Index

Part 1: Circuit implementation

We begin this report with a discussion of our implemented code. We will develop the methodology and purpose of each individual unit and the components that describe them. A basic circuit diagram will accompany each description in addition to a table indicating the function of each input and output signal.

1.1 Adder:

Reference name(s): Adder.vhd

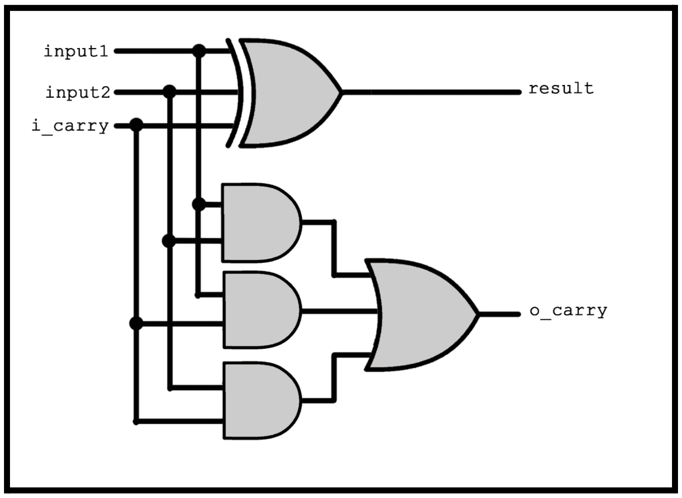


Figure 1: Basic adder diagram with labeled internal signals

The functional behavior of the adder takes the two input bits as well as the input carry and XOR's these inputs in order to determine the result of that addition. The output carry is then created by using the following logic gates: (input1 AND input2) OR (carry_in AND input1) OR (carry_in AND input2). The result of each addition is created through an XOR gate using the inputs and previous carry. A generated statement creates these adders for all bits in the addition, and the signals are then passed to the output of the entity.

Signal	**Description**
A	Input signal – first vector being added
B	Input signal – second vector being added
Cin	Input signal – carry in signal
Y	Output signal – result of the addition between A and B
Cout	Output signal – carry out signal
Ovfl	Output signal – overflow flag to show arithmetic overflow

Table 1: VHDL Interface for Adder.vhd

1.2 Arithmetic Unit:

Reference name(s): ArithUnit.vhd

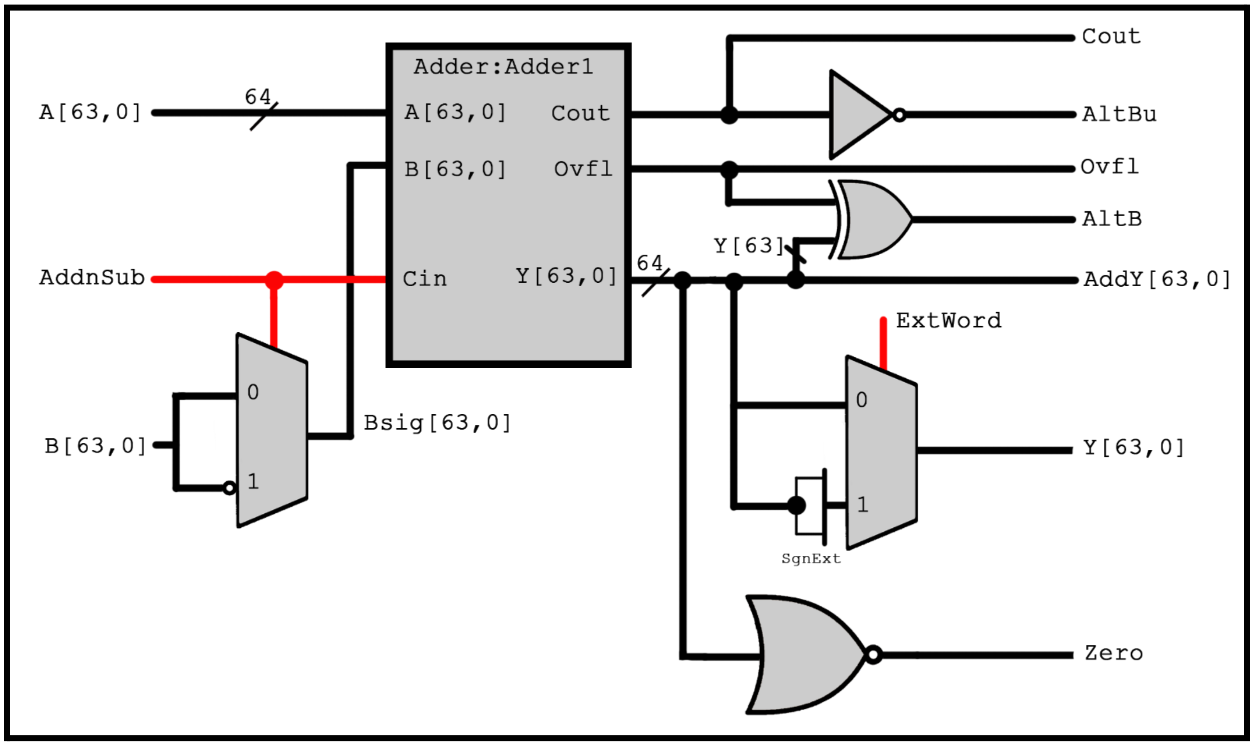


Figure 2: Basic arithmetic unit diagram with labeled internal signals

The functional behavior of the arithmetic unit takes the two input registers and performs arithmetic operations on them through the use of the 64 bit adder entity, while also performing sign extensions for words as well as outputting the carry and overflow flag signals. The circuit takes the input registers A and B and passes them through to the adding unit. If a subtraction occurs, the contents from the B register are converted into their negative 1s complement counterpart before entering the adder alongside an input carry. After the addition, a sign extension operation is performed on the result if necessary, and the result is passed to the output along with the generated status signals. The result also passes through a 64 input NOR gate and outputs a zero flag signal if the result is 0.

