MULTIPLEXER AS FUNCTION GENERATOR IN FPGA

Project report submitted in partial fulfilment of the requirements for the degree of B.Tech. CSE

by

RATHOD JAGADISH
(COE17B046)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, DESIGN AND MANUFACTURING, KANCHEEPURAM

May 2021

Certificate

I, **Rathod Jagadish**, with Roll No: **COE17B046** hereby declare that the material presented in the Project Report titled **"MULTIPLEXER AS FUNCTION GENERATOR IN FPGA"** represents original work carried out by me in the **Department of Computer Science and Engineering** at the **Indian Institute of Information Technology, Design and Manufacturing, Kancheepuram** during the years **2017–2021**. With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: **06/05/2021** Student's Signature

In my capacity as supervisor of the above-mentioned work, I certify that the work presented in this Report is carried out under my supervision, and is worthy of consideration for the requirements of Project work during the period January 10 2021 to May 06 2021.

[illegible]

Advisor's Name: **Dr. Noor Mahammad Sk** Advisor's Signature

Abstract

Re-configurable hardware Fabrics uses multiplexer as function generator to implement logic function. Multiplexer is a universal element and one can implement any primitive gates using the multiplexer. The Objective of this work in to analyze the all the possible functions that one can implement on C-cell(AX-2000 FPGA). Fault analysis on C-cell, and design a fault tolerance function using c-cell.

Acknowledgements

I'd like to express my heartfelt gratitude to **Dr. Noor Mohammad**, my project guide, for his support and motivation during the project. First and foremost, I want to express my gratitude to Ph.d Schlolar **Raghavendra Kumar** for his assistance and assistance in monitoring and completing the project. Thanks to their constructive criticism and support, all of the project's short and long term objectives were met. and I am really thankful to them.

I'd also like to express my gratitude to my lab mates, who have been extremely supportive and helpful during this project. I'd like to express my gratitude to everyone who assisted me with the Final Year Project once more.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

FPGAs are gaining favor among designers because their ability to be reconfigured from a desktop computer allows for easier prototyping, faster time to market, and lower nonrecurring engineering (NRE) costs when opposed to custom integrated circuit (IC) designs. Lower FPGA costs, increased FPGA efficiency, and, perhaps most importantly, a driving need for the rapid time-to-market allowed by FPGA technology are all contributing to this pattern. Sensitive interlock logic can be enforced by FPGA-based control systems and they can also be programmed to resist I/O pushing by an operator. FPGA-based devices have the ability to essentially rewire their internal circuits,allowing for reconfiguration once the control system has been implemented in the field. VLSI devices, such as FPGAs, are prone to faults. The internal circuitry of the FPGA is wired in a system that gives a hardware implementation of the software application during initialization. Various fault-tolerance methods are being implemented in order to improve the stability and dependability of FPGA-based applications. The aims of fault-tolerance strategies are to reduce hardware, scheduling, and power overhead while increasing device stability.

1.1 Motivation

Fault-tolerant re-configurable computing is the need of the hour for military and space applications. These hardware architectures suffer from single-event-upset (SEU). SEU in the reconfigurable hardware fabrics can lead to system failure. The main objective of this project is to design and find the fault technique re-configurable architecture. This work focuses on the design and development of the fault-tolerant c-cell. C-cell is a multiplexer-based function generator. We do fault analysis and fault modeling of the C-cell.

1.2 Background Research

This section gives a summary of fault-tolerant strategies that are applicable to our FPGA architecture discussion. We begin by looking at the FPGA's structure and noting the use of circuits that speed up arithmetic operations, Fault tolerance with the least possibilities, and the proposed design and implementation of the fault-tolerant multi-bit adders and multi-bit multipliers using c-cell using Single event upset(SEU) in the reconfigurable architecture.This section also goes into several significant fault-tolerant strategies that are applicable to our conversation.

1.3 Problem Statement

Designs used in the Space and Military applications use Micro-sim based FPGA, Where programming is done using the Anti-Fuse Technology. The basic logic cell used in the implementation is the C-Cell. All the combinational and sequential circuits are implemented using C-cell and registers cell.

- The objective of this work is to do an error analysis on the C-cell based combinational and sequential circuits.
- Device a mechanism for fault-tolerant design
- Fault analysis on C-Cell and Fault Modeling of C-Cell.
- Possible approaches for fault tolerance..0.0

1.4 The Report's Organization

The report has been divided into several Chapters. First Chapter deals with the introduction of the Project, Problem statement, and Contribution of the work done in this regard. The second chapter tells about a study in this regard that has been tried to implement motivated me for doing work in this area. The third chapter is about the proposed system I am working with and basic requirements and working with the problem statement. The final chapter deals with programming and results, work done, and future work that can be done in this area.

Chapter 2

Literature Survey

2.1 Architecture of the System

The SX-A family architecture is referred to as a ”sea of modules” architecture because it covers the entire device or with a grid of logic modules, with almost no chip area missing to interconnect elements or routing. The Actel SX-A family of logic modules includes two types: register cells (R-cells) and combinatorial cells (C-cells).

2.2 Interconnect Element with Programmable

The Axcelerator family uses a proprietary surface-to-metal programmable interconnected portion between the top two layers of metal. The routing and connection between logic modules (as needed by older FPGAs) will be fully eliminated and the design of the sea of modules is cost-effectual. Anti-fuses are usually open circuits, which form a permanent, passive, low electrical resistance connection until set up to provide the industry with the quickest signal propagation. The small size of these interconnecting modules also provides the Axcelerator family with torrential routing services.

Non-volatile anti-fuse technology from micro-semi provides great protection against theft and biological experiments because it looks bad (Fuse-Lock technology). Never download or hold a bit-stream or programming file on the device, which makes cloning difficult (even if the safety fuse is left deprogrammed). Reverse engineering is almost impractical because of the difficulties of separating programmed from deprogrammed antifreeze and programming methods for anti-fuse devices.

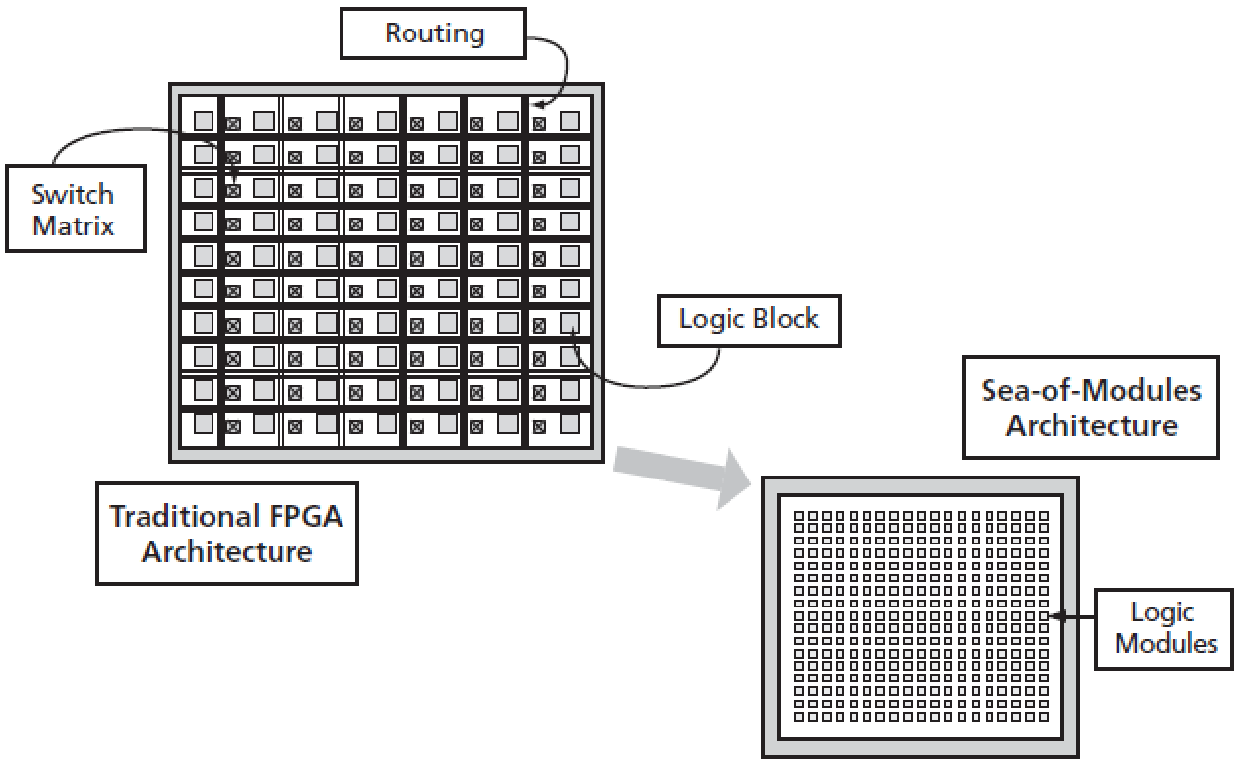

FIGURE 2.1: Sea of Modules Comparison

2.3 Logic Modules

The R-cell and the C-cell are two logic units that the Actel family Axcelerator provides (C-cell). About 4,000 functions can be combined with up to five inputs in this C-cell.

- A flip-flop in the R-cell comprises asynchronous simple, asynchronous predetermined, and Active-low changes management signal(Figure 2.3).
- Register by register can be programmed for the clock polarity of the R-cell registers.
- This makes it easier to map dual data rates into the FPGA (for example, easier to map) while retaining significant clock resources.
- Hardwired clocks, routed clocks, and internal logic are all options for the R-cell clock supply.

- A Cluster is made up of 2 C-cells, 1 Register cell(R-cell), and 2 Transmit (TX) and Receive (RX) routing buffers, while a SuperCluster is made up of two Clusters (Figure 2.4).

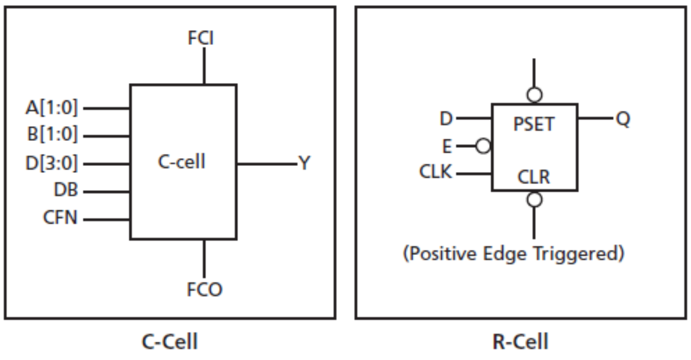

FIGURE 2.2: AX C-cell and R-cell

- Every SuperCluster additionally includes a free Buffer (B) module, which allows the place-and-route tool to insert buffers on high-fanout nets, reducing device delays while increasing logic utilisation.
- The logic modules at intervals the Super-cluster are organised in order that 2 combinatorial modules are side-by-side, giving a C-C-R C-C-R pattern to the Super-cluster.
-

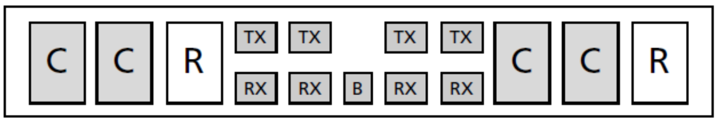

FIGURE 2.3: Ax SuperCluster

The AX design is absolutely fracturable, that means that if one or a lot of of the logic modules during a SuperCluster are employed by a selected signal path, different the opposite logic modules are still obtainable to be used by other methods.

2.4 C-Cell Module Specifications

The C-cell is one of 2 logic module groups in the AX architecture. It is the combinatorial logic resource of the Axcelerator device. The AX architecture employs a replacement combinatorial cell, which is an extension of the C-cell found in the SX-A family. The most important advancement of the new C-cell is the addition of carry-chain logic.

The C-cell is often used to create arithmetic functions in a carry-chain mode. If carry-chain logic isn't required, it's always turned off. The C-cell has the following (Figure 2.4):

- MUX with 8-inputs (data: D0-D3, select: A0, A1, B0, B1). All of these inputs can be used to channel user signals. Any of the C-cell inputs (D0-D3, A0, A1, B0, B1) can be connected to one of the four routed clocks (CLKE/F/G/H).
- The inverter (DB input) can be used to move a complement signal from each of the C-cell inputs.
- There is a carry input and a carry output. The C-carry cell's input is the carry output from the C-cell to the north.
- Carry link with a signal transmission time of less than 0.1 ns for carry-chain logic.
- Both C-cells on the side of a SuperCluster have a hardwired link (direct connect) to the neighbouring R-cell (Register Cell) with a signal propagation time of less than 0.1 ns.

