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.

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 would like to express my sincere gratitude to my project guide **Dr.Noor Mohammad** for his guidance and encouragement through out the project. Also first and foremost, I am very thankful to **Raghavendra kumar** assistance and help in monitoring and finishing the project. Every short term goals and long term goals for this project were accomplished due to their constructive criticism and I am really thankful to them.

I would also like to thank my fellow lab mates who were very supportive and helped me whenever the need be in this due process.I once again thank everyone who helped me during the Final year Project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Field programmable gate arrays (FPGAs) are growing in popularity with designers as their ability to be reconfigured from a desktop computer allows ease of prototyping, rapid time to market, and minimal nonrecurring engineering (NRE) cost compared to custom integrated circuit (IC) designs.There are several reasons for this trend, including lower FPGA costs, increasing FPGA performance, and perhaps most important, a driving necessity for the fast time-to-market enabled by FPGA technologies. FPGA-based control systems can enforce critical interlock logic and also be designed to prevent I/O forcing by an operator.FPGA-based systems can literally rewire their internal circuitry which allows reconfiguration after the control system is deployed to the field. Faults are very common in VLSI systems such as FPGA. During FPGA configuration, the internal circuitry is connected in a way that creates a hardware implementation of the software application. Different fault-tolerance techniques are being developed to increase the reliability and dependability of applications on FPGAs. The goals of fault-tolerance techniques are to minimize the hardware, timing, and power overhead, and maximize the reliability of the system.

1.1 Motivation

Fault tolerant re-configurable computing is need of the hour for the military and space applications. These hardware architectures suffer from single-event-upset (SEU). SEU in the re-configurable hardware fabrics can lead to system failure. The objective of this work is to design the fault tolerant re-configurable architecture. This work focuses on design and development 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 provides an overview of fault tolerant techniques that are relevant for our discussion on FPGA designs. We begin by reviewing the structure of the FPGA and note the use of circuits designed to accelerate arithmetic operations, Fault tolerance with a 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. Some important fault tolerant techniques that are relevant for our discussion are also over-viewed in this section.

1.3 Problem Statement

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

- The objective of this work is do 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 Organization of the Report

The report has been divided into 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 which 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 requirement and working with problem statement. The Fourth chapter is about the programming and results I got with the work done. The final chapter deals with the conclusion and future work that can be done in this area.

Chapter 2

literature Survey

2.1 Device Architecture

AX Architecture, derived from the extremely undefeated SX-A sea-of-modules design, has been designed for top performance and total logic module utilization (Figure two.1). not like in ancient FPGAs, the complete floor of the Axcelerator device is roofed with a grid of logic modules, with just about no chip space lost to interconnect components or routing

2.2 Programmable Interconnect Element

The Axcelerator family uses a proprietary metal-to-metal anti-fuse programmable interconnect component that resides between the higher 2 layers of metal. This fully eliminates the channels of routing and interconnect resources between logic modules (as enforced on ancient FPGAs) and permits the economical sea-of-modules design. The antifuses ar commonly open circuits and, when programmed, kind a permanent, passive, low electrical resistance affiliation, resulting in the quickest signal propagation within the business. additionally, the very little size of those interconnect components provides the Axcelerator family torrential routing resources.

The terribly nature of Actel's non-volatile antifuse technology provides wonderful protection against style pirating and biological research (FuseLock technology). Cloning is impossible (even if the protection fuse is left unprogrammed) as no bitstream or programming file is ever downloaded or keep within the device. Reverse engineering is

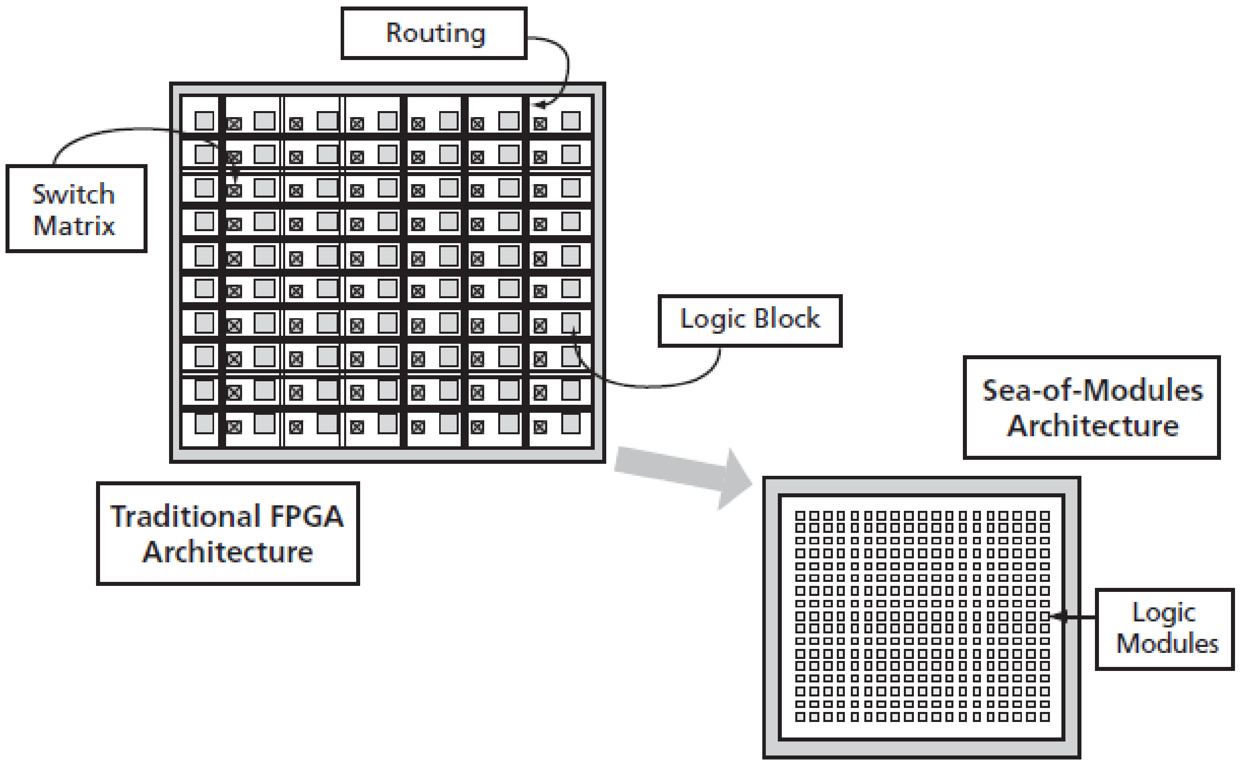

FIGURE 2.1: Sea of Modules Comparison

just about not possible thanks to the problem of making an attempt to differentiate between programmed and unprogrammed antifuses and conjointly thanks to the programming methodology of antifuse devices.

2.3 Logic Modules

Actel's Axcelerator family provides 2 styles of logic modules: the register cell (R-cell) and also the combinatorial cell (C-cell). This C-cell will implement quite 4,000 combinatorial functions of up to 5 inputs.

- The R-cell contains a flip-flop that includes asynchronous clear, asynchronous predetermined, and active-low change management signals (Figure 2-3).
- The R-cell registers feature programmable clock polarity selectable on a register-by-register basis.
- This provides further flexibility (e.g., simple mapping of dual-data-rate functions into the FPGA) whereas protective valuable clock resources.

- The clock supply for the R-cell is chosen from the hardwired clocks, routed clocks, or internal logic.

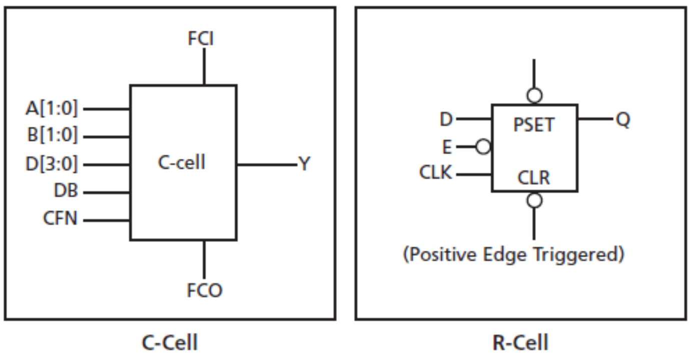

FIGURE 2.2: AX C-cell and R-cell

- Two C-cells, one R-cell, and 2 Transmit (TX) and 2 Receive (RX) routing buffers kind a Cluster, whereas 2 Clusters comprise a SuperCluster (Figure 1.4).
- Every SuperCluster conjointly contains AN freelance Buffer (B) module, that supports buffer insertion on high-fanout nets by the place-and-route tool, minimizing system delays whereas rising logic utilization.
- The logic modules at intervals the SuperCluster are organized in order that 2 combinatorial modules are side-by-side, giving a C–C–R – C–C–R pattern to the SuperCluster.
- This C–C–R pattern permits economical implementation (minimum delay) of two-bit carry logic for improved arithmetic performance.

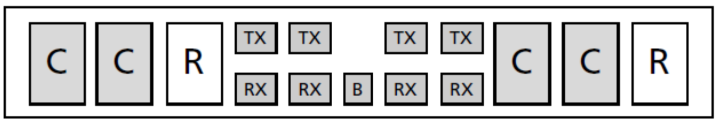

FIGURE 2.3: Ax SuperCluster

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 Module Specifications of C-Cell

The C-cell is one in every of the 2 logic module types within the AX architecture. it's the combinatorial logic resource within the Axcelerator device. The AX architecture implements a replacement combinatorial cell that's an extension of the C-cell implemented within the SX-A family. the most enhancement of the new C-cell is that the addition of carry-chain logic.

The C-cell are often employed in a carry-chain mode to construct arithmetic functions. If carry-chain logic isn't required, it are often disabled.The C-cell features the subsequent (Figure 2.4):

- Eight-input MUX (data: D0-D3, select: A0, A1, B0, B1). User signals may be routed to any of these inputs. Any of the C-cell inputs (D0-D3, A0, A1, B0,B1) may be tied to 1 of the four routed clocks (CLKE/F/G/H).
- Inverter (DB input) can be used to drive a complement signal of any of the inputs to the C-cell.
- A carry input and a carry output. The carry input of the C-cell is that the carry output from the Ccell on to the north.
- Carry connect for carry-chain logic with a signal propagation time of less than 0.1 ns.
- A hardwired connection (direct connect) to the adjacent R-cell (Register Cell) for all C-cells on the side of a SuperCluster with a signal propagation time of but 0.1 ns.