Signal	**Description**
A	Input signal – first vector used for the arithmetic operation
B	Input signal – second vector for arithmetic operation
AddnSub	Input signal – control signal to determine if addition or subtraction
ExtWord	Input signal – control signal to determine if sign extension is necessary
AddY	Output signal – transports result of adding circuit to sign extension processing and to the output Y of the circuit
Y	Output signal – resultant vector of the arithmetic operation
Cout	Output signal – resulting carry from arithmetic operation
Ovfl	Output signal – overflow flag signal
Zero	Output signal – status signal outputs the result of the 64 input NOR gate connected to the output of the adder
AltB	Output signal – status signal for SLT and BR instructions
AltBu	Output signal – status signal for SLT and BR instructions

Table 2: VHDL Interface for ArithUnit.vhd

1.3 Logic Unit:

Reference name(s): LogicUnit.vhd

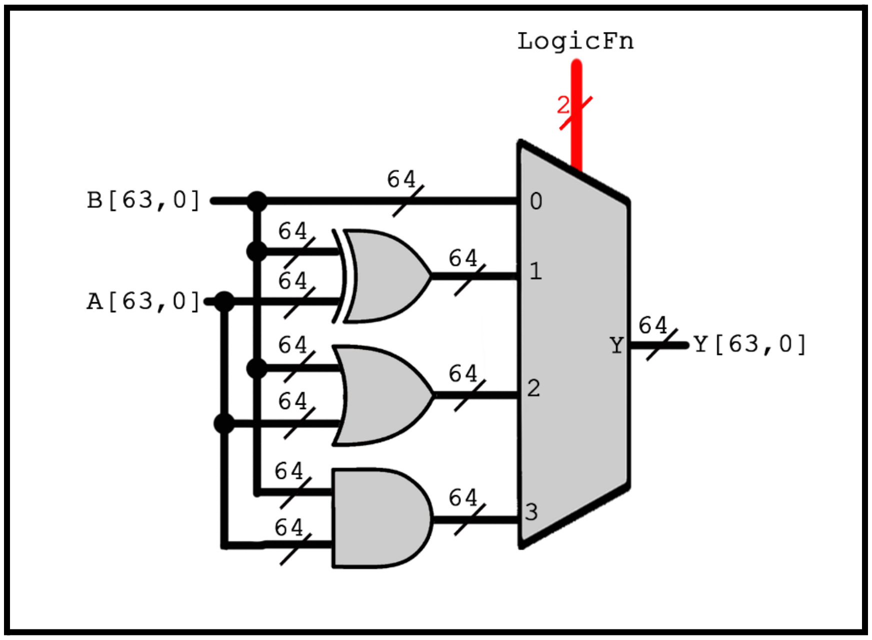


Figure 3: Basic logic unit diagram with labeled internal signals

The functional behavior of the logic unit is to perform basic logic functions on the input vectors. The circuit is simply a multiplexor based on the two bit input signal logicFN to decide whether to pass the value of B register to the output ('00'), to perform an XOR operation on vectors A and B ('01'), to perform an OR operation on vectors A and B ('10'), or finally to perform an AND operation on vectors A and B ('11'). The result of this multiplexor is then passed as the output of the logic unit.

Signal	**Description**
A	Input signal – first vector used for the logic operation
B	Input signal – second vector used for the logic operation
LogicFN	Input signal – control signal to decide the logic operation performed on the contents of A and B
Y	Output signal – resultant vector from the performed logic operation