This C-cell (and C-cell Cluster) architecture allows for the introduction of over 4,000 functions of up to 5 bits. For example, two C-cells may be used in tandem to implement a four-input XOR feature in a single cell delay.

Actel's comprehensive macro library handles the carry-chain setup for the user (for a complete list of available Axcelerator macros, see Actel's Antifuse Macro Library Guide).

2.5 Multiplexer Logic as Function Generators

- There are a lot of logic cells in an FPGA. Each logic cell may be programmed to perform a specific series of tasks.
- The FPGA (Field Programmable Gate Array) is a digital integrated circuit with a matrix of user-programmable logic cells that can implement complex digital circuitry.
- These gates are used to implement combinational and sequential functions that are specified by the user.
- Different vendors' FPGA families use various logic cell architectures.
- The number of inputs and outputs in each logic cell is set.

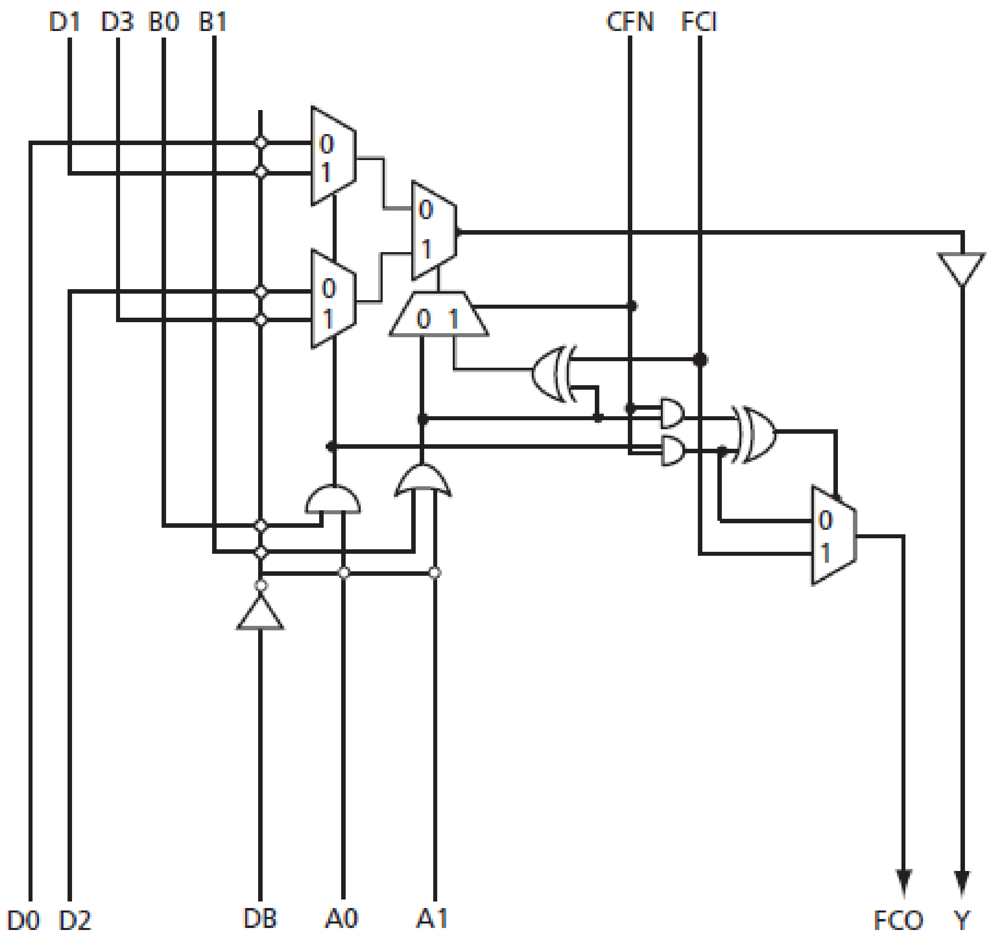

FIGURE 2.4: C-cell

- Logic cells employed in FPGAs are
  - Logic cells with multiplexers (e.g. Actel FPGAs).
  - Logic cells of memory (e.g. Xilinx FPGAs)

2.6 Fault analysis of C-Cell

- The MUX-based logic module is usually made up of 2X1 Multiplexers are using primitive gates such as NOT, AND, OR, NOR, NAND, XOR, and XNOR.
- The MUX-based logic module is usually made of 4X1 Multiplexers using different Primitive gates.

- Fault Analysis is done with Multi-bit Adders using full-Adder C-Cell and multi-bit Array multiplier using single C-Cell.
- In order to calculate the for SEU (Single Event Upset) tolerance analysis, a few steps must be taken. These three stages are often used in fault injection studies.
    1. Fault Target Location
    2. Fault Injection
    3. Observation of Fault Consequences

Fault Target Location: Select lines in the C-Cell are the most common targets for model-based fault injection techniques.

Fault Injection: Fault injection is testing technique used in the digital system to test software and hardware model. which introduce the faults into the system and subsequent of the system of the errors and failure of the proposed C-cell, with error changes in the significant data line using output.

Observation of Fault Consequences: Once the fault has been injected, it's necessary to look at however the system reacts. Usually, a trace of the outputs and also the state of the system is hold on for its interior analysis. once one in all the a lot of totally different completely different parameters is modified throughout this injection of the choose line victimisation 2 different AND and OR gates within the C-Cell.

This paper provides an summary of single-event upsets (SEU), the capabilities provided in FPGAs to mitigate the results of SEU, techniques that may be incorporated in user styles to mitigate the results of SEU, and the way tools and intellectual property(IP) are often accustomed validate a style to make sure high levels of tolerance for SEU.

To handle the SEUs, a standard technique is employed to observe and proper the errors. once associate upset happens, the configuration changes which may cause the FPGA to control in unsought behavior. for instance, a a pair of choose line of Muxes that's set to be associate logic gate and gate would have a configuration of "0001" wherever the last bit represents the case wherever each inputs area unit one. Suppose associate upset happens and changes the configuration to "0101". This makes the Muxes operate as a buffer that continually passes the worth of the second input to the output.

Chapter 3

Methodology

3.1 Realization of Primitive gates using 2X1 Multiplexer

The multiplexer is a circuit with digital mixtures. Among multiple data inputs, one or a lot of select choose inputs. only 1 combination of inputs are as output. Signals are passed from one in every of the inputs to output. therefore there are seven differing types of Preemptive gates used to build 2X1 mux that are using here.

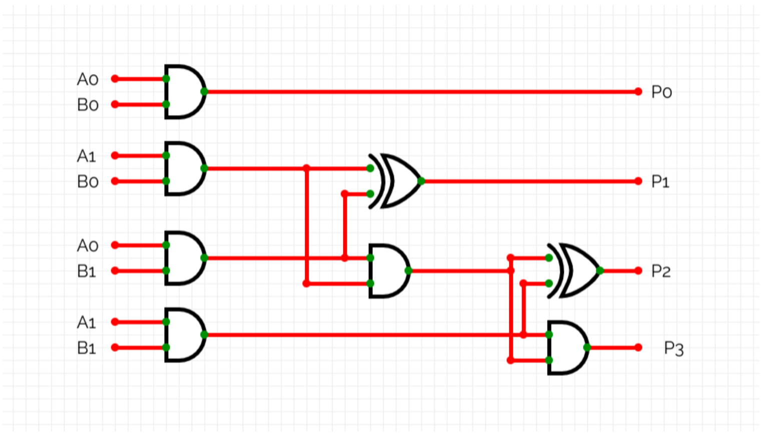

FIGURE 3.1: Circuit diagram of 2X1 MUX

- Multipliers are a kind of adder array that is extremely complicated. This is a standard procedure in a wide variety of applications, and because of its complexity, there has been a lot of research done to try to speed up its implementation.
- Multipliers can be Designed using a variety of hardware. The efficiency characteristics of the multipliers differ depending on the law used.

- Binary adders are important in the use of digital multipliers. With the rise of power as a way of thinking, speed is no longer the primary standard on which many applications are measured.
- By designing multipliers with low-power, energy-efficient adders, the potential usage and skill of multipliers are minimised.
- To investigate the output of these 3 digital multipliers, we required them to use 3 full adder cells.

3.1.1 2X1 Mux using OR gate

we start with 2X1 MUX using 2 inputs are 'A' and 'B', one select line 'S'. The output is 'Y'. The equation is shown below

$$Output = A * S + B * \bar{S}$$

In ORing expression '+' used for above equation. which means we cann't tie S to 0 or 1. So we are make zero to S and it should disappear

If 'S = 0'

$$Output = A * 0 + B * \bar{0} = 0 + B = B$$

If 'S = 1'

$$Output = A * 1 + B * \bar{1} = A + 0 * B = A$$

Similarly, we will keep the '+' concept going by not tying either 'A' or 'B' to zero.

Let's start with letter 'A' and bind it to one.

$$Output = S * 1 + \bar{S} * B$$

$$Output = S + \bar{S} * B$$

We wind up with something like this.

$$Output = S + B$$

is what we really want. You will look at equations whether you are an expert in Boolean algebra or digital logic.

$$Output = S + \bar{S} * B$$

and explain why it's the same as

$$Output = S + B$$

But don't be concerned. We'll show you how right you are.

$$Output = S + \bar{S} * B$$

$$\overline{Output} = \overline{S + \bar{S} * B}$$

[taking bar of both sides]

$$\overline{Output} = \bar{S} * \overline{(\bar{S} * B)}$$

[De-morgan's rule on right hand side]

$$\overline{Output} = \bar{S} * \overline{S + \bar{S}}$$

$$\overline{Output} = \bar{S} * S + \bar{S} * \bar{S}$$

[Expanding right hand side]

$$\overline{Output} = 0 + \bar{S} * \bar{B}$$

[Simplifying first term on right hand side]

$$\overline{Output} = \bar{S} * \bar{B}$$

$$\overline{Output} = \overline{(S + B)}$$

[De-morgan's rule on right hand side]

$$Output = S + B$$

[Taking bar of both sides]

As a result, we show that for a 2X1 MUX, tying input "A = 1" results in an OR gate.

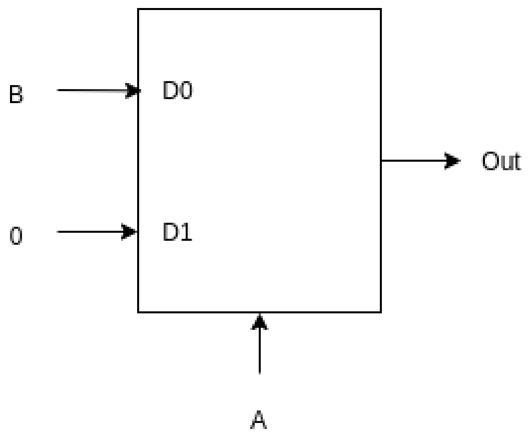

FIGURE 3.2: Logic diagram of 2X1 MUX using OR Gate

A	B	Output = OR
0	0	0
0	1	1
1	0	1
1	1	1

TABLE 3.1: 2X1 MUX truth table for an OR gate

3.1.2 2X1 Mux using AND gate

As we discussed in the previous subdivision, we'll start with the MUX equation and scale it down to the equation of an AND gate, as seen below.

The equation for a 2X1 MUX with inputs A and B, pick line S, and output Out is as follows.

$$Output = S * A + \bar{S} * B$$

The initial product concept inside the product add on the right side is clearly an AND process. What we want to do is exclude the second product expression. By setting 'B = 0', we are able to do this. We'll set the MUX's B input to zero.

$$Output = S * A + \bar{S} * 0$$

$$Output = S * A + 0$$

$$Output = S * A$$

By fixing the B input of the MUX to zero, we can create an AND gate; the resulting circuit is a gate that ANDs input A while the mux selects S.

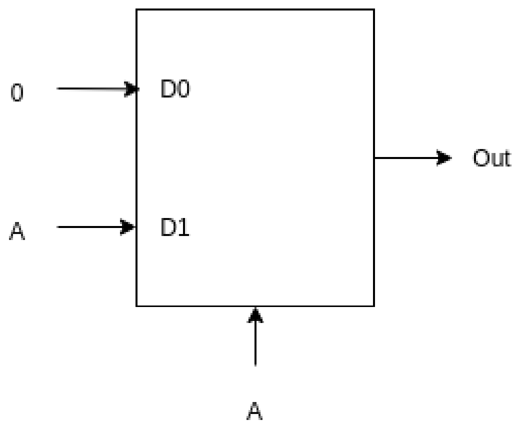

FIGURE 3.3: Logic diagram of 2X1 MUX using AND Gate

A	B	Output = AND
0	0	0
0	1	0
1	0	0
1	1	1

TABLE 3.2: 2X1 MUX truth table for an AND gate

3.1.3 2X1 Mux using NOT gate

The MUX has two inputs, A and B, as well as a select pin and an output pin.

$$Output = S * A + \bar{S} * B$$

When we want to make a NOT gate, we usually look for the shape's equation.

$$Output = S * 0 + \bar{A} * 1$$

$$Output = \bar{A}$$

If the NOT input is zero, the input zero that is bound to one is selected.

