
**University of Tripoli**

**Faculty of Engineering**

**Department of Computer Engineering**

A graduation project submitted in partial fulfilment of requirements for the degree of Bachelor in Computer Engineering

# Design and Implementation of an Alternative Microprocessor Bus Architecture for FPGA-Based Imbedded Systems

By:

Jumanah Abdulhadi Mansur Esra Abdalraof Tellisi

Supervised By:

Dr. Mohamed Muftah Eljhani

Spring 2023

**University of Tripoli**

**Faculty of Engineering**

**Department of Computer Engineering**

A graduation project submitted in partial fulfilment of requirements for the degree of Bachelor in Computer Engineering

# Design and Implementation of an Alternative Microprocessor Bus Architecture for FPGA-Based Imbedded Systems

By:

Jumanah Abdulhadi Mansur Esra Abdalraof Tellisi

| Supervisor | Signature | Date |
|---|---|---|
| Dr. Mohamed Muftah Eljhani | _____ | _____ |

Examiners

| | | |
|---|---|---|
| 1. Dr. Ashoor Emhemed Alsellami | _____ | _____ |
| 2. Dr. Abdoulmenim Ahmed Bilh | _____ | _____ |

**University of Tripoli**

**Faculty of Engineering**

**Department of Computer Engineering**

# Intellectual Property Rights Identification Form for Projects and Scientific Research

This form must be read and signed by students working on graduation projects, master's theses or any other research activities conducted at University of Tripoli / Faculty of Engineering / Department of Computer Engineering.
Intellectual property rights for projects and research activities and their results (such as graduation projects, master's theses, patents and any marketable research product) belong to the University of Tripoli/Department of Computer Engineering. These rights are subject to the laws, regulations and instructions of the University relating to intellectual property and patents.

I agree (Student's Name):

_____

Student's ID:

_____

As a condition of my participation in the graduation project entitled:

_____

_____

_____

All intellectual property rights of the above-mentioned project or scientific research shall be attributable to the University of Tripoli/Department of Computer Engineering This requires me to inform the competent authority of the University of any invention or discovery that may result from such research and to be fully confidential therein and to work through the University to obtain the patent that may result from such research. I am also committed to placing the name of Tripoli University/Department of Computer Engineering and the names of all researchers involved in the research on any scientific bulletin for full research or its results, including publication of graduation projects, master's theses, doctorates, publication in journals, scientific conferences in general and posting on websites. I must adhere to the principles of copyright approved by the University of Tripoli/Department of Computer Engineering.

Student's Signature: ----------

Date: ----------

**University of Tripoli**

**Faculty of Engineering**

**Department of Computer Engineering**

# Plagiarism Declaration

I (Student's Name):

_____

Student's ID:

_____

Hereby declare that I am the sole author of the graduation project entitled:

_____

_____

_____

And that neither any part of the thesis nor the whole of the thesis has been submitted to any University or Institution for obtaining any degree / diploma / academic award.

This project was written by me and in my own words, except for quotations from published and unpublished sources, which are clearly indicated and acknowledged as such. I am conscious that the incorporation of material from other works or a paraphrase of such material without acknowledgement will be treated as plagiarism, subject to the custom and usage of the subject, according to the University Regulations on Conduct of Examinations.

I shall be solely responsible for any dispute or plagiarism issue arising out of the graduation project.

Student's Signature: ----------

Date: ----------

**University of Tripoli**

**Faculty of Engineering**

**Department of Computer Engineering**

# Intellectual Property Rights Identification Form for Projects and Scientific Research

This form must be read and signed by students working on graduation projects, master's theses or any other research activities conducted at University of Tripoli / Faculty of Engineering / Department of Computer Engineering.
Intellectual property rights for projects and research activities and their results (such as graduation projects, master's theses, patents and any marketable research product) belong to the University of Tripoli/Department of Computer Engineering. These rights are subject to the laws, regulations and instructions of the University relating to intellectual property and patents.

I agree (Student's Name):

_____

Student's ID:

_____

As a condition of my participation in the graduation project entitled:

_____

_____

_____

All intellectual property rights of the above-mentioned project or scientific research shall be attributable to the University of Tripoli/Department of Computer Engineering This requires me to inform the competent authority of the University of any invention or discovery that may result from such research and to be fully confidential therein and to work through the University to obtain the patent that may result from such research. I am also committed to placing the name of Tripoli University/Department of Computer Engineering and the names of all researchers involved in the research on any scientific bulletin for full research or its results, including publication of graduation projects, master's theses, doctorates, publication in journals, scientific conferences in general and posting on websites. I must adhere to the principles of copyright approved by the University of Tripoli/Department of Computer Engineering.

Student's Signature: ----------

Date: ----------

**University of Tripoli**

**Faculty of Engineering**

**Department of Computer Engineering**

# Plagiarism Declaration

I (Student's Name):

_____

Student's ID:

_____

Hereby declare that I am the sole author of the graduation project entitled:

_____

_____

_____

And that neither any part of the thesis nor the whole of the thesis has been submitted to any University or Institution for obtaining any degree / diploma / academic award.

This project was written by me and in my own words, except for quotations from published and unpublished sources, which are clearly indicated and acknowledged as such. I am conscious that the incorporation of material from other works or a paraphrase of such material without acknowledgement will be treated as plagiarism, subject to the custom and usage of the subject, according to the University Regulations on Conduct of Examinations.

I shall be solely responsible for any dispute or plagiarism issue arising out of the graduation project.

Student's Signature: ----------

Date: ----------

# Table of Contents

# List of Figures

## List of Table

# Abstract

Field programmable gate array (FPGA) do not have enough tristate drivers to mount large buses in a large application. An alternative to a tristate-based bus structure is a new multiplexer-based bus structure and bus controller. This alternative approach can be used in large design applications with a large number of design blocks, as well as in embedded systems and mobile electronic devices that require high speed and low power consumption. In this project, the basic modules of the proposed microprocessor bus architecture designed, implemented, and simulated using Verilog hardware description language (HDL). The implemented and routed to the Cyclone IV GX FPGA. Compared to a tristate-based bus, microprocessors with a multiplexer-based bus have been proven to almost same power and less time delay. This makes them suitable for applications where power consumption is a concern. The use of multiplexer-based buses is particularly beneficial in system on programmable chip (SoPC) designs, where intellectual property (IP) integration may limit the use of tristate-based buses. Similarly, application-specific integrated circuit (ASIC) designs often utilize internal multiplexer-based buses for the same reason. One of the drawbacks of tristate-based buses is their timing and power consumption issues caused by the capacitive load of the nodes. By adopting a new multiplexer-based bus structure and bus controller, these issues can be mitigated.

Overall, this alternative approach provides an efficient solution for large design applications that require high speed, low power consumption.

# Acknowledgement

First we thanks to Allah for His countless blessings and for making this achievement possible.

We would like to extend our thanks and gratitude to our supervisor “Dr. Mohamed Muftah Eljhani” for his support, suggestions, guidance, and encouragement throughout the project.

We are thankful to all our teachers who taught us and gave us the knowledge and motivation that got us here.

Finally, All thanks to all our family and friends for all support they gave to us. Without their encouragement, this project would not have been possible.

Thank you

# List of Abbreviations

| | |
|---|---|
| **FPGA** | Field programmable gate array |
| **Verilog** | Verifying Logic |
| **Verilog HDL** | Hardware Description Language |
| **SoPC** | System on Programmable Chip |
| **IP** | Intellectual Property |
| **ASIC** | Application-Specific Integrated Circuit |
| **IC** | Integrated Circuit |
| **EDA** | Electronic Design Automation |
| **PC** | Personal Computer |
| **VHDL** | Very High Speed Integrated Circuit Hardware Description Language |
| **CPU** | Central Processing Unit |
| **GPU** | Graphics Processing Unit |
| **MUX** | Multiplexer |
| **ALU** | Arithmetic Logic Unit |
| **SUB** | Subtraction |
| **SHL** | Shift Logical Bit Left |
| **RTL** | Register Transfer Level |
| **PLD** | Programmable Logic Device |

# Chapter 1 Introduction

Early Intel's and other processors were designed by hand, laying out the layers of an integrated circuit (IC) substrate masks using regular drafting techniques. There were little or no electronic design automation (EDA) tools to help the chip developer. This method was so boring that least few people had the patience and skills for such a task. Thankfully, times have changed, and designing custom processors is within reach of many designers of such hobby. There are two predominant HDL languages, Verilog and VHDL. Verilog HDL is adopted in this project. [1], [2]. Buses, although the simplest form of interconnect, is a poor choice from a density or power standpoint because the power and space required to drive them at maximum speed grow exponentially with the capacitance of the bus [3]. Early computer buses were literally parallel electrical wires with multiple connections, but modern computer buses can use both parallel and bit serial connections. Buses can also connect two different components at the same time through the usage of the point-to-point or multipoint technique. SoPC bus architectures have a significant effect on system speed and power dissipation. System designers, as well as the research community, have focused on the issue of exploring, evaluating, and designing personal computer (PC) communication architectures to meet the targeted design goals [4]. The replacements to buses are many, and all have been used successfully in various computers, chips, boards, and FPGAs. These replacements are no panacea, just as buses are not a cure-all for every interconnection illness. Avoiding the fixed routing and timetable of a standard bus can open up new avenues for design, and restore a bit of glamour and creativity to an otherwise mundane project [5]. The EDA design flow typically follows a path from Verilog/VHDL hardware description language [6], or schematic design entry through synthesis and place and route tools to the programming of the FPGA. The design process for this type of system involves creating a block diagram that outlines the various components of the system and their connections. This diagram can then be used to create a Verilog HDL model that can be synthesized into an FPGA implementation. The Proposed microprocessor based on a multiplexer bus system is designed, simulated, and compared against the tri-state system bus using the Verilog HDL, and implemented on the Altera Cyclone IV GX FPGA development board [7], [8]. System on-programmable-chip debug has been an apprehension from the beginning of computer era. FPGA has also taken part in this field. For example, work by Jamal et.al [9, 10] proposes better functional changes during on-chip system debug, employing FPGA edge architecture. Present-day works in this field, particularly system debugging, can be

found in [11–14]. A number of authors extend the idea to other areas such as machine learning [15, 16]. Advantage of using an FPGA for the microprocessor bus structure is that it allows for easy customization and reconfiguration. The FPGA can be programmed to support different protocols or interfaces, making it adaptable to changing requirements or new technologies. An alternative microprocessor bus structure design on an FPGA offers several benefits over traditional designs that use tri-state buses including increased performance, flexibility, and customization. With careful planning and implementation, this approach can lead to more efficient and effective systems in a variety of applications.

## 1.1 Proposed Solution:

1 This project overviews existing digital system buses which are commonly used in SOPC systems, discusses different bus architectures, and propose a new bus architecture. Also, design of bus controller that handles the transactions between data path modules.

2 The alternative for tri-state based bus structure we propose a new multiplexer based bus structure and bus controller.

3 The proposed multiplexer bus system designed, simulated, and compared with the tri-state system bus using the Verilog hardware description language, and implemented on the Altera FPGA development board, Cyclone IV GX FPGA.

4 We designed and compared both modules (Tri-state bus / Multiplexer bus) against each other for (Hardware recourses - Speed - Power Dissipation).

5 The control unit is mainly responsible for directing the various operations of the processor. We designed control unit to control data movement in the design.

## 1.2 Report Outlines

The structure of this report is divided into six chapters. A brief background Field Programmable Gate Array and why we use it, including Programming Languages and Tools requirements outlined in Chapter 2. Related works are presented in Chapter 3, while Chapter 4 covers methodology, system design, implementation, evaluation, results, and discussion. In Chapter 5, the conclusion is discussed. Finally, Chapter 6 provides for future work.

# Chapter 2 Background

## 2.1 What is an FPGA?

An FPGA (Field Programmable Gate Array) is an integrated circuit that can be configured by a customer or designer after manufacturing. The FPGA configuration is generally specified using a HDL, similar to that used for an ASIC. FPGAs contain an array of programmable logic blocks and a hierarchy of reconfigurable interconnects allowing blocks to be wired together. Logic blocks can be configured to perform complex combinational function or act as simple logic gates like AND and XOR. In most FPGs, logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory. Many FPGAs can be reprogrammed to implement different logic function, allowing flexible reconfigurable computing as performed in computer software, Xilinx produced the first commercially viable FPGA in 1995.

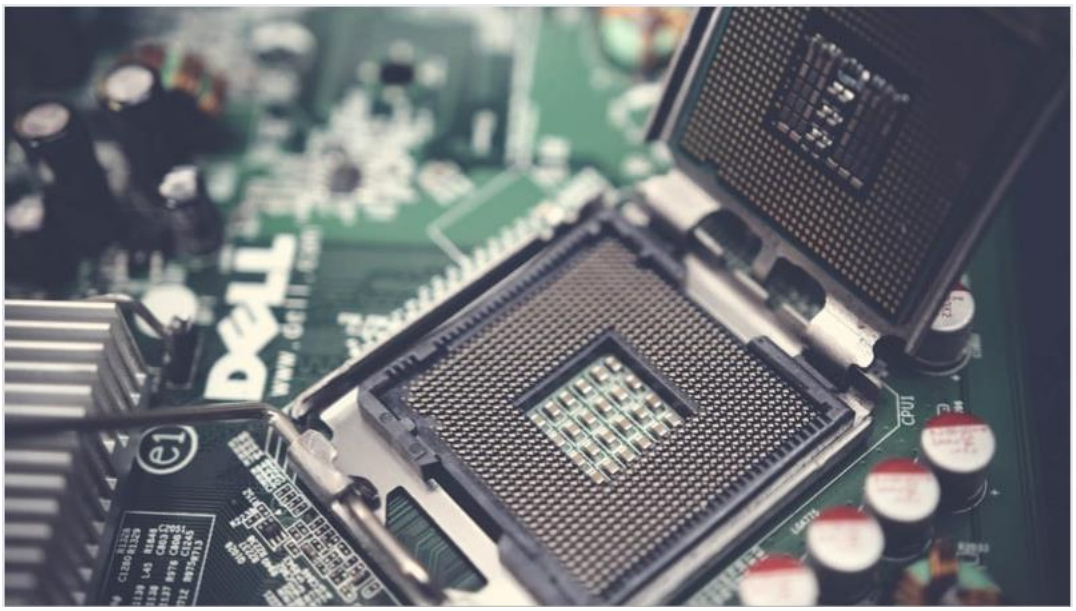



*Figure (2.1) Field Programmable Gate Array*

## 2.2 What are the advantages of using FPGA over other hardware design?

FPGAs have several advantages over other hardware. These include the ability to develop special-purpose hardware more quickly and cost-effectively than ASIC designs. FPGAs can perform many data operations simultaneously, allowing for faster and parallel processing of signal. They are also very flexible, reusable, and quicker to acquire than microcontrollers. FPGAs have a quicker time-to-market because they are not pre-designed. Additionally, FPGAs have their own energy source and do not require a host computer to run, making them more energy-efficient than CPUs or GPUs. FPGAs can easily change their functionality, which is not possibly with ASICs or discrete circuits. Another benefit of FPGAs is their parallel processing ability to perform many data operations simultaneously and their flexibility to be reprogrammed to perform different tasks.

