ENSC 350 FINAL PROJECT PART 1 REPORT

Group 47
Choong Jin Ng | 301226977
Ryan Kiew Ruelt Yean | 301290779
Sachin Momuli | 301297150

Table of Contents

Introduction 2
LogicUnit 2
  Overview 2
  Functional Behaviour 3
  Circuit Synthesis 5
  Timing Simulations 7
ArithUnit 8
  Overview 8
  Functional Behaviour 11
  Circuit Synthesis 14
  Timing Simulations 17
Conclusion 20
Appendix 21
  Appendix A: Logic Gates 21
Table of Tables 23
Table of Figures 23

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.

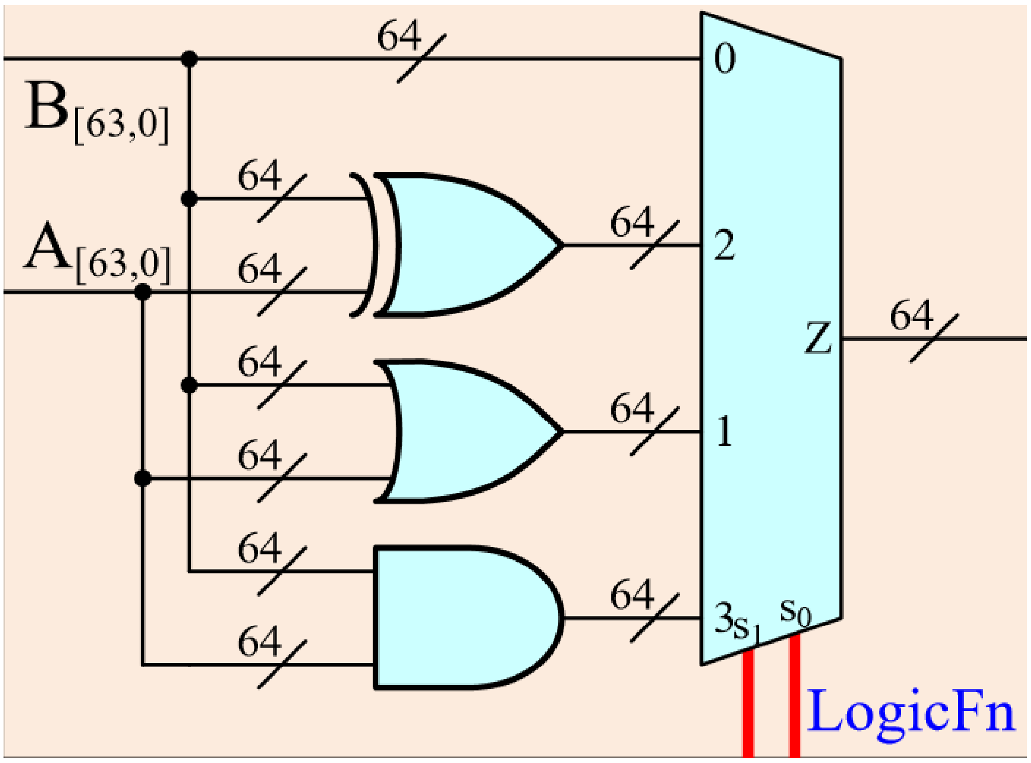
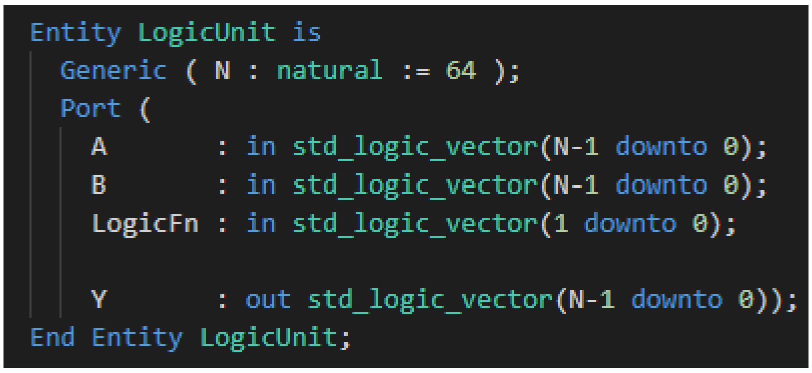


Figure 1: Block Diagram of LogicUnit Circuit

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



```
Entity LogicUnit is
    Generic ( N : natural := 64 );
    Port (
        A        : in std_logic_vector(N-1 downto 0);
        B        : in std_logic_vector(N-1 downto 0);
        LogicFn  : in std_logic_vector(1 downto 0);

        Y        : out std_logic_vector(N-1 downto 0));
End Entity LogicUnit;
```

Figure 2: VHDL Interface of LogicUnit

Functional Behaviour

The functional behaviour of LogicUnit can be demonstrated in Figure 3, Figure 4 and Figure 5. In Figure 3, the four different *LogicFn* signals are demonstrated in the first 10 measurements. Because this figure is demonstrating the functional behaviour, timing is not considered; this simulation is done as a proof of concept that our fundamental logic is correct. We performed 772 measurements, with each measurement taking about 9 ns each (illustrated well in Figure 4).

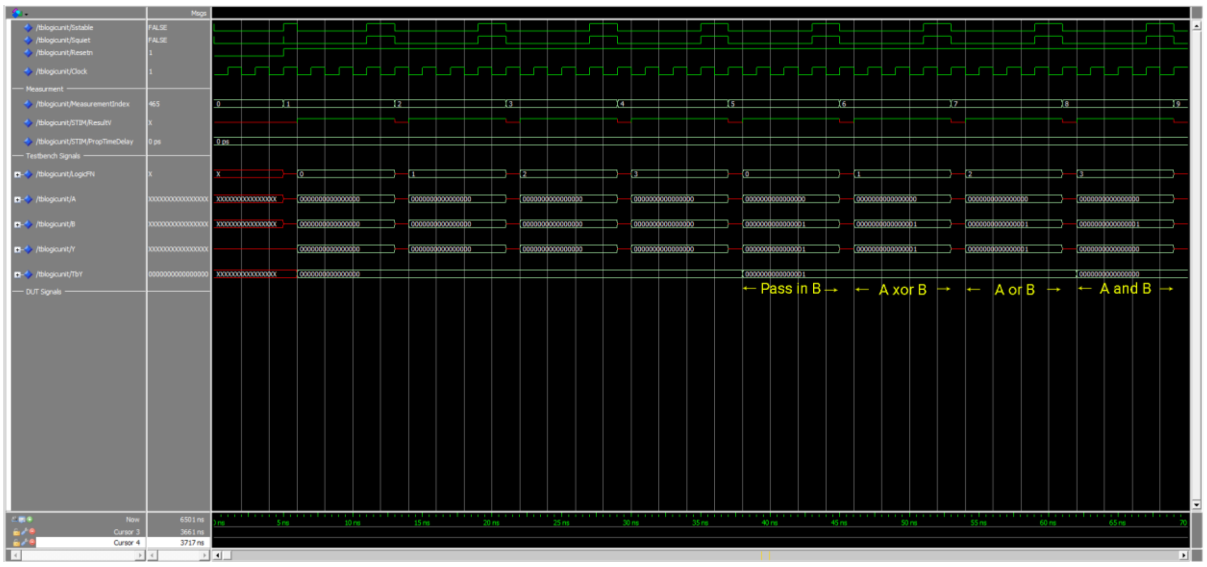


Figure 3: Functional Simulation for LogicUnit from t=0 to t=70ns

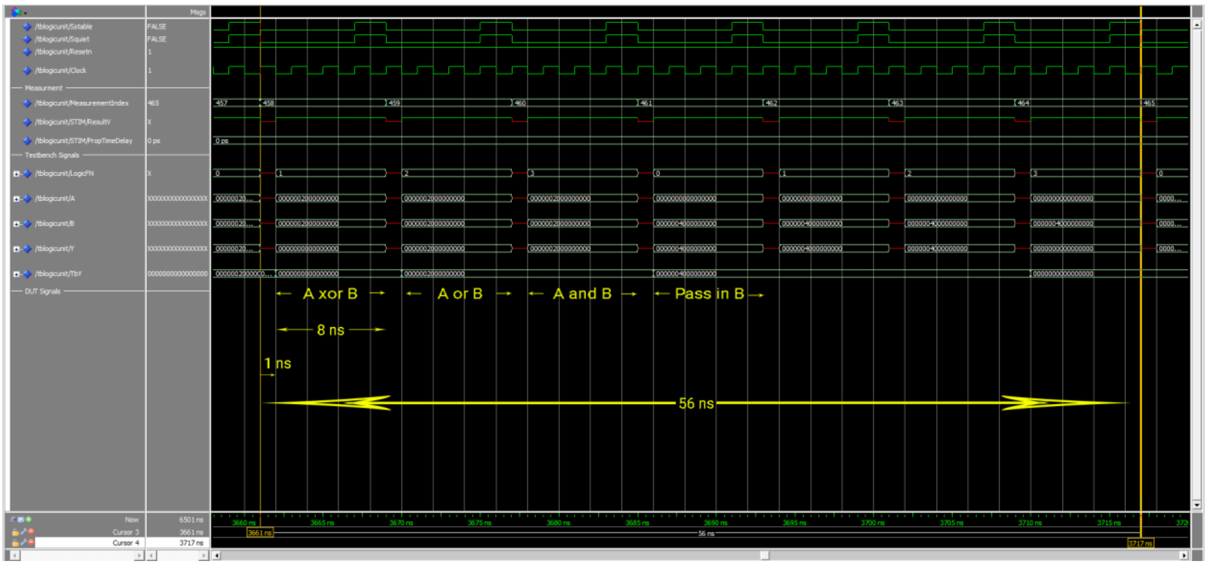


Figure 4: Functional Simulation for LogicUnit from measurement #458 to measurement #464