A	Output = NOT
0	1
1	0

TABLE 3.3: 2X1 MUX truth table for a NOT gate

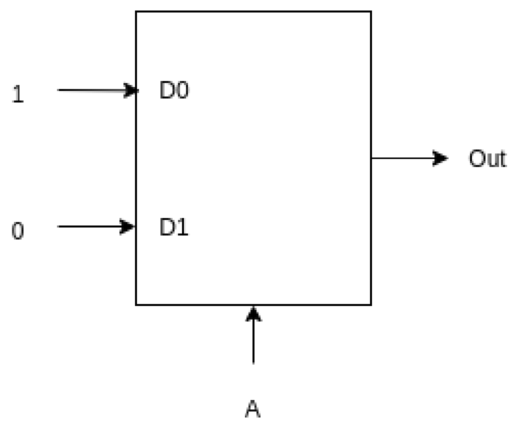

FIGURE 3.4: Logic diagram of 2X1 MUX using NOT Gate

3.1.4 2X1 Mux using NOR gate

As previously said, we usually start with the 2X1 MUX equation, which looks like this. The MUX has two inputs, A and B, as well as a pick pin, S, and an output pin, Output.

$$Output = S * A + \bar{S} * B$$

As we wish to create a NOR gate we tend to find the equation of the form

$$Output = \overline{(A + B)}$$

Just keep in mind that we don't have to use the same pins to get the same equation; we can use either of the three input pins A, B, or S.

So, where do we begin? This is something I sometimes forget to mention when forming a NAND wall, but it applies to both NAND and NOR gates. Using 2X1 MUX, we've already built a gate, OR gate, and inverter. You'll be able to build NAND by combining gate and electrical converter. Similarly, Combining a logic gate and an electrical converter yields a NOR gate. Coming back with a NOR gate using 2X1 MUX would be one choice.

We'll try to come up with a NOR gate using a certain method here.

$$Output = S * A + \bar{S} * B$$

and we want to convert this to a NOR equation.

We want form

$$Output = \overline{(A + B)}$$

which is same as

$$Output = \bar{A} * \bar{B}$$

You can see in our 2X1 MUX equation that we need to own $\bar{B}$ in place of B and that the 'S * A' expression needs to be zeroed out.

We can get to each by substituting $\bar{B}$ for B and connecting 0 to input A.

$$Output = S * 0 + \bar{S} * \bar{B}$$

$$Output = \bar{S} + \bar{B}$$

$$Output = \overline{(S + B)}$$

[DeMorgan's rule]

That is our NOR gate.

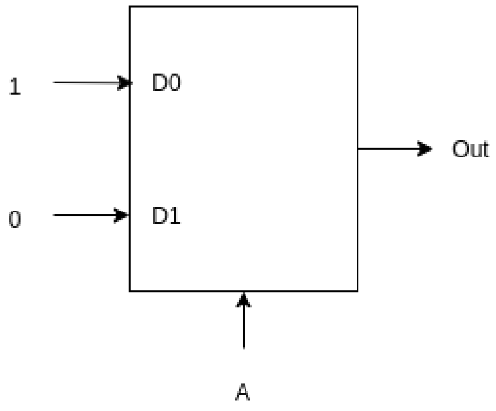

FIGURE 3.5: Logic diagram of 2X1 MUX using NOR Gate

A	B	output = NOR
0	0	1
0	1	0
1	0	0
1	1	0

TABLE 3.4: 2X1 MUX truth table for a NOR gate

3.1.5 2X1 Mux using NAND gate

The MUX has 2 inputs, A and B, as well as a select pin S and an output pin is Output.

$$Output = S * A + \bar{S} * B$$

We'll need a NAND gate to complete the circuit, so the equation for a NAND gate is:

$$Output = \overline{(A * B)}$$

or

$$Out = \bar{A} + \bar{B}$$

[using De Morgan's rule]
Given the calculations we've got to work with, it seems that we'll have a better chance of improving. This is merely speculative.

$$Output = S * A + \bar{S} * B$$

to the latter kind of NAND equation.

$$[Output = \bar{A} + \bar{B}]$$

If we tend to really $\bar{A}$ rather than of pin A to the MUX and if we tie 1 to the B input

$$Output = S * \bar{A} + \bar{S} * 1$$

$$Output = S * \bar{A} + \bar{S}$$

Now lets prove that this is same as

$$Out = \bar{A} + \bar{S}$$

$$Output = S * \bar{A} + \bar{S}$$

$$\overline{Output} = \overline{(S * \bar{A} + \bar{S})}$$

$$\overline{Output} = \overline{(S * \bar{A})} * \bar{\bar{S}}$$

$$\overline{Output} = \overline{(S * \bar{A})} * S$$

$$\overline{Output} = \overline{(\bar{S} + \bar{A})} * S$$

$$\overline{Output} = (\bar{S} + A) * S$$

$$\overline{Output} = \bar{S} * S + A * S$$

$$\overline{Output} = 0 + A * S$$

$$\overline{Output} = A * S$$

$$Output = \overline{(A * B)}$$

By binding $\bar{A}$ instead of A and tying B to 1, we were able to create a NAND gate.

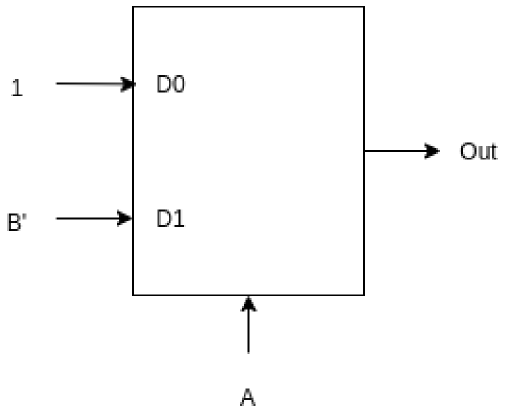

FIGURE 3.6: Logic diagram of 2X1 MUX using NAND Gate

A	B	Output = NAND
0	0	1
0	1	1
1	0	1
1	1	0

TABLE 3.5: 2X1 MUX truth table for a NAND gate

3.1.6 2X1 Mux using XOR gate

The XOR gate is a unique gate. The reality table of the XOR gate is something we're all too familiar with.

To represent the XOR action, we commonly use the symbol OR '+' with a circle around it. There's another way to explain the XOR process, which, as you'll see, supports the truth table. K-maps are one of the most straightforward ways to find an equation representation of use K-maps.

$$Output = A * \bar{B} + \bar{A} * B$$

With this equation in hand, it's much simpler to begin with the 2X1 MUX equation and translate it to the desired XOR equation. The MUX equation for a 2X1 is:

$$Output = S * A + \bar{S} * B$$

We can see that if we replace A with $\bar{B}$, we get what we want.

$$Output = S * \bar{B} + \bar{S} * B$$

For S and B, this is the XOR. The figure for equivalent is shown below.

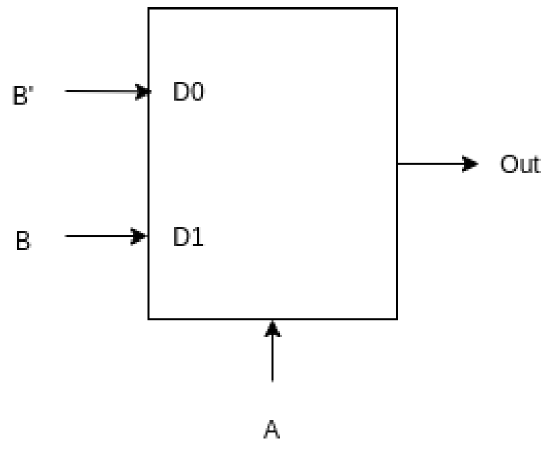

FIGURE 3.7: Logic diagram of 2X1 MUX using XOR Gate

A	B	Output = XOR
0	0	0
0	1	1
1	0	1
1	1	0

TABLE 3.6: 2X1 MUX truth table for a XOR gate

3.1.7 2X1 Mux using XNOR gate

The following is an alternative equation for an XNOR gate

$$Output = \bar{A} * \bar{B} + A * B$$

K-maps can be used to quantify this, as shown below. We know how to make a 2X1 MUX using the following formula

$$Output = S * A + \bar{S} * B$$

$$Output = \bar{A} * \bar{B} + A * B$$

This appears to be to be a simple task. If we substitute $\bar{A}$ for B in the 2X1 MUX equation, we get

$$Output = S * A + \bar{S} * \bar{A}$$

This is the XNOR gate equation for inputs S and A.

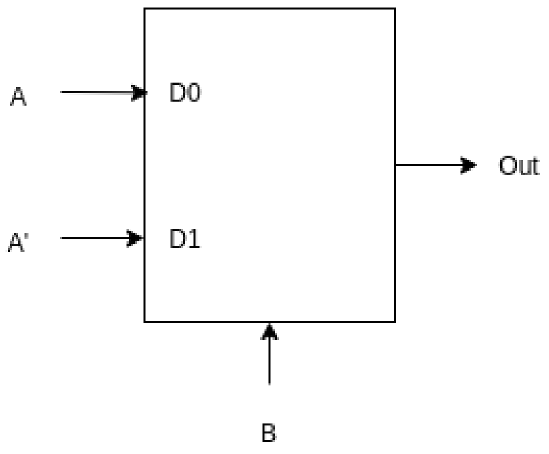

FIGURE 3.8: Logic diagram of 2X1 MUX using XNOR gate

A	B	Output = XNOR
0	0	1
0	1	0
1	0	0
1	1	1

Table 3.7: 2X1 MUX truth table for a XNOR gate

3.2 Realization of Primitive gates using 4X1 Multiplexer

The 4x1 Multiplexer has four data inputs (D0, D1, D2, and D3), two selection lines (S0 and S1), and one output (Output). The block diagram of a 4x1 Multiplexer is seen in the figure below. Based on the combination of inputs present at these two selection lines, one of these four inputs would be connected to the output. The truth table of a 4x1 Multiplexer is depicted below.

The logic diagram of a 4x1 multiplexer is shown below, with the multiplexer decoding the input through the select line.

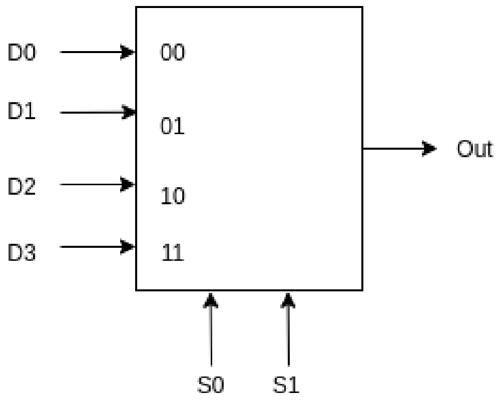

Figure 3.9: Logic diagram of 4X1 MUX

The truth table of a 4X1 multiplexer is shown below, in which four input combinations 00, 10, 01, and 11 on the select lines transfer the inputs D0, D2, D1, and D3 to the output, respectively. That is, if S0=0 and S1=0, the output at Output is D0; similarly, if S0=0 and S1=1, the output at Output is D1, and so on.

