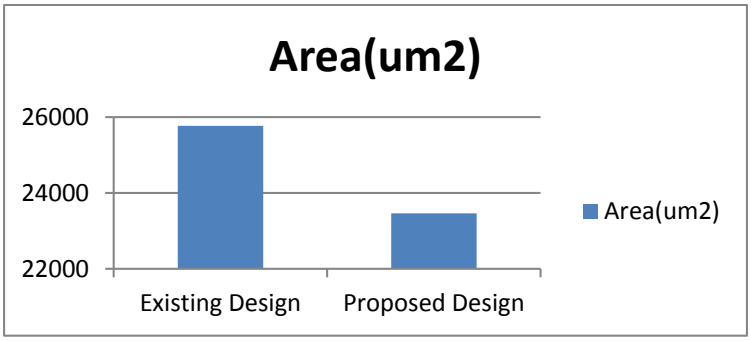


Figure 6.7 (a)

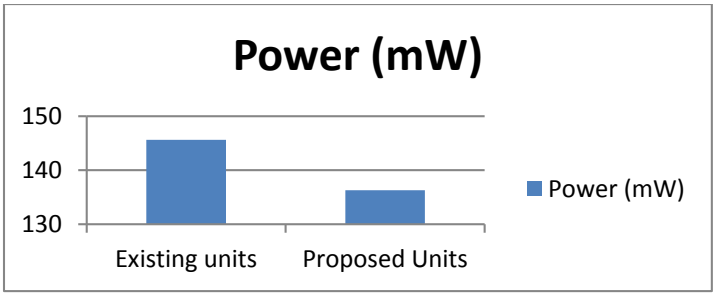


Figure 6.7 (b)

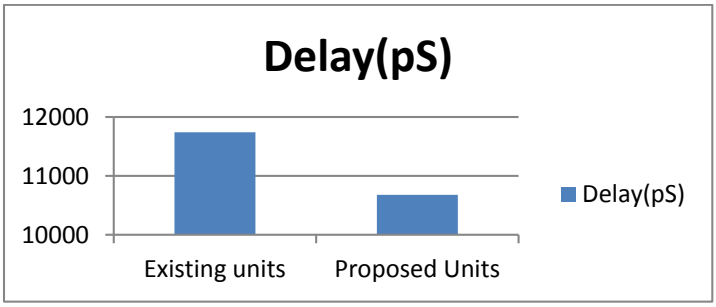


Figure 6.7 (c)

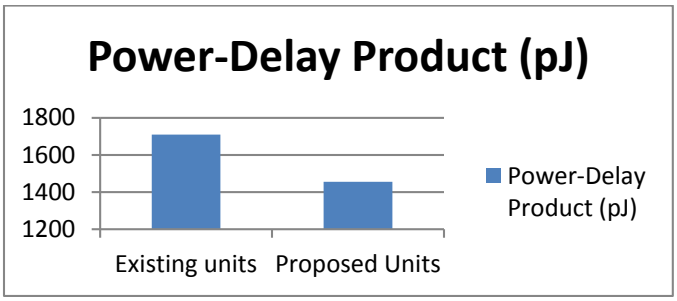


Figure 6.7 (d)

Figure 6.7 A Comparision of Exisitng 32-bit multipliers with the proposed design in terms of (a) Area (b)Power (c) Delay (d) Power-Delay Product

6.3 Conclusions

In this chapter, two widely used arithmetic blocks i.e., multiplier and floating point adder have been designed and their performance studied. Efficient basic functional units described in previous chapters have been used to implement these blocks. From the results obtained it can be seen that the arithmetic units explained in the previous chapter not only are efficient but also usage of these blocks results in performance enhancement of large functional arithmetic units.

Chapter 7
Design of Efficient Arithmetic Block in an Arithmetic and Logic Unit (ALU)

Contents

7.1 Introduction

Arithmetic and Logic Unit is of fundamental importance among all core units of any processor. Thus, optimization of ALU has been pursued for a long time. Arithmetic section of an ALU contains different blocks which perform different arithmetic functions like addition, subtraction, multiplication, incrementing, decrementing, etc... Hence, efficient design of these units is of prime importance to realize an efficient ALU.

7.2 Arithmetic Units in a Processor Core

Figure 7.1 illustrates the block level micro-architecture of a processor core inside an AMD microprocessor [65-68]. IBM (PowerPC), INTEL (ATOM) and ARM (Cortex) cores also will have similar micro-architecture for the arithmetic units in their respective processors. However, the main difference between IBM and other processor cores is that IBM supports decimal operations along with binary operations unlike other processors [69]. It can be noticed that the core has integer and floating point units wherein adder and multiplier play dominant roles [65-69].

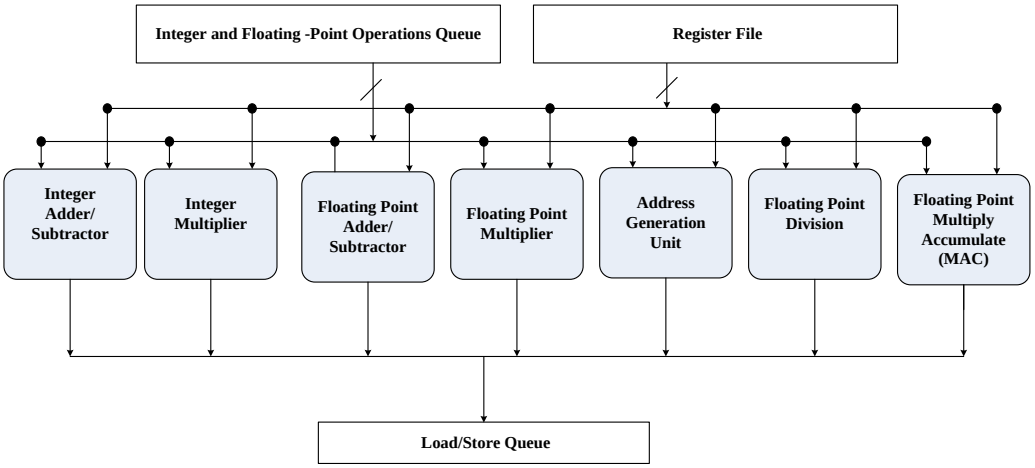


Figure 7.1 Microarchitecture of an Arithmetic unit in an AMD Processor Core

7.3 Efficient Arithmetic Units for a Processor Core

Figure. 7.2 shows the micro-architecture of an arithmetic segment with the proposed units. The main features of this segment are:

- Efficient adder circuit with modified sparse adder having less complexity in wiring
- Efficient tree multiplier using novel counter and compressor circuits
- A high speed INC/DEC unit
- A BCD adder/subtractor which can perform BCD operations using binary Adder/Subtractor as its core
- Binary floating point adder/subtractor unit

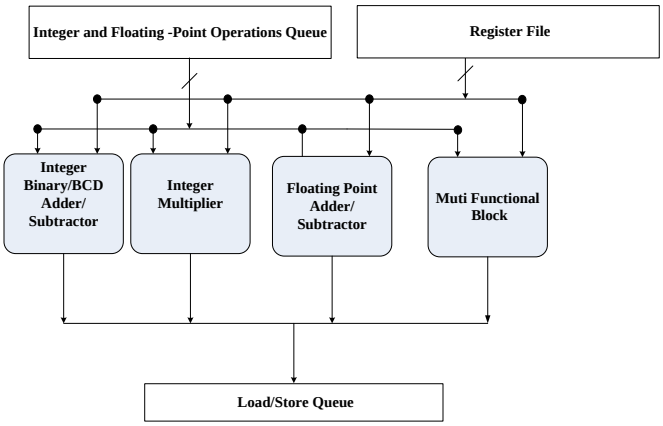


Figure 7.2 Processor Core with modifided functional units

A generic arithmetic segment of an ALU architecture is shown in Fig 7.3. This segment generally consists of a control unit, functional units and bus selection unit. The functional units

are made up of different operational units like a fixed-point operation unit, a floating point operation unit and a multiplier, etc… These units operational can be changed according to the application requirement.