Table 3: VHDL Interface for LogicUnit.vhd

1.4 Barrel Shifters:

Reference name(s): SLL64.vhd | SRL64.vhd | SRA64.vhd

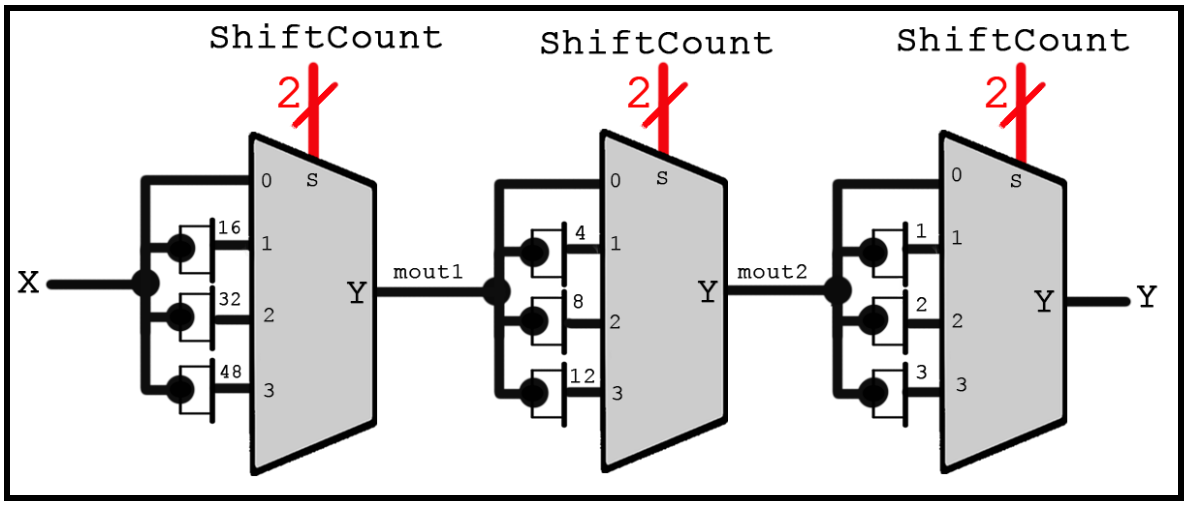


Figure 4: Basic Barrel Shifter diagram with labeled internal signals

The functional behavior of the 64 bit SRA, SRL, and SLL barrel shifting units use a sequence of three 4-1 multiplexors. The first multiplexor, uses the two most significant bits of the shift count as a control signal to decide between the four inputs; 0 shift, 16 shift, 32 shift, and 48 shift. The output of this first multiplexor is then entered into the next multiplexor where the two middle bits of the shift count are used to determine the next shift of 0, 4, 8, or 12 bits. The final multiplexor then uses the two least significant bits to shift the output of the second multiplexor 0, 1, 2, or 3 bits. The output of the third and final multiplexor is then passed through the output signal of the shifting entity.

Signal	Description
X	Input signal – the input vector being shifted
ShiftCount	Input signal – the input vector with the number of bits to be shifted
Y	Output signal – resultant vector from the performed logic operation

Table 4: VHDL Interface for SLL64.vhd | SRL64.vhd | SRA64.vhd

1.5 Shift Unit:

Reference name(s): ShiftUnit.vhd

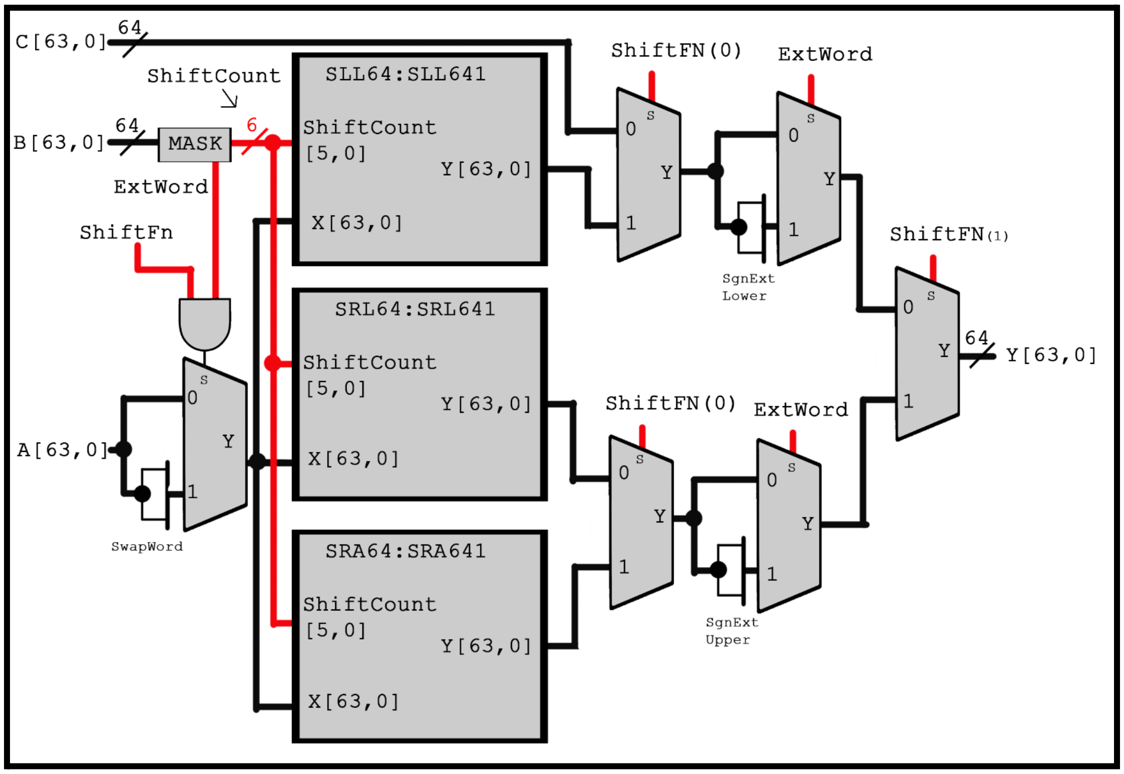


Figure 5: Basic Shifter unit diagram with labeled internal signals

The functional behavior of the shifting unit starts off by passing A through a multiplexor that will choose to swap the upper and lower 32 bits of the vector in order to perform an arithmetic shift right on a word, in all other cases the value of A is unchanged. Vector B is passed through a bit mask in order to extract the shiftcount from the input signal. These results are then passed to the SLL, SRL, and SRA barrel shifting units where the input is shifted in accordance with the shift count. The results are then passed through multiplexors controlled by control signals shiftFN(0) and ExtWord in order to select the desired outcome and perform sign extensions. Input C is passed through to the end if no shift operation is desired and an arithmetic operation occurred elsewhere in the execution unit. After filtering out the desired result, it is passed through to the output of the shift unit.