From the above truth table, we can write the output expressions as

Output = D0 if S1=0 and S0=0.Therefore,

$$Output = D0\bar{S}1\bar{S}0$$

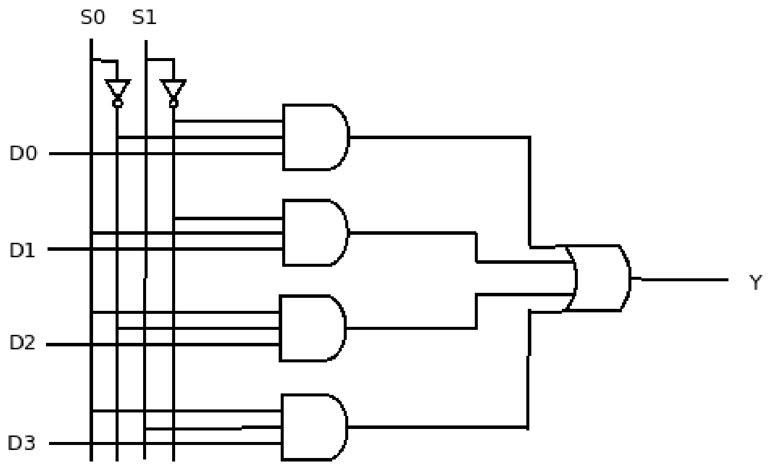

FIGURE 3.10: Circuit diagram of 4:1 MUX

S0	S1	D0	D1	D2	D3	Output
0	0	0	x	x	x	0
0	0	1	x	x	x	1
0	1	x	0	x	x	0
0	1	x	1	x	x	1
1	0	x	x	0	x	0
1	0	x	x	1	x	1
1	1	x	x	x	0	0
1	1	x	x	x	1	1

TABLE 3.8: 4X1 MUX Truth Table

Output = D1 if S1=0 and S0=1.Therefore,

$$Output = D1\bar{S}1S0$$

Output = D2 if S1=1 and S0=0.Therefore,

$$Output = D2S1\bar{S}0$$

Output = D3 if S1=0 and S0=0.Therefore,

$$Output = D3S1S0$$

These product terms must be applied together to achieve the cumulative information contribution from the multiplexer, yielding the following Boolean expression

$$Output = D0\bar{S}1\bar{S}0 + D1\bar{S}1S0 + D2S1\bar{S}0 + D3S1S0$$

By using simple logic gates on top of the output expression, a 4X1 multiplexer can be introduced. The logic circuit for a 4X1 MUX is shown below, and is made up of four 3-input AND gates, two 1-input NOT gates, and one 4-input OR gate.

In this circuit, every information input line is connected as an input to the AND gate, and 2 select lines are connected as alternative two inputs. The output of the AND gate is connected to the inputs of the OR gate, resulting in Output.

3.2.1 4:1 MUX using AND gate

The truth table for a 4X1 multiplexer using an AND gate is shown below, with four input combinations on the select lines 00, 10, 01, and 11 switching the inputs D0, D1, D2, and D3 to the output, respectively. That is, when S1=0 and S0=0, the output at Output is D0, similarly, if the select inputs S1=0 and S0=1 the output at Output is D1, and if the select inputs S1=1 and S0=0, the output at Output is D2, and if the select inputs S1=1 and S0=0, the output at Output is D3. These product terms must be applied together

A	B	Output = AND
0	0	D0 0
0	1	D1 0
1	0	D2 0
1	1	D3 1

Table 3.9: 4X1 MUX truth table with AND gate

to achieve the cumulative information contribution from the multiplexer, yielding the following Boolean expression

$$Output = D0\bar{S}1\bar{S}0 + D1\bar{S}1S0 + D2S1\bar{S}0 + D3S1S0$$

We can implement a 4X1 multiplexer using logic gates based on the output expression, but we can't do it with only the AND gate, we will need the NOT gate.

3.2.2 4:1 MUX using OR gate

The truth table for a 4X1 multiplexer using an OR gate is shown below, with 4 input combinations 00, 10, 01, and 11 on the select lines, respectively, switching the inputs D0, D1, D2, and D3 to the output. If S1=0 and S0=0, the output is D0, while the other three inputs represent the outputs as 1 if S1=0 and S0=1. Similarly, if S1=1 and S0=0, the

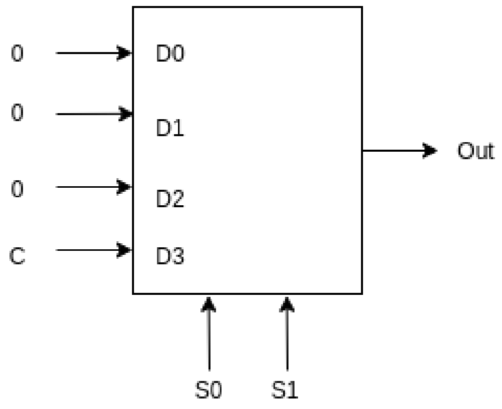

Figure 3.11: Logic diagram of 4:1 MUX using AND gate

A	B	Output = OR
0	0	D0 0
0	1	D1 1
1	0	D2 1
1	1	D3 1

Table 3.10: 4X1 MUX truth table with OR gate

output is D2, while if S1=1 and S0=1, the output is D3. These product terms must be applied together to achieve the cumulative information contribution from the multiplexer, yielding the following Boolean expression

$$Output = D0\bar{S}1\bar{S}0 + D1\bar{S}1S0 + D2S1\bar{S}0 + D3S1S0$$

We can implement a 4X1 multiplexer using logic gates depending on the output expression, but we can't do that with only the OR gate; we will need the NOT gate.

3.2.3 4:1 MUX using NOR gate

The truth table of a 4X1 multiplexer using a NOR gate is shown below, with four input combinations on the select points, 00, 10, 01, and 11, flipping D0, D1, D2, and D3 to the output, respectively. That is, when S1=0 and S0=0, the output at Output=1 is D0, and when S1=0 and S0=1, S1=1 and S0=0, and S1=1 and S0=1, the output at Output=0 is D1,D2,D3. The NOR gate is sometimes used as the opposite of the AND gate. These product terms must be applied together to achieve the cumulative information contribution

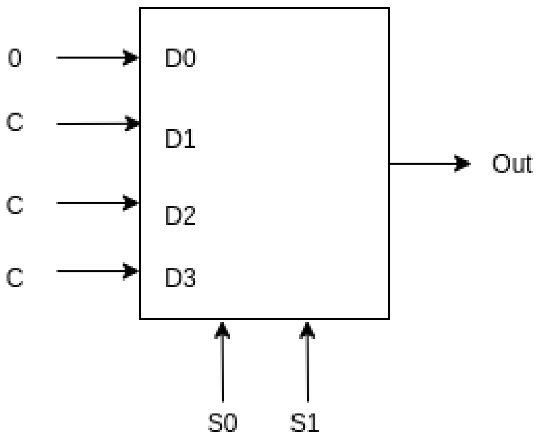

FIGURE 3.12: Logic diagram of 4:1 MUX using OR gate

A	B	Output = NOR
0	0	D0 1
0	1	D1 0
1	0	D2 0
1	1	D3 0

TABLE 3.11: 4X1 MUX truth table with NOR gate

from the multiplexer, yielding the following Boolean expression

$$Output = D0\bar{S}1\bar{S}0 + D1\bar{S}1S0 + D2S1\bar{S}0 + D3S1S0$$

A 4x1 multiplexer can be implemented using logic gates based on the output expression, but it cannot be implemented using just NOR gates.

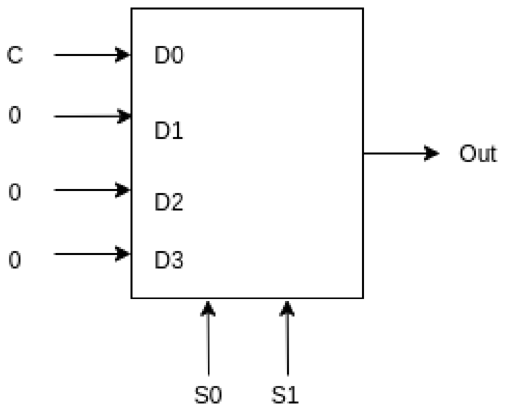

FIGURE 3.13: Logic diagram of 4:1 MUX using NOR gate

3.2.4 4:1 MUX using NAND gate

The truth logic of a 4X1 multiplexer using a NAND gate is depicted below, with four input combinations 00, 10, 01, and 11 on the select lines flipping the inputs D0, D1, D2, and D3 to the output. That is, if S1=0 and S0=0, the output at Output=1 is D0; similarly, if S1=0 and S0=1 and S1=1 and S0=0, the output at Output=0 is D3; and if S1=1 and S0=0, the output at Y=0 is D3; and if S1=1 and S0=0, the output at Y=0 is D3. It always shows the inverse of the AND gate.

A	B	Output = NAND
0	0	D0 1
0	1	D1 1
1	0	D2 1
1	1	D3 0

TABLE 3.12: 4X1 MUX truth table with NAND gate

These product terms must be applied together to achieve the cumulative information contribution from the multiplexer, yielding the following Boolean expression

$$Output = D0\bar{S}1\bar{S}0 + D1\bar{S}1S0 + D2S1\bar{S}0 + D3S1S0$$

A 4X1 multiplexer can be implemented with logic gates depending on the output expression, but it can also be implemented with only a NAND gate, i.e. a 3-input NAND gate.

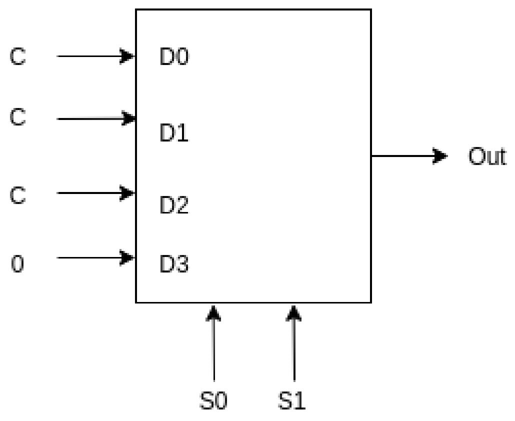

FIGURE 3.14: Logic diagram of 4:1 MUX using NAND gate

3.2.5 4:1 MUX using XOR gate

The truth table of a 4X1 multiplexer using an XOR gate is shown below, with four input variations on the select points, 00, 10, 01, and 11, flipping D0, D1, D2, and D3 to the output, respectively. That is, when S1=0 and S0=0, S1=1 and S0=1, the output at Output=0 is D0 and D3, and when S1=0 and S0=1, the output at Output=1 is D1 and D2.

A	B	Output = XOR
0	0	D0 0
0	1	D1 1
1	0	D2 1
1	1	D3 0

TABLE 3.13: 4X1 MUX truth table with XOR gate

These product terms must be applied together to achieve the cumulative information contribution from the multiplexer, yielding the following Boolean expression

$$Output = D0\bar{S}1\bar{S}0 + D1\bar{S}1S0 + D2S1\bar{S}0 + D3S1S0$$

A 4X1 multiplexer can be implemented using logic gates depending on the output expression, but not the XOR gate alone.

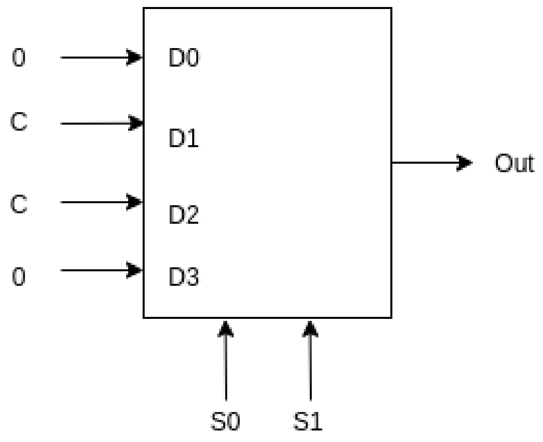

FIGURE 3.15: Logic diagram of 4:1 MUX using XOR gate

3.2.6 4:1 MUX using XNOR gate

The truth table of a 4X1 multiplexer using an XNOR gate is shown below, with four input combinations on the select points, 00, 10, 01, and 11, switching D0, D1, D2, and D3 to the output, respectively. That is, if S1=0 and S0=0, S1=1 and S0=1, the output at Output=1 is D0 and D3, and if S1=0 and S0=1, the output at Output=0 is D1 and D2. It's the XOR gate's opposite.

A	B	Output = XNOR
0	0	D0 1
0	1	D1 0
1	0	D2 0
1	1	D3 1

TABLE 3.14: 4X1 MUX truth table with XNOR gate

These product terms must be applied together to achieve the cumulative information contribution from the multiplexer, yielding the following Boolean expression

$$Y = D0\bar{S}1\bar{S}0 + D1\bar{S}1S0 + D2S1\bar{S}0 + D3S1S0$$

A 4X1 multiplexer can be implemented using logic gates depending on the output expression, but not the XNOR gate alone.

