DESIGN AND SIMULATION OF 32 BIT ALU BASED ON FEEDBACK SWITCH LOGIC

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree of

MASTER OF TECHNOLOGY

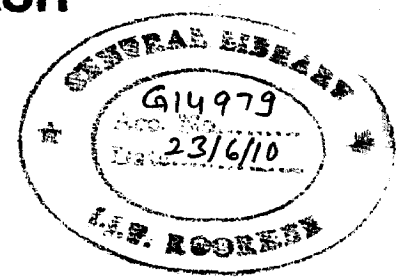
in

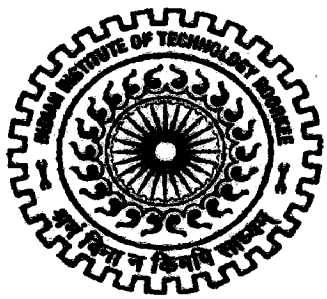
ELECTRONICS AND COMMUNICATION ENGINEERING

(With Specialization in Semiconductor Devices & VLSI Technology)

By

PATANJALI PRAKASH





DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

ROORKEE -247 667 (INDIA)

JUNE, 2009

CANDIDATE'S DECLARATION

I hereby declare that the work, which is being reported in this dissertation report, entitled **"Design and Simulation of 32 bit ALU based on Feedback Switch Logic"**, is being submitted in partial fulfillment of the requirements for the award of the degree of **Master of Technology** in **Semiconductor Devices and VLSI Technology**, in the Department of Electronics and Computer Engineering, Indian Institute of Technology, Roorkee is an authentic record of my own work, carried out from June 2008 to June 2009, under the guidance and supervision of **Dr. A. K. Saxena**, Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology, Roorkee.

The results embodied in this dissertation have not submitted for the award of any other Degree or Diploma.

Date : 24-06-2009
Place : Roorkee

Patanjali Prakash
Patanjali Prakash

CERTIFICATE

This is to certify that the statement made by the candidate is correct to best of my knowledge and belief.

24/6/09

Dr. A. K. Saxena

Professor

ACKNOWLEDGEMENT

At the outset, I express my heartfelt gratitude to Dr. A.K Saxena, Professor, Department of Electronics and Computer Engineering at Indian Institute of Technology Roorkee, for his valuable guidance, support, encouragement and immense help. I consider myself extremely fortunate for getting the opportunity to learn and work under his able supervision. I have deep sense of admiration for his innate goodness and inexhaustible enthusiasm. It helped me to work in right direction to attain desired objectives. Working under his guidance will always remain a cherished experience in my memory and I will adore it throughout my life.

My sincere thanks are also due to rest of the faculty in the Department of Electronics and Computer Engineering at Indian Institute of Technology Roorkee, for the technical knowhow and analytical abilities they have imbibed in us which have helped me in dealing with the problems I encountered during the project. I also extend my sincere thanks to all the technical and non-technical staff of VLSI Design Lab for providing me various tools and encouraging me through out my work.

I am greatly indebted to all my friends, who have graciously applied themselves to the task of helping me with ample morale support and valuable suggestions. Finally, I would like to extend my gratitude to all those persons who directly or indirectly helped me in the process and contributed towards this work.

Patanjali Prakash
M. Tech. (SDVT)

ABSTRACT

This thesis presents the employment of Feedback Switch Logic (FSL) in the development of a 32-bit ALU unit. For the design of ALU, we have chosen Feedback Switch Logic (FSL) because it offers reduced capacitance, fast switching and input-switching dependent activity factor without the need of clock connection.

The Arithmetic Logic Unit is a digital circuit that performs an arithmetic operation (addition, subtraction, etc.) and logic operations (Exclusive-OR, AND, etc.) between two numbers. Demand for performance at low power consumption in today's general purpose processor has put severe limitations on ALU design. ALU are also one of the most power consumed blocks in the processor and are often the possible location of hot-spots. Hence this thesis aims to reduce power consumption and improve performance using FSL.

Three types of adders, two types of shifter structures and one logical unit have been designed in FSL and static CMOS logic styles. Comparisons are drawn among the various designed units and the best one in terms of high speed and low power is chosen. Finally we have done the proper organization of adders, shifter and logic unit to make complete ALU.

Simulations have been performed in CADENCE Virtuoso Front to Back Design Environment on 90nm technology node. Simulation results show that 14% increase in speed has been achieved with FSL trading-off with an 8% increase in power consumption when compared to static CMOS logic.

CONTENTS

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Meaning
ALU	Arithmetic Logic Unit
VLSI	Very Large Scale Integration
IC	Integrated Circuit
DCT	Discrete Cosine Transform
FSL	Feedback Switch Logic
CMOS	Complementary Metal Oxide Semiconductor
CVSL	Cascode Voltage Switch Logic
DCVSL	Differential Cascode Voltage Switch Logic
RCA	Ripple Carry Adder
LSB	Least Significant Bit
MSB	Most Significant Bit
CLA	Carry Look-ahead Adder
DSP	Digital Signal Processing
MUX	Multiplexer

Chapter 1
Introduction

1.1 Background

Much of the research effort of the past years in the area of digital electronics has been directed towards increasing the speed of digital systems. But due to the demand and popularity of portable electronics, designers are striving to achieve smaller silicon area, higher speeds, longer battery life and more reliability. Power is one of the premium resources to which a designer tries to save when designing a system. However, the advancement of digital computer technology requires higher circuit speed also.

The three most widely accepted parameters to measure the quality of a circuit or to compare various circuit styles are area, delay and power dissipation. Portability imposes a strict limitation on power dissipation while still demanding high computational speeds. Hence, in recent VLSI systems, the power delay product becomes the most essential metric of performance. The reduction of power dissipation and the improvement of speed requires optimization at all levels of the design procedure.

Arithmetic Logic Units (or ALUs) are important components in many moderns IC such as processors and re-configurable cores. Demand for performance at low power consumption in today's general purpose processors has put severe limitations on ALU design. ALUs are also one of the most power hungry sections in the processor and are often the possible location of hot-spots. The presence of multiple ALUs in superscalar pipelines further deteriorates the power and thermal issues. Technology scaling has resulted in faster devices but at the same time, the die-to-die delay variations have increased due to different lithographic subtleties. Therefore, low power ALU design while maintaining high yield under tighter delay constraint turns out to be a multidimensional problem [1].

ALU is the heart of a microprocessor and the adder cell is the elementary unit of an ALU. It is basically a combinational circuit that performs a number of arithmetic and logic functions. Hence, its performance is crucial for the design of high performance digital computer system. The basic structure of the ALU is shown in Fig. 1.1. It consists of input

multiplexers, adder core and an output multiplexer. The adder takes operands from register file, data cache or ALU write-back bus. The input multiplexers select the proper operands among these and provide the ALU inputs. The adder output is multiplexed with the logical output through an output multiplexer.

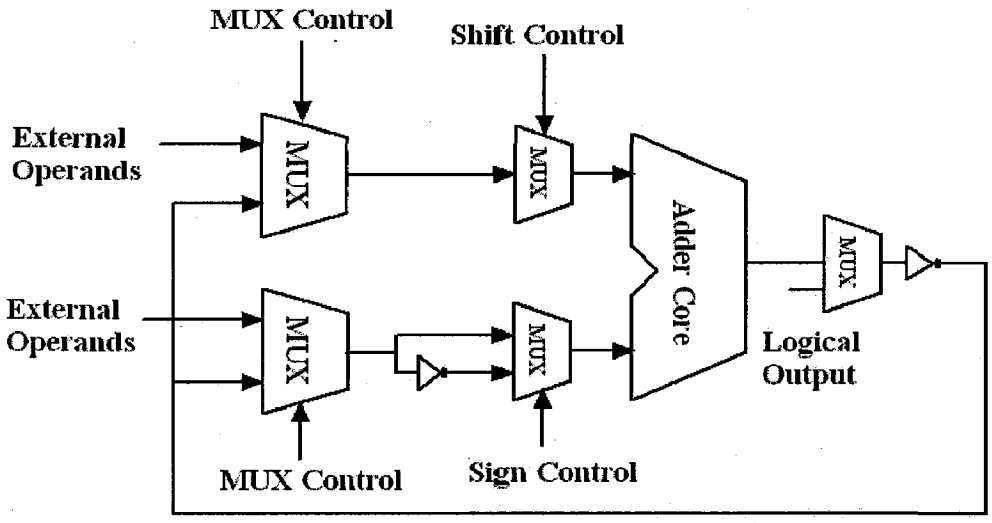


Fig. 1.1: Basic ALU Structure [1].

Since multiplication is a less frequent operation than addition/subtraction/shift operation, the multipliers are usually isolated from the ALU. This also ensures high speed operations of the more frequent instructions in processors. However, for signal processing applications (e.g., filters, DCT), multipliers can be an integral part of the ALU.

1.2 Thesis Contribution

The aim of this thesis is to design a 32-bit low power, high speed logic ALU based on Feedback Switch Logic (FSL) [2]. FSL offers low power consumption and high speed because of fast switching, reduced capacitance and input-switching dependent activity factor without the need of clock connection. The objectives of the thesis are following:

- Study the concept of low power high speed logic, which is Feedback Switch Logic (FSL).
- Design different adders using static CMOS logic and FSL.
- Estimation and comparison of power consumption and delay of different adders with static CMOS logic and FSL logic.
- Design of various shifters using static CMOS and FSL logics.

- Extraction and comparison of power consumption and delay of shifter with both the logics.
- Design of 32-bit ALU by taking best adders and shifters from comparisons in terms of speed and power consumption using static CMOS and FSL logics.
- Finally, power consumption and delay of 32-bit ALU using static CMOS and FSL logics are calculated and compared.

1.3 Thesis Organization

The thesis organization is as follows:

Chapter 2 gives an overview of FSL. This chapter presents the overview of our new logic Feedback Switch Logic. A comparative study between various logics is done, followed by various simulation results and discussions.

In chapter 3, designs of different adders using static CMOS and FSL are presented. It also discusses about comparison results of different adders using both the logics.

Chapter 4 presents design of different shifter units. These shifters are capable of bidirectional and cyclic shifting up to 31-bits and it also presents simulation results and comparison of these shifters using static CMOS logic and FSL in terms of speed and power consumption.

In chapter 5, implementation details of 32-bit ALU have been discussed and simulation results are also presented.

Chapter 6 concludes the thesis and discusses future scope of work.

Chapter 2

Feedback Switch Logic (FSL)

2.1 Introduction

Since the invention of CMOS, speed improvement has been the main goal in the research of both circuit design and device design. Designing high speed low power circuits with CMOS technology has been a major research problem for many years. Several logic families have been proposed to improve circuit performance beyond static CMOS family [3].

In deep submicron technologies, the performance benefits obtained from process scaling decreases as feature size decreases, hence fast circuit families become more attractive now a days. Static CMOS logic is widely used due to its features like low power dissipation, ease of design and higher stability. Static CMOS logic gate is built up by pull up (PUN or PMOS network) and pull down network (PDN or NMOS network). The advantage of having both pull up and pull down networks is that except for the very brief period when the output or the inputs are making transitions, no current flows and no power is consumed. The problem with this fully complementary approach is that for complex gates, substantial amounts of area can be wasted. As a result of the extra area and extra transistors, the capacitive loads on gates of a fully complementary circuit are very high. Each output goes to both a PMOS and NMOS transistors in every gate it drives. PMOS transistor are generally twice the size of NMOS transistor to obtain more balanced rise and fall times. As a result, the total gate load on each output will be three times higher. Hence even though static CMOS consumes less power and gives good noise margin but it slows down the circuit speed.

In order to remove the above problems, many dynamic circuit schemes have been described [4], but they all show some basic features as a common one. Basically, they involve pre charging the output node to a particular level (usually high for NMOS), while the current path to the other level (ground for NMOS) is turned off. Any charging of inputs to the gate must occur during the pre charge phase. At the completion of pre charge, the path to the high level is turned off by a clock and the path to ground is turned

on. Then depending on the state of the inputs, the output will either float at the high level or will be pulled down. The advantage of dynamic circuit is that it has lesser load capacitance on gates than static CMOS logic. In addition, there is no static current path, so power would be much closer to CMOS. However, there are serious problems involved in realizing these apparent speed advantages in real circuits. This happens because useful circuits generally have several logic gates in series and in the dynamic approach, no gate can be activated until its inputs have been stabilized. In practice, considerably more than one gate delay would be needed between successive edges to assure a full gate delay in worst case. Overall then, in a circuit of reasonable complexity, the dynamic approach would not be any faster than conventional CMOS logic [4].

A Domino logic consists of an n-type dynamic logic block followed by a static inverter, so only noninverting logic can be implemented. In domino logic, during precharge, the output of the n-type dynamic gate is charged up to V_{dd}, and the output of the inverter is set to 0. During evaluation, the dynamic gate conditionally discharges and the output of the inverter makes a conditional transition from $0 \rightarrow 1$. Due to static inverter, it has additional advantage that the fan-out of the gate is driven by a static inverter with a low-impedance output which increases noise immunity [5].

Domino logic allows high-speed circuit design due to the low switching threshold of dynamic gates and their reduced load capacitance and also the ability to implement complex functions in a single gate. On the other hand, domino logic suffers from high switching activity since the activity factor of a dynamic gate is input-state dependent and the gate switches twice a cycle. However, unlike static logic, domino logic does not allow false outputs before a gate settles down to its correct output which helps in reducing the activity overhead of domino circuit. One of the drawbacks of domino logic is the inability to recover after noise upsets. Noise tolerant pre charge solves this problem at the price of increased gate capacitance [6]. Another major disadvantage of domino logic is the increased clock loading and the need to deliver the clock signal to every gate, which in turn increases power, routing area and design complexity.

However, the inability of inversion in domino logic poses another major limitation on widely using this high speed circuit family. Dual rail domino, which is a derivative of the differential Cascode Voltage Switch Logic (CVSL) family [7], solves this problem by implementing the logical function and its complement.

The static version of Differential Cascode voltage switch (DCVSL) is a differential style of logic that provides the complementary outputs with true and complementary inputs to the gate. When the input switches, output of the DCVSL gate and its complement are pulled either high or low. This static version slowly transits and highly consumes current since the PMOS pull-up fights the NMOS pull-down trees during the switching period. To increase the performance and reduce the power consumption, many clocked versions of the DCVSL gate have been introduced [7].

One of clocked CVSL is a dual-rail domino logic which combines both domino and CSVL logic in order to solve the problems of both families. Dual-rail domino does not suffer from contention problems, which makes it as fast as standard domino. Also, dual rail domino logic provides both inverting and non-inverting functions, which makes it easy to use in digital logic design. The main disadvantage of dual rail domino gate is its unity activity factor since an evaluate/precharge transition is guaranteed at every cycle regardless of the input activity or input states. Therefore, dual-rail domino logic suffers from high power consumption due to the extra clocking power. Also, dual-rail domino can not recover from noise upsets similar to standard domino logic.

Feed back switch logic [2] is an improved design among all these logic families. This is a dynamic like static circuit family suitable for high speed and low power circuit designs. It is free from contention problem of clocked CVSL. The major advantage of FSL is fast switching speed, reduced capacitance, and input-switching dependent activity factor without the need of clock connection.

FSL is a clock less differential circuit family that provides the output and its complement from a single side of the gate. Fig. 2.1(a) and (b) present two possible structures of FSL gates. In this work, FSL structure shown in Fig. 2.1(a) has been considered. However, the structure in Fig. 2.1(b) can be useful for FSL tree implementations since it allows transistor sharing between complementary networks. Similar to CVSL, the logic function and its complement in FSL are implemented using NMOS networks. Also, two cross coupled PMOS pull-ups are employed. However, one of these pull-ups is a weak keeper and the other one is a strong output driver. The latter provides high drive current during a low-to-high $\overline{OUT}$ transition, while it does not affect the high-to-low transition due to the output feedback that turns it off when it is not needed. The feedback acts as a switch that

decides which of the two pull-down networks should be used to provide the functionality of the gate based on the current output state.

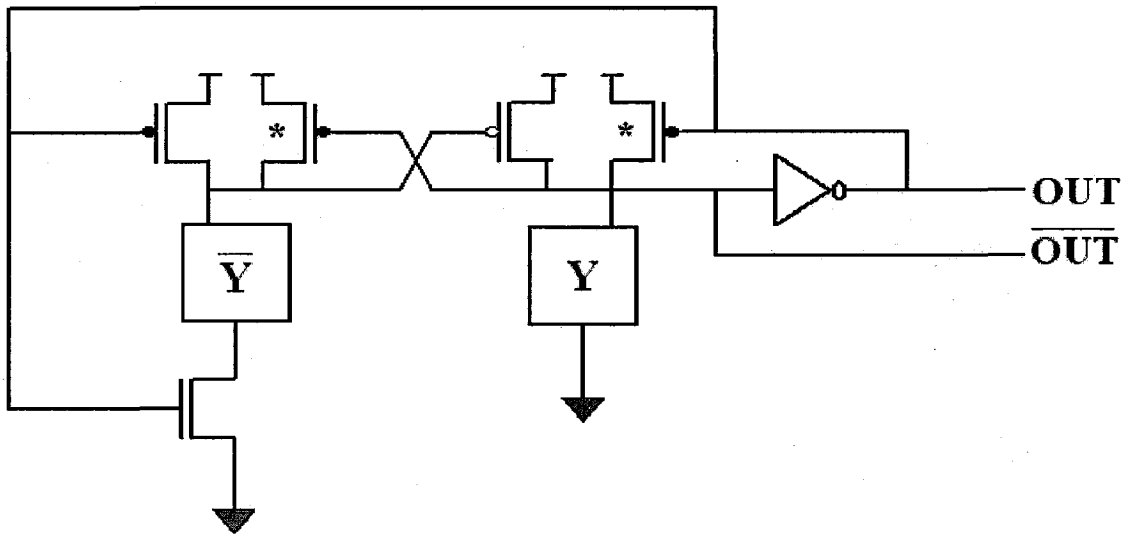


Fig 2.1(a): A Possible Structure 1 of FSL Logic [2].

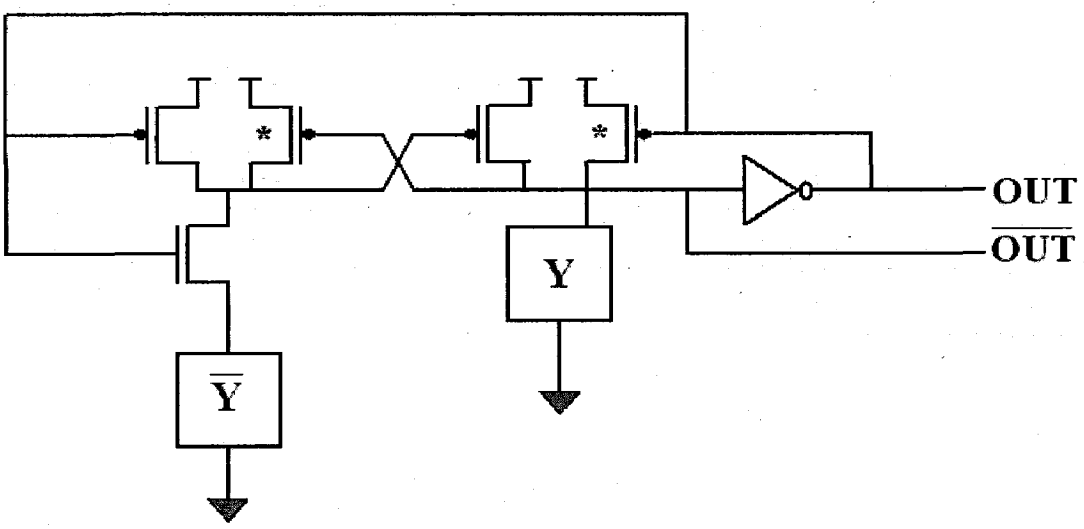


Fig 2.1(b): A Possible Structure 2 of FSL Logic [2].