In this chapter, efficient arithmetic units from previous chapters are substituted in place of functional units of the generic arithmetic segment to study its overall performance. The fixed-point adder/subtractor can perform both decimal and binary operations. The instruction format for this segment is as follows.

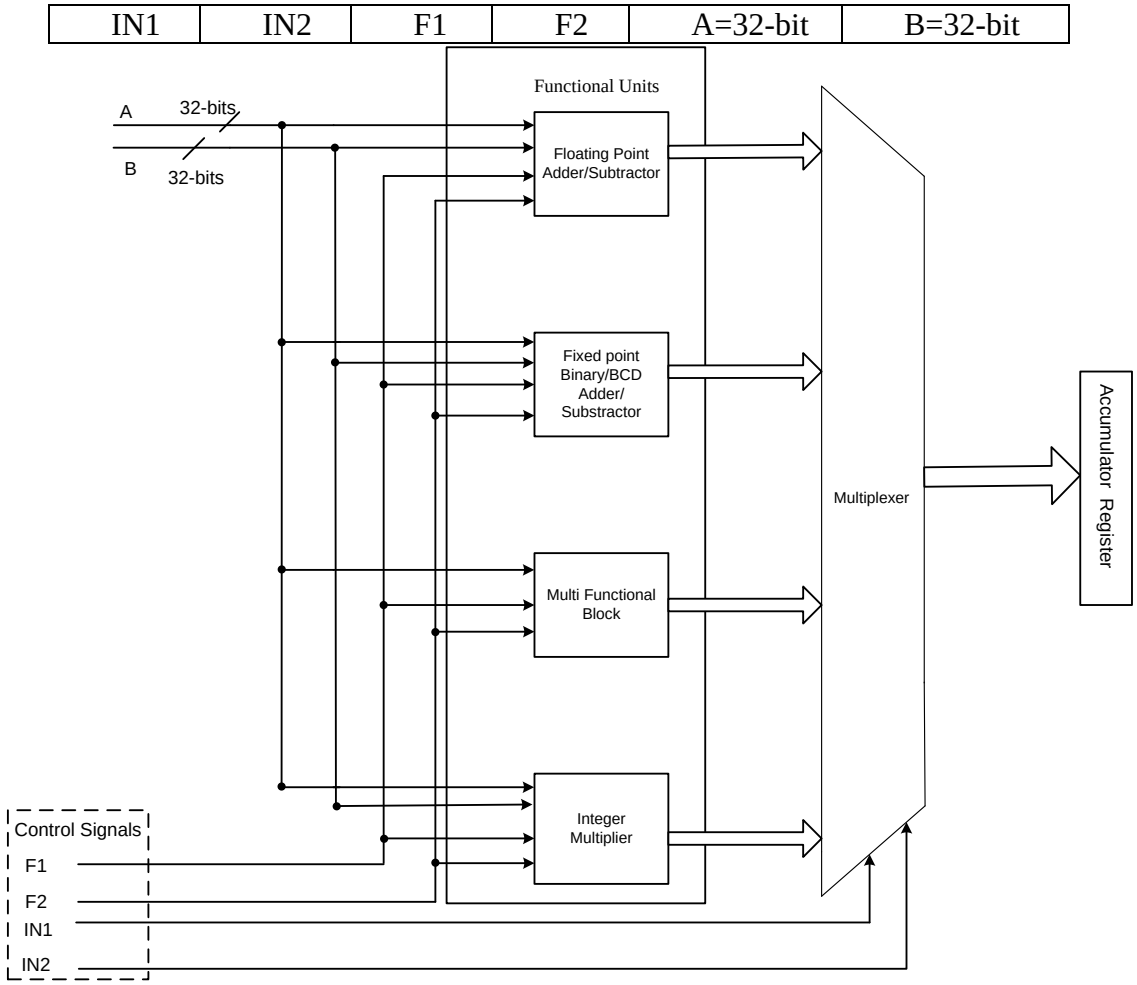


Figure 7.3 A generic architecture of an ALU

In the instruction format, the bits IN1, IN2, F1 and F2 indicate control bits and A and B are 32 bit data inputs. The control unit decodes the instruction and generates control signals that decide the functionality of the arithmetic block. Table 7.1 shows the detailed list of operations that can be performed. IN1 and IN2 control bits define the broad functionality of the block like fixed point operation, floating point operation, multiplication and function specific operations

whereas the control bits F1 and F2 select the specific functionality like addition, subtraction, etc… of the selected operation block. For example, if a fixed point addition needs to be executed (i.e., IN1 = 0, IN2 = 1, F1 = 0, F2 = 0), the decoder of the control unit selects only the fixed point operational block at the output multiplexer and F1, F2 bits select the addition operation.

Table 7.1: Detailed list of operations

IN1	IN2	F1	F2	OPERATION
0	0	0	X	FLOATING POINT ADDITION
0	0	1	X	FLOATING POINT SUBSTRACTION
0	1	0	0	FIXED POINT ADDITION
0	1	0	1	FIXED POINT SUBSTRACTION
0	1	1	0	FIXED POINT BCD ADDITION
0	1	1	1	FIXED POINT BCD SUBSTRACTION
1	0	0	0	INCREMENTER
1	0	0	1	DECREMENTER
1	0	1	0	PRIORITY ENCODER
1	0	1	1	2’S COMPLIMENT
1	1	X	X	MULTIPLICATION

7.3.1 Simulation Results

All the units were structurally described using Verilog HDL and simulated using Cadence Incisive Unified Simulator (IUS) v6.1 covering all functional combinations. These units were mapped on the TSMC 180*nm* Technology typical library (operating conditions 1.8 V, 25ºC), using Cadence RTL Compiler v7.1 and physical implementation is done by SOC encounter.

In the architecture shown in Fig. 7.3., best existing arithmetic units and those proposed units in this thesis were incorporated to compare the relative performance of the ALU. Table 7.2 presents a comparison of in terms of different design parameters such as area, delay, power power-delay, etc…

Table 7.2 Results of simulation results of ALU blocks

	Area (um2)		Power (mW)		Delay (pS)		Power-Delay Product (pJ)	
	Existing	**Proposed**	**Existing**	**Proposed**	**Existing**	**Proposed**	**Existing**	**Proposed**
Control Unit and Bus Selection Unit	825 (100%)	825 (100%)	0.001 (100%)	0.001 (100%)	166 (100%)	166 (100%)	0.00017 (100%)	0.00017 (100%)
Floating point Adder/ Subtractor	20073.2 (100%)	17759.1 (88.47%)	95.437 (100%)	86.07 (90.12%)	10560 (100%)	9823 (93.20%)	1007.81 (100%)	845.47 (83.89%)
Fixed point Binary/BCD Adder/ Subtractor	12068 (100%)	10498 (86.99%)	14.5 (100%)	13.37 (92.21%)	4004 (100%)	3460 (86.41%)	58.06 (100%)	48.67 (83.83%)
Multi Functional Block	1432 (100%)	1637 (114.32%)	0.619 (100%)	0.624 (100.81%)	3497 (100%)	2333 (66.71%)	2.17 (100%)	1.46 (67.28%)
Fixed Point Multiplier	25771.3 (100%)	23461.6 (91.04%)	145.628 (100%)	136.302 (93.60%)	11740 (100%)	10680 (90.97%)	1709.67 (100%)	1455.71 (85.15%)

However , the multifunctional block with proposed units occupies 14% more area and results in a marginal increase in power consumption of about 0.8%. However, the same block performs well both in terms of delay and power-delay product.

Thus it can be expected that the arithmetic and logic units incorporating the efficient functional units proposed in this thesis performs better than with the existing units resulting in an improvement of overall processor performance.

It can be observed from the table that most of the blocks such as floating point adder/subtractor, fixed point Binary/BCD adder/subtractor and fixed point multiplier performs better compared to the corresponding functional units existing in the literature in terms of area, power, delay and power-delay products.

