Group 47

Choong Jin Ng | 301226977

Ryan Kiew Ruelt Yean | 301290779

Sachin Momuli | 301297150

ENSC 350 Final project part 1 report

Table of Contents

[Introduction 2](#_Toc37015642)

[LogicUnit 2](#_Toc37015643)

[Overview 2](#_Toc37015644)

[Functional Behaviour 3](#_Toc37015645)

[Circuit Synthesis 5](#_Toc37015646)

[Timing Simulations 7](#_Toc37015647)

[ArithUnit 8](#_Toc37015648)

[Overview 8](#_Toc37015649)

[Functional Behaviour 11](#_Toc37015650)

[Circuit Synthesis 12](#_Toc37015651)

[Timing Simulations 15](#_Toc37015653)

[Conclusion 16](#_Toc37015654)

[Appendix 17](#_Toc37015655)

[Appendix A: Logic Gates 17](#_Toc37015656)

[Table of Tables 19](#_Toc37015657)

[Table of Figures 19](#_Toc37015658)

# Introduction

The objective of part 1 of this project is to design, synthesize, and test an Arithmetic Logic Unit (ALU) – the first step in assembling a processor. The ALU is responsible for performing arithmetic and bitwise operations on integer binary numbers, and is an essential building block of many other types of computing circuits.

This ALU design is separated into two parts, the Logic Unit and the Arithmetic Unit; the Logic Unit appropriately performs logical bitwise operations on two 64-bit input signals while the Arithmetic Unit is responsible for producing the arithmetic result of two 64-bit input signals, depending on certain context variables.

After designing the respective units, the circuits are synthesized using Quartus Prime’s RTL netlist viewer. The circuits will be synthesized for a Cyclone IV FPGA. We will then be using ModelSim to perform functional and timing simulations, which will be compared to the provided testbench values to verify that the units are working as intended.

# LogicUnit

## Overview

The *LogicUnit* is responsible for selecting and operating Logic Bitwise operations of two 64-bit input signals, *A* and *B*. This design incorporates the following operations:

* Pass the signal of B
* The result of A xor B
* The result of A or B
* The result of A and B

These initial logical operations are computed immediately, with the results passed along by a multiplexer as signal *Y,* depending on the signal *LogicFn*. The block diagram of the *LogicUnit* is represented in Figure 1 and the truth table of the *LogicUnit* is indicated in Table 1. The VHDL representation is given in Figure 2.

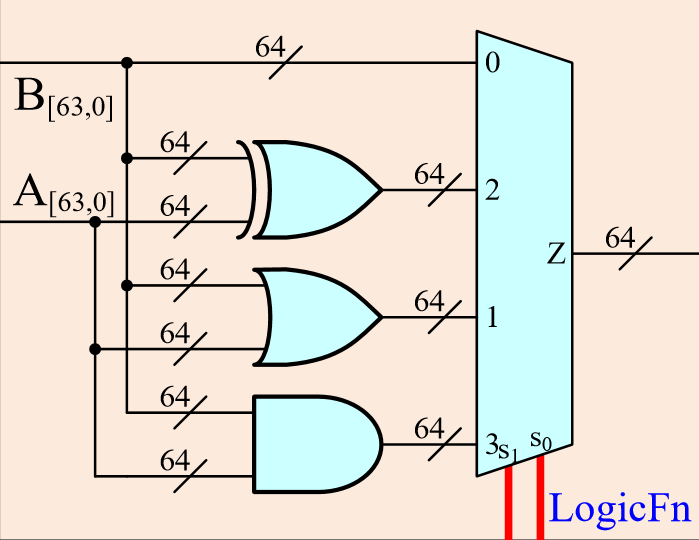


Figure 1: Block Diagram of LogicUnit Circuit

|  |  |
| --- | --- |
| **LogicFn Signal** | **Operation (Signal Y)** |
| 0 0 | B |
| 0 1 | A xor B |
| 1 0 | A or B |
| 1 1 | A and B |

Table 1: Truth Table of LogicUnit

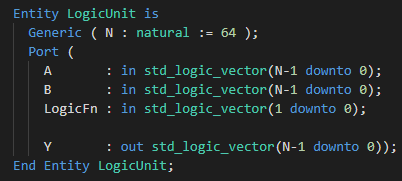


Figure 2: VHDL Interface of LogicUnit

## Functional Behaviour

The functional behaviour of *LogicUnit* can be demonstrated in three example diagrams Figure 3, Figure 4 and Figure 5. In Figure 3, the different *LogicFn* signals are demonstrated.