## 2.3 What are the main difference between FPGA and ASICs?

ASICs are not reprogrammable and require a new design for each new application, while FPGAs can be reprogrammed to perform different tasks. FPGAs are more flexible, reusable, and quicker to acquire than ASICs. FPGAs have a much higher unit cost compared to ASICs, which means that if you are looking to use them for high volume mass production, ASICs are more cost-effective. In summary, ASICs are designed for a specific application and offer higher performance and power efficiency, while FPGAs are more flexible and can be reprogrammed to perform different tasks.

## 2.4 Programming Languages

There are two most popular HDLs today: so one is Verilog HDL, the other is VHDL. In this project, we will be using the language Verilog HDL, which used by designers to specify behavior, functionality, or structure of given hardware, or specified digital circuit.

## 2.5 Tools:

### 2.5.1 The Electronic Design Automation (EDA)

EDA is a category of software tools used by electronic designers to design, analyze, and simulate electronic systems. It encompasses a wide range of tasks involved in the design process, including.

Synthesis: EDA tools can automatically generate optimized gate-level or register transfer level (RTL) designs from high-level descriptions like HDLs. This process is known as synthesis and helps in improving design efficiency.

Simulation and Analysis: EDA tools allow designers to simulate the behavior of electronic circuits before fabrication. This helps in identifying potential issues or optimizing circuit performance. Different types of simulations include analog, digital, mixed-signal, and electromagnetic simulations.

### 2.5.2 ModelSim Intel-Altera

The ModelSim-Altera software is Altera specific and supports behavioral and gate level timing simulations and either VHDL or Verilog HDL simulations and test benches for Altera PLDs [18].

### 2.5.3 Quartus II Intel-Altera

The Quartus II development software provides a complete design environment for SoPC design. Regardless of whether you use a personal computer or a Linux workstation, the Quartus II software ensures easy design entry, fast processing, and straightforward device programming. Quartus II was used to program the FPGA board [19].

### 2.5.4 Altera Cyclone IV GX FPGA Development Board

The Altera Cyclone IV GX FPGA development board is a hardware platform designed for developing and prototyping digital logic circuits using Field-Programmable Gate Array technology. It is specifically based on the Cyclone IV GX FPGA from Intel (formerly Altera), it is provides a platform for engineers, researchers, and hobbyists to experiment with FPGA-based designs.

The board typically includes the Cyclone IV GX FPGA chip, various input/output interfaces (such as USB, Ethernet, HDMI), memory components (such as DDR3 SDRAM), and programmable logic elements. It also offers features like switches, LEDs, and displays for user interaction and debugging. With this development board, users can write their own digital logic designs using hardware description languages like VHDL or Verilog. They can then program the FPGA to implement these designs and test their functionality in real-time. The board often comes with software tools and libraries that facilitate design entry, synthesis, simulation, and programming of the FPGA.

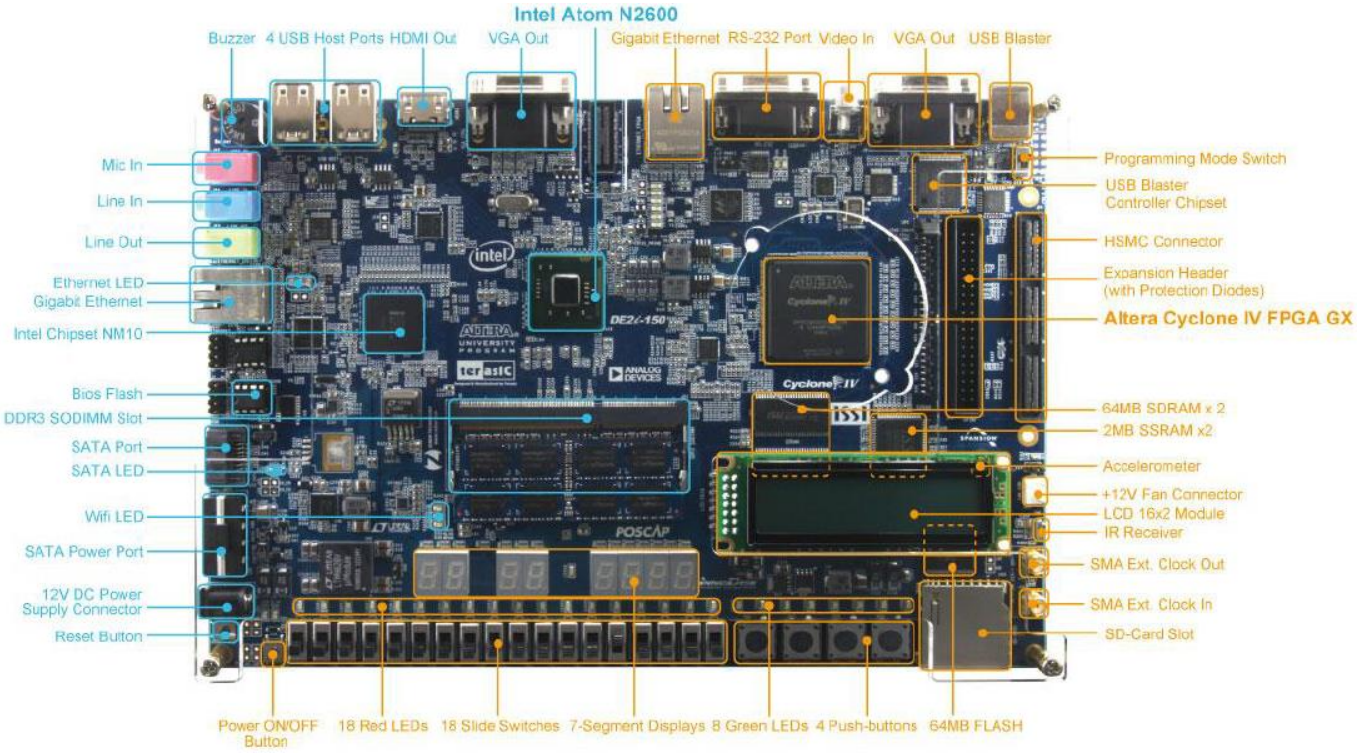



*Figure (2.2) Altera Cyclone IV GX FPGA development board.*

# Chapter 3 Related Work

There have been several related works on alternative microprocessor bus structure designs implemented on FPGA. Some of these works include:

1. "A High-Performance Microprocessor Bus Architecture for FPGA-Based Systems" by Chen et al. This work proposes a novel microprocessor bus architecture that aims to improve the performance of FPGA-based systems. The proposed architecture utilizes a hierarchical bus structure with multiple levels of buses, allowing for efficient data transfer and reduced latency.
2. "Design and Implementation of a Scalable Microprocessor Bus for Reconfigurable Computing" by Zhang et al. This work presents a scalable microprocessor bus design that can be used in reconfigurable computing systems. The proposed bus architecture supports multiple processors and allows for dynamic reconfiguration of the system, enabling efficient utilization of FPGA resources.
3. "An Efficient Microprocessor Bus Architecture for FPGA-Based Embedded Systems" by Li et al. This work proposes an efficient microprocessor bus architecture specifically designed for FPGA-based embedded systems. The proposed architecture utilizes a segmented bus structure with separate data and control buses, enabling parallel data transfer and reducing the overall latency.
4. "A Low-Power Microprocessor Bus Design for FPGA-Based Systems" by Wang et al. This work focuses on designing a low-power microprocessor bus architecture for FPGA-based systems. The proposed design incorporates power-saving techniques such as clock gating and voltage scaling to reduce power consumption while maintaining performance.
5. "A Fault-Tolerant Microprocessor Bus Architecture for Reliable FPGA-Based Systems" by Liu et al. This work presents a fault-tolerant microprocessor bus architecture designed to improve the reliability of FPGA-based systems. The proposed architecture includes redundancy mechanisms and error detection/correction techniques to ensure reliable data transfer in the presence of faults.

These related works, each addressing different aspects such as performance, scalability, power consumption, reliability, or specific application requirements in FPGA-based systems.

# Chapter 4 Methodology

We designed and implemented two bus modules, one using Tristate bus and the other using Multiplexer bus, to compare their performance and functionality, and to conduct a comparative analysis on their effectiveness and efficiency.

## 4.1 Tristate Bus Module

The top-level module for Tri-state bus and four lower-level modules were used to implement the design, the first module of the lower level for 8_bit register, the second module for 8_bit tri-state bus, the third module for arithmetic logic unit (ALU) and the fourth module for MUX 2 to 1 as shown in figure 4.5. Each system contains four register that has three inputs clk, ena and x, and one output q. ALU module has two inputs and select lines to control the operations such as, (addition, subtraction, AND, and shift logical bit left), and has one output. As shown in table 1. We have 4 to 1 multiplexer that has 4 inputs and two select line that implemented to control the output data, and 2 to 1 multiplexer with two inputs, one select line that implemented to control the output data. The top-level module contains six inputs select, operation, move, write, data, enable, clock and has six outputs R0out, R1out, R2out, R3out, Cout and out. Registers are connected with tri state, and 4 to 1 multiplexer, then data is loaded into registers, using move and write input signals we can specify the registers that used to enter data, then the data moved from one register to other register. Registers are associated with two 2to1 multiplexers and connected to ALU. The data path module is consisting of five sub-modules. The system contains four 8-bit registers, register 0 to register 3, figure 4.14 displays how these registers are connected using tri-state drivers to implement the bus structure. The data outputs q of each register is connected to tri-state drivers. When selected by their enable signals, the driver places the contents of the consistent register onto the bus wires. If the enable input is set to 1, then the contents of the register will be changed on the next positive edge of the clock. The enable input on each register is registered ena, which positions for enable. The signal that controls the ena input for registers is designated as [3:0] Write, while the signal that controls the associated tri-state driver and multiplexer is called [3:0] Move. These signals are created by the control unit module. In addition to four registers, there is other module block that linked to the bus. The circuit diagram, figure 4.11 , show how 8-bits of data from an external source that is located on the same bus, using the

control input signal that is created by control unit module called Enable. It is important to ensure that only one circuit block tries to place data onto the bus wires at any assumed time.

### 4.1.1 Register

- Register are used to quickly accept, store, and transfer data and instruction directed by CPU.
- Fast Temporary memory location for CPU.
- It hold also a storage address.
- They exist in microprocessor.
- Optimization of processing time.

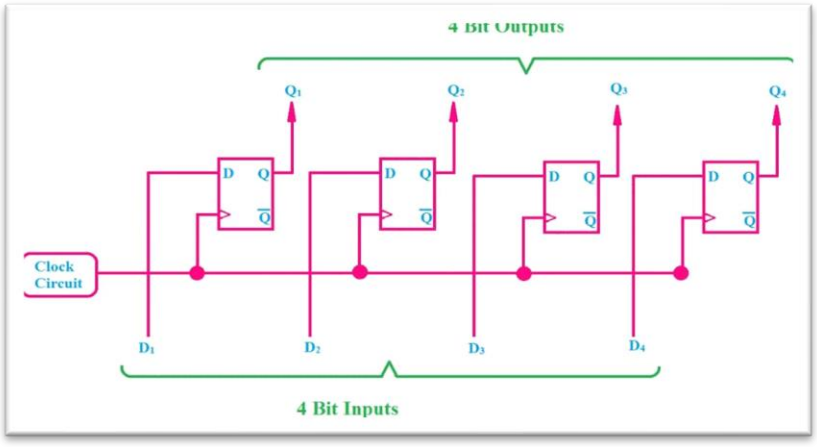



*Figure (4.1) D flip-flops.*

### 4.1.2 Tri-state

Digital buffers and Tri-state buffers can provide current amplification in a digital circuit to drive output loads.

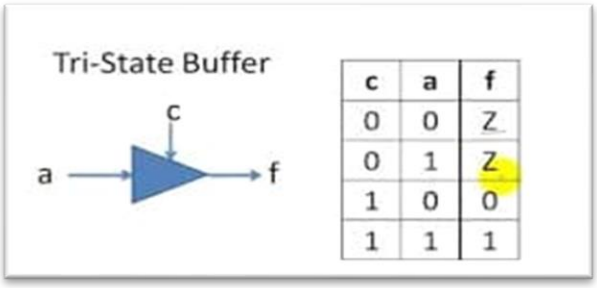



*Figure (4.2) Tri-state buffers.*

### 4.1.3 Arithmetic Logic Unit

In computing, an arithmetic logic unit (ALU) is a combinational digital circuit that performs arithmetic and bitwise operations on integer binary numbers.