This layout of the C-cell (and the C-cell Cluster) enables the implementation of over 4,000 functions of up to 5 bits. for eg: 2 C-cells may be used together to implement a four-input XOR function in a very single cell delay.

The carry-chain configuration is handled automatically for the user with Actel's extensive macro library (please see Actel's Antifuse Macro Library Guide for a whole listing of obtainable Axcelerator macros).

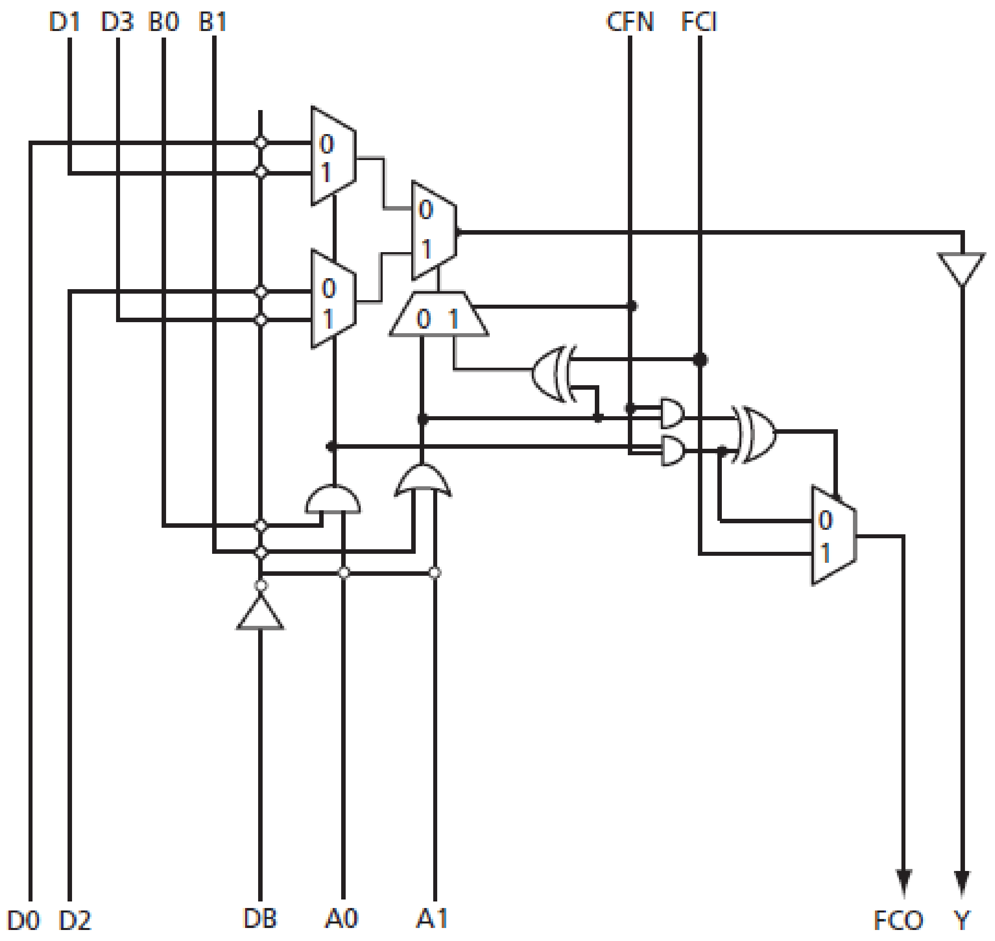

FIGURE 2.4: C-cell

2.5 Multiplexer Logic as Function Generators

- An FPGA contains an outsized number of logic cells. Each logic cell can be configured to implement a particular set of functions.
- FPGA (Field Programmable Gate Array) is an computer circuit containing a matrix of user-programmable logic cells, having the ability to implement complex digital circuitry.
- These gates are designed to implement user-defined combinational and sequential functions.
- FPGA families from different vendors use different logic cells architecture.
- Each logic cell incorporates a fixed number of inputs and outputs.

- Logic cells employed in FPGAs are
  - Multiplexer-based logic cells (e.g. Actel FPGAs).
  - Memory-based logic cells (e.g. Xilinx FPGAs)

2.6 Fault analysis of C-Cell

- A multiplexer-based logic module is typically composed of a tree of 2-to-1 MUXes using different primitive gates like NOT, AND, OR, NOR, NAND, XOR, XNOR gates.
- A multiplexer-based logic module is typically composed of a tree of 4-to-1 MUXes 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.
- There are few required steps in order to measure the for SEU(Single Event Upset) tolerance analysis. These steps are generally used for fault injection experiment, which cover three processes
  1. Fault Target Location
  2. Fault Injection
  3. Observation of Fault Consequences

Fault Target Location The typical targets of model-based fault injection techniques are located at Input Data and Select lines in the C-Cell.

Fault Injection This involves agitating the incidence of a fault within the circuit by re-configuring the inner resources. during this case, I actually have injected error at choose lines of the c-cell with 1 to 4 that significance's that alter the method of the parameters of the C-cell.

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

Work Done

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.

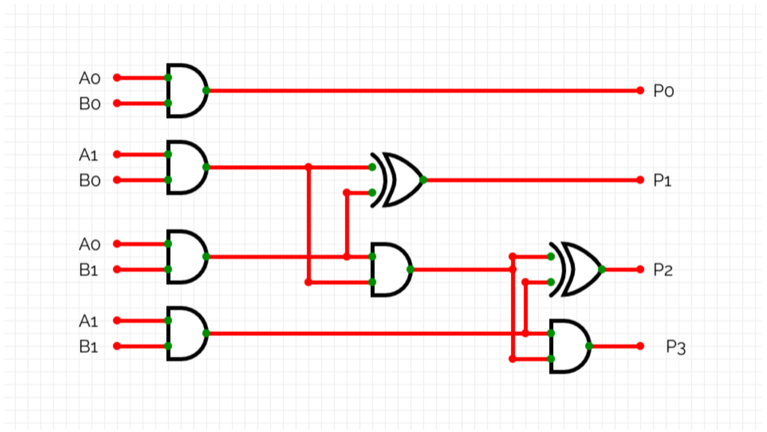

Figure 3.1: Circuit diagram of 2:1 MUX

- Multipliers are complex complicated adder arrays. this can be associate degree operation common to an oversized range of applications, and also the complexness of this operate has cause an out-sized quantity of analysis directed at rushing up its execution.

- Multipliers may be enforced exploitation totally different algorithms. looking on the rule used, the performance characteristics of the multipliers vary.
- In the implementation of digital multipliers, binary adders are an important part. With the emergence of power as a style thought, speed isn't the sole criterion by that numerous implementations are judged.
- Designing multipliers with low power, energy-efficient adders reduces the ability consumption and competence of multipliers.
- To study the performance analysis of these 3 parallel digital multipliers we tend to enforced them mistreatment 3 adder cells.

3.1.1 2:1 Mux using OR gate

We begin with the equation of 2:1 MUX, wherever inputs of the mux are 'A' and 'B'. select line is 'S' and output pin is 'Out'.

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

We need to come back with an or gate, we want we want to stay the ORing term '+' within the equation, this implies that we are able to not tie 'S' to one or zero. attributable to tie 'S' to one or zero, one among the terms within the equation of MUX can have a product of zero which term can disappear.

What I mean is :

If we tie 'S' to 0.

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

If we tie 'S' to 1.

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

In each cases, we glance at the ORing term '+'.

Same way, we are able to not tie either 'A' or 'B' to zero to keep up the '+' term.

Let's begin with 'A' and tie to one.

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

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

This is what we end up with. We really want :

$$Out = S + B$$

. If you are a master at Boolean algebra or digital logic, you can look at equation

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

and tell that it's equivalent to

$$Out = S + B$$

But do not worry. We'll prove that to be the case.

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

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

[taking bar of both sides]

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

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

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

[Applying De-morgan's rule on right most term]

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

[Expanding right hand side]

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

[Simplifying first term on right hand side]

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

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

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

$$Out = S + B$$

[Taking bar of both sides]

Hence we prove that if we tie input 'A' to 1, for a 2:1 MUX, we get an OR gate.

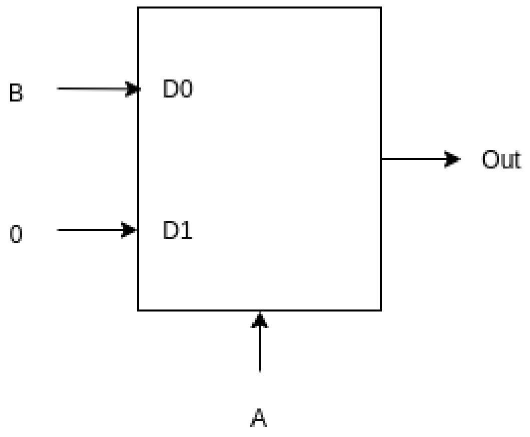

FIGURE 3.2: Logic diagram of 2:1 MUX using OR Gate

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

TABLE 3.1: Truth Table of 2:1 MUX using OR gate

3.1.2 2:1 Mux using AND gate

As we have a tendency to mentioned within the previous subdivision, we will begin with the equation of the MUX and scale back it right down to the equation of an AND gate just like the following.

For a 2:1 MUX with input A and B and select line S and output is Out, the subsequent is that the equation.

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

We can see that initial product term within the add of product on right side is truly an AND operation. All we'd like to try to to is zero out the second product term. we are able to accomplish this by creating B as '0'. we will tie B input of MUX to zero.

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

$$Out = S * A + 0$$

$$Out = S * A$$

Thus we will get AND gate by fastening B input of MUX to zero, ensuing circuit is gate that ANDs input A and mux selects S.