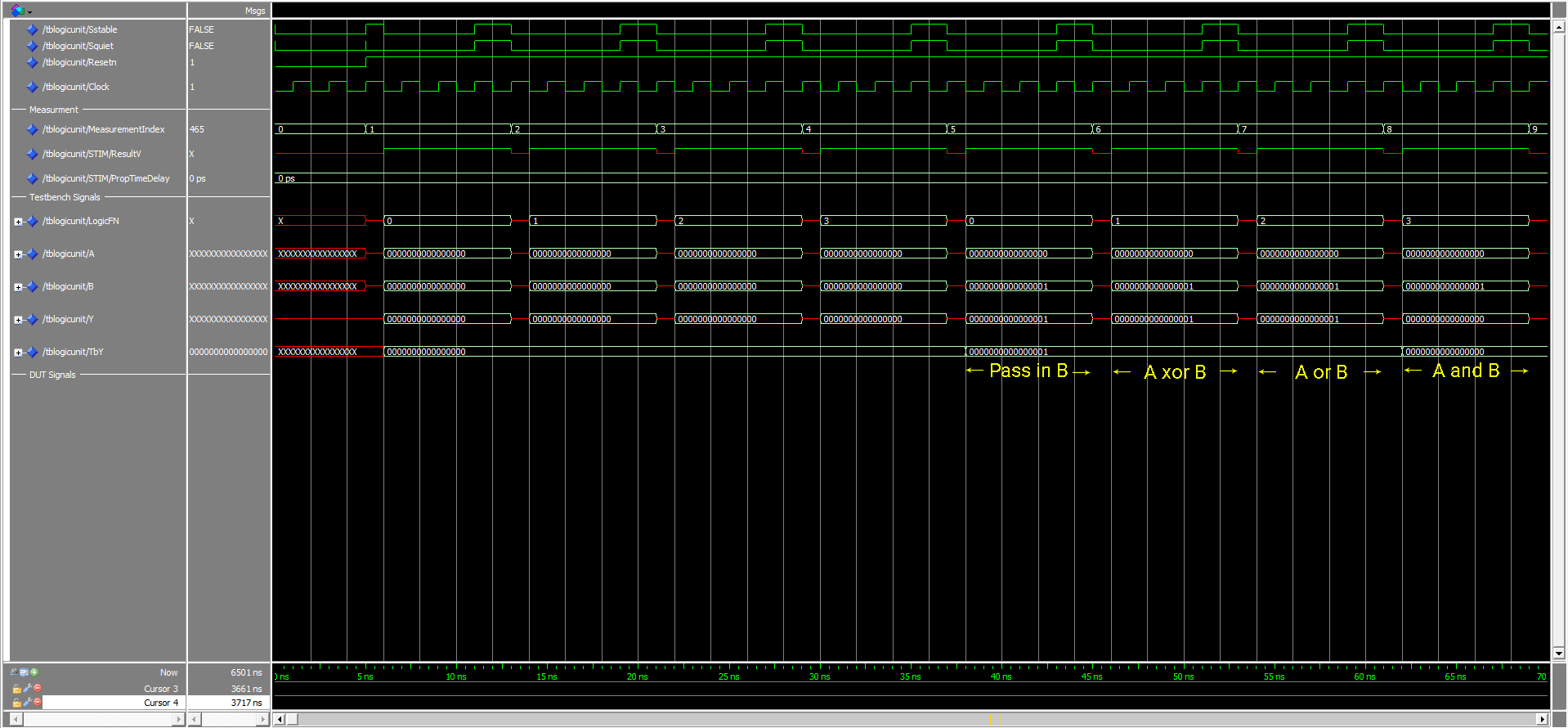


Figure 3: Functional Simulation for LogicUnit from t=0 to t=70ns

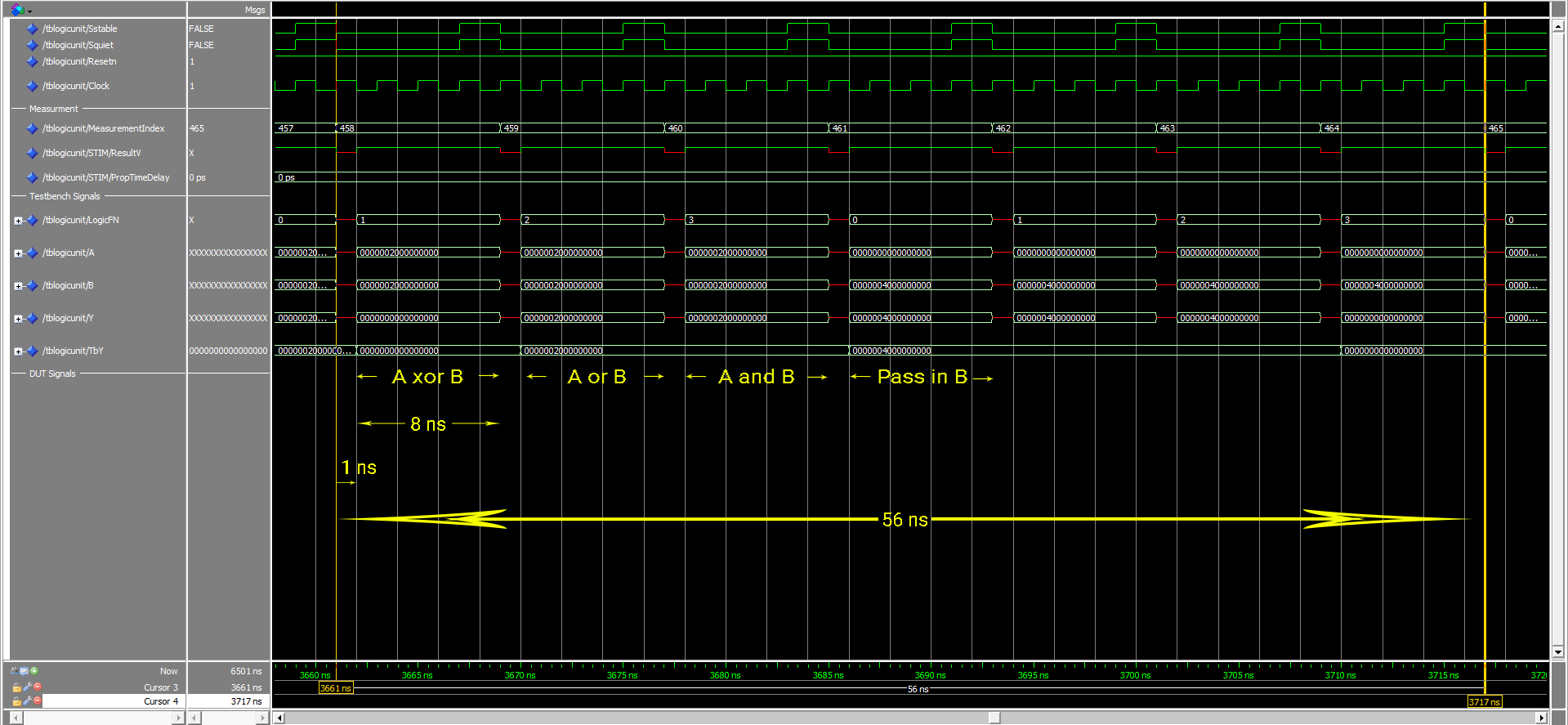


Figure 4: Functional Simulation for LogicUnit from measurement #458 to measurement #464

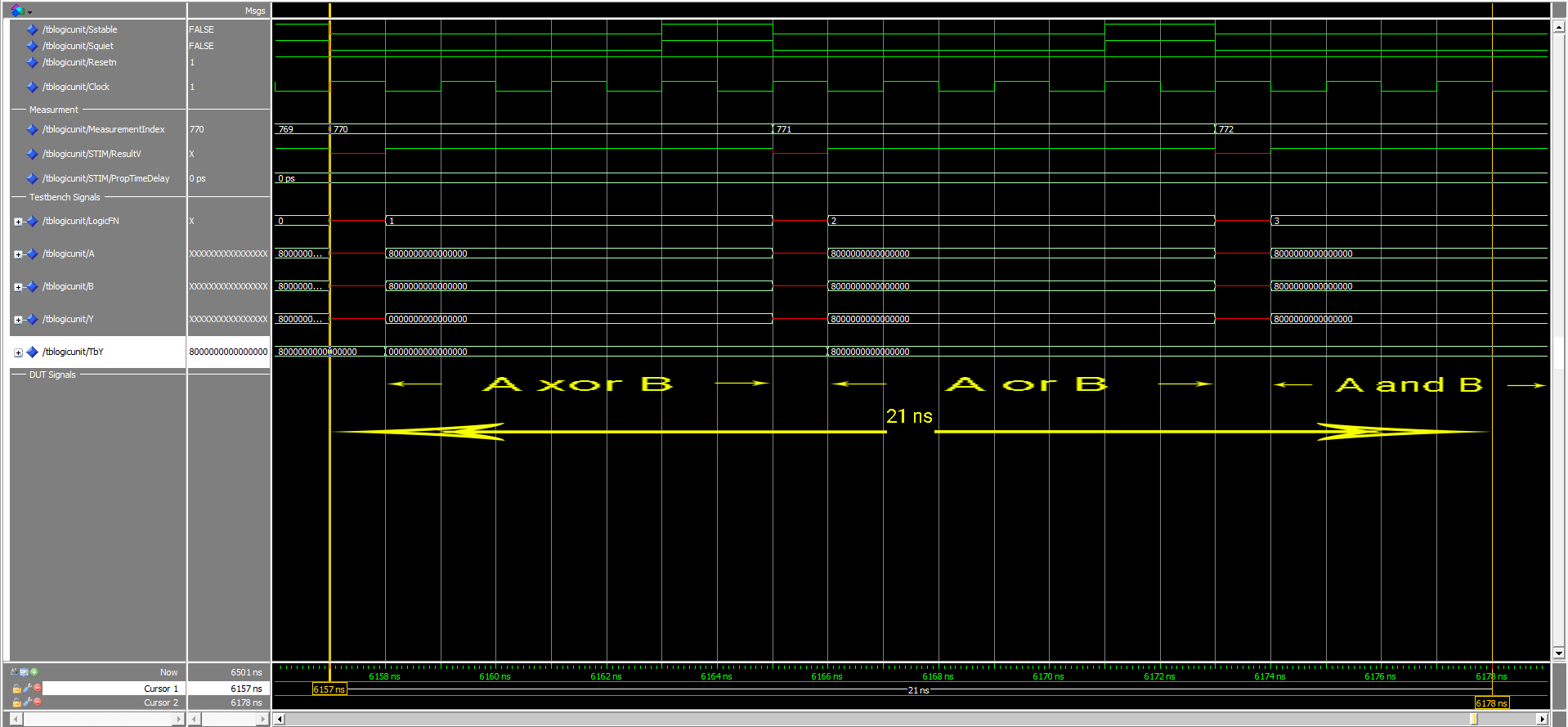


Figure 5: Functional Simulation for LogicUnit from measurement #770 to measurement #772

## Circuit Synthesis

As indicated, the three logical diagrams are *XorGate*, *AndGate*, and *OrGate*. The VHDL interface for the three block diagrams and synthesised circuits are listed in Appendix A: Logic Gates.