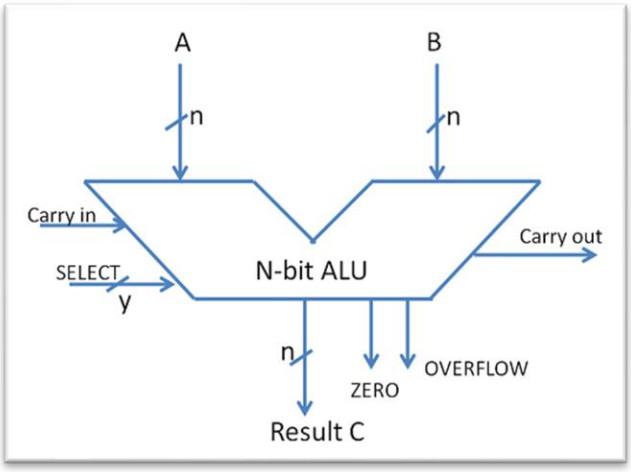



*Figure (4.3) Arithmetic Logic Unit.*

### 4.1.4 Multiplexer

In electronics, a multiplexer (or mux; spelled sometimes as multiplexor), Also known as a data selector, is a device that selects between several Analog or digital input signals and forwards the selected input to a single output line.

#### 4.1.4.1 MUX 2 to1

The selection is directed by a separate set of digital inputs known as select lines,A multiplexer of inputs has select lines, which are used to select which input line to send to the output.

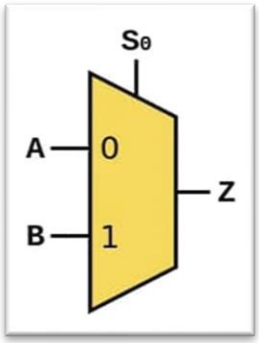



*Figure (4.4) MUX 2 to 1.*

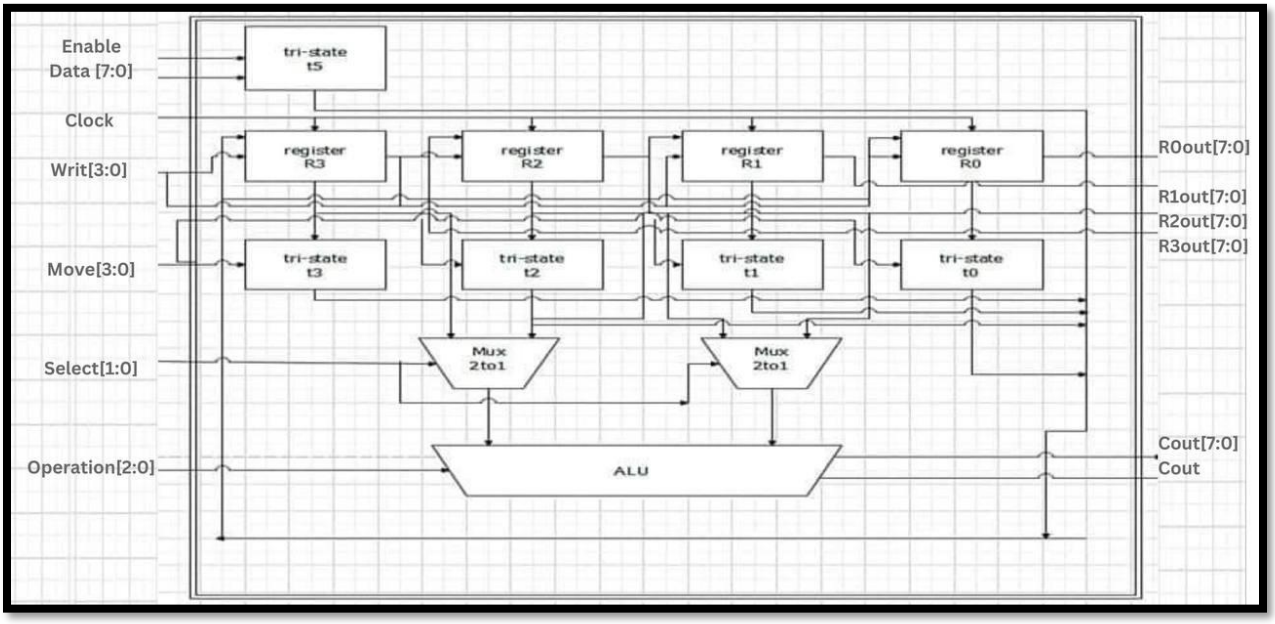



*Figure (4.5) Tri-state top-level schematic.*

## 4.2 Multiplexer Bus Module

The second-way using multiplexed-bus. The top-level module for multiplexed-bus and four lower-level modules were used to implement the design, the first module of lower level for 8_bit register, the second module for 8_bit MUX 4 to 1, the third module for ALU and the fourth module for MUX 2 to 1 as shown in figure 4.10. The system contains four register that has three inputs clk, ena and x, and one output q. ALU module has two inputs and select lines to control the operations such as, (addition, subtraction, AND, and shift logical bit left), and has one output. As shown in table 1. We have 4 to 1 multiplexer that has 4 inputs

and two select line that implemented to control the output data, and 2 to 1 multiplexer with two inputs, one select line that implemented to control the output data. The top-level module contains six inputs select, op, move, write, data, enable, clock and has six outputs R0out, R1out, R2out, R3out, Cout and out. Registers are connected with tri state, and 4 to 1 multiplexer, then data is loaded into registers, using move and write input signals we can specify the registers that used to enter data, then the data moved from one register to other register. Registers are associated with two 2 to 1 multiplexers and connected to ALU. The multiplexer design is containing of two main modules data path module and control unit module. The data path module is consisting of five sub-modules. The system contains four 8-bit registers, register 0 to register 3, figure 4.17 displays how these registers are connected using multiplexer to implement the bus structure. The data outputs q of each register is connected to multiplexer. When selected by their enable signals, the driver places the contents of the consistent register onto the bus wires. If the enable input is set to 1, then the contents of the register will be changed on the next positive edge of the clock. The enable input on each register is registered ena, which positions for enable. The signal that controls the ena input for registers is designated as [3:0] Write, while the signal that controls the associated multiplexer is called [1:0] Move. These signals are created by the control unit module. In addition to four registers, there is other module block that linked to the bus. The circuit diagram, figure 4.11 shows how 8-bits of data from an external source that is located on the same bus, using the control input signal that is created by control unit module called Enable. It is important to ensure that only one circuit block tries to place data onto the bus wires at any assumed time.

### 4.2.1 Register

- Register are used to quickly accept, store, and transfer data and instruction directed by CPU.
- Fast Temporary memory location for CPU.
- It hold also a storage address.
- They exist in microprocessor.
- Optimization of processing time.

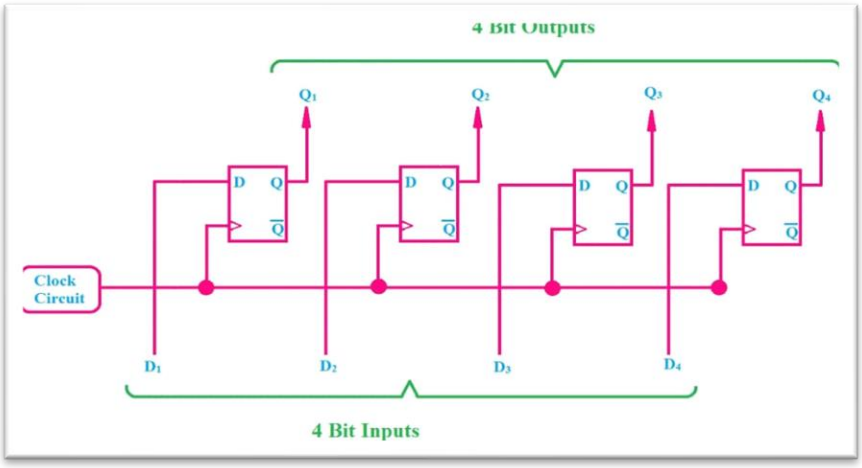



*Figure (4.6). D flip-flops.*

### 4.2.2 Arithmetic Logic Unit

In computing, an arithmetic logic unit (ALU) is a combinational digital circuit that performs arithmetic and bitwise operations on integer binary numbers.

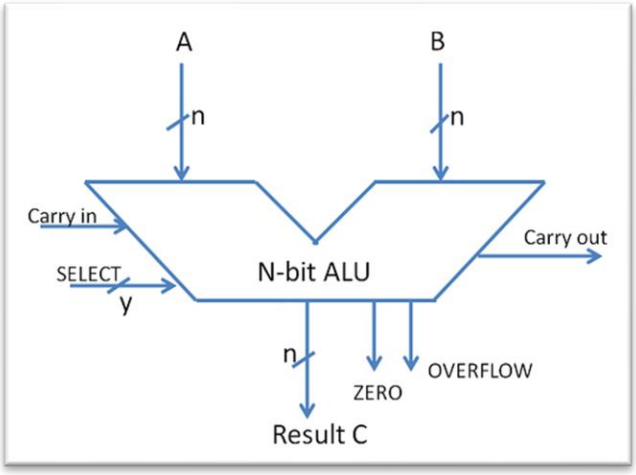



*Figure (4.7) Arithmetic Logic Unit.*

### 4.2.3 Multiplexer

In electronics, a multiplexer (or mux; spelled sometimes as multiplexor), Also known as a data selector, is a device that selects between several

Analog or digital input signals and forwards the selected input to a single output line.

### 4.2.3.1 MUX 2 to1

The selection is directed by a separate set of digital inputs known as select lines. A multiplexer of inputs has select lines, which are used to select which input line to send to the output.

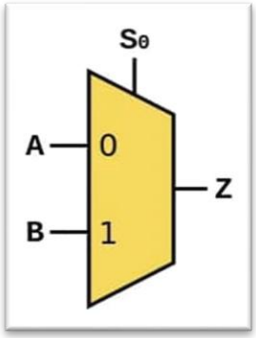



*Figure (4.8) MUX 2 to 1.*

### 4.2.3.2 MUX 4 to1

A 4-bit multiplexer would have N input each of 4 bits where each input can be transferred to the output by the use of select signal.

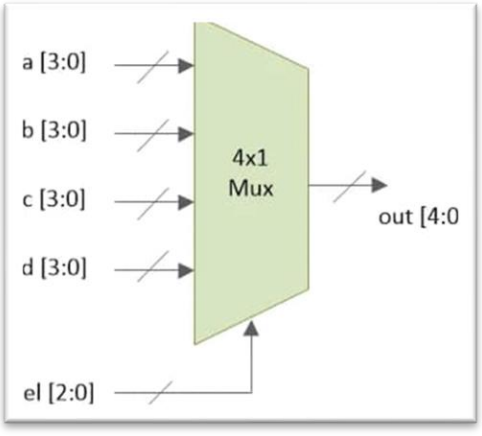



*Figure (4.9) Arithmetic Logic Unit.*

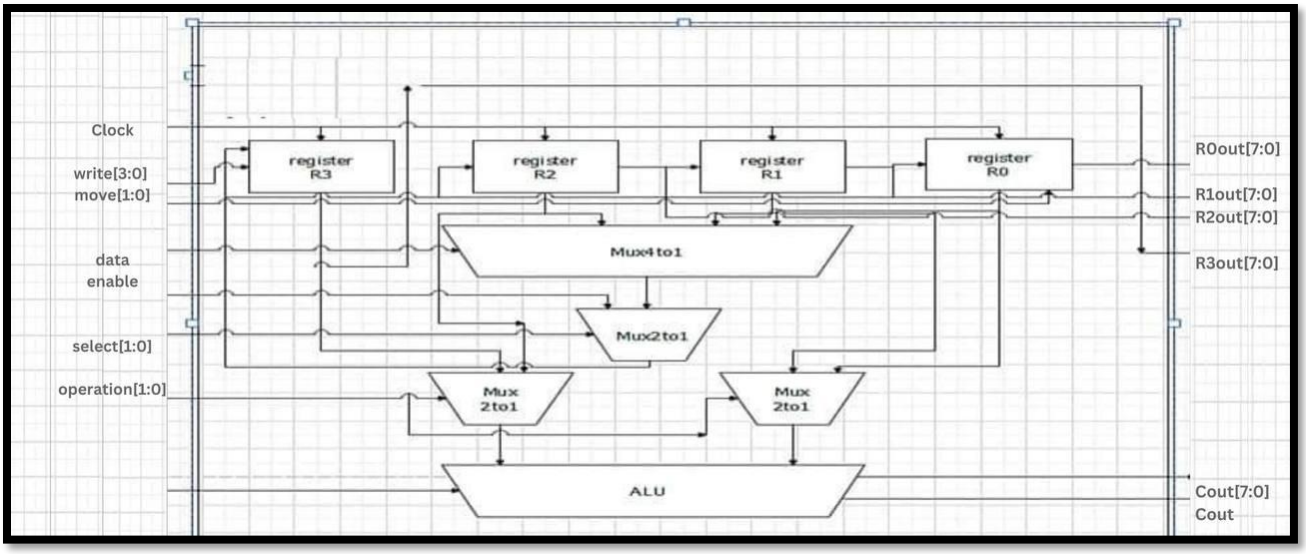



*Figure (4.10) MUX top-level schematic.*

## 4.3 Control Unit

A control unit is a component of a computer's central processing unit (CPU), in a computer the control unit often steps through the instruction cycle successively. This consists of fetching the instruction, fetching the operands, decoding the instruction, coordinating input/output operations, executing the instruction, and then writing the results back to memory. When the next instruction is placed in the control unit, it changes the behaviour of the control unit to complete the instruction correctly. So, the bits of the instruction directly control the control unit, which in turn controls the computer.