Signal	**Description**
A	Input signal – input vector being shifted
B	Input signal – input vector containing the number of the shift count
C	Input signal – input vector containing the value to be passed without being shifted
ShiftFN	Input signal – control signal to determine the shift operation being performed
ExtWord	Input signal – control signal to determine if a sign extension is necessary
Y	Output signal – result of the shifting operation performed (if any)

Table 5: VHDL Interface for ShiftUnit.vhd

1.6 Execution Unit:

Reference Name(s): ExecUnit.vhd

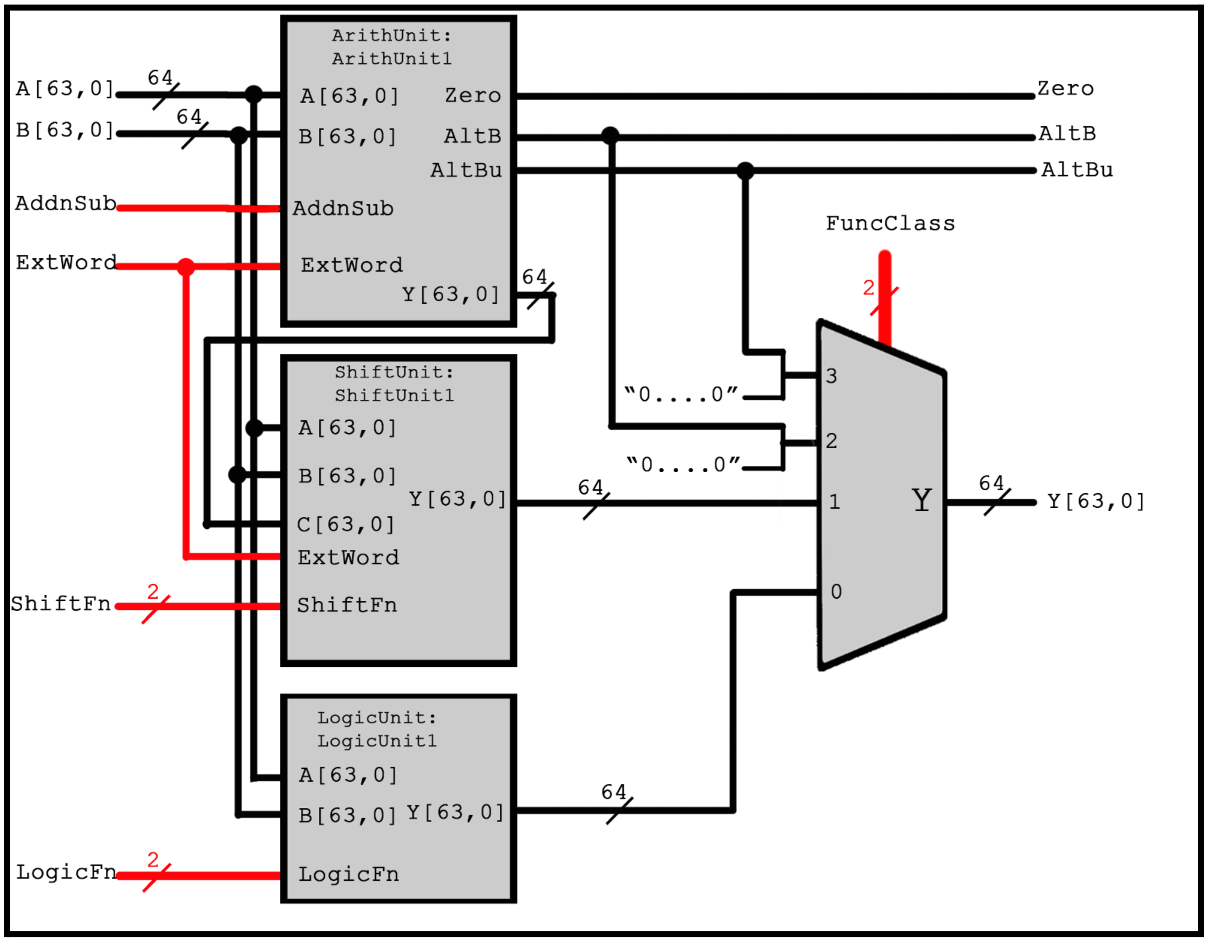


Figure 6: Basic execution unit diagram with labeled internal signals

The functional behavior of the execution unit passes the various input signals into their corresponding units in order to compute a result. The input signals are port mapped to each component described above, with the outputs of each component collected into a multiplexor using the funcClass control signal as the select signal in order to pass the desired result to the output of the execution unit.

Signal	Description
A	Input signal – input signal from register A
B	Input signal – input signal from register B
FuncClass	Input signal – control signal to determine function being performed
LogicFN	Input signal – control signal to determine logic operation to be performed
ShiftFN	Input signal – control signal to determine the shift operation being performed
AddnSub	Input signal – control signal to determine if addition or subtraction arithmetic used
ExtWord	Input signal – control signal to determine if word extensions are necessary
Y	Output signal – final output from operations performed in execution unit
Zero	Output signal – status signal to determine if arithmetic operation equals zero
AltB	Output signal – status signal for SLT and BR instructions (signed)
AltBu	Output signal – status signal for SLT and BR instructions (unsigned)

Table 6: VHDL Interface for ExecUnit.vhd

Part 2: Functional Simulation

Having completed the circuit implementation, we now discuss the findings related to the functional simulation for each individual component within the execution unit. We use the functional simulation mainly as a means of confirming our implementation is correct via comparison of simulated output and expected output. As we will discuss in this section, verifying the circuit in this manner allows us to quickly confirm the validity without attempting to dissect synthesis diagrams or account for additional factors such as delay in the timing simulation.

2.1 Arithmetic Unit:

After running the functional simulation for the arithmetic unit, we were able to immediately detect an issue with the output ‘Zero’ signal due to an incorrect NOR gate implementation. After correcting this issue, we were able to confirm that every test case passed correctly. Next, we discuss a select few simulation runs demonstrating the circuit.