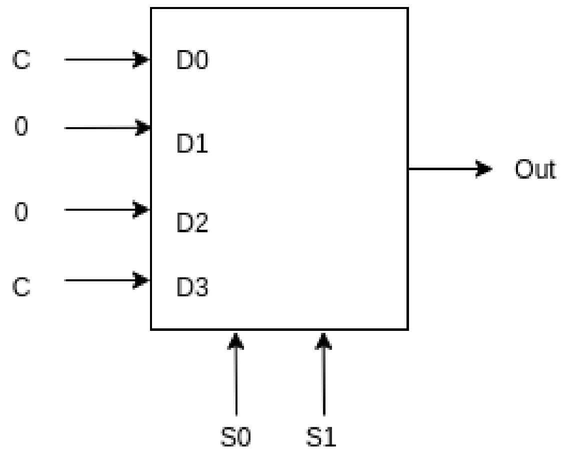

FIGURE 3.16: Logic diagram of 4:1 MUX using XNOR gate

3.2.7 4:1 MUX using NOT gate

The truth table of a 4X1 multiplexer with a NOT gate is shown below, with two input combinations of 0 and 1 and a dummy variable S1.

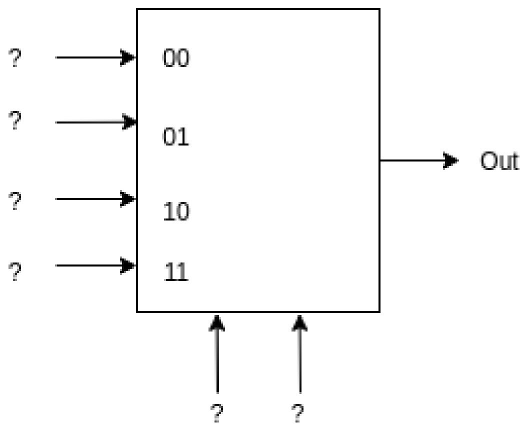

FIGURE 3.17: Logic diagram of 4:1 MUX using Dummy X

A	B	Output = NOT
0	0	D0 1
0	1	D1 1
1	0	D2 0
1	1	D3 0

TABLE 3.15: 4X1 MUX truth table with NOT gate

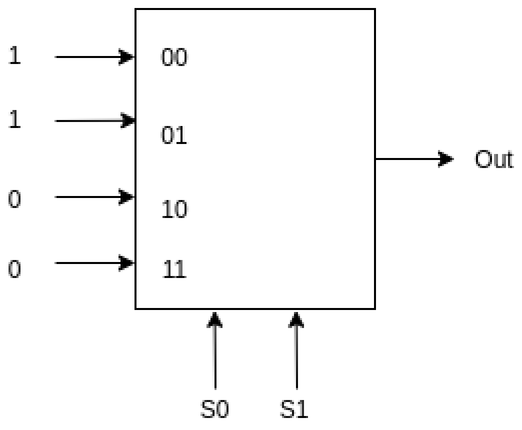

FIGURE 3.18: Logic diagram of 4:1 MUX using NOT gate

Now set S1 to either 0 or 1 (both will work) Given below is two different solutions

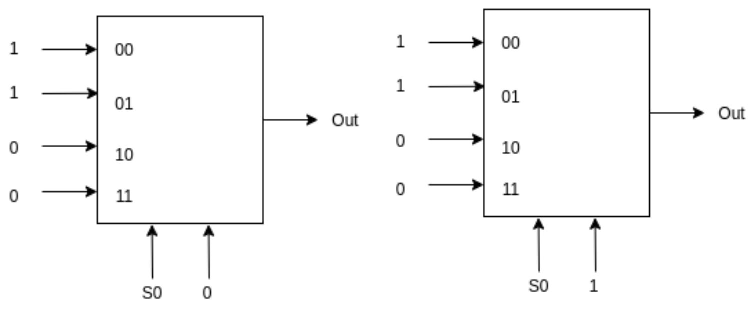

FIGURE 3.19: Logic diagram of 4:1 MUX using Alternative NOT gate

3.3 Proposed system

The register cell (R-cell) and the combinatorial cell are two types of logic modules available in the RTAX-S family (C-cell). More than 4,000 combinatorial functions of up to five inputs can be implemented by the RTAX-S C-cell (Figure 2.3). Carry logic is used in the C-cell for even more robust implementation of arithmetic functions. And for a large number of effective arithmetic functions. The C-cell structure is very synthesis-friendly due to its small size, which simplifies the overall design while also reducing design time.

The basic 4:1 MUX is Combinatiorial cell (C-Cell) using two different select lines S0 = (A0*B0) as AND gate and S1 = (A1 + B1) as OR gate. The RTAX-S C-cell we've implemented upto four inputs with different stages and Normalize the inputs to form single output.

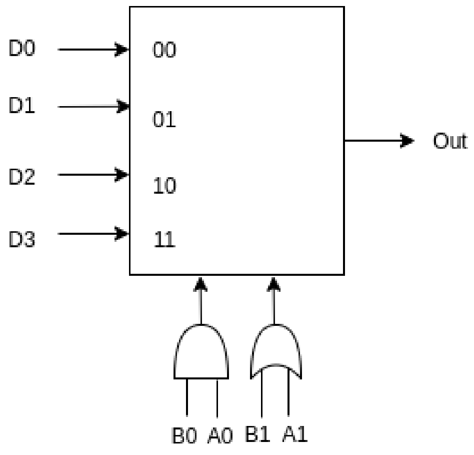

FIGURE 3.20: Logic diagram of C-Cell

As we compare total data output from the multiplexer of Boolean expression is given as

$$Output = D0\bar{S}1\bar{S}0 + D1\bar{S}1S0 + D2S1\bar{S}0 + D3S1S0......(1)$$

A 4X1 multiplexer can be implemented using logic gates, i.e. simplified using select lines S0 = (A1 + B1) as OR gate and S1 = (A0 * B0) as AND gate, as seen in the output expression above.

The generalized expression from the equation (1) is

$$Output = D0\overline{(A1 + B1)}.\overline{(A0.B0)} + D1\overline{(A1 + B1)}.(A0.B0) + D2(A1 + B1).\overline{(A0.B0)}$$

$$+D3(A1 + B1).(A0.B0)$$

A0 B0	A1 B1	A0*B0	A1+B1	Output
0 0	0 0	0	0	D0
0 0	0 1	0	1	D1
0 0	1 0	0	1	D1
0 0	1 1	0	1	D1
0 1	0 0	0	0	D0
0 1	0 1	0	1	D1
0 1	1 0	0	1	D1
0 1	1 1	0	1	D1
1 0	0 0	0	0	D0
1 0	0 1	0	1	D1
1 0	1 0	0	1	D1
1 0	1 1	0	1	D1
1 1	0 0	0	0	D2
1 1	0 1	0	1	D3
1 1	1 0	0	1	D3
1 1	1 1	0	1	D3

Table 3.16: Truth Table of C-cell Architecture

The generalized expression from the equation (1) is

$$Output = D0\overline{(A1 + B1)}.\overline{(A0.B0)} + D1\overline{(A1 + B1)}.(A0 * B0) + D2(A1 + B1).\overline{(A0 * B0)}$$

$$+D3(A1 + B1).(A0 * B0)$$

using De Morgan's rule

$$Output = D0(\bar{A}1 * \bar{B}1).(\bar{A}0 + \bar{B}0) + D1(\bar{A}1 * \bar{B}1).(A0 * B0) + D2(A1 + B1).(\bar{A}0 + \bar{B}0)$$

$$+D3(A1 + B1).(A0 * B0)$$

$$Output = (D0\bar{A}1 * D0\bar{B}1).(\bar{A}0 + \bar{B}0) + D1(\bar{A}1 * \bar{B}1).(A0 * B0) + (D2A1 + D2B1).(\bar{A}0 + \bar{B}0)$$

$$+(D3A1 + D3B1).(A0 * B0)$$

$$Output = D0\bar{A}0\bar{A}1\bar{B}0 + D0\bar{A}0\bar{B}0\bar{B}1 + D1\bar{A}1\bar{B}1A0B0 + D2A1\bar{A}0 + D2B1\bar{A}0$$

$$+D2A1\bar{B}0 + D2B1\bar{B}0 + D3A0A1B0 + D3A0B0B1$$

Analysis the generalized equation when A0 = 0

$$Output = D0\bar{A}1\bar{B}1 + D0\bar{A}1\bar{B}0\bar{B}1 + D2A1\bar{B}0 + D2B1 + D2A1\bar{B}0 + D2B1\bar{B}0$$

Analysis the generalized equation when A1 = 0

$$Output = D0\bar{A}0\bar{B}1 + D0\bar{B}0\bar{B}1 + D1A0B0\bar{B}1 + D2\bar{A}0B1 + D2\bar{B}0B1$$

Analysis the generalized equation when B0 = 0

$$Output = D0\bar{A}0\bar{A}1\bar{A}0 + D0\bar{A}1\bar{B}1 + D2A1\bar{A}0 + D2B1\bar{A}0 + D2A1 + D2B1$$

Analysis the generalized equation when B1 = 0

$$Output = D0\bar{A}0\bar{A}1\bar{B}1 + D0\bar{A}1\bar{B}0 + D1\bar{A}1A0B0 + D2A1\bar{A}0 + D2A1\bar{B}0 + D3A0A1B0$$

Analysis the generalized equation when A0 = 0 and A1 = 0

$$Output = D0\bar{B}1 + D0\bar{B}0\bar{B}1 + D2B1 + D2B1\bar{B}0$$

Analysis the generalized equation when A0 = 0 and A1 = 1

$$Output = D2 + D2\bar{B}0 + D2B1 + D2B1\bar{B}0 + D3B0$$

Analysis the generalized equation when A0 = 1 and A1 = 0

$$Output = B0\bar{B}1\bar{B}0 + D1\bar{B}1B0 + D2B1\bar{B}0 + D3B0B1$$

Analysis the generalized equation when A0 = 1 and A1 = 1

$$Output = D2\bar{B}0 + D2B1\bar{B}0 + D3B0 + D3B0B1$$

Analysis the generalized equation when B0 = 0 and B1 = 0

$$Output = D0\bar{A}1\bar{A}0 + D1\bar{A}1 + D2A1\bar{A}0 + D2A1$$

Analysis the generalized equation when B0 = 0 and B1 = 1

$$Output = D0A1\bar{A}0 + D2A1 + D2\bar{A}0 + D2$$

Analysis the generalized equation when B0 = 1 and B1 = 0

$$Output = D0\bar{A}1\bar{A}0 + D1\bar{A}1A0 + D2A1\bar{A}0 + D3A0A1$$

Analysis the generalized equation when B0 = 1 and B1 = 1

$$Output = D0A1\bar{A}0 + D2\bar{A}0 + D3A0A1 + D3A0$$

Analysis the generalized equation when A0 = 0, A1 = 0 and B0 = 0

$$Output = D0\bar{B}1 + D0B1 + D2B1$$

Analysis the generalized equation when A0 = 0, A1 = 0 and B0 = 1

$$Output = D0\bar{B}1 + D2B1$$

Analysis the generalized equation when A0 = 0, A1 = 1 and B0 = 0

$$Output = D0\bar{B}1 + D2 + D2B1$$

Analysis the generalized equation when A0 = 0, A1 = 1 and B0 = 1

$$Output = D2 + D2B1$$

Analysis the generalized equation when A0 = 1, A1 = 0 and B0 = 0

$$Output = D0\bar{B}1 + D2B1$$

Analysis the generalized equation when A0 = 1, A1 = 0 and B0 = 1

$$Output = D1\bar{B}1 + D2B1$$

Analysis the generalized equation when A0 = 1, A1 = 1 and B0 = 0

$$Output = D2 + D2B1$$

Analysis the generalized equation when A0 = 1, A1 = 1 and B0 = 1

$$Output = D3B1$$

Analysis the generalized equation when A0 = 0, A1 = 0 and B1 = 0

$$Output = D0 + D0\bar{B}0$$

Analysis the generalized equation when A0 = 0, A1 = 0 and B1 = 1

$$Output = D2 + D2\bar{B}0$$

Analysis the generalized equation when A0 = 0, A1 = 1 and B1 = 0

$$Output = D2 + D2\bar{B}0$$

Analysis the generalized equation when A0 = 0, A1 = 1 and B1 = 1

$$Output = D2 + D2\bar{B}0$$

Analysis the generalized equation when A0 = 1, A1 = 0 and B1 = 0

$$Output = D1B0$$

Analysis the generalized equation when A0 = 1, A1 = 0 and B1 = 1

$$Output = D3B0 + D2\bar{B}0$$

Analysis the generalized equation when A0 = 1, A1 = 1 and B1 = 0

$$Output = D2\bar{B}0 + D3B0$$

Analysis the generalized equation when A0 = 1, A1 = 1 and B1 = 1

$$Output = D2\bar{B}0 + D3B0$$

3.4 Design Of C-Cell using Half Adder

Half Adder is basic combinational circuit with 2 inputs and 2 outputs.