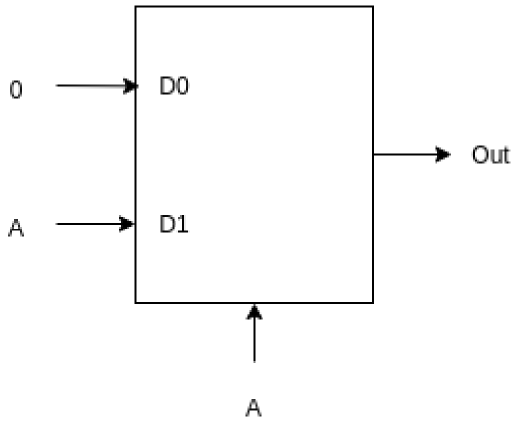

FIGURE 3.3: Logic diagram of 2:1 MUX using AND Gate

A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

TABLE 3.2: Truth Table of 2:1 MUX using AND gate

3.1.3 2:1 Mux using NOT gate

MUX has 2 inputs A and B and also the select pin is S and output pin is Out.

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

As we wish to create a NOT gate we tend to are searching for the equation of the shape.

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

$$Out = \bar{A}$$

So if input of NOT is zero, it selects the input zero that is tied to one

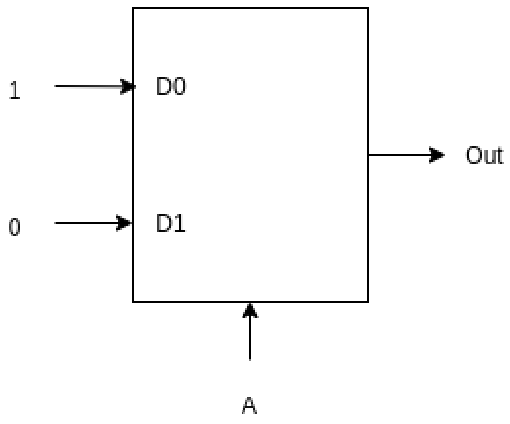

Figure 3.4: Logic diagram of 2:1 MUX using NOT Gate

A	Out
0	1
1	0

Table 3.3: Truth Table of 2:1 MUX using NOT gate

3.1.4 2:1 Mux using NOR gate

As previously mentioned we tend to begin with the equation for 2:1 MUX like following. MUX has 2 inputs A and B and select pin is S and output pin Out

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

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

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

Just confine mind that we don't need to get very same equation with a similar pins, we might use any of the 3 input pins A, B, and S.

Where will we start? this is often one thing I failed to mention whereas forming a NAND gate, however this is applicable to each NAND and NOR gate. we've already created gate, OR gate, and inverter using 2:1 MUX. you'll be able to use gate and electrical converter and mix them to create NAND. Similarly, you'll be able to use logic gate and electrical converter and mix them to create a NOR gate. that will be one choice to come back with a NOR gate using 2:1 MUX.

Here we'll attempt to come back with a NOR gate exploitation another means.

Going back we have

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

and we want to convert this to a NOR equation.

We want form

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

which is same as

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

In our 2:1 MUX equation, you'll be able to able to simply see that we'd to own $\bar{B}$ in place of B and that we go to zero out the 'S * A' term.

We can reach each by attachment of in place of B and tying 0 to input A.

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

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

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

[DeMorgan's rule]

That is our NOR gate.

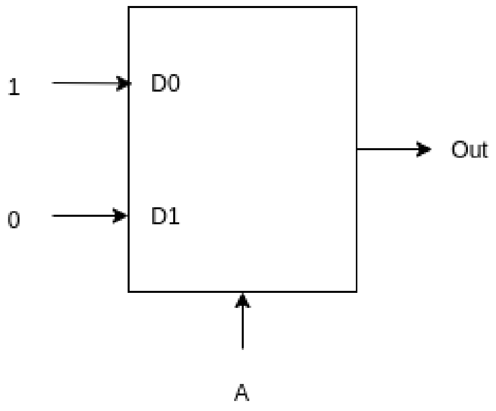

FIGURE 3.5: Logic diagram of 2:1 MUX using NOR Gate

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

TABLE 3.4: Truth Table of 2:1 MUX using NOR gate

3.1.5 2:1 Mux using NAND gate

MUX has 2 inputs A and B and also the select pin is S and output pin Out.

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

We need to come back with a NAND gate and therefore the equation of a NAND gate is of the form :

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

or

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

[using De Morgan's rule]
Given the form of equations that we've got to start with it's like we'd have a more robust

shot of changing. This can be purely a guess.

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

to the latter kind of NAND equation.

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

If we tend to really rather than of pin A to the MUX and if we tie 1 to the B input

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

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

Now lets prove that this is same as

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

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

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

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

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

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

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

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

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

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

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

Our guess worked and we have a NAND gate by tying $\bar{A}$ instead of A and tying B to 1.

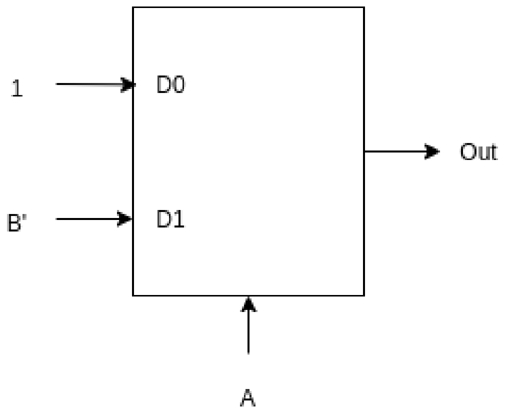

FIGURE 3.6: Logic diagram of 2:1 MUX using NAND Gate

A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

TABLE 3.5: Truth Table of 2:1 MUX using NAND gate

3.1.6 2:1 Mux using XOR gate

XOR gate is quite a special gate. we have a tendency to are acquainted with the truth table of the XOR gate.

We often use the symbol OR symbol '+' with a circle round it to represent the XOR operation. there's an alternate way to describe XOR operation, which one will observe supported the truth table. one amongst the simplest ways that to seek out out an equation illustration of any table is to use K-maps.

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

Now having this equation at our hand it's easier to start out with the 2:1 MUX equation and convert it to the XOR equation that we would like. 2:1 MUX equation is :

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

We can see that if we replace A with , we get what we want.

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

This is the XOR between S and B. Following is that the figure for identical.

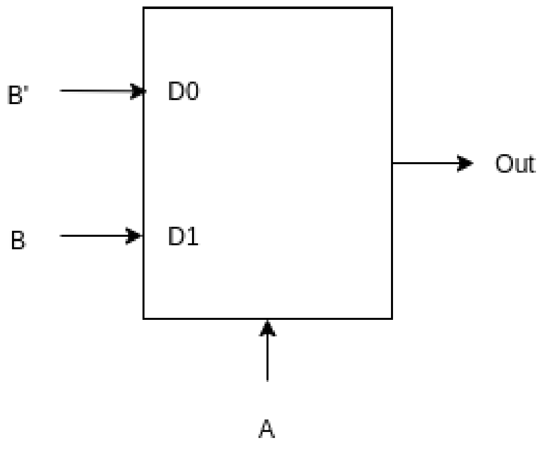

FIGURE 3.7: Logic diagram of 2:1 MUX using XOR Gate

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

TABLE 3.6: Truth Table of 2:1 MUX using XNOR gate

3.1.7 2:1 Mux using XNOR gate

Alternate equation for an XNOR gate is :

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

This can be derived using Kmaps as following We know that the equation for a 2:1 MUX is of following form :

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

We need to turn this of the form

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

It seems easy enough to achieve this. In the 2:1 MUX equation, if we tie $\bar{A}$ in place of B, we get

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

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

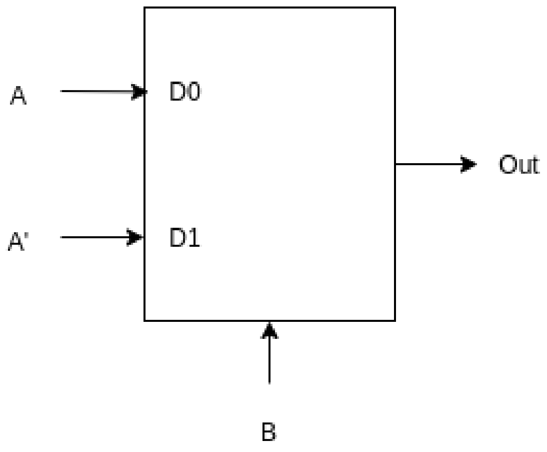

FIGURE 3.8: Logic diagram of 2:1 MUX using XNOR gate

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	1

TABLE 3.7: Truth Table of 2:1 MUX using XNOR gate

3.2 Realization of Primitive gates using 4X1 Multiplexer

A 4x1 multiplexer consists four data input lines as D0 to D3, two select lines as S0 and S1 and a single output line Y. The select lines S1 and S2 select one among the four input lines to attach the output line. the actual input combination on select lines selects one among input (D0 through D3) to the output.

The figure below shows the Logic diagram of a 4-to-1 multiplexer in which the multiplexer decodes the input through select line.

The truth table of a 4x1 multiplexer is shown below within which four input combos 00, 10, 01 and 11 on the select lines severally switches the inputs D0, D2, D1 and D3 to the output. which means when S1=0 and S0=0, the output at Y is D0, similarly Y is D1 if the choose inputs S1=0 and S0=1 and then on.