Figure 7.4 shows the pie chart of the power contributed by the individual arithmetic units in the overall arithmetic section. From the pie chart, it can be observed that the multiplier is the highest power consumed unit in the section followed by floating point unit. As explained earlier, even for an addition operation, the multiplier and floating point units are active and contributes to the overall power, which can be handled by using power-gating technique in the ALU.

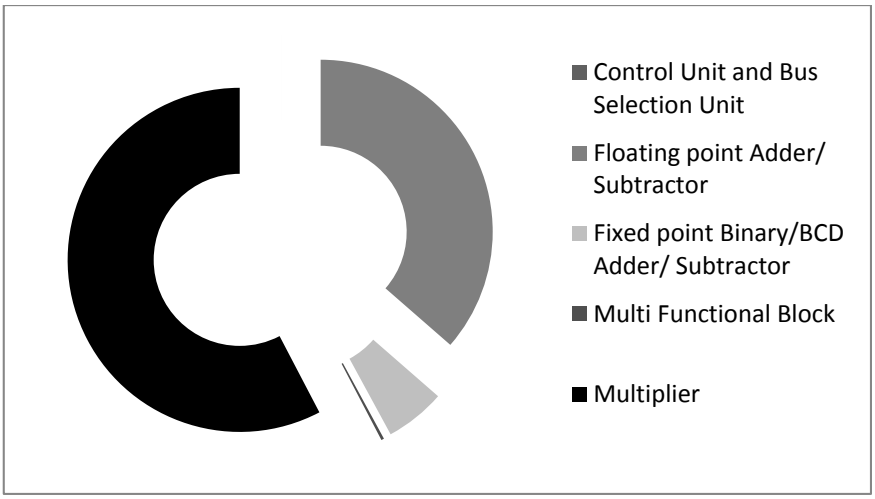


Figure 7.4 Power contribution of different arithmetic blocks in ALU

In this architecture, irrespective of the operation, all the blocks will be active and contribute to the total power. In order to address this issue and to reduce overall power consumption, any of the low power techniques like power-gating technique can be incorporated in the design. In this work, this technique has been incorporated to understand its effect on over power consumption in the generic ALU. Power gating technique is briefly explained below.

7.4 Power gating applied to the arithmetic units.

In arithmetic and data path circuits, the performance mainly depends upon the digital circuits. A prime requirement for a digital circuit is that it performs the function it is designed for. However, as the technology is scaling down to nanometer, there is a scaling of V_t along with V_{dd} to maintain traditional 30% delay reduction with technology. This results in different types of leakage currents due to which the measured response of fabricated circuit may deviates from the expected response [70]. Leakage currents can be broadly classified in to are dynamic and static leakage currents. Dynamic leakage occurs when the circuit is switching or operating (gate leakage) whereas static leakage occurs when the circuit is in standby or in idle state (sub-threshold). Some of the well known leakage reduction techniques are (a) pre-determined input vector method (b) forced stacking (c) sleep transistor technique (power gating), (d) Multi-voltage scaling (e) dual-V_t design (g) clock gating, etc [70]. However, power gating has become one of the most widely used circuit design techniques for reducing of leakage current.

Power gating is a technique wherein circuit blocks that are not active are temporarily turned off to reduce the overall leakage power [70]. These units are typically power gated by header and footer transistors, when in one operational unit is computing these power gating transistors switches off other computational units. Generally, even when digital circuit is in idle state there will be a leakage current passing from the power source to ground resulting in leakage power. To reduce this power, the circuit should have connection to the power Supply (V_{DD}) and ground (GND) only when they are active and cut-off when they are in idle state. This can be achieved by having a power switch cell like header and footer transistors, as illustrated below.

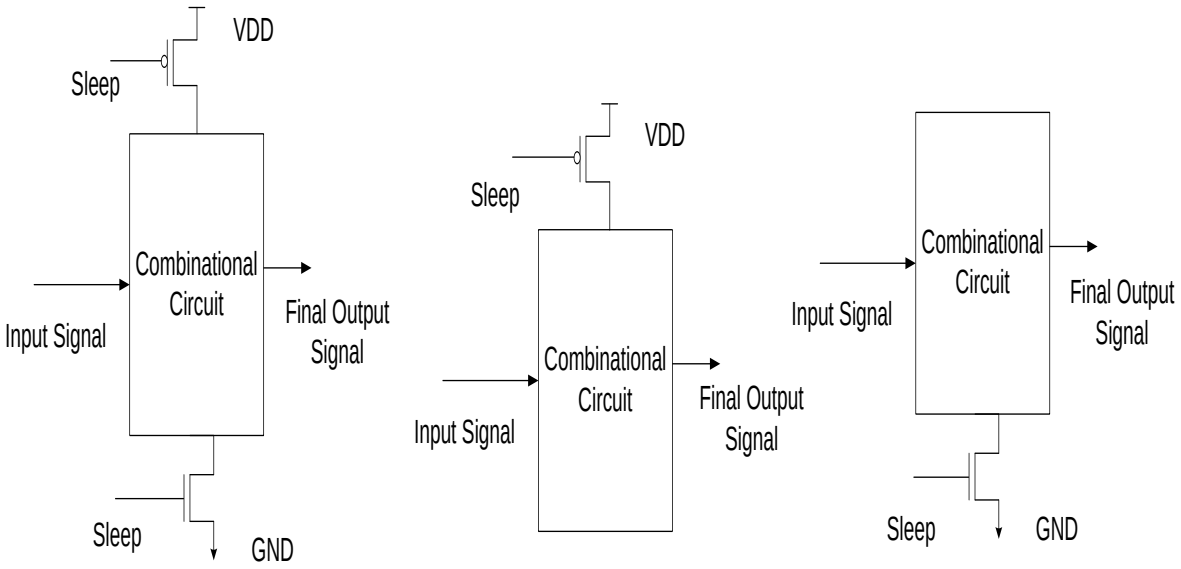


Figure 7.5 Power Gating Technique

It can be seen from the figure 7.5 the circuit incorporating power gating technique is designed by adding a header transistor (PMOS) to the VDD and a footer transistor (NMOS) to the GND or only a header transistor to VDD or only a footer transistor to GND depending on the design specification. These header/footer transistors have two modes of operation 'ON' or 'OFF' depending on the "Sleep" Signal. When the 'sleep' signal is active, the circuit will be connect to the VDD and/or GND and works as a normal circuit, otherwise the circuit will be disconnected from the VDD and/or GND and thus doesn't have a power source/ or path to ground thereby reducing the leakage power [70].

Figure 7.6 shows the architecture of arithmetic section of an ALU with power gating technique. The control unit, which controls the power gating transistors, is also shown in the figure. The power gating transistors act as switches to connect the main power supply V_{DD} and

ground V_{SS}, with derived (or power switch nets) supplies like V_{DD1}/V_{SS1}, V_{DD2}/V_{SS2}, V_{DD3}/V_{SS3} and V_{DD4}/V_{SS4} for fixed point operation, floating point operation, function specific block and multiplier respectively. The decoder of the control unit takes IN1 and IN2 bits as inputs and generates four control signals (I0, I1, I2, I3) for power gating logic that enables/disables different blocks in the segment. For example, if a fixed point addition need to be executed (i.e., IN1 = 0, IN2 = 1, F1 = 0, F2 = 0), the decoder of the control unit selects only fixed point operational block and helps in power gating the other blocks thus reducing the power consumption. The proposed arithmetic units are integrated in this architecture to realize an efficient arithmetic section of an ALU.