### 4.3.1 Control Design

The control of the MUX project it designed to enter data and send it to the data path to be uploaded into the register, and shifting between register, and the implementation of arithmetic logical operations on the data. The control is connected to the data path to control the input according to the control state. The control unit also produces the signals [3:0] Write, which determine when data is loaded into each register. In general, the control unit perform a number of functions, such as loading resisters with data and transferring the data stored in one register into another register. The control

circuit is synchronized by a clock input, which is the equal clock signal that controls four registers.

The top-level module contains two lower-level modules, the first module of lower level is data path of MUX, and the second module of lower level is control as shown in figure (4.11). The module of control contains three input clk, ena, and reset, and six output write, move, data, enable, select, and operation.

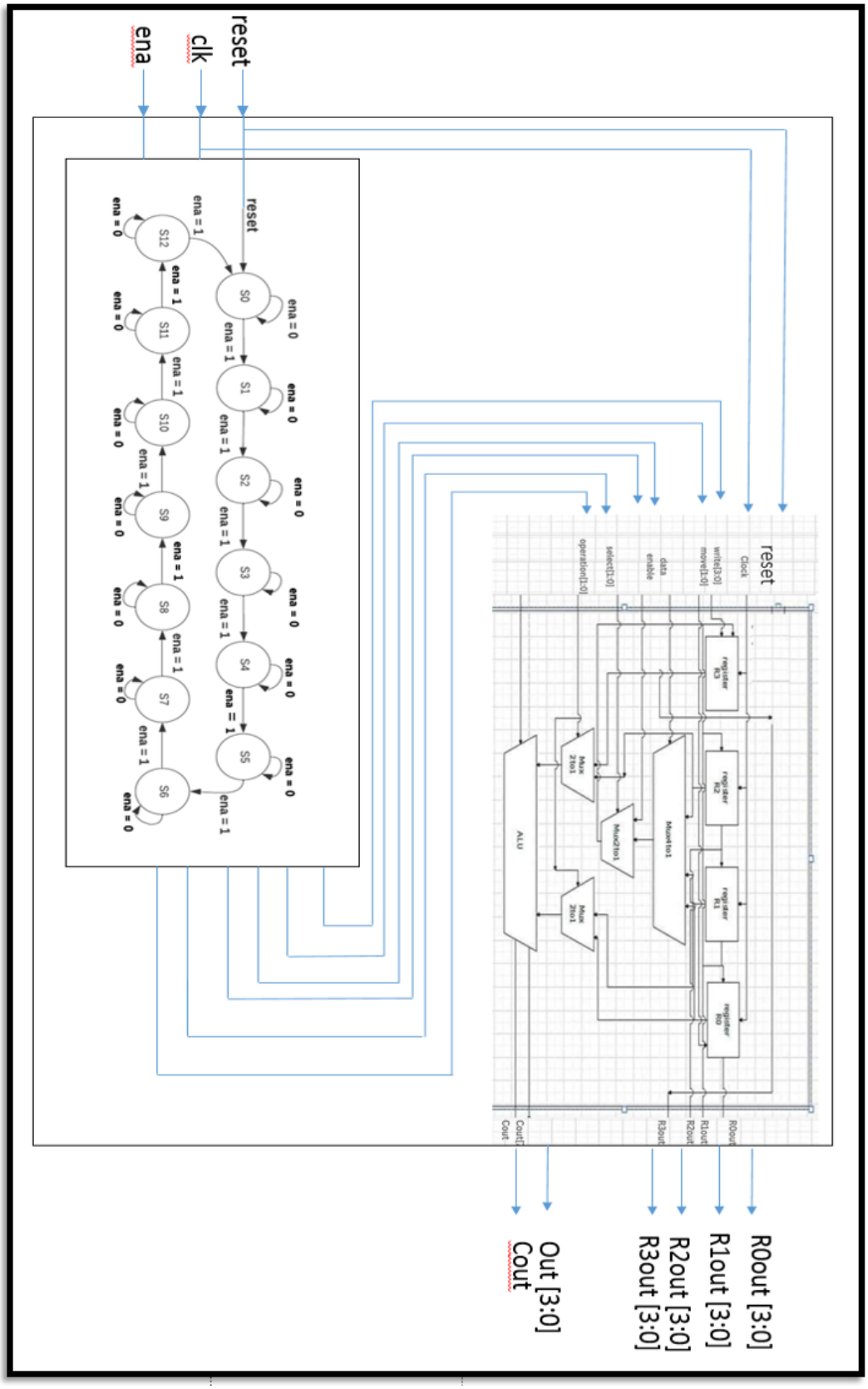



*Figure (4.11) Top module MUX*

We divide module of data path to five module the first module for 4_bit register, the second module for 4_bit MUX 4 to 1, the third module for MUX 2 to 1 and the fourth module for ALU. The system of control module contains has three inputs are clock, ena and reset, and six outputs are data, write, move, enable, select, and operation.

### 4.3.2 State Machine of Control.

The control designed using a state machine approach, which means that it operates based on a set of predefined states. In this case, 13 different states were created to represent the various possible conditions or modes of the control. These states define the behavior and functionality of the control in under different inputs or conditions.

state0: Reset all the register.

state1: Enter data (9) and send it to the data path, it is loaded into the Register (R3).

state2: Enter data (11) and send it to the data path, it is loaded into the Register (R2).

state3: Enter data (13) and send it to the data path, it is loaded into the Register (R1).

state4: Enter data (0) and send it to the data path, it is loaded into the Register (R0).

state5: Shift between the registers (move the data of Register 1 to Register 0).

state6: Shift between the registers (move the data of Register 2 to Register 1).

state7: Shift between the registers (move the data of Register 3 to Register 2).

state8: Shift between the registers (move the data of Register 0 to Register 3).

state9: Execution operation on register (R3 and R1) the operation add.

state10: Execution calculation on register (R3 and R1) the operation subtraction.

state11: Execution calculation on register (R2 and R1) the operation shift left of R2.

state12: Execution calculation on register (R2 and R0) the operation AND.

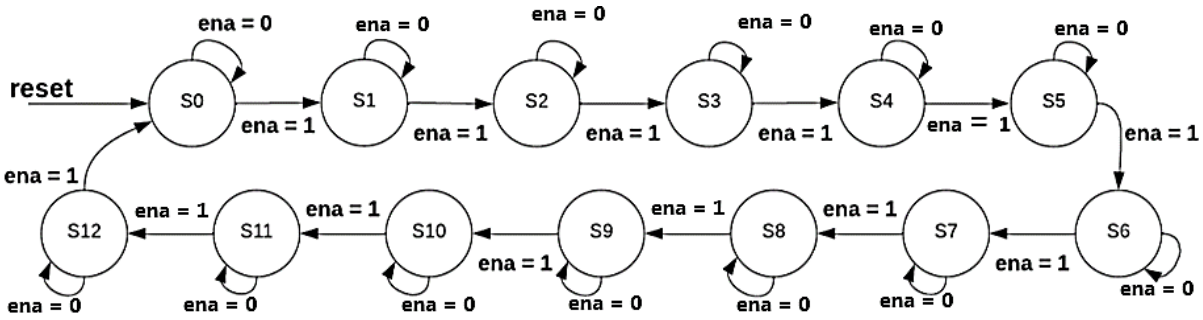



*Figure (4.12) Finite State Machine FSM diagram.*

This table is clarified state according to enable, if 1 will move to next state and if 0 waits in the same state.

*Table (4-1) Truth table of state machine for control unit*

State Table

| | Source State | Destination State | Condition |
|---|---|---|---|
| 1 | S0 | S1 | (ena) |
| 2 | S0 | S0 | (!ena) |
| 3 | S1 | S1 | (!ena) |
| 4 | S1 | S2 | (ena) |
| 5 | S2 | S2 | (!ena) |
| 6 | S2 | S3 | (ena) |
| 7 | S3 | S3 | (!ena) |
| 8 | S3 | S4 | (ena) |
| 9 | S4 | S4 | (!ena) |
| 10 | S4 | S5 | (ena) |
| 11 | S5 | S5 | (!ena) |
| 12 | S5 | S6 | (ena) |
| 13 | S6 | S6 | (!ena) |
| 14 | S6 | S7 | (ena) |
| 15 | S7 | S7 | (!ena) |
| 16 | S7 | S8 | (ena) |
| 17 | S8 | S8 | (!ena) |
| 18 | S8 | S9 | (ena) |
| 19 | S9 | S9 | (!ena) |
| 20 | S9 | S10 | (ena) |
| 21 | S10 | S11 | (ena) |
| 22 | S10 | S10 | (!ena) |
| 23 | S11 | S11 | (!ena) |
| 24 | S11 | S12 | (ena) |
| 25 | S12 | S0 | (ena) |
| 26 | S12 | S12 | (!ena) |

## 4.4 Results and Discussion

These results illustrate the work of ALU using four operations and output results shown in figure (4.13).

*Table (4-2) Truth table of ALU.*

| Select | Instruction | Operation |
|---|---|---|
| 00 | ADD | Out = A+B (Cout is carry) |
| 01 | SUB | Out =A-B |
| 10 | SHL | A<<1 |
| 11 | AND | A & B |

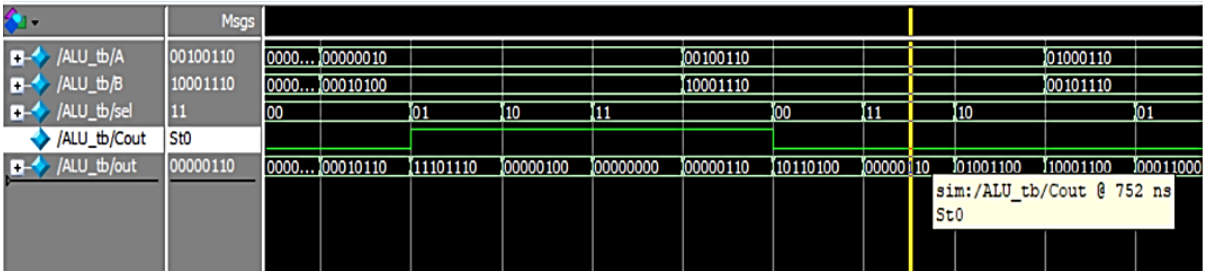



*Figure (4.13) ALU operation.*

After designing and simulating the multiplexer bus submodules individually, the system was instantiated, simulated, and validated as a top-level module. The system was then compared to a tristate bus system to evaluate the capabilities, for speed, and power consumption of the proposed multiplexer bus system and the tristate bus system specially designed and implemented for this purpose. In the test-bench, the module reads four values of "data" respectively 55, 77, 99, 00, it is stored in registers and transferred via bus to different register using "move". In figure (4.14) data is loaded to the registers, and in figure (4.15) date is shifted right between registers.

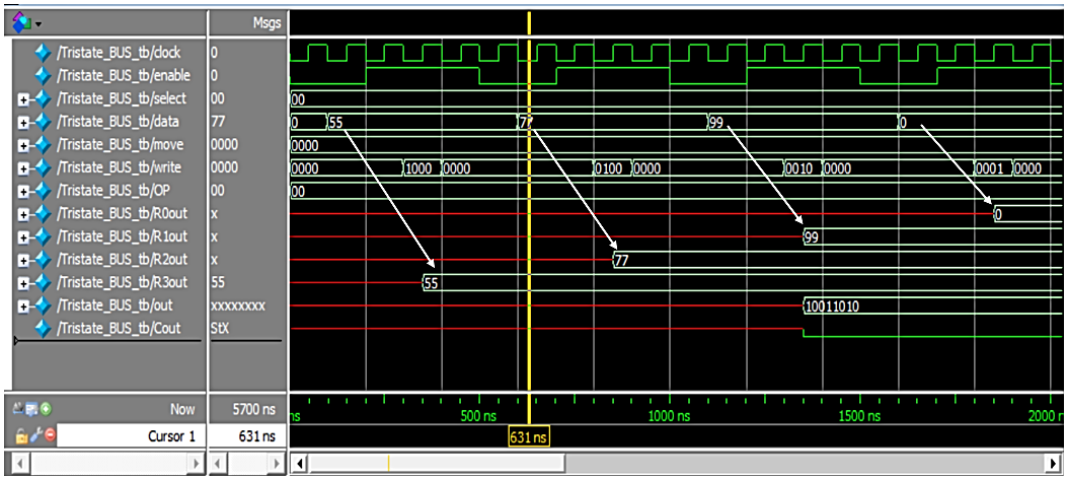



*Figure (4.14) Simulation of the tri-state.*

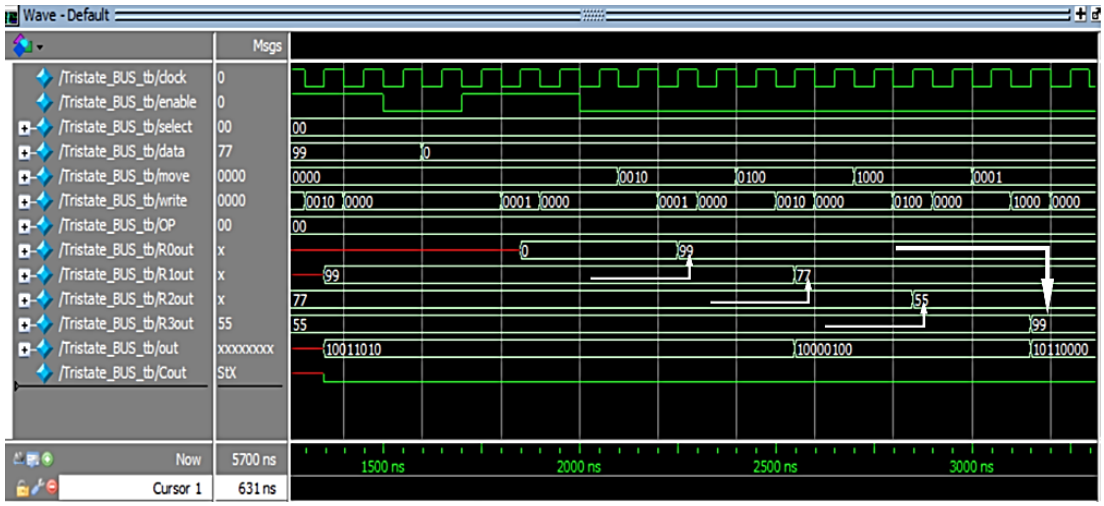



*Figure (4.15) The transfer via bus to different registers.*

As shown in table (4-3), after shifting operation of registers, the register that contains the instruction is chosen by "select", and instruction selection is depends on the operation. Then the instruction is executed, the result of the operation is written into Out, and when the remainder is obtained according to some instruction, it is written into Cout. The simulated waveforms of the tristate bus system register show in figure (4.16).