In figure 7, we demonstrate signals Cout, Ovfl, Zero, AltB, and AltBu as well as the final output Y respond correctly to control signals AddnSub, ExtWord and input registers. Looking at measurement 2 which corresponds to a subtract instruction (Y=A-B), we can look at the most significant 4 bits of each register. Registers A and B contain 0xB=’1011’ and 0x5=’0101’ respectively. From these bits alone, we can determine that upon inverting register B due to the AddnSub signal, we will receive a carry out. We confirm this by viewing a Cout=’1’ during this test. Another measurement of interest within this capture is measurement 6. Intuitively, we expect an output vector containing only 0s if given two registers with identical values and an AddnSub signal of ‘1’.

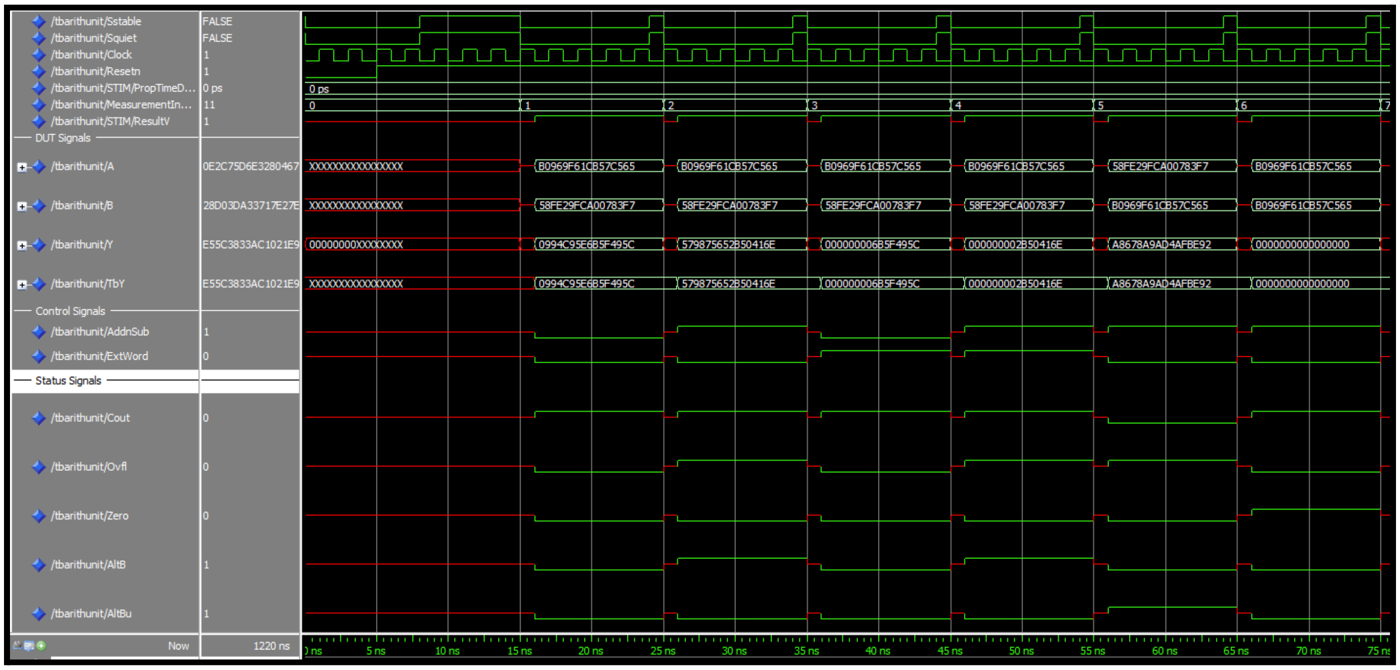

/tbarithunit/Sstable FALSE
/tbarithunit/Squiet FALSE
/tbarithunit/Clock 1
/tbarithunit/Resetn 1
/tbarithunit/STIM/PropTimeD... 0 ps
/tbarithunit/MeasurementIn... 11
/tbarithunit/STIM/ResultV 1
DUT Signals
/tbarithunit/A 0E2C75D6E3280467
/tbarithunit/B 28D03DA33717E27E
/tbarithunit/Y E55C3833AC1021E9
/tbarithunit/TbY E55C3833AC1021E9
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 1220 ns
0 ps
0 1 2 3 4 5 6 7
XXXXXXXXXXXXXXXX B0969F61CB57C565 B0969F61CB57C565 B0969F61CB57C565 B0969F61CB57C565 58FE29FCA00783F7 B0969F61CB57C565
XXXXXXXXXXXXXXXX 58FE29FCA00783F7 58FE29FCA00783F7 58FE29FCA00783F7 58FE29FCA00783F7 B0969F61CB57C565 B0969F61CB57C565
00000000XXXXXXXX 0994C95E6B5F495C 579875652B50416E 000000006B5F495C 000000002B50416E A8678A9AD4AFBE92 0000000000000000
XXXXXXXXXXXXXXXX 0994C95E6B5F495C 579875652B50416E 000000006B5F495C 000000002B50416E A8678A9AD4AFBE92 0000000000000000
0 ns 5 ns 10 ns 15 ns 20 ns 25 ns 30 ns 35 ns 40 ns 45 ns 50 ns 55 ns 60 ns 65 ns 70 ns 75 ns

Figure 7: Waveform for t = 0 to 76 ns – Measurement 1 to 6 (waveFAU1.png)

Next, in figure 8, we can see a clear demonstration of 32-bit operations. Measurement 63 corresponds to an addition resulting in a 0x7=’0111’ for the most significant bit (MSB). This results in a sign extension filling the upper 32 bits of the register with 0s. We can see a similar case for test 64 where the MSB value is 0xE=’1110’ which fills the upper 32 bits with 1s.