- The half adder circuit is designed to 4:1 mux using two different select lines with a 2 inputs for each.
- 2 select lines are AND(A0, B0) and OR(A1, B1) to form a C-cell.This circuit has two outputs sum and carry.
- D0 to D3 are the required inputs.
- Sum = (A0 * B0)'(A1 + B1) + (A0 * B0)(A1 + B1)'
- Carry = (A0 * B0)(A1 + B1)

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

TABLE 3.17: Truth Table of 4:1 MUX using Half Adder

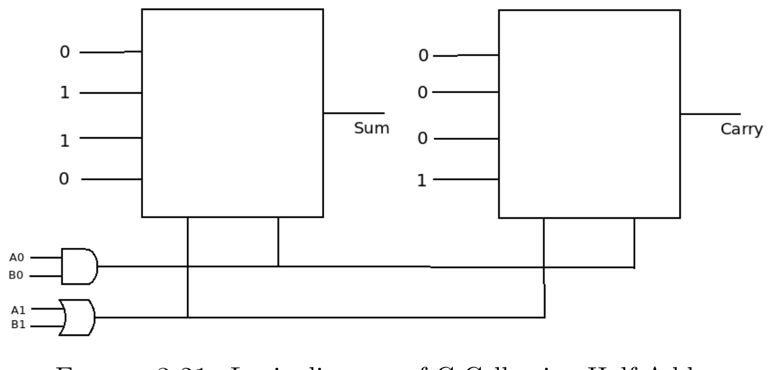

FIGURE 3.21: Logic diagram of C-Cell using Half Adder

Finally the sum and carry can be taken out as output.

3.5 Design Of C-Cell using Full Adder

To implement full adder,first it is required to know the expression for sum and carry.

- The full adder circuit is designed to 4:1 mux using two different select lines with a 2 inputs for each.
- 2 select lines are AND(A0, B0) OR(A1, B1) to form a C-cell. This circuit has two outputs sum and carry.
- D0 to D3 are the required inputs.
- From the above calculation B and C are taken as select lines(taken from the above truth table of Full adder).
- And the calculation is done on the A input.
- Sum = (A0 * B0)'(A1 + B1) + (A0 * B0)(A1 + B1)'
- Carry = (A0 * B0)(A1 + B1)

A	B	C	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 3.18: Truth Table of 4:1 MUX using Full Adder

Now it's needed to place the expression of add and carry within a MUX Tree. For mux tree calculation let's think about the subsequent parameters for MUX.

For Sum the SOP form has been rounded off with circles which are(1,2,4,7) and correspondingly either A or is selected depending on the rounding of the number at which it comes.If any certain pair doesn't match any 0 will appear but in sum expression there is none but in carry expression there is one zero term.Similarly on the same approach,the carry can also be calculated.

Now putting all these in the circuit it looks like:

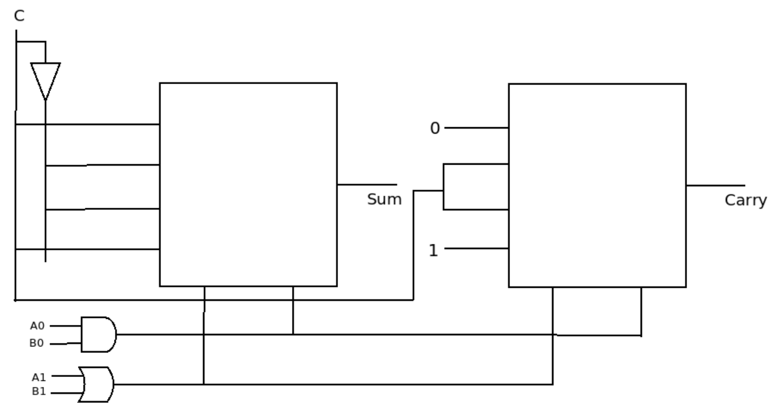

FIGURE 3.22: Logic diagram of C-Cell using Full Adder

3.6 Design Of C-Cell using 4 Bit Adder

A Full adder Its arithmetic number of three input bits is formed by a C-cell, which is a combinational circuit. It contains two select lines, S0 = AND(A0,B0) and S1 = OR(A1,B1), which represent the two significant bits to be inserted, as well as C input, which is a carry-in from the previous significant location.

The 4-bit binary adder can be implemented in one of two ways.

- Use half adder for doing the addition of 2 Least vital bits and three Full adders for doing the addition of 3 higher vital bits.
- The 4-bit C-cell adder performs the addition of two 4-bit numbers.
- For consistency, use four full adders. The Full adder, which is used to add the least significant bits, becomes a Half adder since the original carry Cin empty.
- It has two outputs: S, which is the number of the two input bits, which can be D0-D3, and Cout, which carries the value if the output from example we have inserted S value is 12 , 13 since the binary forms of this need two digits for representation, output is always given 0, and output 25, which represents in binary form.

We found the second choice for the time being. The following figure depicts the block diagram of a 4-bit binary adder.

The four full adders are rippled here. Each Full adder receives the bits from two parallel inputs, S0 and S1. The carry input of the resulting higher order Full adder is the carry

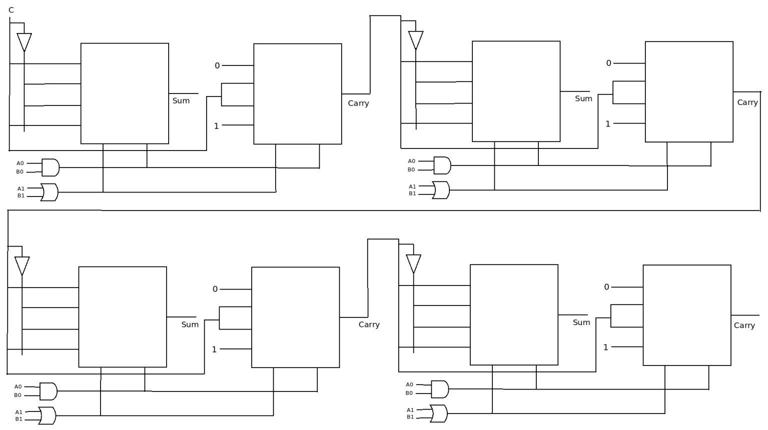

Figure 3.23: Logic diagram of C-Cell using 4 bit Adder

output of one Full adder. The resulting add of this 4-bit binary adder has a maximum of 5 bits. As a result, the MSB will carry out the last stage Full adder.

By rippling the required set of Full adders in this form, we can create any higher order binary adder. Since the carry propagates (ripples) from one point to the next, this binary adder is often referred to as a ripple carry (binary) adder.

3.7 Design Of C-Cell using 8 Bit Adder

A Full Adder C-cell can be an arithmetic combined circuit with 3 input bits in total The two selected lines S0 = AND(A0,B0) and S1 = OR(A1,B1) , Cin input is a carry-in from the previous significant location which represents the two significant bits to be inserted.

- The 8bit adder removes the 8bit binary inputs, as well as the output result. I could use eight 1-bit adders and bind them to create a Full 8-bit adder.
- It has two outputs: S, which is the number of the two input bits, which can be D0-D3, and Cout, which carries the value if the output from example we have inserted S value is 192 , 255 since the binary forms of this need two digits for representation, output is always given 0, and output 477, which represents in binary form.
- We combine 8 of those Full adders, which we've already shown function, to create an 8-bit adder.

From one full adder S which is sum of the single C-cell and Cout which is carry to next full adder's S and same procedure is followed till 8bits and every time the sum as stored

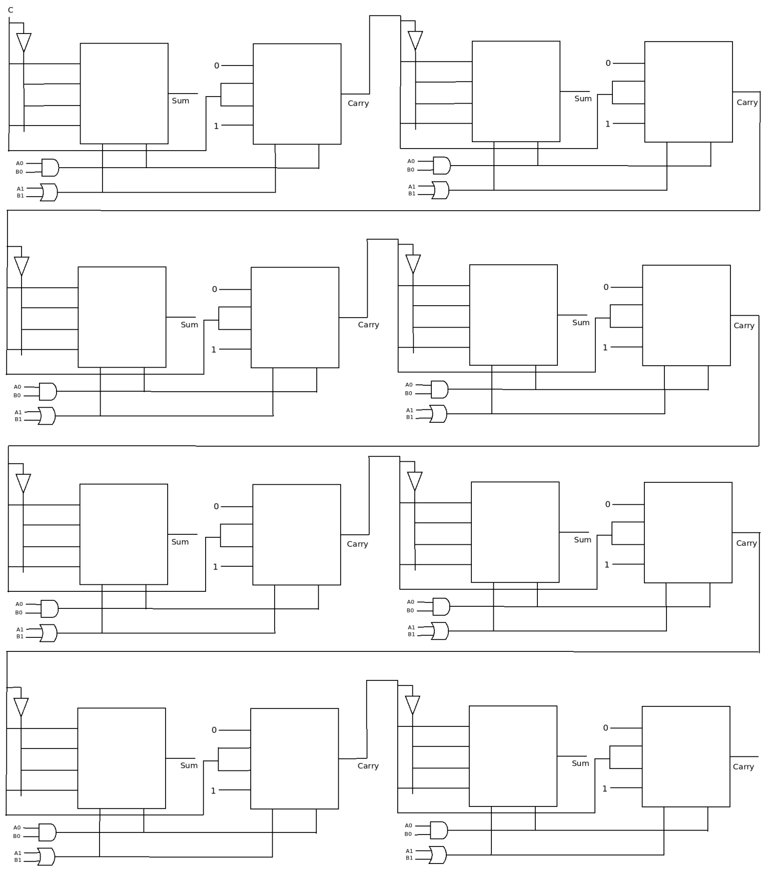

Figure 3.24: Logic diagram of C-Cell using 8 bit Adder

as output of the above logic design.The 8 full adders are rippled below. Each Full adder receives the bits from two parallel inputs, S0 and S1. The carry input of a later higher order Full adder is the carry output of one Full adder. The product of this 8bit binary adder is a sum of at most 8 bits. The resulting number of this 8-bit binary adder has a maximum of 8 bits. As a result, the Most Significant Bits performs the last stage Full adder.

The 8bit Adder is made up of 8 Full Adders connected by their bring in and out. The Full Adder adds two 1-bit inputs together. Similarly, the 8bit Adder accepts two 8-bit inputs in total. Four inputs, two selectors, and one output make up the 4x1 c-cell multiplexer.

3.8 Design Of C-Cell using Array Multiplier

An array multiplier factor is a digital combinational circuit used for multiplying 2 binary numbers by using an array of full adders and 0.5 adders. This array is employed for the nearly coinciding addition of the varied product terms concerned.

The Array Multiplier has a standard architecture framework that is based on the add shift algorithm theorem.

Partial product = Multiplier * Multiplicand..... (2)

The combination of Full Adders and half Adders where AND gates are used for the component is completed wherever the partial product is shifted by bit order. The n*n multiplier calculates the partial product with the n*n AND gates and thus the insertion of partial products is mostly done by using n* (n – 2) full adders and n half adders.The select lines of the Adders are taken has S0 = (A0.B0) as AND gate and S1 = (A1 + B1) as OR gate with Four information inputs lines of the C-Cell.