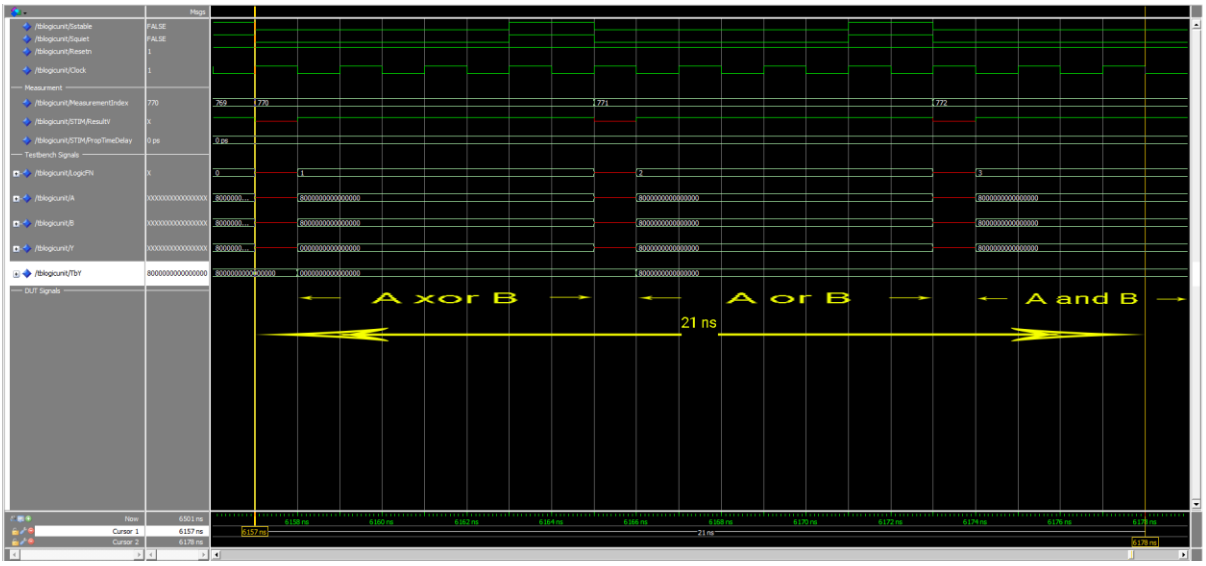

Msgs
/tblogicunit/Sstable FALSE
/tblogicunit/Squiet FALSE
/tblogicunit/Resetn 1
/tblogicunit/Clock 1
Measurment
/tblogicunit/MeasurementIndex 770
/tblogicunit/STIM/ResultV X
/tblogicunit/STIM/PropTimeDelay 0 ps
Testbench Signals
/tblogicunit/LogicFN X
/tblogicunit/A XXXXXXXXXXXXXXXX
/tblogicunit/B XXXXXXXXXXXXXXXX
/tblogicunit/Y XXXXXXXXXXXXXXXX
/tblogicunit/TbY 8000000000000000
DUT Signals
A xor B
A or B
A and B
21 ns
Now 6501 ns
Cursor 1 6157 ns
Cursor 2 6178 ns

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

Circuit Synthesis

In Figure 6, we determine there are two levels of propagation delay. The first is due to the initial logic operations. 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 if the reader wishes to learn of its implementation. The second is due to the propagation delay of passing the results from the logic gates (or *B*) through the multiplexers. This second delay is expected to be much shorter than the initial logical operations.

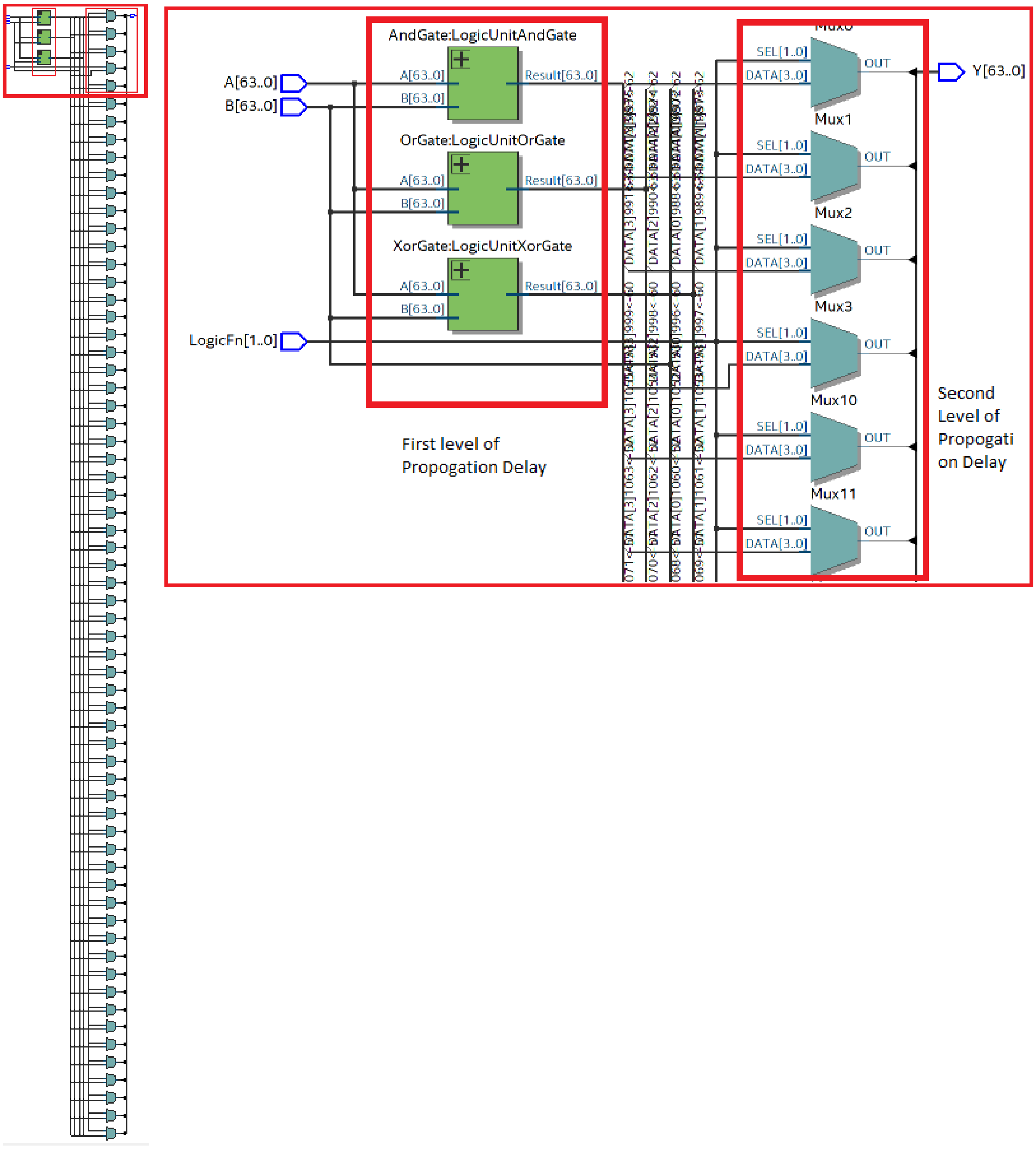


Figure 6: Synthesised Circuit of LogicUnit

Timing Simulations

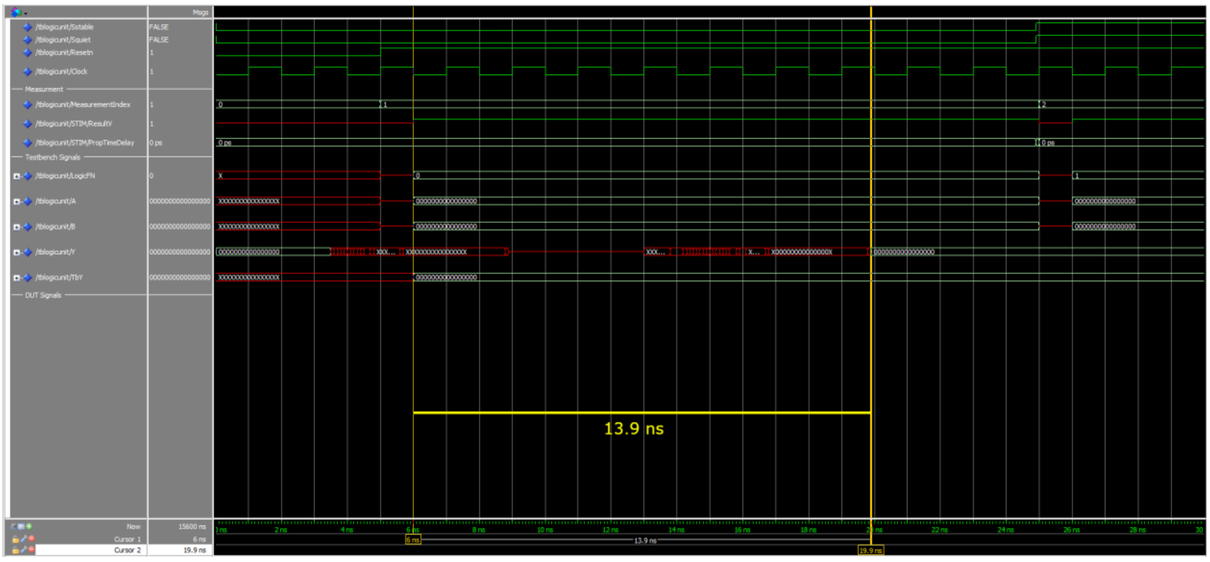


Figure 7: Timing Simulation for LogicUnit of Measurement #1

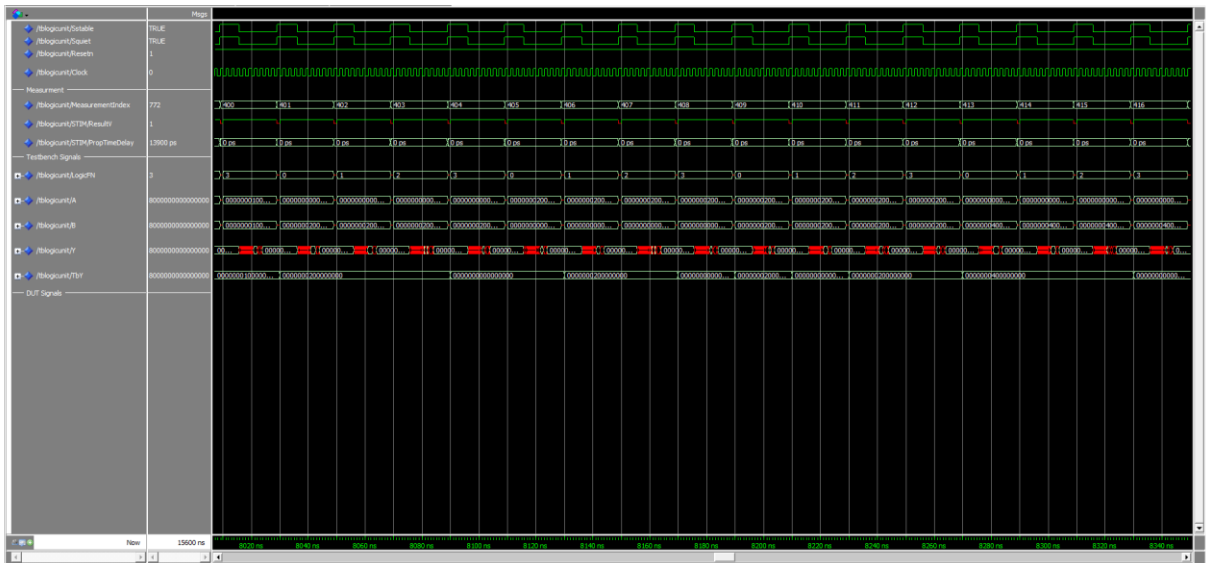


Figure 8: Timing Simulation for LogicUnit from measurement #400 to measurement #416