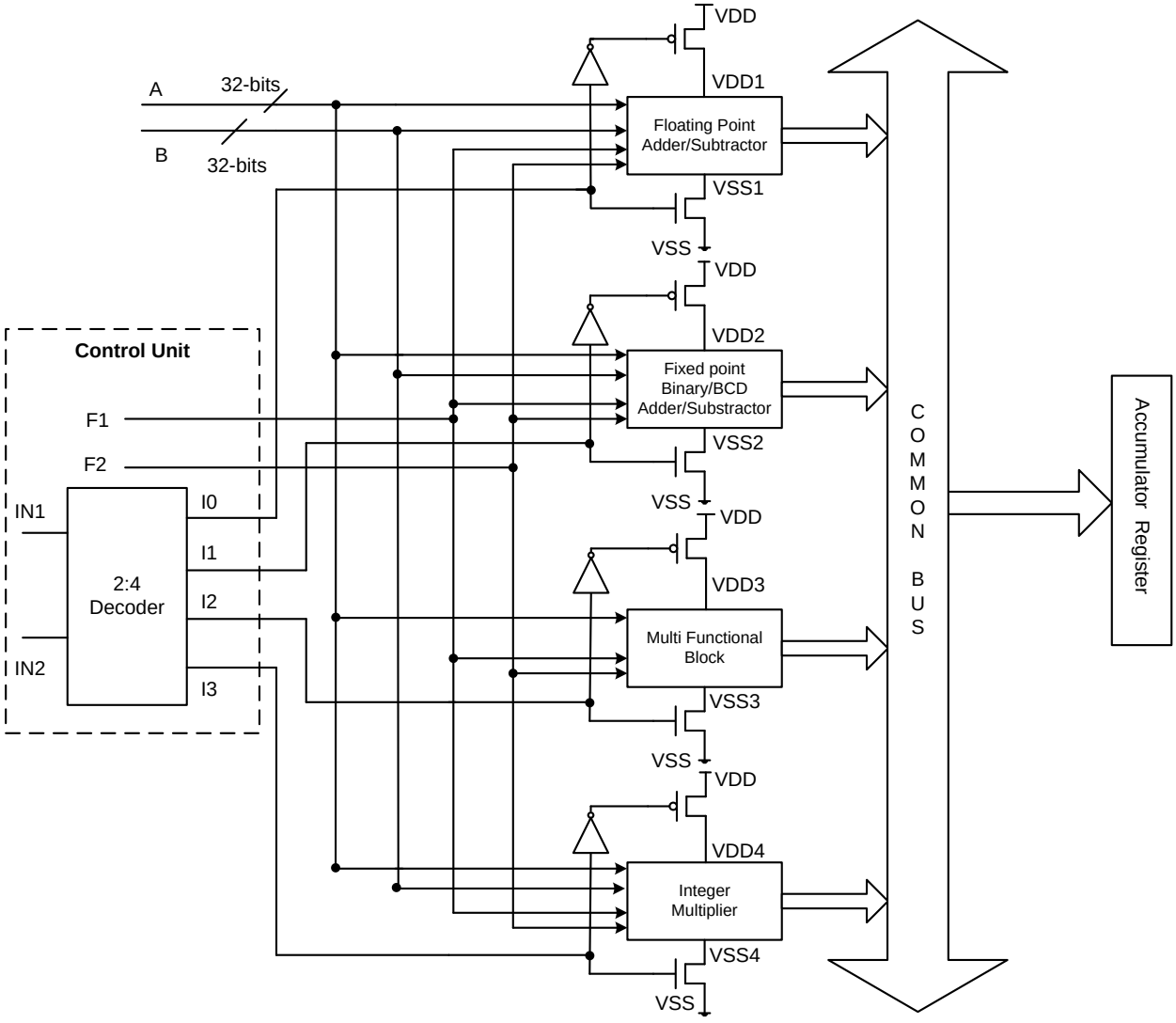


Figure 7.6 Arithmetic section of an ALU with power gating technique

7.4.1 Simulation Results

All the units were structurally described using Verilog HDL and simulated using Cadence Incisive Unified Simulator (IUS) v6.1 covering all functional combinations. These units were mapped on the TSMC 180*nm* Technology typical library (operating conditions 1.8 V, 25ºC), using Cadence RTL Compiler v7.1 and physical implementation is done by SOC encounter.

Power gating technique is incorporated in the architecture to reduces the power consumed by the overall ALU. These power gating cells are available in the design library. For the current design, based upon delay requirements along with area optimization, different power gating cells have been inserted in the RTL compiler input file and validated. From the synthesis report, the (W/L) ratios of power gating header (PMOS) and footer (NMOS) transistors have been found to be 4u/0.18um and 1.8u/0.18um respectively for optimized delay and area. Vt was 0.32V (approx.). The control unit with the help of select lines will enable respective arithmetic operation block and switching off other arithmetic operation blocks using power-gating transistors. Hence, the power consumed by the entire arithmetic unit is the power dissipated by the selected arithmetic operation block, control unit, power-gating unit and bus selection unit. From the table 7.3 it can be seen that when the power gating unit is enabled and the floating point operation is performed, the leakage and dynamic power dissipated by other units are reduced which results in almost 60% reduction in power consumption when compared with the segment without power gating.

Table 7.3 Simulation Results for ALU while performing floating-point addition operation

Type	**Power (mW)**
Without Power-Gating Technique	236.367
With Power-Gating Technique	97.30

7.5 Conclusions

In this chapter, a segment of a processor core i.e., ALU has been designed using the arithmetic units proposed earlier and a detailed analysis has been carried out. Results indicate that this design results in a better performance in terms of area, delay and power when compared with those reported in literature. Further, incorporating power-gating technique has resulted in significant savings in power as is evident from the results.

Chapter 8
Conclusions and Future Work

Contents

8.1 Conclusions

This thesis focused on optimizing arithmetic circuits which when used together lead to efficient realization of Arithmetic Unit of an ALU.

The first contribution of the thesis was the development of efficient adder architectures that address the problems of high fan-out, delay and power consumption. These architectures while having a delay overhead have the advantage of relatively small fan-out and reduced energy consumption overall. Further, efficient counter/compressor blocks, that help in reducing the partial products in multipliers, have been designed resulting in efficient high speed parallel multipliers.

The second contribution of the thesis is the design of a multi-functional INC/DEC/ 2's complement/ priority encoder circuit which has been shown to be efficient in terms of speed of operation without resulting in extra power consumption. Since such a unit plays a major role in an ALU, its incorporation results in efficient arithmetic and logic units.

The third contribution of the thesis is the design of a unified BCD/Binary adder/subtractor that is efficient in terms of delay while consuming less energy when compared to similar existing designs.

The fourth contribution of the thesis is the design of two widely used arithmetic blocks i.e., multiplier and floating point adder using the functional blocks mentioned above resulting in their efficient implementation.

Finally, all the individual arithmetic units have been combined to realize the arithmetic part of an ALU resulting in an efficient design compared to those existing in literature.

8.2 Future Work

The arithmetic units designed and implemented in this work have mainly been targeted at 90nm or 180nm technologies. It is well known, however, that the dynamic power dominates the overall power consumption at these technology nodes. With shrinking technology, where designs are being targeted at even 18nm, new challenges are being thrown up such as leakage power dominating the overall power consumption. Thus, it would be interesting to understand the performance of the designs proposed in this work at lower technology nodes.

Bibliography

[1] B. Parhami, “Computer Arithmetic: Algorithms and Hardware Designs”, 2nd edition, Oxford University Press, New York, 2010

[2] Milos Ercegovac, Tomas Lang, "Digital Arithmetic", Morgan Kaufman, 2004.

[3] I . Koren, Computer Arithmetic Algorithms. Englewood Cliffs, NJ, Prentice Hall, 1993.

[4] R. Zimmermann, Binary Adder Architectures for Cell-Based VLSI and their Synthesis, PhD thesis, Swiss Federal Institute of Technology (ETH) Zurich, Hartung-Gorre Verlag, 1998.

[5] Neil Weste, David Harris, "CMOS VLSI Design: A Circuits and Systems Perspective" Pearson Education, 2004.

[6] Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic, “Digital Integrated. Circuits: A Design Perspective” Pearson Education, 2003.

