Summary:

This dissertation undertakes the analysis of the Jackson and Talwar's approach for designing adder. The project includes designing a low valency parallel prefix adder based on Jackson and Talwar's approach of factorisation. Jackson and Talwar's approach is a generalisation of Ling's Theory of factorisation. It evaluates approach for designing a fast and a small adder by reducing the fan-out in a design and calculating the delay caused by the logic gates. It locates and examines the lacunae present in Jackson and Talwar's approach and analyses the design of a fast and small adder by reducing the complexity in the R(reduced group) and thus making D (not kill) group much more complicated. It further looks at implementing the Sparse carry tree technique for better performance in an adder. Sparse tree method reduces the fan-out of the logic gates and so the delay in a circuit. The current adders touches the Ling's theory of factorisation for reducing the complexity in the logic but Jackson and Talwar have the better tendency to reduce the complexities of the terms to a depth by manipulating the equations while designing adders. This thesis reveals the ways in which Jackson and Talwar's approach can be utilised in different ways in an adder and also it reflects the benefits of using sparse trees in conjunction with Jackson and Talwar's approach resulting in small and fast adders. This dissertation seeks to provide a richer context for the evaluation of Jackson and Talwar's approach for designing an adder than Ling's theory, and in doing so proposes new ways of looking at the perspective. Furthermore it verifies that sparse carry tree is a good method to be implemented in designing.

- I have designed three 16-bit parallel prefix adder using different logics.
- The practical implementation of the theory given by Jackson and Talwar for designing novel adders is done. Further analysing novel designs to fill in the gap between Jackson and Talwar approach and designing in CMOS VLSI.
- Comparison using Sparse carry tree is done in two designs for analysing these adders to give better performance in terms of area and speed.

Table of Contents

List of Tables

List of Figures

1 Introduction:

Addition of two binary numbers is a critical operation underlying digital microelectronics. It is desirable to have an adder with good performance and area trade off characteristics as adders are responsible for cycle time in processors. There are various types of adders such as carry - skip adder, conditional - sum adder, carry look - ahead adder, carry select adder and hybrid adders. Selection of an adder structure involves diverse choice of algorithm as well as its own advantages and disadvantages. Various adders are designed for achieving different targets like high speed, less area and low power. For almost all the adders, the logic can be divided into two types namely Carry logic and Sum logic. Implementation of these logic can be basically done by employing prefix operators such as MUX, the AND-OR gates etc. Each type of prefix operators design provides different prospects of their implementations. Parallel prefix adders serve with excellent performance in binary additions carried out in VLSI.

Parallel prefix adders use simple logic cells to maintain various tradeoffs like fan out, quantity of cells used and the amount of logic levels implemented, and also balance the connection within logic cells. This is the most widely used adder as it performs the functions in parallel thus reducing the delays. In adders, with an increase in the width of the gates, the delay is increased, thus requiring more time to propagate or generate a carry from MSB to LSB. Hence, parallel prefix adders modularise large logic gates into fast and smaller adder implementations.

Ling in 1981, proposed a new carry recurrence to reduce the logical depth for carry computation in carry look-ahead structures. The algorithm introduced by Ling factored out the ‘not kill’ bit from the Generate function which reduced the logical depth. It further reduced the complexity only at the first level whereas the complexity of all other levels remained same. The ‘not kill’ term was produced with the generate term only at the end level which reduced delay.

In High Speed Binary Addition 2004, Jackson and Talwar focused on constructing the carry based on Ling’s approach. They investigated the families of adders that can be derived from the reduced generate and hyper propagate functions which are a generalization of the Ling’s pseudo carry. Their successful approach was not only to take out the ‘not kill’ bit from the last stage but from all other stages as well.

Priti Patil in 2008 showed that Ling factorisation can be applied to all the stages in carry computation while constructing an adder. This factorisation reduces the complexity of the critical path thus reducing the delay and making it faster. However the adder designed is not smaller and I situate my research in this critical gap.

Adder is a topic of significant importance and since ages substantial research has been performed on it. However, enough scope for innovating novel concepts in designing of an adder still exist. The undertaken project involves comparatively less amount of logic levels and thereby reducing the delay in adder implementation, hence providing an adder which is fast in speed and smaller in area as compared to the conventional models. The obtained designs are implemented with Sparse carry tree logic which reduces the amount of logic cells in a circuit. Sparse tree is a novel method. Hence its usage needs to be justified in adder implementation methods. This project provides a base for the justification of Sparse carry tree by proving its verification for the usage in designing an adder.

Speed of an adder is an important parameter for estimating the delay in a circuit. The technique used in this project for evaluation of delay in critical path is **Logical effort.** Logical effort delay model is a design methodology for estimating the total number of CMOS stages required to implement logic function and to calculate delay. Logical method is the simplest and the most accurate VLSI area model for prefix adders implemented in CMOS. It is a measure of the worst performance for the output current of any logic gate in comparison to an inverter. This comparison in gates is viable only when same capacitance is applied across their input. Logical effort can also be defined as the slope achieved by dividing delay v/s fan-out of logic gate to the inverter.

To justify that the novel adder is smaller, the total number of gates in a complete design of novel parallel prefix adder are individually compared with the total number of gates in latest conventional 16-bit Ladner-Fischer Parallel prefix adder. The length of a gate is equal to the width of these Polysilicon gates and the area covering these gates vertically is known as polysilicon grids. So the count of the polysilicon grid defines the size of a gate. For comparison, the logic gates in a design are multiplied with their polysilicon grids present in a logic cell. A tabulated comparison of the total number of gate size in a design reflects the overall size of an adder which is further compared with the total count of polysilicon grids (size) of Conventional 16-bit Ladner-Fischer Parallel prefix adder.

The thesis is categorised in many chapters. Chapter 2 in this thesis discusses the background of the entire project. Initialising from the concepts of adder, equations and notations used in parallel prefix adder till conventional adder used. This chapter also talks about all the gates used in VLSI, techniques used for evaluating size and speed of the designs made. Ling making fast adders, the history supporting his achievement, various papers published regarding Ling's theory, holds a strong base for this chapter.

Chapter 3 is shows the work done for this project. It starts with the research methodology given for Jackson and Talwar's approach which is itself derived from Jackson and Talwar Published paper, reason for doing was this to give a more grounded explanation of what theory is all about. Further chapter 3 depicts all novel designs for 16- bit J & T adders followed by the calculations done for evaluating their speed and size.

Chapter 4 tabulates the result and gives explanation of experimented results.

Chapter 5 is doing the critical evaluation of the work done by explaining the choices made, instead what other choices could have been opted , were the choices for made in this ongoing project, was worth . Chapter does not only appraise the choices made but it also explains the reasons of not taking other options even if they are considered.

Chapter 6 speaks about the conclusions observed after gaining results. Also it suggests the future work that can conclude this project on a better platform.

1.1 Aim:

The primary aim of the project was to design a novel adder which is Faster and Smaller by implementing R. Jackson and Sunil Talwar approach presented in a paper published "High speed binary addition" (R.C.Jackson and S. Talwar, Nov 2003). Secondly the project seeks to provide new avenues for researching and experimentation in Jackson and Talwar's approach, providing new insights to designing adders in VLSI using CMOS methodology and comparing it for better results than Ling's theory (H.Ling, 1981). Finally the project proposes a new design methodology of an adder in CMOS.

1.2 Objectives:

1. Understanding the basic equations for parallel prefix adder, theory of Ling factorisation and Jackson and Talwar's approach:
   The initial task was to understand the concepts of parallel prefix adder in order to correlate the terms with the notations used in Ling's theory. Also in-depth study of Ling's theory of factorisation was aimed which provides a base for Jackson and Talwar theory. The understanding and forming a base for Jackson and Talwar's approach was difficult as the paper High speed binary addition do not materialise any good knowledge about this approach.

2. Designing an adder using Jackson and Talwar's approach:
   Various configurations of parallel prefix adders (Knowles, April 1999) are considered for designing adders. The more complicated Jackson and Talwar's approach is the more flexibility it provides for designing. Schematics of adder designs are given for low valency. Manipulation of equations is to be done for using only valency -2 and valency -3 input gates.

1. Proving that Jackson and Talwar's approach is more beneficial to use than Ling's theory. Different critical paths in a design are considered and then logical effort delay model (Sutherland,Sproull, Harris, 1999) is applied on different path for calculating delay.

2. Using different novel methods like Sparse carry tree (Harris, 2010) in the adders.
   Merging Sparse carry tree with Jackson and Talwar's approach for reducing the area of an adder. Implementation of above mentioned adder with Sparse carry tree and without Sparse carry tree is done then tabulated results are compared.

2 Background:

2.1 Concept of Adders:

Addition of two numbers is the fundamental operation underlying digital microelectronic circuits. It is desirable to have an adder with good performance and area trade-off characteristics. Adder should be designed in a way that it computes results fast also it should be small to occupy less chip area. An adder is used to add to binary numbers in digital devices. Generally, all adders work by considering 2's complement. Adders are desired to give good performance in efficiency , speed and small chip area as these are the most required parameters ,so a mutual understanding technology along with the algorithm is needed for its implementation Prefix formulations of adders gives transparent way of using, thus the adders following these formulations can be efficient and covers less chip area.

Basically while designing adders we are concerned with Sum and Carry logic. The sum logic is calculated by considering the carry from every single bit from LSB to MSB .Similarly for carry logic, best approach is, implementing carry from the least significant bit in order to propagate it to all other bits as the most significant bit allows maximum carry sharing. The mentioned approach is known as Layered approach (Neil H.E Weste and David Harris, 2005) (Y.Wang and K.K Parhi, 2010) in which the subgroups of carries i.e excluding one carry generated from most significant bit allows maximum carry sharing in at different bit positions. In this process the last bit get becomes lot more complicated in the carry incorporation path. All the logic discussed above is implemented using NAND and NOR's gates in VLSI technology as it saves more logic gates and thereby reducing the delay of the circuit.

An adder should be constructed with less number for logic cells in order to reduce the delay and the capacitance in the circuit .The properties like delay, capacitance, fan-in and fan-out are used to check the performance in an adder. A full adder cell can be implemented in many different logic structures .The properties like fan-in and fan–out completely depend on the type of implementation technology or he logic structure used for designing the basic operator and the type of prefix operator in an adder. Also, the combination of the amount of internal wiring and fan-out of intermediate nodes can give an incredible output of small and fast adders.

An adder (Burgess N. , 2009) is used to add to binary numbers in digital devices. Based on different algorithms and methodology various adder designs are build for achieving different targets of high speed or less area consumption. Adders are basically divided into two types (Neil H.E Weste and David Harris, 2005), depending upon the number of inputs and the outputs we have:

1) Half Adder
2) Full adder

1) **Half Adder:** This is suitable for single bit addition. For single bit addition, an adder has two inputs(a and b) and the output obtained is the sum of two bits, in terms of Sum and Carry. In a half adder, addition of two bits is done without any carry in implementation.

The truth table is used for better explanation. In this the carry propagates from one column to other column on moving from LSB to MSB.

Inputs		Outputs	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Table 1 : truth table for Half adder

Half adder is the basic design of any functional digital logic design which is formed by two logic gates

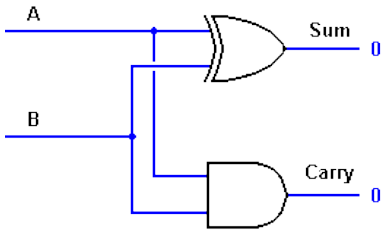


Figure 2 : logic gate circuit for Half adder

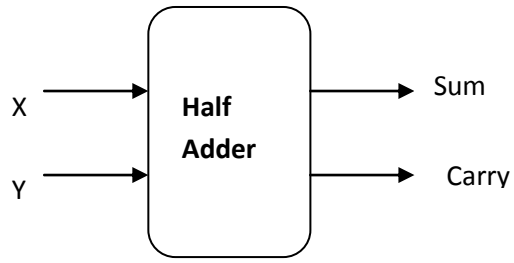


Figure 1:Block diagram for Half adder.

The sum is simply an XOR and the carry is an OR gate so we use two gates to implement this half adder logic:

- $S = X \oplus Y$
- $C_{out} = A \cdot B$

2) **Full adder:** It is a logic circuit implementing three bit binary number. Full adder is designed by joining the two half adders in a way that the sum (output) of one half bit adder is given as input to the other half adder. In full adder there are two carries so we take one as C_{IN} and the other as C_{OUT} and we also have the third input as S to designate the final two bit number output as Sum and carry-in.

The notations used for output in Full adder are usually expressed in terms of Generate, Propagate and kill.

- Kill = when both the processes are zeroes.
- Propagate = when the C_{in} is a C_{out} .
- Generate = when both the functions are 0 or 1.

The carry propagates to others with the MSB generating a carry and all other carries are sub groups .The Sum and Carry are expressed as:

- $S = XY'Z' + X'YZ' + X'Y'Z + XYZ$
  $S = (X \oplus Y) \oplus Z$
- $C_{out} = XY + XZ + YZ$

$= XY + Z(X+ Z)$
$= (X'Y' + Z'(X'+Y'))'$
$= maj (X,Y,Z)$

The resulting truth table is shown:

A	B	C_{in}	C_{out}	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1

Table 2: Truth table for Full adder

We can implement the logic for Sum (S) by using 3 – bit XOR gate made up of 3- bit majority function.

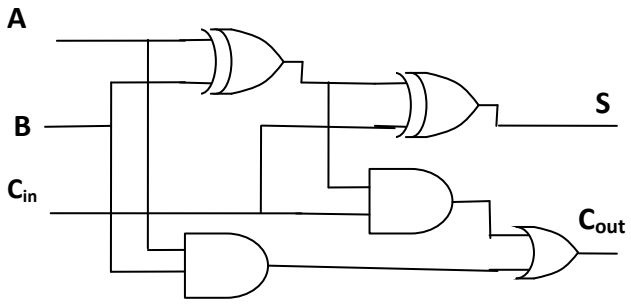


Figure 3: Logic circuit for Full adder

In order to perform multi bit addition the total number of bit to be added should be equal to the total number of adder to get the output with proper carry and sum bit logic. In the above circuit we are using three levels of logic so the carry out delay will be equal to 3 gate delays. The carry gate uses factorization and is called a majority gate.

2.2 Different type of N- bit adders:

The simplest type of adder used for the additon of two bits or the simplest method used for designig a full adder is Ripple adder (R.E. Ladner and M.J. Fischer, 1980).Other adders types are Ladner- Fischer , kogge-stone, brent-kung ,knowles, Han carlson . These above mentioned adders are based on he theory of parallel prefix adder. With the research carried out in the paper of Family of adders (Knowles S. , April 1999), Ladner –Fischer (R.E. Ladner and M.J. Fischer, 1980)and Kogge – stone (P.M. Kogge and H.S. Stone, 1973) methods are considered the best option. Priti's thesis (Patil, 2007) examines Ladner – Fischer against Kogge – stone where Ladner –Fischer came out to be faster in terms of delay and fan-out than Kogge – stone. This being the reason , Ladner –Fischer is taken as the base adder for undertaken project work. Brief explanation of different relevant types of adder is given .

2.2.1 Ripple Adder:

The Ripple carry adder (Lin, 2003)is the simplest method for adding two binary digits but it is slow in performance as the carry bit transferred has to cover all the logic levels for incorporating the value of sum . In Ripple adder the sum is computed only after carry from the previous bit is transferred to next bit for which the sum has to wait inducing the delay in the circuit, sum is computed after it receives all the three input . As discussed full adder has one input as the carry bit of the previous bit which slows down the process for calculating sum. The carry bit C_{out} from generated of 'i' bit position serves as an input C_{in}

to the 'i+1' bit .These type of adders in which the carry bit ripples from one bit to other is known as ripple carry adder .Ripple carry adders generates only one bit at a time. If only the carry bit from the 'i' bit is generated it will be the input to the 'i+1' bit and then that bit will generate carry , this whole process of waiting for the carry introduces propagation and generation delay.

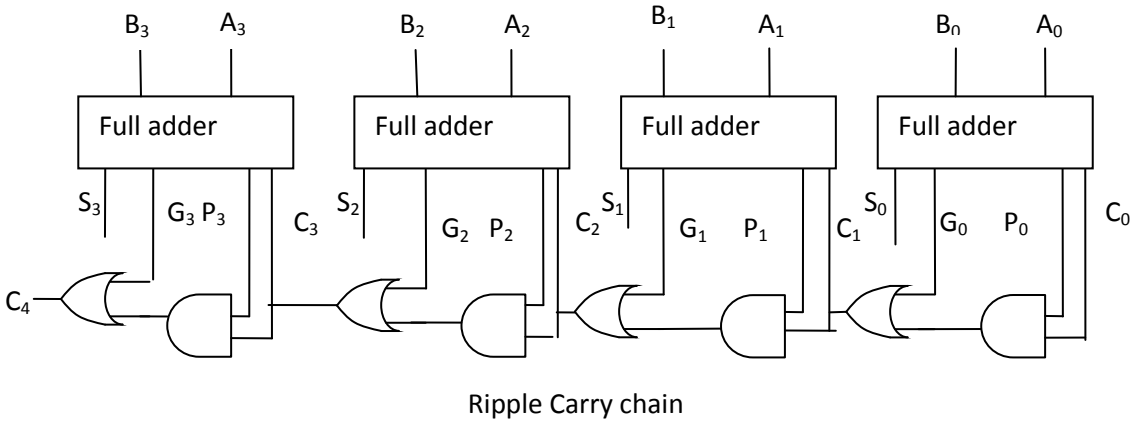


Figure 4 shows 4 bit Ripple carry adder (google webpage, 2010)

As shown in the above diagram the output carry from the previous bit is given as the input carry to the next bit and this process of rippling the carry takes long time which introduces delay in the circuit and making the method slow.

2.2.2 Adders designed on the basis of generated carry:

Variety of adders which can be used as the base model for designing an adder such as carry skip adder, carry look ahead adder, parallel prefix adder etc but generally the adder designs are dependent on the carry groups that is generated from the previous bits and then the logic obtained for the carry group is implemented using different NOR and NAND gates to suffice the carry out for the recent bit. Depending on whether the carry is generated, not kill or propagated the application of the gates is done. In these kind of adders, with the increasing bits, the logic becomes more and more complicated which is intense to realise with the simple gates. These kinds of adders are tough to realise reason being that for every single bit there is some logic applied. Sometimes in the concern of realising the logic, makes its application difficult. These kinds of adders have various CMOS gates with different valency. Depending on carry if it is propagated or generated or kill here are some examples of the carry expressions:

$C_5 = G_{4:0} = g_4 + p_4g_3 + p_{4:3}\, g_2 + p_{4:2}\, g_1 + p_{4:1}g_0 + Cin\, p_{4:0}$

Similarly for other bits:

$C_4 = G_{3:0} = g_3 + \sim k_3g_2 + \sim k_{3:2}\, g_1 + \sim k_{3:1}\, g_0 + C_{in} \sim k_{3:0}$

$C_3 = G_{2:0} = g_2 + \sim k_2g_1 + \sim k_{2:1}\, g_0 + C_{in} \sim k_{2:0}$

$C_2 = G_{1:0} = g_1 + \sim k_1g_0 + C_{in} \sim k_{1:0}$

$C_1 = G_{0:0} = g_0 + C_{in} \sim k_0$

$C_0 = C_{in}$.