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

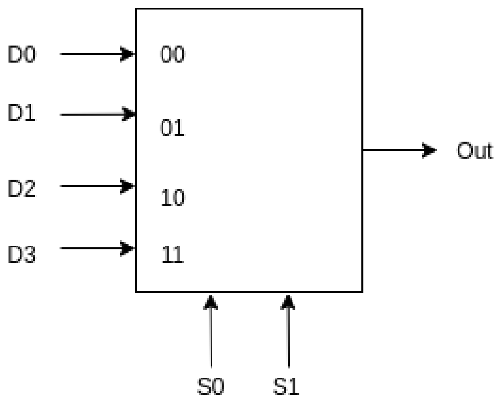

FIGURE 3.9: Logic diagram of 4:1 MUX

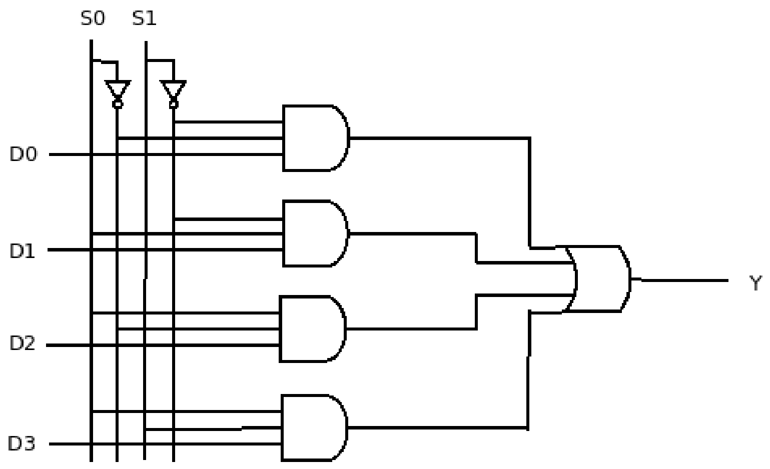

FIGURE 3.10: Circuit diagram of 4:1 MUX

S0	S1	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3

TABLE 3.8: Truth Table of 4:1 MUX

If S1=0 and S0=0 then Y = D0 Therefore,

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

If S1= 0 and S0=1, the Y = D1 Therefore,

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

If S1=1 and S0=0, then Y = D2 Therefore,

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

If S1=1 and S0=1 the Y = D3 Therefore,

$$Y = D3S1S0$$

To get the overall information output from the multiplexer, of these product terms are to be summed so the final Boolean expression of this multiplexer is given as

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

From the on top of expression of the output, a 4-to-1 multiplexer will be implemented by exploitation basic logic gates. The below figure shows the logic circuit of 4:1 MUX that is implemented by four 3-inputs AND gates, two 1-input NOT gates, and one 4-inputs OR gate.

In this circuit, every information input line is connected as input to AND gate and 2 select lines are connected as alternative 2 inputs to that. The AND gate output is connected to with inputs of OR gate therefore as to turn out the output Y.

3.2.1 4:1 MUX using AND gate

The truth table of a 4-to-1 multiplexer is using AND gate is below in which four input combinations 00, 10, 01 and 11 on the select lines respectively switches the inputs D0, D1, D2 and D3 to the output. That means when S1=0 and S0=0, the output at Y is D0, similarly Y is D1 if the select inputs S1=0 and S0=1 and Y is D2 if if the select inputs S1=1 and S0=0, its shows the Output for Y is D3 if the select inputs S1=1 and S0=1.

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

TABLE 3.9: Truth Table of 2:1 MUX using AND gate

As we compare total data output from the multiplexer all these product terms are to besummed and then the final Boolean expression of this multiplexer is given as

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

From the expression of the output, a 4-to-1 multiplexer can be implemented by using logic gates but we cannot implement it using only AND gate but along with NOT gate.

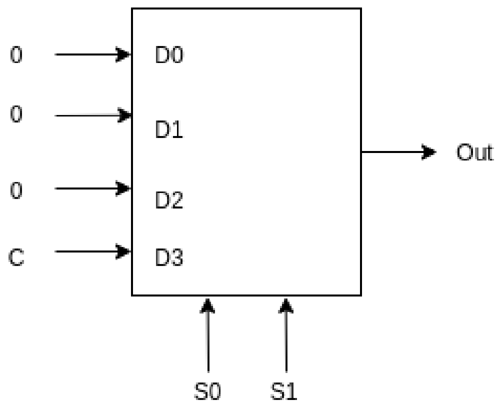

FIGURE 3.11: Logic diagram of 4:1 MUX using AND gate

3.2.2 4:1 MUX using OR gate

The truth table of a 4-to-1 multiplexer is using OR gate is below in which four input combinations 00, 10, 01 and 11 on the select lines respectively switches the inputs D0, D1, D2 and D3 to the output. That means when S1=0 and S0=0, the output at Y is D0 where remaining 3-inputs shows the outputs as 1 at Y is D1 if inputs are S1=0 and S0=1 similarly Y is D2 if S1=1 and S0=0 and Y is D3 S1=1 and S0=1.

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

TABLE 3.10: Truth Table of 4:1 MUX using OR gate

As we compare total data output from the multiplexer all these product terms are to besummed and then the final Boolean expression of this multiplexer is given as

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

From the expression of the output, a 4-to-1 multiplexer can be implemented by using logic gates but we cannot implement it using only OR gate but along with NOT gate.

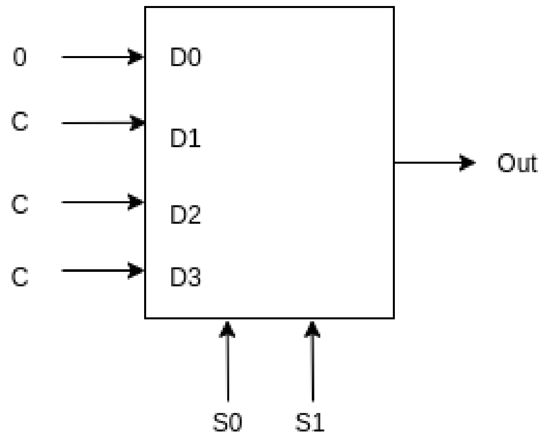

FIGURE 3.12: Logic diagram of 4:1 MUX using OR gate

3.2.3 4:1 MUX using NOR gate

The truth table of a 4-to-1 multiplexer is using NOR gate is below in which four input combinations 00, 10, 01 and 11 on the select lines respectively switches the inputs D0, D1, D2 and D3 to the output. That means when S1=0 and S0=0, the output at Y=1 is D0, for Remaining three inputs are Y=0 is D1,D2,D3 if the select inputs S1=0 and S0=1, S1=1 and S0=0 and S1=1 and S0=1. It's always the negation of the OR gate. As we

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

TABLE 3.11: Truth Table of 2:1 MUX using NOR gate

compare total data output from the multiplexer all these product terms are to besummed and then the final Boolean expression of this multiplexer is given as

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

From the expression of the output, a 4-to-1 multiplexer can be implemented by using logic gates but we cannot implement it using only NOR gate.

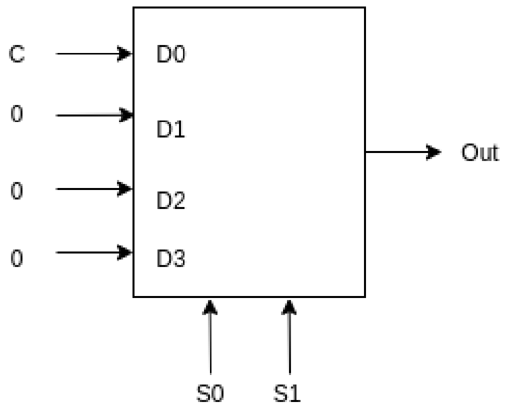

FIGURE 3.13: Logic diagram of 4:1 MUX using NOR gate

3.2.4 4:1 MUX using NAND gate

The truth table of a 4-to-1 multiplexer is using NAND gate is below in which four input combinations 00, 10, 01 and 11 on the select lines respectively switches the inputs D0, D1, D2 and D3 to the output. That means when S1=0 and S0=0, the output at Y=1 is D0, similarly for D1, D2 if the select inputs S1=0 and S0=1 and S1=1 and S0=0, its shows the Output for Y=0 is D3 if the select inputs S1=1 and S0=1. It shows always negation of AND gate.

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

TABLE 3.12: Truth Table of 2:1 MUX using NAND gate

As we compare total data output from the multiplexer all these product terms are to besummed and then the final Boolean expression of this multiplexer is given as

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

From the expression of the output, a 4-to-1 multiplexer can be implemented by using logic gates but we can implement it using only NAND gate i.e 3-input NAND gate.

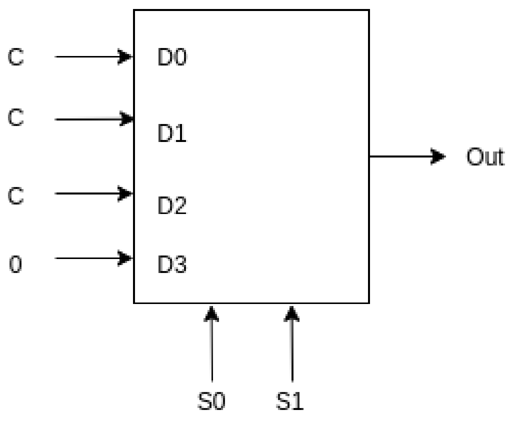

Figure 3.14: Logic diagram of 4:1 MUX using NAND gate

3.2.5 4:1 MUX using XOR gate

The truth table of a 4-to-1 multiplexer is using XOR gate is below in which four input combinations 00, 10, 01 and 11 on the select lines respectively switches the inputs D0, D1, D2 and D3 to the output. That means when S1=0 and S0=0 , S1=1 and S0=1 the output at Y=0 is D0 and D3, similarly Y=1 is D1 and D2 if the select inputs S1=0 and S0=1, S1=1 and S0=0. It's negation of XNOR gate.

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

Table 3.13: Truth Table of 2:1 MUX using XOR gate

As we compare total data output from the multiplexer all these product terms are to besummed and then the final Boolean expression of this multiplexer is given as

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

From the expression of the output, a 4-to-1 multiplexer can be implemented by using logic gates but we cannot implement it using only XOR gate.