The timing of the LogicUnit is verified with this simulation. From Figures 7, 8, and 9, we can see that the results are obtained before the time period is up, with measurement #1 taking 13.9 ns, and measurement #772 taking 14.0 ns. For simplicity sake, we will take the propagation delay to be 14 ns.

From Figure 9, we can see that measurement #772 stabilizes at the correct result after around 10 ns, but it starts to fluctuate again due to a single bit having an undefined behaviour. This behaviour was observed on almost all the test cases. The computation is still happening during the 'fluctuation' phase despite it the correct answer 'calculated' early. This is why we wait for the results to settle before reading the answer; not taking this into account may lead to garbage results due to timing.

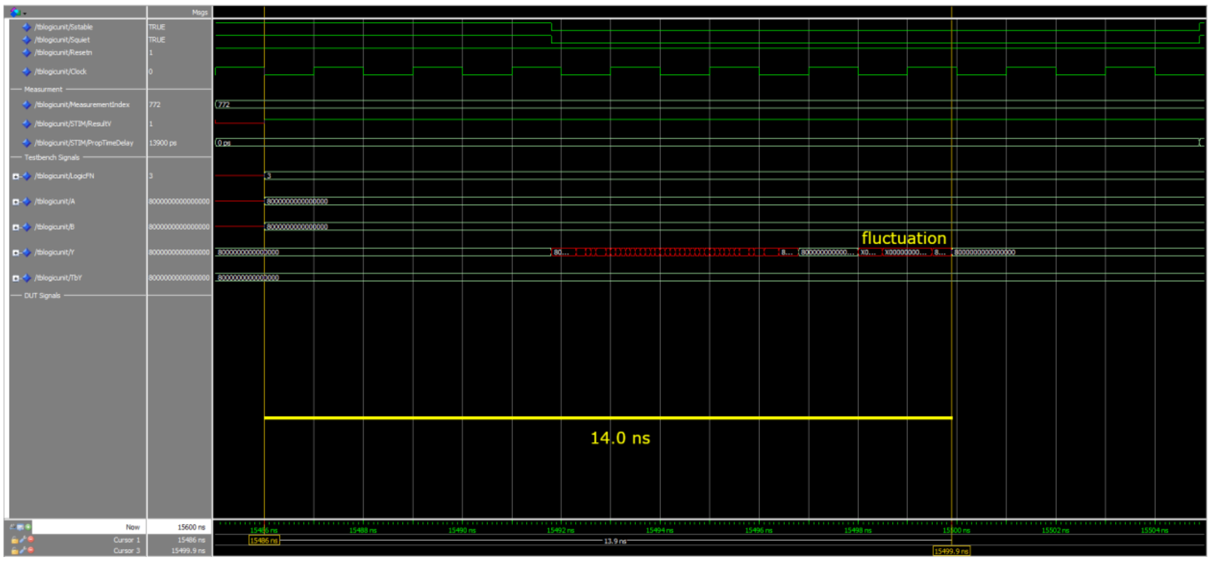

Msgs
/tblogicunit/Sstable TRUE
/tblogicunit/Squiet TRUE
/tblogicunit/Resetn 1
/tblogicunit/Clock 0
Measurment
/tblogicunit/MeasurementIndex 772
/tblogicunit/STIM/ResultV 1
/tblogicunit/STIM/PropTimeDelay 13900 ps
Testbench Signals
/tblogicunit/LogicFN 3
/tblogicunit/A 8000000000000000
/tblogicunit/B 8000000000000000
/tblogicunit/Y 8000000000000000
/tblogicunit/TbY 8000000000000000
DUT Signals
fluctuation
14.0 ns
Now 15600 ns
Cursor 1 15486 ns
Cursor 3 15499.9 ns

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.

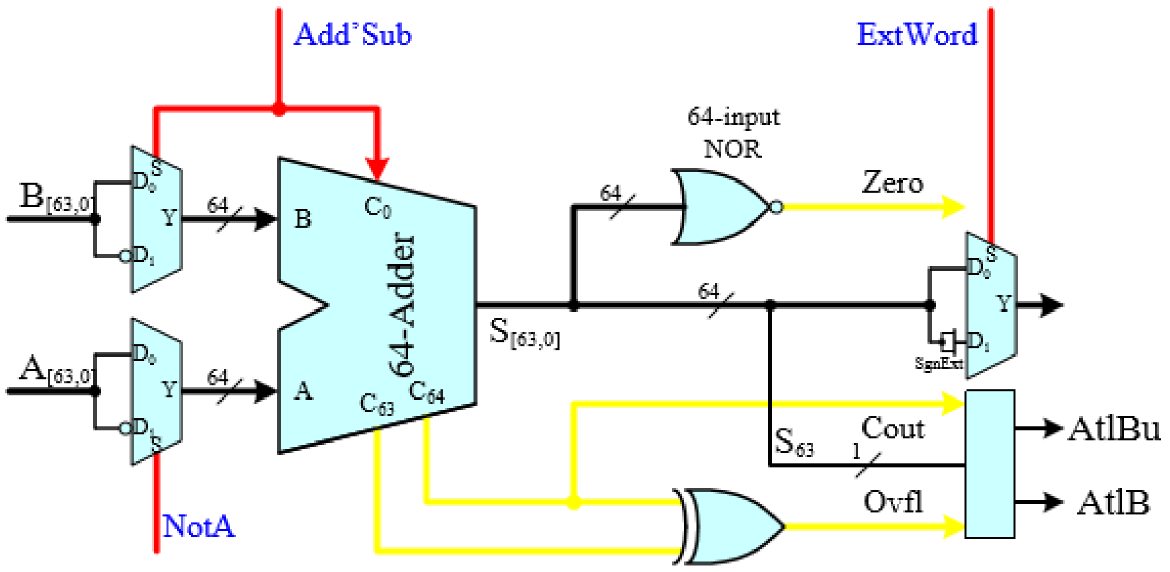
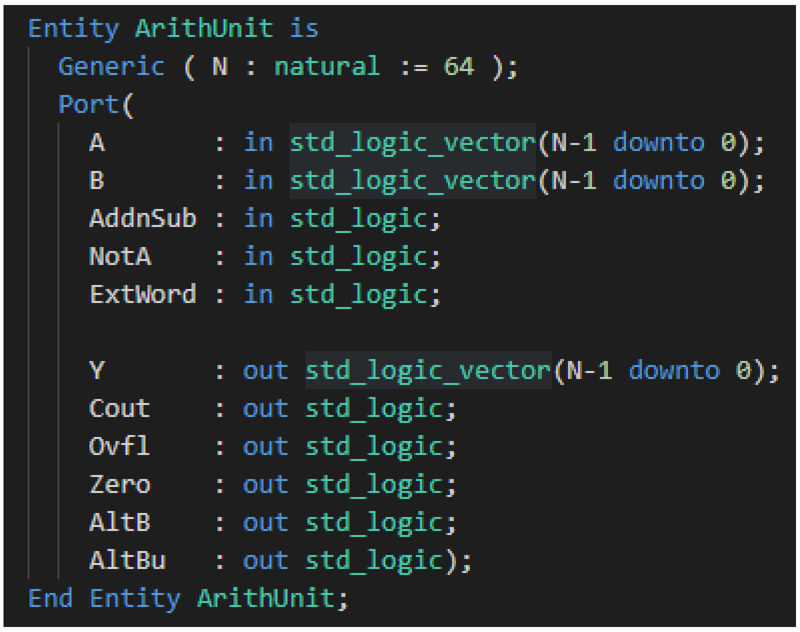