The Generalisation (A. B. Smitth and C.C.Lim, 2001) of the above terms is given as :

$$\mathbf{C_i = g_{i-1}} + \sum_{k=0}^{i-2} (\mathbf{gk} \,.\, (\sum_{f=k+1}^{i-1} \mathbf{pf})') \qquad \textbf{Equation 1: summation of carry terms}$$

The not kill term (~k) is often represented as propagated term(p) as well as the usage of not kill logic can be implemented using a basic NOR gate instead of using the high delay

EXOR gate .As discussed above these terms lot more complicated towards MSB this being the these higher complicated terms are subdivided into smaller groups which compute the terms based on the basic 'X term' and 'Y term ' . The realisation of the basic terms makes it simple to implement the logic in the higher bits. The carry obtained above are used to constructing the sum bit in every single bit as the carry propagates, so while designing this kind of adders it is very important to combine every single carry bit from LSB to MSB.The expressions to evaluate the value of Group generate, group propagate and not kill term is calculated from the truth table (Patil, 2007) given below:

X	**Y**	**C_{in}**	**C_{out}**
0	**0**	**X**	**0**
0	**1**	**C**	**C**
1	**1**	**C**	**C**
1	**1**	**0**	**1**

Table 3: truth table for carry chain

- **Kill :** $X=Y=C_{out}=0$
- $K_i = X_i \text{ ' } . Y_i \text{ ' }$ **Equation 2: kill terms**

- **Propagate :** $X=0$;$Y=1$; $C_{in} = C_{out} = C$
- Propagate : $X=1$;$Y=0$; $C_{in} = C_{out} = C$
- $P_i = X_i \oplus Y_i$ **Equation 3:propagate terms**

- **Generate** : $X=1$;$Y=1$; C_{in} = don't care; $C_{out} = 1$
- $\mathbf{G_i = X_i . Y_i}$ **Equation 4 : generate terms**

The calculation for carry bit generated and carry bit propagated terms in a 4 –bit adder should be considered in this way:

$C_3 = G_{2:0} = g_2 + g_1p_2 + g_0p_1p_2 + C_{in}\, p_0p_1p_2$

Then $C_2 = g_1 + g_0p_1 + C_{in}\, p_0p_1$

$C_1 = g_0 + C_{in}\, p_0$

$C_0 = C_{in}$

Where g_0 = generate at bit position 0 and similarly we have g_1, g_2 generate at bit position 1 and 2. p_2, p_1, p_0 = propagate at bit position 2, 1 and 0.

These kind of adder constructed are often faster because every bit have a different logic and the logic is mathematically supported to construct the simplest logic for every single bit .In these kind of adder the second stage computes the carry which forms a carry tree as it takes sequentially the carry from all other bits from LSB and thus forming a carry tree.

2.3 Carry Look-Ahead Adder:

The CLA (Yu-Ting Pai and Yu-Kumg Chen, 2004) (Y.Wang and K.K Parhi, 2010) (google webpage)is a complex method then Ripple carry adder. It is widely used because of its high performance and over exceeding factor of reducing the time span by generating and propagating the carry simultaneously unlike Ripple carry adder. It increases the speed by reducing the amount of time required in generating carry groups from most significant bit to the least significant bit. The procedure includes generation and propagation of carry alongside of the calculation of sum bit. Carry Look Ahead adder does not wait for previous bit to generate carry and then after generate, propagate carry in the next cycle rather it performs both the function for carry together so that the propagation and generation delay is reduced.

CLA is widely used in designs because of its parallel processing of carries for n-bit additions of high valency cells resulting in less delay. It sends the divided input in groups of k–bit and evaluates different signals for generation or propagation of a carry groups . It derives different signals from different groups ,at same time, to increase the speed of an adder by $\log_k n$ gates .

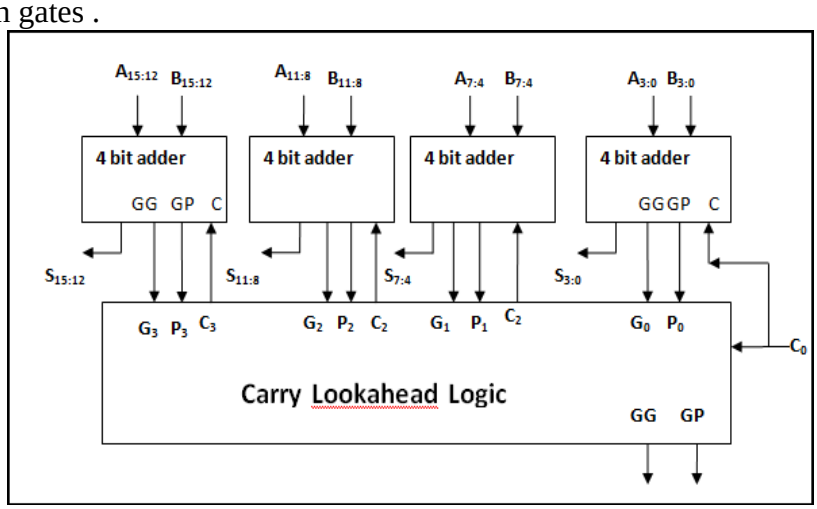


Figure 5:4- bit Carry lookahead adder

The diagram shown above is of 16 bit adder .In this, there are groups of 4 bit taken together. The carry is generated from every logic cell and propagated to the next cell. Group generate (GG) and Group propagate (GP) are obtained parallel giving a faster adder.

In carry lookahead adder the critical path travels in vertical direction instead of travelling in horizontal direction. The fan-out is only dependent on the total number of logic levels coming across the critical path instead of considering the delay for every logic gate in the design. In Carry lookahead fan in and fan out depends maximally on the critical path with the maximum amount of gates. The depth in a carry look ahead adder resembles the size of different groups in it and this being the reason the delay is almost equal to the log function of total size of an adder. The log function for depth makes it faster than any other design. Carry lookahead adder are widely used for designing.

2.4 Parallel Prefix Adders:

Adders play very important role in VLSI implementation and this being the reason they should be fast and small with very high efficiency. Parallel prefix adders (A. B. Smitth and C.C.Lim, 2001) serve with excellent performance in binary additions implemented in VLSI. These Parallel prefix adders (Choi, Y. and Earl E swartzlander Jr.) (Knowles S. ,

1999) use simple logic cells to maintain various trade-offs like fan out, quantity of cells used and the amount of logic levels implemented and balance connection within logic cells. Also parallel prefix adder presents the general adder structure that exhibits flexible area trade offs for an adder design.

In adders, with an increase in the width of the gates the delay is even increased, thus requires more time propagate or generate a carry from MSB to LSB so parallel prefix adders modularises large logic gates into fast and smaller adder implementations .
Also as discussed above we know that prefix adders depend a lot on the valency and if the valency is increased it will shoot up the number of CMOS cells used in the design which will further increase the delay in the critical path even if the total number of logic levels is decrease. The addition uses the terms (H.Ling, 1981) like kill (k_i), propagate (p_i) and generate (g_i) functions for individual bit positions referenced as 'i= 1,2.....0' and for the adder width as ' w'.

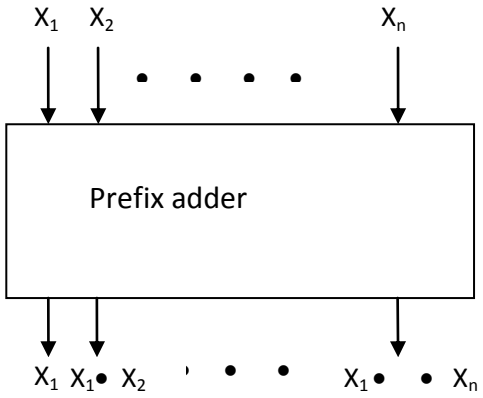


Figure 6: Block diagram Prefix circuit

Stages for Designing a Parallel Prefix Adder:
The designing of parallel prefix adders depends on the three stages describe below:

- At the first stage, two bit binary number is supplied as inputs to compute generate, kill and propagate functions. These functions are formulated using general CMOS gates like NAND, NOR e.t.c.

- The second stage calculates the computational logic for carries from each bit position and is known as Carry logic as in eq. 4 .In this stage the calculated carry from each bit is combined recursively with the other bits to finally produce either of generate, kill or propagate functions. PG logic stage varies depending on configurations. Starting from MSB to LSB, generate and propagate or not kill is computed over each group of bits and are merged until the logic for each bit is traced. Two types of cells namely the grey cells and the black cells are used to implement the logic .This kind of prefix implementation adder is suitable for high valencies. Total number of stages depends upon the logic of $\log_2 n$ where n is the valency of an adder.

- The last stage is for the totalling the sum from every bit position as in equation 5. The use EXOR gate simplifies the logic in this stage. The important point in this stage is that the output carry of the previous position is supplied as an input to the next bit position. The diagram below gives an idea for designing basic Parallel prefix adder in three stages.

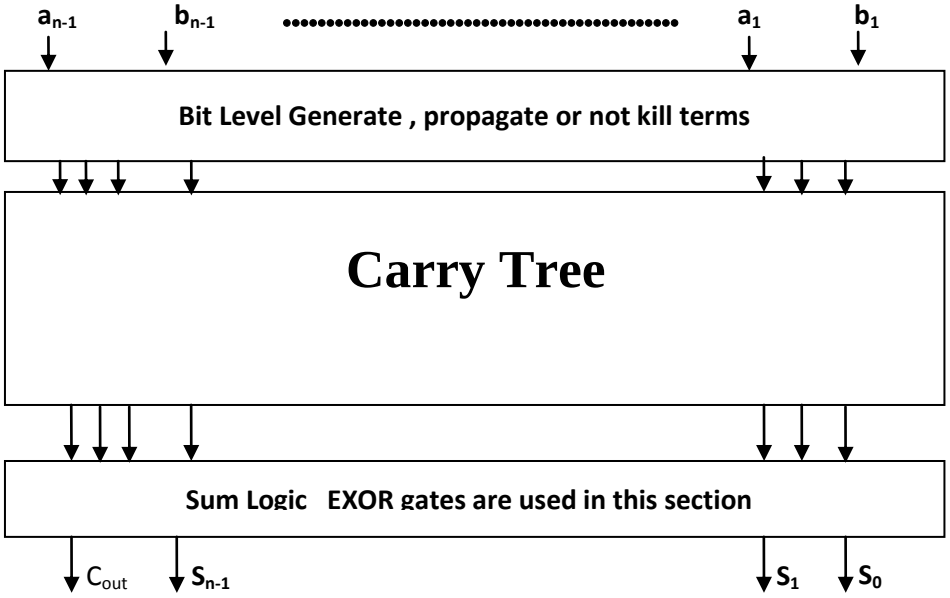


Figure 7: Block Diagram of Parallel prefix adder.

For calculating all the above terms, here are some equations used while designing:
Let the two inputs for the first stage be $a = a_{n-1}\ a_{n-2}\ a_{n-3}\ldots\ldots\ a_1\ a_0$ and $b = b_{n-1}\ b_{n-2}\ b_{n-3}\ \ldots\ldots\ b_1\ b_0$ going from LSB to MSB.

The first stage computes Generate (g). Propagate (p) and not kill (~k) terms with the formulas given below. This is better explained in the above section.

- $g_i = a_i \,.\, b_i$
- $\sim k_i = {}_{\sim}a_i \bullet \sim b_i$
  $k_i = a_i + b_i$
- $p_i = a_i \oplus b_i$

Second stage computes carry for output functions (g,p,~k) obtained from first stage .
Giving the Carry group. The generation and propagation of carry bits is with the formula given below:

$C_{i+1} = g_i + \sim k_i C_i$ Equation 5

The last stage is for computation of Sum which employs EXOR gate for calculating the values at every bit position:

$S_i = p_i \oplus c_i$ Equation 6

The carry function can be repeated following the **recurrence** in equation 5 using the terms k_i , g_i and C_o :

$C_{i+1} = g_i + (\Sigma_{j=0} (\Pi_{k=j}^{1} K_k^{-}) g_J + (\Pi_{k=0}^{1} K_k^{-}) C_0$ Equation 7

Properties of Parallel Prefix Adders:

The Parallel prefix adders depend on generate and not kill carry functions. An ordered pair (nk and g) is defined with '•' Operator. (A. B. Smitth and C.C.Lim, 2001) .

So,$(nk_i.\ g_j) \bullet (nk_j,\ g_j) = (\ nk_i\, nk_j\ ,\ g_i + nk_j\, g_i\ )$

Three important properties of '•' operator are :

1) **Associativity:**

$gk_n^m = (gk_n^k \bullet gk_{k-1}^j) \bullet gk_{j-1}^m$

Associativity allows the carry parallel application of above carry equation.

2) **Idempotency:**

For the values $m \leq k < j \leq n$ or $m < k = j < n$

then $gk_n^m = gk_n^k \bullet gk_j^m$

The overlapping if the terms give a regular function. It permits the sub trees of adders to work in parallelisation.

The first and the third stage of a prefix adder will be fast as the carry calculation is dependent on individual cell logic where as the carry in the second stage covers comparatively longer distance which in return introduces delay .The different valency prefix operator are :

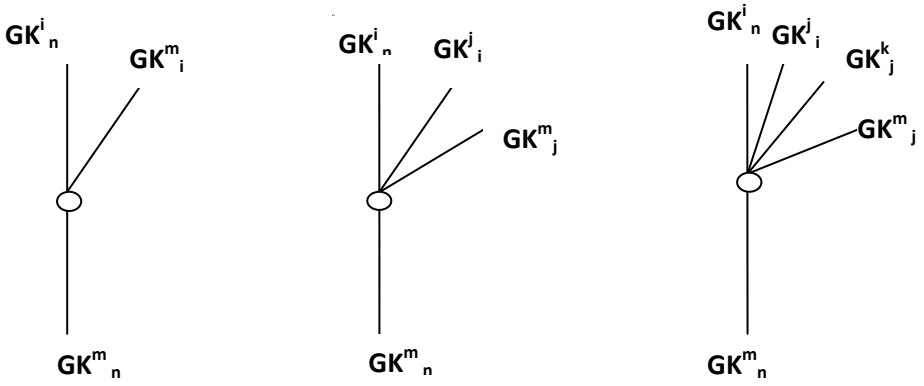


Figure 8: prefix cells with different valency

Black and Grey cells:

While designing an adder we basically deal with almost same logic for every different bit. The logic we deal is of grey cell and black cell. As shown in the diagram of prefix cell we have grey and black cell represented as :

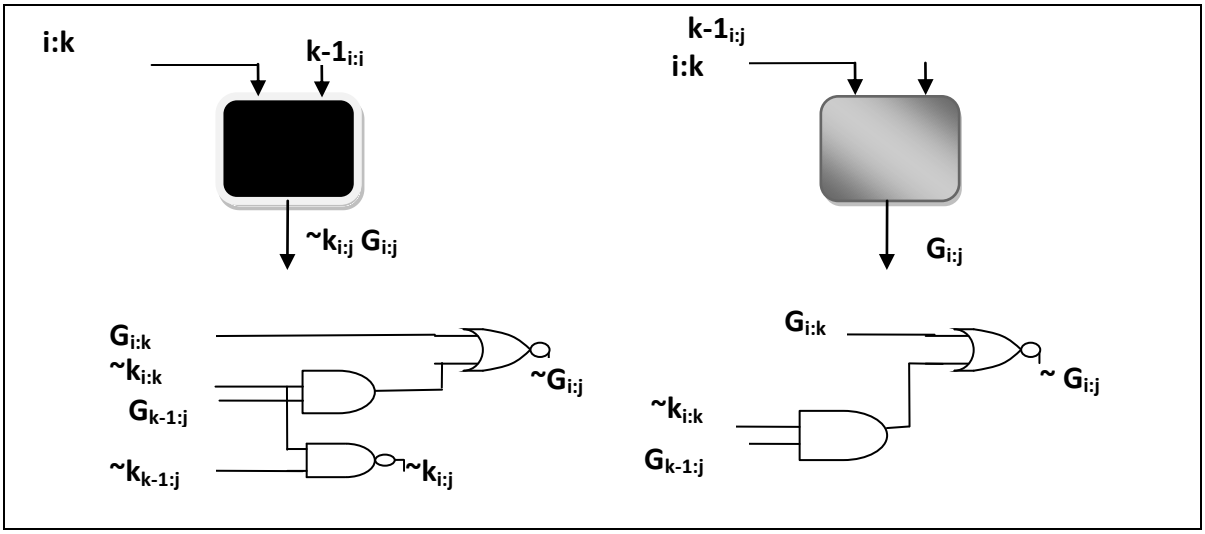


Figure 9: Black and Grey Prefix cells

2.5 Types of Parallel Prefix adders:

Variety of Parallel prefix adders are presented, a tough base for these adders is provided based on various trade –offs like delay, area and capacitance. Here two types of parallel prefix adders are discussed as published paper "Family of adders" discusses different kind of parallel prefix adder but Knowles have focussed mainly on Ladner and Fischer adder and Kogge-Stone adder. (Knowles S. , April 1999).

2.5.1 Kogge-Stone:

Kogge and Stone (P.M. Kogge and H.S. Stone, 1973) is another type of parallel prefix adder which can be used for designing adders. Kogge and Stone proposed a general recurrence scheme for parallel computation in 1973. Adders implemented using this technique is in favour due to the following:

- Regular layout
- Controlled fan-out

In this implementation of parallel prefix adder the logical depth (number of carry computation stages) is maintained same which is equal to log2n. It generates the carry signals in O(log n) time, and is widely considered the fastest adder design possible. It is the common design for high-performance adders in industry.

The lateral fan-out of each node in every stage is limited to unity by using the Idempotency property of prefix adder but this increases the number of gates by large amount and also corresponding increase in the amount of wiring due to the large number of overlapping of prefix terms being pre-computed, resulting in larger area. This type of adder also needs some buffering inverters to drive the capacitive load resulting due to a good amount of span of wires. The binary trees of "BK" cells in the Kogge and Stone adders are not sharing. Consequently the signal fan-out is reduced to the minimum at the expense of more "BK" cells. Since the delay increases with the fan-out, it is now a bit short. For this type , fan-out decreases with increasing performance.

However, they are nothing but prefixed carry-lookahead adders. The intermediate carries is generated by replicating the prefix tree. Following figure shows the resultant prefix graph of a 16-bit prefix-2 Kogge Stone adder.

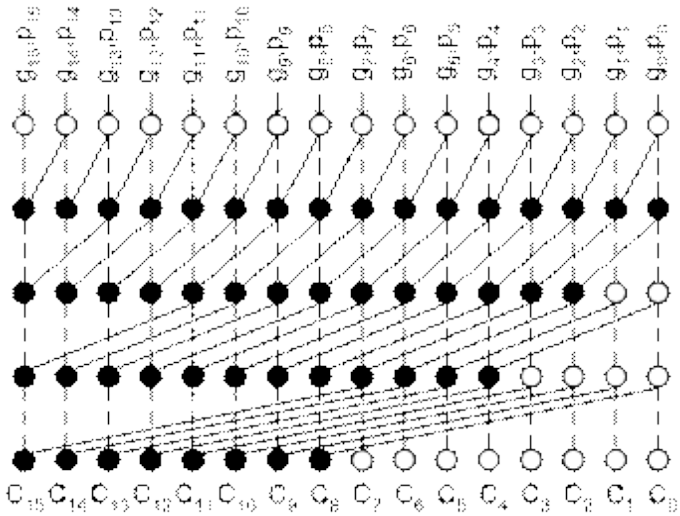


Figure 10: Prefix graph of 16-bit prefix-2 Kogge Stone adder (P.M. Kogge and H.S. Stone, 1973)

2.5.2 Ladner and Fischer:

The Ladner –Fischer adder is used as a base adder in this project. As mentioned by Ladner –Fischer (R.E. Ladner and M.J. Fischer, 1980), this type utilises the model of double recursion. It minimises the number of logical cells and is calculated as (log2 n -1), n being the number of stages. It therefore eliminates Associativity present in prefix adders. In Ladner –Fischer adders the capacitive load and the fan-out of the circuit amplifies due to large separation between the wires and the cells. The product circuit is obtained using double recursion and the buffering is done using the high load inverting amplifier.