[7] Weinberger, J.L. Smith, “A Logic for High-Speed Addition,” Nat. Bur. Stand. Circ., 591:3-12, 1958.

[8] H. Ling, “High-Speed Binary Adder,” IBM Journal of Research and Development, vol. 25, no.3, pp. 156-166, May 1981

[9] Harris D, “A taxonomy of parallel prefix networks”, in. Proc. 37th Asilomar Conf. Signals, Systems, and. Computers, Nov. 2003, Vol. 2, pp. 2213-2217.

[10] R.P. Brent and H.T. Kung, “A Regular Layout for Parallel Adders,” IEEE Trans. Computers, vol. 31, no. 3, pp. 260-264, Mar. 1982.

[11] D. Harris, " Logical Effort of Higher Valency Adders," in Proc. 38th Asilomar Conf. Signals, Systems, and Computers, vol. 2, pp. 1358 - 1362, 2004.

[12] S. Mathew, M.A. Anders, B. Bloechel, T. Nguyen, R.K. Krishnamurthy, S. Borkar, “A 4GHz 300-mW 64-bit integer execution ALU with dual supply voltages in 90nm CMOS,” IEEE J. Solid State Circuits, vol. 40, no.1, pp. 44-51, Jan. 2005.

[13] Kumashikar et al., “Sparse Tree Adder”, US Patent 20070299902A1.

[14] T.L. Lynch, E.E. Swartzlander, “A Spanning Tree Carry Lookahead Adder”, IEEE Trans. Comput., Vol.41, N°8, pp.931-939, 1992.

[15] V. Kantabutra, “A Recursive Carry-Lookahead/Carry-Select Hybrid Adder”, IEEE Trans. Comput., Vol.42, N°12, pp.1495-1499, 1993.

[16] O. Kwon, E. Swartzlander, and K. Nowka, “A fast hybrid Carry-lookahead/Carry-select adder design”, Proc. of the 11th Great Lakes symposium on VLSI, pp.149-152, March 2001.

[17] Oklobdzija, V.G.; Zeydel, B.R.; Dao, H.; Mathew, S.; Ram Krishnamurthy; , "Energy-delay estimation technique for high-performance microprocessor VLSI adders," Computer Arithmetic, 2003. Proceedings. 16th IEEE Symposium on , vol., no., pp. 272- 279, 15-18 June 2003.

[18] Tyagi, A.; , "A reduced-area scheme for Carry-select adders," Computers, IEEE Transactions on , vol.42, no.10, pp.1163-1170, Oct 1993

[19] Chetan Vudadha, Sai Phaneendra, Syed Ershad Ahmed, Sreehari Veeramachaneni, Moorthy Muthukrishnan and Srinivas M.B "An Improved Sum Computation Block for adders with High Sparseness", in 20th International Workshop on Logic & Synthesis (IWLS 2011), June 2011, San Diego, CA, USA.

[20] Sai Phaneendra P, Sreehari Veeramachaneni, N Moorthy Muthukrishnan, M.B. Srinivas, "Conditional Sum Block for High Sparse Adders", The 2011 Asia Pacific Conference on Postgraduate Research in Microelectronics & Electronics (PrimeAsia), 2011, Macau, China, 6-8 October 2011. (GOLD LEAF Certificate)

[21] Chetan Kumar, V; Sai Phaneendra, P; Ershad Ahmed, S; Sreehari, V; Moorthy Muthukrishnan, N; Srinivas, M.B.; , "Higher radix sparse-2 adders with improved grouping technique," TENCON 2011 - 2011 IEEE Region 10 Conference , vol., no., pp.676-679, Fukuoka, Japan, 21-24 Nov. 2011.

[22] A. A Farooqui, V. G. Oklobdzija, F. Chehrazi, "64-Bit Media Adder", Proceedings of the 1999 IEEE International Symposium on Circuits and Systems, (ISCAS '99), Orlando, Florida, USA, 30 May-2 June 1999.

[23] J.-F. Li, J.-D. Yu, Y.-J. Huang, "A design methodology for hybrid Carry-lookahead/Carry-select adders with reconfigurability", in Proc. IEEE International Symp. on Circuits and Systems (ISCAS), pp.77-80, May2005.

[24] Chetan Kumar V., Sai Phaneendra P., S. Ershad Ahmed, Sreehari Veeramachaneni, N. Moorthy Muthukrishnan, M. B. Srinivas: A Prefix Based Reconfigurable Adder, IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2011), Chennai, India,July 4-6, 2011.

[25] C. S. Wallace, A suggestion for a fast multiplier, IEEE Trans. on Electronic Comp. EC-13(1): 14-17 (1964)

[26] L.Dadda, "Some Schemes for Parallel Multipliers," Alta Frequenza, Vol.34, pp.349-356,1965.

[27] K. Prasad and K. K. Parhi, "Low-power 4-2 and 5-2 compressors," in Proc. of the 35th Asilomar Conf. on Signals, Systems and Computers, vol. 1, 2001, pp. 129–133.

[28] C. H. Chang, J. Gu, M. Zhang, "Ultra low-voltage low-power CMOS 4-2 and 5-2 compressors for fast arithmetic circuits" IEEE Transactions on Circuits and Systems I: Regular Papers, Volume 51, Issue 10, Oct. 2004 Page(s):1985 – 1997

[29] R. Zimmermann and W.Fichtner, "Low-power logic styles: CMOS versus pass-transistor logic," IEEE J. Solid-State Circuits, vol. 32, pp. 1079–1090, July 1997.

[30] Sreehari Veeramachaneni , Kirthi Krishna M, Lingamneni Avinash, Sreekanth Reddy P, M.B.Srinivas, "Novel Architectures for High-speed and Low-power 3-2, 4-2 and 5-2 Compressors" Proceedings of the 20th IEEE/ACM International Conference on VLSI

Design and Embedded Systems(VLSI DESIGN -2007), Bangalore, India, 6-10th January 2007.

[31] Earl E. Swartzlander, Jr., “A review of large parallel counter designs,” Proceedings. IEEE Computer society Annual Symposium on VLSI, 2004, pp.89-98, Feb. 2004.

[32] Earl E. Swartzlander, Jr., “Parallel Counters,” IEEE Transactions on computers, Vol C-22, pp.1021-1024, Nov. 1973.

[33] Sreehari Veeramachaneni, Lingamneni Avinash, Kirthi Krishna M, M.B.Srinivas, “Novel Architectures for Efficient (m, n) Parallel Counters” In 17th ACM Great Lakes Symposium on VLSI (GLSVLSI-2007), Stresa-Lago Maggiore, Italy, March 11-13, 2007.

[34] R. Hashemian and C. P. Chen. “A New Parallel Technique for Design of Decrement/Increment and Two's Complement Circuits.” in Proceedings of the 34th Midwest Symposium on Circuits and Systems, volume 2, pages 887-890, 1991.

[35] Shaoqiang Bi, Wei Wang, and Asim Al-Khalili, “Multiplexer-based Binary Incrementer/decrementers,” The 3rd International IEEE-NEWCAS Conference,19-22 June 2005. pp. 219-222.

[36] Veeramachaneni, S. Avinash, L. Kirthi, K.M. Srinivas, M.B. “A Novel High-Speed Binary and Gray Incrementer/Decrementer for an Address Generation Unit”, International Conference on Industrial and Information Systems (ICIIS), 9-11 August 2007. pp.427-430.

[37] Kumar, V. Chetan; Phaneendra, P. Sai, Ahmed, Syed Ershad, Sreehari, V., Muthukrishnan, N. Moorthy, Srinivas, M.B., "A Reconfigurable INC/DEC/2's Complement/Priority Encoder Circuit with Improved Decision Block," International Symposium on Electronic System Design (ISED), 2011, vol., no., pp.100-105, 19-21 Dec. 2011.