Figure 10: Block diagram of the ArithUnit



```
Entity ArithUnit is
  Generic ( N : natural := 64 );
  Port(
    A        : in std_logic_vector(N-1 downto 0);
    B        : in std_logic_vector(N-1 downto 0);
    AddnSub  : in std_logic;
    NotA     : in std_logic;
    ExtWord  : in std_logic;

    Y        : out std_logic_vector(N-1 downto 0);
    Cout     : out std_logic;
    Ovfl     : out std_logic;
    Zero     : out std_logic;
    AltB     : out std_logic;
    AltBu    : out std_logic);
End Entity 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.

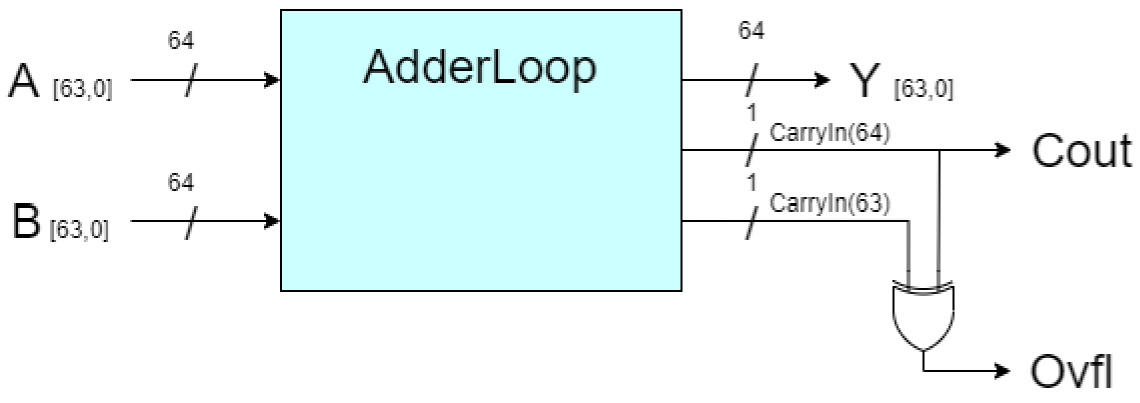
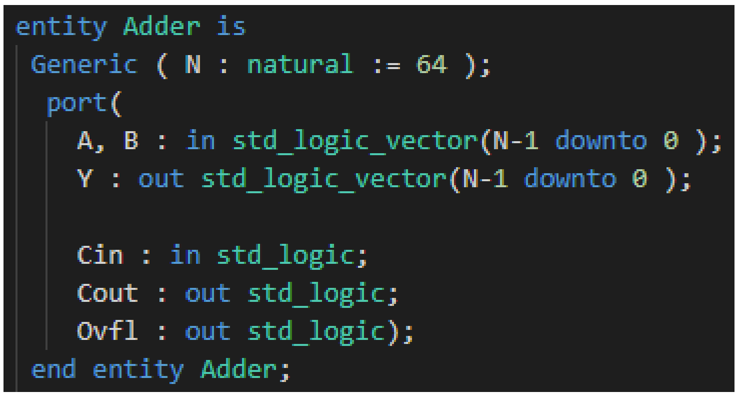


Figure 12: Block diagram of the Adder



```
entity Adder is
 Generic ( N : natural := 64 );
  port(
     A, B : in std_logic_vector(N-1 downto 0 );
     Y : out std_logic_vector(N-1 downto 0 );

     Cin : in std_logic;
     Cout : out std_logic;
     Ovfl : out std_logic);
 end entity Adder;
```

Figure 13: VHDL interface of the Adder

Functional Behaviour

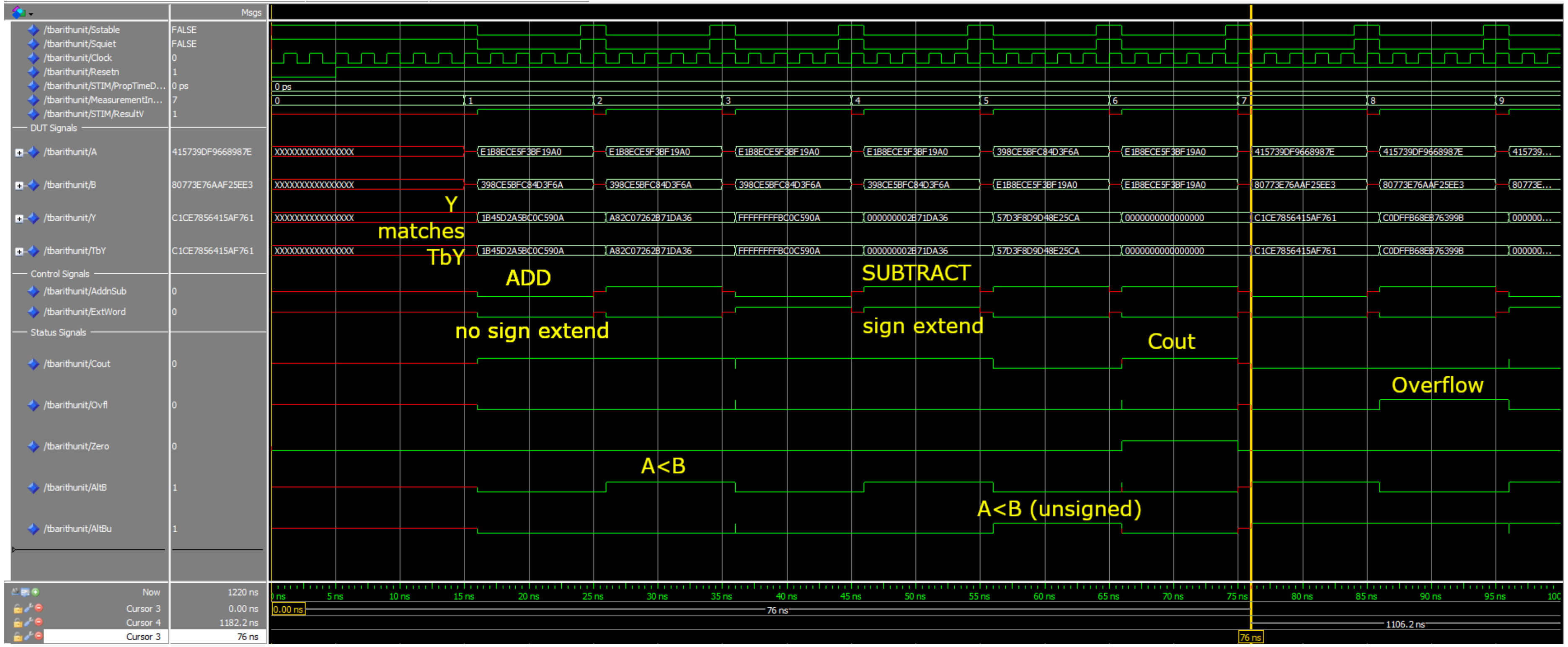
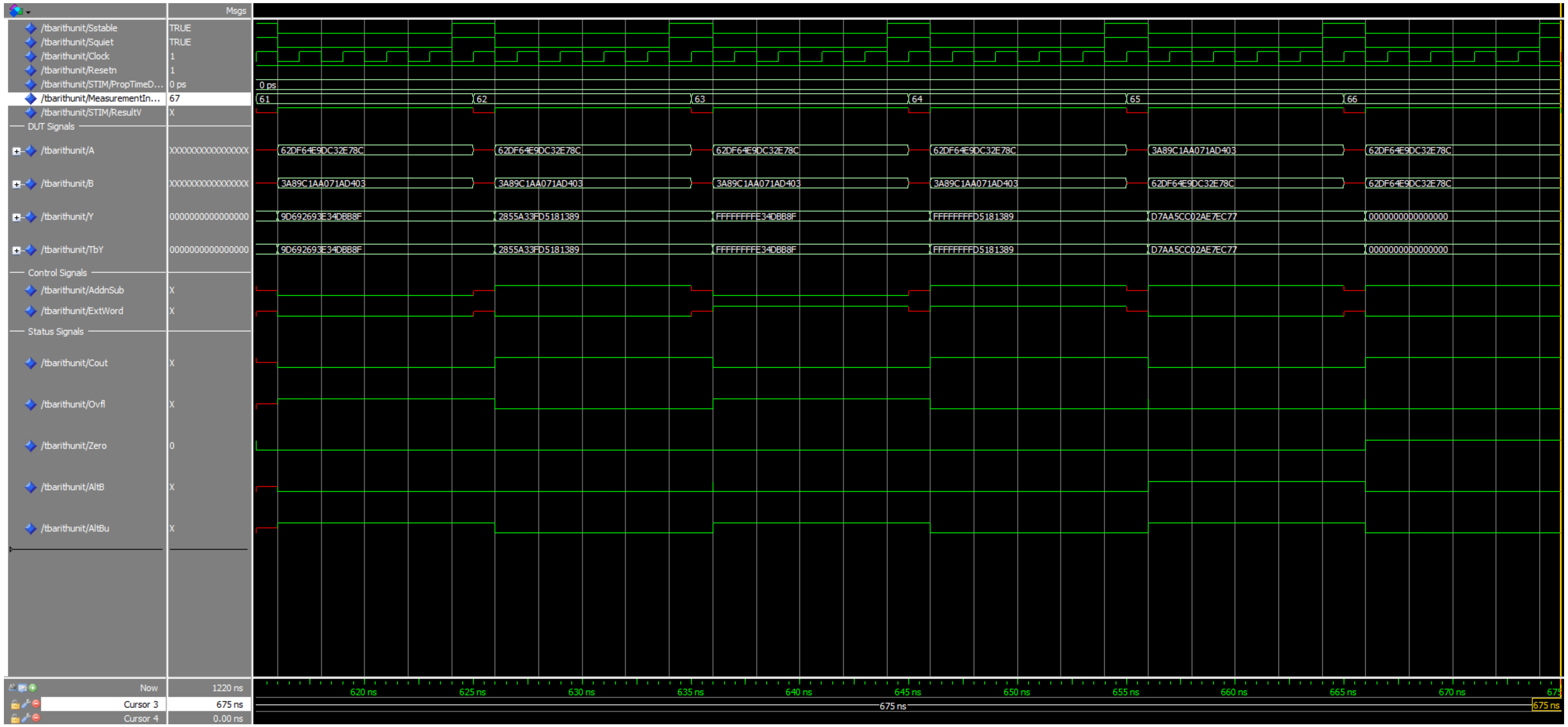


Figure 14: Functional Simulation of the ArithUnit from t = 0 to 76 ns

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.