Longer distance between the wire increases more hardware time required thus increasing the fan-out. Similarly the computational time increases with increase in the depth of the circuit,

So less the number of logics, decreased will be the delay . The circuit with the exact depth of log2n and the minimum possible length are detected by applying recursion. (log2 n -1) give the total number stages required in computation.

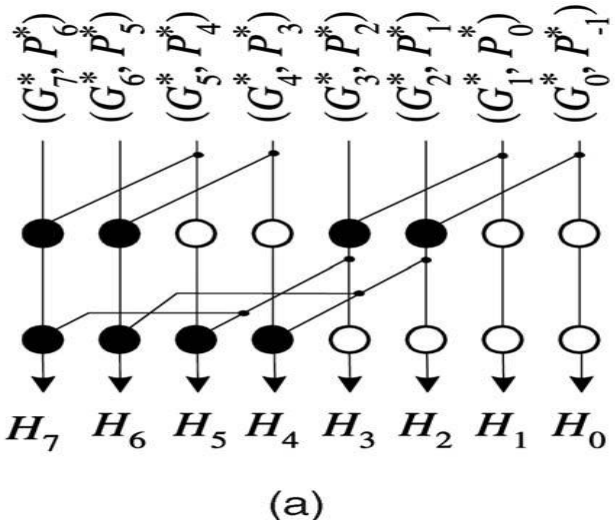


Figure 11:Prefix graph of Ladner - Fischer adder (R.E. Ladner and M.J. Fischer, 1980)

In the Diagram the solid dots are representing the nodes, thus implementing prefix operation .The diagram shows only the carry bit generation and not sum. Depending on the position black and grey cells are used. Different gates are used to evaluate the value for G, k and p in the first row while the second gives the carry for every bit position.

2.6 Ling Adders:

Ling in 1981, proposed a new carry recurrence to reduce the logical depth for carry computation in parallel prefix adder structures. The algorithm introduced by Ling factored out the not kill bit from the Generate function (Vazquez, A.; Antelo, E., 2008) which reduced the logical depth and further reduced the complexity only at the first level where as the complexity of all other levels were same. The not kill term was producted with the generate term only at the end level which reduced delay. (H.Ling, 1981)

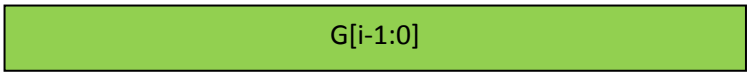
Let a and b denote two n bit numbers:

$a=a_{n-1}\ a_{n-2}\ a_{n-3}\ldots\ldots\ a_1\ a_0$

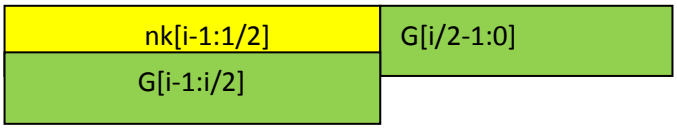
$b=b_{n-1}\ b_{n-2}\ b_{n-3}\ \ldots\ldots\ \ b_1\ b_0$

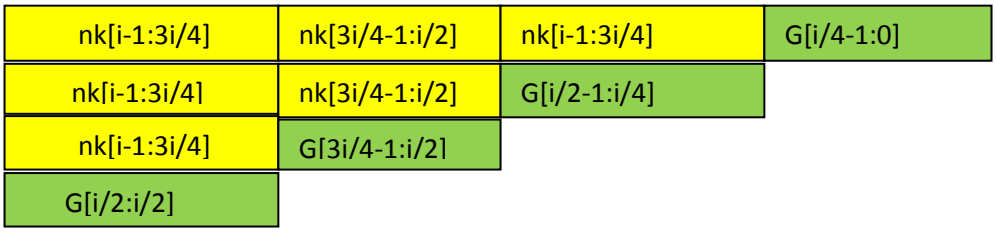
Non-Ling Addition (Burgess N. , Implementation of recursive Ling adders in CMOS VLSI, 2009)**: This is the addition done with the implementation of Ling's theory.**

• $G_{i-1:0} = g_{i-1} + nk_{i-1}.g_{i-2} + nk_{i-1}.nk_{i-2}.g_{i-3} + nk_{i-1}.nk_{i-2}.nk_{i-3}.g_{i-4} + ...$



• $G_{i-1:0} = G_{i-1:i/2} + nK_{i-1:i/2}.G_{i/2-1:0}$; $nK_{i-1:0} = nK_{i-1:i/2}.nK_{i/2-1:0}$





Background:

As previously discussed the generate and not kill term are given with these equations:

- $g_i = a_i.b_i$
- $nk_i = a_i + b_i$

The General generate function for adder is :

[$G_{n:0} = g_n + nk_n g_{n-1} + nk_{n-1} g_{n-2} + nk_{n-1} nk_{n-2}\ g_{n-3:7}$] **Equation 8: General non - Ling equation for adder**

where:

- $g_{n=}\ a_n\ AND\ b_n$
- $k_{n=}\ a_n\ Or\ b_n$.

The functions used in adder are purposely known as generate, kill and propagate functions. The obtained functions are carry generate and carry propagate, and carry generate is more intricate then carry propagate. On further solving the equations the delay can be reduced by decreasing the delay in most significant bit of the sum.

Equations are *:*

1. **Generate function:**

1.$H_{n:0} = g_n + g_{n-1} + nk_{n-1}\ g_{n-2} + nk_{n-1}\ nk_{n-2}\ g_{n-3:7}$ Equation 9:Ling generate function

In this equation Ling factored out the nk term (not kill bit) resulting on in the generate bit for all term and not kill bit for all stages except for the first stage Here the term $\boldsymbol{H_{n:0}}$ is Ling's pseudo carry term.

The generalised form of the above equation is :

$\boldsymbol{H_{n:0} = g_{n:k} + k_{n:k}\ g_{k-1:0}}$. **Equation 10: generalised form of equation 9**

With the help of factorisation, Ling proved that Ling's pseudo carry term with generate group is equal to the original generate term obtained i.e. the factoring out of the not kill bit is sufficed by again multiplying the not kill bit with generate group in later stages, which reduces the use of an AND gate at every stage. The notations are shown below:

$G_{n:0} = nk_n \cdot H_{n:0}$

$\boldsymbol{g_{n:0} = nk_n \cdot g_{n:0}}$ **Equation 11: Ling's consideration of $G_{n:0}$**

2. **Kill function:** The kill group can be considered over a wide range from n to k and k-1 to 0. The equation is given below:

$k_{n:0} = k_{n:k\ to\ k_{k-1:0}}$.

Proving equation 10:

From equation 9 we have :

$H_{n:0} = g_n + g_{n-1} + nk_{n-1}\ g_{n-2} + nk_{n-1}\ nk_{n-2}\ g_{n-3} + nk_{n-1}\ nk_{n-2}\ nk_{n-3}\ g_{n-4} + \ldots\ldots\ldots$

Assuming the equation for valency of 7 we have:

$H_{7:0} = g_7 + g_6 + nk_6\ g_5 + nk_6\ nk_5\ g_4 + nk_6\ nk_5\ nk_4\ g_3 + \ldots\ldots\ldots$ Equation 12

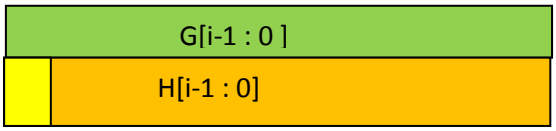
Using equation 10 in 12:

$H_{7:0} = g_7 + g_6 + nk_6\ nk_5\ (g_5 + g_4) + nk_{6:3}\ (g_3 + g_2) + \ldots\ldots\ldots$ Equation 13

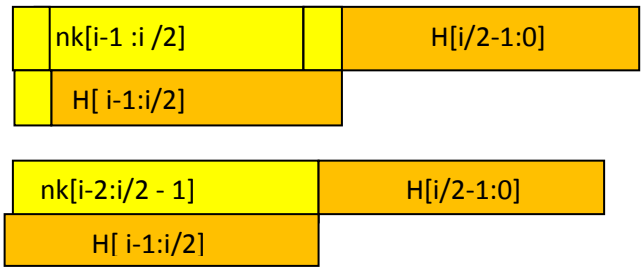
NOTE: Ling's pseudo carry saves 1 logic level in comparison to Generalised Generate function. Whereas the implementation of Ling's Pseudo carry $H_{n:0}$ is easy to implement then $G_{n:0}$.These Implementation of equations will be done using CMOS methodologies. Various Gates such as AOI, NAND, NOR will be used.

LING ADDITION: Ling addition is explained on the basis of these blocks where the continuation of linked terms is done in same colour representing different groups .

• $G_{i-1:0} = nk_{i-1}.H_{i-1:0}$



• $H_{i-1:0} = H_{i-1:\ i/2} + nK_{i-2:i/2-1}.H_{i/2-1:0}$



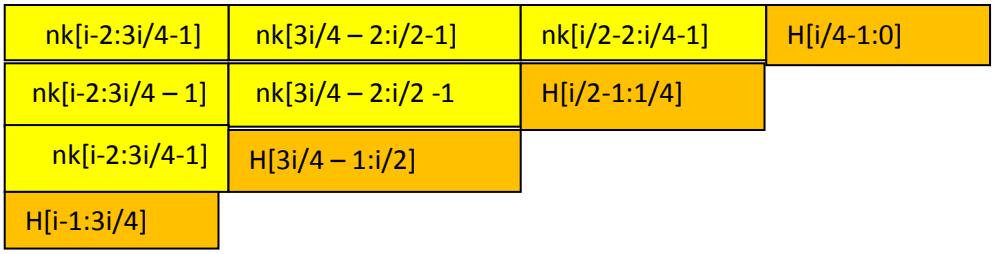
Basis of evaluation of Sum:

The key point is to product the Pseudo carry with not kill bit in order to obtain the actual carry for a particular bit. This is achieved in the last stage of n bit adder and any extra delay is avoided by checking the critical path for a n-bit adder .The critical path can be evaluated from Least significant bit to n-1 bit .The carry from n-1th bit is required in evaluating the sum and the expression for sum is :

$S_n = a_n \oplus \mathbf{b}_n \oplus G_{n-1:0}$ **Equation 14: Sum function for Ling theory**

Recursions in pseudo carry generation:
As Ling adder carries most of the properties of Prefix adder it also adopts the property of combination of pseudo carries over sub functions to generate pseudo carry of group .

• $H_{i-1:i-2} = g_{i-1} + g_{i-2}$; $nK_{i-2:i-3} = nk_i\text{-}2.n$



$H_{n:0} = g_n + G_{n-1:0}$
$H_{n:0} = g_n + G_{n-1:x} + nk_{n-1:x} G_{x-1:y}$
$H_{n:0} = g_n + G_{n-1:x} + nk_{n-1:x} (g_{x-1} + g_{x-2:y})$
$H_{n:0} = H_{n:x} + nk_{n-1:x-1} H_{x-1:y}$
The general generate function and the pseudo carry function are the same in complexities. So the speed of adders can be far more increased if Ling's method becomes applicable to not only the first bit but to all other bit in order to decrease the complexity of generate functions and make them suitable as pseudo generate functions.

2.7 Sparse trees:

It is a method of separating the carry tree into critical and non critical sections in a design. (Sanu Mathew,Mark Anders,Ram K. Krishnamurthy and Shekhar Borkar, MAY 2003)In this carry from every single bit is not generated as in Kogge stone designs instead carry bit is generated from every fourth bit .The main concern is to move some carry logic to non critical path thus reducing the number of propagate and generate bits in critical path. These carries combine normally with the sum logic in the multiplexer giving better results. This implementation of sparse carry tree reduces the fan-outs for generate and propagate bits but the number of logics remain the same in a circuit .Designing of adder is done using sparse – 2 and sparse -4 in this project. In sparse trees the carry prefixes from every n^{th} column is considered despite of other D terms going into the same multiplexers.

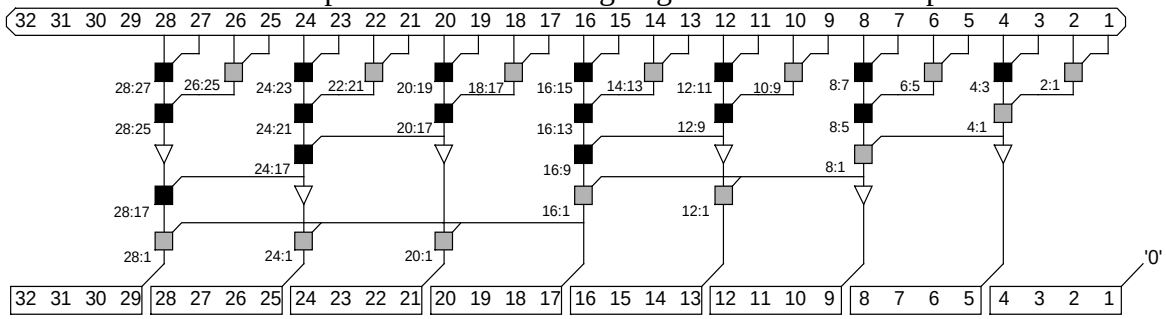


Figure 12: Circuit for Sparse-4 carry tree (Harris, 2010)

Above shown is a 32- bit adder , the carry generated in the 3rd stage is C_4, C_8, C_{12}, C_{16}• • • •C_{28}, C_{32} instead of generating carries as C_0, C_1, C_2, C_3• • • • C_{31}, C_{32} .

2.8 Knowles 16 – bit adder:

Binary addition can be done with different types of adder. The most common types of adder discussed in Knowles paper Ladner – Fischer and Kogge- stone. The paper published (Knowles S. , 1999) reflects that both these type of adders share common

property of minimum logical depth. The structures presented permits trade off between the intermediate nodes and the total amount of wiring in the circuit resulting into a fast and fast adder then adder types of adder . The provided structure shows Knowles 16- bit Ladner – Fischer adder .This adder is a famous adder used commercially. Also we have taken this adder for comparison in our project. The diagram is shown:

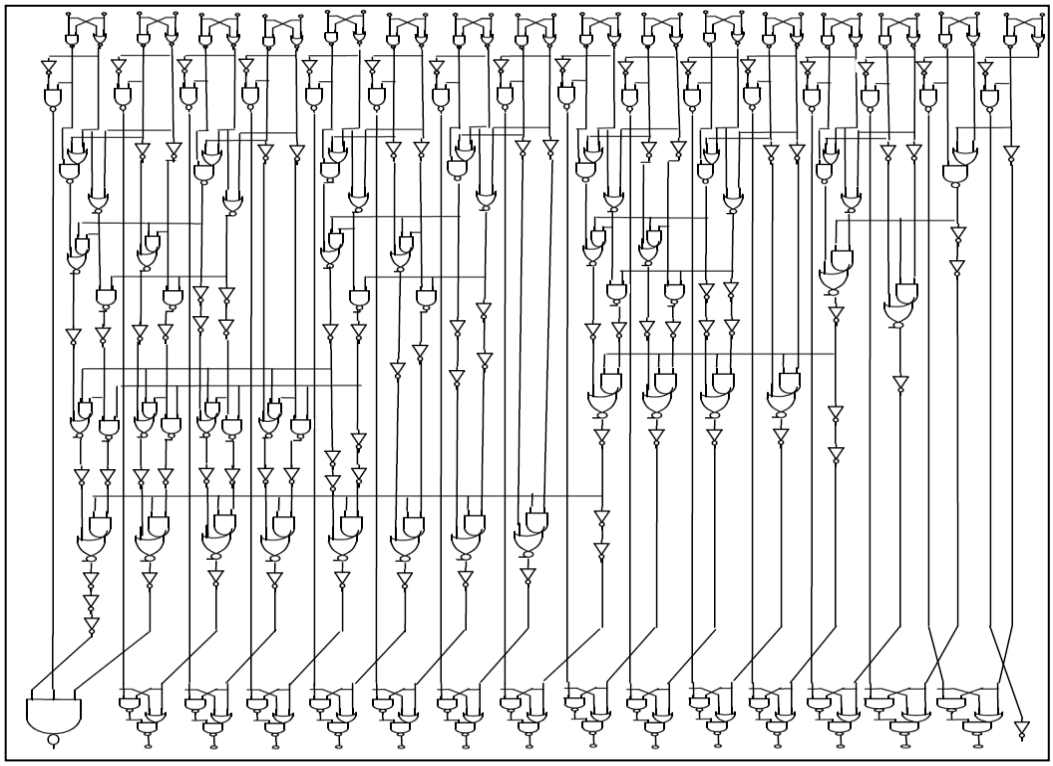


Figure 13:16- bit Ladner - Fischer adder

As per the above diagram is16 bit design configuration i.e. S_0 to S_{15} as it is clearly visible that $R_{14:0}$ is driving S_{15}. The 16 bit design also has three logic levels but more than 7 stages from initial logic till sum logic. Here in this section total count is shown for 16 bit adder design and then it is compared from that of the novel design to prove the size of the novel adder being small.

Note: The techniques used for calculating the **delay and size of** an adder are after this topic, listed in detail.

The table given below shows the total count of the number of gates in every stage of both adders. Starting from the first column, it represents the level of stage i.e. for which stage have we considered the gate type and the other columns shows the different types of gates used. For evaluating the size of a gate consideration of **polygrid** in a gate are very important in a gate. So to make it simpler, different varieties of gates are placed in separate columns.

Stages	Total number of gates in 16bit L-F		
Stage 1	**2x16 NAND2**	**16 x NOR2**	**16 x INV**
Stage 2	**8 x OAI21**	**7 x NOR2**	**15 x INV**
Stage 3	**8 x AOI21**	**6 x NOR2**	**28 x INV**
Stage 4	**8 x OAI21**	**6x NAND2**	**18 x INV**
Stage 5	**8 x AOI21**		**11 x INV**
Stage 6	**16 x XOR**		**1 x INV**

Table 4: Shows different types of gate used in Conventional Ladner- Fischer adder.

After having an account of different types of gate used in a conventional 16 –bit adder , the evaluation of total number of gates included and also the calculation of the size of each gate is done ,so that we can finally know the size of an adder.
In the table given below, some notations are used which are given as:
$Tot_{l\text{-}f}$ = Total gates used in the diagram of 16- bit Ladner- Fischer adder (l-f) .
$S_{l\text{-}f}$ = Total gate of same type like NAND and A2O2I etc
N_g = Number of grids in each gate.
$P_{l\text{-}f}$ = **Total size of an adder** in terms of number of **Grids** i.e. $S_{l\text{-}f}$ and N_g (=$Tot_{l\text{-}f}$)

For 16 bit Ladner – Fischer adder		
Gates used	**$Tot_{l\text{-}f}$= $S_{l\text{-}f}$ x N_g**	**$P_{l\text{-}f}$ = $Tot_{l\text{-}f}$**
NAND 2	**38 x 3**	**114**
NOR 2	**23 x 3**	**69**
AOI21	**16 x 4**	**64**
OAI21	**16 x 4**	**64**
XOR	**16 x 7**	**112**
INV	**89 x 2**	**178**
	$\Sigma S_{l\text{-}f}$ =198	**$\Sigma P_{l\text{-}f}$ =601**

Table 5: Evaluating Total Number of Gates and Total number of Grids for counting the size of 16 bit L-F adder

So, as shown above, the total number of **grids** used in designing conventional 16- bit adder is **601.**

2.9 Poly grids:

For the obtaining size of an adder, we have considered grids for individual gates. An approximation of width and length for every gate is done on the basis of grids. Using grids is appropriate for estimating the height and width of a gate as grids cover the overall area in a gate. The distance between source and drain in a gate describes the length of a gate and this length is set by the width of ploysilicon gate .In every gate the grids are present showing the size of an adder and the presence of grids supports placement of any silicon piece in between them.