When "select"' is equals '10', the registers R2, R1 are used to select the instruction according to the opcode, opcode = 00, so the instruction is ADD. Since the value of R1 is

equal 01001101 and the value of R2 is equal 00110111, the result of the addition process is equal to 10000100, the result is kept in out and in this case there is no remainder, so the value of Count equals zero.

*Table (4-3) Select resisters and ALU operation.*

| Select | Resisters | Operation | Instruction | Output |
|---|---|---|---|---|
| **00** | R3,R1 | 001 | ADD | Out = R3+R1, (Cout is carry) |
| **00** | R3,R1 | 010 | SUB | Out = R3-R0, (Cout is carry) |
| **00** | R3,R1 | 011 | SHL | Out = R2<<1 |
| **00** | R3,R1 | 100 | AND | Out = R2 & R0 |
| **01** | R3,R0 | 001 | ADD | Out = R3+R1, (Cout is carry) |
| **01** | R3,R0 | 010 | SUB | Out = R3-R0, (Cout is carry) |
| **01** | R3,R0 | 011 | SHL | Out = R2<<1 |
| **01** | R3,R0 | 100 | AND | Out =R2 & R0 |
| **10** | R2,R1 | 001 | ADD | Out = R3+R1, (Cout is carry) |
| **10** | R2,R1 | 010 | SUB | Out = R3-R0, (Cout is carry) |
| **10** | R2,R1 | 011 | SHL | Out = R2<<1 |
| **10** | R2,R1 | 100 | AND | Out =R2 & R0 |
| **11** | R2,R0 | 001 | ADD | Out = R3+R1, (Cout is carry) |
| **11** | R2,R0 | 010 | SUB | Out = R3-R0, (Cout is carry) |
| **11** | R2,R0 | 011 | SHL | Out = R2<<1 |
| **11** | R2,R0 | 100 | AND | Out = R2 & R0 |

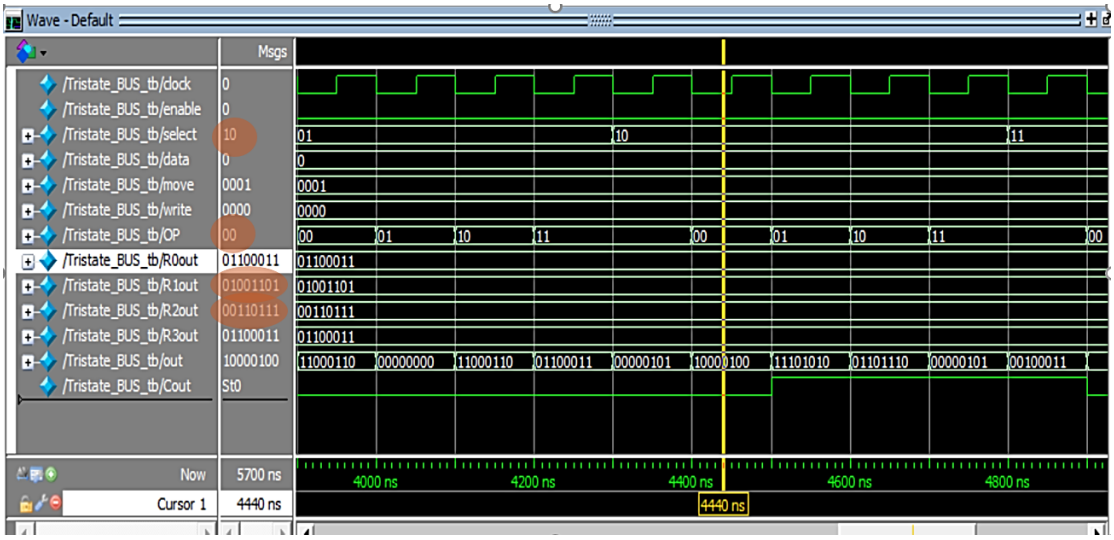



*Figure (4.16) ALU "ADD" operation.*

The resulting simulation waveform of post synthesis models as shown in figures (4.17),(4.18), (4.19) demonstrates that both systems use the same dataset, except that the

first module uses the tristate bus and the second module uses the multiplexer bus to interconnect the datapath registers. Both systems show  that they work identical to each others.

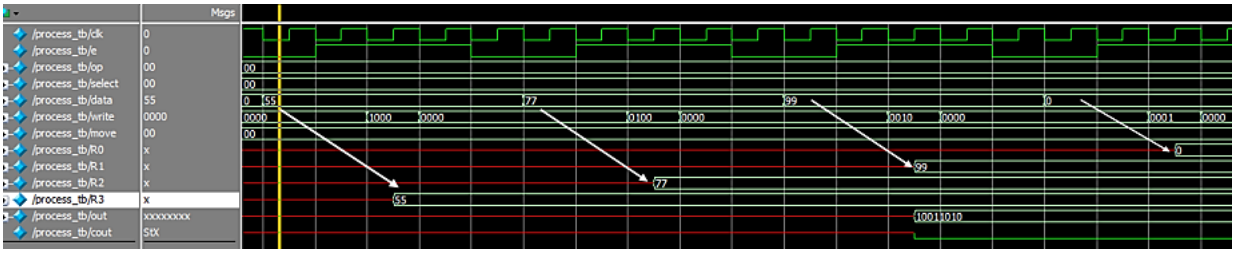



*Figure (4.17) Simulation of the multiplexer.*

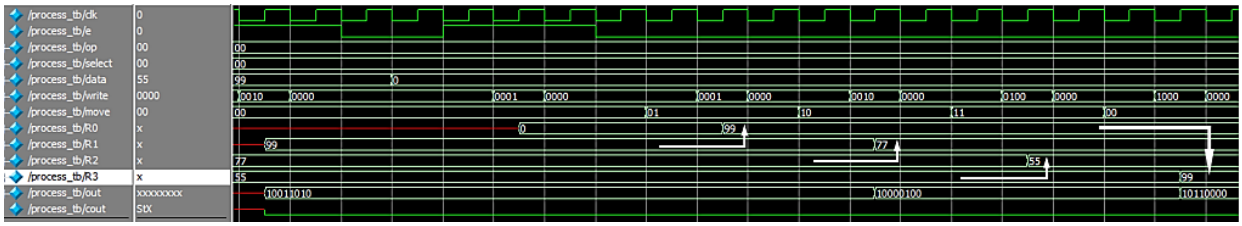



*Figure (4.18) The transfer via bus to different registers.*

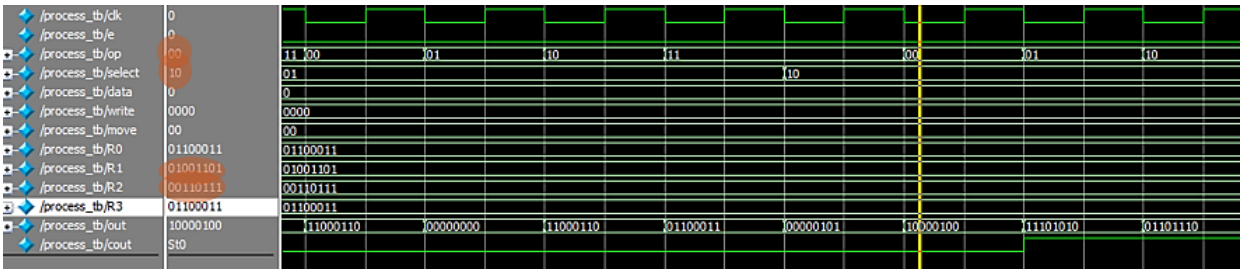



*Figure (4.19) ALU "ADD" operation.*

### 4.4.1 Results and Discussion of Control

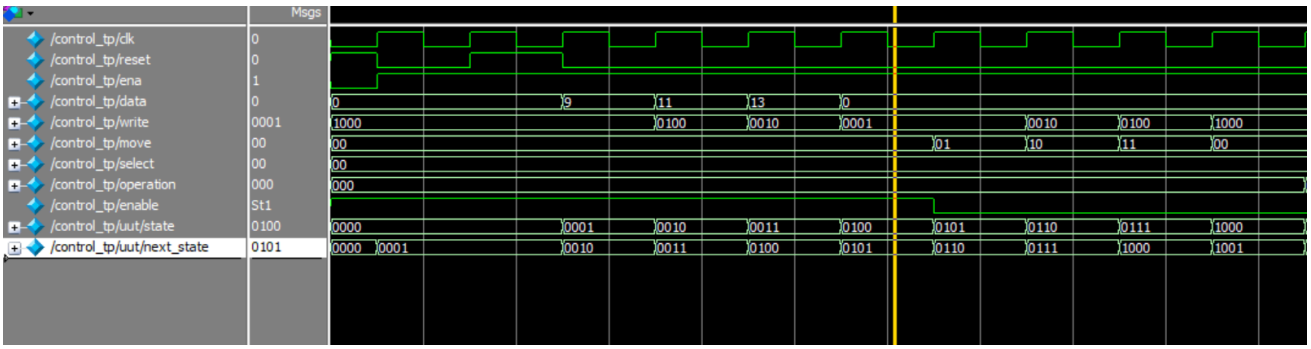



*Figure (4.20) Simulation of control "enter data in registers and shifting between registers".*

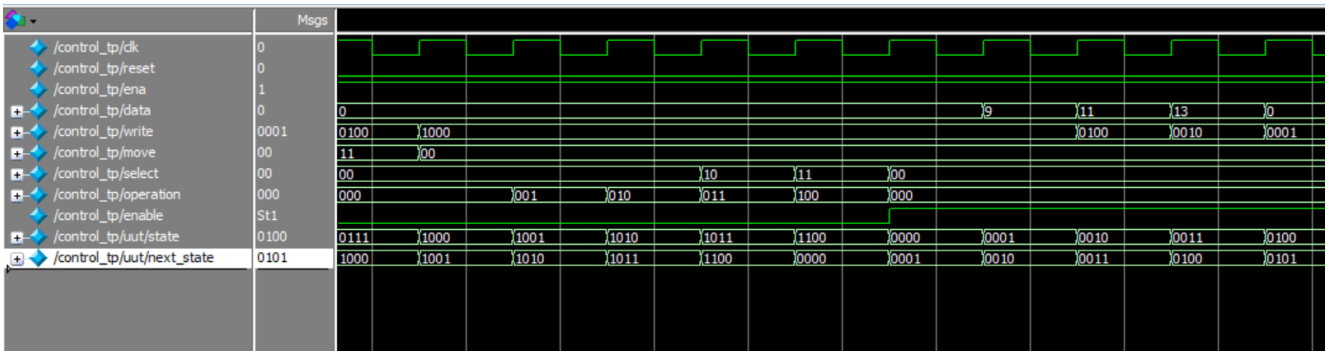



*Figure (4.21) Simulation of control "select register and executed the operation".*

### 4.4.2 Results and Discussion of Top Module MUX

After designing and simulating the multiplexer bus submodules individually, the system was instantiated, simulated, and validated as a top-level module. In the control, the module reads four values of "data" respectively 9, 11, 13, 0, it is stored in registers and transferred via bus to different register using "move".(data is loaded to the registers), and date is shifted right between registers as shown in figure (4.22).

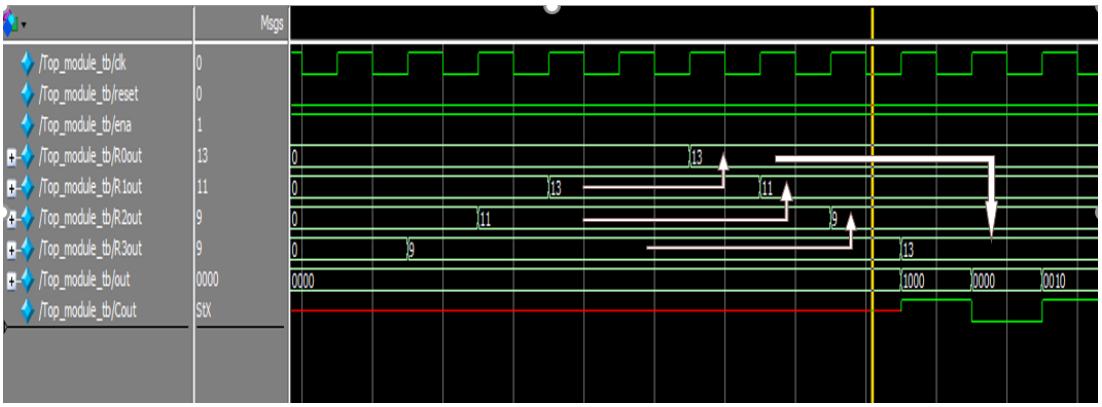



*Figure (4.22) Simulation of the top module.*

As shown in table (4-4), after shifting operation of registers, the register that contains the instruction is chosen by "select", and instruction selection is depends on the operation. Then the instruction is executed, the result of the operation is written into Out, and when the remainder is obtained according to some instruction, it is written into Cout. The simulated waveforms of the top module MUX bus system register show in figure (4.23).