3.2.6 4:1 MUX using XNOR gate

The truth table of a 4-to-1 multiplexer is using XNOR gate is below in which four input combinations 00, 10, 01 and 11 on the select lines respectively switches the inputs D0, D1,

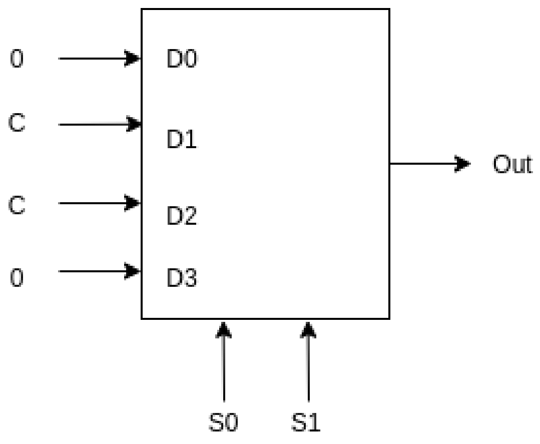

FIGURE 3.15: Logic diagram of 4:1 MUX using XOR gate

D2 and D3 to the output. That means when S1=0 and S0=0 , S1=1 and S0=1 the output at Y=1 is D0 and D3, similarly Y=0 is D1 and D2 if the select inputs S1=0 and S0=1, S1=1 and S0=0. It's negation of XOR gate.

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

TABLE 3.14: Truth Table of 2:1 MUX using XNOR gate

As we compare total data output from the multiplexer all these product terms are to besummed and then the final Boolean expression of this multiplexer is given as

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

From the expression of the output, a 4-to-1 multiplexer can be implemented by using logic gates but we cannot implement it using only XNOR gate.

3.2.7 4:1 MUX using NOT gate

The Truth table of 4-to-1 multiplexer is using NOT gate is below in which two input combinations 0 and 1 using dummy variable S1

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

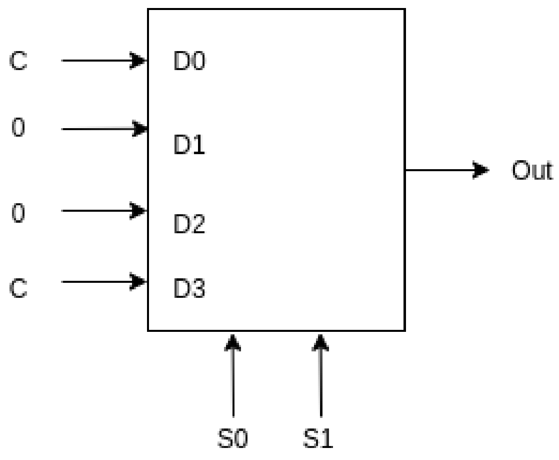

FIGURE 3.16: Logic diagram of 4:1 MUX using XNOR gate

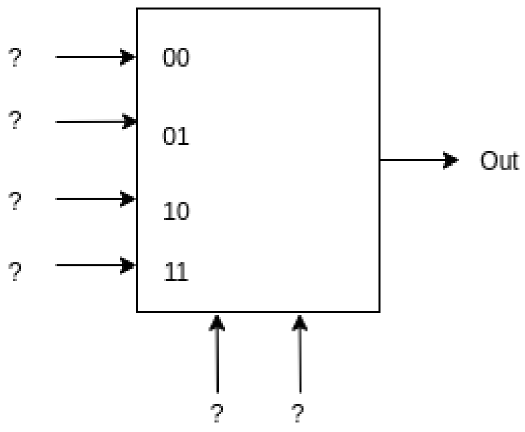

FIGURE 3.17: Logic diagram of 4:1 MUX using Dummy X

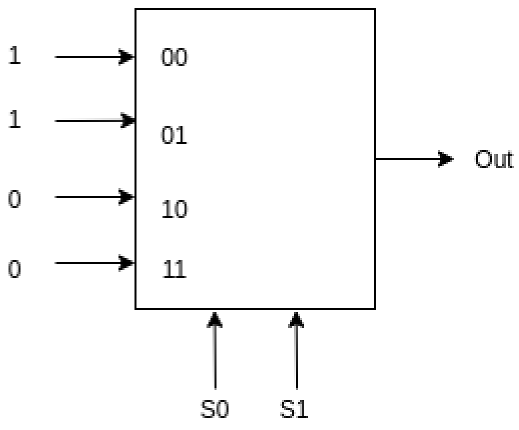

FIGURE 3.18: Logic diagram of 4:1 MUX using NOT gate

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

TABLE 3.15: Truth Table of 4:1 MUX using NOT gate

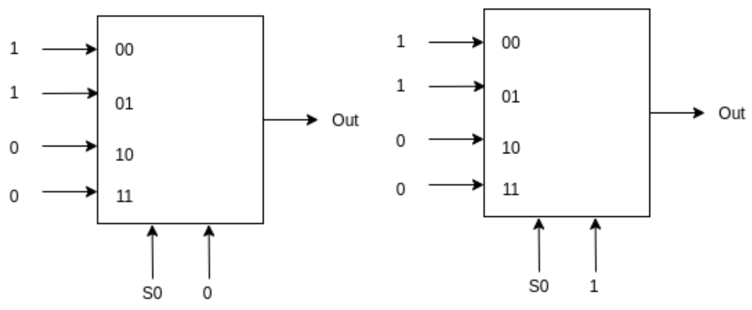

FIGURE 3.19: Logic diagram of 4:1 MUX using Alternative NOT gate

3.3 Proposed system

The RTAX-S family provides two types of logic modules: the register cell (R-cell) and the combinatorial cell (C-cell). The RTAX-S C-cell can implement more than 4,000 combinatorial functions of up to five inputs (Figure 2.3 on page 5). The C-cell contains carry logic for even more efficient implementation of arithmetic functions. For even a lot of efficient implementation of arithmetic functions. With its tiny size, the C-cell structure is very synthesis-friendly, simplifying the general design also as 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.

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

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

From the above expression of the output, a 4-to-1 multiplexer can be implemented by using logic gates i.e generalized using select lines S0 = (A1 + B1) as OR gate and S1 = (A0 * B0) as AND gate.

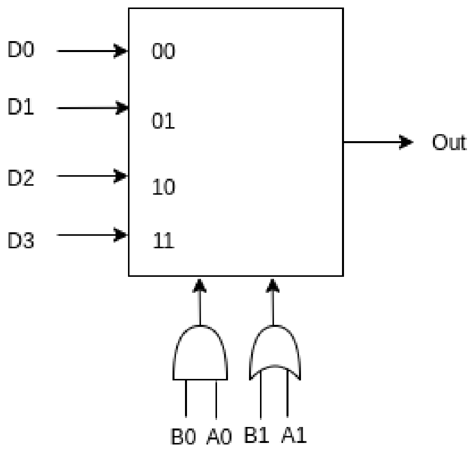

FIGURE 3.20: Logic diagram of C-Cell

The generalized expression from the equation (1) is

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

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

A0 B0	A1 B1	A0*B0	A1+B1	Y
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

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

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

using De Morgan's rule

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

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

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

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

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

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

Analysis the generalized equation when A0 = 0

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

Analysis the generalized equation when A1 = 0

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

Analysis the generalized equation when B0 = 0

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

Analysis the generalized equation when B1 = 0

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$Y = D2 + D2B1$$

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

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

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

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

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

$$Y = D2 + D2B1$$

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

$$Y = D3B1$$

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

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

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

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

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

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

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

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

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

$$Y = D1B0$$

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

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

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

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

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

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

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) 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

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.

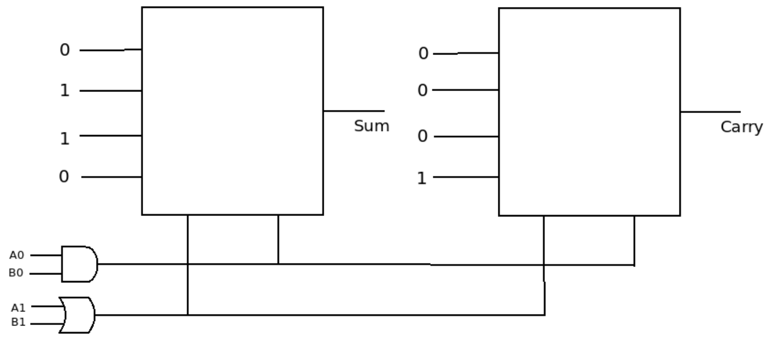

Figure 3.21: Logic diagram of C-Cell using Half Adder

- 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:

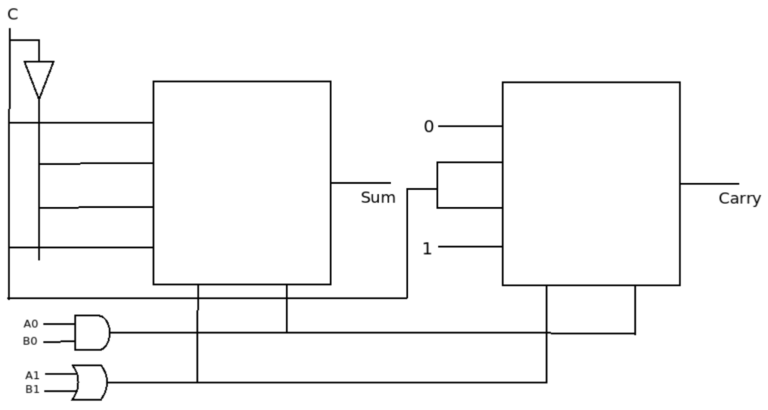

FIGURE 3.22: Logic diagram of C-Cell using Full Adder

3.6 Design Of C-Cell using 4 Bit Adder

A full adder C-cell is a combinational circuit that forms the arithmetic sum of three input bits. It has two select lines S0 = AND(A0,B0) and S1 = OR(A1,B1), that represent the two significant bits to be added, and C input that is a carry-in from the previous significant position. We can implement 4-bit binary adder in one of the two following 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.
- Use four Full adders for uniformity. Since, initial carry Cin zero, the Full adder which is used for adding the least significant bits becomes Half adder.
- It has two outputs: S which is the sum of the two input bits which can be D0-D3 and Cout to carry the value in case the output from S is 12 or 13 because the binary forms of these require two digits for their representation.