Grids are the vertical tracks which are not drawn to scale but can be easily counted around polysilicon gate. All gates are examined for size based on their grids. Grids are always counted as one more then input. Considering total inputs as N the total number of grids in that gate will be N+1.

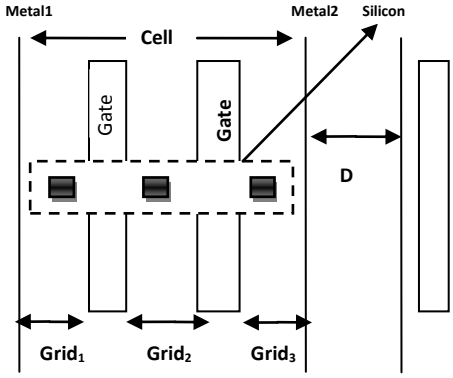


Figure 14: Internal structure of cell highlighting grids

The diagram above shows an internal structure of a gate where:
Cell – The width of a complete gate.
Gate – Two polysilicon wire representing gate in the structure.
$Grid_1$ -The distance from M_1 to first polysilicon wire.
$Grid_2$ - Distance from first polysilicon wire to other polysilicon wire.
$Grid_3$ - Distance from first polysilicon wire to metal M_2.

While designing, this method helps in arranging lot many gates symmetrically and thus helps EDA tools to adjust everything in a regular grid .For all gates we can find the size by counting the number of grids. Below are some of the examples of cell layouts for different gates. For example, to find the number of grids for a 3 input NAND gate, there are 4 grids for valency 3 NAND gate because we have 3 polysilicon wires.

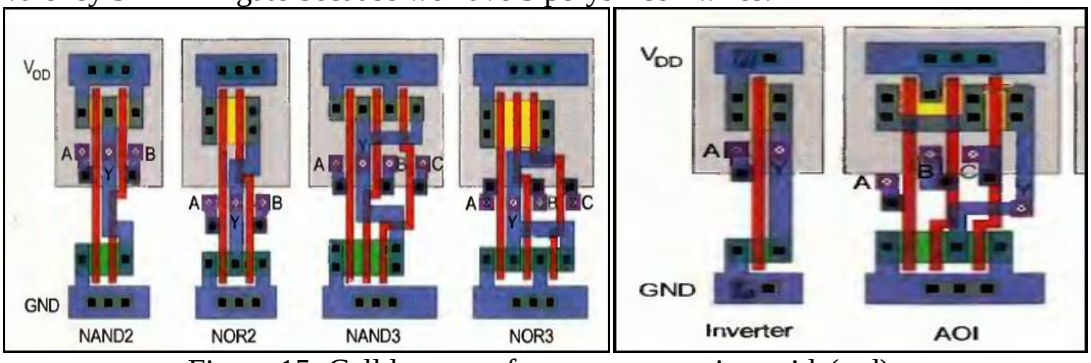


Figure 15: Cell layouts of gates representing grids(red).

So for counting the length of cell, the height and width of ploysilicon is considered and this is the reason for us considering grids for evaluating the size of logic gates.

2.10 Logical Effort:

Logical effort delay model is a design methodology for **estimating the total number of CMOS stages required to implement logic function and to calculate delay**. Logical method is the simplest and the accurate VLSI area model for prefix adders implemented in CMOS. It is a measure of the worst performance for the output current of any logic gate in comparison to an inverter. This comparison in gates is viable only when same capacitance is applied across their input. Logical effort can also be defined as the slope achieved by dividing delay v/s fan-out of logic gate to the inverter.

Terms related (Burgess N. , New Models of Parallel Prefix Adder Topologies, 2005):

Logical effort ,g - FET gate capacitance to minimum sized inverter for same output current.
Electrical effort, h - ratio of output capacitance for each CMOS logic gate along critical path. Branching effort, b - ratio of total capacitive load on one CMOS logic gate output to the gate capacitance of the next CMOS on critical path.

Total logical effort: It gives the collective logical effort of all inputs present in the logic gate together. The Total logical effort of NOR gate means logical effort of both the inputs whereas the logical effort on NOR gate means taking logical effort of one input instead of two input transistors. It is therefore the sum of the logical effort per input for each input to the logic gate.

Calculating Logical effort:

For calculating the logical effort, we design a logic gate with the characteristics almost equivalent to that of an inverter specifically the output current drive. Logical effort (Sutherland, 1999) is the ratio of capacitances and so for its calculation we consider the arbitrary values of capacitance which enables simple calculations.

Initially all the transistors are considered of the minimum length so that the width covers maximum transistor sizes. Usually the pull up transistor is broader then pull down transistor for holding the same conductance value.

$\mu = \mu_n / \mu_p$ where, μ gives the ratio of PMOS and NMOS with in an inverter.

Basically, the value of NMOS / pull down transistor is ‘w’ whereas the pull up /PMOS transistor is ‘2w’, so the overall capacitance is calculated as 3 which is the value of an inverter as shown in fig. below:

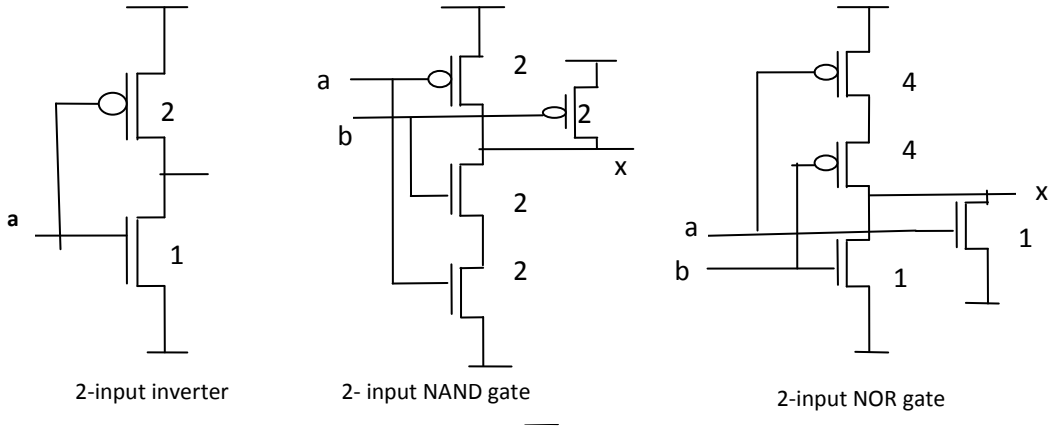


Figure 16: Different gates showing different values of capacitance

The circuits shown are of two different input logic gates i.e. NAND and NOR. In NAND, two pull down /NMOS transistors are arranged in series. According to ohm's law, value of the capacitance remains equal if the transistors are in series. Reason for previous statement is that these transistors must have twice the value of conductance in comparison to an inverter. Whereas the value of PMOS/ pull up transistors in a NAND gate, remains the same to achieve same drive as an inverter. These conductance values of NMOS and PMOS are important for further calculating the different efforts like logical effort (g), branching effort (b) and parasitic delay (p).

The total input capacitance for one input signal is considered as the sum of widths for NMOS and PMOS gates. For NAND gate, the value of the CMOS capacitances is 2+2 = 4. For an inverter the total capacitance is equal to 1+2 = 3. Therefore the logical effort for 2 – input NAND is **g** = 4/3. On the contrary, for NOR gate, the pull down transistor will have their regular capacitance values (i.e., 1) but for pull up transistor, width of each transistor is will be 4 units wide because it has to be equate the width for pull up transistor of an inverter gate which is 2 units .So the resulting capacitance of a NOR logic gate will be **g = 4+1 = 5/3.**From above we see that for NOR gate **g(logical effort)** is more than NAND's logical effort because the pull down transistor are the major contributor for building capacitance whereas pull up are not similarly important for the same task.

Asymmetric Logic gate:

Asymmetric logic gates are the one whose logical effort differs with the total number of inputs. Till now we have evaluated logical effort for symmetric gates (Sutherland, 1999) but for asymmetric gates like A2O2I, the circuits have been shown as below :

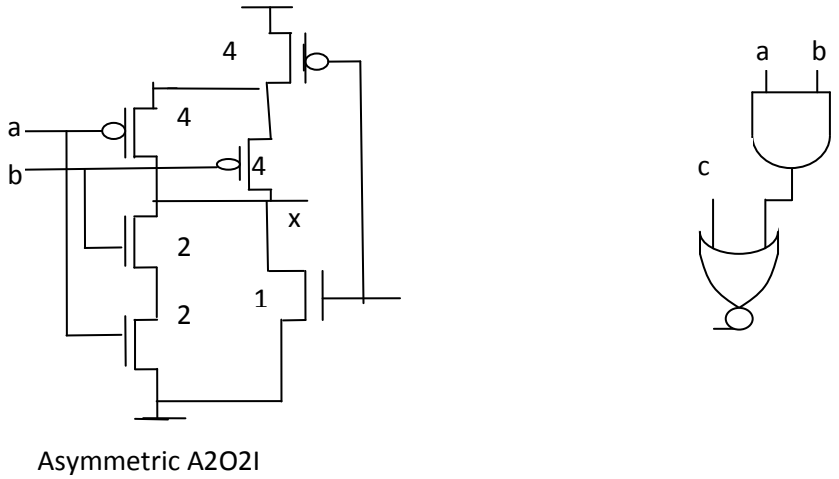


Figure 17: Transistor arrangements with values for Asymmetric gates. (Sutherland,Sproull, Harris, 1999)

In asymmetric gates as well the transistors tries to equate their values with an inverter. In the pull-up structure, each pull-up transistor is having the size of 4 but the pulldown transistors remain same with the size of 1.The table should be referred while calculating logical effort. According to the table and the formulas given, the logical effort for A2O2I is obtained as, g = 17/3.

Term	Stage Expression	Path Expression
Logical Effort	g (Table 5)	$G= \Pi g_i$
Electrical Effort	$h= C_{out}/C_{in}$	$H= C_{out\text{-}path}/C_{in\text{-}path}$
Branching Effort	-	$B=\Pi b_i$
Effort	F=gh	$F=GBH=\Pi f_i$
Effort delay	F	$D_F=\Sigma f_i$ minimised when $f_i=F^{1/N}$
Number of stages	1	N
Parasitic Delay	P (table 6)	$P=\Sigma p_i$
Delay	d= f+p	$D= D_F+P$

Table 6: Formulas for Logical Effort calculation (Sutherland,Sproull, Harris, 1999)

The values of g can be obtained by referring the table below. These assumptions of the values arranged in table simplifies the use of logical effort .

	Number of Inputs					
Gate Type	**1**	**2**	**3**	**4**	**5**	**N**
Inverter	1					
NAND		4/3	5/3	6/3	7/3	(n+2)/3
NOR		5/3	7/3	9/3	11/3	(2n+1)/3
Multiplexer		2	2	2	2	2
XOR (Parity)		4	12	32		

Table 7: Calculating the values of g for Logical effort

Parasitic delay:

In a logic gate, transistors have values for internal resistance. This internal resistance is highly dependent on the sizes of transistors. Due to this internal resistance in the transistors there is a parasitic delay caused. So, parasitic delay, in a logic gate is the count of internal delay caused because of internal resistance. The total diffusion capacitance on output node of a CMOS logic gate in relevance to the input FET gate capacitance of minimum sized inverter gives an estimation of parasitic delay caused. On the whole, it is the drain capacitance present in a logic gate .Transistor sizes does not help in inculcating the drain capacitance. The value of gate capacitance is equal to that of the drain capacitance and so the value of parasitic delay gives us an average of overall drain capacitance in comparison to inverter. Table 6 given below will help in calculating parasitic delay in a gate.

Gate Type	Parasitic Delay
Inverter	P_{inv}
n-input NAND	nP_{inv}
n-input NOR	nP_{inv}
n-way multiplexer	$2nP_{inv}$
XOR, XNOR	$4P_{inv}$

Table 8: Parasitic delay

Electrical effort: The electrical effort is the total output load capacitance over input capacitance. The methodology of Logical effort is based upon capacitances where Electrical effort gives the ratio of output over input capacitance. The valuation of input and output capacitance is dependent on the given values of transistors sizes.

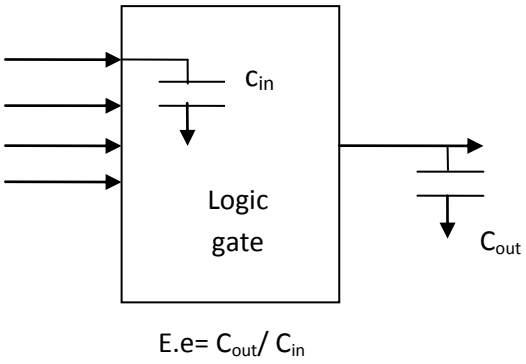


Figure 18: Electrical effort in a gate (Sutherland,Sproull, Harris, 1999)

Optimisation of critical path:

During the final stage of calculation of Logical effort, the critical path is chosen and for the overall estimation of delay the critical path is optimized. The parameters and the formulas given below are used while calculating delay. The technological speed parameter we take is **τ**, usually for a .25μm process the delay is 16- 20ps and it varies depending on the design process taken.

Summary of the method:

- After taking the critical path , the total logical effort is calculated as F= GBH .The path Logical effort (g) of each gate on the critical path is calculated by considering the width of each gate and is defined as the product of the logical effort of all

individual gates employed in the critical path. Then the value of branching effort b is calculated by taking the product of branching effort of all the gates in the path and finally the value of H which is C_{out}/ C_{in} which gives the overall capacitance of initial and the final stages in the design.

- Secondly the total number of stages in a circuit is very important as it should give the least delay , if the number if stages obtained after the calculation with the formula of N=round($\log_4 F$) is more than encountered from the diagram then it needs to be equalised by adding an inverter in the design. This ways the actual number of stages in a CMOS circuit is obtained by adding or removing stages till the number of N.
- Total delay of the circuit when all N stages have equal effort, can be calculated as: $D= (N \alpha + \Sigma p)$ by taking the sum of all the parasitic delays and also $\alpha = F^{1/N}$.
- Finally the delay is obtained in terms of FO4. Fan-out-of-4 is considered equivalent to 5τ for any design so the delay is divided by 5 at the end . FO4 is explained as the total delay of an inverter driving four copies of it.

FO4

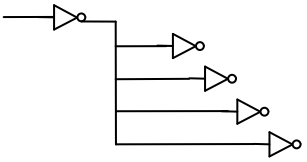


Figure 19:FO4 circuit

2.11 Table of different cells:

Given below is the tabular representation of the cells used in this thesis. These gates are mostly used for VLSI technology. This table gives a clear idea about the logic these gates implementing equations as they are used in the circuits.

Name of gate	Logical equation
A22O2I	Z=~((a .b) +(c.d))
O22A2I	Z=~((a + b)•(c+d))
EXOR	Z=~(a ⊕ b)
A2O2I	Z= ~((a. b)+c)
O2A2I	Z=~((a +b)c)
A2O3I	Z=~(((a.b)+c)+d)
A3O2I	Z=~((a. b .c)+d)
O3A2I	Z=~((a+b+c)d)
O2A3I	Z=~(((a + b) c)d)
NAND	Z=~(a . b)
NOR	Z=~(a + b)

Table 9: Different VLSI notations of gates

3 Research Methodology and Adder Design:

3.1 High Speed Binary Addition:

Jackson and Talwar (2004) focused on constructing the carry at all levels in the hierarchy. They investigated that no single technique for implementation of an adder is present. All the adders are based on some or the other techniques. The families of adders mainly parallel prefix adder can be derived from the reduced generate and hyper propagate functions which are a generalization of the Ling pseudo carry. Ling reduced the delay by minimizing the generate function but this simplification was only applied to first level and otherwise the complexity at all the other levels resembles to that of a parallel prefix adder. High Speed Binary Addition (R.C.Jackson and S. Talwar, Nov 2003) provides the novel methodology of simplifying group generate function at all levels of logic thus reducing the complexity of group generate at all levels of the logic instead of just first level. Jackson and Talwar approach deals with generation of a carry and balancing reduced generate group with the not kill group.

Note: This section discusses Jackson and Talwar technique (R.C.Jackson and S. Talwar, Nov 2003) in a more clear way. An effort is made to make equations more users friendly.

Considering addition of two binary numbers $a_n, a_{n-1}, a_{n-2} \ldots\ldots a_0$ and $b_n, b_{n-1}, b_{n-2} \ldots\ldots b_0$ .Parallel prefix adders computes carry propagate (p) and generate (g) functions. The initial stage gives the two mentioned functions for every bit:

$g_n = a_n b_n$

$p_n = a_n + b_n$.

The binary step methods combines generate term and propagate term to give complete generate function. As discussed

$G_{n:0} = G_{n:k} + P_{n:k}\ G_{k-1:0}$ Equation 15: group generate

$P_{n:0} = P_{n:k}\ P_{k-1:0}$ Equation 16: propagate group

Above given functions can be written for high valency formulations. For high valency the groups are more segregated. The feedback for higher radix formulation is that while approaching towards high radix the functions become complex with the increased complexity in logic. So, in a design forming different groups is helpful.

For the equation below three segregation of group is done from n to k , k-1 to k' and k'-1 to 0.

$G_{n:0} = G_{n:k} + P_{n:k}\ G_{k-1:k'} + P_{n:k}\ P_{k-1:k'}\ G_{k'-1:0}$ Equation 17: generate equation for higher radix cells

$P_{n:0} = P_{n:k}\ P_{k-1:k'} P_{k'-1:0}$ Equation 18: Propagate function for higher radix cells.

The factorised term from ling's addition is our counterpart.

The equation that Jackson and Talwar gave for reducing the carry bit at every single stage is:

$R_{n:0} = g_n + g_{n-1} + g_{n-2} + nk_{n-1}nk_{n-2:7}\ g_{n-3:7} + ...$ Equation 22
where
$R_{n:0}$ is the reduced carry function which removes not kill term from as many as levels as required in order to reduce complexity from the generated term for every bit.

The products the Reduced carry function with the not kill factor to obtain the carry generate function as:
$G_{n:0} = nk_n \cdot R^{(j)}_{n:0}$ *Equation 23*
Where j= range for which the number of not kill bits are factored out or the superscript describes the range over which the carry is evaluated.