```
                                   Msgs
/tbarithunit/Sstable               TRUE
/tbarithunit/Squiet                TRUE
/tbarithunit/Clock                 1
/tbarithunit/Resetn                1
/tbarithunit/STIM/PropTimeD...     0 ps
/tbarithunit/MeasurementIn...      67
/tbarithunit/STIM/ResultV          X
DUT Signals
/tbarithunit/A                     XXXXXXXXXXXXXXXX
/tbarithunit/B                     XXXXXXXXXXXXXXXX
/tbarithunit/Y                     0000000000000000
/tbarithunit/TbY                   0000000000000000
Control Signals
/tbarithunit/AddnSub               X
/tbarithunit/ExtWord               X
Status Signals
/tbarithunit/Cout                  X
/tbarithunit/Ovfl                  X
/tbarithunit/Zero                  0
/tbarithunit/AltB                  X
/tbarithunit/AltBu                 X

Now                                1220 ns
Cursor 3                           675 ns
Cursor 4                           0.00 ns
```

Figure 15: Functional Simulation of the ArithUnit from beginning of measurement #61 to end of measurement #66

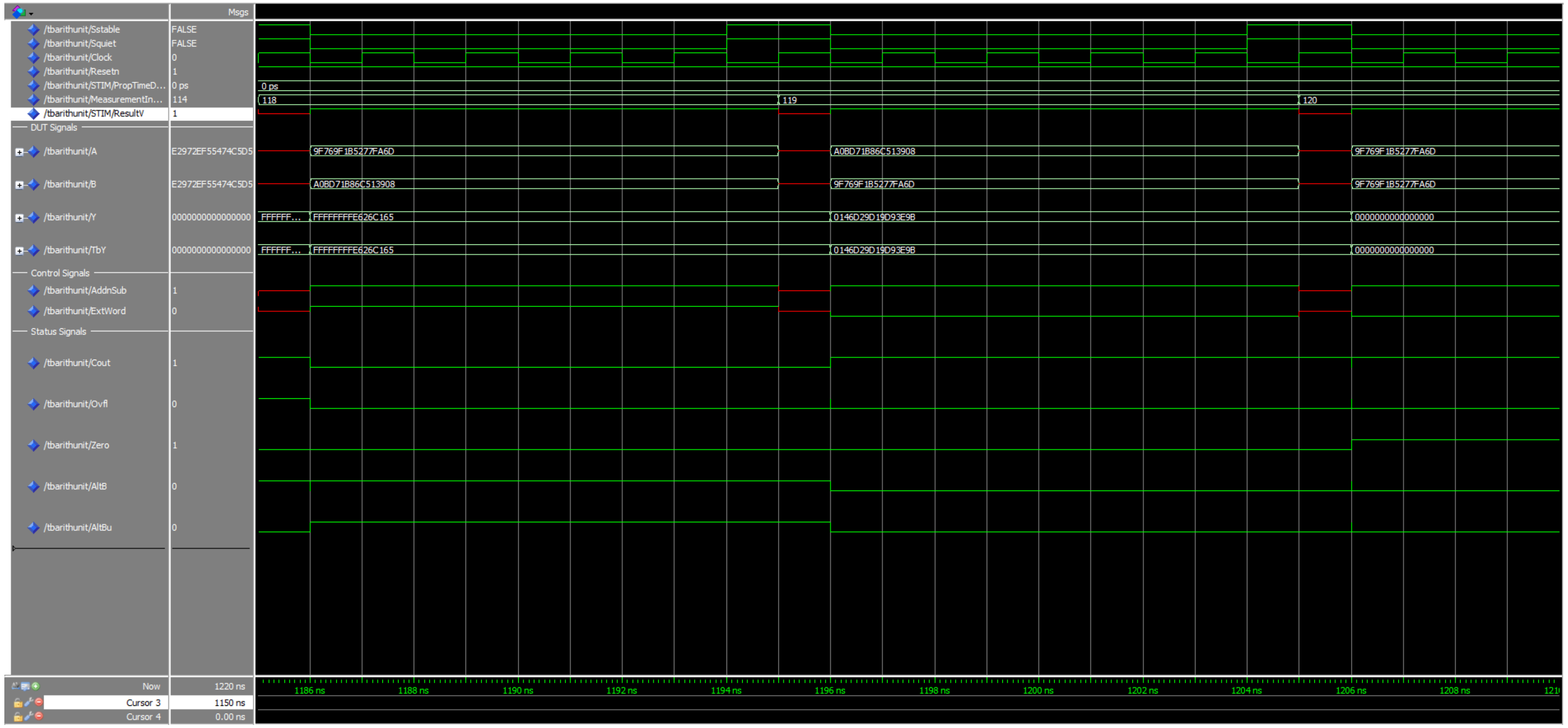


Figure 16: Functional Simulation of the ArithUnit from beginning of measurement #118 till 5 ns after the start of measurement #120