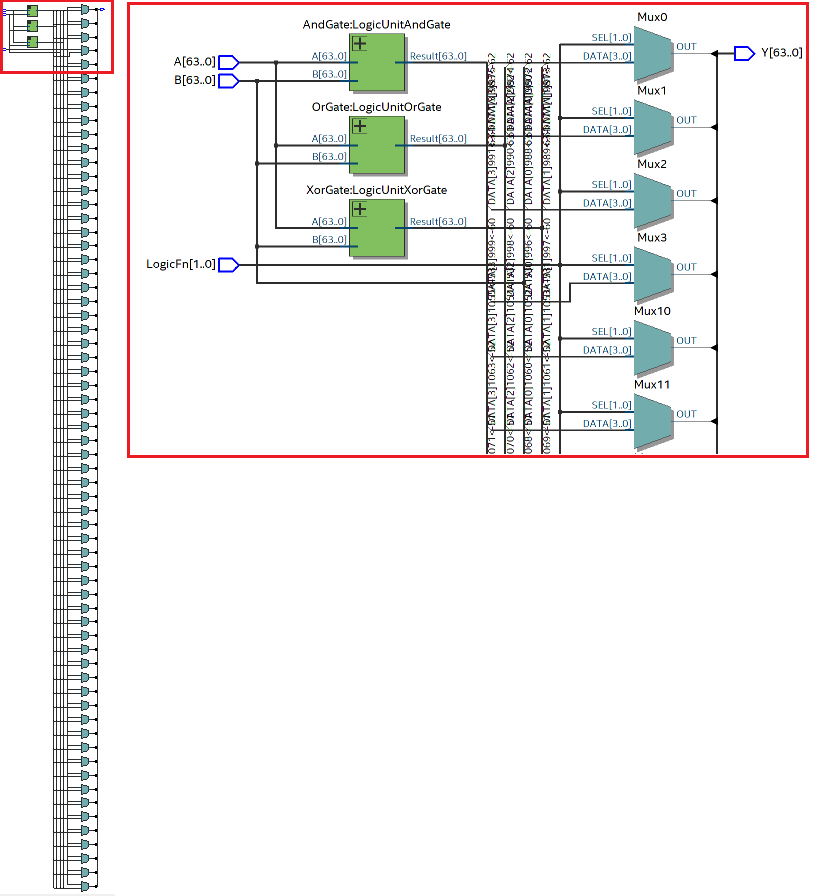


Figure 6: Synthesised Circuit of LogicUnit

## Timing Simulations

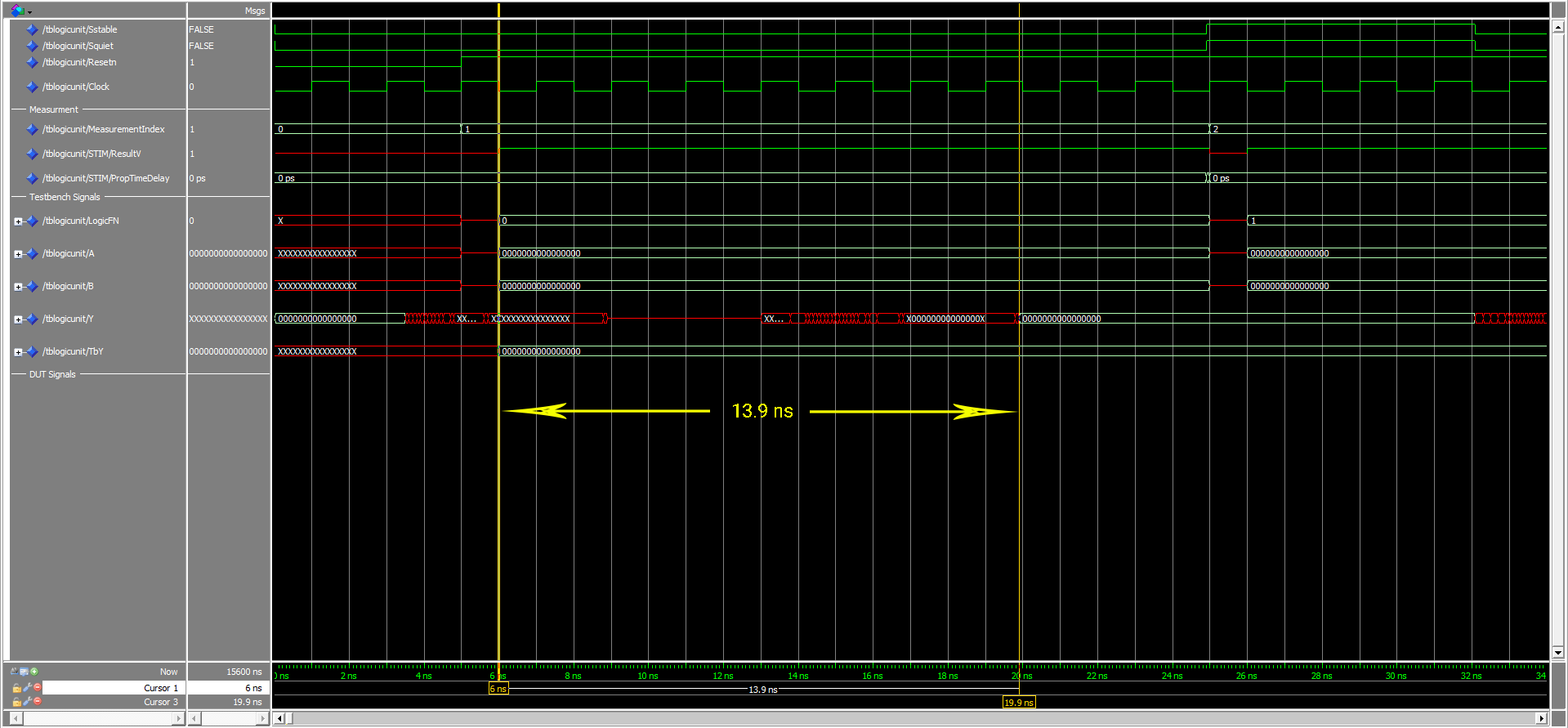