For the time being, we considered second approach. The block diagram of 4-bit binary adder is shown in the following figure Here, the 4 Full adders are cascaded. Each Full

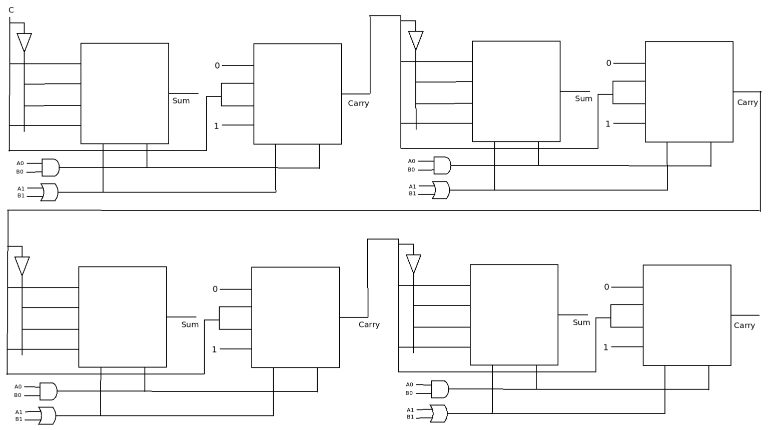

FIGURE 3.23: Logic diagram of C-Cell using 4 bit Adder

adder is getting the respective bits of two parallel inputs S0 S1. The carry output of 1 Full adder are the carry input of resulting higher order Full adder. This 4-bit binary adder produces the resultant add having at the most 5 bits. So, carry out of last stage Full adder are going to be the MSB.

In this method, we will implement any higher order binary adder simply by cascading the specified range of Full adders. This binary adder is additionally referred to as as ripple carry (binary) adder as a result of the carry propagates (ripples) from one stage to future stage.

3.7 Design Of C-Cell using 8 Bit Adder

A full adder C-cell may be a combinational circuit that forms the arithmetic total of 3 input bits It has two select lines S0 = AND(A0,B0) and S1 = OR(A1,B1), that represent the two significant bits to be added, and C input that is a carry-in from the previous significant position.

- The 8-bit adder adds and therefore the 8-bit binary inputs and also the result's made within the output. in order to make a Full 8-bit adder, I might use eight Full 1-bit adders and connecting them.
- It has two outputs: S which is the sum of the two input bits which can be D0-D3 and Cout to carry the value in case the output from S is 192 or 255 because the binary forms of these require two digits for their representation, output is given zero always and output 477 that represents in the binary representation.

- We take eight of those full adders, currently that we've shown that they work, and mix them to make an 8-bit Adder.

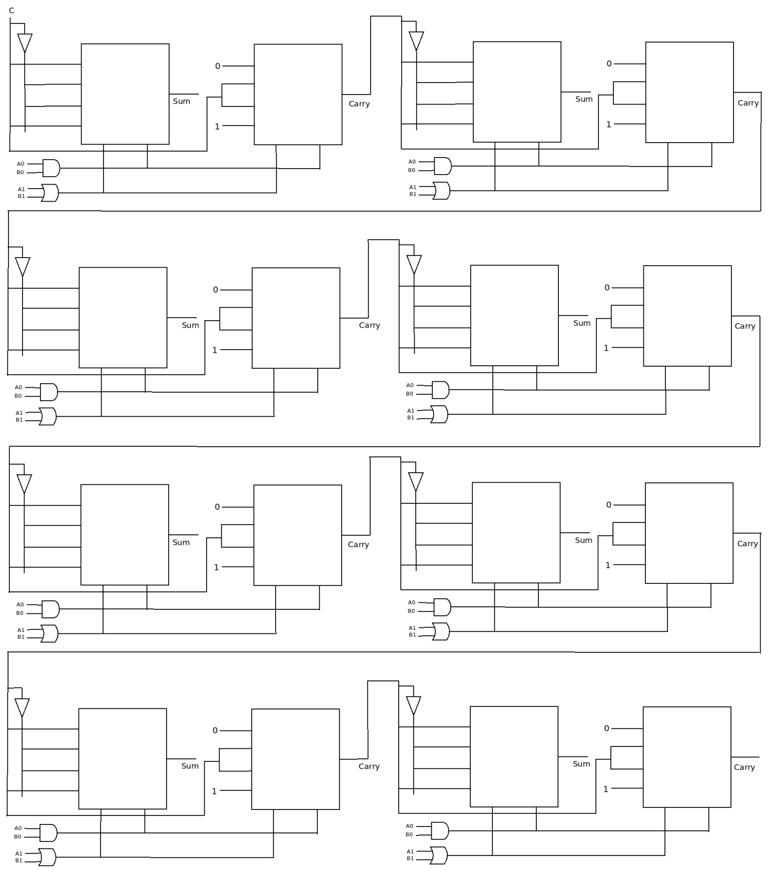

FIGURE 3.24: Logic diagram of C-Cell using 8 bit Adder

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 as output of the above logic design.Here, the 8 Full adders are cascaded. Each Full adder is getting the respective bits of two parallel inputs S0 S1. The carry output of 1 Full adder are the carry input of later higher order Full adder. This eight-bit binary adder produces the resultant total having at the most 8 bits. This 8-bit binary adder produces the resultant sum having at most 8 bits. So, perform of last stage Full adder are the MSB.

The 8bit Adder is comprised of eight Full Adders which are connected by their carry in/out. The Full Adder gives the sum of two, 1-bit inputs. Likewise, the 8-bit Adder provides the total of two 8-bit inputs. The four to one c-cell multiplexer itself has four inputs, a pair of selectors, and one output.

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 design structure of the array Multiplier is regular, it is based on the add shift algorithm principle.

Partial product = the multiplicand * multiplier bit......(2)

where AND gates are used for the product, the summation is finished mistreatment Full Adders and half Adders wherever the partial product is shifted per their bit orders. In an n*n array multiplier, n*n AND gates calculate the partial product and therefore the addition of partial products are often performed 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.

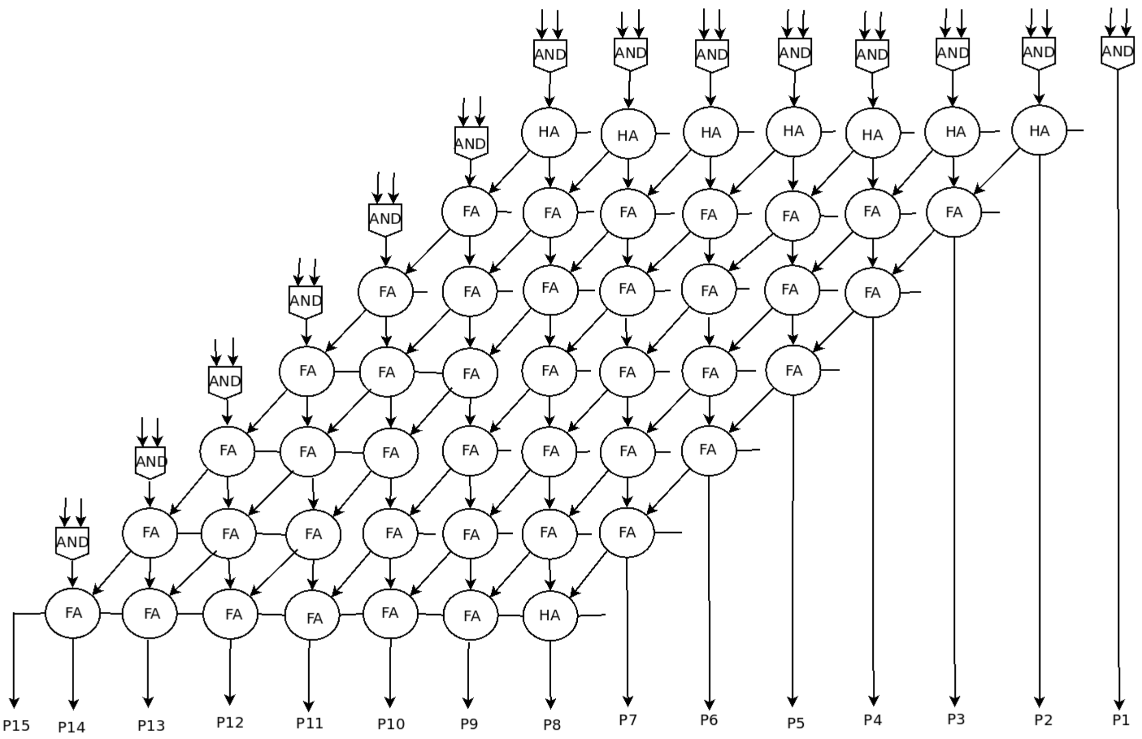

Figure 3.25: Array Multiplier

3.9 Design Of C-Cell using 4 Array Multiplier

The 4×4 array multiplier of C-cell shown has eight inputs and eight outputs. A full adder has 3 input lines and 2 output lines, wherever we use this as a basic building block of an array multiplier. the following is that the instance of a 4×4 array multiplier of C-cell. The left bit is that the LSB bit of partial product. The right bit is that the msb bit of partial product. The partial product are currently shifted towards the left facet on multiplication and that they're added to induce the last word product. This method is continual till no 2 partial product exit for addition

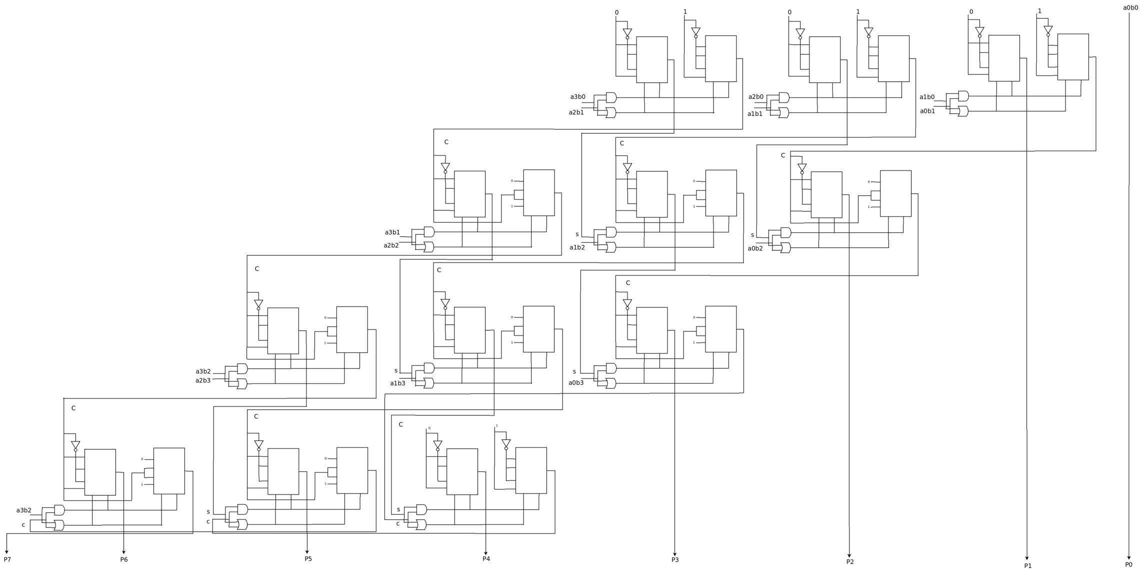

FIGURE 3.26: Logic diagram of 4X4 Array Multiplier using C-Cell

Where a0,a1,a2,a3 and b0,b1,b2,b3 are number and multiplier, summation of all product are partial product.The results of the total of the partial product may be a product.