FSL takes the advantage of the fact that low-to-high transitions are the speed-critical transitions for all CVSL circuit families since only NMOS networks are used, while high-to-low transitions are less important in terms of speed. Therefore, the high-to-low transition can follow its complementary low-to-high transition, while the speed is effectively based on the earlier low to-high transition. Thus, even though a signal in an FSL gate can pass through three serial stages in the worst case, which are $\overline{OUT}$, OUT, and the output inverter stage, the effective FSL gate delay is always based on two stages. FSL

offers the switching speed of dynamic logic, while providing activity-dependent switching behavior similar to static logic without the need of a clock connection. However, FSL is slower than dynamic logic since the load of a gate is driven from a single side.

2.2 Operation and Analysis:

There are two possible FSL structures for any basic logic gate. In Fig 2.2 (a-d) two-inputs NOR/OR and NAND/AND gates based on FSL are shown. To understand the operation of FSL based gate, consider figure 2.2(a).

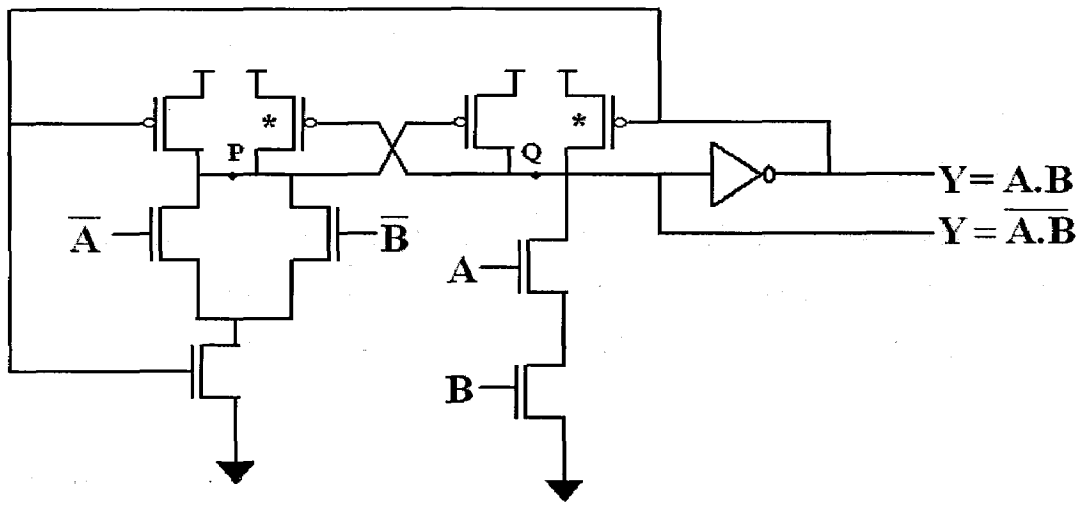


Fig 2.2(a): A NAND/AND gate based on Structure 1 of FSL Logic.

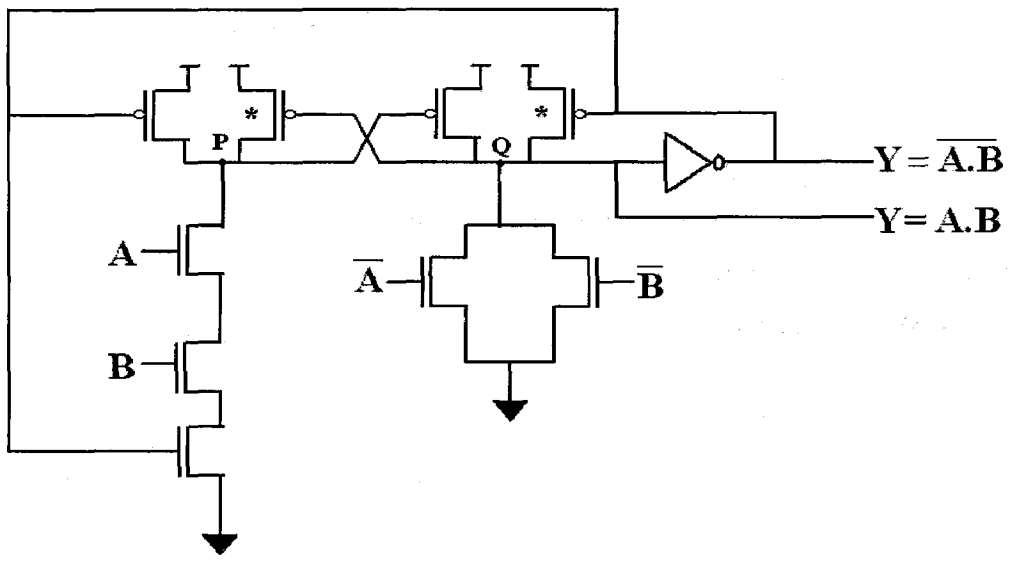


Fig 2.2(b): A NAND/AND gate based on Structure 2 of FSL Logic.

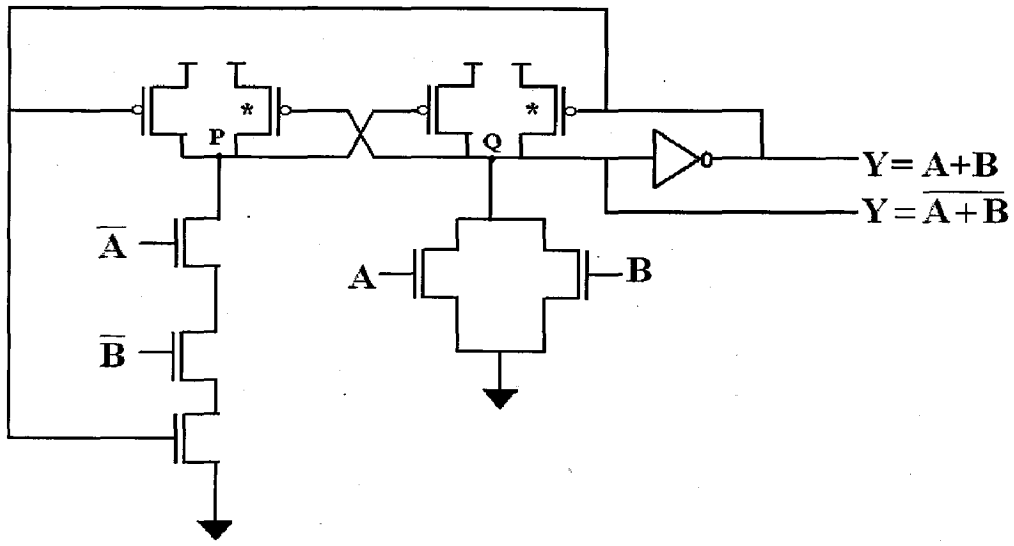


Fig 2.2(c): A NOR/OR gate based on Structure 1 of FSL Logic.

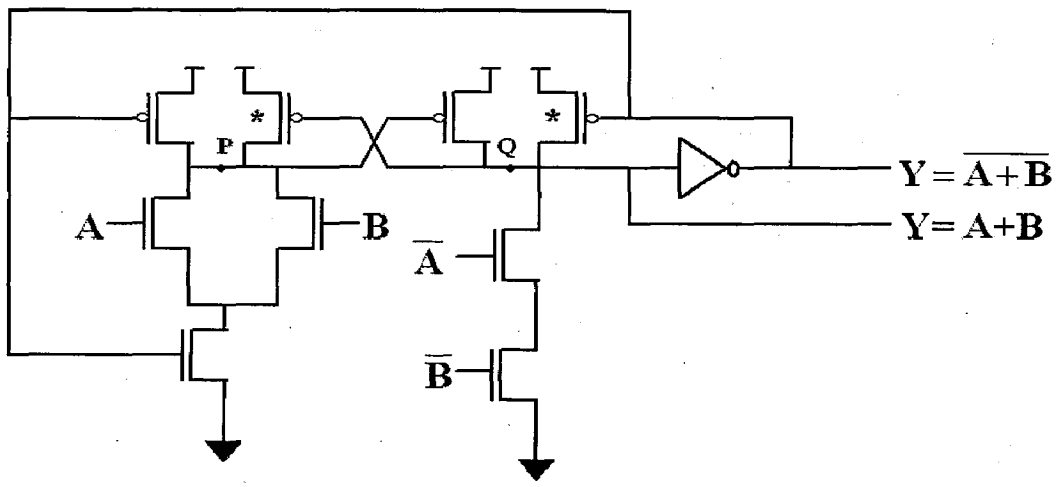


Fig 2.2(d): A NOR/OR gate based on Structure 2 of FSL Logic.

In Fig.2.2 (a), when Y is on and $\overline{Y}$ is off, the NAND node is at state "0" and the AND node is at state '1', the feedback-driven NMOS and PMOS are on and off, respectively, and node P is high. When Y turns off and $\overline{Y}$ turns on, node P discharges since it is held high by weak transistor. While the strong PMOS pulls, the NAND node is high which inturn discharges the AND node causing node P to charge again turning off the strong PMOS driver. Now the state of the gate is held at node $\overline{Y}$ by a weak feedback keeper and $\overline{Y}$ is ready to evaluate and so on. The simulation waveforms of 2-input NAND/AND gate is shown in Fig. 2.3

From the simulation waveforms, node P is high all the time except when $\overline{Y}$ turns on where a fast and short high low high pulse appears. This gives FSL gate an extra advantage by making a strong speed critical PMOS device tolerant against aging and Negative Bias Temperature Instability (NBTI) effects. Similarly for NOR/OR operation in FSL, simulated waveform is shown in Fig. 2.4.

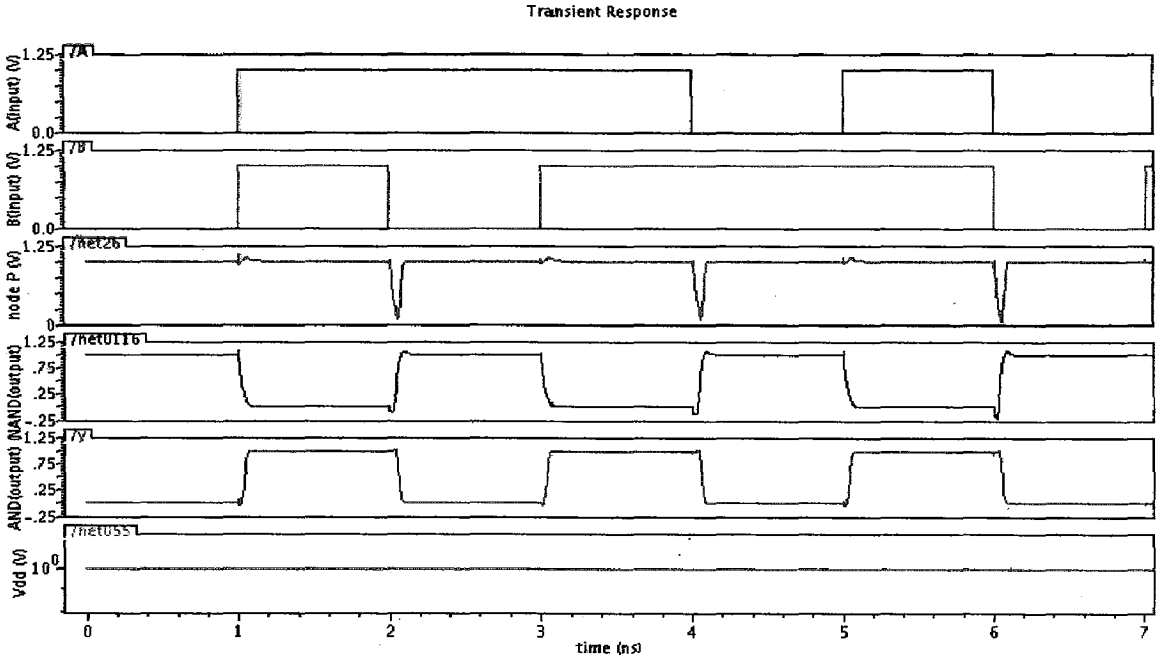


Fig 2.3: FSL NAND/AND operation

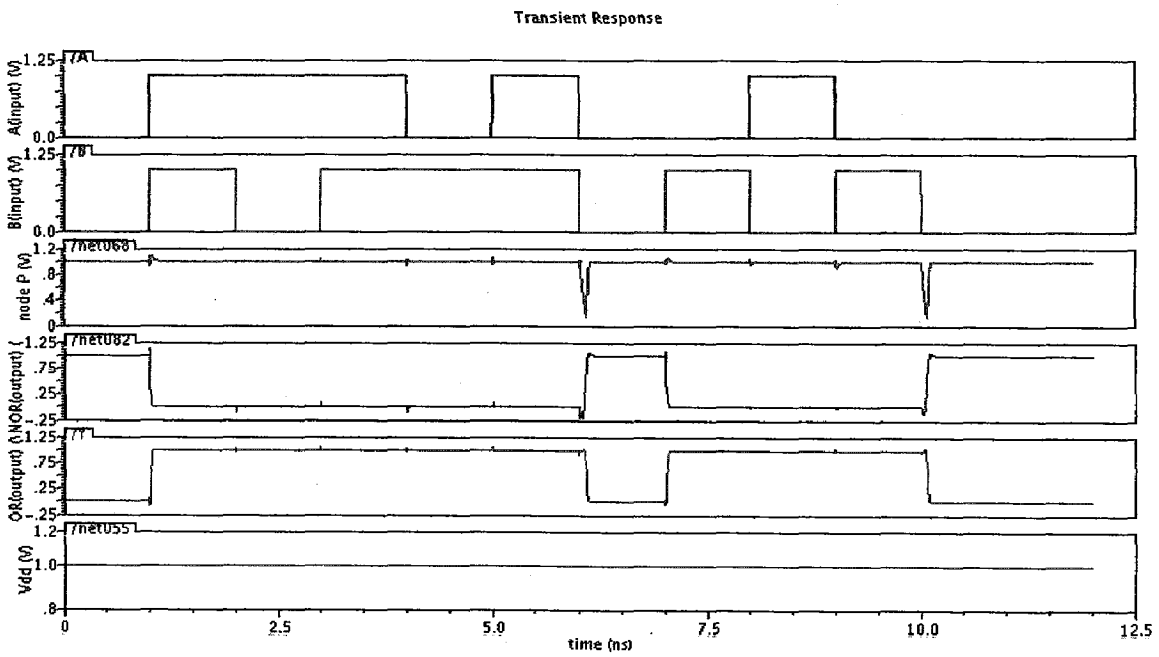


Fig 2.4: FSL NOR/OR operation

2.3 Simulation Results

The following are the comparison results of various logic gates using static CMOS and FSL logics.

The power consumption and delay of various logic gates using FSL and static CMOS are tabulated in Table 2.1.

Table 2.1: Summary of simulation results for various logical gates

Logical Gates	Power Consumption (in nW)		Delay (in ps)	
	CMOS	FSL	CMOS	FSL
AND2	839.3	914.8	21.595	17.805
AND3	860	927.1	28.45	22.988
AND4	871.2	932.2	33.205	29.552
OR2	947.9	1036.1	20.275	17.213
OR3	981.8	1061.3	25.725	22.021
OR4	1040.8	1116.8	31.185	27.786
XOR2	892.5	981.75	27.215	23.786

The power consumption comparison of different logic gates using FSL and static CMOS logic is shown in Fig.2.4.

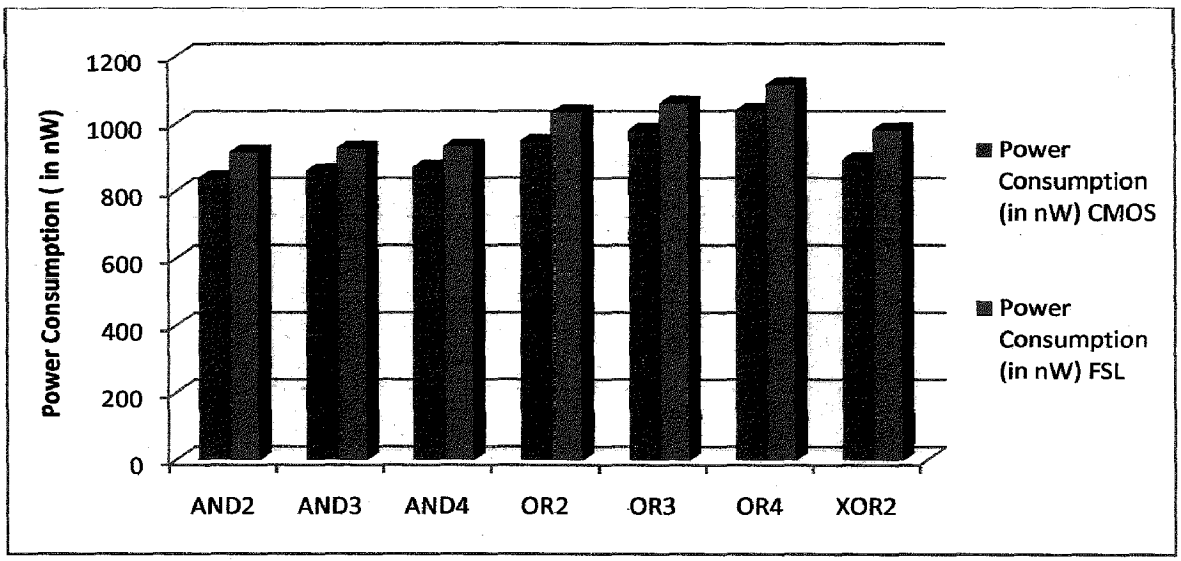


Fig 2.4: Comparisons of power consumption of different logic gates in static CMOS and FSL logics.

Since FSL is a combination of static and dynamic logic, the power consumption of logic gates using FSL is very much close to the power consumption of static CMOS gates.

The delay comparison of various logic gates using FSL and static CMOS is shown in Fig.2.5.

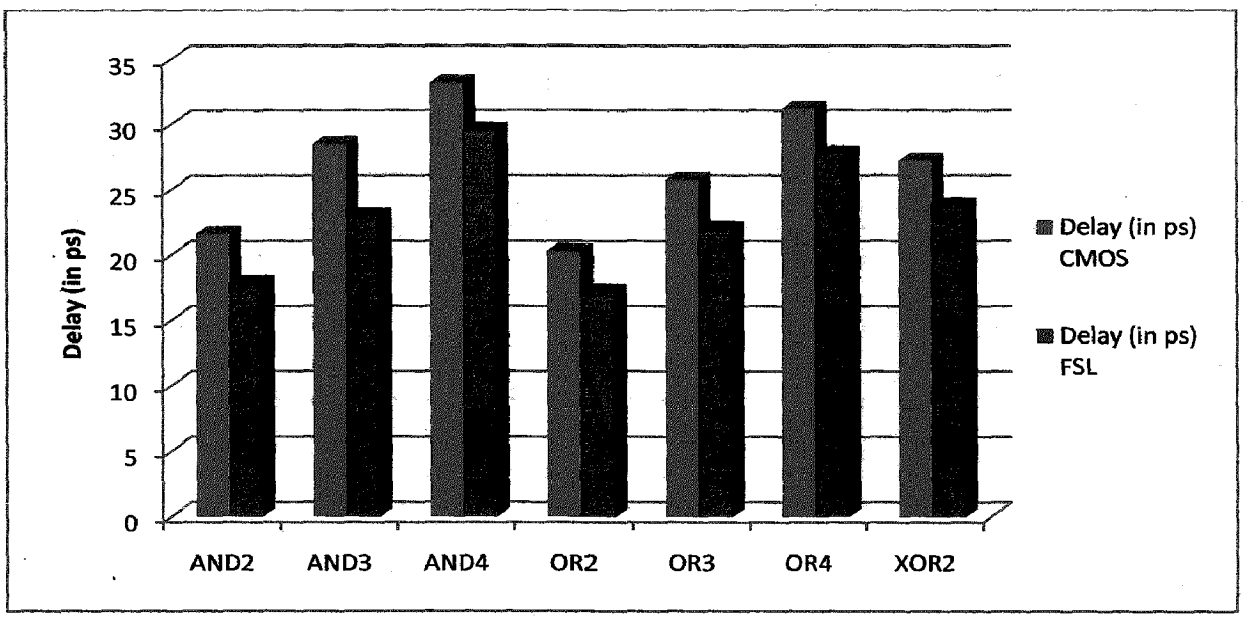


Fig 2.5: Comparisons of delays of different logic gates in static CMOS and FSL logics.