Figure 7: Timing Simulation for LogicUnit of Measurement #1

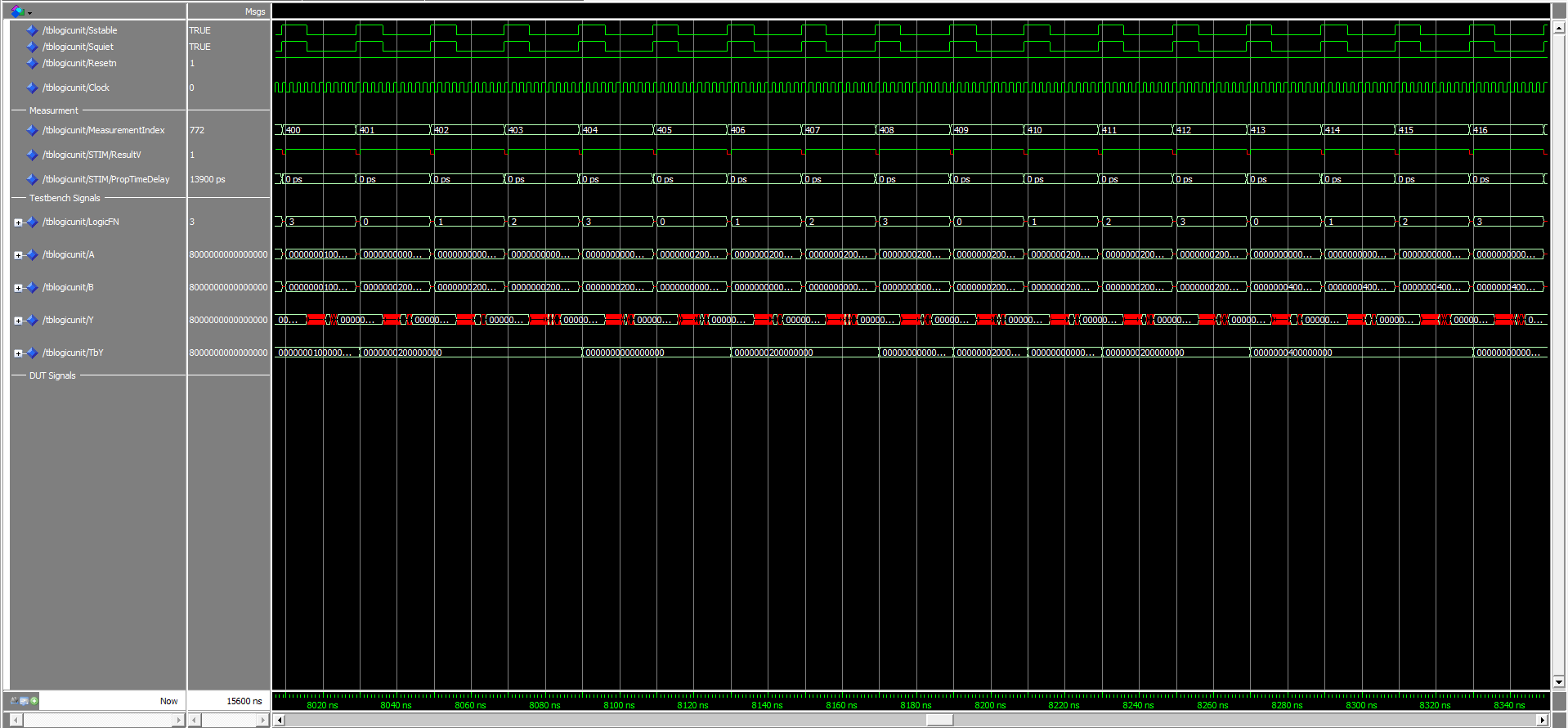


Figure 8: Timing Simulation for LogicUnit from measurement #400 to measurement #416