Factorization of reduced carry equation for the valency of 7:
$R^{(3)}_{7:0} = g_7 + g_6 + g_5 + nk_{4:0}\ g4_{:7} + ...$ *Equation 24*

As we have seen in equation 23
$G_{7:0} = nk_7 \cdot R^{(3)}_{7:0}$
$G_{7:0} = (g_7 + nk_7) \cdot R^{(3)}_{7:0}$ *Equation 25*

Multiply with the factor in order to make it compatible for 8- bits:
$(g_7 + nk_7)$

Replacing R (3) 7:0 with the factorization value given in Equation 24 , we obtain:
$G_{7:0} = (g_7 + nk_7nk_6) \cdot (g_7 + g_6 + g_5 + nk_6nk_{5:7}\ g_{4:7} + ...)$ *Equation 26*

Making the equations simple:
consider (nk_7nk_6) as A and $(g_6 + g_5 + nk_6nk_{5:7}\ g_{4:7} + ...)$ as B
and now further solving:
$G_{7:0} = (g_7 + A) \cdot (g_7 + B)$
$G_{7:0} = g_7 + g_7 A + A B + g_7 B$
$G_{7:0} = g_7 + (1 + A + B) + AB$ (1+ A +B)=1 in Boolean algebra
$G_{7:0} = g_7 + AB$ *Equation 27*
$G_{7:0} = g_7 + (nk_7nk_6)(g_6 + g_5 + nk_6nk_{5:7}\ g_{4:7} + ...)$
$G_{7:0} = (g_7 nk_7 + g_7 nk_6) \cdot (g_6 + g_5 + nk_6nk_{5:7}\ g_{4:7} + ...)$
$G_{7:0} = (g_7 + g_7 nk_6) \cdot (g_6 + g_5 + nk_6nk_{5:7}\ g_{4:7} + ...)$ equation 9 $g_{n:0=}\ nk_n \cdot g_{n:0}$

On further solving we get the generate equation 4
$G_{7:0} = g_7 + nk_7g_6 + nk_6g_5 + nk_6nk_5\ g_{4:7}$ *Equation 28*

Thus it **proves** :
$[G_{n:0} = nk_n \cdot R^{(j)}_{n:0}]$ The 2nd bracketed term is Ling's Pseudo carry . The first term is generalised as ***D*** which is the combination function generate function and propagate function over the limit of n to i.
The logic functions we need to factorize are:
$D_{n:i} = G_{n:i} + nk_{n:i}$ *Equation 29*
$= G_{n:i+1} + nk_{n:i}$

$B_{n:i} = g_n + g_{n-1} + g_{n-2} ..... + g_{n-i}$ *Equation 30*
If in case the addition of a and b terms *i. e:* $\mathbf{a_{n-1}\ a_{n-2}\ a_{n-3}..... a_0 + b_{n-1}\ b_{n-2}\ b_{n-3}\ .....\ b_1\ b_0}$ is high then a carry out is generated and the other function $B_{n:i}$ is high if the value of $D_{n:i}$ is high with in n to i range.

So, in general form:
$G_{n:k} = D_{n:i}[B_{n:i} + G_{n-1:i}]$ *Equation 31*
$[B_{n:i} + G_{n-1:i}]$ is the reduced complexity function.

The correction term 'D' can be is multiplied with Reduced carry function in order to obtain Generate function:
$G_{n-1:0} = D_{n-1:n-k} . R^{(k)}_{n:0}$ *Equation 32*

The Ling factorization is used to obtain reduced carry bit and for higher valencies the correction term is equal to the not kill bits.
$D_{n-1:n-1} = nk_{n-1}$
$D_{n-1:n-2} = g_{n-1} + nk_{n-1:n-2}$
$D_{n-1:n-3} = g_{n-1} + nk_{n-1:n-2} + nk_{n-1:n-3}$
$D_{n-1:n-k} = [G_{n-1} + nk_{n-1:n-k}]$ *Equation 33*

So the complexity of reduced carry function is reduced with the increased correction term. Thus, the logic expression D have less complexity than the generalised group generate function *G* in terms of R. So we use less CMOS logic cells to implement this. The correction term can be evaluated by considering it over the limits n-1 to 0 for generate term and the not kill bits over the limits of n-1 to n-k.For obtaining high performance in the Adders we have to balance these two terms i.e R and D so that both are of reduced complexities for high valency as well.
The equation to compute sum for bit position n is given as,

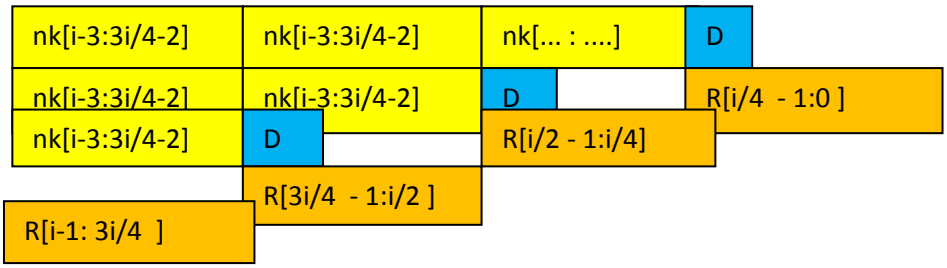
$S_n = p_n \oplus G_{n-1:0}$
$S_n = p_n \oplus D.R_{n-1:0}$
$S_n = (R_{n-1:0})^{-1}(p_n) + R_{n-1:0}(p_n \oplus D)$

Hyper propagate function, q:
As seen above we know that recursion covers an important part of ling's equations. This refers that the Pseudo carries can be build over the sub groups and preceding groups ahead. And Recursion is also very important for our higher radix implementations in reduced carry functions (G. Dimitrakopouloas and D. Nikolos, February 2005.)
(R.C.Jackson and S. Talwar, Nov 2003).
This can be proved by considering equations given in High Speed Binary Equations:
$G_{n:k} = D_{n:i}[B_{n:i} + G_{n-1:i}]$
$G_{n:k} = [G_{n:k} + nk_{n:k} G_{k-1:i}]$ *or*
$G_{n:k} = [G_{n:k} + P_{n:k} G_{k-1:i}]$
The complexity of reduced function if decreased will enhance the performance in an adder. The range of function is divided in to three different ranges which are from n-1:k ,k-1:k' and k'-1:0. Later the midpoints are selected for detecting other subgroups.

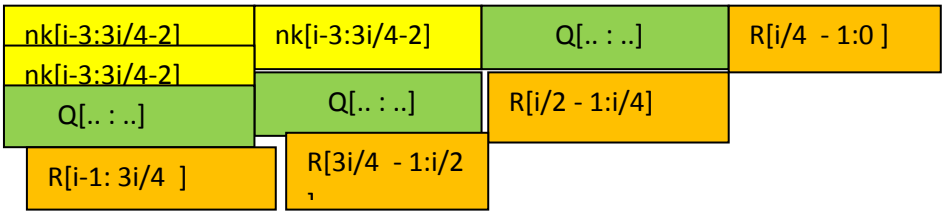
Considering the reduced generate function:
$R_{n-1:0}^{(n-m)} = B_{n-1:m} + G_{m-1:0}$ *Equation 34*
The expression $G_{m-1:0}$ *is* further divided in different limits ,so now we have:
$R_{n-1:0}^{(n-m)} = B_{n-1:m} + G_{m-1:k'} + P_{n:k} G_{k'-1:0}$
$R_{n-1:0}^{(n-m)} = [B_{n-1:mk}] + [B_{k-1:m} G_{m-1:k'}] + P_{m-1:k'} G_{k'-1:0}$ *Equation 35*
The 1st and 2nd bracketed should be removed in order to obtain the final generate term because it all recursion.

$G_{k'-1:0} = D_{k':1:m'} [B_{k'-1:m'} + G_{m'-1:0}]$

• $Q^{(m)}_{i-1:0} = nK^{(m)}_{i-1:i-m.} D_{i-m-1:0}$



The width of Q function is same as that of R function. The Q function can be obtained for every stage by combining the not kill term and Correction term as shown below:



Substituting the 35 we get;

$R_{n-1:0}^{(n-m)} = R_{n-1:k}^{(n-k)} + R_{k-1:k'}^{(k-m)} + [P_{m-1:k'} D_{k'-1:m'}] R_{k'-1:0}^{(k'-m')}$ *Equation 36*

Terms $P_{m-1:k'} D_{k'-1:m'}$ are referred to as Hyper Propagate, and this recursive functions needs to be build recursively. The construction of Hyper Propagate considering the group of propagated functions and the building of generated functions can be examined below:

$D_{n:k} = D_{n:i}[B_{n:i} + D_{n-1:i}]$ *Equation 37*

$D_{n:k} = G_{n:i} + P_{n:i} + D_{n-1:i}$ *Equation 38*

The function $P_{m-1:k'} D_{i'-1:m'}$ can be generalised as:

$Q_{n-1:0}^{(n-m)} = P_{n-1:m'} D_{m-1:0}$ *Equation 39*

The range of function is divided in to three different ranges which are from n-1:k ,k-1:k' and k'-1:0 .

The limits are further divided to m and m' as the intersection of other sub groups. Using equation 37

$Q_{n-1:0}^{(n-m)} = P_{n-1:m'} D_{m-1:i'} [B_{m-1:i'} + D_{i'-1:0}]$ *Equation 40*

$Q_{n-1:0}^{(n-m)} = P_{n-1:i} P_{i-1:m} D_{m-1:i'} [B_{m-1:i'} + D_{i'-1:0}]$ *Equation 41*

Further solving the D function and the 1st and 2nd bracketed term can be structured as:

$Q_{n-1:0}^{(n-m)} = P_{n-1:i} P_{i-1:m} D_{m-1:i'} [B_{m-1:i'} + G_{i'-1:m'} + P_{i'-1:m'} D_{m'-1:0}]$

$Q_{n-1:0}^{(n-m)} = Q_{n-1:i}^{(n-k)} Q_{i-1:i'}^{(i-m)} [R_{m-1:m'}^{(m-i')} + Q_{i'-1:0}^{(i'-m')}]$ *Equation 42*

The above shown equation is the reduced form of hyper propagate function.

The width of Q is same as R. We can observe that with different cases the reduced generate function is less complex then the general generate function. It is really helpful as it replaces not kill bits with Q function with in turn reduces a logic level while realising in CMOS.

3.2 New adder designs:

For implementing adder designs based on the discussed theory of Jackson ad Talwar , low valency gates are used . I have designed two novel designs of adder. The published paper Family of adders discusses different kind of parallel prefix adder but Knowles have focussed mainly on Ladner and Fischer adder. Then out of these two types Ladner and Fischer is proven to be better in the previous MSc students' research .This being the reason I have considered the base configuration of Ladner – Fischer and using the technique of Jackson and Talwar on that parallel prefix adder to obtain the small and fast structure.

Also as we have discussed about sparse trees and it is observed that application of sparse tree is a wise decision as it reduces the size of an adder by reducing the fan-out through reduced pseudo generate groups and thereby reducing the number of gates. Also the usage of sparse tree depends upon the configuration and type of adder used for designing along with the theory applied to it. These two adders designs are tested with the new method of implementing sparse trees and the results are listed. Valency 2 and 3 are used to design the new adder designs.

For proving the size of the new adder is smaller, a comparison of the total number of gates used in original 16 bit Ladner -Fischer adder is done with the total number of gates used in novel 16- bit adder in a tabular form.

All the designing procedure and key points required to logically design an adder are mentioned in this chapter. The equations obtained after applying theory, gates used in designing, their logical effort with individual gate capacitance are shown. This being the designing project provides the basic designs of an adder. The adder is small and fast, this is theoretically proved by Logic delay model and by comparing the size of the novel adder with that of the recently used adder.
All the above mentioned adders and their testing with sparse are explained in this chapter.

3.3 16- bit Parallel Prefix adder based on Jackson and Talwar's approach:

The logic diagram of 16 bit adder is given presenting all the logic gates. There are total three stages for calculating carry bit .Valency two and valency three gate are used as the delay increases with the valency. Sum is computed at the end of every individual bit using Multiplexers. In this design, the initial level provides output as generate and not kill values of every single bit taken as input and these bits are propagated till msb to calculate the required value of carry and sum. All D, Q terms and other terms are clearly drawn in the diagram for simplicity.

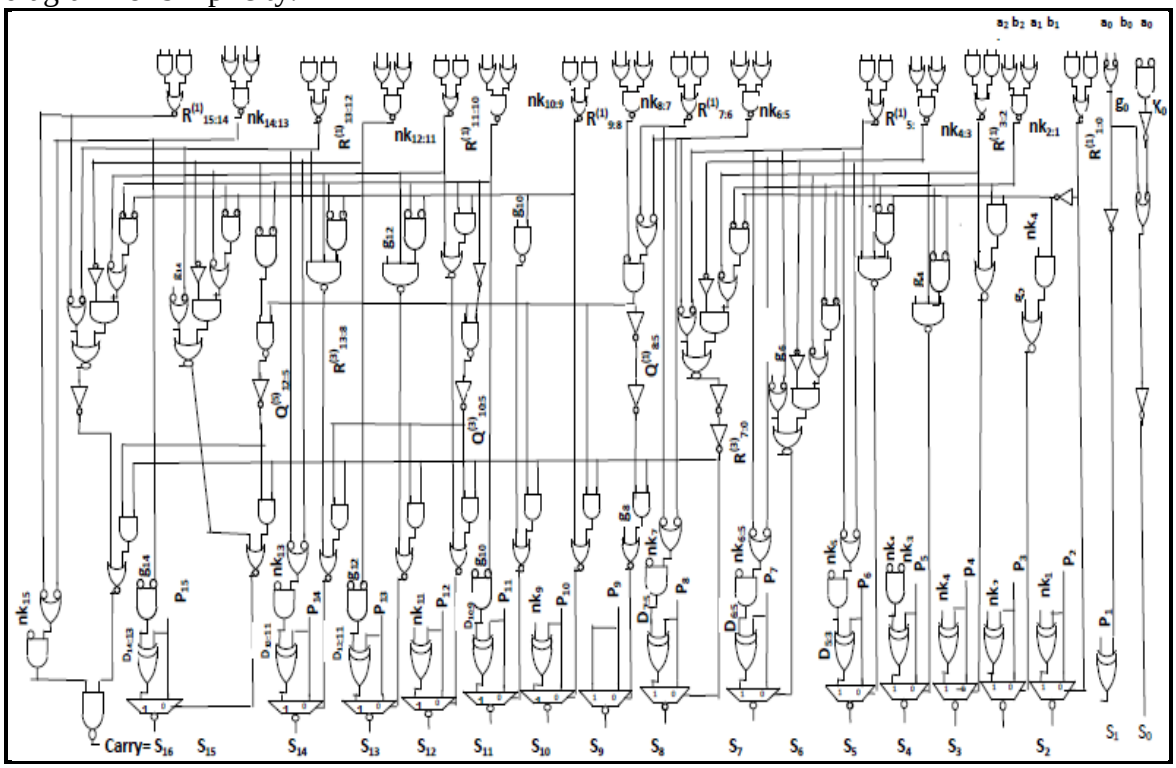


Figure 20: Logic Diagram for 16 - bit parallel prefix adder based on J & T approach

The diagram showing grey and black cells (dot diagram) is shown below .This diagram shows the logic for carry estimation in the design.

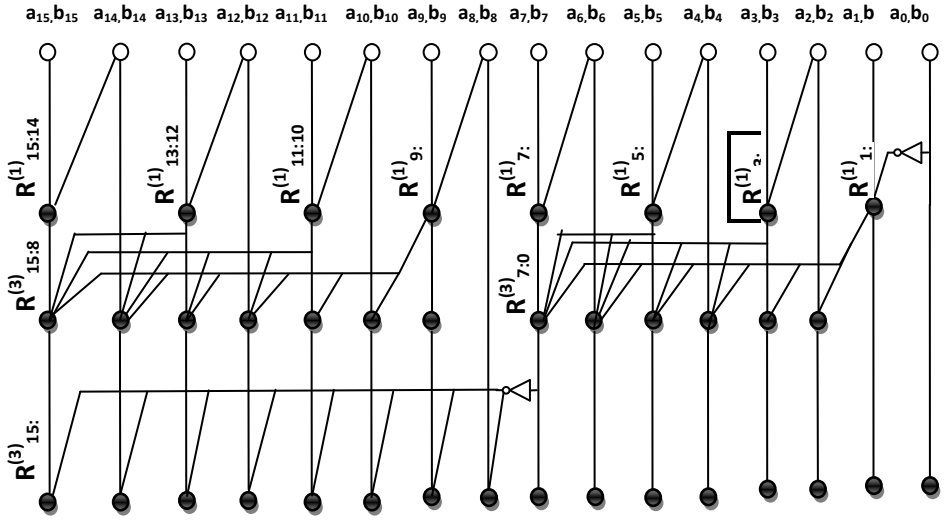


Figure 21: dot diagram representation of adder carry tree

The dot diagram clearly shows a 2*4*2 adder. The 3 levels calculating carry are represented by dark circular dots. In the initial level, a and b are given as the input to the A22O2I gates and O22A2I gates. This level generates all the reduced generate term from $R^{(1)}_{1:0}$ to $R^{(1)}_{15:14}$, and every term will have one not kill taken out or missing. The reduced generate terms are calculated using A2O2I gates and the other group in the first level is not kill group which is from $\sim k_{2:1}$ to $\sim k_{14:13}$ and are generated using O2A2I gates .

In the second stage we have **valency 4, but in the design, two gates of valency 2** are used. This means that the total inputs are 4 but to reduce the delay in the logic gate, logical equations are mathematically modified to use valency 2 gates instead of valency 4 gate. Though stage two uses one stage but it still requires two logic gates because the carry computation logic is complicated and the fan out load shoots up in this stage and keeps on increasing towards msb. As discussed for the requirements in adder, it is very important to have buffering among carry logic because high fan out requires the inverter to sustain excessive fan out load. In the second stage also required amount of not kill bit can be factored out so the combinations of at maximum 4 reduced groups can be combined to generate the sub groups of $R^{(3)}_{7:0}$ to $R^{(3)}_{15:8}$. In this stage we obtained G term by multiplying the number of not kill bits in reduced group term i.e. $D \bullet R$ so there are three not kill bit missing in the reduced generate terms.

Depending on the valency gates are taken, the implementation of valency 2 can be done using single gate as it has two inputs. For valency 3, factorisation of one bit can be done or else implementation of three valency can be done. In the adder designed, two logic levels are used for valency 4 logic with 3 not kill bits factored out and the total number of input are four after mathematical reduction in the equation. If the gate is in the critical path and more number of not kill bit is factored out than it increases the number of terms in D group and so the complexity which further add to the delay in the circuit .

The last stage for calculating carry consist of valency 2 gates which computes the carry for two groups of reduced generate term again and the output is given to the multiplexers. The implementation of multiplexers is done using sparse tree at the end for computing sum.
As shown, for computing sum, for every multiplexer, a R term is present. This R term takes lot more time than other not kill group for computation. Delay in the circuit majorly depends on the complexity of R term. With the supply of R term in a mux, it is tried that the complexity should be reduced by increasing more number of bits in the D term. For 16-multiplexer we have 16 R terms going into it.