From the simulation results logic gates using FSL having lesser delay compared to static CMOS logic because FSL offers reduced capacitance and fast switching.

Chapter 3

Design of an Adder

3.1 Introduction

Addition is one of the fundamental arithmetic operations. It is extensively used in many VLSI systems such as application-specific DSP architectures and microprocessors. In addition to its main task, that is the adding of two binary numbers, it is the nucleus of many other useful operations such as subtraction, multiplication, division, address calculation, etc [8]. In most of these systems, the adder is the part of the critical path that determines the overall performance of the system. That is why enhancing the performance of the adder is a significant goal. It often referred as the speed-limiting element as well [5].

Addition is most often is one of the critical paths of modern day microprocessor. Adder is also the "hot spot" among all other execution units in a processor core. Thus, a fast and energy-efficient adder is essential to a high-performance microprocessor. Implementing fast addition has been an important subject since the 1950s. Ripple carry adder is the first and the most fundamental adder that is capable of performing binary number additions. Since its delay is proportional to the length of its input operands, it is not very useful. Weinberger and Smith [9] have made the first major break-through in speeding-up additions by proposing the well-known scheme, Carry Look-ahead adder [10]. Rather than rippling carries throughout the adder, it uses parallelism to propagate carries much quickly.

There are several levels of hierarchy which can improve addition operations. First, it can be improved at architecture level. By using an algorithm that incorporates a faster carry propagation method such as prefix tree adder schemes, addition time can be reduced. By adding clock gating and sleep mode, power can be saved when the adder is not in use. Multiple power supplies can also be used to supply lower voltage to less critical paths in order to reduce power. For mobile applications, dynamic voltage scaling is a widely used technique to reduce power consumption. It automatically optimizes the supply voltage and the operation frequency for a given workload.

To reduce the adder delay and power, one can also improve it at circuit level. For example, selecting dynamic instead of static CMOS logic would improve the speed of operation. Transistor sizing can be used to improve speed and power. A transistor sizing tool parameterizes the sizes of transistors in a circuit based on circuit characteristics such as driver size, load, supply voltage, operating temperature etc. After completing a transistor level design, the customized cell placement can be used, which may produce a better layout than automatic place and route tool. With today's advanced technology, a careful planning of wires and cells before placement is often necessary in achieving a better timing and power.

Finally, speed can be improved at process level. A more advanced process technology can be used. This could be scaling the transistor's feature size, using copper interconnects rather than aluminum interconnects, using SOI (Silicon on Insulator) CMOS technology rather than Bulk CMOS technology, using thinner and higher K dielectric insulator as gate oxide, etc.

3.1. Adders

Adders are basic components of microprocessors and any arithmetic circuit and are frequently on the critical path. Fast adders speed up the addition calculation through a rearrangement of the adder equations or through some intelligent observations about the addition process. While adder speed is essential, adder area is also important, especially for arithmetic circuits that may require many adders. The next few sections would discuss about different adder designs and their comparisons.

3.1.1 Basics of Adder

A multi-bit addition operation can be decomposed into half adder and full adder structures, with fast adders containing some additional circuitry. Half adders and full adders compute the well-known logic functions given as follows:

Half adder: $\text{Sum} = A \oplus B$, $C_{out} = AB$

Full adder: $\text{Sum} = A \oplus B \oplus C$, $C_{out} = AB+BC+CA$

3.1.2 Ripple Carry Adder

This is the simplest adder circuit. An N-bit ripple carry adder [11] consists of N full adders with the carry signal propagating from one full-adder stage to the next from LSB to MSB. A 4-bit ripple carry adder structure is shown in Fig. 3.1. It consists of four cascaded full adder which takes input A, B and Cin and generates sum and carry out (Cout).

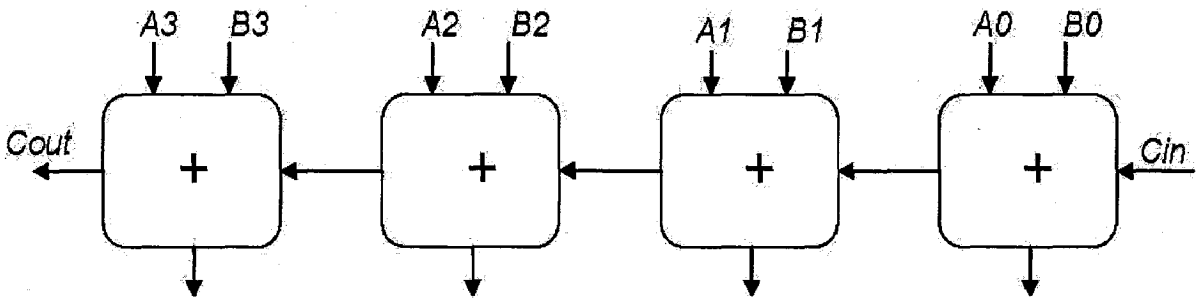


Fig. 3.1: 4-bit Ripple Carry Adder

The ripple carry adder is a good baseline design for comparison with other adders. It has many advantages which include low power, low area and a simple layout. The drawback of the ripple carry adder, though, is its slow speed. The delay of the adder is linearly dependent on the bit-width (N) of the adder. The critical path of the ripple carry adder consists of the carry chain from the first full adder to the last. Therefore, during circuit-level design, the carry signal is frequently assigned to the transistor closest to the gate output for the carry computation.

Circuits are optimized to produce fast carries because it constitutes a large fraction of the critical path. The delay of a ripple carry adder is given by the following equation [5]:

$$T(RCA) = (N-1)\times T(Carry) + T(Sum)$$

For designing a single bit of full adder for a RCA, we can use different types of full adder as reported by Alioto and Palumbo [12]. For our design of RCA, we used Mirror Full adder [13] as shown in Fig. 3.2. Mirror Full adder CMOS Full adder is a simple implementation of following equations.

$$Sum = A \oplus B \oplus C = ABC + \overline{C_{out}}\,(A + B + C)$$

and

$$\text{Carry } (C_{out}) = AB + BC + AC = AB + (B + A)C$$

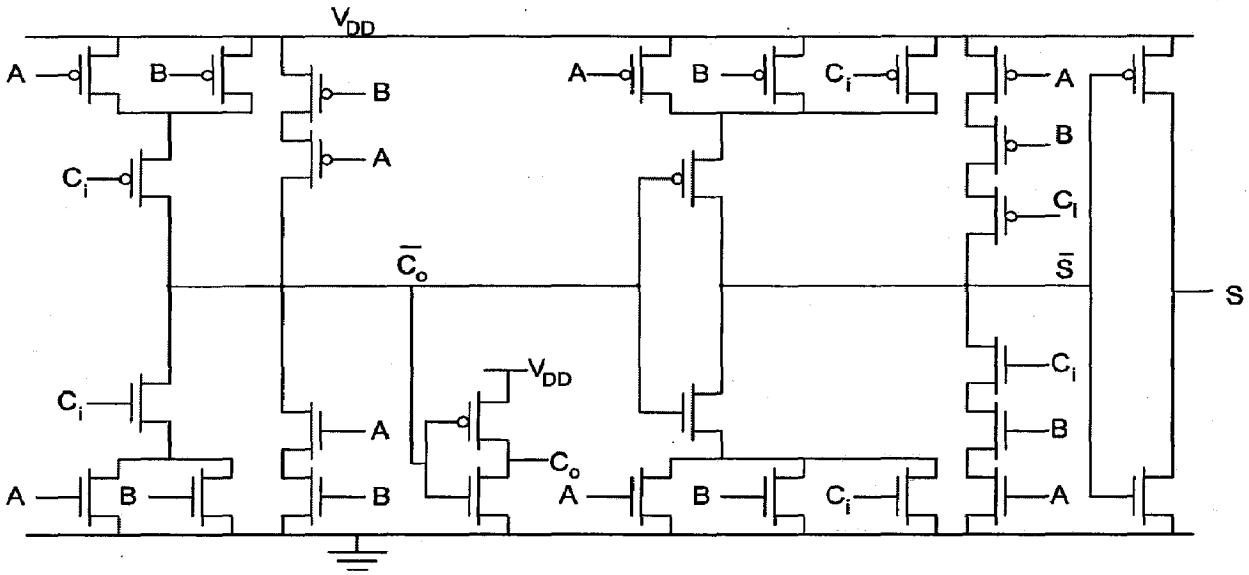


Fig. 3.2: A Single-Bit Mirror Full Adder [12].

3.1.3 Carry Look-ahead Adder

The Carry Look-ahead Adder (CLA) is theoretically one of the fastest methods for addition. Weinberger and Smith invented the CLA in 1958 [9]. The CLA uses intermediate information to determine in advance if there will be a carry out of a given bit position. Table 3.1 shows the truth table for a full adder, including this extra carry information. For the delete condition, there will be no carry out of the bit position. For the propagate condition, there will only be a carry out if there is a carry in. For the generate/propagate condition, there will always be a carry out at that position.

Table 3.1: Generate and propagate information for a CLA

A	B	C	Sum	C_{out}	Condition
0	0	0	0	0	Delete
0	0	1	1	0	Delete
0	1	0	1	0	Propagate
0	1	1	0	1	Propagate
1	0	0	1	0	Propagate
1	0	1	0	1	Propagate
1	1	0	0	1	Generate/Propagate
1	1	1	1	1	Generate/Propagate

Fig. 3.3 shows the block diagram for a 4-bit section of a CLA. In CLA, Reduced Adders (RA) are used as they are no longer required to compute the output carry. The CLA block at the top of the diagram is a set of circuitry that creates, generates and propagates signals

for a group of full adders, as well as the carry-out from that group. The following equations compute each position's generate and propagate signals [14]:

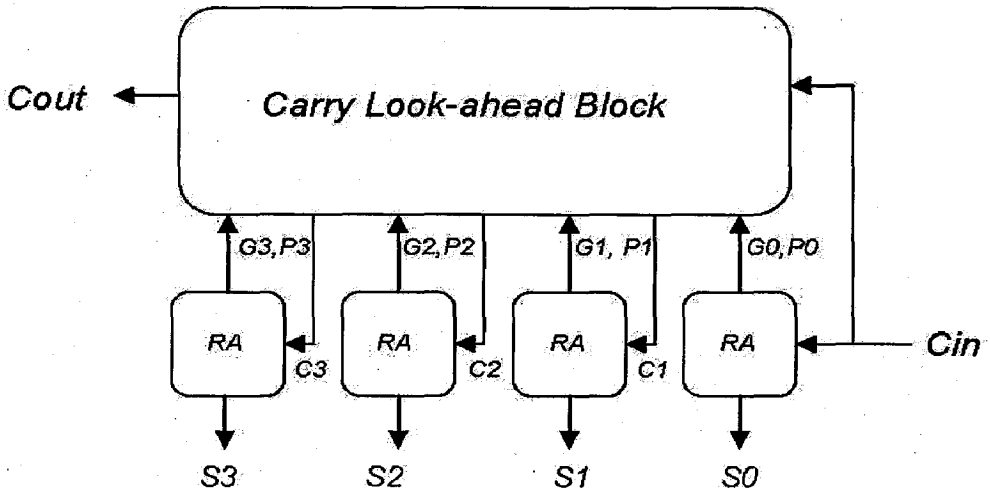


Fig. 3.3: 4-bit Carry Look-ahead Adder

Generate: $Gi = Ai \cdot Bi$

Propagate: $Pi = Ai + Bi$

In some books, they define the propagate signal as the Exclusive-OR of the A and B signals, but this does not change the result of addition. The above definition generally chosen because the implementation of the OR operation is more efficient than that of Exclusive-OR in most technologies. The equations for the carries in a CLA are given by:

$$C1 = G0 + P0 \cdot Cin$$

$$C2 = G1 + G0 \cdot P1 + P0 \cdot P1 \cdot Cin$$

$$C3 = G2 + G1 \cdot P2 + G0 \cdot P1 \cdot P2 + P0 \cdot P1 \cdot P2 \cdot Cin$$

For example, computing C3 requires the use of a 4 input AND gate and a 4 input OR gate. Hence, usually the size of the look-ahead logic is limited to 3 carries. AND Gates with 5/6 inputs would be needed for the next 2 carry signals, which makes their implementation in CMOS very slow due to the stacked transistors in the pull-up or pull-down paths.

As the carry calculation is performed by the carry look-ahead block, the one-bit adder equations for a CLA are the reduced full-adder equations because carry calculation is no longer needed. The reduced full adder performs the operation given by equation below:

$$Sum = A \oplus B \oplus C = A\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C + ABC$$

3.2 Prefix Adders

Among the various binary adder architectures, prefix adders [15] are particularly attractive because they have the minimum possible logic depth. These adders are also termed logarithmic adders because their critical path is O ($\log_2$N). Examples of prefix adders are Kogge-Stone [16], Brent-Kung [17] and Han-Carlson adders [18].

The addition of two binary numbers can be formulated as a *prefix problem*. In a prefix problem, outputs {y(n-1), y(n-2)... y(0)} are computed from n inputs {(x(n-1), x(n-2)... x(0))} using an arbitrary associative operator (*) as follows:

$$y(0) = x(0)$$
$$y(1) = x(1) * x(0)$$
$$y(n-1) = x(n-1) * x(n-2) * ...x(1) * x(0)$$

he problem can be formulated recursively as

$$y(0) = x(0)$$
$$y(i) = x(i) * y(i-1) \text{ where } i = 1, 2... ,n-1$$

In other words, in a prefix problem every output depends on all inputs of equal and lower magnitude and every input influences all outputs of equal or higher magnitude. Due to the associativity of the prefix-operator, the individual operations can be carried out in any order. This is a fundamental property which explains why there are various tree structures for addition.

Let us define the * operator to be the carry-merge operator which combines generate and propagate signals using the following equations:

$$(Gout, Pout) = (G2, P2) * (G1,P1) = (G2 + P2 \bullet G1, P2 \bullet P1)$$

We need to calculate the Generate and Propagate signals for each bit to perform the carry-merge operation from the equation above at each junction in the carry merge tree for the desired logarithmic adder, Brent-Kung, Kogge-Stone, or Han-Carlson. The underlying carry-merge operation is the same for all logarithmic adders but the connections in the tree are different. After $\log_2$N stages, where N is the bit-width of the adder, the carry signals will be fully generated. As with the CLA, a reduced full-adder is used to find the sum using the carry, generate and propagate signals.

3.2.1 Kogge-Stone Adder

Kogge and Stone proposed a general recurrence scheme for parallel computation in 1973 [16]. The Kogge-Stone adder is a parallel prefix form carry look-ahead adder. Time needed to generate the carry signals is equal to the number of level and it is equal to O ($\log_2$ N). It is widely considered the fastest adder design possible. And the cost is n^2 because of the large no. of vertical tracks required to embed wires in the upper stage. It takes more area to implement but it has a lower fan-out at each stage, which increases performance. A 32-bit Kogge-Stone adder is shown in Fig.3.4.

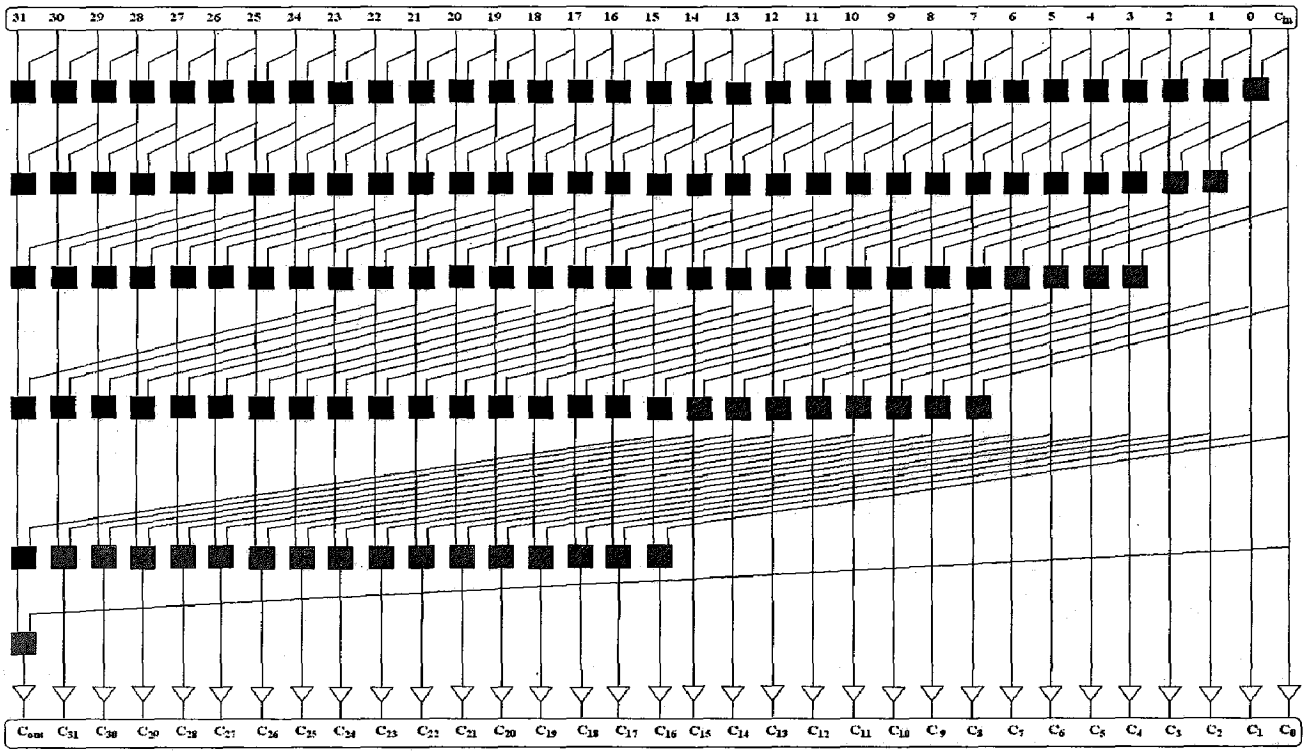


Fig. 3.4: Diagram of Kogge-Stone Adder

3.2.2 Brent-Kung Adder

The replicated Kogge Stone structure to generate intermediate carries shown in Fig. 3.5 is very attractive to high-performance applications. However, it comes at the cost of area and power. A simpler tree structure could be formed if only the carry at every power of two positions is computed as proposed by Brent and Kung [17]. It is based on divide and conquer approach. The inputs are first combined pairwise to obtain the sequence of length n/2 and the even-indexed prefix are then computed by odd-prefix. The Schematic diagram of 32-bit Brent-Kung adder is shown in Fig.3.5.