[38] Phaneendra, P.S.; Vudadha, C.; Ahmed, S.E.; Sreehari, V. Muthukrishnan, N,.M. Srinivas, M.B. , "Increment/decrement/2's complement/priority encoder circuit for varying operand lengths," 11th International Symposium on Communications and Information Technologies (ISCIT), 2011 , vol., no., pp.472-477, 12-14 Oct. 2011

[39] D. Norris, “Comparator circuit,” U.S. Patent 5,534,844, April 3, 1995.

[40] Shun-Wen Cheng , “A high-speed magnitude comparator with small transistor count” in Proceedings of the 2003 10th IEEE International Conference on Electronics, Circuits and Systems (ICECS), pp. 1168-1171, 2003.

[41] Stine, J.E., Schulte, M.J., “A combined two's complement and floating-point comparator,” IEEE International Symposium on Circuits and Systems (ISCAS), pp. 89-92, 2005.

[42] Sreehari Veeramachaneni ,Kirthi Krishna M, Lingamneni Avinash , Sreekanth Reddy P, M. B. Srinivas, “Efficient Design of a 32-Bit Comparator Using Carry Look-Ahead Logic" in Joint Conference on 50th IEEE Mid West Symposium on Circuits and Systems (MWSCAS-2007) and 5th North East Symposium on Circuits and Systems (NEWCAS-2007), August 5-8, 2007, Montreal, Canada.

[43] Michael F. Cowlishaw, “Decimal Floating-Point: Algorism for Computers”, IEEE Symposium on Computer Arithmetic 2003: 104-111, June 2003.

[44] Morris Mano, “Digital Design”, Third Edition, Prentice Hall.

[45] Thapliyal, H.; Kotiyal, S.; Srinivas, M.B., “Novel BCD adders and their reversible logic implementation for IEEE 754r format”, 19th International Conference on VLSI Design, 20063-7 Jan. 2006

[46] M.S.Schmookler and A. Weinderger. “Decimal Adder for Directly Implementing BCD Addition Utilizing Logic Circuitry”, International Business Machines Corporation, *US* patent *3629565*, pages 1 – 19, Dec 1971.

[47] M. J. Adiletta and V. C. Lamere. “BCD Adder Circuit”. Digital Equipment Corporation, US patent 4805131, pages 1 – 18, Jul 1989.

[48] U. Grupe.“Decimal Adder“, Vereinigte Flugtechnische Werke-Fokker gmbH, US patent 3935438, pages 1 – 11, Jan 1976.

[49] S. Hwang. “High-Speed Binary and Decimal Arithmetic Logic Unit”, American Telephone and Telegraph Company, AT&T Bell Laboratories, US patent 4866656, pages 1-11, Sep 1989.

[50] Flora, Laurence P., “Fast BCD/Binary Adder”, US Patent 5007010.

[51] H. Fischer and W. Rohsaint. “Circuit Arrangement for Adding or Subtracting Operands Coded in BCD-Code or Binary-Code”, Siemens Aktiengesellschaft, US patent 5146423, pages 1 – 9, Sep 1992.

[52] W. Haller, U. Krauch, and H. Wetter. “Combined Binary/Decimal Adder Unit”, International Business Machines Corporation, US patent 5928319, pages 1-9, Jul 1999.

[53] D.R.Humberto Calderón, G. N. Gaydadjiev, S. Vassiliadis, “Reconfigurable Universal Adder”, Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP 07), pages 186-191, July 2007.

[54] Sreehari Veeramachaneni ,Kirthi Krishna M, Lingamneni Avinash , Sreekanth Reddy P, "Novel, High-Speed 16-Digit BCD Adders Conforming to IEEE 754r Format",: in the proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI-2007), May 9-11, 2007, Porto Alegre, Brazil.

[55] Sreehari Veeramachaneni ,Kirthi Krishna M, Subroto Sen, Prateek G V, Bharat Sankhlecha, M.B.Srinivas, “A Novel Carry-look ahead approach to an Unified BCD and Binary Adder/Subtractor”, In the Proceedings of the 21st IEEE/ACM International Conference on VLSI Design and Embedded Systems(VLSI DESIGN -2008), Hyderabad , India, 4th -8th January 2008.

[56] Anshul Singh, Aman Gupta, Sreehari Veeramachaneni, M.B.Srinivas, “A High Performance Unified BCD and Binary Adder/Subtractor”, In the proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI-2009), May 13-15, 2009, Tampa, Florida.

[57] Chetan Kumar, V. Sai Phaneendra, P. Ahmed, Syed Ershad, Veeramachaneni, Sreehari; Muthukrishnan, N. Moorthy; Srinivas, M.B.; , "A Unified Architecture for BCD and

Binary Adder/Subtractor," Digital System Design (DSD), 2011 14th Euromicro Conference on , vol., no., pp.426-429, Aug. 31 2011-Sept. 2 2011.

[58] H. H. Saleh and E. E. Swartzlander, Jr., “A Floating-Point Fused Add-Subtract Unit,” Proceedings of the 51st IEEE Midwest Symposium on Circuits and Systems, 2008, pp. 519 - 522.

[59] Jongwook Sohn, Earl E. Swartzlander Jr. “Improved Architectures for a Fused Floating-Point Add-Subtract Unit”. IEEE Trans. on Circuits and Systems 59-I(10): 2285-2291 (2012)

[60] Jongwook Sohn “Improved architectures for a fused floating-point add-subtract unit”, M.S. Thesis , The University of Texas at Austin,2011.

[61] H.H. Saleh, "Fused Floating-Point Arithmetic for DSP", PhD dissertation, Univ. of Texas, 2009.

[62] Sreehari Veeramachaneni, M. B. Srinivas, “Design of Efficient Arithmetic Circuits for Realization of Floating Point Adder/Subtractor Units”, 2013 IEEE/IFIP 21th International Conference on VLSI and System-on-Chip (VLSI-SoC), 7-9 Oct. 2013.

[63] W.J. Townsend, E.E. SwartzlanderJr., and J.A. Abraham, "A Comparison of Dadda and Wallace Multiplier Delays," *Proc. SPIE*, Advanced Signal Processing Algorithms, Architectures, and Implementations XIII, pp. 552-560, 2003.

[64] Ron S. Waters, Earl E. Swartzlander, “A Reduced Complexity Wallace Multiplier Reduction”, IEEE Transactions on Computers, vol. 59, no. 8, pp. 1134-1137, Aug. 2010.

[65] Butler, Mike. “Bulldozer: A new approach to multithreaded compute performance”, Hot Chips XXII, August 2010.

[66] Conway, P. et al. “Blade Computing with the AMD Opteron Processor (“Magny-Cours”)”, Hot Chips XXI, August 2009.

[67] www.realworldtech.com/barcelona

[68] D. A. Patterson, J. L. Hennessy “Computer Organization & Design: The Hardware/Software Interface”, Revised 4th Edition, Morgan Kaufmann Publishers (Elsevier), 2012.

[69] Kalla R, Sinharoy B “POWER7: IBM's next generation server processor”, IEEE symposium on high performance chips (Hot chips 21), Stanford, August 2009.

[70] Narendra, Siva G. Chandrakasan, Anantha (Eds.), “Leakage in Nanometer CMOS Technologies” Springer Publications, 2006.

List of Publications

Journals

[J1] Sandeep Saini ,Mahesh Kumar, Sreehari Veeramachaneni, M.B.Srinivas, "An Alternate approach to Buffer Insertion for Delay and Power Reduction in VLSI Interconnects", The Journal of Low Power Electronics (JOLPE), Vol. 6, number 3, pp 429-435 October 2010.

[J2] Mahesh Kumar, Sreehari Veeramachaneni, M.B.Srinivas, "Design of a Low Power Variable-Resolution Flash ADC", The Journal of Low Power Electronics (JOLPE), Vol. 5, number 3, pp 279-290 October 2009.