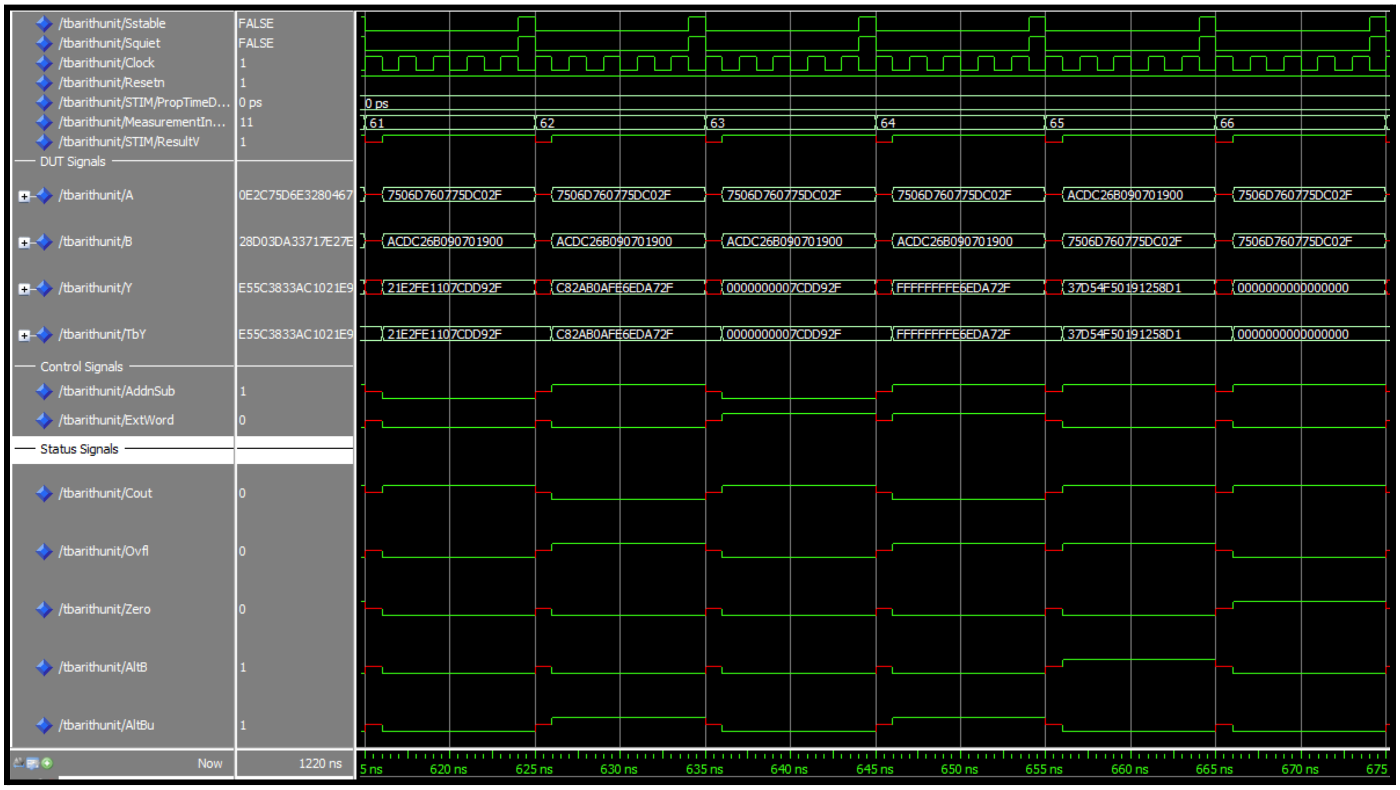


Figure 8: Waveform for t = 615ns to 675ns – Measurement 61 to 66 (waveFAU2.png)

Finally, we can visually inspect the simulation waveform and determine that the simulation ran to completion as we can see the final measurement 120 output and held for the remainder of the simulation time. Several aspects of the simulation can be confirmed both visually and via the console output, the aforementioned simulation completeness included.