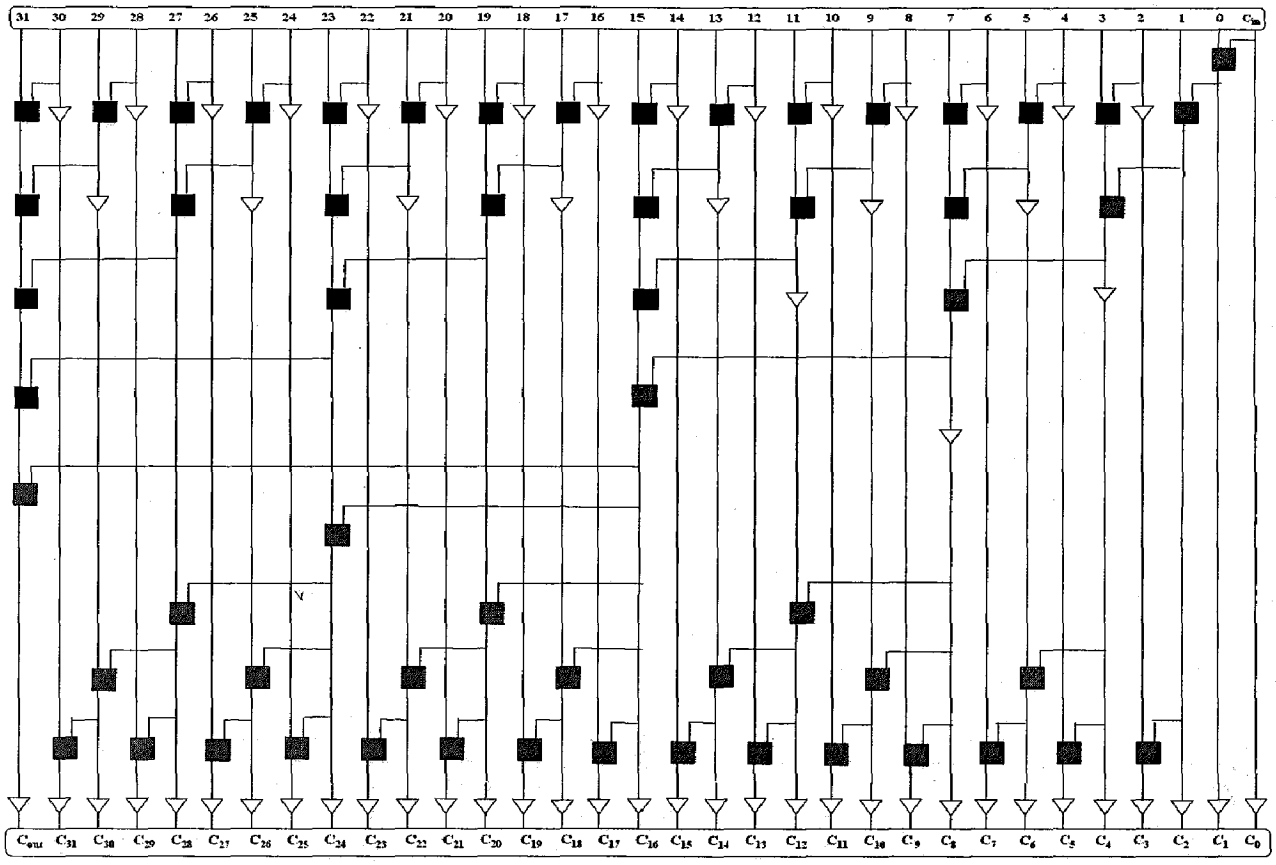


Fig. 3.5: Diagram of Brent-Kung Adder

The Brent-Kung adder has the lowest area and the slowest speed of all the logarithmic adders.

3.2.3 Han-Carlson Adder

Similar to Brent and Kung's scheme, Han and Carlson also proposed a scheme to reduce the complexity of prefix tree [18]. It combines both Brent-Kung and Kogge-Stone adders. It is different from Kogge-Stone scheme. This scheme performs carry-merge operations on even bits only. Generate and propagate signals of odd bits are transmitted down the prefix tree. They recombine with even bits carry signals at the end to produce the true carry bits. Thus, the reduced complexity is at the cost of adding an additional stage to its carry-merge path. The time to compute is only one stage more than Kogge-Stone adder. The schematic of a 32-bit Han-Carlson Adder is shown in Fig.3.6.

The Han-Carlson adder combines the Brent-Kung and Kogge-Stone carry merge trees to achieve a balance both with respect to speed and area.

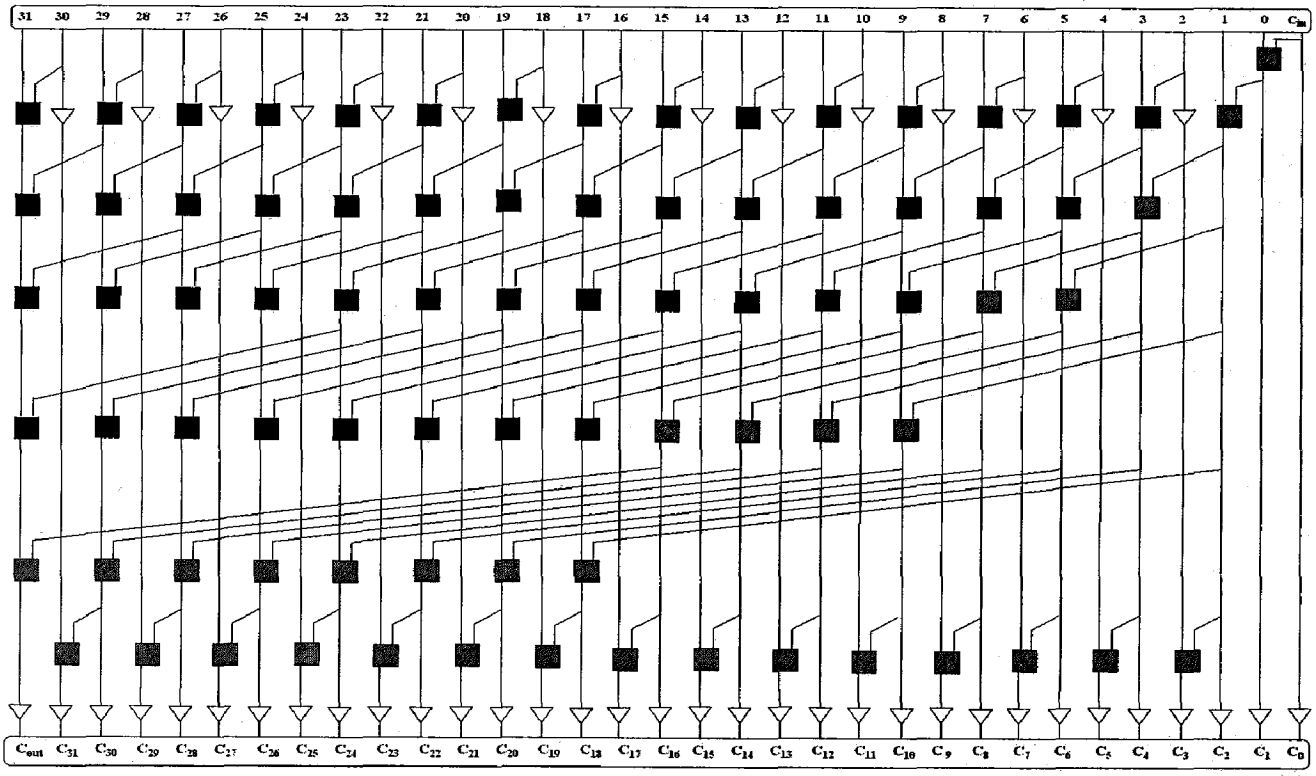


Fig. 3.6: Diagram of Han-Carlson Adder

In all of the above parallel prefix adders, black block, grey block, propagation and generate block are implemented as follows:

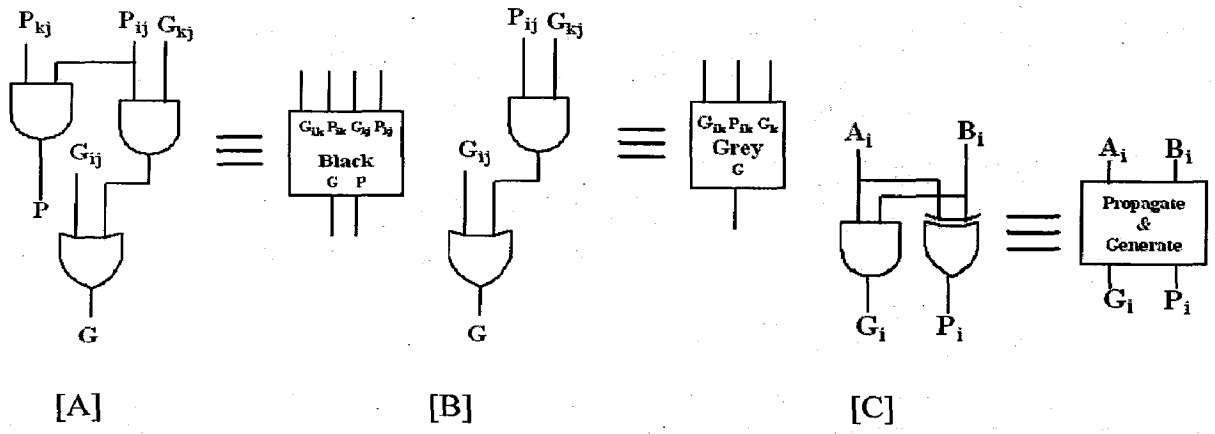


Fig. 3.7: Different blocks in prefix adders, (A) Black block, (B) Grey block, (C) Propagate and Generate block

where P_{ij} and G_{ij} are propagate and generate lines from present lines while P_{kj} and G_{kj} are propagate and generate are due to just previous bits.

3.3 Adder Comparisons

This section describes about comparison results of different adders in terms of power and delay.

The power consumption and delay of Ripple Carry adder, Kogge-Stone adder and Han-Carlson adder using FSL and static CMOS are evaluated and tabulated in Table 3.2.

Table 3.2: Summary of simulation results for various Adders

Adder Architecture	Power Consumption (in μW)		Delay (in ps)	
	CMOS	FSL	CMOS	FSL
Ripple Carry Adder(RCA)	220.34	240.55	270.7	239.6
Kogge-Stone Adder	328.19	360.25	210.7	178.4
Han-Carlson Adder	273.01	292.45	264.5	223.4

The power consumption comparison of various adders using FSL and static CMOS is shown in Fig.3.7.

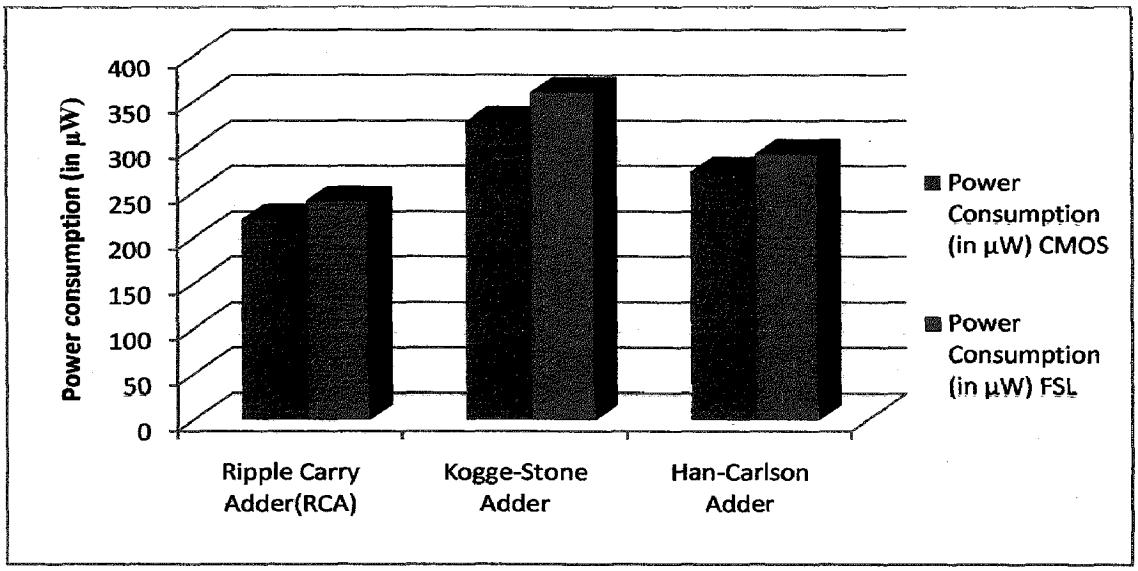


Fig 3.7: Comparisons of power consumption of different adders in static CMOS and FSL logics.

From simulation results ripple carry adder consumes less power compared to Kogge-Stone and Han-Carlson adders in both the logics.

Delay comparison of different adders using static CMOS and FSL is shown in Fig.3.8.

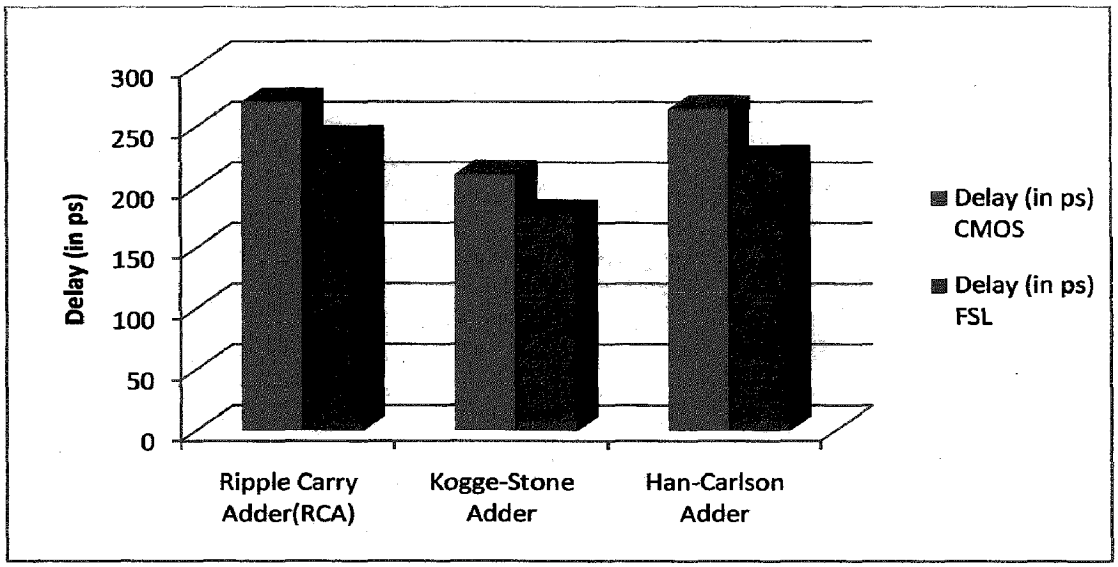


Fig 3.8: Comparisons of delays of different adders in static CMOS and FSL logics.

From the simulation results, it shown that the delay of Kogge-Stone adder is lower than ripple carry adder and Han-Carlson adder in both the logic. But quantitatively delay of Kogge-Stone adder using FSL is less than the delay of Kogge-Stone adder using static CMOS.

Chapter 4
Shifter Design

Data shifting is required in many key computer operations from address decoding to computer arithmetic. Binary shifters, similar to adders and multipliers, are essential in high performance microprocessors, especially in those applications that support floating-point operations. A cyclic shifter is a crucial component for communication applications such as encryption and error control coding where we require rotation operations. Yet the importance of shifter logic is underestimated in circuit design due to its simplistic nature. Literature on shifter design is relatively scarce compared to that of adders and multipliers and textbooks typically cover shifter in just one or two pages. The main reason is that the complexity of shifters comes from the internal wire connections which do not fit into the traditional logic-centric design methodology [19].

In terms of design style, there are three types of shifters for circuit designers, which are:

1. Array Shifter
2. Barrel Shifter
3. Logarithmic Shifter

4.1 Array Shifter

An array shifter decodes the shift value into individual shift bit lines that mesh across all input data values. At each crossing point, a NMOS transistor will either allow or not allow the input data value to pass to the output line controlled by a shift bit line. The advantage of this design is that there is always only one NMOS transistor between the input data lines and the output data lines, hence, it is fast. The basic structure of an array shifter is shown in Fig.4.1.

The disadvantages of this design are: firstly, the requirement of a decoder, and secondly, the fact that each input data line sees a load for every shift bit line [21]. A simple one bit bidirectional array shifter is shown in Fig.4.2. According to the control signals, the input word is either shifted left or right or else it remains unchanged. Multi-bit shifters can be

built by cascading a number of the units. Array shifter becomes complex and too slow for higher shift values.

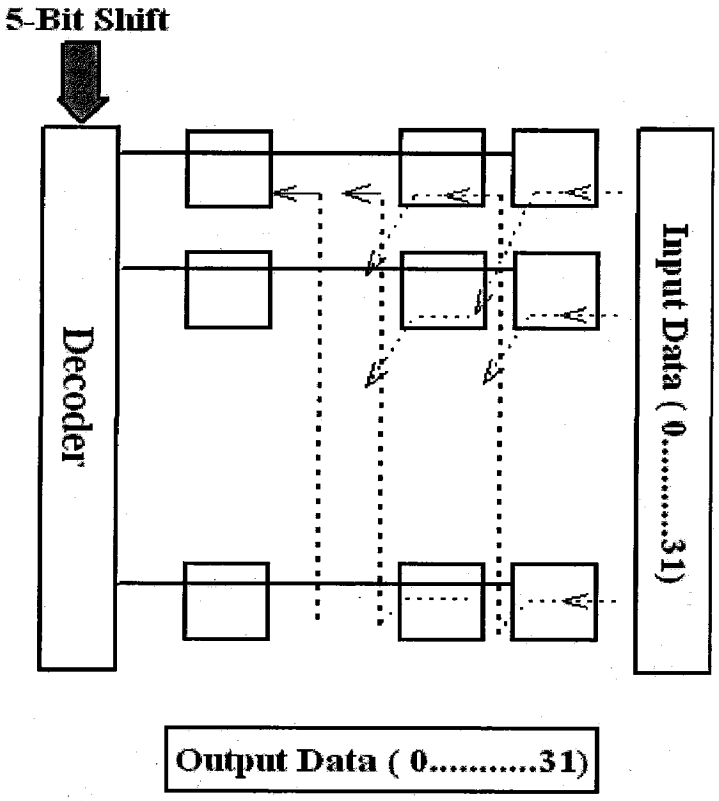


Fig.4.1: Structure of an Array shifter [20]

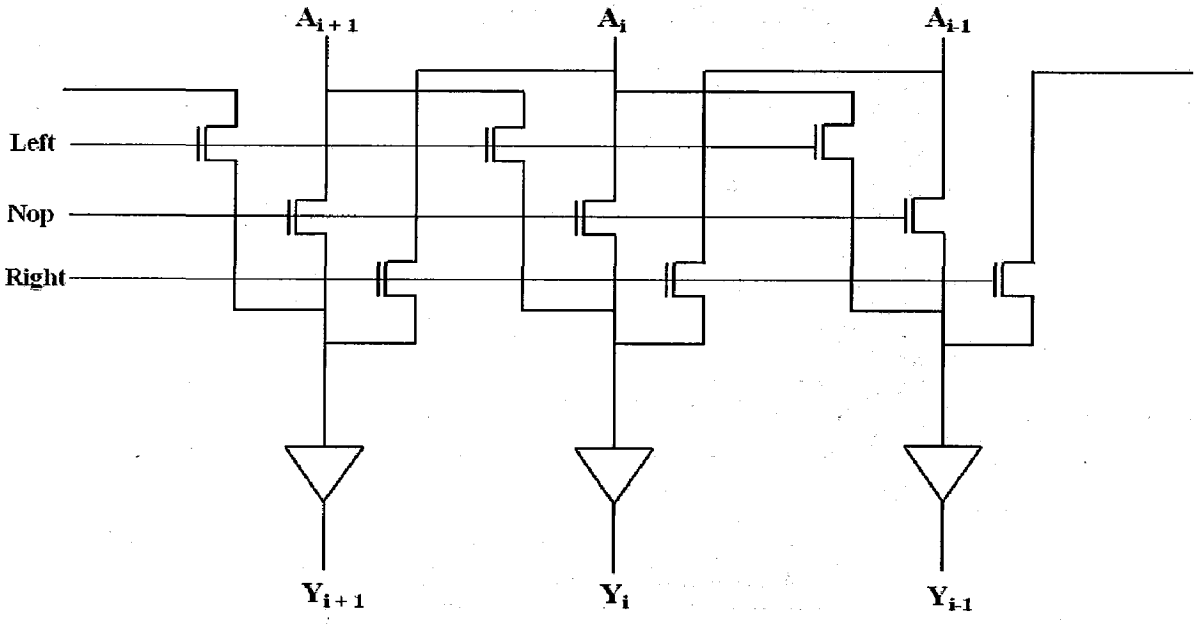


Fig. 4.2: A simple one Bit Bidirectional Array shifter [5].