*Table (4-4) Select resisters and ALU operation of top module MUX bus*

| Select | Resisters | Operation | Instruction | Output |
|---|---|---|---|---|
| **00** | R3,R1 | 001 | ADD | Out = R3+R1, (Cout is carry) |
| **01** | R3,R0 | 010 | SUB | Out = R3-R0, (Cout is carry) |
| **10** | R2,R1 | 011 | SHL | R2<<1 |
| **11** | R2,R0 | 001 | AND | Out = R2 & R0 |

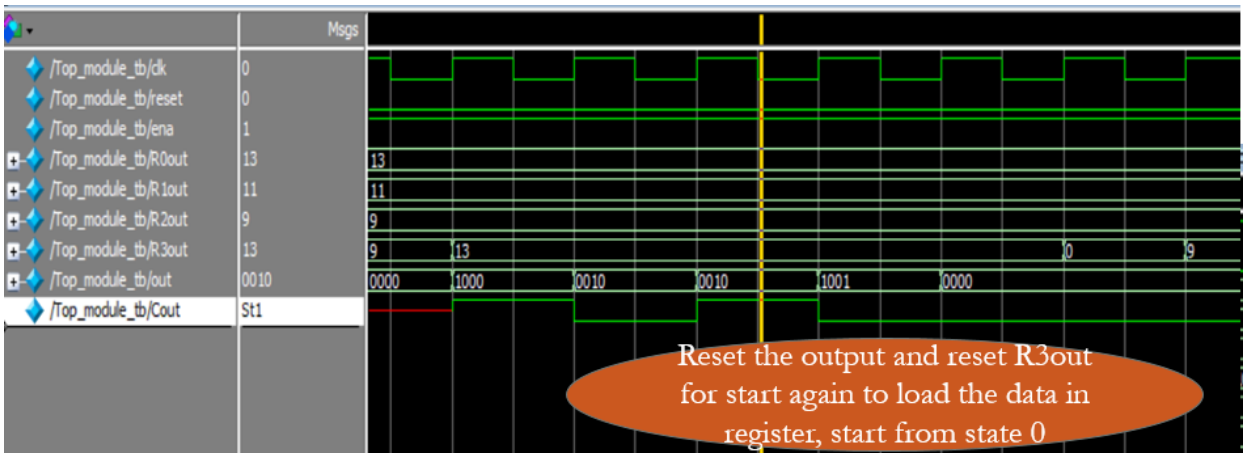



*Figure (4.23) Simulation of the top module show executed the operations ADD, SUB, SHL, AND.*

### 4.4.3 Register Transfer Level

The proposed multiplexer bus module requires less FPGA chip resources to implement the bus system because it has fewer register transfer levels than the tristate bus module. Figure(4.24) shows the register transfer level (RTL) of the tristate bus module, and figure (4.25) shows the RTL of the multiplexer bus module. And figure(4.26) shows the RTL of data path MUX 4-bit bus.

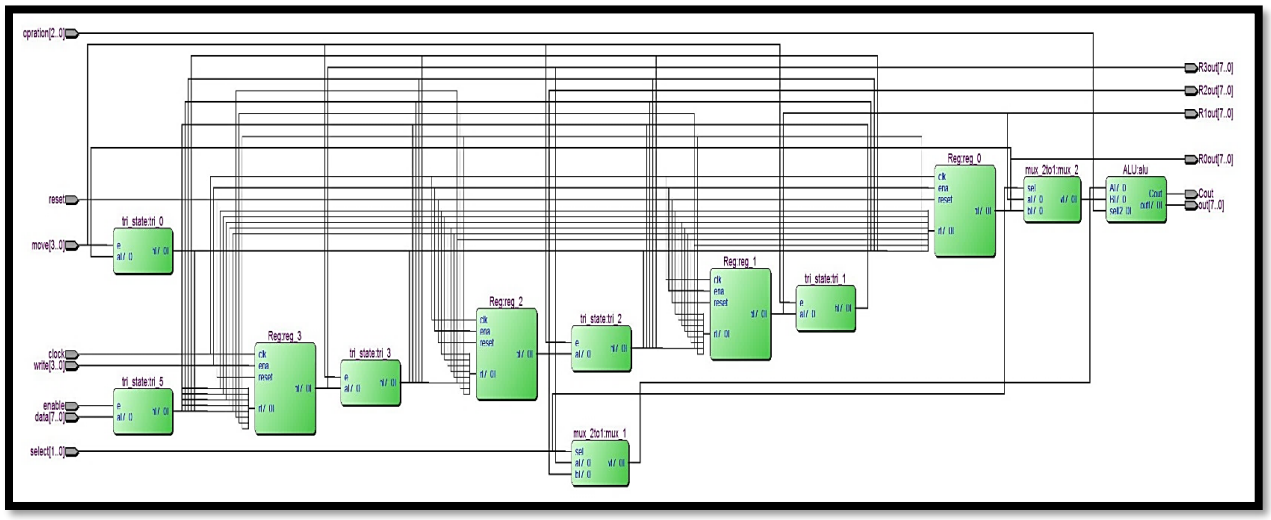


*Figure (4.24) Register transfer level schematic for data path Tristate 8_bit.*

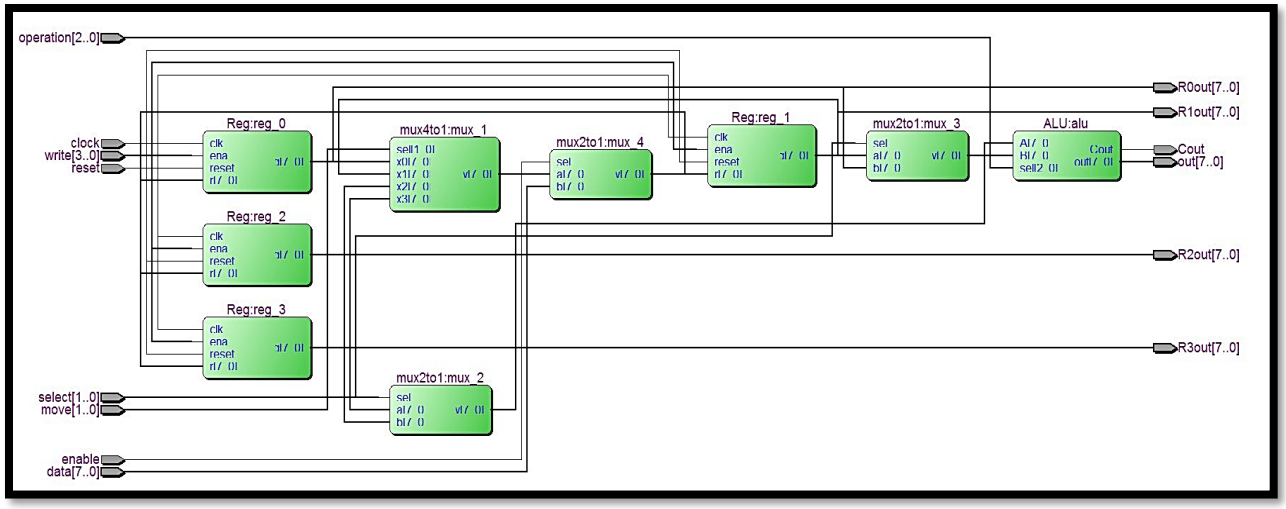



*Figure (4.25) Register transfer level schematic for data path MUX 8_bit.*

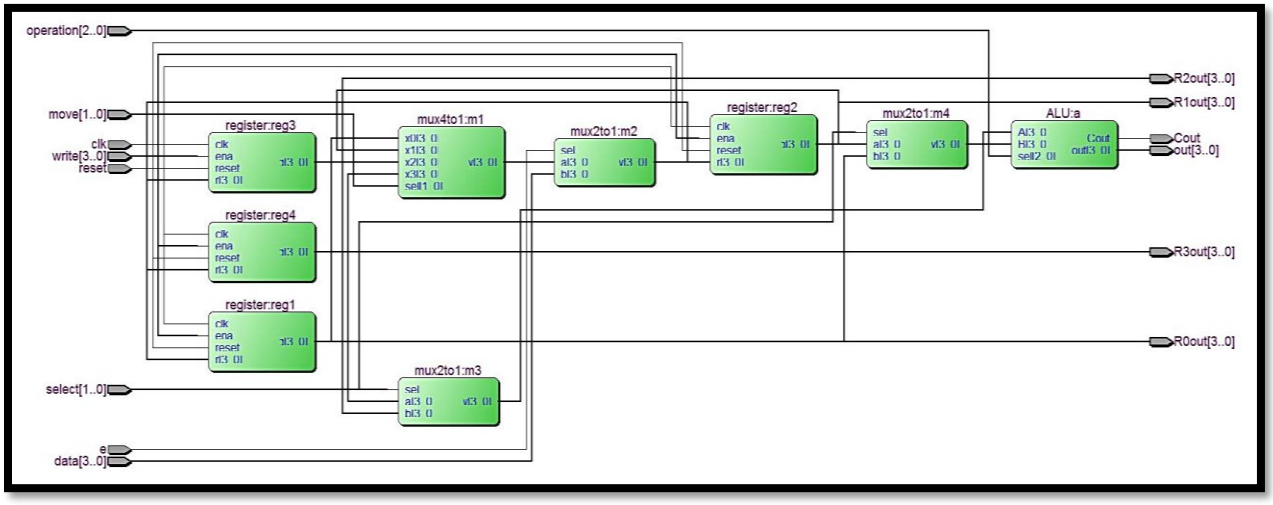



*Figure (4.26) Register transfer level schematic for data path MUX 4-bit.*

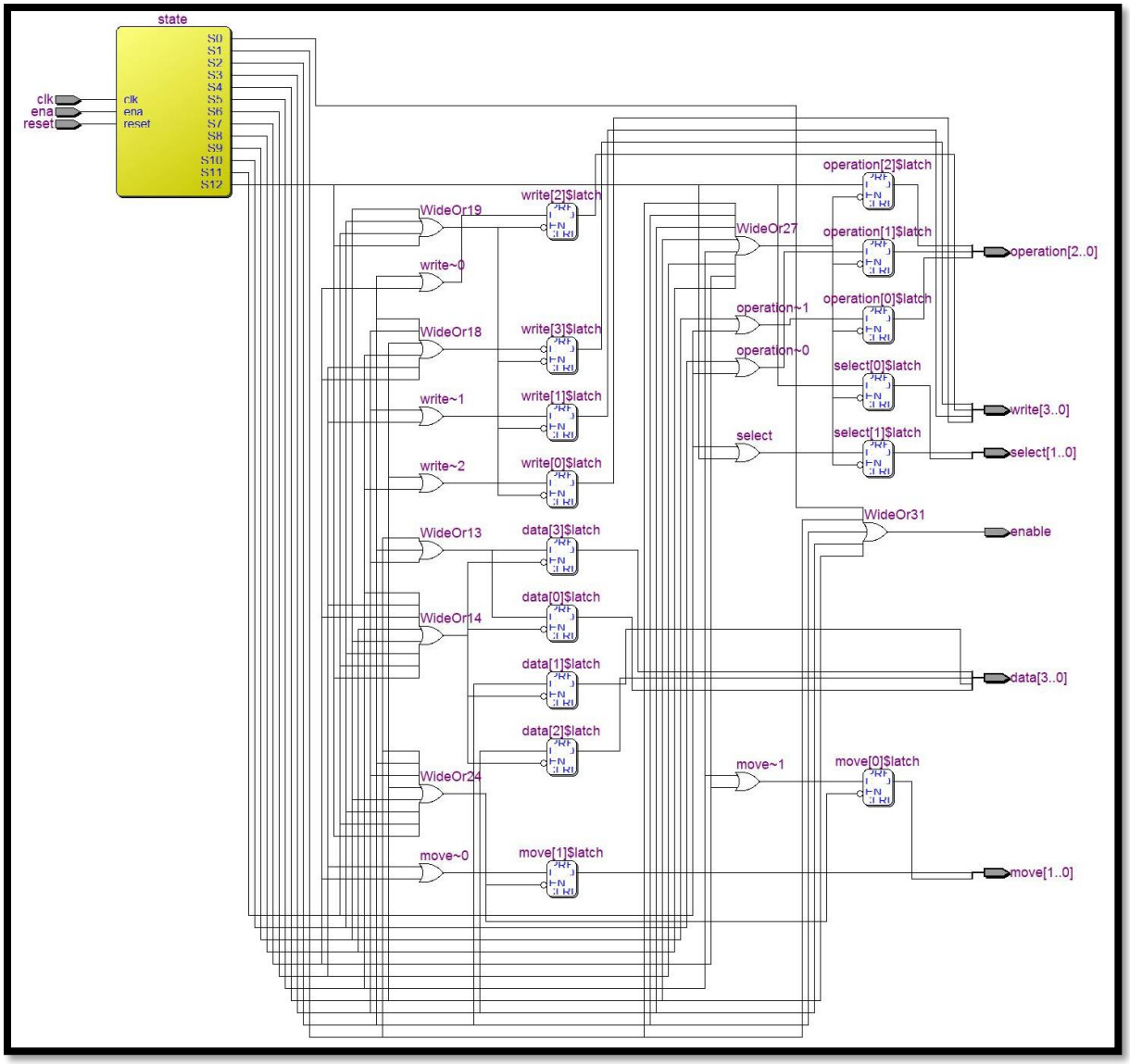



*Figure (4.27) Register transfer level schematic for control MUX.*

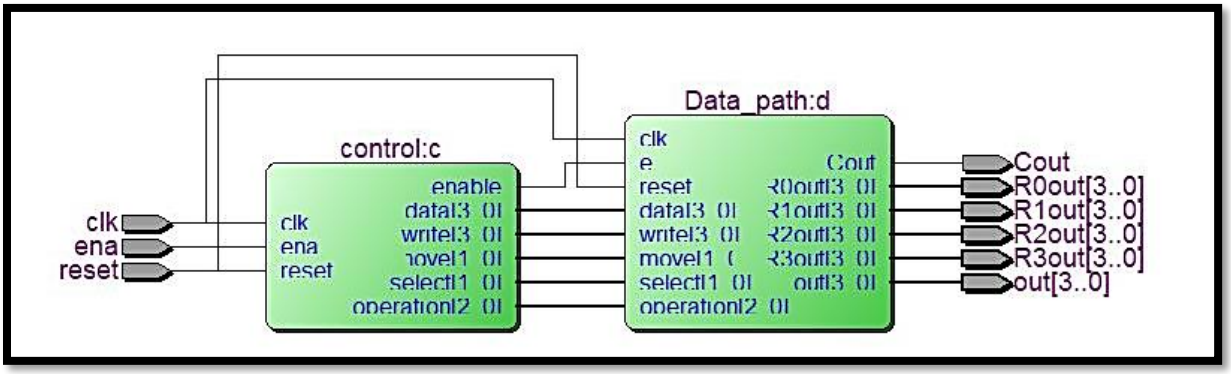



*Figure (4.28) Register transfer level schematic for top module MUX.*

### 4.4.4 Clock to Output Time

Is a timing analyser used by time quest applications under Intel-Altera Quartus II software tools. Time tests of both modules in figures (4.29) and (4.30) shows that both modules have approximately same time delay. In figure (4.31) shows that the multiplexer bus module has a lower power consumption than the tristate bus module.