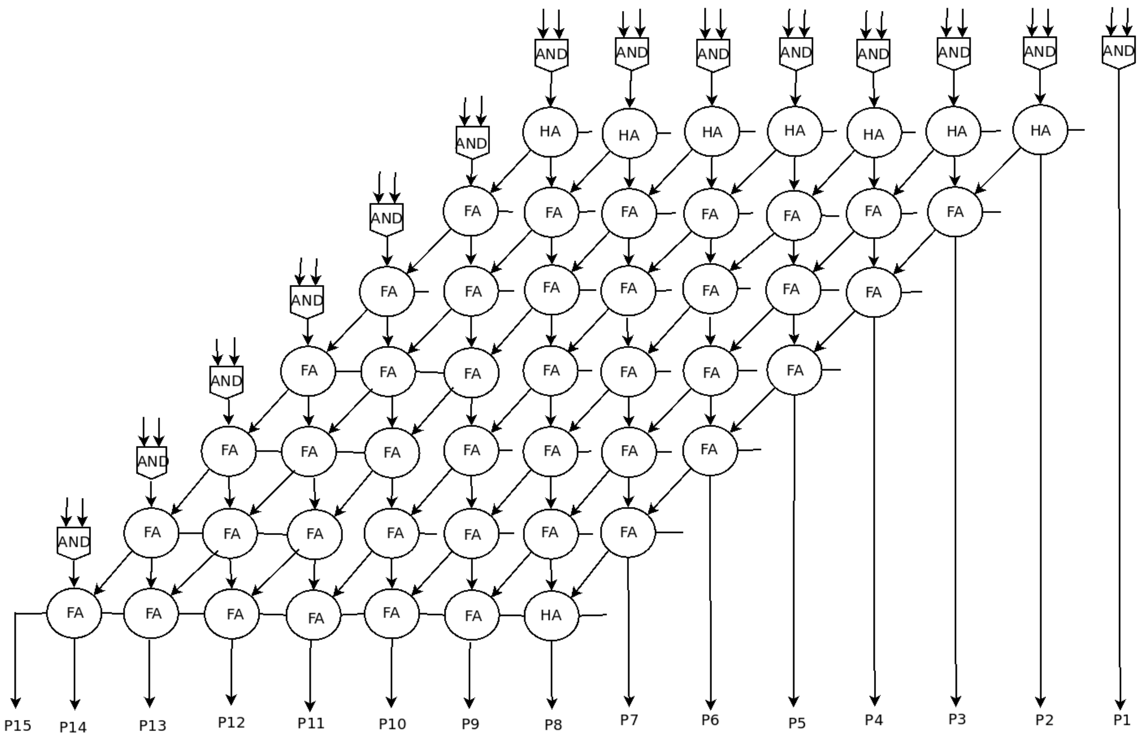

Figure 3.25: Array Multiplier

3.9 Design Of C-Cell using 4 Array Multiplier

The C-cell 4X4 multiplier has eight inputs and eight outputs. Wherever we have a complete adder as a simple building block of an array multiplier, it has 3 input lines and 2 output lines. The following is an example of a C-cell 4X4 array multiplier. The left bit is the partial product's Least significant bit. The Most Significant bit of partial product is the correct bit. On multiplication, the partial products are now shifted to the left facet, and they're added to produce the final product. This procedure is repeated before the no. 2 partial product is ready for inclusion.

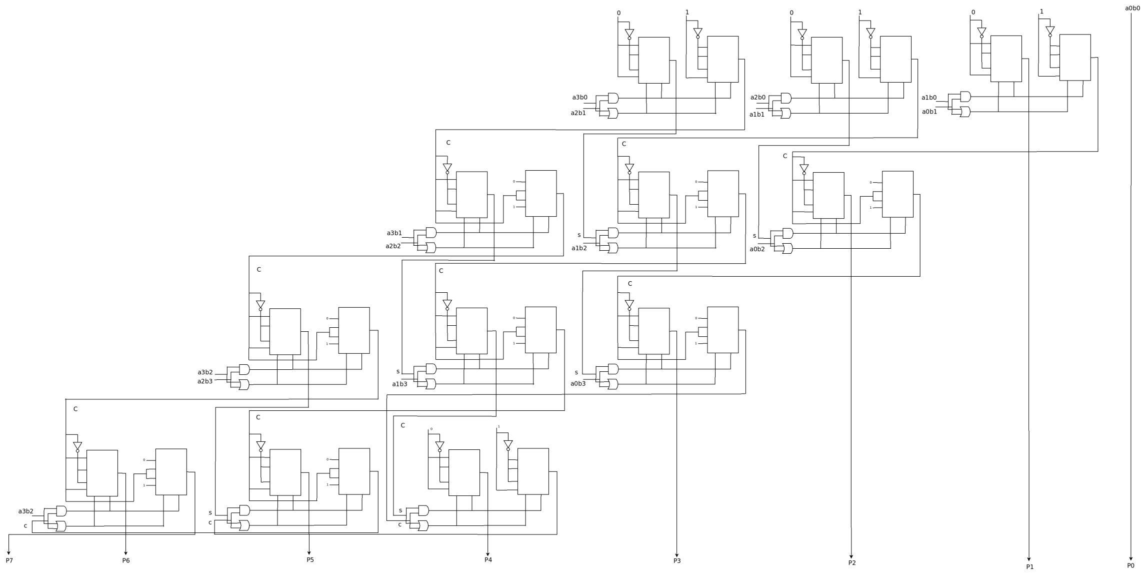

FIGURE 3.26: Logic diagram of 4X4 Array Multiplier using C-Cell

Where the multiplicand and multiplier are A0,A1,A2,A3 and B0,B1,B2,B3 the sum of all products is partial product. A product is the consequence of the partial product's addition.

The Total number of 4 Half Adder C-cells, and 8 Full Adder C-cells are needed to build an 4x4 Array Multiplier. Here are a total number of 12 C-cell Adders are required.

3.10 Design Of C-Cell using 8 Array Multiplier

The 8×8 array multiplier of C-cell shown has 16 inputs and 16 outputs. Wherever we have a complete adder as a simple building block of an array multiplier, it has three input lines and two output lines. The following is an example of a 8x8 c-cell array multiplier. The LSB little bit of partial product is on the left. The MSB little bit of partial substance is the right bit. On multiplication, the partial products are now moved to the left side, and they are pushing for the final product. This process is repeated until there are no further partial items to add.

Where the multiplicand and multiplier are A0,A1,A2,A3,A4,A5,A6,A7 and B0,B1,B2,B3,B4,B5,B6,B7, the sum of all products is partial product. A product is the consequence of the partial product's addition.

The Total number of 8 Half Adder C-cells, and 48 Full Adder C-cells are needed to build an 8x8 Array Multiplier. Here are a total number of 56 C-cell Adders are required.

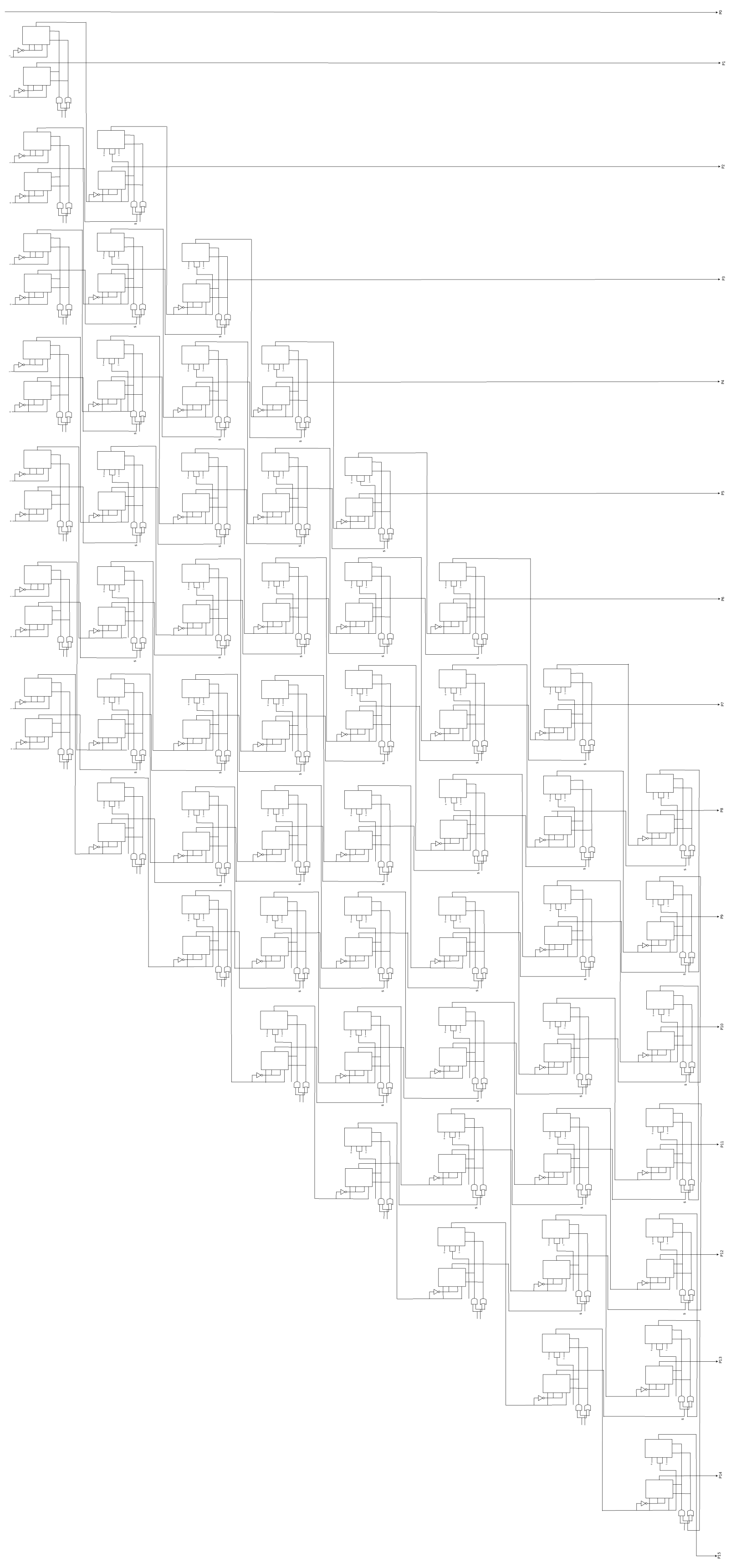

FIGURE 3.27: Logic diagram of 8X8 Array Multiplier using C-Cell

Chapter 4

Results and Conclusion

4.1 Results

4.1.1 Error Analysis in C-Cell Adder and Multiplier

- In C-Cell ,the single event upset can occur at select lines of multiplexer.
- The error analysis is done by injecting the error at select lines of multiplexer.
- Error injection is applied on select lines of adder and multiplier at different bit positions with respect the n-bit adder and n-bit multiplier.
- Generating the Error Percentage from the original Result and the Error Result.
- Both original result and error results are produced from user inputs A and B.
- With the analysis ,one can determine the highest error percentage can occur at MSB's select lines of C-Cell adder and multiplier.
- This experimental analysis is done in python simulation and generated the graphs as shown below.

Original value	Error value	Difference Original and Error	Error Percentage
25	26	-1	-4
25	27	-2	-8
25	21	4	16
25	17	8	32

TABLE 4.1: Table for 4bit Error Percentage for certain example

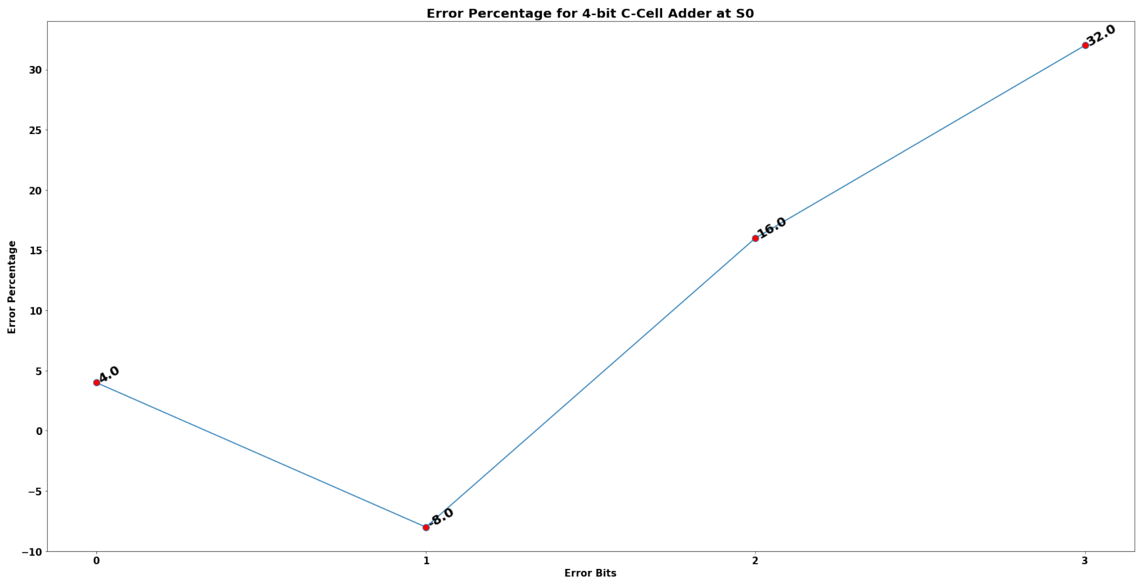

FIGURE 4.1: 4bit C-Cell adder with Error injection at select line

Original value	Error value	Difference Original and Error	Error Percentage
437	436	1	0.228833
437	435	2	0.457666
437	433	4	0.915332
437	429	8	1.830664
437	421	16	3.661327
437	405	32	7.322654
437	373	64	14.645309
437	309	128	29.290618