4.2 Barrel Shifter

A Barrel shifter consists of an array of transistors, in which the number of rows and columns are equal to the word length of the data and the maximum shift width respectively. The control wires are routed diagonally through the array. A major advantage of barrel shifter is that the signal has to pass through at the most one transmission gate. The propagation delay is constant and independent of the shift value. However, this is not always true, because the capacitance at the input of the buffers rises linearly with the maximum shift width.

A 4-bit barrel shifter shown in Fig.4.3, needs four control signals to shift over three bits. The signals, Sh_3, Sh_2, Sh_1 and Sh_0 take on the value 1000. Only one of the signals is high. For instance, the encoded control word needs only two control signals and is represented as 11 for a shift over three bits. To translate this shift bit, an extra module known as the decoder is required.

The barrel shifter is primarily used in floating-point arithmetic hardware. For a floating-point add or subtract operation, the fractions of the numbers must be aligned, which require shifting the smaller number (in magnitude) to the right and increasing its exponent until it matches the exponent of the larger number. This is done by subtracting the exponents and using the barrel shifter to shift the smaller number to the right by the difference in one cycle. If a simple shifter was used, shifting by n bit positions would require n clock cycles.

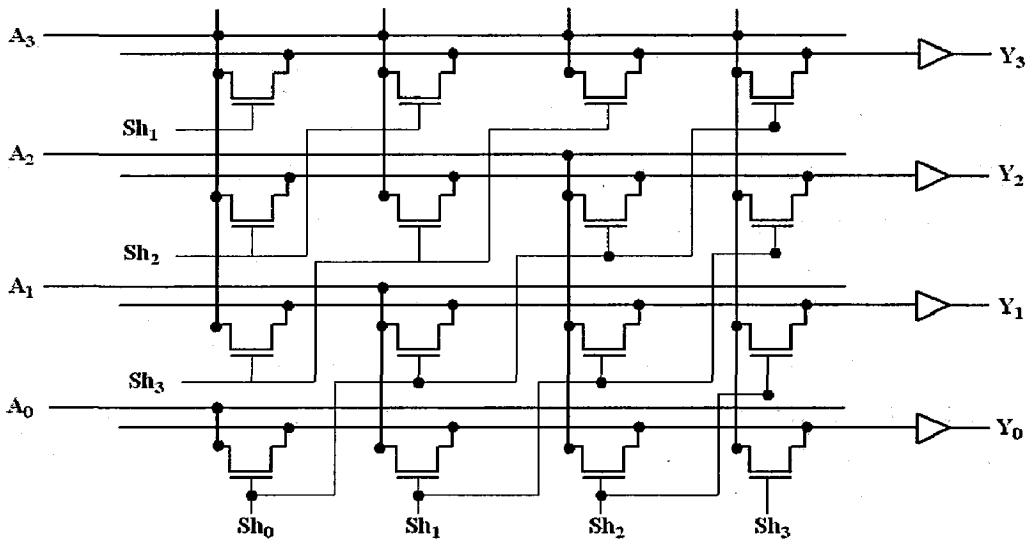


Fig.4.3: A 4-bit Barrel Shifter [5]

4.3 Logarithmic Shifter

A Logarithmic shifter is based on stage approach. The total shift value is decomposed into shifts over powers of two. No. of total stages for a maximum shift width M is $\log_2 M$ stages, where the ith stage either shifts over 2^i or passes the data unchanged.

A Logarithmic shifter for a maximum right shift width of 7 bits is shown in Fig.4.4. To shift over 5 bits, the first stage is set to shift mode, second stage to pass mode and the last stage again to shift mode. The control word of this shifter is already encoded and no separate decoder is required.

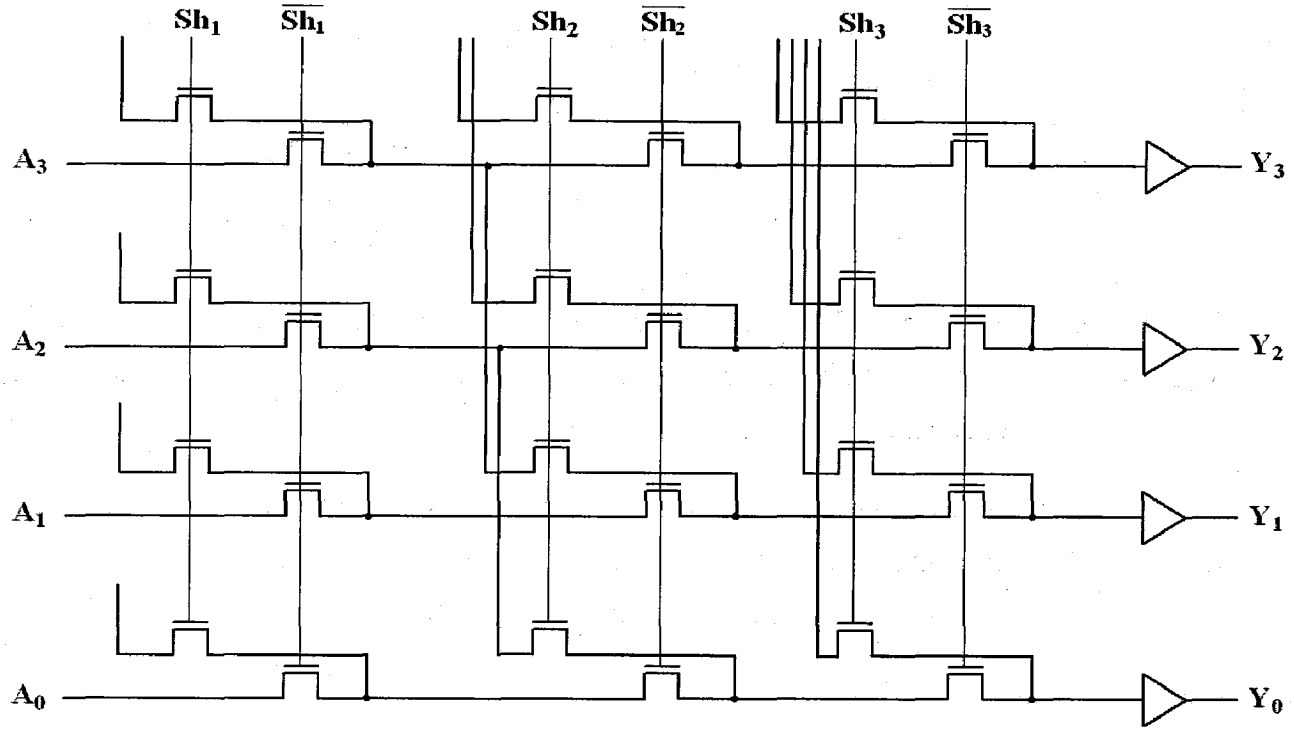


Fig. 4.4: A 4-Bit Right Shift Logarithmic Shifter [5].

In a logarithmic shifter, the shifter is divided into $\log_2$ M stages, where M is the input data length. Thus, the speed of the logarithmic shifter depends on the shift width in a logarithmic way. Each bit of the encoded shift value is sent to a different stage of the shifter. Each stage handles a single power of- two shifts. The input data will be shifted or not shifted by each of the stages in sequence depending on the shift value. Five stages would be required when considering 32 bit data as shown in Fig.5.5. The advantage of a logarithmic shifter is that it occupies small area and does not require a decoder, but the disadvantage is that there are five levels of gates separating the input data from the output

data. Furthermore, the series connection of pass transistor slows the shifter down for larger shift values.

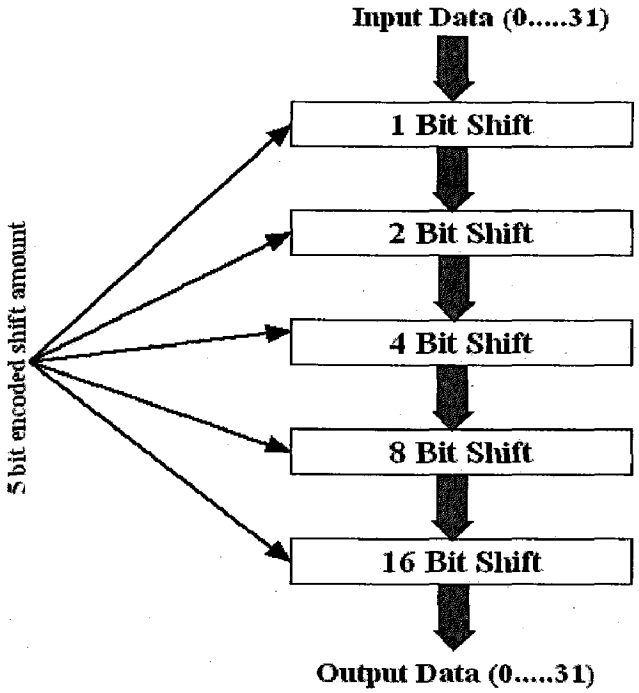


Fig. 4.5: Structure of a Logarithmic Shifter [20]

From this analysis, we can observe that use of array shifter is limited to lower shift bits. As the number of shift bits increase, the number of stages of the shifter also increases finally leading to higher delay and complexity of design. In the case of Barrel shifter, we can conclude that it is appropriate only for smaller shift values. For larger shift values, the logarithmic shifter becomes more effective both in terms of area and speed [20-21].

4.4 Design of Cyclic Shifter

Circular shift is a permutation of the entries in a tuple where the last element becomes the first element and all the other elements are shifted, or where the first element becomes the last element and all the other are shifted. Equivalently, a circular shift is a permutation with only one cycle. The cyclic shift of a binary operand is a basic operation which is required at many different places in circuit design. For communication applications such as encryption and error control coding, the cyclic shifter is a critical component because rotation operations are enormously needed. Circular shifts are also used often in cryptography as part of the permutation of bit sequences.

A design of cyclic shift enabled array shifter and logarithmic shifter is as follows:

4.4.1 Cyclic Array Shifter

For designing cyclic enable array shifter, we need a single bit array shifter capable of shifting 1-bit at a time as shown in Fig.4.6. By using this single bit shifter we designed array shifter which is capable of shifting upto 31 bits at a time as shown in Fig.4.7. We need two 5x32 decoder for decoding shift values Sh_5, Sh_4, Sh_3, Sh_2 and Sh_1. For enabling the cyclic shifter we have used two gates in which one is AND gate takes cyclic enable input (CirLeft or CirRight) and output of it is given to OR gate which generates the shifting value for LShift or RShift. The schematic of final cyclic shifter is shown in Fig.4.8.

4.4.2 Cyclic Logarithmic Shifter

For designing cyclic enable logarithmic shifter, we need a 32 bit logarithmic shifter without circular shifting as shown in Fig. 4.9. By using this shifter we designed a cyclic logarithmic shifter by adding different multiplexers as shown in Fig. 4.10. These multiplexers provide shifting values to various shift positions such as RA_0 to RA_{30} for right direction or LA_1 to LA_{31} for Left direction. The schematic of final cyclic shifter is shown in Fig.4.8.

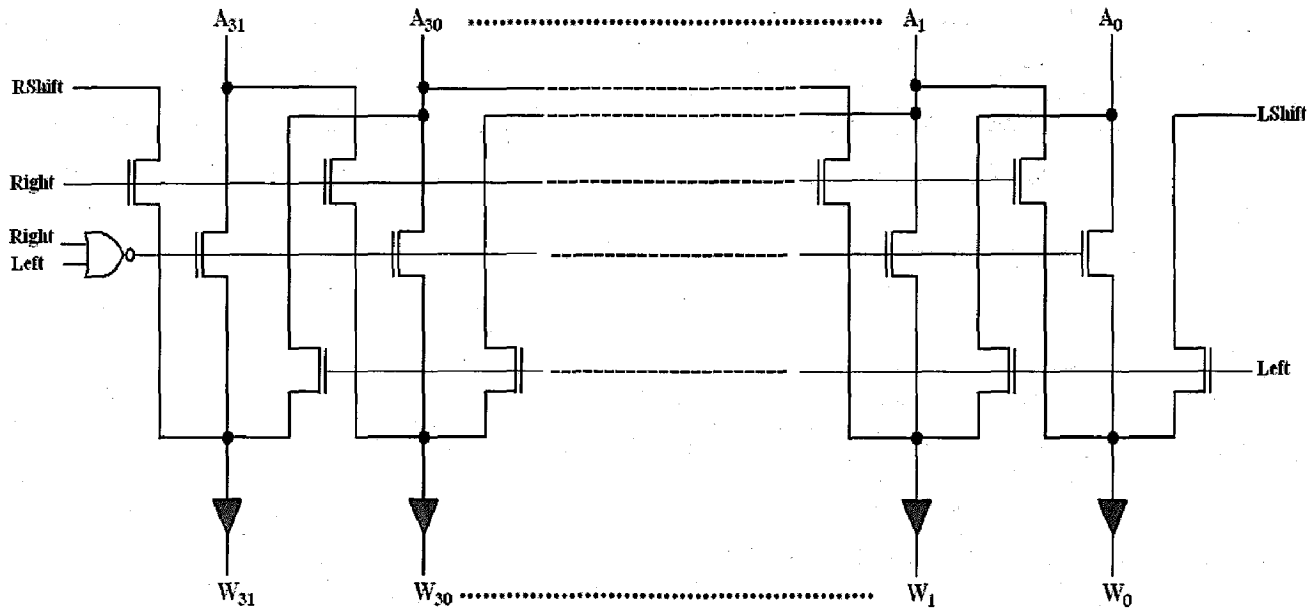


Fig. 4.6: Single bit array shifter.