Figures 15 and 16 further goes to show that our Arithmetic Unit is functioning properly, with both Y and TbY matching each other. However, it is important to also consider the timing of the unit, as a circuit that requires too long for computation may not be feasible. We will be performing timing simulations in the next section."

Circuit Synthesis

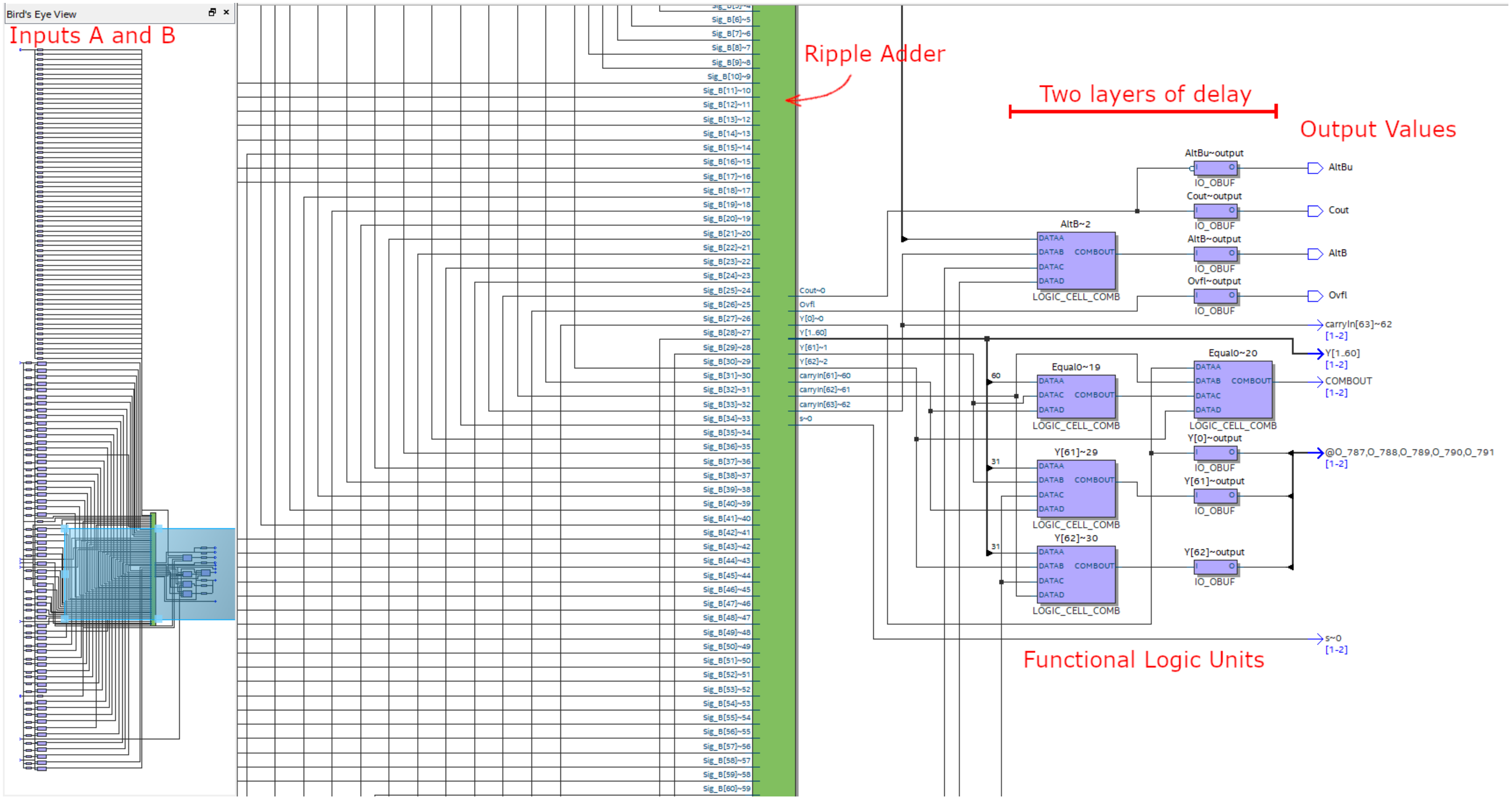


Figure 17: RTL synthesized circuit of the ArithUnit

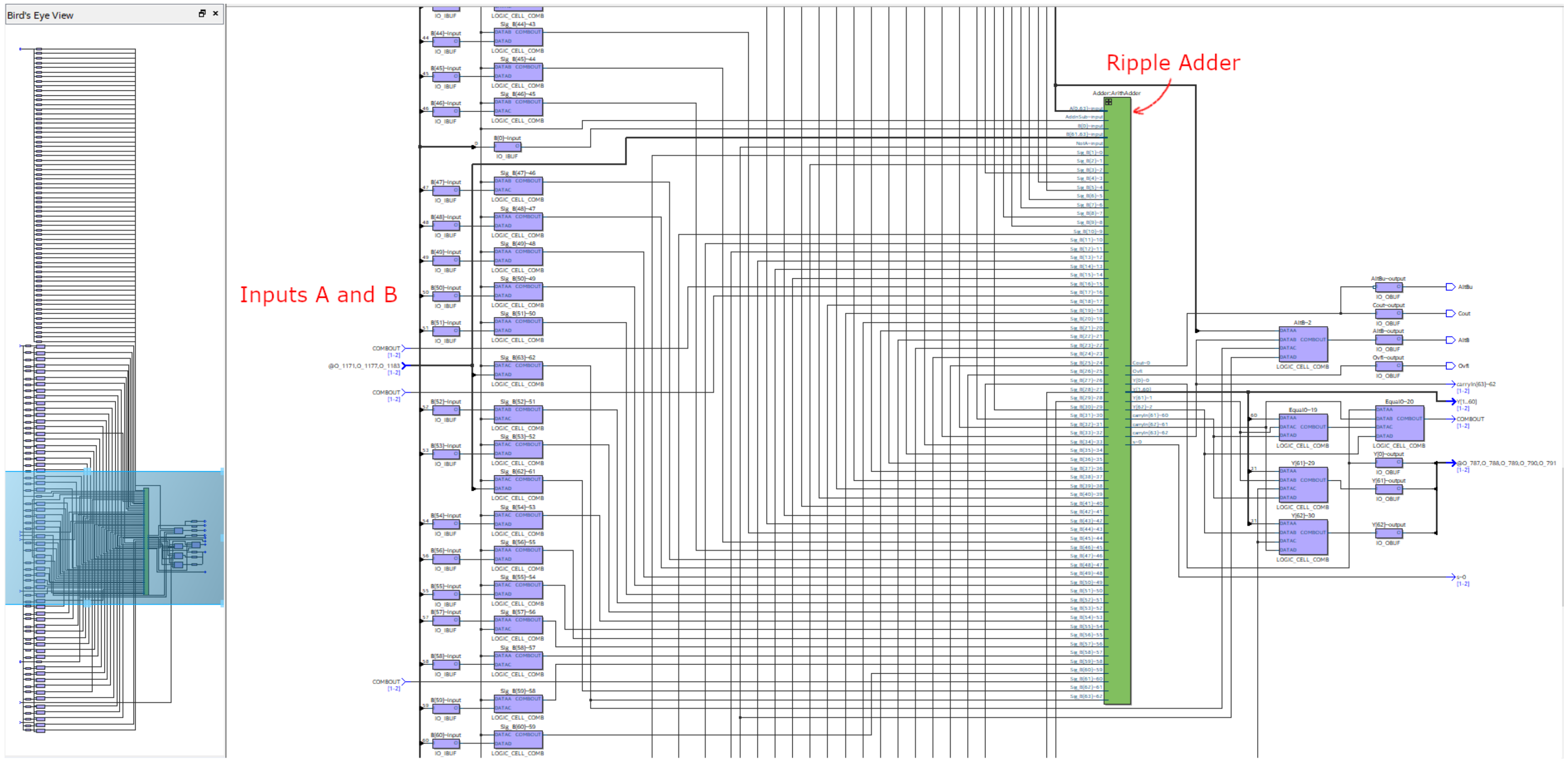


Figure 18: Panned-out view of the RTL synthesized circuit of the ArithUnit

Figures 18 and 19 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*.

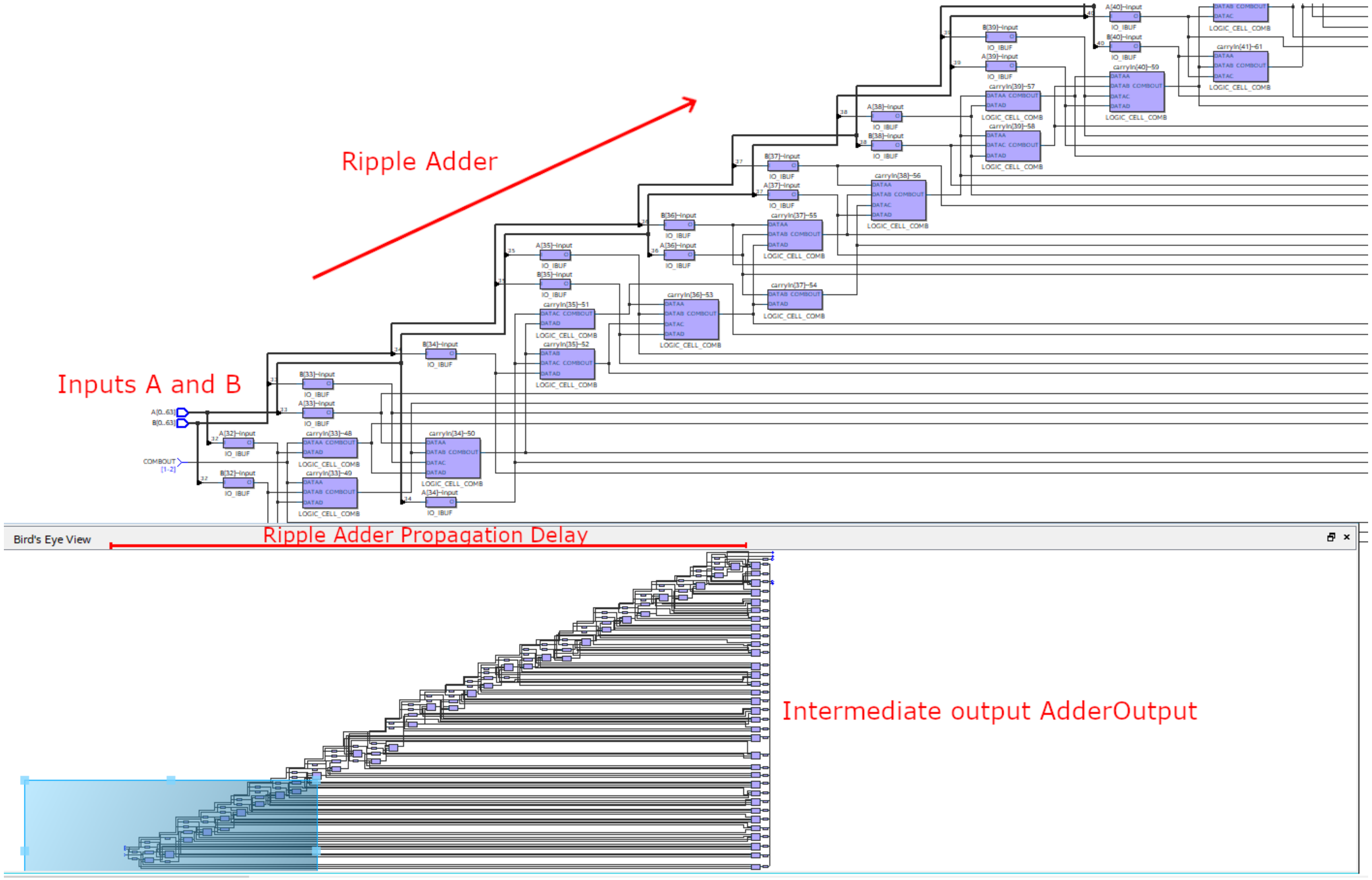


Figure 19: RTL synthesized circuit diagram of Adder

Timing Simulations

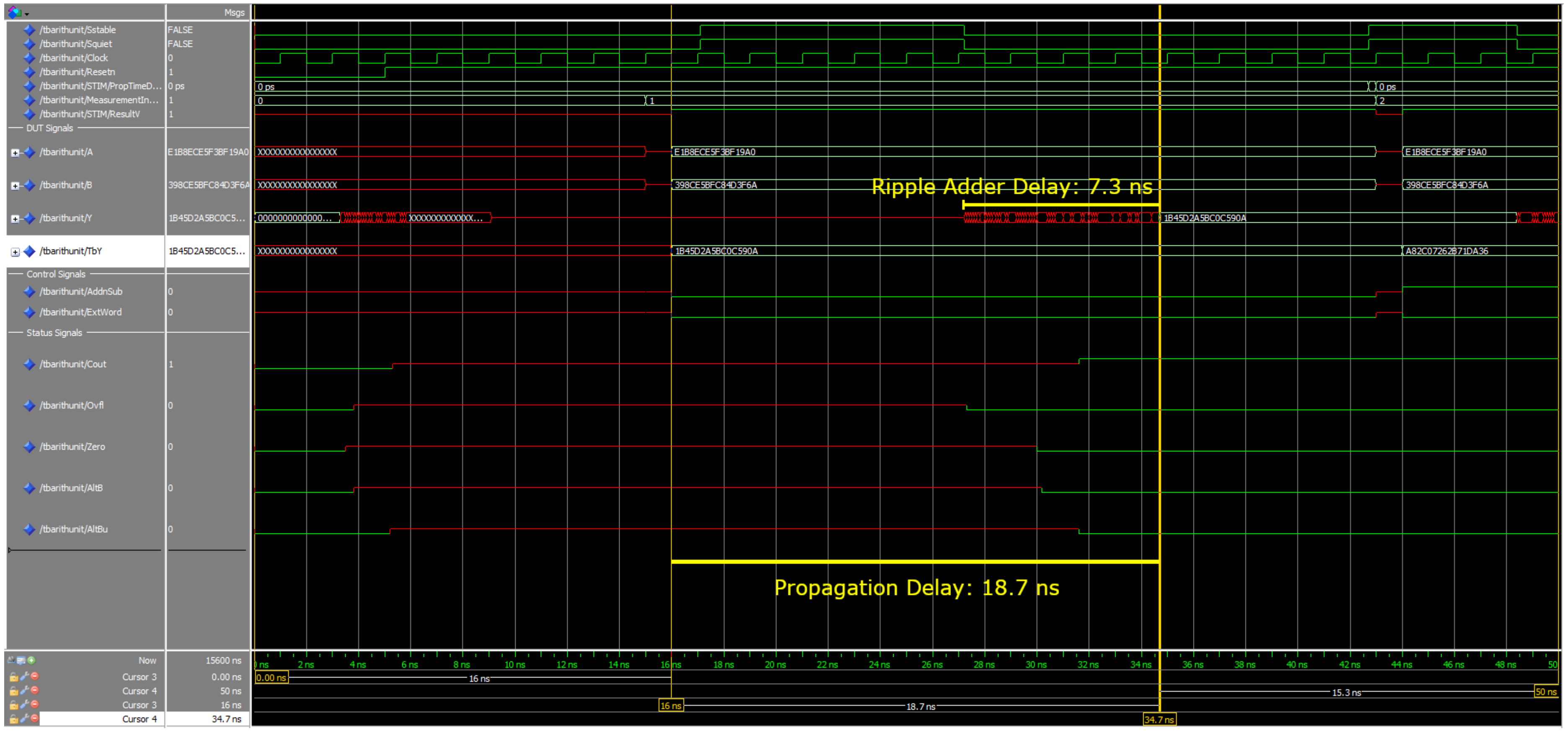
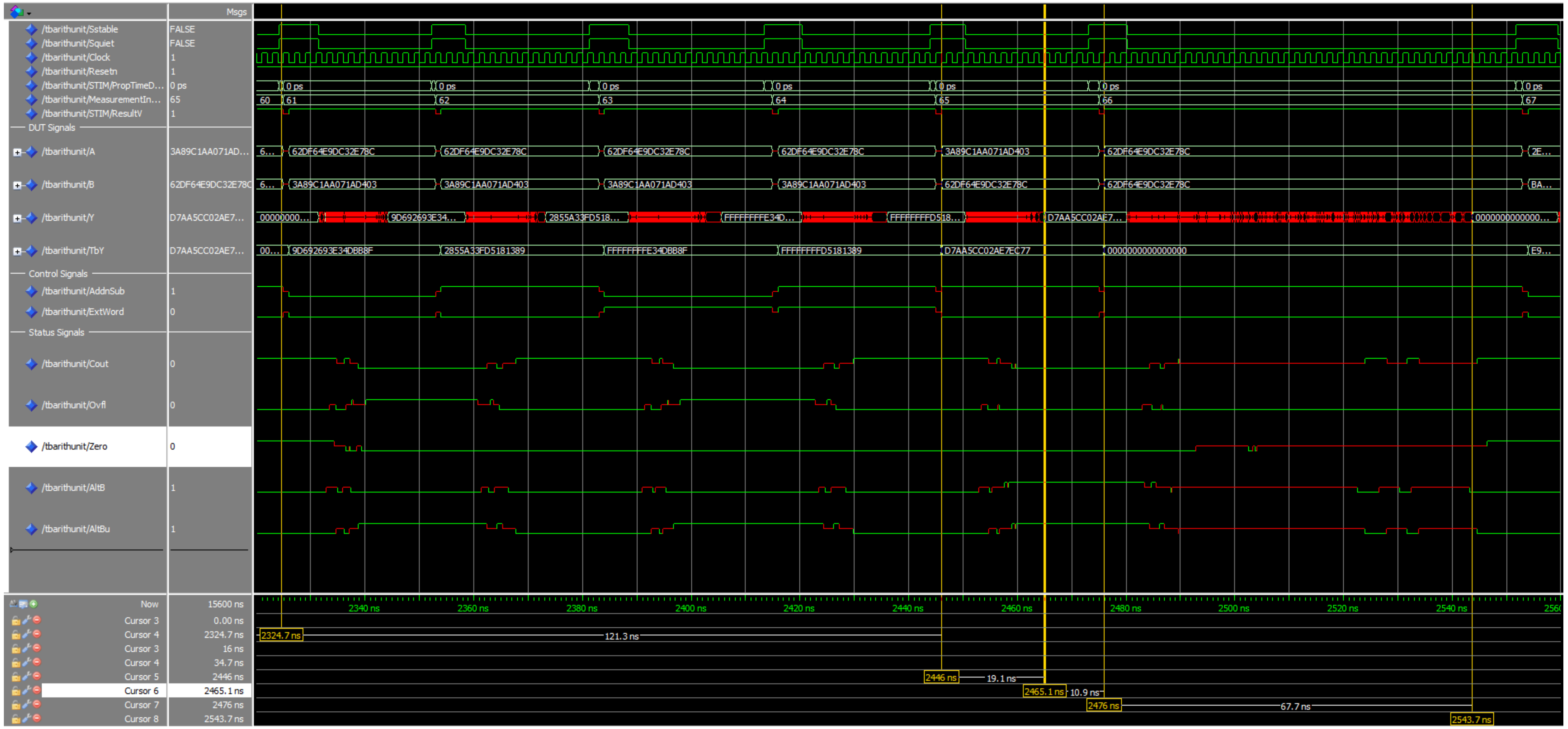


Figure 20: Timing Simulation for the ArithUnit from t = 0 to a few ns into the beginning of measurement #2

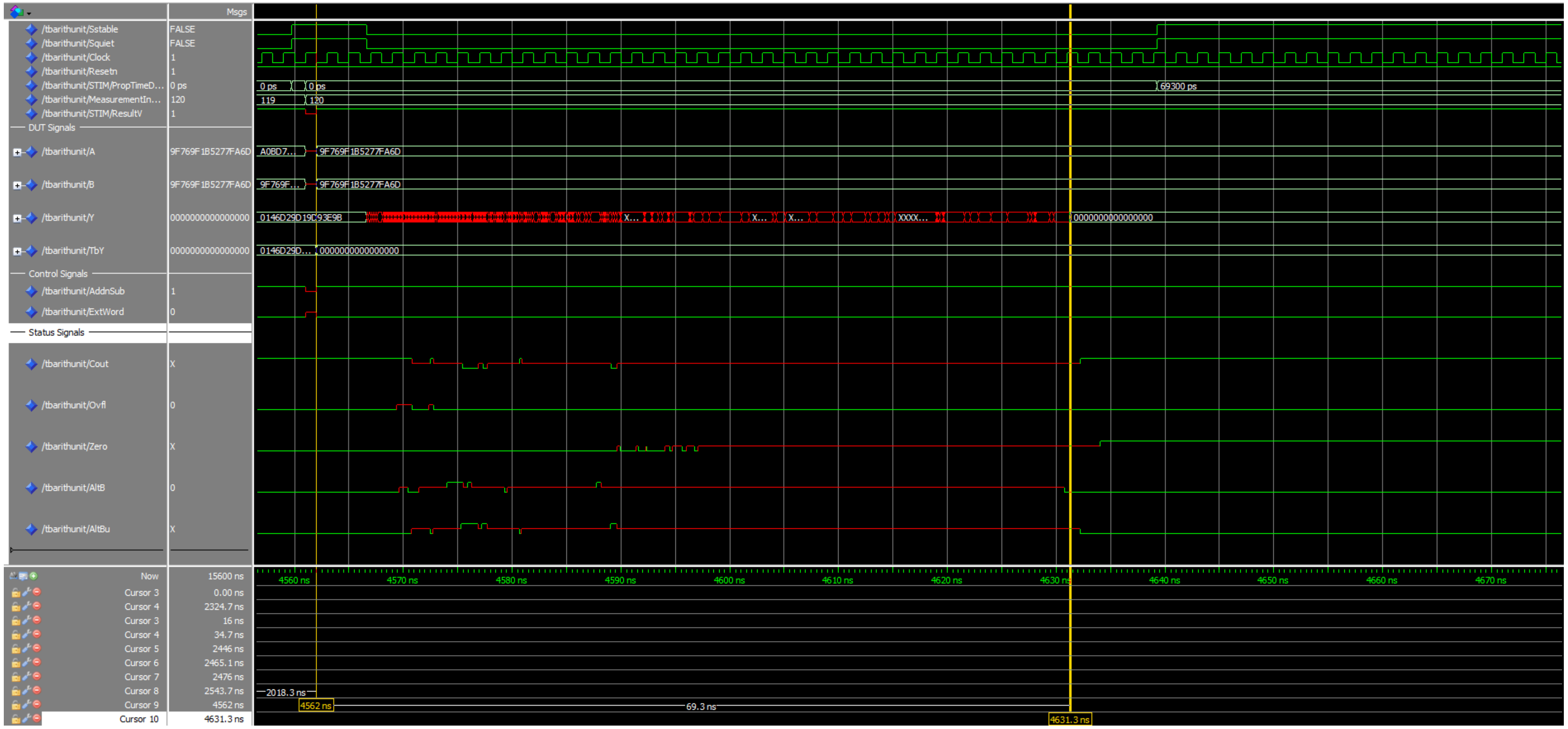
The timing of the ArithUnit can be verified by observing that the results are obtained well within the allocated time period. For measurement #1, the total propagation delay is 18.7 ns, with 8.3 ns to spare. The ripple adder took 7.3 ns to compute the appropriate result – roughly 40% of the entire time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