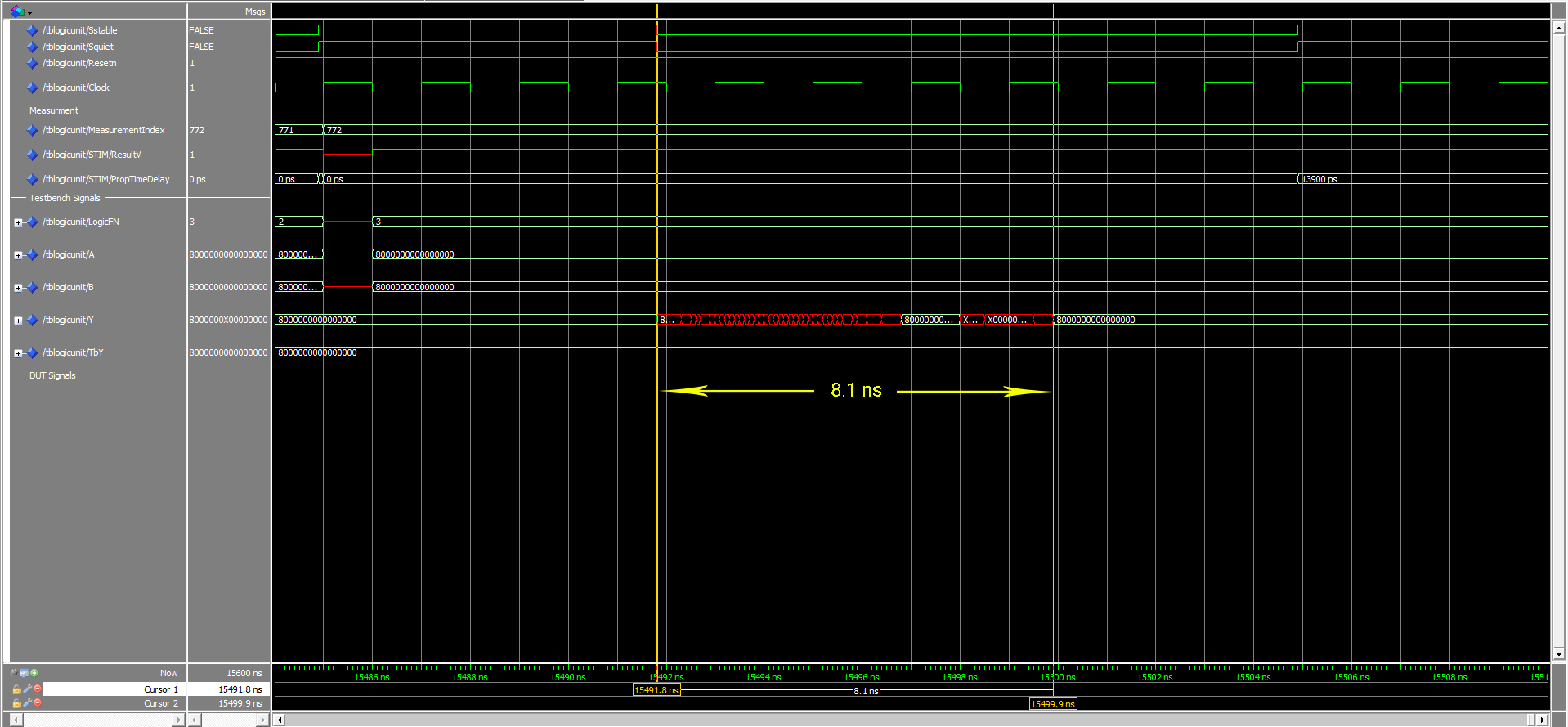


Figure 9: Timing Simulation for LogicUnit of Measurement #772

# ArithUnit

## Overview

The ArithUnit is responsible for producing the appropriate arithmetic result depending on the context. The input context variables here are:

* A, B : 64-bit input signals to be used in the adder
* AddnSub : determines whether the operation carried out is an add or a subtract
* NotA : to be used later for retrieval of instructions
* ExtWord : determines whether or not to sign extend the value

The output variables are:

* Y : result of the arithmetic operation
* Cout : outgoing carry of the result
* Ovfl : signifies an overflow in the result
* Zero : signifies if A is equal to B
* AltB, AltBu : signed and unsigned flags that indicate whether A is less than B

The *ArithUnit* was designed to handle both 64 and 32-bit numbers through the *ExtWord* flag. It currently does not support any other operations other than addition and subtraction. If the operation performed is a subtraction, it simply negates the value in *B* and feeds it into the adder. The output signals are designed with future function implementations in mind. For example, *AltB* and *AltBu* may seem insignificant, but will eventually be used to implement branching instructions in the future.

The block diagram of the *ArithUnit* is represented in Figure 10. The VHDL interface of the *ArithUnit* is given in Figure 11.

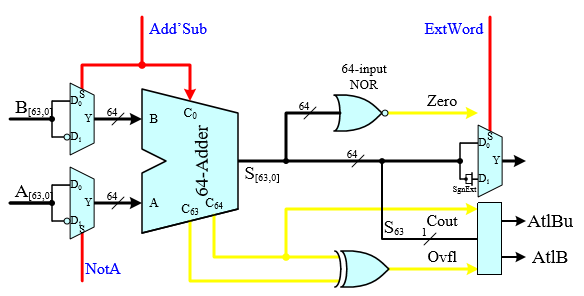


Figure 10: Block diagram of the ArithUnit

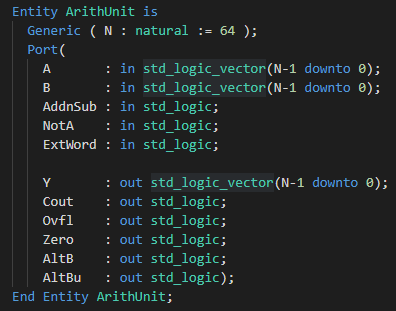


Figure 11: VHDL interface of the ArithUnit

The adder we used to is a simple ripple adder. The adder performs the addition bit-by-bit and propagates any carry that exists. Other than the result of the operation, it also returns the outgoing carry value, *Cout*, and the Overflow flag, *Ovfl*. *Cout* is simply the final carry value of the carry array, and *Overflow* is computed as the XOR of the last and second-to-last carry values of the carry array. These values are used in the ArithUnit to compute *AltB* and *AltBu*.