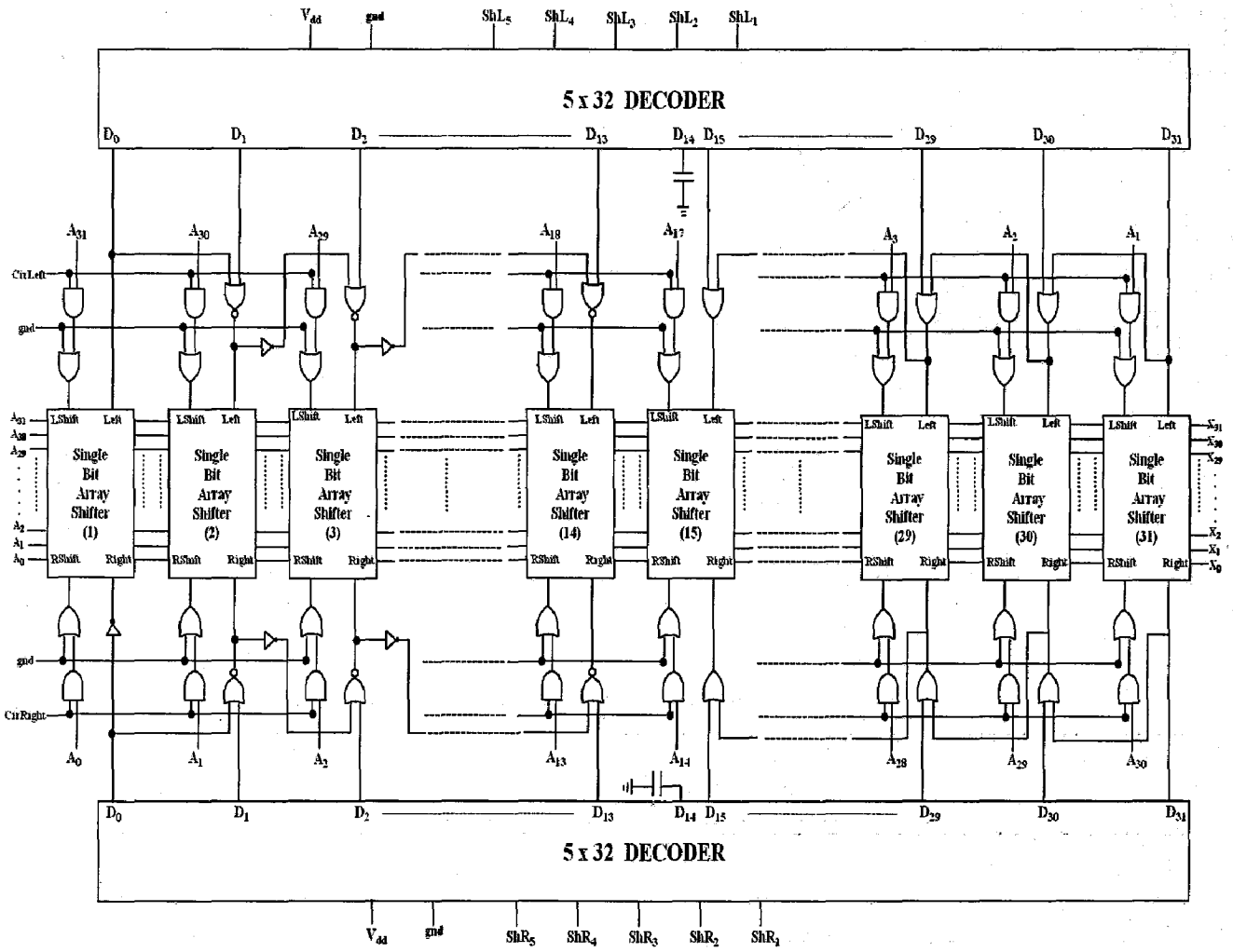


Fig. 4.7 Cyclic Array Shifter

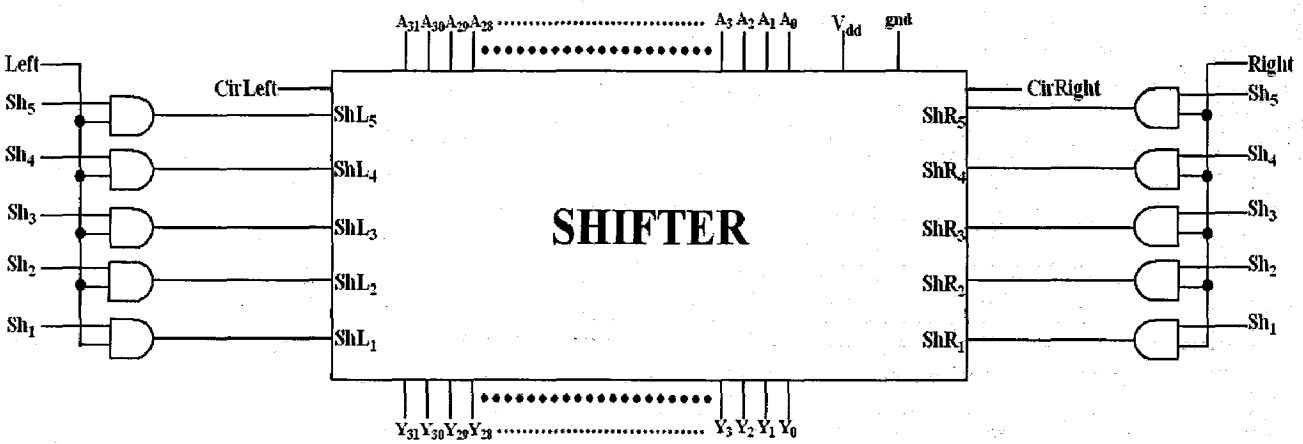


Fig. 4.8: Final schematic of cyclic enable shifter

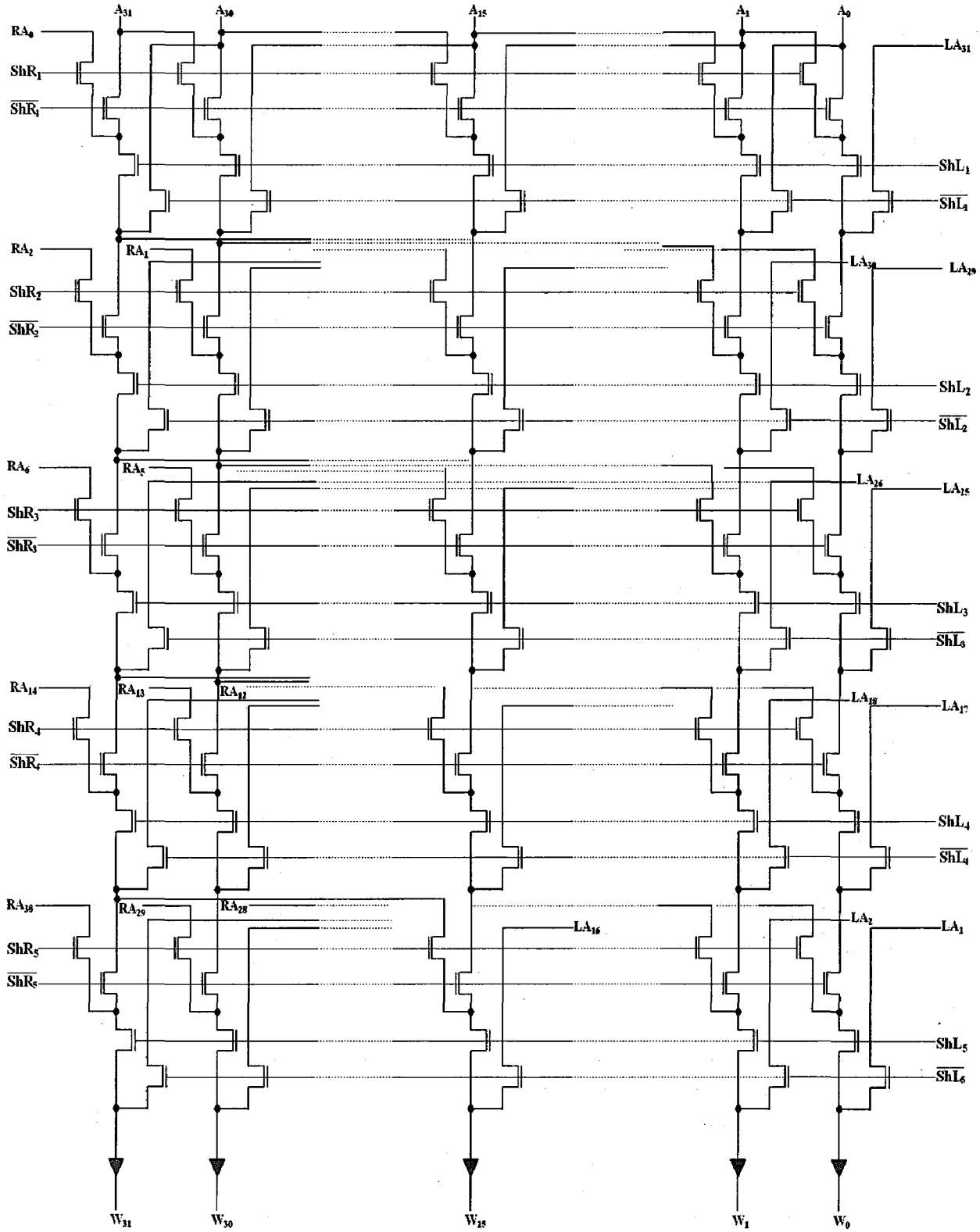


Fig. 4.8: 32 Bit Logarithmic Shifter without circular shifting.

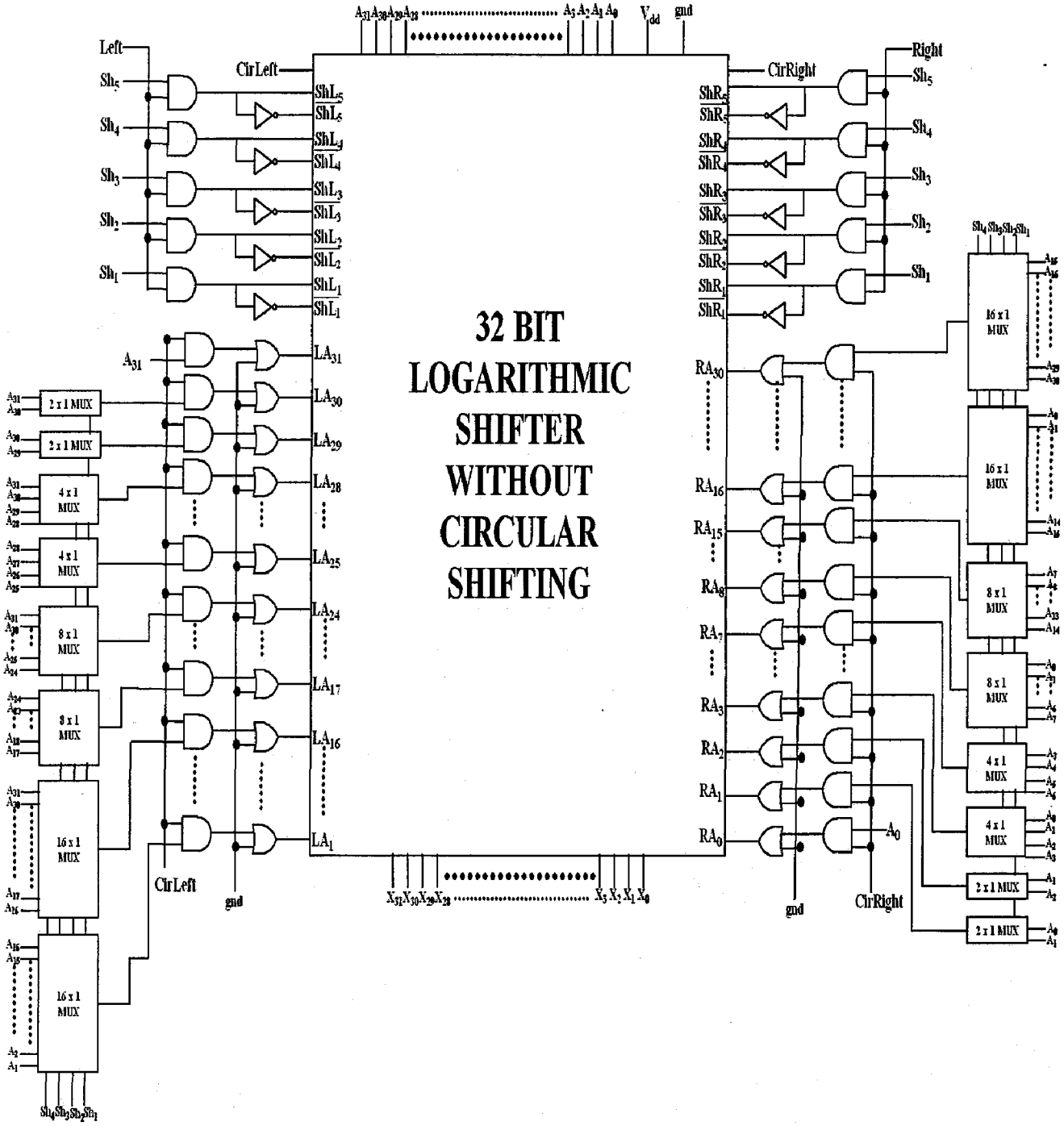


Fig. 4.8: 32 Bit Cyclic enable Logarithmic Shifter.

4.5 Simulation Results

The simulation results of various shifters using FSL and static CMOS are follows.

The power consumption and delay of different shifters using static CMOS logic and FSL are calculated and tabulated in Table 4.1.

Table 4.1: Summary of simulation results for Shifters.

Shifting Operation	Shifter Architecture	Power Consumption (in μW)		Delay (in ps)	
		CMOS	FSL	CMOS	FSL
Logical Left Shift	Array	201.6	214.7	260.4	254.3
	Logarithmic	204.5	216.9	220.1	213.4
Logical Right Shift	Array	238.1	255	251.7	247.3
	Logarithmic	250.3	267.3	218.7	213.1
Circular Left Shift	Array	397.3	431.9	421.3	417.1
	Logarithmic	410.7	447.3	415.2	411.3
Circular Right Shift	Array	423.6	461.3	445.2	439.1
	Logarithmic	430.4	469.6	427.3	421.5

The power consumption of comparison of various shifters among FSL and static CMOS logics is shown in Fig.4.6.

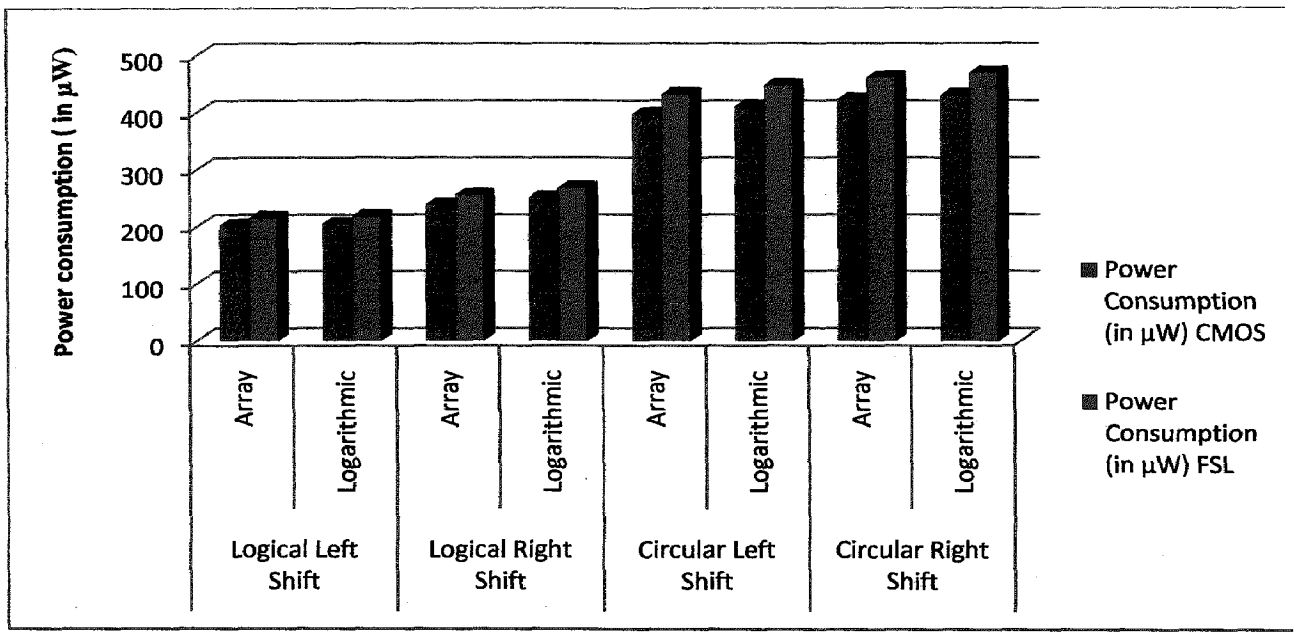


Fig 4.6: Comparisons of power consumption of different shifters in static CMOS and FSL logics.

Simulation results shows that power consumption of array shifter is less compared with power consumption of other shifters in both the logics.

Comparison results of delay using FSL and static CMOS among various shifters is shown in Fig.4.7.

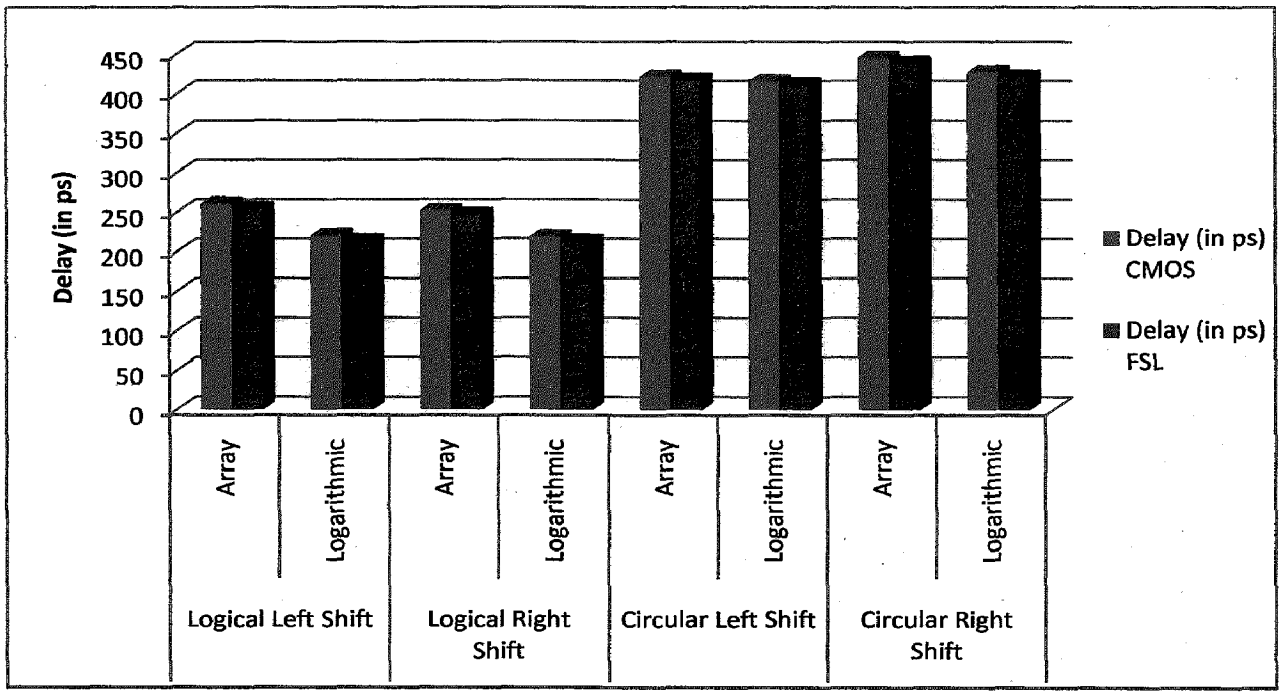


Fig 4.7: Comparisons of delays of different shifter in static CMOS and FSL logics.

Simulation results shows that delay of logarithmic shifter is lower than other shifters in FSL as well as in static CMOS logic. But quantitatively delay of logarithmic shifter is less using FSL compared with delay of logarithmic shifter using static CMOS.

Chapter 5
ALU Design

5.1 Introduction

The Arithmetic Logic Unit is a digital circuit that performs an arithmetic operation (addition, subtraction, etc.) and logic operations (Exclusive-OR, AND, etc.) between two numbers. Demand for performance at low power consumption in today's general purpose processor has put severe limitations on ALU design [23].

ALU are also one of the most power consumed blocks in the processor and are often the possible location of hot-spots. The presence of multiple ALUs in pipelined processors further deteriorates the power and thermal issues [24]. Technology scaling has resulted in faster devices but at the same time, the die-to-die delay variations have increased due to different lithographic subtleties. Therefore, low power ALU design while maintaining high yield under tighter delay constraint turns out to be a multi-dimensional problem.

The core unit of ALU is an adder which takes operand from register file, data cache or ALU write back bus.

5.2 ALU Architecture

The design of the ALU can be divided into four parts as Control Unit, Arithmetic Unit, Logical Unit and Shifter Unit. An implemented design of ALU is shown in Fig. 5.1. A 2x4 decoder is used as control unit to select the different unit for desired operation using control signals S_3 and S_2. A functional table of the designed ALU is shown in Table 5.1.

The Arithmetic unit consists of Adder unit and 4x1 Multiplexer unit. Adder Unit is used for addition of two operands while multiplexer unit is responsible for selecting appropriate input operand for adder unit according to the control signal S_2 and S_1. Different logical and shifting operations are selected by another 2x4 decoder having S_1 and S_0 control signals. Shifting Values for shifter unit are adjusted by Sh_5, Sh_4, Sh_3, Sh_2 and Sh_1. These control signals for shifter unit are binary weighted. Hence the maximum value of shift is 31 ($2^5 - 1$). Finally, all the outputs of different units of ALU are

combined with output OR array in which 3 input OR gates are followed by buffer to provides output as 'Y'.

A brief discussion about all components of ALU is followed in subsequent subsections.

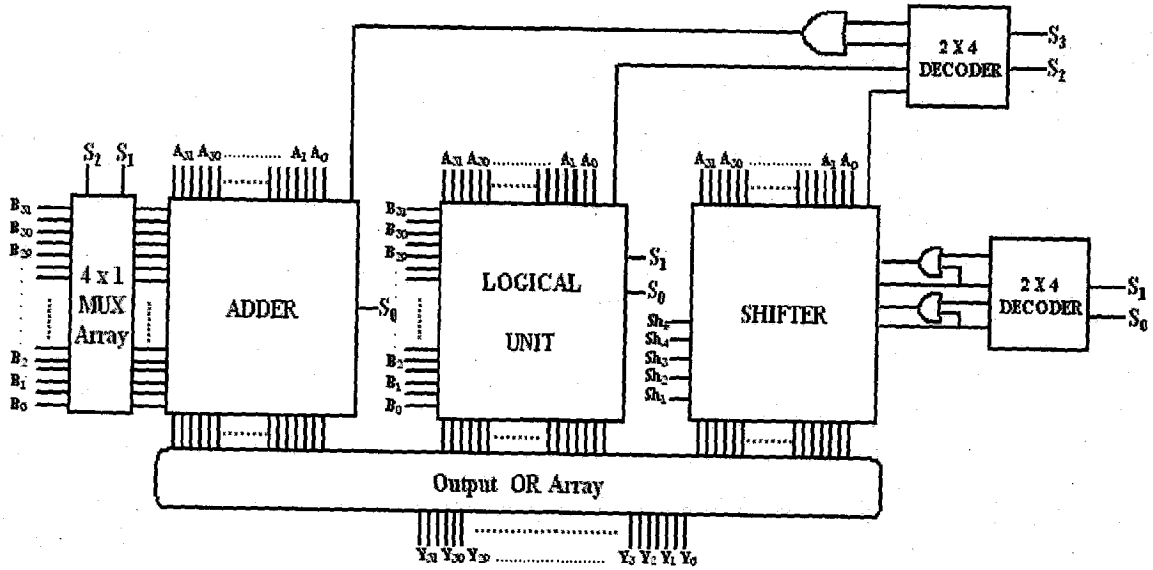


Fig. 5.1: Implemented Design of ALU.