[J3] Sreehari Veeramachaneni, M.B. Srinivas "Design of Efficient Arithmetic Circuits for Realization of Floating Point Adder/ Subtractor Units" (Communicating to Microelectronics Journal – Elsevier)

[J4] Sreehari Veeramachaneni, M.B. Srinivas "Efficient Hardware Accelerator For Accurate Scientific And Numerical Computing." (Communicating to Microelectronics Journal – Elsevier)

Conferences

[C1] Sreehari .Veeramachaneni , Pradeep Yarlagadda, and M. B. Srinivas "A Fully Multiplexer-based Implementation of Redundant Number System" Proceedings of the 15th IEEE/ACM SIGDA International Workshop on Logic & Synthesis (IWLS-2006), June 7-9, 2006, Vail, Colorado ,.USA.

[C2] Sreehari Veeramachaneni , Kirthi Krishna M, Lingamneni Avinash, Sreekanth Reddy P, M.B.Srinivas, "Design of a High Performance and Efficient Arithmetic Logic Unit (ALU)", In International Microelectronics & Packaging Society (IMAPS)India National Conference (IINC-2006) on Microelectronics & VLSI, Hyderabad, India, 8-10th December 2006. (Best paper award).

[C3] Sreehari Veeramachaneni, Lingamneni Avinash, Rajashekhar Reddy M, M.B.Srinivas "Efficient Modulo (2k ±1) Binary to Residue Converters" Proceedings of the 6th IEEE International Workshop on SOC Real-Time Applications (IWSOC- 2006), Cairo, Egypt, December 26-28th 2006.

[C4] Sreehari Veeramachaneni , Kirthi Krishna M, Lingamneni Avinash, Sreekanth Reddy P, M.B.Srinivas, "Novel Architectures for High-speed and Low-power 3-2, 4-2 and 5-2 Compressors" Proceedings of the 20th IEEE/ACM International Conference on VLSI Design and Embedded Systems(VLSI DESIGN -2007), Bangalore, India, 6-10th January 2007

[C5] Sreehari Veeramachaneni, Lingamneni Avinash, Kirthi Krishna M, M.B.Srinivas, "Novel Architectures for Efficient (m, n) Parallel Counters" In 17th ACM Great Lakes

Symposium on VLSI (GLSVLSI-2007), Stresa-Lago Maggiore, Italy, March 11-13, 2007.

[C6] Sreehari Veeramachaneni , Kirthi Krishna M, Lingamneni Avinash, Sreekanth Reddy P, M.B.Srinivas, "Novel High-Speed Redundant Binary to Binary Converter Using Prefix Networks" IEEE International Symposium on Circuits and Systems (ISCAS-2007) May 27-30, 2007, New Orleans, USA

[C7] Sreehari Veeramachaneni, Lingamneni Avinash, Rajashekhar Reddy M, M.B.Srinivas, Efficient Implementations of Residue to Binary Converters for (2k-1, 2k , 2k +1) Moduli Set, The 2007 World Congress in Computer Science, Computer Engineering, and Applied Computing (WORLDCOMP'07), Monte Carlo Resort, Las Vegas, Nevada, USA (June 25-28, 2007).

[C8] Sreehari Veeramachaneni , Mayank Agarwal, Siddartha K, Rajashekhar Reddy M, Sri Harish Reddy M and M. B. Srinivas "New Full Adder Cells for Sub-threshold Operations" Proceedings of the 16th IEEE/ACM SIGDA International Workshop on Logic & Synthesis (IWLS-2007), May 30 - June 1, 2007,Paradise Point Resort & Spa San Diego, CA ,USA. (Co-located with the (DAC) Design Automation Conference).

[C9] Sreehari Veeramachaneni ,Kirthi Krishna M, Lingamneni Avinash , Sreekanth Reddy P, "Novel, High-Speed 16-Digit BCD Adders Conforming to IEEE 754r Format",: in the proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI-2007), May 9-11, 2007, Porto Alegre, Brazil.

[C10] Sreehari Veeramachaneni ,Kirthi Krishna M, Lingamneni Avinash , M B Srinivas, " A Novel High-Speed Binary and Gray Incrementer/Decrementer for an Address Generation Unit," In 2nd IEEE International Conference on Industrial and Information Systems (ICIIS 2007), 8-11, August 2007, University of Peradeniya, Srilanka.

[C11] Sreehari Veeramachaneni ,Kirthi Krishna M, Lingamneni Avinash , Sreekanth Reddy P, M. B. Srinivas, " Efficient Design of a 32-Bit Comparator Using Carry Look-Ahead Logic" in Joint Conference on 50th IEEE Mid West Symposium on Circuits and Systems (MWSCAS-2007) and 5th North East Symposium on Circuits and Systems (NEWCAS-2007), August 5-8, 2007, Montreal, Canada.

[C12] Sreehari Veeramachaneni ,Kirthi Krishna M, Subroto Sen, Prateek G V, Bharat Sankhlecha, M.B.Srinivas, "A Novel Carry-look ahead approach to an Unified BCD and Binary Adder/Subtractor", In the Proceedings of the 21st IEEE/ACM International Conference on VLSI Design and Embedded Systems(VLSI DESIGN -2008), Hyderabad , India, 4th -8th January 2008 .

[C13] Sreehari Veeramachaneni, M.B.Srinivas, "New Improved 1-Bit Full Adder Cells", In 21st IEEE Canadian Conference on Electrical and Computer Engineering (CCECE-2008) Symposium on Circuits, Devices and Systems, Sheraton Fallsview Niagara Falls Ontario, Canada, May 4-7, 2008

[C14] Sreehari Veeramachaneni, M.B.Srinivas, 'Novel High-Speed Architecture for 32-Bit Binary Coded Decimal (BCD) Multiplier,In International Symposium on Communications and Information Technologies (ISCIT-2008), Don Chanh Palace Hotel, Vientaine , Lao PDR, October 21-23,2008.

[C15] Mahesh Kumar ,Sreehari Veeramachaneni, M.B.Srinivas, "Low Voltage High Performance Flash ADC", In 9th of the biennial Asia Pacific Conference on Circuits and Systems (IEEE APCCAS 2008), Venetian Macao-Resort-Hotel, Macao, China November 30 - December 3, 2008.

[C16] Sreehari Veeramachaneni, A. Mahesh Kumar , Venkat Tummala, M.B.Srinivas,"Design of a Low Power Variable-Resolution Flash ADC", In the Proceedings of the 22nd IEEE/ACM International Conference on VLSI Design and Embedded Systems (VLSI DESIGN -2009),New Delhi , India, 5th -9th January 2009.

[C17] Anshul Singh, Aman Gupta, Sreehari Veeramachaneni, M.B.Srinivas, "A High Performance Unified BCD and Binary Adder/Subtractor", In the proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI-2009), May 13-15, 2009, Tampa, Florida.

[C18] Sreehari Veeramachaneni , M.B.Srinivas, "Redefining CMOS Logic Style for Subthreshold Operation" In the proceedings of the IEEE 5th International Conference on Ph.D. Research in Microelectronics and Electronics (PRIME'09) ,University College Cork, Ireland, 12-17 July.

[C19] Mahesh Kumar, Sreehari Veeramachaneni , M.B.Srinivas, "Design of Low Power and High Speed Reconfigurable Resolution two step flash ADC" ,In the proceedings of the 1st Asia Symposium on Quality Electronic Design (ASQED 2009) 15-16 July 2009 in Kuala Lumpur, Malaysia.

[C20] Mahesh Kumar, Sreehari Veeramachaneni , M.B.Srinivas, "A Novel Low Power, Variable Resolution Pipelined Analog to Digital Converter" ,In the proceedings of the 22nd IEEE International SOC conference (IEEESOCC 2009) 9th -11th September 2009 in Belfast, Northern Ireland, UK.

[C21] Ronak Bajaj, Saransh Chhabra, Sreehari Veeramachaneni M.B.Srinivas "'A Novel, Low-Power Array Multiplier Architecture" In the proceedings of the 9th International Symposium on Communications and Information Technologies (ISCIT 2009) 28th-30th September 2009 in Incheon, Korea.