Equations for every sum bit: Below are the equations based on Jackson and Talwar technique used for designing the 16 bit adder. In this adder, gates are designed based on logic equations obtained. Depending on the terms in the equation, valency for each gate is decided. Here are the important equations given for having an idea of the work done in the presented adder:

Carry from all last 7 bits are propagated in this sum bit. $R^{(3)}_{7:0}$ group with 3 not kill bits is fed to the multiplexer and obtained result is

MUX : $\mathbf{\sim R^{(3)}_{7:0}(P_8)}$ **&** $\mathbf{R^{(3)}_{7:0}(P_8 \oplus D_{7:5})}$

$S_8 = P_8 \oplus G_{7:0}$

$G_{7:0} = D_{7:5} \bullet R^{(3)}_{7:0}$

$S_8 = P_8 \oplus D_{7:5} \bullet R^{(3)}_{7:0}$

$D_{7:5} = g_7 + nk_7 g_6 + nk_{7:5}$

$\mathbf{R^{(3)}_{7:0} = [(R^{(1)}_{7:6} + R^{(1)}_{5:4}) + nk_{4:3}\,(R^{(1)}_{3:2} + nk_{2:1}\,R^{(1)}_{1:0})]}$

Except for the sum bit ,in all the equations given below , the terms on the left hand are evaluated in second stage that means all are reduced pseudo generate groups.The sum at bit position 15 will be obtained as:

$S_{15} = (a_{15} \oplus b_{15}) \oplus G_{14:0}$

$G_{14:0} = D_{14:13} \bullet R^{(2)}_{14:0}$

$S_{15} = P_{15} \oplus D_{14:13} \bullet R^{(2)}_{14:0}$

$D_{14:13} = g_{14} + nk_{14} \bullet nk_{13}$

$\mathbf{R^{(2)}_{14:0} = R^{(2)}_{14:8} + Q^{(5)}_{12:5} \bullet R^{(3)}_{7:0}}$

$\mathbf{Q^{(5)}_{12:5} = nk_{12:9}\,Q^{(1)}_{8:0}}$

$\mathbf{R^{(2)}_{14:8} = [(g_{14} + R^{(1)}_{13:12}) + nk_{12:11}\,(R^{(1)}_{11:10} + nk_{10:11}\,R^{(1)}_{9:8})]}$

MUX: $\mathbf{\sim R^{(2)}_{14:0}\,(P_{15})}$ **&** $\mathbf{R^{(2)}_{14:0}\,(P_{15} \oplus D_{14:13})}$

The carry term obtained at 16- bit position is:

$S_{16} = \mathbf{G_{15:0}} = D_{15:13} \bullet R^{(3)}_{15:0}$

$\mathbf{R^{(3)}_{15:0} = R^{(3)}_{15:8} + Q^{(5)}_{12:5} \bullet R^{(3)}_{7:0}}$

$\mathbf{R^{(3)}_{15:8} = R^{(1)}_{15:14} + R^{(1)}_{13:12} + nk^{(1)}_{12:11}\,(R^{(1)}_{11:10} + nk_{10:9} R^{(1)}_{9:8})}$

$\mathbf{Q^{(5)}_{12:5} = nk_{12:9}\,Q^{(1)}_{8:0}}$

The critical path taken in this design is S_{14} as the D term is more complex in this and this path covers maximum of the gates.

Critical path in 16-bit adder:

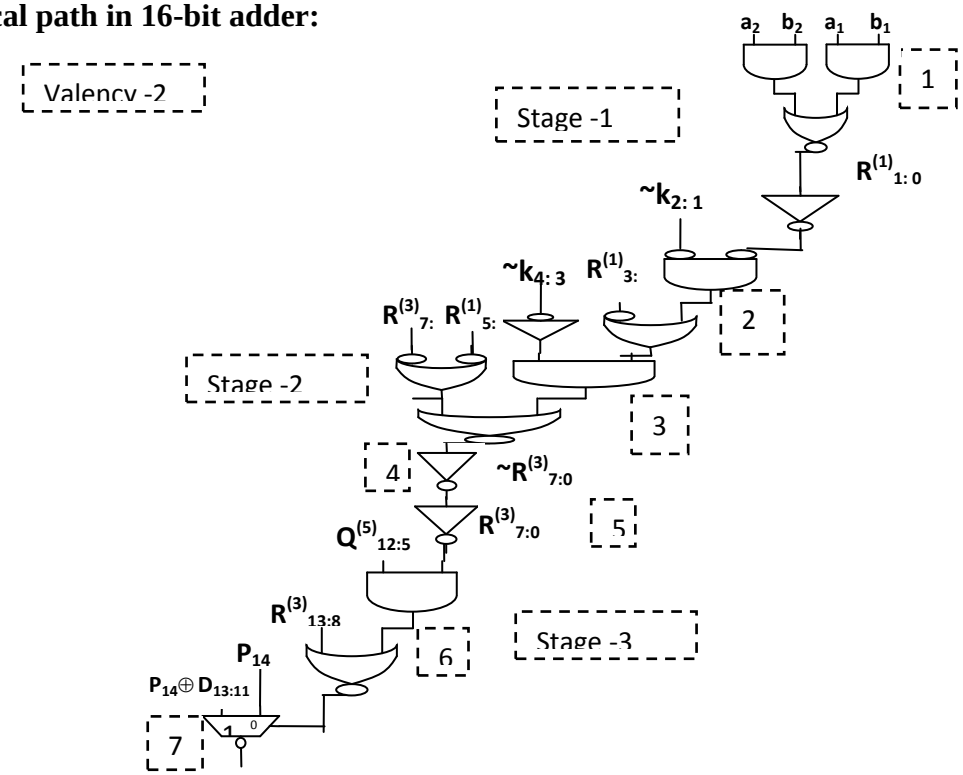


Figure 22: Critical path of 16- bit adder.

From the above diagram, it can be observed that the total number of stages is 7 and that is including everything from the first stage of O22A2I to the last sum logic into the multiplexer. It has three levels of logic divided in 7 stages. The first stage generates a Reduced generate term which along with the other inputs goes into stage 2 where it computes the subgroups of R term.

The second stage consist total of two logic levels resulting in two inputs gates to compute valency 2 equations. Output from O22A2I gates travels into valency-2 O2A2I and then in A2O2I. Resulting signal from above gates completes stage two. Now substitution of two inverters is done. The first inverter is employed as a buffer for driving four gates in third stage and the second inverter is placed to equalise the number of stages to be equal to the value of N obtained while calculating the logical effort.

The output signal obtained after second stage is the propagation of carry from all the lower bits which goes into the A2O2I gate to give an R term which is further given as an input to the multiplexer to compute the sum. The R term take more time to travel then other inputs to the multiplexer.

In sum logic, each multiplexer is supplied with an R term. This design is made by employing the technique of Jackson and Talwar.

Logical Effort Calculations for above 16 bit adder: Referring the table 5 and 6 given in logical effort description will be helpful to evaluate the values of g, b and p. The table below shows the values of capacitance in terms of g.b and p and also the cells on the critical path to calculate the total delay in the circuit.
For calculating the delay in the circuit we add the capacitance of wire as 2/3 as per according to the standard assumptions. (Burgess N. , Implementation of recursive Ling adders in CMOS VLSI, 2009)

	Cell	Load	Gb	P
$R^{(1)}_{1:0}$	A22O2I +O22A2I	INV +O2A2I +wire	1+6/3 +6	12/3
$R^{(1)}_{3:0}$	O2A2I	A2O2I+wire	6/3 + 2/3	8/3
$R^{(3)}_{7:0}$	A2O2I	INV+wire	1+2/3	7/3
$R^{(3)}_{7:0}$	INV	8*A2O2I+wire	8* 6/3 +7	1
$R^{(2)}_{14:0}$	AOI21	1*Mux +wire	1* 6/3 +2/3	7/3
S_{15}	Mux	A22O2I+O22A2I+wire	16/3	5
			ΠGb=13085	Σp=17.3

Table 10: Calculations for Delay in 16 bit adder

- F= GBH = 13085
- N= Round($\log_4 F$) = 6.83 = 7
- $\alpha = F^{1/N} = 3.72 = 3.8$
- D= (7 * 3.8 +17.33)/5 = 8.80FO4

Table for calculating gates :

Stage	Total number of gates in novel 16bit adder			
Stage 1	**15xA22O2I/O22A2I**	**2x NAND2/NOR2**	**1x INV**	
Stage 2	**10xA2O2I/O2A2I**	**3x NAND2/NOR2**	**4x INV**	**3xA2O3I**
Stage 3	**4xA2O2I**	**6xNAND2/NOR2**		
Stage 4			**8x INV**	
Stage 5	**7 xA2O2I/O2A2I**			
Stage 6	**12xA2O2I/O2A2I**	**30x NAND2/NOR2**	**14 x XOR**	**14xMUX**

Table 11: Listing all the gates used in 16- bit adder with Sparse -2 logic

The table given above shows all different types of adders used in the design but the critical evaluation of this design is shown in the table given below, which employs multiplying gates with their relevant number of grids.

For 16- bit J and T adder		
Gates used	**$Tot_{n-a} = S_{n-a} \times N_g$**	**$P_{n-a}=Tot_{n-a}$**
NAND2	**18 x 3**	**54**
NOR2	**15 x 3**	**45**
A22O2I/O22A2I	**15 x 5**	**75**
AOI21/OAI21	**19 x 4**	**76**
XOR	**14 x 7**	**98**
INV	**13 x 2**	**26**
MUX	**14 x 7**	**98**
AO1211/OAI211	**12 x 5**	**60**
	ΣS_{n-a} =120	**ΣP_{n-a} =532**

3.4 16- bit adder parallel prefix adder using Sparse-2 carry tree logic:

The second adder designed is with Sparse-2 carry tree logic. Sparse-2 carry tree logic is not yet proposed, so taking an initiative of testing sparse -2 carry tree logic with Jackson and Talwar approach. The aim of designing this adder is to test, whether Sparse-2 carry tree logic shows any impact in the size or delay of an adder. The bit configuration used for designing this adder is 8*8 and low valency cells 2*3*2 are used for implementing the logic obtained. The Logic diagram is shown below.

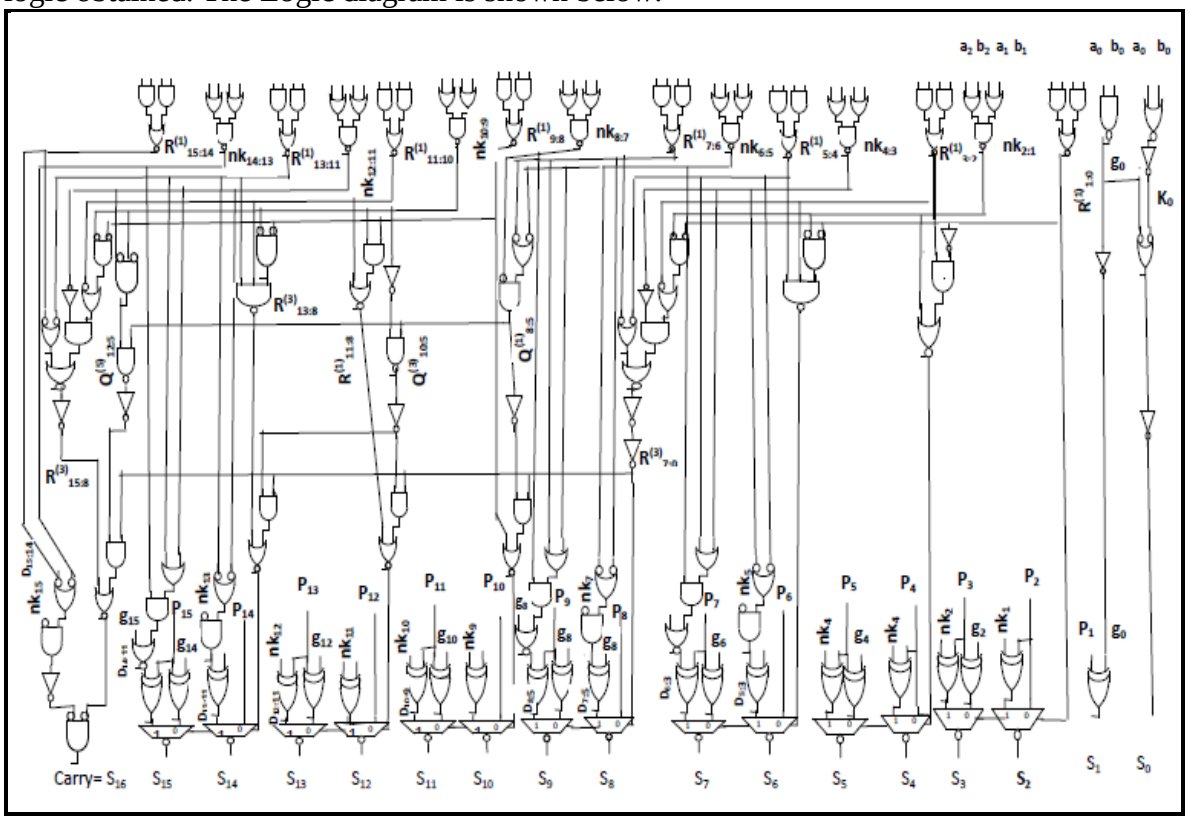


Figure 24 Logic diagram of 16 bit adder with sparse 2 carry logic

As shown , each R term goes into two multiplexers which reduces the delay as every single generate term propagates through two multiplexers thus instead of 16 R terms going to 16 Multiplexers , the implementation of 8 R term is done into 16 multiplexers which saves the wiring capacitance from $R^{(1)}_{1:0}$ to $R^{(3)}_{7:0}$ and $R^{(3)}_{7:0}$ to $R^{(3)}_{15:8}$ thus reducing the overall wire capacitance by almost half the value and giving the same R and D terms in the circuit. Also in sum computation , the D group is made more complex as this reduces the complexity in the R term which adds to the delay in the critical path and having complex D term does not affect the delay as they are not in critical path.

Critical path in 16- bit adder is taken as S_{15} as it is the longest path covering maximum number of gates in this adder . With the implementation of sparse-2 carry logic the Reduced pseudo generate term $R^{(2)}_{13:0}$ goes into both the multiplexers of S_{15} and S_{14} . Diagram below shows the critical path of this adder.

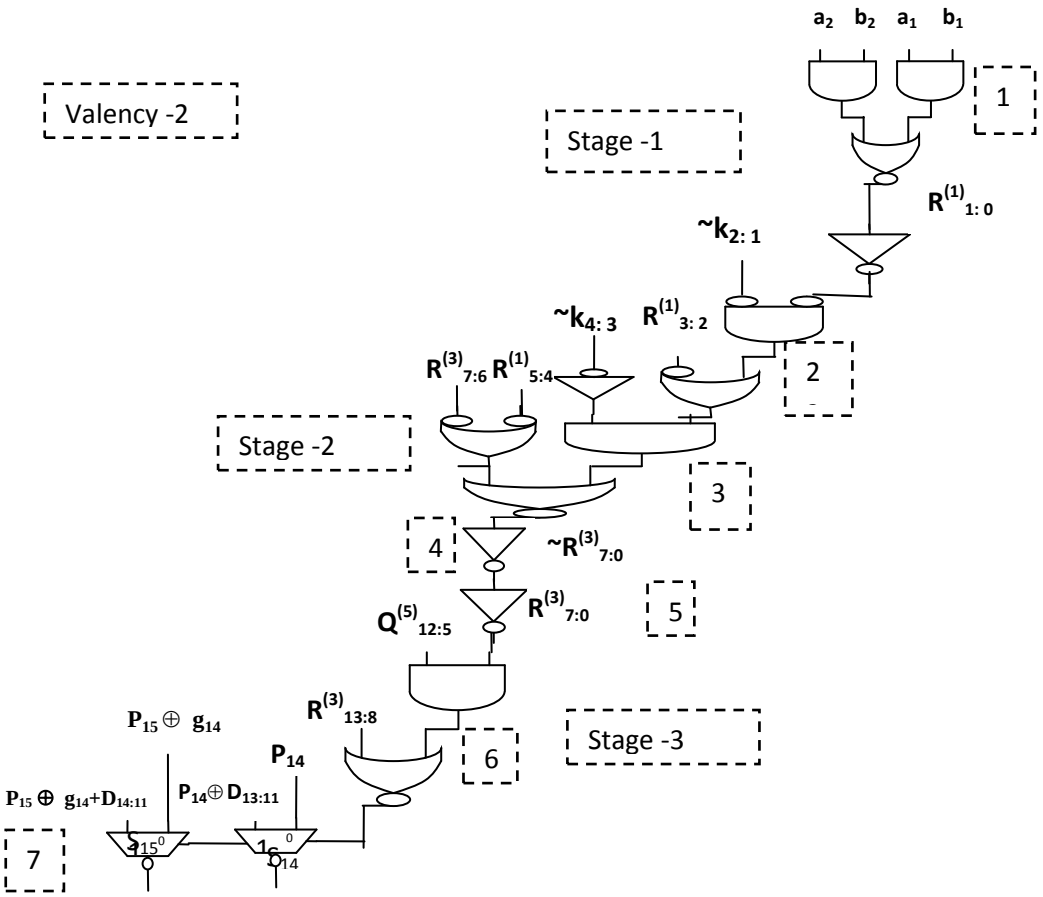


Figure 25:Critical path in 16- bit adder .

Logical effort calculating the delay is shown in the table below. The difference in the values s noticeable from $R^{(3)}_{7:0}$. In this design, instead of driving 8 gates only four A2O2I gates are driven by the Reduced term of $R^{(3)}_{7:0}$. This reduction in the logic is due to the use of Sparse carry tree which helps in reducing the fan-out from $R^{(3)}_{7:0.}$
Also now each R signal is going into to multiplexer and therefore we can observe change in the table exposing $R^{(2)}_{14:0}$ driving two multiplexers.

	Cell	Load	Gb	P
$R^{(1)}_{1:0}$	A22O2I +O22A2I	INV +O2A2I +wire	1+6/3 +3	12/3
$R^{(3)}_{5:0}$	O2A2I	A2O2I+wire	6/3 + 2/3	8/3
$R^{(3)}_{7:0}$	A2O2I	INV+wire	1+2/3	7/3
$R^{(3)}_{7:0}$	INV	4*A2O2I+wire	4* 6/3 +10	1
$R^{(2)}_{14:0}$	AOI21	2*Mux +wire	2* 6/3 +4/3	7/3
S_{15}	Mux	A22O2I+O22A2I+wire	16/3	5
			Πgb=13653	Σp=17.3

Table 12: Calculating Delay in 16- bit adder with Sparse -2 logic

- F= GBH = 13653
- N= Round($\log_4$ F) = 6.86 = 7
- $\alpha = F^{1/N} = 3.89 = 3.9$

- D= (7 * 3.90 +17.33)/5 = **8.81FO4**

Total delay of **8.81 FO4** is listed in this design.

Using the same methodology of counting the gates as used in the previous adder the total gates are calculated. In table given below relative calculation is given:

Stages	Total number of gates in novel 16bit adder			
Stage 1	**15xA22O2I/O22A2I**	**2x NAND2/NOR2**	**1x INV**	
Stage 2	**7xA2O2I/O2A2I**	**3x NAND2/NOR2**	**4x INV**	**2xA2O3I**
Stage 3	**4xA2O2I**	**4xNAND2/NOR2**		
Stage 4			**8x INV**	
Stage 5	**7 xA2O2I/O2A2I**			
Stage 6	**12xA22O2I/O22A2I**	**23x NAND2/NOR2**	**20 x XOR**	**14xMUX**

Table 13: Listing all the gates used in 16- bit adder with Sparse -2 logic

The table given above shows all different types of adders used in the design but the critical evaluation of this design is shown in the table given below, which employs multiplying gates with their relevant number of grids.