Table 5.1: Functional Table of the designed ALU

Operation Select				Operation	Function
S_3	S_2	S_1	S_0		
0	0	0	0	Addition	$F = A + B$
0	0	0	1	Addition with carry	$F = A + B + 1$
0	0	1	0	Subtraction with barrow	$F = A + \bar{B}$
0	0	1	1	Subtraction	$F = A + \bar{B} + 1$
0	1	0	0	Decrement	$F = A - 1$
0	1	0	1	Transfer	$F = A$
0	1	1	0	Transfer	$F = A$
0	1	1	1	Increment	$F = A + 1$
1	0	0	0	A AND B	$F = A \cdot B$
1	0	0	1	A OR B	$F = A + B$
1	0	1	0	A XOR B	$F = A \oplus B$
1	0	1	1	Complement A	$F = \bar{A}$
1	1	0	0	Logical Left Shift	
1	1	0	1	Circular Left Shift	
1	1	1	0	Logical Right Shift	
1	1	1	1	Circular Right Shift	

5.2.1 Control Unit

A Control Unit in the ALU is responsible to select the desired operation as in functional table (Table 5.1). A control unit is just a 2x4 decoder which can decode the control signal S_3 and S_2. A Decoder is a combinational circuit that converts binary information from *n* input lines to a maximum of 2^n unique output lines. A Functional table of Control Unit shown in Fig. 5.2, is given in Table 5.2. The main idea of controlling different units of ALU using control unit is to control the power supply of different units of ALU. At any instant, the power supply is only given to the unique part of ALU which needs appropriate operation according to control signals and enables that part only to save power.

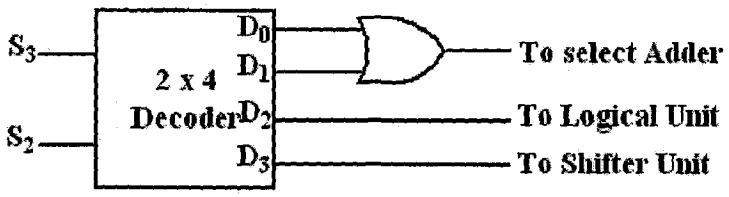


Fig. 5.2: Control Unit of ALU.

Table 5.2: Functional Table of the Control Unit

Operation select S_3	S_2	Function
0	0	Enable Arithmetic Unit Only
0	1	Enable Arithmetic Unit Only
1	0	Enable Logical Unit Only
1	1	Enable Shifter Unit Only

5.2.2 Arithmetic Unit

In a designed ALU, arithmetic unit gives a choice to select seven different operations such as addition, subtraction, increment, decrement and transfer of data. It is enabled by selecting selection line $S_3 = 0$ and $S_2 = 1$ or 0 in control unit. An arithmetic unit is shown in Fig. 5.3 .It consists of heart of the ALU i.e. Adder Unit and a Multiplexer Unit. An Adder Unit is mainly dedicated to perform arithmetic operation in ALU. One of the input operand as "A" is always is given directly to Adder while the other operand is given to adder by the 4x1 MUX (Multiplexer) unit.

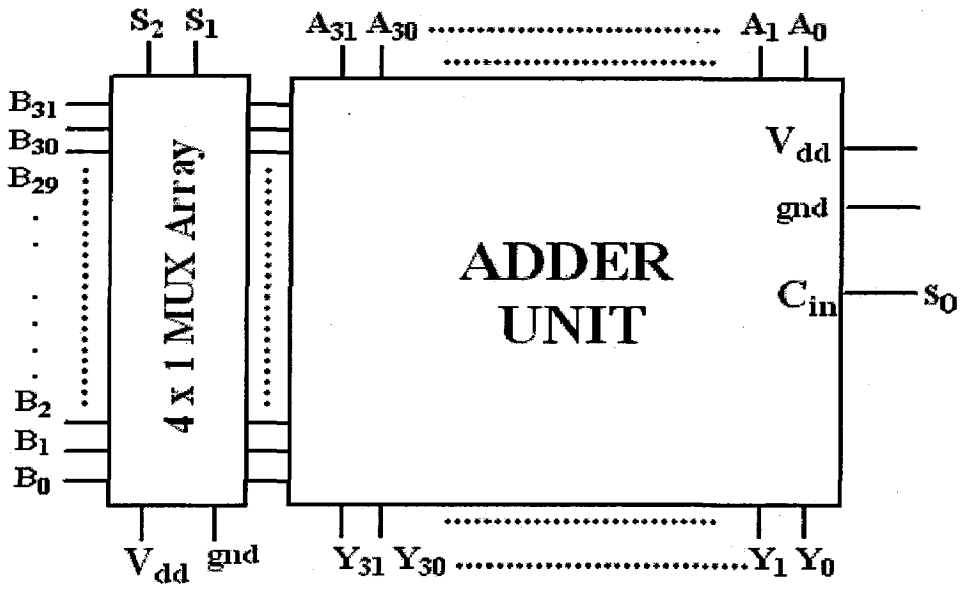


Fig. 5.3: An Arithmetic Unit of ALU.

A Multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output lines. The selection of particular input line is controlled by a set of selection lines. There are 2^n input lines and n selection lines whose bit combinations determine which input is selected. A 4x1 MUX unit as shown in Fig. 5.4 gives a single output W_i which is directly cascaded to second operands of Adder and this output is selected among B_i, $\overline{B_i}$, High Value '1' as 'V_{dd}' and Low Value '0' as 'gnd' using selection lines S_2 and S_1. A functional table of 4x1 MUX is shown in Table 5.3. A 4x1 MUX array is used to provide 32-bit operands as a input to 32-bit Adder. A 4x1 MUX array is consists of 32 4x1 MUX, each MUX unit provides one bit of data as an output.

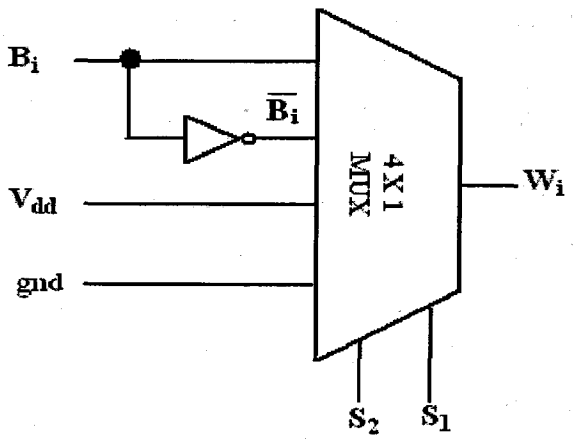


Fig. 5.4: a symbol of 4x1 Mux.

Table 5.3: Functional Table of the 4x1 Multiplexer (MUX)

Operation Select S_2	S_1	Operation	Function
0	0	Select Input B_i as output	$F = B_i$
0	1	Select Input $\overline{B_i}$ as output	$F = \overline{B_i}$
1	0	Select Input 'V_{dd}' as output	$F = 1$
1	1	Select Input 'gnd' as output	$F = 0$

A functional table of arithmetic unit is shown in Table 5.4. A selection line S_0 is used as input carry for the adder. For performing subtraction operation ($F = A-B$) using adder unit, we need a 2's complement of input operand '*B*' as a second input which can be obtained as a combination of 1's complement of '*B*' and Carry input ' C_{in} = 1 or S_0 =1'. Complemented input '*B*' is provided by 4x1 MUX array by selecting $S_2 = 0$ and $S_1 = 1$. Similarly an increment and decrement operation is done by selecting $S_2 = 1$ and $S_1 = 1$ & $S_2 = 1$ and $S_1 = 0$ respectively. A transfer operation refers to a directly transmitting input operand '*A*' without any change.

Table 5.4: Functional Table of the Arithmetic Operation.

Operation Select S_2	S_1	S_0	Operation	Function
0	0	0	Addition	$F = A + B$
0	0	1	Addition with carry	$F = A + B + 1$
0	1	0	Subtraction with barrow	$F = A + \overline{B}$
0	1	1	Subtraction	$F = A + \overline{B} + 1$
1	0	0	Decrement	$F = A - 1$
1	0	1	Transfer	$F = A$
1	1	0	Transfer	$F = A$
1	1	1	Increment	$F = A + 1$

5.2.3 Logical Unit

A Logical Unit of the ALU is responsible for logical operations such as AND, OR, XOR and NOT. Several times, we need complement signal, NOT operation helps to get complement signal. It is very important component of ALU and consumes very less power and area in compared to other parts of ALU. It is very simple to implement as shown is Fig. 5.4. It is enabled by selecting selection line $S_3 = 1$ and $S_2 = 0$ in control unit. A functional table of Logical unit is shown in Table 5.5

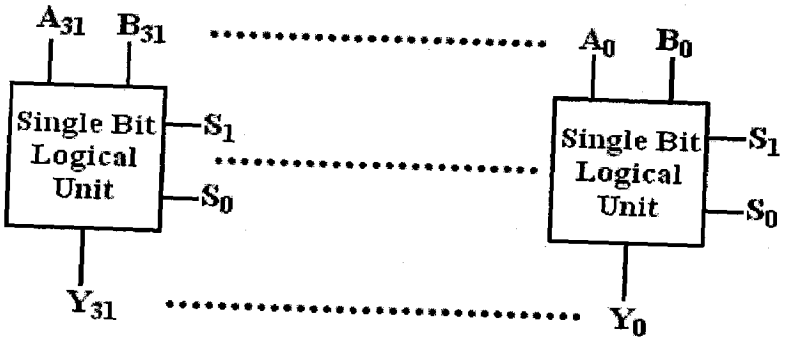


Fig. 5.4: The Logical Unit of ALU.

A single bit Logical unit is shown in Fig. 5.5. A specific logical operation is selected by using selection lines S_1 and S_0.

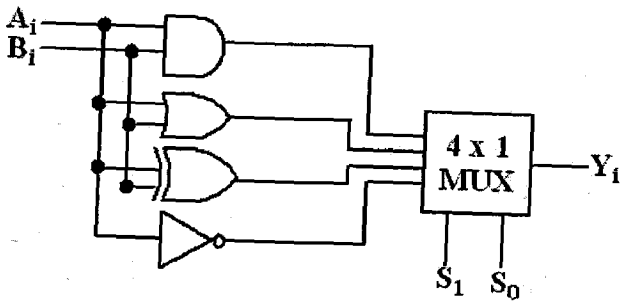


Fig. 5.5: A Single Bit Logical Unit for Logical Unit of ALU.

Table 5.5: Functional Table of the Logical Unit.

Operation Select S_1	Operation Select S_0	Operation	Function
0	0	A AND B	$F = A \cdot B$
0	1	A OR B	$F = A + B$
1	0	A XOR B	$F = A \oplus B$
1	1	Complement A	$F = \overline{A}$

5.2.4 Shifter Unit

A Shifter Unit of the ALU is responsible for bidirectional logical and cyclic or circular shift operations. It is enabled by selecting selection line $S_3 = 1$ and $S_2 = 1$ in a control unit. As a result, the power supply V_{dd} of shifter will be HIGH and shifting operation will occur. A simple shifter unit is shown in Fig. 5.5. A functional table of shifter unit is shown in Table 5.6.

Table 5.6: Functional Table of the Shifter Unit.

Operation Select S_1	S_0	Operation
0	0	Logical Left Shift
0	1	Circular Left Shift
1	0	Logical Right Shift
1	1	Circular Right Shift

A Logical Shift operation refers to shifting of operands bit in particular direction while a shifted position '0' bit is inserted. In a logical left shift, the bits that are shifted out are discarded and zeros are shifted in (on either end). While in the logical right shift, insert bits with value 0 instead of copies of the sign bit. So, the logical shift is suitable for unsigned binary numbers only. A Logical Shift operation is shown below.

Left Shift

7	6	5	4	3	2	1	0
X_6	X_5	X_4	X_3	X_2	X_1	X_0	0

Right Shift

7	6	5	4	3	2	1	0
0	X_7	X_6	X_5	X_4	X_3	X_2	X_1

A Circular Shift operation is also referred as a bit rotation. This operation is useful where it is necessary to retain all the existing bits. It is frequently used in digital cryptography and error control coding. In this shifting, at a shifted position instead of inserting '0' bit, a discarded bit is inserted. It can be easily explained by the following example.

Left Shift

7	6	5	4	3	2	1	0
X_6	X_5	X_4	X_3	X_2	X_1	X_0	X_7

Right Shift

7	6	5	4	3	2	1	0
X_0	X_7	X_6	X_5	X_4	X_3	X_2	X_1

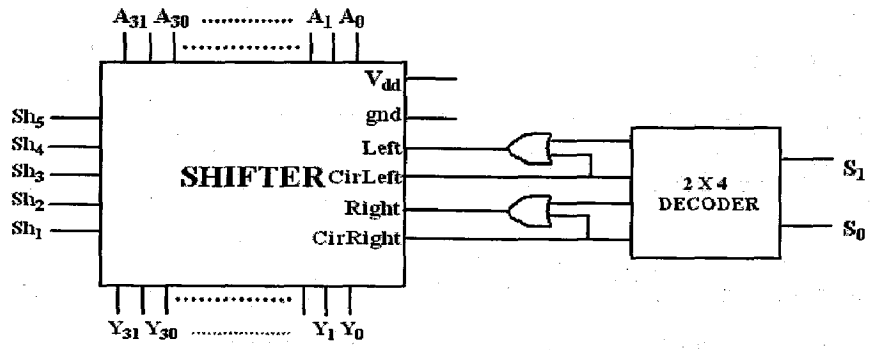


Fig. 5.5: A Shifter Unit of ALU.

5.3 Performance and Results

The following results are reported during simulation for different operations.

(A) Arithmetic Operation

The power consumption and delay in arithmetic operation for different adders using static CMOS logic and FSL are calculated and tabulated in Table 5.7.

Table 5.7: Summary of simulation results for arithmetic operation in designed ALU.

Adder Architecture	Power Consumption (in μW)		Delay (in ps)	
	CMOS	FSL	CMOS	FSL
Ripple Carry Adder(RCA)	510.7	555.1	550.3	480.2
Kogge-Stone Adder	612.05	658.57	410.7	349.5
Han-Carlson Adder	590.6	638.44	505.2	436.4

A comparison of the power consumption in various ALU based on different adder architecture during arithmetic operation among FSL and static CMOS logics is shown in Fig. 5.9.

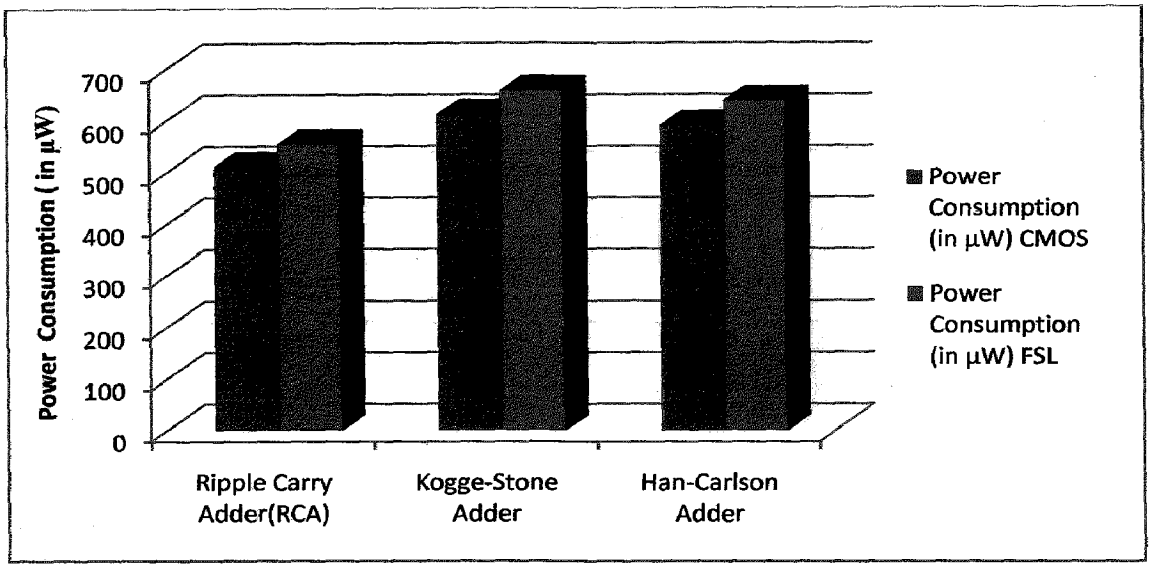


Fig 5.9: Comparisons of power consumption in arithmetic operations in static CMOS and FSL logics.

A comparison of the delay in various ALU based on different adder architecture during arithmetic operation among FSL and static CMOS logics is shown in Fig. 5.10.

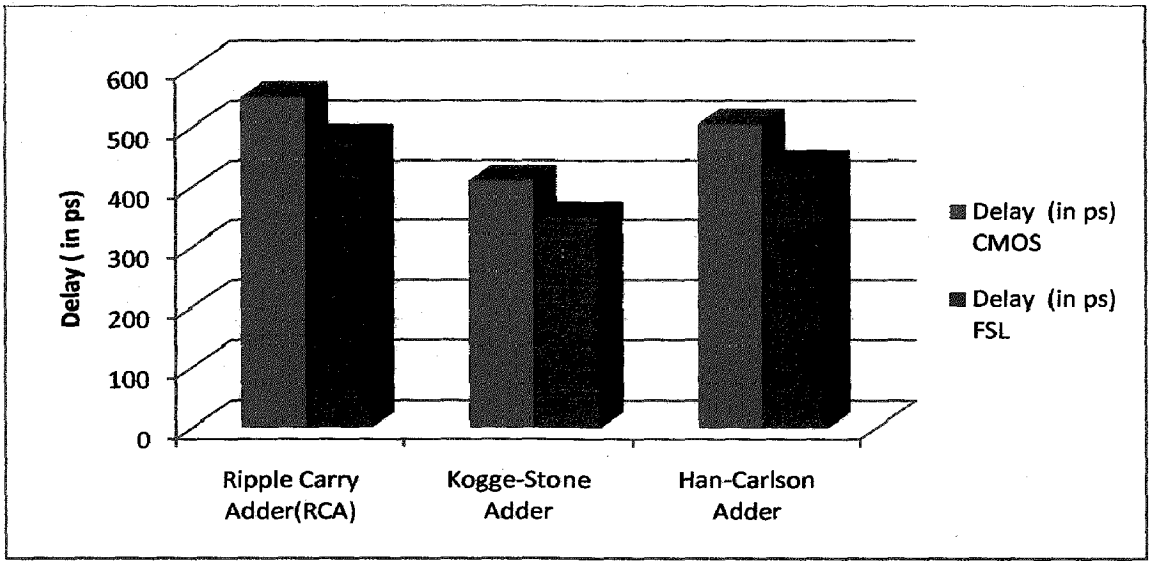


Fig 5.10: Comparisons of delay in arithmetic operations in static CMOS and FSL logics.

From the above results, we can conclude that Kogge-Stone adder architecture gives highest performance in terms of speed in both CMOS and FSL Logics. In terms of power ripple carry adder consumes least among other architecture.