TABLE 4.2: Table for 8bit Error Percentage for certain example

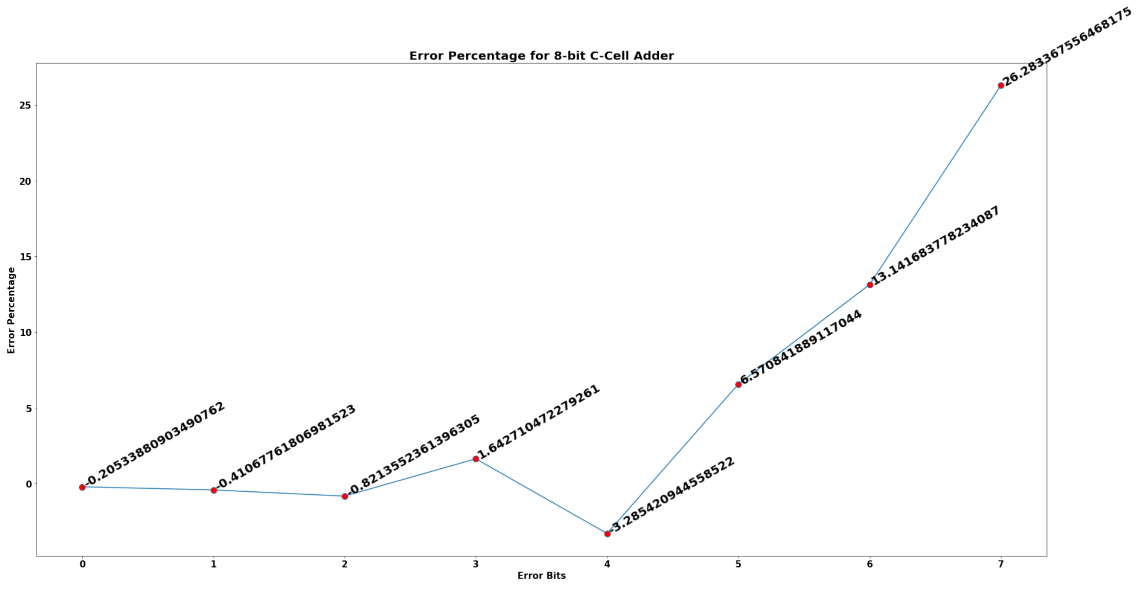

FIGURE 4.2: 8bit C-Cell adder with Error injection at select line

Original value	Error value	Difference Original and Error	Error Percentage
156	158	-2	-1.282051
156	162	-6	-3.846154
156	162	-6	-3.846154
156	170	-14	-8.974359
156	186	-30	-19.230769
156	162	-6	-3.846154
156	186	-30	-19.230769
156	186	-30	-19.230769
156	138	18	11.538462
156	106	50	32.051282

TABLE 4.3: Table for 4X4 Error Percentage for certain example

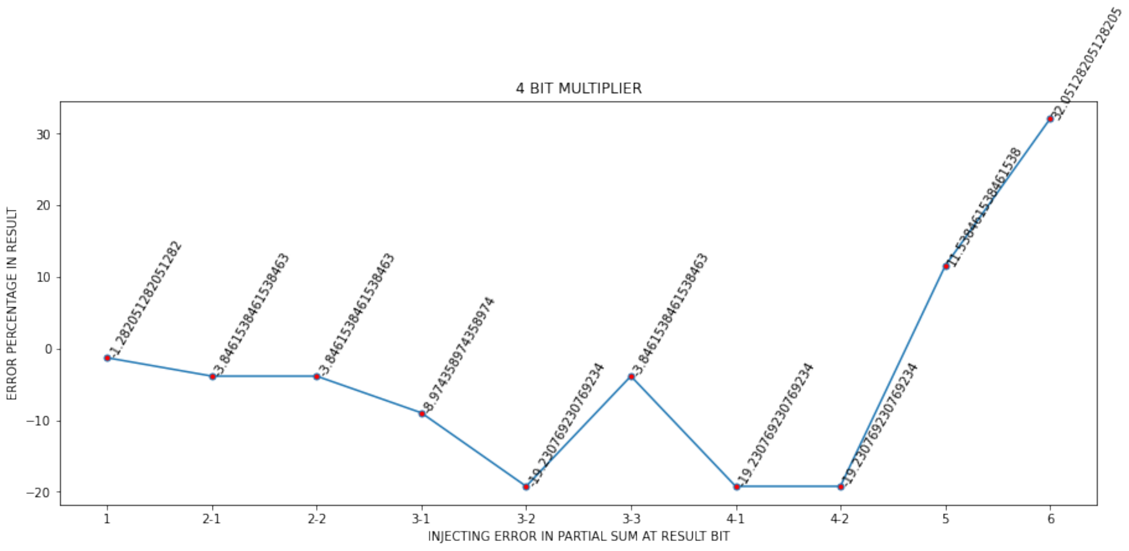

FIGURE 4.3: Graph analysis of 4x4 array multiplier at Select line

4.2 Conclusion

In this project, designed the C-Cell adder and multiplier which are used in FPGA. Error analysis is performed at a different positions in the C-Cell adder and multiplier by injecting the error at select lines. The error percentage is calculated concerning the original result and the error result. The error analysis is focused on the 4-bit to 128-adder and on the 4-bit,8-bit and 16-bit multiplier.

Error analysis helps to avoid the single bit error by applying the fault-tolerant techniques and also reduce the hardware space head-over by applying the fault-tolerant techniques such TMR, DMR, TIR and CDMR etc., on the MSB's Select lines of C-Cell adder and Multiplier. It also improves the hardware performance for generating the result.

Original value	Error value	Difference Original and Error	Error Percentage
46410	46410	0	0.000000
46410	46412	-2	-0.004309
46410	46408	2	0.004309
46410	46408	2	0.004309
46410	46400	10	0.021547
46410	46400	10	0.021547
46410	46400	10	0.021547
46410	46416	-6	-0.012928
46410	46416	-6	-0.012928
46410	46416	-6	-0.012928
46410	46448	-38	-0.081879
46410	46416	-6	-0.012928
46410	46416	-6	-0.012928
46410	46416	-6	-0.012928
46410	46416	-6	-0.012928
46410	46416	-6	-0.012928
46410	46352	58	0.124973
46410	46352	58	0.124973
46410	46352	58	0.124973
46410	46352	58	0.124973
46410	46352	58	0.124973
46410	46352	58	0.124973
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	45456	954	2.055591
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830
46410	46480	-70	-0.150830

Table 4.4: Table for 8X8 Error Percentage for certain example

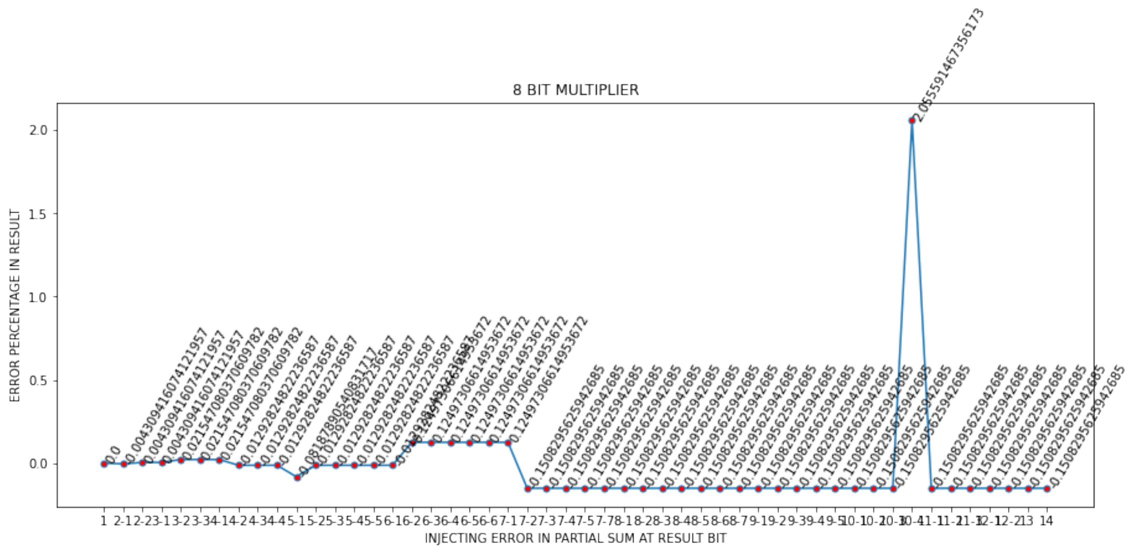

Figure 4.4: Graph analysis of 8x8 array multiplier at Select line

4.3 Future Work

- The C-cell connection with register(Flip-Flops) will be analyzed.
- Sequential circuit behaviour is estimated using c-cell.
- Errors in Sequential circuit will be estimated in the c-cell.

Bibliography

[1] J. Kamatam and K. Gajula, “Design of array multiplier using mux based full adder,” *International Journal of Engineering Research Technology*, vol. 6, August 2017.

[2] N. M. Mahyuddin and G. Russell, “Single-event-upset sensitivity analysis on low-swing drivers,” *The Scientific World Journal*, vol. 2014, march 2014.

[3] P. Subramanyan, V. Singh, K. K. Saluja, and E. Larsson, “Multiplexed redundant execution: A technique for efficient fault tolerance in chip multiprocessors,” April 2010.

[4] L.Rajaa, B.M.Prabhub, and K.Thanushkodic, “Design of low power digital multiplier using dual threshold voltage adder module,” *International Conference on Communication Technology and System Design*, vol. 30, pp. 1179–1186, 2012.

[5] A. C. Persya and T. Nair, “Fault tolerant real time systems,” *International Conference on Managing Next Generation Software Application*, 2008.

[6] G. H. B. Talib, A. H. El-Maleh, and S. M. Sait, “Design of fault tolerant adders: A review,” *Arabian Journal for Science and Engineering*, vol. 43, pp. 6667–6692, 2018.

[7] S. Peng and R. Manohar, “Fault tolerant asynchronous adder through dynamic self-reconfiguration.”

[8] S. gupta, A. Jasuja, and R. shandilya, “Real-time fault tolerant full adder using fault localization,” *2018 IEEE International Students’ Conference on Electrical, Electronics and Computer Science (SCEECS)*, no. 18274829, 2018.

[9] S. K. Mitra and A. R. Chowdhury, “Minimum cost fault tolerant adder circuits in reversible logic synthesis,” *2012 25th International Conference on VLSI Design*, no. 12616219, 2018.

[10] D. H. K. Hoe, L. P. D. Bollepalli, and C. D. Martinez, “Fpga fault tolerant arithmetic logic: A case study using parallel-prefix adders,” Master’s thesis, 2013.

[11] S. Chakraborty, "Fault-tolerant, real-time reconfigurable prefix adder," University of Virginia Technical Report, University of Virginia, 2009.

[12] L. Phani and D. Bollepalli, "Design and implementation of fault tolerant adders on field programmable gate arrays," Electrical Engineering Theses, University of Texas, Spring 4-27-2012.

[13] V. Piuri, M. Berzieri, A. Bisaschi, and A. Fabi, "Residue arithmetic for a fault-tolerant multiplier: The choice of the best triple of bases," *Microprocessing and Microprogramming*, vol. 20, pp. 15–23, April 1987.

[14] P.-Y. Yin, Y.-H. Chen, C.-W. Lu, S.-S. Shyu, C.-L. Lee, T.-C. Ou, and Y.-S. Lin, "A multi-stage fault-tolerant multiplier with triple module redundancy (tmr) technique," Master's thesis, April 2013.

[15] P. Palsodkar, P. Palsodkar, and R. Giri, "Multiple error self checking-repairing fault tolerant adder-multiplier," *2018 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, 2019.

[16] N. Ahmad, A. H. Mokhtar, N. binti Othman, C. F. Soon, and A. A. H. A. Rahman, "Vlsi implementation of fault tolerance multiplier based on reversible logic gate," *International Research and Innovation Summit (IRIS2017)*, vol. 226.

[17] A. Sahu and A. K. Sahu, "High speed fault tolerant reversible vedic multiplier," *International Journal of Innovative Research in Advanced Engineering*, vol. 2.

[18] M. E. S. Chowdhury, N. Ahmed, and L. Jamal, "A new perspective in designing an optimized fault tolerant reversible multiplier," *2019 Joint 8th International Conference on Informatics, Electronics Vision (ICIEV) and 2019 3rd International Conference on Imaging, Vision Pattern Recognition (icIVPR)*, vol. 1.

[19] D. J. Sorin, "Fault tolerant computer architecture synthesis lectures on computer architecture."