For novel 16- bit adder		
used	**ot$_{n-a}$ = S$_{n-a}$ x N$_g$**	**$_{n-a}$=Tot$_{n-a}$**
NAND2	**16 x 3**	**98**
NOR2	**16 x 3**	**98**
A22O2I/O22A2I	**15 x 5**	**75**
AOI21/OAI21	**19 x 4**	**76**
XOR	**20 x 7**	**140**
INV	**13 x 2**	**26**
MUX	**14 x 7**	**98**
AO1211/OAI211	**12 x 5**	**60**
	ΣS_{n-a} =115	**ΣP_{n-a} =521**

Table 14: Calculating total gates in 16- bit adder using Sparse -2 carry logic

The size of the design in terms of total grids is given as 521. This means that the design is accommodating 115 gates with the total number of 521 grids i.e. this is the size of an adder design.

Here are the equations given in order to understand the building up of this Sparse-2 carry adder .Starting for LAB these equations go to MSB. Each and every bit is explained below. All generate, not kill and propagate term are written with the equations.

As there is only the input bits for S_0 we have it as
$S_0 = P_0 = a_0 \oplus b_0 = (\sim a_0 \bullet b_0) + (a_0 \bullet \sim b_0)$

For S_1 the carry from S_0 has to be propagated and the combination of generate and not kill bit is done in first stage, so $S_1 = P_1 \oplus G_{0:0} \Rightarrow G_{0:0} = g_0$

- $\mathbf{S_1 = ((a_0 \oplus b_0) \oplus (a_0 \bullet b_0))}$

For S_2 again the carry from $R^{(1)}_{1:0}$ is propagated so
$S_2 = P_2 \oplus G_{1:0} \Rightarrow G_{1:0} = g_1 + nk_1\, g_0 \Rightarrow R^{(1)}_{1:0}$ Multiplexer is used as $R^{(1)}_{1:0}$ give different values when it is true or false. The use of sparse tree make this $R^{(1)}_{1:0}$ i.e. also for S_3 the same $R^{(1)}_{1:0}$ signal is used.
$S_2 = P_2 \oplus nk_1\, R^{(1)}_{1:0}$
MUX : $\sim\mathbf{R^{(1)}_{1:0}(P_2)}$ & $\mathbf{R^{(1)}_{1:0}(P_2 \oplus nk_1)}$

For S_3 carry from $R^{(1)}_{1:0}$ with one not kill term missing is propagated so $S_3 = P_3 \oplus G_{2:0} \Rightarrow G_{2:0} = g_2 + nk_2 + nk_{2:1}g_0$ now because of $R^{(1)}_{1:0}$ signal being propagated , the multiplication of $G_{2:2}$ term is multiplied as this term is missing .

- $S_3 = P_3 \oplus g_2 + (nk_2.R^{(1)}_{1:0})$
- **MUX : $\sim\mathbf{R^{(1)}_{1:0}(P_3 \oplus g_2)}$ & $\mathbf{R^{(1)}_{1:0}(P_3 \oplus (g_2 + nk_2))}$**

For S_4 ,the carry from $R^{(1)}_{3:0}$ is propagated so $S_4 = P_4 \oplus G_{3:0} \Rightarrow G_{3:0} = nk_3 \bullet R^{(1)}_{3:0}$ and multiplexer is used to obtain sum bit as $R^{(1)}_{3:0}$ give different values . $R^{(1)}_{3:0}$ goes into for S_4 and S_5 same as $R^{(1)}_{1:0}$ signal .

- $\mathbf{S_4 = P_4 \oplus nk_3\, R^{(1)}_{3:0}}$ where $\mathbf{R^{(1)}_{3:0} = R^{(1)}_{3:2} + (nk_{2:1}\, R^{(1)}_{3:0})}$
- **MUX : $\sim\mathbf{R^{(1)}_{3:0}(P_4)}$ & $\mathbf{R^{(1)}_{3:0}(P_4 \oplus nk_3)}$**

For S_5 carry from $R^{(1)}_{3:0}$ with one not kill term missing is propagated so $S_5 = P_5 \oplus G_{4:0} \Rightarrow G_{4:0} = g_4 + (nk_4 \bullet R^{(1)}_{3:0})$ now because of $R^{(1)}_{3:0}$ signal being propagated , the multiplication of $G_{4:4\backslash}$ term is multiplied as this term is missing .

- $S_5 = P_5 \oplus g_4 + (nk_4\, R^{(1)}_{3:0})$
- **MUX : $\sim\mathbf{R^{(1)}_{3:0}(P_5 \oplus g_4)}$ & $\mathbf{R^{(1)}_{3:0}(P_5 \oplus g_4 \oplus nk_4)}$**

For S_6 carry from $R^{(3)}_{5:0}$ is propagated and we have a group of D terms which reduces the complexity of $R^{(3)}_{5:0}$ term and thus reducing the delay. $S_6 = P_6 \oplus G_{5:0} \Rightarrow G_{5:0} = D_{5:3} \bullet R^{(3)}_{5:0}$ where $\mathbf{R^{(3)}_{5:0} = R^{(1)}_{5:4} + R^{(1)}_{3:2} + (nk_{2:1}\, R^{(1)}_{1:0})}$

- $S_6 = P_6 \oplus D_{5:3} \bullet R^{(3)}_{5:0}$ and the $D_{5:3} = g_5 + nk_5 nk_4 g_4 + nk_{5:3}$ which can be implemented by simple two input gate.
- **MUX : $\sim\mathbf{R^{(3)}_{5:0}(P_6)}$ & $\mathbf{R^{(3)}_{5:0}(P_6 \oplus D_{5:3})}$**

For S_7 , the extra multiplication of $G_{6:6}$ is done as the iput to the mux is $R^{(3)}_{5:0}$. $S_7 = P_7 \oplus G_{6:0} \Rightarrow G_{6:0} = g_6 + nk_6\, D_{5:3} \bullet R^{(3)}_{5:0}$ where $(g_6 + nk_6\, D_{5:3}) = g_6 + D_{6:3}$

- $S_7 = P_7 \oplus g_6 + D_{6:3} \bullet R^{(3)}_{5:0}$
- **MUX : $\sim\mathbf{R^{(3)}_{5:0}(P_7 + g_6)}$ & $\mathbf{R^{(3)}_{5:0}(P_7 \oplus (g_6 + D_{6:3}))}$**

This is the last bit calculated in first stage .Carry from all last 7 bits are propagated in this sum bit. $R^{(3)}_{7:0}$ group with 3 not kill bits is fed to the multiplexer and obtained result is $S_8 = P_8 \oplus G_{7:0} \Rightarrow G_{7:0} = D_{7:5} \bullet R^{(3)}_{7:0}$ where $D_{7:5} = g_7 + nk_7 g_6 + nk_{7:5}$

- $S_8 = P_8 \oplus D_{7:5} \bullet R^{(3)}{}_{7:0}$ where $\mathbf{R^{(3)}_{7:0} = R^{(1)}_{7:6} + R^{(1)}_{5:4} + n_k{}^{(1)}_{4:3} (R^{(1)}_{3:2} + nk_{2:1} R^{(1)}_{1:0})}$
- **MUX :** $\mathbf{\sim R^{(3)}_{7:0} (P_8)}$ **&** $\mathbf{R^{(3)}_{7:0} (P_8 \oplus D_{7:5})}$

S_9 computes the second stage of carry logic where carry from reduced generate groups i.e from $R^{(1)}{}_{1:0}$ to $R^{(3)}{}_{7:0}$ obtained is combined to give subgroups. But according to spase tree $G_{8:8}$ is multiplied with $R^{(3)}{}_{7:0}$ to get the value as $S_8 = P_9 \oplus G_{8:0} \Rightarrow G_{8:0} = g_8 + nk_8 D_{7:5} R^{(3)}{}_{7:0}$ where $g_8 + nk_8 D_{7:5} = g_8 + D_{7:5}$

- $S_9 = P_9 \oplus g_8 + D_{8:5} \bullet R^{(3)}{}_{7:0}$
- **MUX :** $\mathbf{\sim R^{(3)}_{7:0} (P_9 \oplus g_8)}$ **&** $\mathbf{R^{(3)}_{7:0} (P_9 \oplus (g_8 + D_{8:5}))}$

Second logic stage for carry computation in which all the carries from LSB are transferred in $R^{(3)}{}_{7:0}$ and then this signal sometimes from other required signals to give the carry selection for later sum bits. $S_{10} = P_{10} \oplus G_{9:0} \Rightarrow G_{9:0} = D_{9:9} \bullet R^{(1)}{}_{9:0}$ where $D_{9:9} = nk_9$

- $S_{10} = P_{10} \oplus nk_9 \bullet R^{(1)}{}_{9:0}$ where $\mathbf{R^{(1)}_{9:0} = R^{(1)}_{9:8} + Q^{(1)}_{8:5} \bullet R^{(3)}_{7:0}}$ **also** $\mathbf{Q^{(1)}_{8:5}}$ with one not kill missing term reduces the complexity of the input to the multiplexer. $\mathbf{Q^{(1)}_{8:5}} = = g_8 + g_7 + nk_7 g_6 + nk_{7:5}$ and can be represented as a 3 input gate .
- **MUX :** $\mathbf{\sim R^{(1)}_{9:0} (P_{10})}$ **&** $\mathbf{R^{(1)}_{9:0} (P_{10} \oplus nk_9)}$

MSB are the complex terms with high fan out load . $S_{11} = P_{11} \oplus G_{10:0} \Rightarrow G_{10:0} = g_{10} + nk_{10} D_{9:9} \bullet R^{(1)}{}_{9:0}$ where $g_{10} + nk_{10} nk_9 = D_{10:9}$

- $S_{11} = P_{11} \oplus g_{10} + D_{10:9} \bullet R^{(1)}{}_{9:0}$
- **MUX :** $\mathbf{\sim R^{(1)}_{9:0} (P_{11} \oplus g_{10})}$ **&** $\mathbf{R^{(1)}_{9:0} (P_{11} \oplus (D_{10:9} + g_{10}))}$

Carry from $R^{(1)}{}_{11:0}$ is propagated $S_{12} = P_{12} \oplus G_{11:0} \Rightarrow G_{11:0} = D_{11:11} \bullet R^{(1)}{}_{11:0}$, $\mathbf{R^{(1)}_{11:0} = R^{(1)}_{11:8} + Q^{(3)}_{10:5} \bullet R^{(3)}_{7:0}}$ here Q bits are increased to $Q^{(3)}{}_{10:5}$ with 3 not kill bits missing and $\mathbf{R^{(1)}_{11:8} = R^{(1)}_{11:10}} + nk_{10} nk_9 \mathbf{R^{(1)}_{9:8}}$

- $S_{12} = P_{12} \oplus nk_{11} \bullet R^{(1)}{}_{11:0}$
- **MUX :** $\mathbf{\sim R^{(1)}_{11:0} (P_{12})}$ **&** $\mathbf{R^{(1)}_{11:0} (P_{12} \oplus nk_{11})}$

$G_{12:12}$ is multiplied with $D_{11:11} \bullet R^{(1)}{}_{11:0}$ to obtain $S_{13} = P_{13} \oplus G_{12:0} \Rightarrow G_{12:0} = g_{12} + nk_{12} D_{11:11} \bullet R^{(1)}{}_{11:0}$ where $g_{12} + nk_{12} nk_{11} = g_{12} + D_{12:11}$

- $S_{13} = P_{13} \oplus g_{12} + (D_{12:11} \bullet R^{(1)}{}_{10:0})$
- **MUX :** $\mathbf{\sim R^{(1)}_{11:0} (P_{13} \oplus g_{12})}$ **&** $\mathbf{R^{(1)}_{11:0} (P_{13} \oplus g_{12} + D_{12:11})}$

Carries from $R^{(3)}{}_{13:0}$ are propagated ,in order to give $S_{14} = P_{14} \oplus G_{13:0} \Rightarrow G_{13:0} = D_{13:11} \bullet R^{(3)}{}_{13:0}$ where $\mathbf{R^{(1)}_{13:0} = R^{(3)}_{13:8} + Q^{(3)}_{10:5} \bullet R^{(3)}_{7:0}}$ also $\mathbf{R^{(3)}_{13:8} = R^{(1)}_{13:12} + R^{(1)}_{11:10}} + nk_{12:11} nk_{10:9} \mathbf{R^{(1)}_{9:8}}$ **and** $\mathbf{Q^{(3)}_{10:5}} = nk_{10:8} D_{7:5}$

- $S_{14} = P_{14} \oplus D_{13:11} \bullet R^{(3)}{}_{13:0}$
  where **MUX :** $\mathbf{\sim R^{(3)}_{13:0} (P_{14})}$ **&** $\mathbf{R^{(3)}_{13:0} (P_{14} \oplus D_{13:11})}$

This is the last sum bit to combine in a multiplexer as $S_{15} = P_{15} \oplus G_{14:0} \Rightarrow G_{14:0} = g_{14} + nk_{14} D_{13:11} \bullet R^{(3)}{}_{13:0}$ where $g_{14} + nk_{14} D_{13:11} = g_{14} + D_{14:11}$

- $S_{15} = P_{15} \oplus g_{14} + D_{14:11} \bullet R^{(3)}{}_{13:0}$
- **MUX :** $\mathbf{\sim R^{(3)}_{13:0} (P_{15} \oplus g_{14})}$ **&** $\mathbf{R^{(3)}_{13:0} (P_{15} \oplus (g_{14} + D_{14:11}))}$

Carry bit is obtained only at the end with just a use of an nor gate , $S_{16} = \mathbf{G_{15:0}} = D_{15:13} \bullet R^{(3)}{}_{15:0}$ where $\mathbf{R^{(3)}_{15:0} = R^{(3)}_{15:8} + Q^{(5)}_{12:5} \bullet R^{(3)}_{7:0}}$ and $\mathbf{R^{(3)}_{15:8} = R^{(1)}_{15:14} + R^{(1)}_{13:12} + nk}{}^{(1)}_{12:11} \mathbf{(R^{(1)}_{11:10} + nk_{10:9} R^{(1)}_{9:8})}$ also $\mathbf{Q^{(5)}_{12:5} = nk_{12:9} Q^{(1)}_{8:0}}$

3.5 16- bit adder design based on Jackson and Talwar approach with Sparse-4 carry tree logic:

A 16- bit adder is given which uses Sparse-4 carry logic. The adder is made It is 16- bit adder designed with valency 2*3*2 and with the bit group configuration of 4*6*6 from MSB to LSB. As using Sparse-4 carry tree in this Jackson and Talwar adder design the carry group in this adder is derived from every fourth bit position which reduces the fan-out of carries in carry tree logic.

The bit configuration in this is taken 4*6*6 so that design should consider computing two stages almost simultaneously. The stages of 6*6 takes almost same time as both the stages have 3 not kill bits missing and apply same logic of computation. The diagram given below shows the logic diagram of an adder.

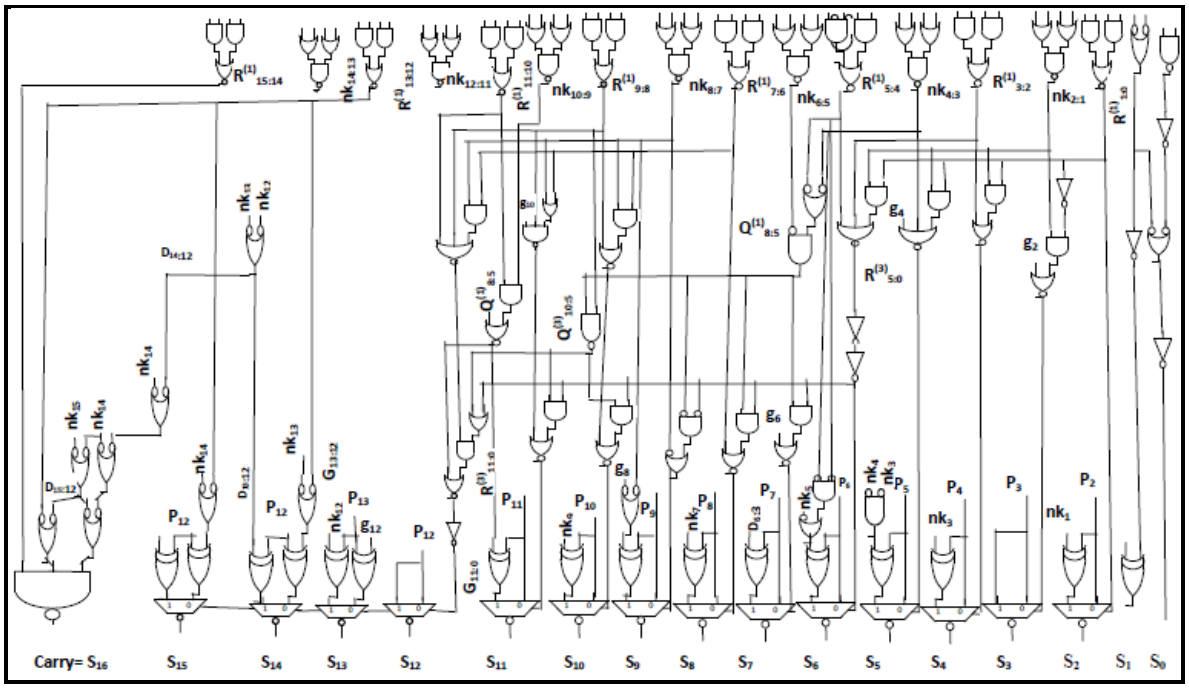


Figure 26 shows 16 bit adder using sparse 4 carry logic

The critical evaluation path is shown below:

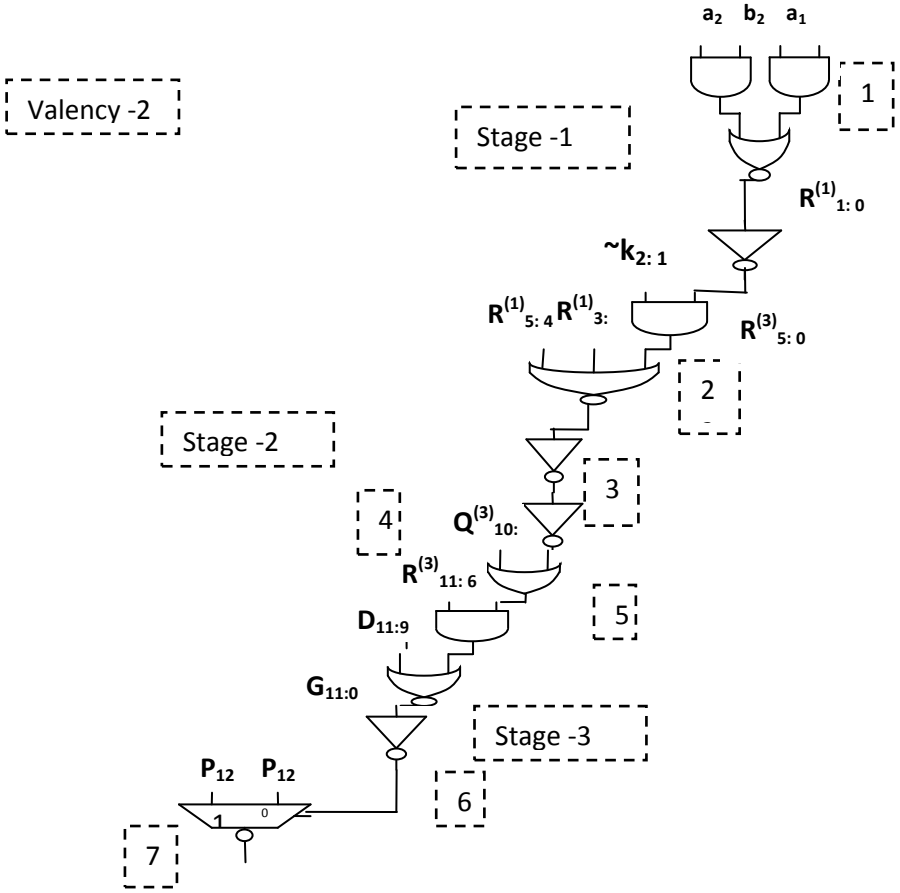


Figure 27: critical path for 16-bit adder using Sparse-4 carry logic

The critical path showed above implements seven stages. Equations for above shown is:

$S_{12} = G_{11:0}$ xor $p_{12} = \sim G_{11:0}$ & $p_{12} + G_{11:0}$ & $\sim p_{12}$

$S13 = g_{12} + nk_{12}$ & $G_{11:0} = G_{11:0}$ & $(g_{12}$ xor $p_{12}) + G_{11:0}$ & $(nk_{12}$ xor $p_{12})$
which could be rewritten as
$\sim G_{11:0}$ $(G_{12:12}$ xor $p_{12})$ & $G_{11:0}$ ($D_{12:12}$ xor $p_{12})$

$S_{14} = g_{13} + nk_{13}g_{12} + nk_{13}nk_{12}\&G_{11:0} = G_{13:12} + nK_{13:12}$ & $G_{11:0}$
$= \sim G_{11:0}\&(G_{13:12}$ xor $p_{12}) + G_{11:0}$ & $(D_{13:12}$ xor $p_{12})$

$S_{15} = g_{14} + nk_{14}g_{13} + nK_{14:13}g_{12} + nK_{14:12}\&G_{11:0} = G_{14:12} + nK_{14:12}\&G_{11:0}$
$= \sim G_{11:0}$ & $(G_{14:12}$ xor $p_{12})$ + $G_{11:0}$ & $(D_{14:12}$ xor $p_{12})$

After the total gates of wide variety are identified, calculations of the total number of gates used in designing an adder are calculated using this table below. In this table, first column has the type of gate which is used N_g = polygrid of referenced gate and S_{S-4} - sum of total number of gates for new adder . P_{S-4} is the sum of total number of gates for new adder and polygrid of referenced gate.