(B) Logical operation

The power consumption and delay in logical operation in ALU using static CMOS logic and FSL are calculated and tabulated in Table 5.8.

Table 5.8: Summary of simulation results for Logical operation in designed ALU.

Logical Operation	Power Consumption (in μW)		Delay (in ps)	
	CMOS	FSL	CMOS	FSL
AND	434.3	461.2	340.7	284.4
OR	471.9	495.9	397.6	339.3
XOR	504.8	539.3	403	357.1

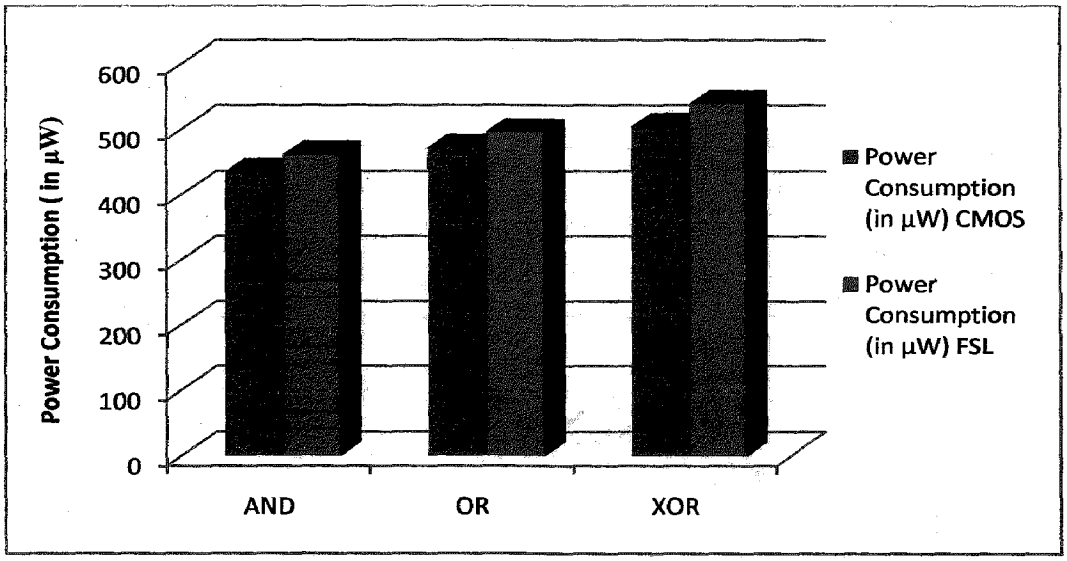


Fig 5.11: Comparisons of power consumption in logical operations in static CMOS and FSL logics.

A comparison of the power consumption and delay in logical operation among FSL and static CMOS logics is shown in Fig. 5.11 and Fig. 5.12 respectively.

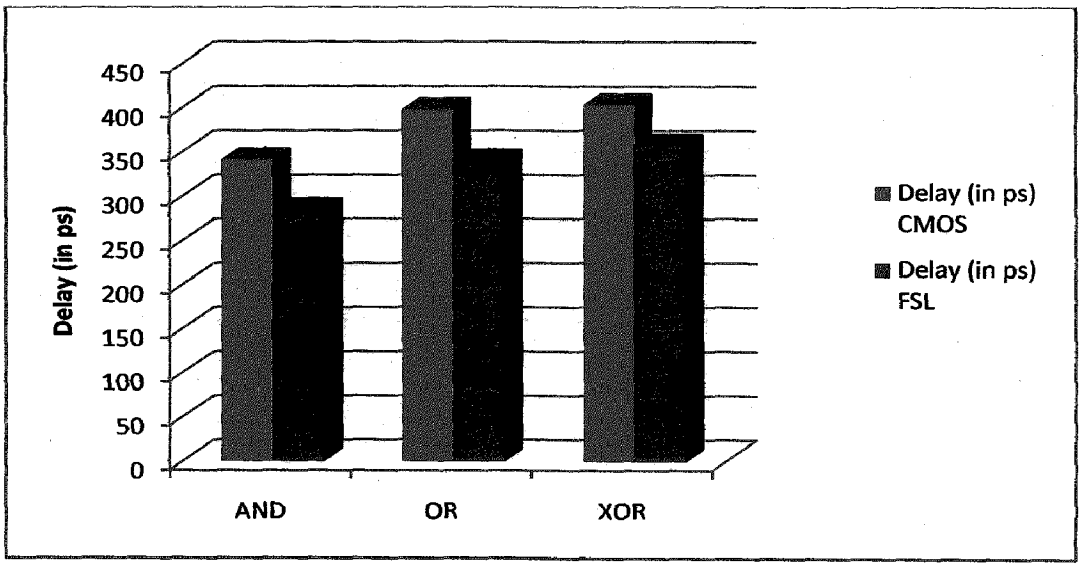


Fig 5.12: Comparisons of delay in logical operations in static CMOS and FSL logics.

From the above results, we can conclude that in FSL Logic about 14 % delay reduces in compared to delay in CMOS logic. But about 6% increases in power consumption is found in FSL.

(C) Shifter Operation

The power consumption and delay in various shifting operation for different shifters using static CMOS logic and FSL are calculated and tabulated in Table 5.9.

Table 5.9: Summary of simulation results for shifting operation in designed ALU.

Shifting Operation	Shifter Architecture	Power Consumption (in μW)		Delay (in ps)	
		CMOS	FSL	CMOS	FSL
Logical Left Shift	Array	229.4	245.7	390.7	381.3
	Logarithmic	237.3	252	360.7	358.2
Logical Right Shift	Array	268.4	288.5	385.9	383.1
	Logarithmic	300.4	314.9	365.6	361.7
Circular Left Shift	Array	415.9	453.3	510.3	507.4
	Logarithmic	436.3	478.2	495.6	483.5
Circular Right Shift	Array	483.6	530	515.2	509.5
	Logarithmic	512.7	562.9	501.1	497.4

A comparison of the power consumption and delay in various shifting operation for different shifters among FSL and static CMOS logics is shown in Fig. 5.13 and Fig. 5.14 respectively.

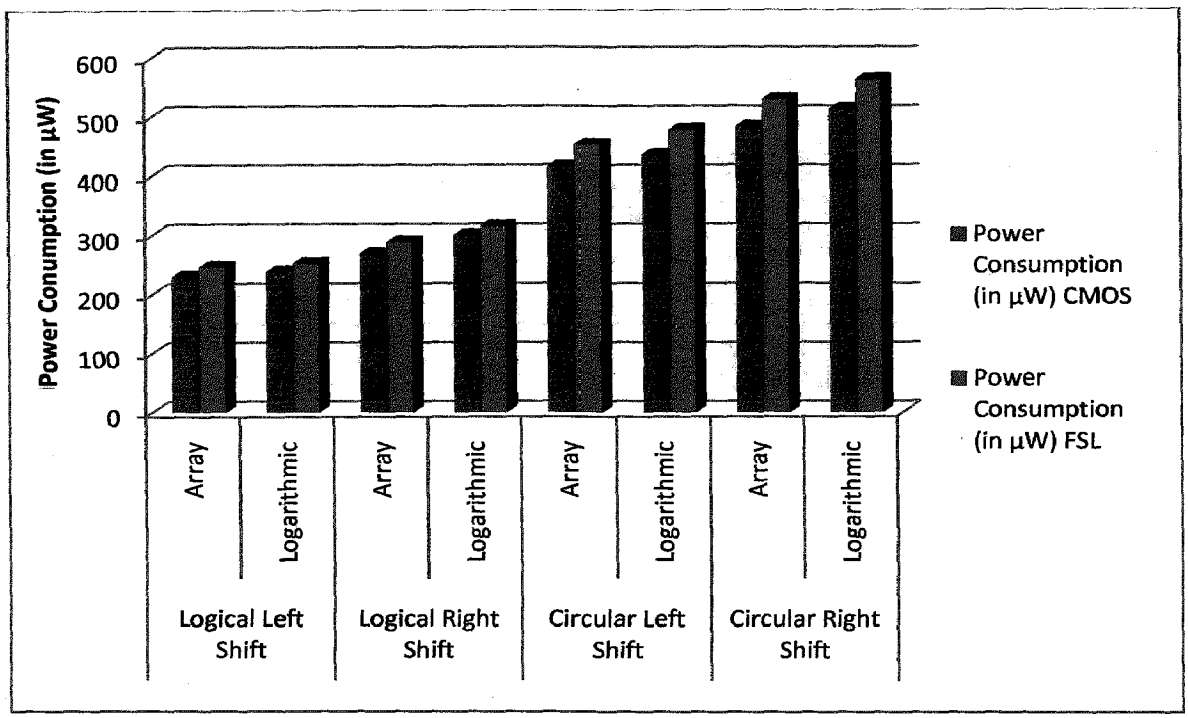


Fig 5.13: Comparisons of delay in shifting operations in static CMOS and FSL logics.

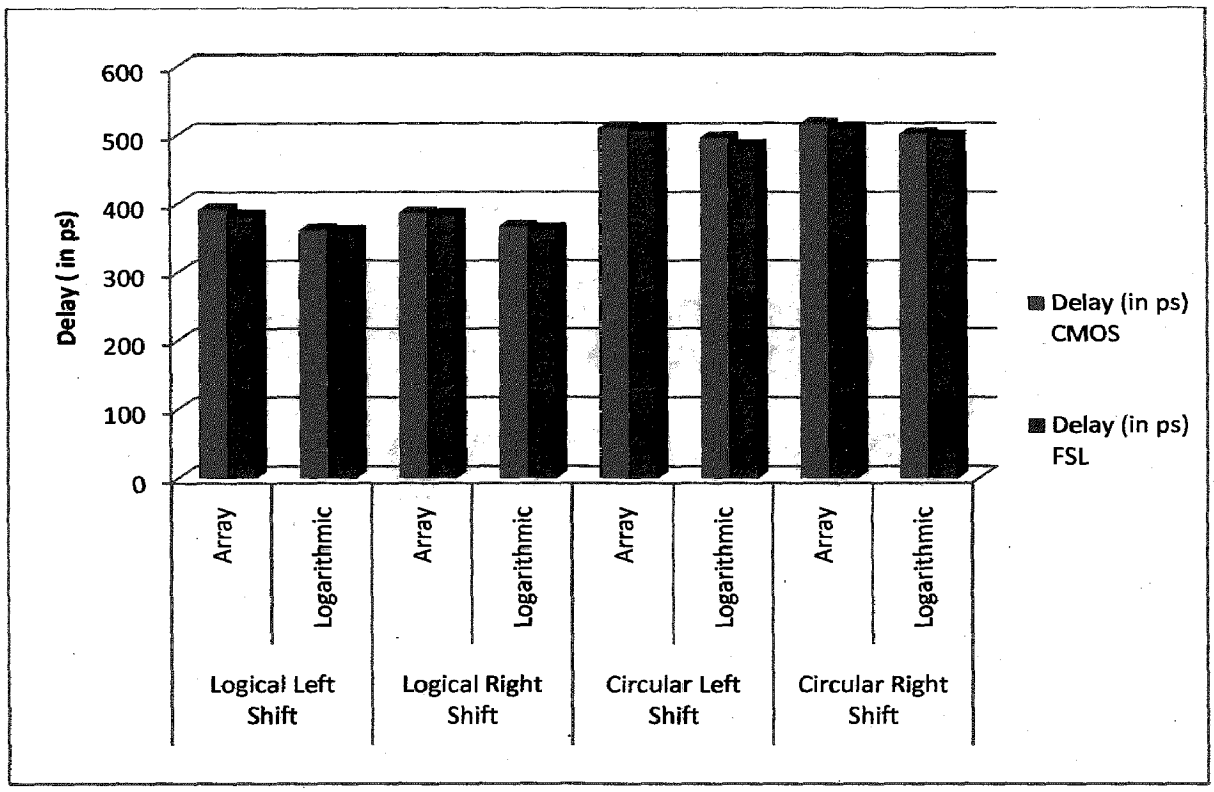


Fig 5.14: Comparisons of power consumption in shifting operations in static CMOS and FSL logics.

From the above results, we can conclude that FSL and CMOS logics both show nearly equal performance in terms of speed. CMOS logic consumes less power in compared to FSL logic.

Chapter 6

Conclusion

6.1 Conclusion

In this thesis, a 32-Bit FSL based Arithmetic Logic Unit has been designed. The advantage of using the scheme of Feedback Switch Logic has been compared to that of the existing static CMOS technology. The primary focus was on the point of exploring and evaluating the performance of FSL in a High Speed Low power design environment.

In Chapter 2, a brief discussion on the need of high speed low power designs in building the architecture of an ALU has been presented. The main operation and functioning of FSL was explained in this chapter. Simulations were performed and comparisons were drawn with static CMOS logic for basic gates which perform the logical operation of the ALU.

Chapter 3 dealt with the design of adder circuits. Adders of different kinds based on architecture were studied and three of them were implemented in FSL and static CMOS logics. Their performances on the basis of power and delay were compared.

In Chapter 4, three types of shifter designs have been discussed. Also, a cyclic shift operation was introduced in Array and Logarithmic shifters which find application in the field of error control coding. The designs were developed in FSL and static CMOS logic structures and the results were compared.

Finally, in Chapter 5, the design of the Arithmetic Logic Unit which combines the Adder and Shifter circuits was presented. Also, a logical unit was designed which consisted of AND, OR, XOR and NOT operations. Once again, the design was developed in both FSL and static CMOS logics. The overall performance analysis was done for the combined arithmetic, logical and shifting operations. Kogge-Stone adder based ALU was found to be a suitable candidate for high speed operation. For low power operation, Ripple-carry adder based ALU was found to be more suitable. This was the same for both logic styles. The performance of shifter units for both logics was found to be almost similar in terms

of both power and delay. For the logical unit, FSL showed a clear advantage in terms of speed with a slight increase in power consumption.

A final comparison between the logic styles is drawn, it has been observed that there was a 14% decrease in delay when we used FSL at the cost of 8% increase in power consumption when compared to static CMOS logic. We can thus recommend the use of Kogge-Stone based FSL adder and FSL based logical unit for achieving high speed, and static CMOS based shifter for maintaining low power. This completes our aim of designing a 32-bit High speed Low power Arithmetic Logic Unit.

6.2 Future Scope

An area minimization is one of the challenging tasks in design of ALUs. Further work could be extended to determine the area and make the layout more area efficient. We can further work to determine interconnect capacitances and to reduce them. Realizing ALUs with some other high speed low power logic can be one of the future works.

Finally, Future work could also include extending the current design for more number of input bits to increase the functionality of ALU.

References

[1] Swaroop Ghosh and Kaushik Roy, "Exploring high-speed low-power hybrid arithmetic units at scaled supply and adaptive clock-stretching," *Asia and South Pacific Design Automation Conference, 2008 (ASPDAC 2008)*, pp.635-640, 21-24 March 2008.

[2] C.J. Akl and M.A. Bayoumi, "Feedback-Switch Logic (FSL): A High-Speed Low-Power Differential Dynamic-Like Static CMOS Circuit Family," *9th International Symposium on Quality Electronic Design, 2008(ISQED 2008)*, pp.385-390, 17-19 March 2008.

[3] A. P. Chandrakasan, S. Sheng and R. W. Brodersen, " Low-power CMOS digital design," *IEEE J. Solid-State Circuits,* vol.27, no.4, pp.473-484, Apr 1992.

[4] V. Friedman and S. Liu," Dynamic logic CMOS circuits," *IEEE J. Solid-State Circuits,* vol.19, no.2, pp. 263-266, April 1984.

[5] Jam M. Rabaey, Anantha Chandrakasan and Borivoje Nikolic, *Digital Integrated Circuits-A Design Perspective, 2nd ed.,* Prentice Hall of India Pvt Ltd, New Delhi, 2006.

[6] F. Murabayashi, T. Yamauchi, H. Yamada, T. Nishiyama, K. Shimamura, S. Tanaka, T. Hotta, T. Shimizu and H. Sawamoto, " 2.5 V CMOS circuit techniques for a 200 MHz superscalar RISC processor, "*IEEE J. Solid-State Circuits,* vol.31, no.7, pp.972-980, Jul 1996.

[7] K.M. Chu and D.L.Pulfrey, "A comparison of CMOS circuit techniques: differential cascade voltage switch logic versus conventional logic," *IEEE J. Solid State Circuits*, vol. 22, no.4, pp.528-532, Aug.1987.

[8] Ahmed M. Shams, Tarek K. Darwish, and Magdy A. Bayoumi, "Performance Analysis of Low-Power 1-Bit CMOS Full Adder Cells" *IEEE Trans. Very Large Scale Integration (VLSI) system,* vol. 10, no. 1, pp 20-29, February 2002.

[9] A. Weinberger and J. L. Smith, "A logic for high-speed addition," *National Bureau of Standards Circular*, vol. 591, pp. 3-12, 1958

[10] R. W. Doran," Variants of an improved carry look-ahead adder, "*IEEE Trans. on Computers*, vol. 37, no. 9, pp. 1110-1113, Sep.1988.

[11] Neil H E Weste and Kamran Eshraghian, *Principles of CMOS VLSI Design A systems Perspective*, 2' Edition, Addison-Wesley Publication 1992.

[12] Massimo Alioto and Gaetano Palumbo, "Analysis and Comparison on Full Adder Block in Submicron Technology" *IEEE Trans. Very Large Scale Integration (VLSI) system,* vol. 10, no. 6, pp 806-823, December 2002.

[13] John P. Uyemurya, *"Introduction to VLSI Circuits and Systems"*, John Wiley & Son Publication, 2002.

[14] Behrooz Parhami, *"Computer Arithmetic: Algorithms and Hardware Design"*, Oxford University Press, Oxford, 1999.

[15] Y. Choi and Jr. E. E. Swartzlander, "Parallel prefix adder design with matrix representation," *17th IEEE Symposium on Computer Arithmetic, 2005 (ARITH-17 2005),* pp. 90-98, 27-29 June 2005.

[16] P. M. Kogge and H. S. Stone. "A parallel algorithm for the efficient solution of a general class of recurrence equations". *IEEE Trans. on Computers*, vol. 22, pp.786–793, Aug. 1973.

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

[18] T. Han and D. A. Carlson. "Fast area-efficient VLSI adders." *Proc. 8th IEEE Symposium on Computer Arithmetic*, pages 49–56, May 1987.

[19] Haikun Zhu, Yi Zhu, Chung-Kuan Cheng and D.M. Harris, "An interconnect-centric approach to cyclic shifter design using fanout splitting and cell order optimization," *Asia and South Pacific Design Automation Conference, 2007. ASP-DAC '07.*, pp.616-621, 23-26 Jan. 2007.

[20] K.P. Acken, M.J. Irwin and R.M. Owens, "Power comparisons for barrel shifters," *International Symposium on Low Power Electronics and Design, 1996,* pp.209-212, 12-14 Aug 1996

[21] R. Ramadoss, "A new breed of power-aware hybrid shifters," *IEEE International Conference on SOC, 2005.* pp.143-146, 19-23 Sept. 2005

[22] Matthew R. Pillmeier, Michael J. Schulte and E. George Walters III, "Design alternatives for barrel shifters", *Advanced Signal Processing Algorithms, Architectures, and Implementations XII, Franklin T. Proceedings of the SPIE,* Volume 4791, pp. 436-447 (2002).
available at http://www.egwalters.com/publications/c_2002_SPIE_2.pdf

[23] Chandra Srinivasan, "Arithmetic Logic Unit (ALU) Design using Reconfigurable CMOS Logic", *M.S. thesis, Louisiana State University*, December 2003.

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

G14979