The block diagram of the Adder is represented in Figure 12. The VHDL interface of the Adder is given in Figure 13.

A screenshot of a cell phone

Description automatically generatedFigure 12: Block diagram of the Adder

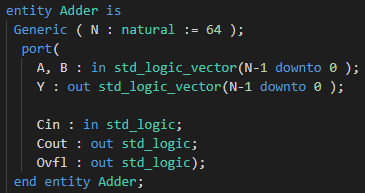


Figure 13: VHDL interface of the Adder

## Functional Behaviour

A picture containing screen, monitor, television, clock

Description automatically generated

Figure 14: Functional Simulation of the Arithmetic Unit

We can confirm that the *ArithUnit* is functioning as intended by observing that the Unit’s *Y* values are the same as the testbench – *TbY*. This figure also encapsulates all the different combinations of the context variables and its resulting output. For example, as annotated on the figure, we have a test case where the arithmetic operation is an ADD with no sign extension, that produces a *Cout* and no *Overflow*. We also have another test case that showcases SUBTRACT with sign extension, that similarly produces a *Cout* with no *Overflow*, but this time the *AltB* flag is also set.

## A close up of a map Description automatically generatedCircuit Synthesis

Figure 15: RTL synthesized circuit of the ArithUnit

A close up of a map

Description automatically generatedFigures 15 and 16 represent the RTL synthesized circuit of the *ArithUnit*. The circuit is organized so that every input bit can be observed from the figure. The functional logic portion of the unit are mostly performed in parallel, and only take up two layers of delay before producing the final result. The ripple adder used in this circuit is represented in green, and the synthesized circuit diagram of it is displayed in Figure 17. Since a basic ripple adder was used, the resulting propagation delay is quite significant. We could reduce this delay by implementing a carry-skip or Brent Kung adder instead. The output of the Adder is then used in the *ArithUnit* in the process of obtaining the final output value *Y*.

Figure 16: Panned-out view of the RTL synthesized circuit of the ArithUnit

## A picture containing screenshot Description automatically generated

Figure 17: RTL synthesized circuit diagram of Adder

## Timing Simulations

Figure 18: TIming Simulation for the ArithUnit

A picture containing monitor, television, screen, mounted

Description automatically generated

The timing of the *ArithUnit* can be verified by observing that the results are obtained well within the allocated time period. For measurement #1, the total propagation delay is 18.7 ns, with 8.3 ns to spare. The ripple adder took 7.3 ns to compute the appropriate result – roughly 40% of the entire time taken. If we were to improve this circuit in the future, improving the implementation of the adder will significantly improve the circuit’s efficiency.

# Conclusion

In this report, we outline two units for computational purposes and its performance. The Logic Unit determines whether bit-wise logical operations are performed while the Arithmetic Unit adds two 64-bit inputs together. These two combined will form the Arithmetic Logic Unit, the first steps to a general-purpose processor. Their propagation delay is respectively X and Y.

# Appendix

## Appendix A: Logic Gates

Inside the *LogicUnit*, the logic gates are represented by three entities, *AndGate*, *OrGate* and *XorGate*. They are use the operators *and*, *or* and *xor* which are available in the IEEE standardised library. The logic gates entity interfaces and implementation for *AndGate*, *OrGate* and *XorGate* is represented in Figure 19, Figure 20 and Figure 21 respectively.

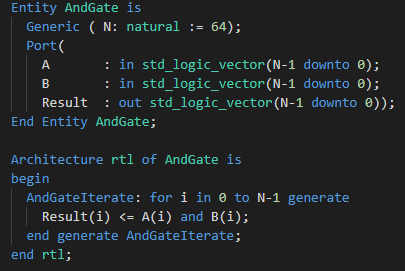


Figure 19: VHDL Interface and Implementation of AndGate

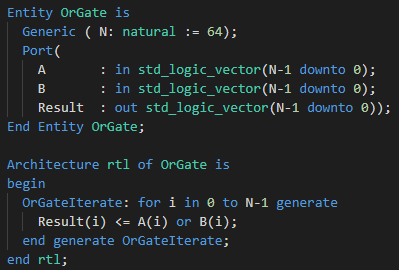


Figure 20: VHDL Interface and Implementation of OrGate

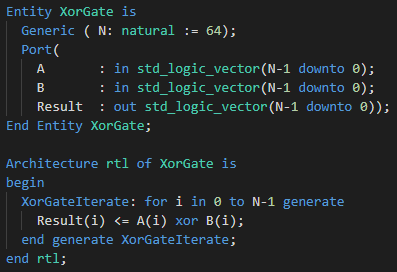


Figure 21: VHDL Interface and Implementation of XorGate

The synthesised circuits are represented in Figure 22.