Possibility of Improvement:
The Logic applied for higher bits can be improved by taking some gates common from S_{12} to S_{16}. Reusing of some gates will utilise the not kill signal generated from first stage of reduced groups.

Calculations involved:

Delay calculation:

	Cell	Load	Gb	P
$R^{(1)}_{1:0}$	A22O2I +O22A2I	INV +O2A3I + wire	1+7/3 +3*2/3	12/3
$R^{(1)}_{3:0}$	O2A3I	INV+wire	1 + 2/3	11/3
$R^{(3)}_{7:0}$	INV	4*A2O2I+A2O3I+O2A2O2I+ wire	4*6/3+7/3+7/3+10/3	1
$R^{(3)}_{7:0}$	O2A2O2I	INV+wire	1 +2/3	7/3
$R^{(2)}_{14:0}$	INV	4*Mux +wire	4* 6/3 +6/3	1
S_{15}	Mux	A22O2I+O22A2I+wire	16/3	5
			Πgb=11799	Σp=17

- F= GBH = 11799
- N= Round($\log_4$ F) = 6.71 = 7
- $\alpha = F^{1/N} = 3.8$
- D= (7 * 3.8 +14)/5 = 8.72 FO4

Total number of gates used**:**

Stag	Total number of gates in novel			
Stage 1	15xA22O2I/ O22A2I			
Stage 2	7xA2O2I/O2 A2I	2x NAND2/N OR2	3x INV	4xA2O3I/ O2A3I
Stage 3	1xA2O2I	3xNAND2/ NOR2	3x INV	
Stage 4	1X O2A2O2I		2x INV	
Stage 5	4 xA2O2I/O2A 2I	7X NAND	1X A2O3I	
Stage 6	16 x XOR	35x NAND2/N OR2	17 x mux	

Table 15: Different type of gates in Sparse-4 adder

This table presents the calculation of counting the total grids and gates in the designs.

Gates used	$Tot_{S-4} = S_{S-4} \times N_g$	$P_{S-4}=Tot_{S-4}$
NAND2	13 x 3	39
NOR2	24 x 3	72
A22O2I/O22A2I	15 x 5	75
A2O2I/O2A2I	11 x 4	44
XOR	16 x 7	112
INV	8 x 2	16
MUX	17 x 7	119
A2O3I/O2A3I	4 x 4	16
O2A2O2I	1 x 5	5
	ΣS_{S-4} =109	ΣP_{S-4} =498

Table 16: Total number of gates in 16- bit adder with Sparse 4 tree logic

The total number of gates used in this design is 109 and the total grids are 498 which is comparatively smaller than other designs.

3.6 Design methodology of an adder:

In this section the adder implemented consist of three stages for calculating the sum. The points mentioned below were experienced practically while designing different adders presented in this thesis. Here are some key points disclosed for designing an adder.

1. First stage consists of deriving the bit generate and bit propagate terms. A 2- bit binary number is supplied as an input to a gate for computing all single bits. The gate here can be used as A22O2I or O22A2I gates over 2- bits each for generating terms for R terms. For example if we have a 16-bit adder then the generation of $R^{(1)}_{15:14}$ - $R^{(1)}_{1:0}$ with missing one not kill bit is evaluated in first stage.

2. Second stage is used for calculating carry tree logic as it converts the generated groups in first stage into the reduced generate groups .In this stage the reduced generate group is combined up to valency 4 or valency 3 equations such as $R^{(3)}_{7:0}$ with 3 missing not kill bits .Removing of not kill bit from Generate group(G) thus forming reduced group(R) means that the removed not kill bit will be present in D term. Depending upon the valency, desired gate can be used. For example, valency-2 does not require any not kill bit to factor out as the logic can be implemented using single gate. For valency 3-4 we can use single gate but with some bits factored out in D term. Different logic gates can be implemented for generating reduced terms. It is beneficial to use low valency gates because high valency gates like valency 6 requires two level of logic where more number of bits can be factored out. With increasing number of bits factored out, the complexity of D term increases which might upshot the critical path through this D term thus increasing delay in the design.

3. Third stage contributes for calculating sum in a design. In this stage the reduced term is added to sum logic at every single bit position. An EX-OR gate is employed having one input as the propagate bit and the second input comprises of carry into that bit position. A multiplexer is employed and depending on the polarity of input, it has one inverted input and other not inverted. The inverted input is driven faster than not inverted and this is why slow gate like EX-OR are attached on the inverted input.

4 Experimental Results:

The result table given below shows all the results obtained for the adders designed with Jackson and Talwar approach. Also reflects the result of adder designed by merging Jackson and Talwar's approach with Sparse -2 and Sparse-4 carry logic. All the designs are 16-bit designs and are strictly based on the taken approach taken. Manual calculation is used for calculating delay and the size of the designs.

Note: Comparing delay and size with Knowles 16-bit conventional adder(as discussed in section 1.8) in the table shows the improvement gained in the designs obtained in this project.

Type	Bit Configuration	Delay(FO4)	Size(number of grids)
J & T	8*8	8.81	544
J& T with Sparse-2	8*8	8.81	521
J & T with sparse -4	4*6*6	8.7	504
16- bit Knowles adder	8*4*2*1	10.7	601
16- bit Preeti's adder	8*8	9.07	586

Explanation of Results:

1. The design obtained by implementing the approach of Jackson and Talwar with low valency of 2*3*2 carry tree showed a drastic improvement of almost 1 .5 FO4 in delay. The size of an adder was almost decreased by 69 gates. As verified by comparing the above two adders i.e. Knowles adder and new J &T adder it's been observed that the number gates in the latest L – F are 198 and the sum of total grid depicting the size of an adder is 601 whereas for novel adder the number of gates are 120 and the number of grids are 532 saving 69 gates and hence proving the 16 – bit new J & T adder to be smaller and faster than other latest adders.
2. The second 16- bit adder which is merging J & T with Sparse carry-2 came out to be smaller and faster than the conventional Knowles adder. But comparing the new designs i.e. 16- bit adder J & T and 16- bit adder J & T with sparse-2 , than it can be observed that using Sparse two definitely helps in reducing the size of an adder but the delay difference is not much .It gives almost same delay as 16- bit adder J & T adder.
3. The 16- bit adder J & T adder with sparse -4 having the bit configuration of 4*6*6 proved out to be the best way of designing the adders. The results showed delay to be reduced by a considerable margin and the number of gates used was relatively less than that used for other two designs discussed.

5 Critical evaluation:

The aim of the project was to design a faster and smaller adder in VLSI using CMOS technology. The above mentioned aim is achieved by designing 16- bit adder which is fast and small. Achievement of the aim is done by sequentially considering different aspects of designing. Various steps leading to success for this project are listed. Also, the choices made, owe an explanation for being the best options are provided below.

1. The choice of taking this project was done because Jackson and Talwar approach is one of the best techniques that can help in designing comparatively fast and small adders. This approach provides flexibility in designing as it balances both, the reduced generate group and the not kill group further reducing the delay and fan-out in critical path of an adder design making it faster and smaller. Ling Technique could also be used and it is used for designing fast adders but Jackson and Talwar approach is a new approach which was not yet proved. So, taking Jackson and Talwar approach gave me a challenge of proving the theory.

2. An important decision was made by choosing Ladner - Fischer adder instead of Kogge- stone adder. According to the published paper of Knowles (Knowles S. , 1999) Ladner – Fischer and Kogge-stone adder are the only two adders which are chief among the known construction of other adders as they share common property of minimum logical depth. Then in the thesis conducted by one of the M.Sc student of Bristol University proved that in Ladner Fischer adder the logic levels for computing the carry group is kept minimum. The depth of $\log_2 n$ is considered which reduces the delay better than what Kogge-stone does. This being the reason Ladner -Fischer adder over runs Kogge – stone adder. And it is taken as the base design for comparison in this thesis.

3. While designing, low Valency gates such as valency 2-3 are used. Using low valency cells give better reduction in fan-out between the carry generated groups. Also low valency cell contributes in reducing the overall size of an adder by decreasing the amount of load used for driving gates. Instead of using Low valency the employment of high valency gates can be done. But it is observed that using low valency gates is more beneficial than using high valency as low valency the number of grids in a gate will also be less which will contribute for reducing the size of a full circuit.

4. The technique used for calculating the speed of an adder is Logical effort. This is the most appropriate method for evaluate the designing of adder in VLSI area model as it considers all different capacitances involved in gates and wiring in a circuit. The algorithm of Logical effort traces the reducibility and idempotency of the tree topologies. Instead of this method the speed of adder can be tested by using Cadence tool. But due to the limited time span, simulation of an adder using Cadence was not possible.

5. For comparing the size of new adder, a tabulated comparison of the total number of overall gates in a new design with the total number of gates in conventional 16- bit Ladner-Fischer adder is done . Depending on the theory that the count of the total grids defines the size of a gate. Grids are the vertical spaces covering Polysilicon gate which are not scaled to paper but can be easily counted. The logic gates in a design are multiplied with their polysilicon grids present in a logic cell. A

comparison of the total number of gate size in a design reflects the overall size of an adder. Instead of comparing manually, an actual layout of an adder can be designed using cadence tool but again due to limited time, use of tool was not feasible.

6. Further merging of sparse carry tree is merged with Jackson and Talwar approach. Implementing this concept of merging was definitely the best option as it resulted in a much smaller and faster adder, even as compared to Jackson and Talwar adder. The result obtained clearly reflects the advantages of implementing sparse carry tree logic. In this logic carry from every fourth bit is supplied instead of every single bit which reduces the fan-out in the design. Sparse carry tree is a broad concept which can be further optimized to give remarkable results.

6 Conclusions:

After working on this thesis some important points are clear. With the consideration of designs given in the thesis, the results satisfied are concluded below:

1. The first design of 16-bit adder proves that the documented theory Jackson and Talwar approach presented in the paper "High Speed Binary Addition" can be practically applied to all the stages in designing of an adder. A 16- bit adder is designed implementing Jackson and Talwar approach is designed considering the bit groups of 8*8 and the valency of 2*3*2. In this adder the method used factorises a kill bit from every stage further reducing the complexity of the carry term thus making the critical path faster in a circuit designed. Reducing the complexity of the critical path increases the complexities of other terms but these complexities can be unbiased following a faster adder. As an obtained result this 16- bit adder was proven to be **smaller and faster** than Knowles 16- bit adder. Knowles 16-bit adder is the conventional design used for adder presently.
2. Second design of an adder implements Jackson and Talwar Approach merging with Sparse-4 carry tree. It is a 16- bit adder designed with valency 2*3*2 and with the bit group configuration of 4*6*6 from LSB to MSB. By using the Sparse-4 carry tree in this Jackson and Talwar adder design the carry group in this adder is derived from every fourth bit position which reduces the fan-out of carries in the carry tree logic. As a result this 16 –bit adder came out to be relatively **faster and smaller** as compared to my first design of 16- bit designed with Jackson and Talwar approach.

3. Third design of 16- bit adder implements Sparse-2 carry tree. This adder was designed to examine whether Sparse-2 carry tree can be implemented or not as Sparse-2 carry tree is not proposed yet by anyone in this field. This 16- bit adder is designed by taking bit groups as 8*8 from LSB to MSB and the valency of 2*3*2 again. So now the Sparse-2 carry tree was examined but also the bit groups configuration was chosen to be different than the previous Sparse-4 carry tree adder. The carry in this adder was taken from every second bit position instead of every fourth bit position. As a result the number of gates utilised for **designing were reduced** but it did not have much effect on delay as compared to Jackson and Talwar adder described the first 16- bit adder design. Thereby concluding that Sparse-2 carry tree is not better than Sparse-4 carry tree as the results obtained showed same size of second and third designs and also the delay calculated was more than Sparse-4 carry tree i.e second 16- bit adder design.

4. While designing the adder it was observed that using low valency cells give better reduction in fan-out between the carry generated groups. Also low valency cell contributes in reducing the overall size of an adder by decreasing the amount of load used for driving gate.

The graph shown below declares the improvement gained in the designs starting from Knowles adder till Jackson and Talwar adder design merged with Sparse-4 carry tree method. Graph comparing the progress of Knowles, Ling, Jackson and Talwar , Jackson and Talwar with sparse.

7 Future work:

- The design of 16- bit adder can be implemented at architectural level testing the functionality of a design using languages like C or system Verilog language for hardware description.
- EDA tool can be used to simulate the new adder designs laid out with the appropriate placement of logic cells used and the net-list of the design can be obtained. Using the net-list and various desired constraints a layout of a adder design can be generated.
- The obtained designs can be further tested for **Power** as power consumption is an important parameter and is not been discussed upon.
- Adder with wider number of bits like 24 bits or 32 bits can be designed by again merging Jackson and Talwar approach with sparse tree. The methods, Jackson and Talwar approach and sparse tree have much more further scope of optimization which will be helpful in designing adders.
- The fast adders can be further used in digital signal processing , graph computations etc.

8 Bibliography

1. A. B. Smitth and C.C.Lim. (2001). Parallel Prefix Adder designs. *IEEE* .
2. Burgess, N. (2009). Implementation of recursive Ling adders in CMOS VLSI. *Signals, Systems and Computers, 2009 Conference Record of the Forty-Third Asilomar Conference on* , 1777 - 1781.
3. Burgess, N. (2005). New Models of Parallel Prefix Adder Topologies. *Journal of VLSI Signal Processing 40* , 125-141.
4. Burgess, N. (2009). Recursive Ling adders. *Journal of VLSI signal* .
5. Choi, Y. and Earl E swartzlander Jr. . (n.d.). Parallel Prefix adder Design with matrix presentation. *Sun microsystems* .
6. G. Dimitrakopouloas and D. Nikolos. (February 2005.). High Speed Parallel Prefix VLSI Ling Adders. *IEEE Transaction on Computers, VOL.54, No. 2* .
7. *google webpage*. (n.d.). Retrieved july 9, 2010, from google: http://www.ece.cmu.edu/~ee760/760docs/mcfarland-lingadder.pdf
8. *google webpage*. (2010). Retrieved june 12, 2010, from google: http://www.writphotec.com/mano4/Supplements/Carrylookahead_supp4.pdf
9. H.Ling. (1981). High Speed Binary adder. *IBM Journal of Research and Development* , Vol. 25 , No.3.
10. Harris, D. (2010). *Jackson Adders.* Private source.
11. Knowles, S. (1999). A Family of Adders. *14th IEEE symposium on Computer Arithmetic* .
12. Knowles, S. (April 1999). Family of Adders. *IEEE Symposium on Computer Arithmetic* , 30-34.
13. Lin, C. C. (2003). *Half adder , Full adder , Ripple carry adder.* Retrieved July 15, 2010, from google: http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Comb/adder.html
14. Neil H.E Weste and David Harris. (2005). CMOS VLSI Design. pp. 655 and 638-639,: Pearson Education.
15. P.M. Kogge and H.S. Stone. (1973). A parallel Algorithm for Efficient Solution of a General Class of Recurrence Equations. *IEEE Trans. Computers, Vol. C-22 , No.8* , 786-793.
16. Patil, P. (2007). High Valency Recursive Ling Adder Assessment. *University of Bristol* .
17. R.C.Jackson and S. Talwar. (Nov 2003). High Speed Binary Adder. *IEEE Journal* .
18. R.E. Ladner and M.J. Fischer. (1980). Parallel Prefix Computation. *Journal of ACM* , 831-838.

19. Sanu Mathew,Mark Anders,Ram K. Krishnamurthy and Shekhar Borkar. (MAY 2003). A 4-GHz 130-nm Address Generation Unit with 32- bit sparse tree adder core . *IEEE JOURNAL OF SOLID-STATE CIRCUITS, VOL. 38, NO. 5, .*

20. Sutherland, I. (1999). *Logical effort.* USA: Morgan Kaufmann.

21. Sutherland,Sproull, Harris. (1999). *Logical Effort.* United States of America: Morgan Kaufmann Publishers.

22. Vazquez, A.; Antelo, E. (2008). New insights on Ling adders. *Application-Specific Systems, Architectures and Processors, 2008. ASAP 2008. International Conference on* , Page(s): 227 - 232.

23. weste, h. a. (2005). *CMOS VLSI.* boston.

24. Y.Wang and K.K Parhi. (2010). *A Unified Full adder Design*. Retrieved july 25, 2010, from IEEE: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00986901

25. Yu-Ting Pai and Yu-Kumg Chen. (2004). The Fastest Carry Lookahead Adder. *Second IEEE International Workshop on Electronic Design, Test and Applications (DELTA'04)* .

9 Appendix

Preeti's 16 bit adder:

	Cell	Load	Gb	P
$Q^{(1)}_{2:1}$	O22A2I	O3I+A2O3I+Wire	7/3+7/3+2/3	12/3
$R^{(3)}_{7:2}$	A2O3I	O2I+wire	5/3+2/3	11/3
$R^{(3)}_{7:0}$	O2I	Inv +wire	3/3 +2/3	6/3
$R^{(3)}_{7:0}$	Inv	8 * A3O2I +wire	8* 7/3 +16/3	3/3
$R^{(2)}_{14:0}$	A3O2I	Mux2 +wire	9/3+4/3	8/3
S_{15}	Mux2	O22A2I + A22O2I+wire	6/3+6/3+ 4/3	1+12/3
			Πgb=12492.2	Σp=18.33

- F=GBH =12492.2
- N= round($\log_4$F) = 7 we need 1 more gate(an inverter).
- $\alpha = F^{1/N} = 3.87$
- D= (7 * 3.87 +18.33)/5 = 9.03FO4