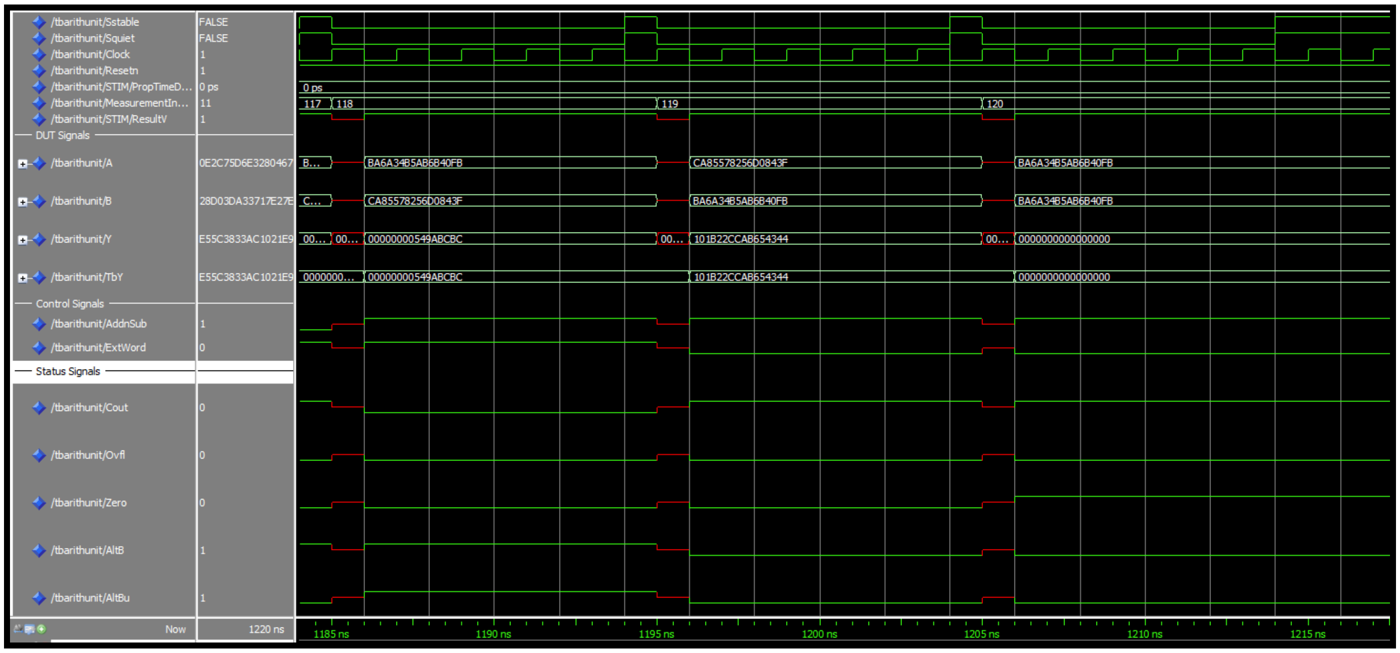
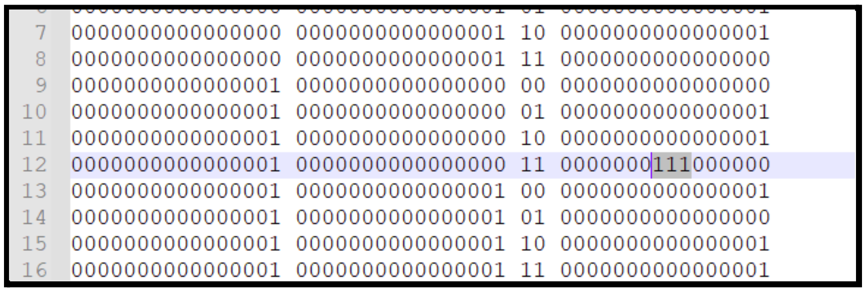


Figure 9: Waveform for t = 1185ns to 1215ns – Measurement 118 to 120 (waveFAU3.png)

2.2 Logic Unit:

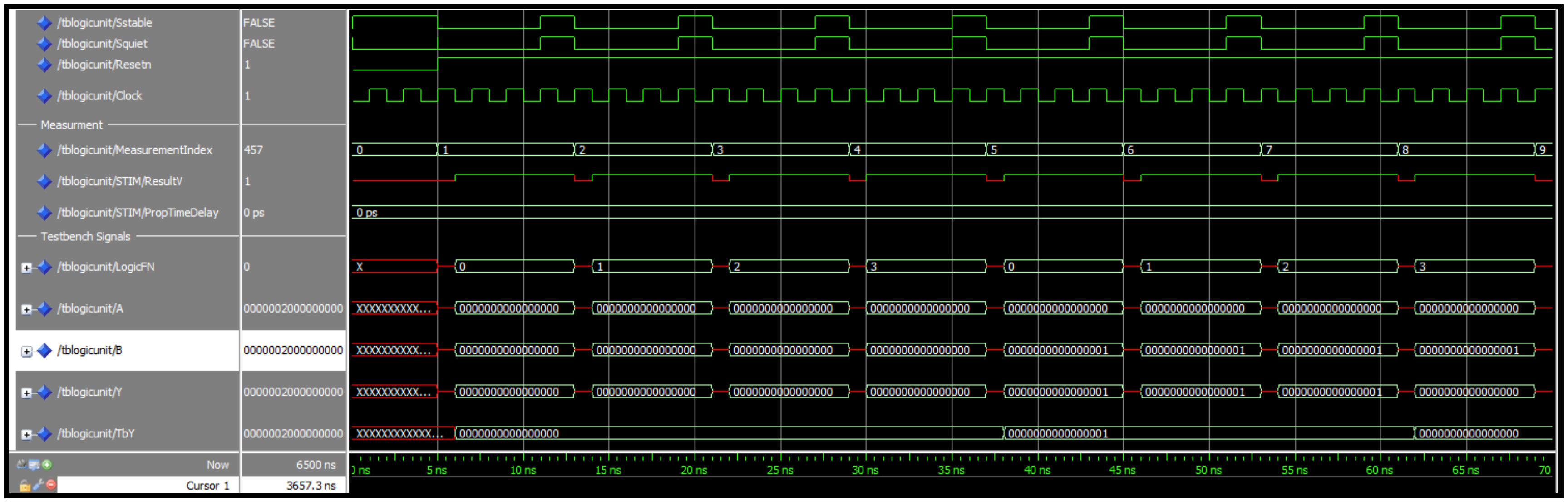
We now move to the logic unit functional simulation where we are first presented with the challenge of correcting the input test vector file. We can quickly determine that the error occurs within test 12 as the LogicFn='11' signal corresponds to an AND operation. A vector with a '1' in the least significant bit (LSB) and a vector of 0s should correspond with an output vector of only 0s given the AND operation. Therefore, we replace the '111' with '000' within the expected output of measurement 12. Next, we will briefly discuss the functional simulations as we received the correct output with our first implementation.



```
 7 0000000000000000 0000000000000001 10 0000000000000001
 8 0000000000000000 0000000000000001 11 0000000000000000
 9 0000000000000001 0000000000000000 00 0000000000000000
10 0000000000000001 0000000000000000 01 0000000000000001
11 0000000000000001 0000000000000000 10 0000000000000001
12 0000000000000001 0000000000000000 11 0000000111000000
13 0000000000000001 0000000000000001 00 0000000000000001
14 0000000000000001 0000000000000001 01 0000000000000000
15 0000000000000001 0000000000000001 10 0000000000000001
16 0000000000000001 0000000000000001 11 0000000000000001
```