```
                                 Msgs
/tbarithunit/Sstable             FALSE
/tbarithunit/Squiet              FALSE
/tbarithunit/Clock               1
/tbarithunit/Resetn              1
/tbarithunit/STIM/PropTimeD...   0 ps
/tbarithunit/MeasurementIn...    65
/tbarithunit/STIM/ResultV        1
DUT Signals
/tbarithunit/A                   3A89C1AA071AD...
/tbarithunit/B                   62DF64E9DC32E78C
/tbarithunit/Y                   D7AA5CC02AE7...
/tbarithunit/TbY                 D7AA5CC02AE7...
Control Signals
/tbarithunit/AddnSub             1
/tbarithunit/ExtWord             0
Status Signals
/tbarithunit/Cout                0
/tbarithunit/Ovfl                0
/tbarithunit/Zero                0
/tbarithunit/AltB                1
/tbarithunit/AltBu               1

Now        15600 ns
Cursor 3   0.00 ns
Cursor 4   2324.7 ns
Cursor 3   16 ns
Cursor 4   34.7 ns
Cursor 5   2446 ns
Cursor 6   2465.1 ns
Cursor 7   2476 ns
Cursor 8   2543.7 ns
```

Figure 21: Timing Simulation for the ArithUnit from beginning of measurement #61 to end of measurement #66



```
                                 Msgs
/tbarithunit/Sstable             FALSE
/tbarithunit/Squiet              FALSE
/tbarithunit/Clock               1
/tbarithunit/Resetn              1
/tbarithunit/STIM/PropTimeD...   0 ps
/tbarithunit/MeasurementIn...    120
/tbarithunit/STIM/ResultV        1
DUT Signals
/tbarithunit/A                   9F769F1B5277FA6D
/tbarithunit/B                   9F769F1B5277FA6D
/tbarithunit/Y                   0000000000000000
/tbarithunit/TbY                 0000000000000000
Control Signals
/tbarithunit/AddnSub             1
/tbarithunit/ExtWord             0
Status Signals
/tbarithunit/Cout                X
/tbarithunit/Ovfl                0
/tbarithunit/Zero                X
/tbarithunit/AltB                0
/tbarithunit/AltBu               X