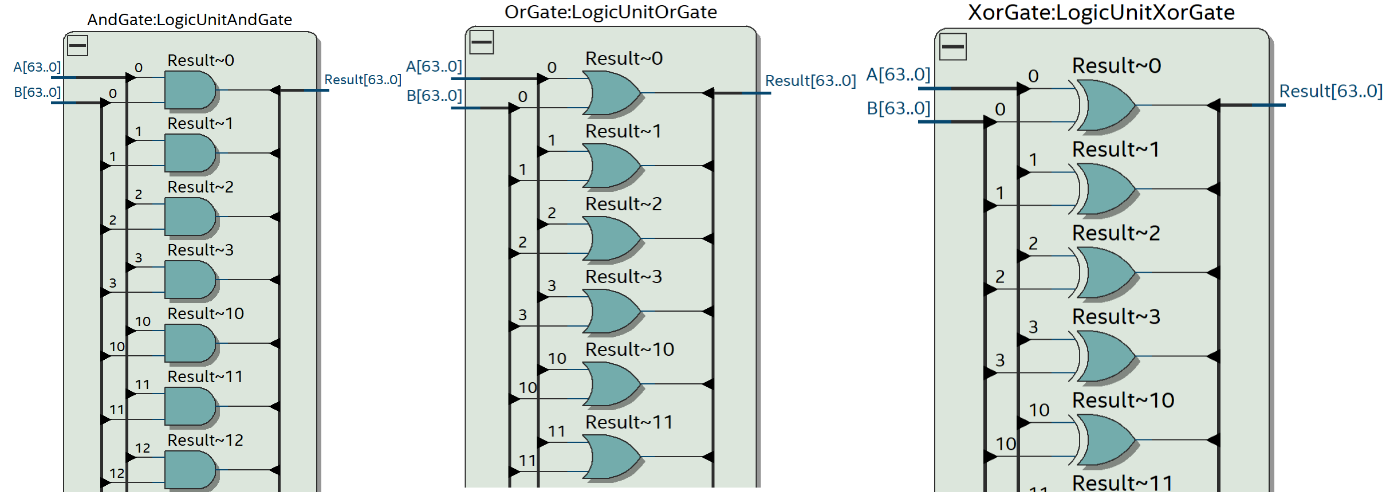


Figure 22: Synthesised Circuits of AndGates, OrGates and XorGates respectively

# Table of Tables

[Table 1: Truth Table of LogicUnit 3](#_Toc37015641)

# Table of Figures

[Figure 1: Block Diagram of LogicUnit Circuit 3](#_Toc37015619)

[Figure 2: VHDL Interface of LogicUnit 3](#_Toc37015620)

[Figure 3: Functional Simulation for LogicUnit from t=0 to t=70ns 4](#_Toc37015621)

[Figure 4: Functional Simulation for LogicUnit from measurement #458 to measurement #464 4](#_Toc37015622)

[Figure 5: Functional Simulation for LogicUnit from measurement #770 to measurement #772 5](#_Toc37015623)

[Figure 6: Synthesised Circuit of LogicUnit 6](#_Toc37015624)

[Figure 7: Timing Simulation for LogicUnit of Measurement #1 7](#_Toc37015625)

[Figure 8: Timing Simulation for LogicUnit from measurement #400 to measurement #416 7](#_Toc37015626)

[Figure 9: Timing Simulation for LogicUnit of Measurement #772 8](#_Toc37015627)

[Figure 10: Block diagram of the ArithUnit 9](#_Toc37015628)

[Figure 11: VHDL interface of the ArithUnit 9](#_Toc37015629)

[Figure 12: Block diagram of the Adder 10](#_Toc37015630)

[Figure 13: VHDL interface of the Adder 10](#_Toc37015631)

[Figure 14: Functional Simulation of the Arithmetic Unit 11](#_Toc37015632)

[Figure 15: RTL synthesized circuit of the ArithUnit 12](file:///C:\Users\Jin-Desktop\Documents\Projects\ensc350-finalproject\ExU\Documentation\FP1-Report-G47-350-1202.docx#_Toc37015633)

[Figure 16: Panned-out view of the RTL synthesized circuit of the ArithUnit 13](file:///C:\Users\Jin-Desktop\Documents\Projects\ensc350-finalproject\ExU\Documentation\FP1-Report-G47-350-1202.docx#_Toc37015634)

[Figure 17: RTL synthesized circuit diagram of Adder 14](#_Toc37015635)

[Figure 18: TIming Simulation for the ArithUnit 15](file:///C:\Users\Jin-Desktop\Documents\Projects\ensc350-finalproject\ExU\Documentation\FP1-Report-G47-350-1202.docx#_Toc37015636)

[Figure 19: VHDL Interface and Implementation of AndGate 17](#_Toc37015637)

[Figure 20: VHDL Interface and Implementation of OrGate 17](#_Toc37015638)

[Figure 21: VHDL Interface and Implementation of XorGate 18](#_Toc37015639)

[Figure 22: Synthesised Circuits of AndGates, OrGates and XorGates respectively 18](#_Toc37015640)