For a 4×4 Array Multiplier, it needs 16 AND gates, 4 Half Adder C-cell, 8 Full Adders C-cell. Total 12 C-cell Adders.

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. A full adder has 3 input lines and 2 output lines, wherever we use this as a basic building block of an array multiplier. the subsequent is that the example of an 8×8 array multiplier of C-cell. The left bit is that the LSB little bit of partial product. The right bit is that the MSB little bit of partial product. The partial product are currently shifted towards the left facet on multiplication and that they are side to urge the ultimate product. This method is recurrent till no 2 partial products exit for addition.

Where a0,a1,a2,a3,a4,a5,a6,a7 and b0,b1,b2,b3,b4,b5,b6,b7 are multiplicand and multiplier, summation of all product are partial product.The results of the add of the partial product is a product.

For a 8×8 Array Multiplier, it needs 32 AND gates, 8 Half Adder C-cell, 48 Full Adders C-cell. Total 56 C-cell Adders.

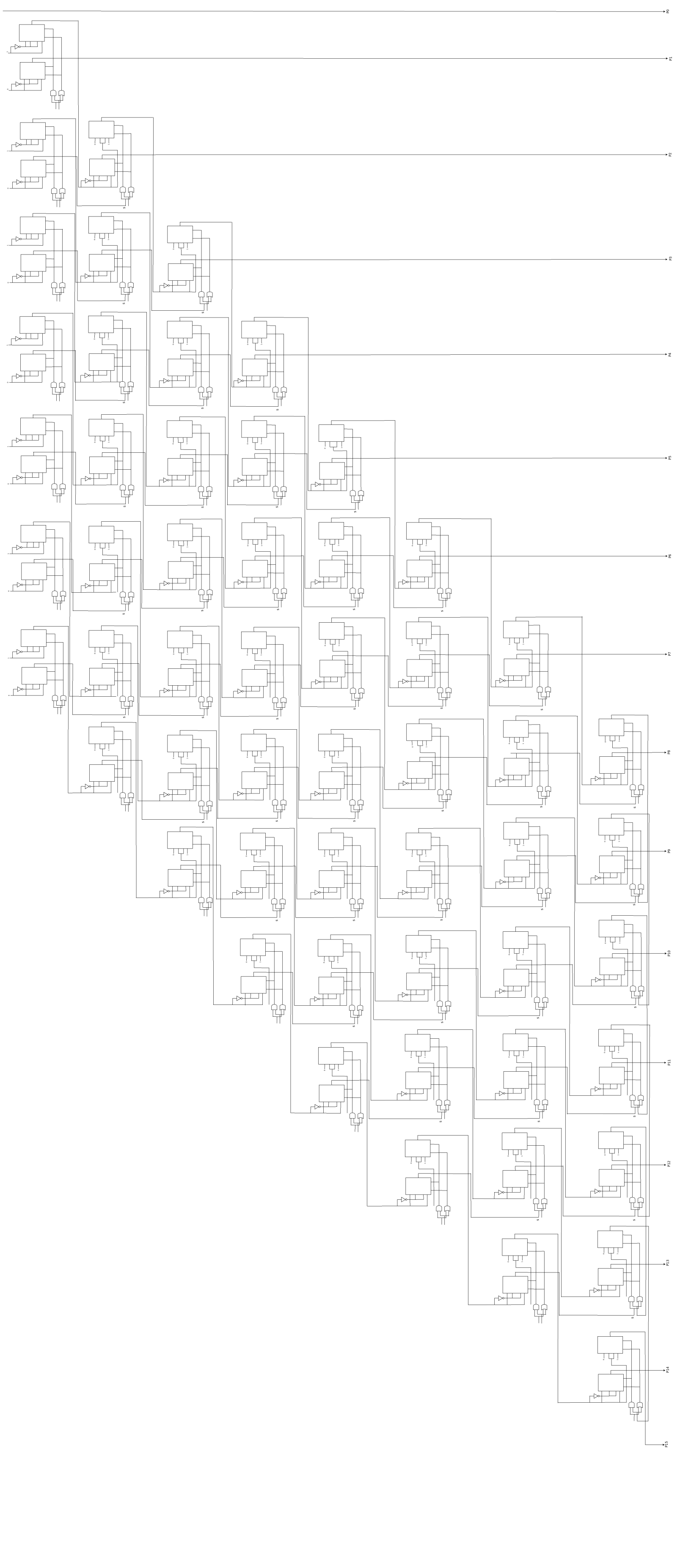

Figure 3.27: Logic diagram of 8X8 Array Multiplier using C-Cell

Chapter 4

Results and Conclusion

4.1 Results

4.1.1 Existing fault tolerant full adder designs

- In C-Cell ,the single event upset can occur at the select line of multiplexer.
- The proposed design of the multi-bit c-cell Adders can detect errors and repair the faults using SEU approaches without using TMR-based approach and it can't detect the stuck-at fault.
- Error injection in different bit positions of adder with respective 4-bit ,8-bit and so on upto 128 bit c-cell.
- Generating the Error Difference between original Result and Error Result.
- Error Injection from LSB.
- Too find the highest MSB percentage using c-cell array-multiplier.
- This analysis can be helpful to add the fault tolerance techniques at the highest error percentage bit.
- n-bit C-Cell adder with Error injection at S0 select line and S1 select line.

The Results of 4bit Adder with some example within 4bit bit

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

4-bit C-Cell adder with Error injection at S0 select 4-bit C-Cell adder with Error injection