Now        15600 ns
Cursor 3   0.00 ns
Cursor 4   2324.7 ns
Cursor 3   16 ns
Cursor 4   34.7 ns
Cursor 5   2446 ns
Cursor 6   2465.1 ns
Cursor 7   2476 ns
Cursor 8   2543.7 ns
Cursor 9   4562 ns
Cursor 10  4631.3 ns
```

Figure 22: Timing Simulation for the ArithUnit from from beginning of measurement #120 till 5 ns after the end of measurement #120.

Conclusion

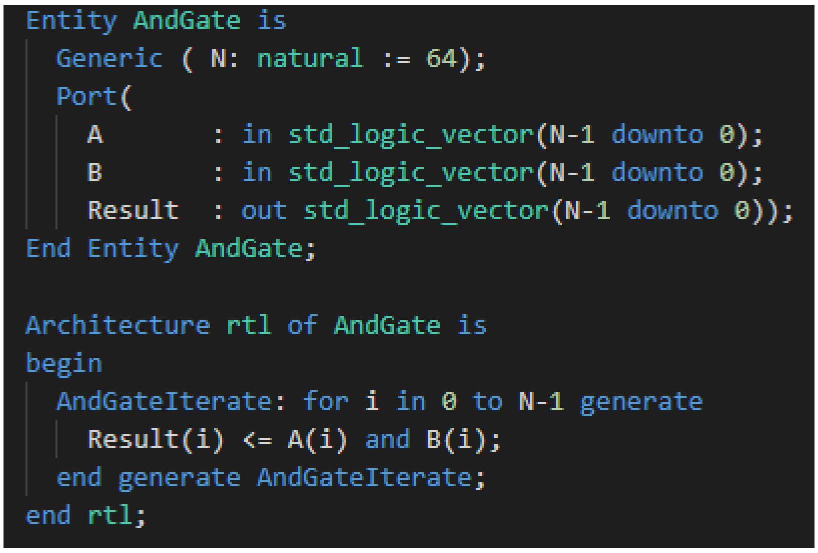
In this report, we outline two units for computational purposes and its performance. The Logic Unit determines whether bit-wise logical operations are performed while the Arithmetic Unit adds two 64-bit inputs together. These two combined will form the Arithmetic Logic Unit, the first steps to a general-purpose processor. Their propagation delay is respectively 14.0 ns and 18.7 ns.

Appendix

Appendix A: Logic Gates

Inside the LogicUnit, the logic gates.

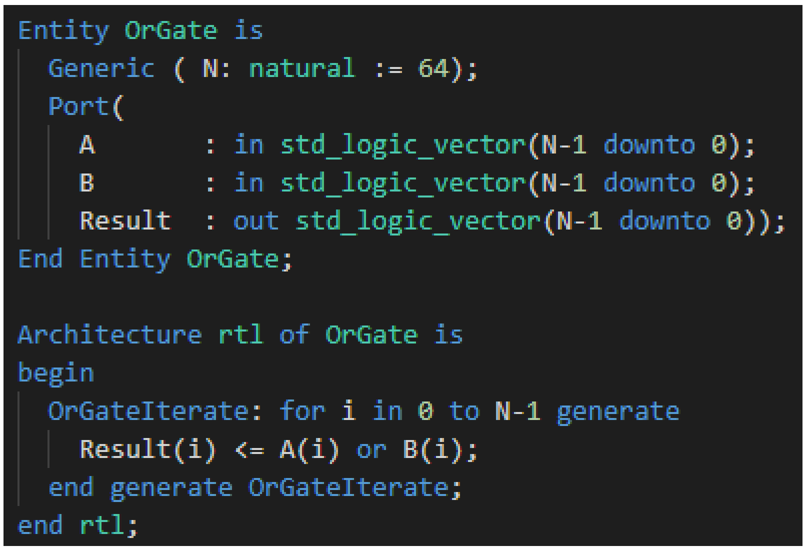
The operators *and*, *or* and *xor* are available in the IEEE standardised library. The logic gates entity interfaces and implementation for *AndGate*, *OrGate* and *XorGate* is represented in Figure 23, Figure 24 and Figure 25 respectively.



```
Entity AndGate is
  Generic ( N: natural := 64);
  Port(
    A       : in std_logic_vector(N-1 downto 0);
    B       : in std_logic_vector(N-1 downto 0);
    Result  : out std_logic_vector(N-1 downto 0));
End Entity AndGate;

Architecture rtl of AndGate is
begin
  AndGateIterate: for i in 0 to N-1 generate
    Result(i) <= A(i) and B(i);
  end generate AndGateIterate;
end rtl;
```

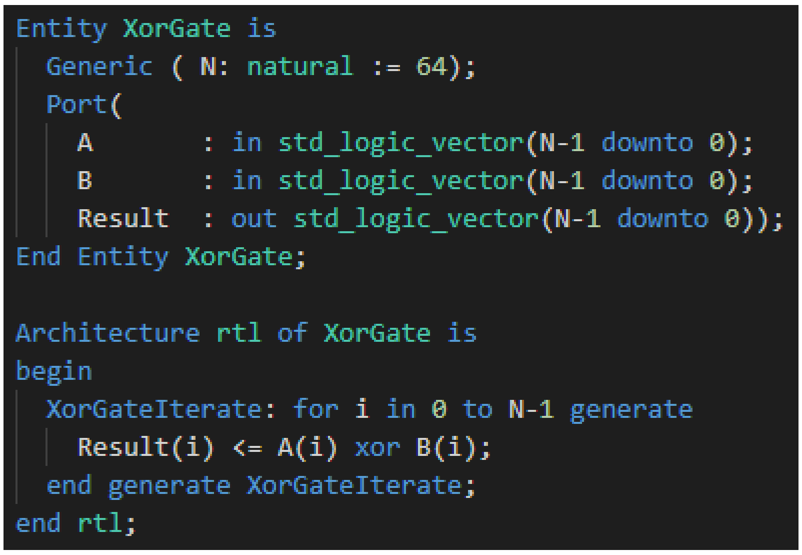
Figure 23: VHDL Interface and Implementation of AndGate



```
Entity OrGate is
  Generic ( N: natural := 64);
  Port(
    A       : in std_logic_vector(N-1 downto 0);
    B       : in std_logic_vector(N-1 downto 0);
    Result  : out std_logic_vector(N-1 downto 0));
End Entity OrGate;

Architecture rtl of OrGate is
begin
  OrGateIterate: for i in 0 to N-1 generate
    Result(i) <= A(i) or B(i);
  end generate OrGateIterate;
end rtl;
```

Figure 24: VHDL Interface and Implementation of OrGate



```vhdl
Entity XorGate is
  Generic ( N: natural := 64);
  Port(
    A       : in std_logic_vector(N-1 downto 0);
    B       : in std_logic_vector(N-1 downto 0);
    Result  : out std_logic_vector(N-1 downto 0));
End Entity XorGate;

Architecture rtl of XorGate is
begin
  XorGateIterate: for i in 0 to N-1 generate
    Result(i) <= A(i) xor B(i);
  end generate XorGateIterate;
end rtl;
```

Figure 25: VHDL Interface and Implementation of XorGate

The synthesised circuits are represented in Figure 26.

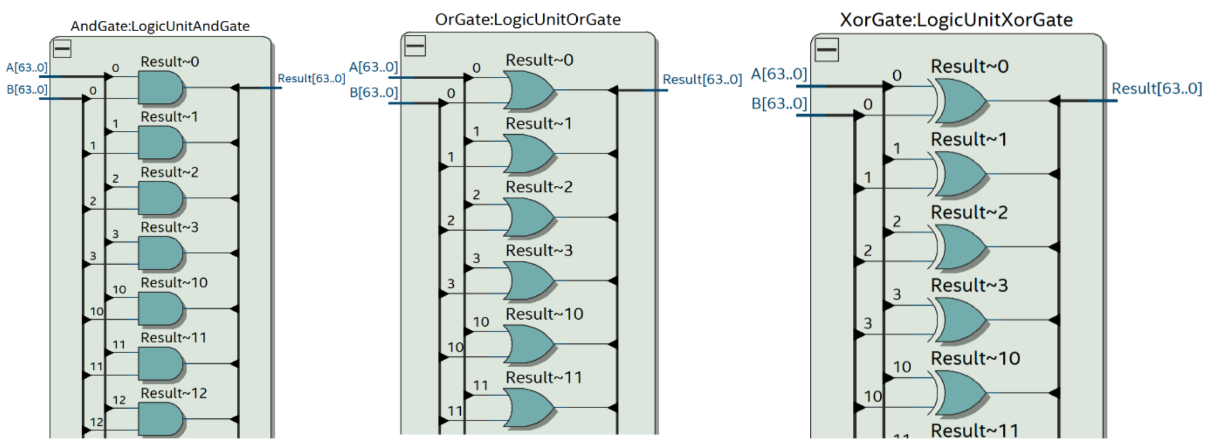


Figure 26: Synthesised Circuits of AndGates, OrGates and XorGates respectively

Table of Tables

Table 1: Truth Table of LogicUnit 3

Table of Figures

Figure 1: Block Diagram of LogicUnit Circuit 3
Figure 2: VHDL Interface of LogicUnit 3
Figure 3: Functional Simulation for LogicUnit from t=0 to t=70ns 4
Figure 4: Functional Simulation for LogicUnit from measurement #458 to measurement #464 4
Figure 5: Functional Simulation for LogicUnit from measurement #770 to measurement #772 5
Figure 6: Synthesised Circuit of LogicUnit 6
Figure 7: Timing Simulation for LogicUnit of Measurement #1 7
Figure 8: Timing Simulation for LogicUnit from measurement #400 to measurement #416 7
Figure 9: Timing Simulation for LogicUnit of Measurement #772 8
Figure 10: Block diagram of the ArithUnit 9
Figure 11: VHDL interface of the ArithUnit 9
Figure 12: Block diagram of the Adder 10
Figure 13: VHDL interface of the Adder 10
Figure 14: Functional Simulation of the ArithUnit from t = 0 to 76 ns 11
Figure 15: Functional Simulation of the ArithUnit from beginning of measurement #61 to end of measurement #66 12
Figure 16: Functional Simulation of the ArithUnit from beginning of measurement #118 till 5 ns after the start of measurement #120 13
Figure 17: RTL synthesized circuit of the ArithUnit 14
Figure 18: Panned-out view of the RTL synthesized circuit of the ArithUnit 15
Figure 19: RTL synthesized circuit diagram of Adder 16
Figure 20: Timing Simulation for the ArithUnit from t = 0 to a few ns into the beginning of measurement #2 17
Figure 21: Timing Simulation for the ArithUnit from beginning of measurement #61 to end of measurement #66 18
Figure 22: Timing Simulation for the ArithUnit from from beginning of measurement #120 till 5 ns after the end of measurement #120. 19
Figure 23: VHDL Interface and Implementation of AndGate 21
Figure 24: VHDL Interface and Implementation of OrGate 21
Figure 25: VHDL Interface and Implementation of XorGate 22
Figure 26: Synthesised Circuits of AndGates, OrGates and XorGates respectively 22