[C22] Sandeep Saini, Sreehari Veeramachaneni, M.B.Srinivas, "Schmitt Trigger as an Alternative to Buffer Insertion for Delay and Power Reduction in VLSI Interconnects", Tencon 2009, 23-26 Nov 2009.

[C23] Sandeep Saini, A. Mahesh Kumar, Sreehari Veeramachaneni, M.B.Srinivas, "Alternative approach to Buffer Insertion for Delay and Power Reduction in VLSI Interconnects", Proceedings of the 23th IEEE/ACM International Conference on VLSI Design and Embedded Systems(VLSI DESIGN -2010), Bangalore, India, 3-7th January 2010.

[C24] Mahesh Kumar Adimulam, Krishna Kumar Movva, Sreehari Veeramachaneni, N. Moorthy Muthukrishnan, M.B.Srinivas: A low power, variable resolution two-step flash ADC. ACM Great Lakes Symposium on VLSI 2010 (GLSVLSI 2010): pp.39-44, Brown University Campus, Providence, Rhode Island, USA ,May 16-18, 2010.

[C25] Mahesh Kumar Adimulam, Sreehari Veeramachaneni, N. Moorthy Muthukrishnan, M.B.Srinivas , " A Novel, Variable Resolution Flash ADCwith Sub Flash Architecture ",: in the proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI-2010), July 5-6, 2010,lixouri,Kefalonia,Greece.

[C26] Mahesh Kumar Adimulam , Sreehari Veeramachaneni , M.B.Srinivas, "Towards Realizing Variable Resolution Analog to Digital Converters" In the proceedings of the 2010 Asia Pacific Conference on Postgraduate Research in Microelectronics & Electronics (PRIME-ASIA 2010)September 22 – 24, 2010 Shanghai Olympic Hotel, China.

[C27] Mahesh Kumar, Sreehari Veeramachaneni , N. Moorthy Muthukrishnan, M.B.Srinivas, "A New Low Power Flash ADC with Configurable Resolution" In TENCON-2010, Fukuoka, Japan ,November 21-24, 2010.

[C28] Mahesh Kumar, Sreehari Veeramachaneni, N. Moorthy Muthukrishnan, M.B.Srinivas, "A Multiple-Bandwidth 12-bit Pipelined Analog to Digital Converter with Self-Clock Generator" In International Symposium on Communications and Information Technologies (ISCIT-2010, October 26-29 2010,), Meiji University, Tokyo, Japan.

[C29] Mahesh Kumar, Sreehari Veeramachaneni , N. Moorthy Muthukrishnan, M.B.Srinivas, "Low Power, Variable Resolution Pipelined Analog to Digital Converter with Sub Flash Architecture" In the Proceedings of the 11th Biennial Asia Pacific Conference on Circuits and Systems (APCCAS2010), December 6th -9th 2010 , Kuala Lumpur ,Malaysia.

[C30] Chetan Vudadha, Sai Phaneendra, Syed Ershad Ahmed, Sreehari Veeramachaneni, Moorthy Muthukrishnan and Srinivas M.B "An Improved Sum Computation Block for adders with High Sparseness ", in 20th International Workshop on Logic & Synthesis (IWLS 2011), June 2011, San Diego, CA, USA.

[C31] Chetan Kumar V., Sai Phaneendra P., S. Ershad Ahmed, Sreehari Veeramachaneni, N. Moorthy Muthukrishnan, M. B. Srinivas: A Prefix Based Reconfigurable Adder, IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2011), Chennai, India,July 4-6, 2011.

[C32] Chetan Kumar, V. Sai Phaneendra, P. Ahmed, Syed Ershad, Veeramachaneni, Sreehari; Muthukrishnan, N. Moorthy; Srinivas, M.B.; , "A Unified Architecture for BCD and Binary Adder/Subtractor," Digital System Design (DSD), 2011 14th Euromicro Conference on , vol., no., pp.426-429, Oulu, Finland,Aug. 31 2011-Sept. 2 2011.

[C33] Ahmed, Syed Ershad, Sreehari Veeramachaneni, N Moorthy Muthukrishnan, M.B. Srinivas, "Reconfigurable Adders for Binary/BCD addition/Subtraction", The IEEE 2011

Asia Pacific Conference on Postgraduate Research in Microelectronics & Electronics (PrimeAsia), 2011, Macau, China,6-8 October .

[C34] Sai Phaneendra P, Sreehari Veeramachaneni, N Moorthy Muthukrishnan, M.B. Srinivas, “Conditional Sum Block for High Sparse Adders”, The 2011 IEEE Asia Pacific Conference on Postgraduate Research in Microelectronics & Electronics (PrimeAsia), 2011, Macau, China,6-8 October 2011. (GOLD LEAF Certificate).

[C35] Phaneendra. P, Vudadha. C, Ahmed.S, Sreehari Veeramachaneni, Muthukrishnan, N, Srinivas, M.B., "Increment/decrement/2's complement/priority encoder circuit for varying operand lengths," 11th International Symposium on Communications and Information Technologies (ISCIT), 2011 , vol., no., pp.472-477, 12-14 Hangzhou, China,Oct. 2011

[C36] Chetan Kumar, Sai Phaneendra, Ershad Ahmed, Sreehari Veeramachaneni, Moorthy Muthukrishnan, Srinivas M.B., "Higher radix sparse-2 adders with improved grouping technique," TENCON 2011, IEEE Region 10 Conference , vol., no., pp.676-679, 21-24 Bali, Indonesia, Nov. 2011.

[C37] V. Chetan, Phaneendra, P, Syed Ershad, SreehariVeeramachaneni, Muthukrishnan Moorthy, Srinivas, M.B, "A Reconfigurable INC/DEC/2's Complement/Priority Encoder Circuit with Improved Decision Block," International Symposium on Electronic System Design (ISED), 2011, vol., no., pp.100-105, 19-21 Kochi, India Dec. 2011.

[C38] Sai Phaneendra P, Chetan Vudadha, Goutham Makkena, Venkata Swamy Nayudu Mandala, Sreehari Veeramachaneni, Ershad Ahmed S, Moorthy Muthukrishnan N and Srinivas M.B. “Low Power Self Reconfigurable Multiplexer Based Decoder for Adaptive Resolution Flash ADCs” ", In the Proceedings of the 25nd IEEE/ACM International Conference on VLSI Design and Embedded Systems (VLSI DESIGN -2012), Hyderabad , India, 7th -11th January.

[C39] Sreehari Veeramachaneni, M. B. Srinivas, “Design of Efficient Arithmetic Circuits for Realization of Floating Point Adder/Subtractor Units”, 2013 IEEE/IFIP 20th International Conference on VLSI and System-on-Chip (VLSI-SoC), Istanbul, Turkey, 7-9 Oct. 2013.

[C40] Sreehari Veeramachaneni, M. B. Srinivas, “Design of Optimized Arithmetic Circuits for Multiplier Realization”, The 2013 IEEE Asia Pacific Conference on Postgraduate Research in Microelectronics & Electronics (PrimeAsia), 2013, Visakhapatnam, India, 19-21 December 2013.

[C41] B Naveen Kumar Reddy, Chandra Sekhar Mummidi, Sreehari Veeramachaneni and M B Srinivas. “A Novel Low Power Error Detection Logic for Inexact Leading Zero Anticipator in Floating Point Units”, In the Proceedings of the 27th IEEE/ACM International Conference on VLSI Design and Embedded Systems (VLSI DESIGN - 2014), Mumbai , India, 5th -9th January 2014.

[C42] Sai Phaneendra P, Chetan Vudadha, Sreehari Veeramachaneni, Srinivas M.B. “An Optimized Design of Reversible Quantum Comparator”, In the Proceedings of the 27th IEEE/ACM International Conference on VLSI Design and Embedded Systems (VLSI DESIGN -2014), Mumbai , India, 5th -9th January 2014 .