**Clock to Output Times**

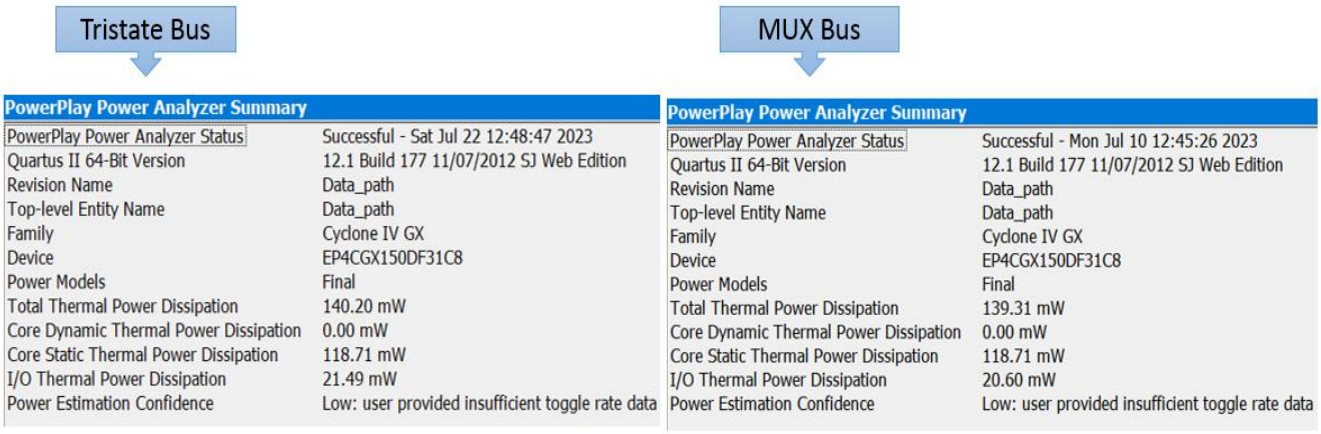
| | Data Port | Clock Port | Rise | Fall | Clock Edge | Clock Reference |
|---|---|---|---|---|---|---|
| 1 | R0out[*] | clock | 9.758 | 9.753 | Rise | clock |
| 1 | R0out[0] | clock | 9.758 | 9.753 | Rise | clock |
| 2 | R0out[1] | clock | 8.756 | 8.650 | Rise | clock |
| 3 | R0out[2] | clock | 8.242 | 8.192 | Rise | clock |
| 4 | R0out[3] | clock | 8.449 | 8.361 | Rise | clock |
| 5 | R0out[4] | clock | 9.226 | 9.071 | Rise | clock |
| 6 | R0out[5] | clock | 8.233 | 8.165 | Rise | clock |
| 7 | R0out[6] | clock | 9.470 | 9.305 | Rise | clock |
| 8 | R0out[7] | clock | 8.501 | 8.413 | Rise | clock |
| 2 | R1out[*] | clock | 10.792 | 10.692 | Rise | clock |
| 1 | R1out[0] | clock | 9.421 | 9.220 | Rise | clock |
| 2 | R1out[1] | clock | 8.743 | 8.639 | Rise | clock |
| 3 | R1out[2] | clock | 8.424 | 8.329 | Rise | clock |
| 4 | R1out[3] | clock | 8.717 | 8.608 | Rise | clock |
| 5 | R1out[4] | clock | 9.188 | 9.032 | Rise | clock |
| 6 | R1out[5] | clock | 10.792 | 10.692 | Rise | clock |
| 7 | R1out[6] | clock | 8.795 | 8.671 | Rise | clock |
| 8 | R1out[7] | clock | 8.799 | 8.658 | Rise | clock |
| 3 | R2out[*] | clock | 9.394 | 9.181 | Rise | clock |
| 1 | R2out[0] | clock | 9.394 | 9.181 | Rise | clock |
| 2 | R2out[1] | clock | 8.731 | 8.603 | Rise | clock |
| 3 | R2out[2] | clock | 8.694 | 8.554 | Rise | clock |
| 4 | R2out[3] | clock | 9.197 | 9.045 | Rise | clock |
| 5 | R2out[4] | clock | 9.133 | 9.000 | Rise | clock |
| 6 | R2out[5] | clock | 8.755 | 8.638 | Rise | clock |
| 7 | R2out[6] | clock | 8.863 | 8.766 | Rise | clock |
| 8 | R2out[7] | clock | 9.155 | 9.004 | Rise | clock |
| 4 | R3out[*] | clock | 9.507 | 9.326 | Rise | clock |
| 1 | R3out[0] | clock | 9.248 | 9.121 | Rise | clock |
| 2 | R3out[1] | clock | 9.491 | 9.292 | Rise | clock |
| 3 | R3out[2] | clock | 9.507 | 9.326 | Rise | clock |
| 4 | R3out[3] | clock | 9.136 | 8.995 | Rise | clock |
| 5 | R3out[4] | clock | 8.752 | 8.626 | Rise | clock |
| 6 | R3out[5] | clock | 8.754 | 8.668 | Rise | clock |
| 7 | R3out[6] | clock | 8.481 | 8.396 | Rise | clock |
| 8 | R3out[7] | clock | 9.142 | 9.000 | Rise | clock |
| 5 | out[*] | clock | 15.373 | 14.907 | Rise | clock |
| 1 | out[0] | clock | 12.989 | 12.761 | Rise | clock |
| 2 | out[1] | clock | 12.934 | 12.675 | Rise | clock |
| 3 | out[2] | clock | 14.631 | 14.318 | Rise | clock |
| 4 | out[3] | clock | 13.678 | 13.437 | Rise | clock |
| 5 | out[4] | clock | 14.473 | 14.274 | Rise | clock |
| 6 | out[5] | clock | 15.373 | 14.907 | Rise | clock |
| 7 | out[6] | clock | 13.448 | 13.256 | Rise | clock |
| 8 | out[7] | clock | 15.191 | 14.827 | Rise | clock |
| 6 | Cout | opration[2] | 7.468 | 7.378 | Rise | opration[2] |
| 7 | out[*] | opration[2] | 10.655 | 10.208 | Rise | opration[2] |
| 1 | out[0] | opration[2] | 10.123 | 9.737 | Rise | opration[2] |
| 2 | out[1] | opration[2] | 9.448 | 9.016 | Rise | opration[2] |
| 3 | out[2] | opration[2] | 10.655 | 10.208 | Rise | opration[2] |
| 4 | out[3] | opration[2] | 9.755 | 9.360 | Rise | opration[2] |
| 5 | out[4] | opration[2] | 10.137 | 9.690 | Rise | opration[2] |
| 6 | out[5] | opration[2] | 9.807 | 9.343 | Rise | opration[2] |
| 7 | out[6] | opration[2] | 9.459 | 9.074 | Rise | opration[2] |
| 8 | out[7] | opration[2] | 10.583 | 10.129 | Rise | opration[2] |
| 8 | out[*] | opration[2] | 10.655 | 10.208 | Fall | opration[2] |
| 1 | out[0] | opration[2] | 10.123 | 9.737 | Fall | opration[2] |
| 2 | out[1] | opration[2] | 9.448 | 9.016 | Fall | opration[2] |
| 3 | out[2] | opration[2] | 10.655 | 10.208 | Fall | opration[2] |
| 4 | out[3] | opration[2] | 9.755 | 9.360 | Fall | opration[2] |
| 5 | out[4] | opration[2] | 10.137 | 9.690 | Fall | opration[2] |
| 6 | out[5] | opration[2] | 9.807 | 9.343 | Fall | opration[2] |
| 7 | out[6] | opration[2] | 9.459 | 9.074 | Fall | opration[2] |
| 8 | out[7] | opration[2] | 10.583 | 10.129 | Fall | opration[2] |

*Figure (4.29) Tristate 8-bit in terms of time delay.*

**Clock to Output Times**

| | Data Port | Clock Port | Rise | Fall | Clock Edge | Clock Reference |
|---|---|---|---|---|---|---|
| 1 | R0out[*] | clock | 9.458 | 9.287 | Rise | clock |
| 1 | R0out[0] | clock | 9.458 | 9.287 | Rise | clock |
| 2 | R0out[1] | clock | 9.384 | 9.175 | Rise | clock |
| 3 | R0out[2] | clock | 9.101 | 8.936 | Rise | clock |
| 4 | R0out[3] | clock | 8.536 | 8.457 | Rise | clock |
| 5 | R0out[4] | clock | 8.858 | 8.765 | Rise | clock |
| 6 | R0out[5] | clock | 8.781 | 8.662 | Rise | clock |
| 7 | R0out[6] | clock | 8.765 | 8.642 | Rise | clock |
| 8 | R0out[7] | clock | 8.793 | 8.681 | Rise | clock |
| 2 | R1out[*] | clock | 9.600 | 9.411 | Rise | clock |
| 1 | R1out[0] | clock | 9.466 | 9.302 | Rise | clock |
| 2 | R1out[1] | clock | 9.489 | 9.335 | Rise | clock |
| 3 | R1out[2] | clock | 8.732 | 8.614 | Rise | clock |
| 4 | R1out[3] | clock | 9.600 | 9.411 | Rise | clock |
| 5 | R1out[4] | clock | 8.369 | 8.285 | Rise | clock |
| 6 | R1out[5] | clock | 9.169 | 9.005 | Rise | clock |
| 7 | R1out[6] | clock | 9.158 | 8.992 | Rise | clock |
| 8 | R1out[7] | clock | 9.188 | 9.075 | Rise | clock |
| 3 | R2out[*] | clock | 9.653 | 9.460 | Rise | clock |
| 1 | R2out[0] | clock | 9.368 | 9.185 | Rise | clock |
| 2 | R2out[1] | clock | 9.415 | 9.181 | Rise | clock |
| 3 | R2out[2] | clock | 9.572 | 9.379 | Rise | clock |
| 4 | R2out[3] | clock | 9.517 | 9.341 | Rise | clock |
| 5 | R2out[4] | clock | 9.110 | 8.961 | Rise | clock |
| 6 | R2out[5] | clock | 8.407 | 8.314 | Rise | clock |
| 7 | R2out[6] | clock | 8.808 | 8.670 | Rise | clock |
| 8 | R2out[7] | clock | 9.653 | 9.460 | Rise | clock |
| 4 | R3out[*] | clock | 9.754 | 9.513 | Rise | clock |
| 1 | R3out[0] | clock | 9.521 | 9.343 | Rise | clock |
| 2 | R3out[1] | clock | 9.348 | 9.129 | Rise | clock |
| 3 | R3out[2] | clock | 8.141 | 8.092 | Rise | clock |
| 4 | R3out[3] | clock | 9.754 | 9.513 | Rise | clock |
| 5 | R3out[4] | clock | 8.597 | 8.521 | Rise | clock |
| 6 | R3out[5] | clock | 9.016 | 8.876 | Rise | clock |
| 7 | R3out[6] | clock | 9.093 | 8.943 | Rise | clock |
| 8 | R3out[7] | clock | 8.408 | 8.304 | Rise | clock |
| 5 | out[*] | clock | 15.959 | 15.771 | Rise | clock |
| 1 | out[0] | clock | 13.356 | 13.076 | Rise | clock |
| 2 | out[1] | clock | 14.115 | 13.865 | Rise | clock |
| 3 | out[2] | clock | 13.620 | 13.287 | Rise | clock |
| 4 | out[3] | clock | 13.667 | 13.414 | Rise | clock |
| 5 | out[4] | clock | 14.194 | 13.895 | Rise | clock |
| 6 | out[5] | clock | 15.959 | 15.771 | Rise | clock |
| 7 | out[6] | clock | 14.098 | 13.803 | Rise | clock |
| 8 | out[7] | clock | 15.423 | 15.074 | Rise | clock |
| 6 | Cout | operation[2] | 8.054 | 7.871 | Rise | operation[2] |
| 7 | out[*] | operation[2] | 11.497 | 11.165 | Rise | operation[2] |
| 1 | out[0] | operation[2] | 10.635 | 10.283 | Rise | operation[2] |
| 2 | out[1] | operation[2] | 10.833 | 10.485 | Rise | operation[2] |
| 3 | out[2] | operation[2] | 10.342 | 9.988 | Rise | operation[2] |
| 4 | out[3] | operation[2] | 10.313 | 9.956 | Rise | operation[2] |
| 5 | out[4] | operation[2] | 9.915 | 9.490 | Rise | operation[2] |
| 6 | out[5] | operation[2] | 11.497 | 11.165 | Rise | operation[2] |
| 7 | out[6] | operation[2] | 9.423 | 8.999 | Rise | operation[2] |
| 8 | out[7] | operation[2] | 10.607 | 10.165 | Rise | operation[2] |
| 8 | out[*] | operation[2] | 11.497 | 11.165 | Fall | operation[2] |
| 1 | out[0] | operation[2] | 10.635 | 10.283 | Fall | operation[2] |
| 2 | out[1] | operation[2] | 10.833 | 10.485 | Fall | operation[2] |
| 3 | out[2] | operation[2] | 10.342 | 9.988 | Fall | operation[2] |
| 4 | out[3] | operation[2] | 10.313 | 9.956 | Fall | operation[2] |
| 5 | out[4] | operation[2] | 9.915 | 9.490 | Fall | operation[2] |
| 6 | out[5] | operation[2] | 11.497 | 11.165 | Fall | operation[2] |
| 7 | out[6] | operation[2] | 9.423 | 8.999 | Fall | operation[2] |
| 8 | out[7] | operation[2] | 10.607 | 10.165 | Fall | operation[2] |