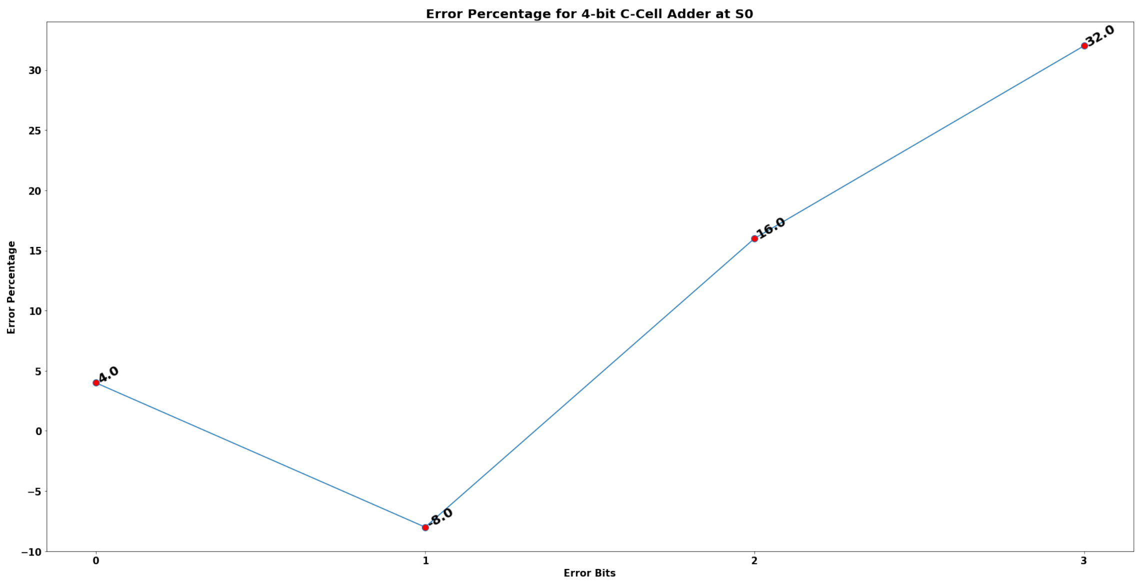

FIGURE 4.1: Graph analysis of 4bit Adder at Select line S0

at S1 select line

The Results of 8bit Adder with some example within 8bit bit

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

8-bit C-Cell adder with Error analysis using Graph

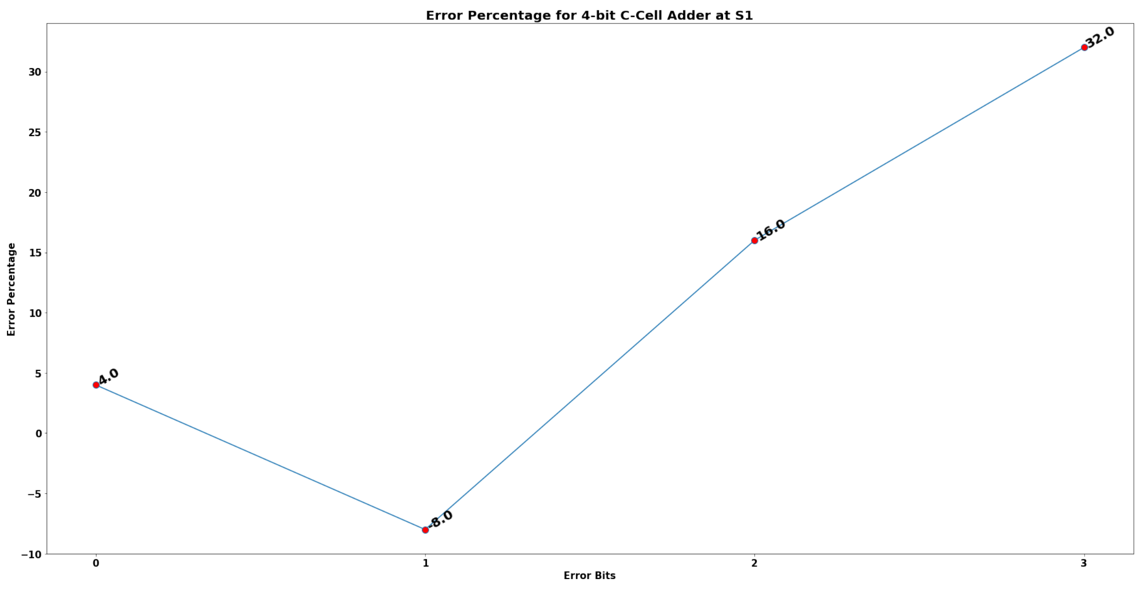

Figure 4.2: Graph analysis of 4bit Adder at Select line S1

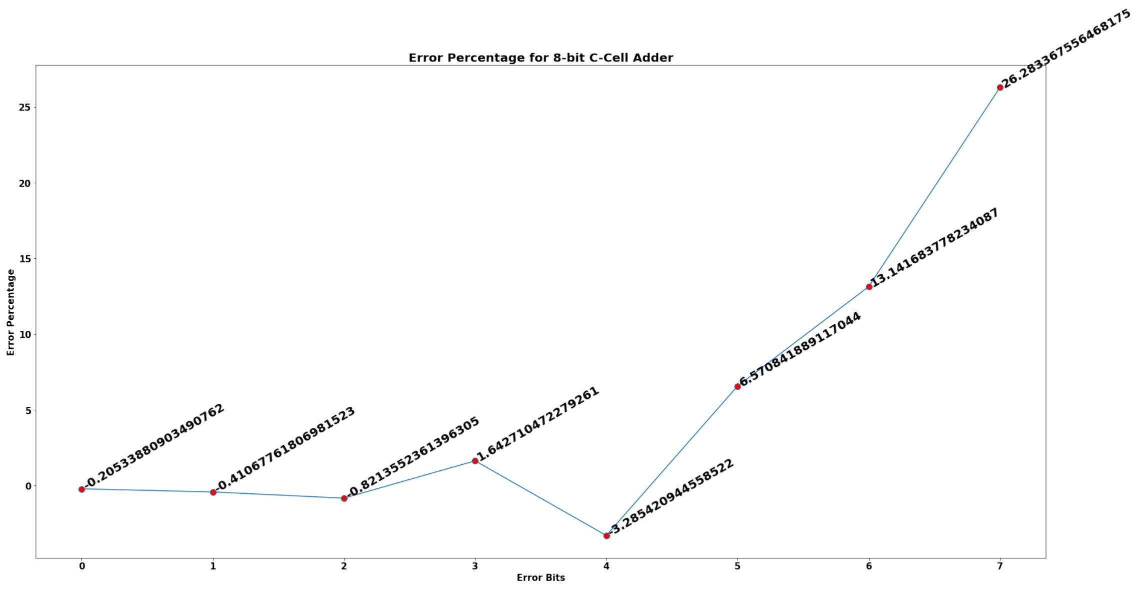

Figure 4.3: Graph analysis of 8bit Adder at Select line S1

4.1.2 Existing fault tolerant multipliers

- We should find the errors at the select lines.
- Bit position will always defines on the Adders position in the select line.
- The Bit position will start with position 1, because the zeroth position is directly multiplied.
- Here we should always enter the Adder position to find the exact errors.
- Error injection in different bit positions of multiplier with respective 4-bit ,8-bit and so on upto 128 bit c-cell.
- This analysis can be helpful to add the fault tolerance techniques at the highest error percentage bit.
- Below some example for 4x4 array multiplier and 8x8 array multiplier using example.

Bit-position	Adders number position in select line
1	1
2	2,8
3	3,9,12
4	10,13
5	14
6	15

4x4 array multiplier C-Cell with Error injection at select line

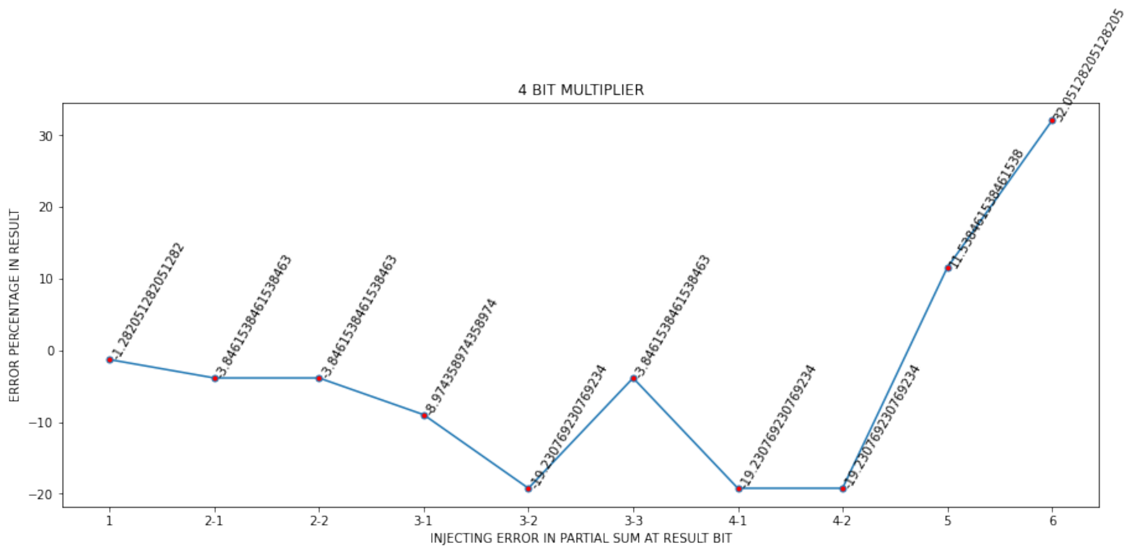

FIGURE 4.4: Graph analysis of 4x4 array multiplier at Select line

Bit-position	Adders number position in select line
1	1
2	2,16
3	3,17,24
4	4,18,25,32
5	5, 19, 26, 33, 40
6	6, 20, 27, 34, 41, 48
7	7, 21, 28, 35, 42, 49
8	22, 29, 36, 43, 50, 56, 57
9	30, 37, 44, 51, 58
10	38, 45, 52, 59
11	46, 53, 60
12	54, 61
13	62
14	63

8x8 array multiplier C-Cell with Error injection at select line

4.2 Conclusion

In this paper, a new technique for self checking and self repairable carry save adder with minimal area overhead has been proposed. The proposed design can detect and repair both

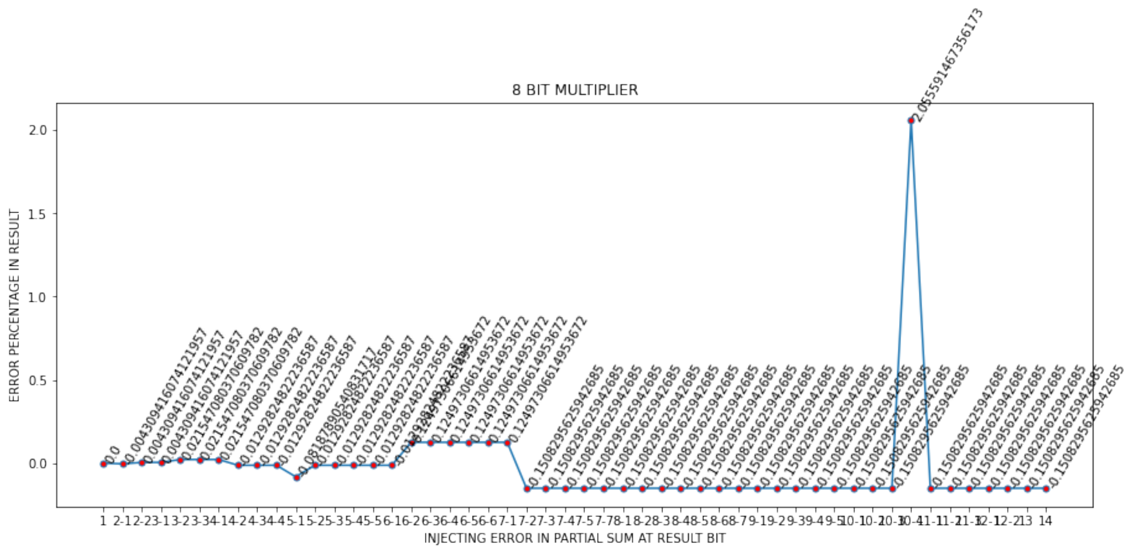

FIGURE 4.5: Graph analysis of 8x8 array multiplier at Select line

single and multiple faults at a time. Hence, this design is free from the problem of fault propagation through carry. The proposed fault tolerant full adder is compared in terms of single and multi-bit error detection and correction possibility with the Single Event Upset(SEU) - based,self testing and self repairing full adder design and array multiplier Design.

The comparison results of the proposed designs are found better that ensure its superior performance capability. The proposed design is extendable up to a desirable level. A 4-bit fault tolerant adder, 8-bit fault adder and up to 128-bits fault tolerant adder is also implemented using the proposed design. It works efficiently when cascaded and can handle single and multi-bit faults successfully.

The comparison results of the proposed designs are found better to ensure its performance capability. The proposed design is extendable up to a desirable level. A 4-bit fault tolerant multiplier c-cell and 8-bit fault tolerant c-cell multiplier is also implemented using the proposed design. It works more efficiently and it can handle single and multi-bit faults successfully.

4.3 future work

- The C-cell connection with register(Flip-Flops) will be analyzed.
- Minimising the error behaviour of the Axcelerator super-Cluster (Two clusters).

- 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] P. Kumar and R. K. Sharma, "Real-time fault tolerant full adder design for critical applications," *Engineering Science and Technology, an International Journal*, vol. 19, pp. 1465–1472, 2016.

[7] 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.

[8] S. Peng and R. Manohar, "Fault tolerant asynchronous adder through dynamic self-reconfiguration."

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

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