*Figure (4.30) MUX 8-bit in terms of time delay.*




**PowerPlay Power Analyzer Summary**

| | |
|---|---|
| PowerPlay Power Analyzer Status | Successful - Sat Jul 22 12:48:47 2023 |
| Quartus II 64-Bit Version | 12.1 Build 177 11/07/2012 SJ Web Edition |
| Revision Name | Data_path |
| Top-level Entity Name | Data_path |
| Family | Cyclone IV GX |
| Device | EP4CGX150DF31C8 |
| Power Models | Final |
| Total Thermal Power Dissipation | 140.20 mW |
| Core Dynamic Thermal Power Dissipation | 0.00 mW |
| Core Static Thermal Power Dissipation | 118.71 mW |
| I/O Thermal Power Dissipation | 21.49 mW |
| Power Estimation Confidence | Low: user provided insufficient toggle rate data |

**PowerPlay Power Analyzer Summary**

| | |
|---|---|
| PowerPlay Power Analyzer Status | Successful - Mon Jul 10 12:45:26 2023 |
| Quartus II 64-Bit Version | 12.1 Build 177 11/07/2012 SJ Web Edition |
| Revision Name | Data_path |
| Top-level Entity Name | Data_path |
| Family | Cyclone IV GX |
| Device | EP4CGX150DF31C8 |
| Power Models | Final |
| Total Thermal Power Dissipation | 139.31 mW |
| Core Dynamic Thermal Power Dissipation | 0.00 mW |
| Core Static Thermal Power Dissipation | 118.71 mW |
| I/O Thermal Power Dissipation | 20.60 mW |
| Power Estimation Confidence | Low: user provided insufficient toggle rate data |

*Figure (4.31) Different between tristate and MUX in terms of thermal power.*

Time tests of top module in figures (4.32) shows time delay for all output. In figure (4.33) show total terminal power of the module.

**Clock to Output Times**

| | Data Port | Clock Port | Rise | Fall | Clock Edge | Clock Reference |
|---|---|---|---|---|---|---|
| 1 | R0out[*] | clk | 8.978 | 8.870 | Rise | clk |
| 1 | R0out[0] | clk | 8.223 | 8.175 | Rise | clk |
| 2 | R0out[1] | clk | 8.224 | 8.162 | Rise | clk |
| 3 | R0out[2] | clk | 8.565 | 8.467 | Rise | clk |
| 4 | R0out[3] | clk | 8.978 | 8.870 | Rise | clk |
| 2 | R1out[*] | clk | 9.495 | 9.344 | Rise | clk |
| 1 | R1out[0] | clk | 9.495 | 9.344 | Rise | clk |
| 2 | R1out[1] | clk | 8.715 | 8.640 | Rise | clk |
| 3 | R1out[2] | clk | 9.218 | 9.078 | Rise | clk |
| 4 | R1out[3] | clk | 8.799 | 8.708 | Rise | clk |
| 3 | R2out[*] | clk | 10.927 | 10.642 | Rise | clk |
| 1 | R2out[0] | clk | 9.784 | 9.594 | Rise | clk |
| 2 | R2out[1] | clk | 9.276 | 9.216 | Rise | clk |
| 3 | R2out[2] | clk | 10.927 | 10.642 | Rise | clk |
| 4 | R2out[3] | clk | 9.741 | 9.590 | Rise | clk |
| 4 | R3out[*] | clk | 13.527 | 13.330 | Rise | clk |
| 1 | R3out[0] | clk | 10.225 | 10.019 | Rise | clk |
| 2 | R3out[1] | clk | 9.832 | 9.668 | Rise | clk |
| 3 | R3out[2] | clk | 13.527 | 13.330 | Rise | clk |
| 4 | R3out[3] | clk | 12.400 | 12.310 | Rise | clk |
| 5 | out[*] | clk | 16.891 | 16.401 | Rise | clk |
| 1 | out[0] | clk | 13.276 | 13.136 | Rise | clk |
| 2 | out[1] | clk | 15.924 | 15.574 | Rise | clk |
| 3 | out[2] | clk | 16.891 | 16.401 | Rise | clk |
| 4 | out[3] | clk | 16.486 | 16.362 | Rise | clk |
| 6 | out[*] | control:c\|operation[0] | 12.134 | 11.676 | Rise | control:c\|operation[0] |
| 1 | out[0] | control:c\|operation[0] | 7.109 | 6.977 | Rise | control:c\|operation[0] |
| 2 | out[1] | control:c\|operation[0] | 11.108 | 10.758 | Rise | control:c\|operation[0] |
| 3 | out[2] | control:c\|operation[0] | 12.134 | 11.676 | Rise | control:c\|operation[0] |
| 4 | out[3] | control:c\|operation[0] | 11.140 | 11.056 | Rise | control:c\|operation[0] |
| 7 | Cout | control:c\|operation[0] | 7.809 | 7.788 | Fall | control:c\|operation[0] |
| 8 | out[*] | control:c\|operation[0] | 11.855 | 11.491 | Fall | control:c\|operation[0] |
| 1 | out[0] | control:c\|operation[0] | 7.072 | 6.932 | Fall | control:c\|operation[0] |
| 2 | out[1] | control:c\|operation[0] | 10.865 | 10.557 | Fall | control:c\|operation[0] |
| 3 | out[2] | control:c\|operation[0] | 11.855 | 11.491 | Fall | control:c\|operation[0] |
| 4 | out[3] | control:c\|operation[0] | 10.869 | 10.862 | Fall | control:c\|operation[0] |
| 9 | out[*] | control:c\|state.S0 | 16.592 | 16.102 | Rise | control:c\|state.S0 |
| 1 | out[0] | control:c\|state.S0 | 12.740 | 12.600 | Rise | control:c\|state.S0 |
| 2 | out[1] | control:c\|state.S0 | 15.388 | 15.038 | Rise | control:c\|state.S0 |
| 3 | out[2] | control:c\|state.S0 | 16.592 | 16.102 | Rise | control:c\|state.S0 |
| 4 | out[3] | control:c\|state.S0 | 15.945 | 15.821 | Rise | control:c\|state.S0 |
| 10 | out[*] | control:c\|state.S10 | 16.634 | 16.144 | Fall | control:c\|state.S10 |
| 1 | out[0] | control:c\|state.S10 | 12.782 | 12.642 | Fall | control:c\|state.S10 |
| 2 | out[1] | control:c\|state.S10 | 15.430 | 15.080 | Fall | control:c\|state.S10 |
| 3 | out[2] | control:c\|state.S10 | 16.634 | 16.144 | Fall | control:c\|state.S10 |
| 4 | out[3] | control:c\|state.S10 | 15.987 | 15.863 | Fall | control:c\|state.S10 |

*Figure (4.32) Top module MUX in terms of time delay.*

| PowerPlay Power Analyzer Summary | |
|---|---|
| PowerPlay Power Analyzer Status | Successful - Thu Jul 13 21:01:10 2023 |
| Quartus II 64-Bit Version | 12.1 Build 177 11/07/2012 SJ Web Edition |
| Revision Name | Top_module |
| Top-level Entity Name | Top_module |
| Family | Cyclone IV GX |
| Device | EP4CGX150DF31C8 |
| Power Models | Final |
| Total Thermal Power Dissipation | 133.51 mW |
| Core Dynamic Thermal Power Dissipation | 0.00 mW |
| Core Static Thermal Power Dissipation | 118.69 mW |
| I/O Thermal Power Dissipation | 14.81 mW |
| Power Estimation Confidence | Low: user provided insufficient toggle rate data |

*Figure (4.33) Top module MUX in terms of thermal power.*

### 4.4.5 System Frequency

The system of top module of MUX is working on 1000 MHz as shown in figure (4.34).

Clocks

| | Clock Name | Type | Period | Frequency | Rise | Fall | Duty Cycle | Divide by | Multiply by | Phase |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | clk | Base | 1.000 | 1000.0 MHz | 0.000 | 0.500 | | | | |
| 2 | control:c\|operation[0] | Base | 1.000 | 1000.0 MHz | 0.000 | 0.500 | | | | |
| 3 | control:c\|state.S0 | Base | 1.000 | 1000.0 MHz | 0.000 | 0.500 | | | | |
| 4 | control:c\|state.S10 | Base | 1.000 | 1000.0 MHz | 0.000 | 0.500 | | | | |

*Figure (4.34) Top module MUX system frequency*

# Chapter 5 Conclusion

This project aims to develop a highly efficient microprocessor system by utilizing a multiplexer-based bus structure. The primary objective is to design, simulate, and control this system on the Altera FPGA development board, specifically the Cyclone IV GX FPGA. The significance of this contribution lies in its applicability to FPGA designs with limited tristate bus resources. Through extensive simulations and testing, the obtained waveforms and results demonstrate that the proposed microprocessor structure effectively reduces hardware resource usage compared to traditional tristate-based bus structures. Furthermore, it achieves comparable time delay while consuming less thermal power.

# Chapter 6 Future Work

In the future work, the investigation can be extended to include implementations of multiprocessors system that can mix two kind of buses to interconnect between internal registers and control data movement for system instead of only tri-state bus. After verifying the design sub-modules individually, the submodules are instantiated, simulated and verified, the system was implemented and tested using Cyclone EP1C6Q240C8 FPGA evaluation platform that designed specially to test the functionality of the system in hardware.

# Reference


[1].Li Jingpeng, “An optimized design of MCU including predication,” Microelectronics and computer, vol.23, pp.25-27, 2006.

[2].Tian Hongli, Yan Huiqiang, Geng Hengshan, Liu Su, “Design an implementation of 8-bit micro-controller,” Computer Engineering and Applications, Vol.46, pp.60-63, 2010.

[3].Johnson and Graham, “High Speed Digital Design: a Handbook of Black Magic,” Prentice Hall,1993.

[4]. Nikil Dutt, Kaustav Banerjee, Luca Benini, Kanishka Lahiri, Sudeep Pasricha, "Tutorial 5: SoC Communication Architectures: Technology, Current Practice, Research, and Trends", vlsid, pp.8, 20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems (VLSID'07), 2007.

[5].Altera Corporation, “Comparing IP Integration Approaches for FPGA Implementation”.

[6].The IEEE Standard Hardware Description Language based on the Verilog Hardware Description Language (IEEE Std 1364-2001).

[7].Micheal D. Ciletti, “Advanced Digital Design with the Verilog HDL “Prentice Hall,2004.

[8].William Stallings, “Computer Organization and Architecture, Designing for Performance”, Prentice Hall, 2001.

[9]. A.-S. Jamal, J. Goeders, and S. J. E. Wilton, “An FPGA overlay architecture supporting rapid implementation of functional changes during on-chip debug,” in 2018 28th International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2018.

[10]. A.-S. Jamal, “An FPGA overlay architecture supporting software-like compile times during on-chip debug of high-level synthesis designs,” Ph.D. dissertation, University of British Columbia, 2018.

[11]. P. Mishra and F.Farahmandi, Post-Silicon Validation and Debug. Cham, Switzerland: Springer, 2019.

[12]. H. Oh, T. Han, I. Choi, and S. Kang, “An on-chip error detection method to reduce the post-silicon debug time,” IEEE Transactions on Computers, vol. 66, no.1, pp . 38_44, Jan 2017.

[13]. H. Oh, I. Choi, and S. Kang, “DRAM-based error detection method to reduce the post-silicon debug time for multiple identical cores,” IEEE Transactions on Computers, vol. 66, no. 9, pp. 1504–1517, Sep. 2017.

[14]. Y. Cao, H. Palombo, S. Ray, and H. Zheng, “Enhancing observability for post-silicon debug with on-chip communication monitors,” in 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), July 2018, pp. 602–607.
[15]. D. Holanda Noronha, R. Zhao, J. Goeders, W. Luk, and S. J. E. Wilton, “On-chip fpga debug instrumentation for machine learning applications,” in Proceedings of the 2019 ACM / SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2019, pp. 110–115.
[16]. K. Rahmani and P. Mishra, “Feature-based signal selection for post-silicon debug using machine learning,” IEEE Transactions on Emerging Topics in Computing, pp. 1–1, 2017.
[17]. Https://en.wikipedia.org/wiki/Field-programmable_gate_array.
[18]. https://home.engineering.iastate.edu/~zzhang/courses/cpre581-f05/resources/modelsim .pdf
[19]. https://www.intel.com/content/www/us/en/programmable/quartushelp/13.0/merge dProjects /quartus/gl_quartus_welcome.htm.
[20]. https://www.xilinx.com/products/silicon-devices/resources/programming-an-fpga-an-introduction-to-how-it-works.html.