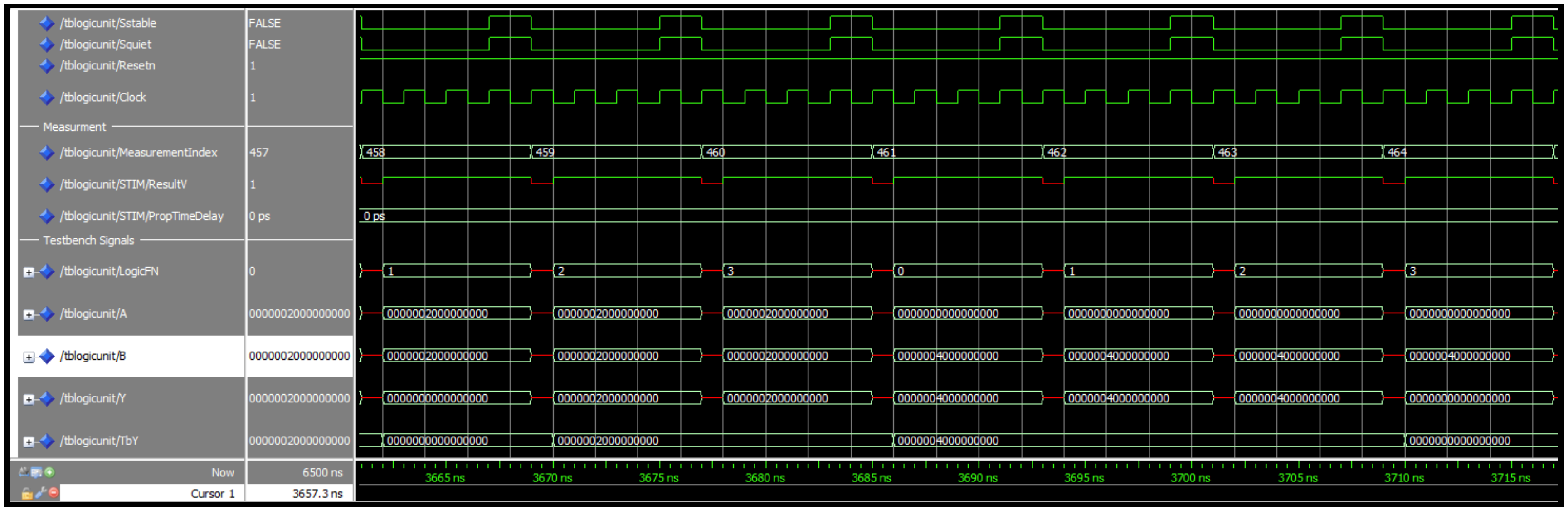
Figure 10: Measurement error within the LogicUnit00.tvs test vector file (line 12)

For the functional simulations, due to the relatively simple implementation of the logic unit, confirmation of the behavior of the circuit is quickly achieved. In figure 11, we expect that in measurement 1, two 0 vectors will only ever produce a 0 vector at the output. Similarly, when we have a ‘1’ in the LSB of one vector, all but the AND operation will pass the same vector with a ‘1’ in the LSB. Similar results are confirmed in figure 12 and figure 13.



/tblogicunit/Sstable FALSE
/tblogicunit/Squiet FALSE
/tblogicunit/Resetn 1
/tblogicunit/Clock 1
Measurment
/tblogicunit/MeasurementIndex 457
/tblogicunit/STIM/ResultV 1
/tblogicunit/STIM/PropTimeDelay 0 ps
Testbench Signals
/tblogicunit/LogicFN 0
/tblogicunit/A 0000002000000000
/tblogicunit/B 0000002000000000
/tblogicunit/Y 0000002000000000
/tblogicunit/TbY 0000002000000000
Now 6500 ns
Cursor 1 3657.3 ns

Figure 11: Waveform for t = 0 to 70 ns – Measurement 1 to 8 (waveFLU1.png)



/tblogicunit/Sstable FALSE
/tblogicunit/Squiet FALSE
/tblogicunit/Resetn 1
/tblogicunit/Clock 1
Measurment
/tblogicunit/MeasurementIndex 457
/tblogicunit/STIM/ResultV 1
/tblogicunit/STIM/PropTimeDelay 0 ps
Testbench Signals
/tblogicunit/LogicFN 0
/tblogicunit/A 0000002000000000
/tblogicunit/B 0000002000000000
/tblogicunit/Y 0000002000000000
/tblogicunit/TbY 0000002000000000
Now 6500 ns
Cursor 1 3657.3 ns

Figure 12: Waveform for t = 3661ns to 3715ns – Measurement 458 to 464 (waveFLU2.png)

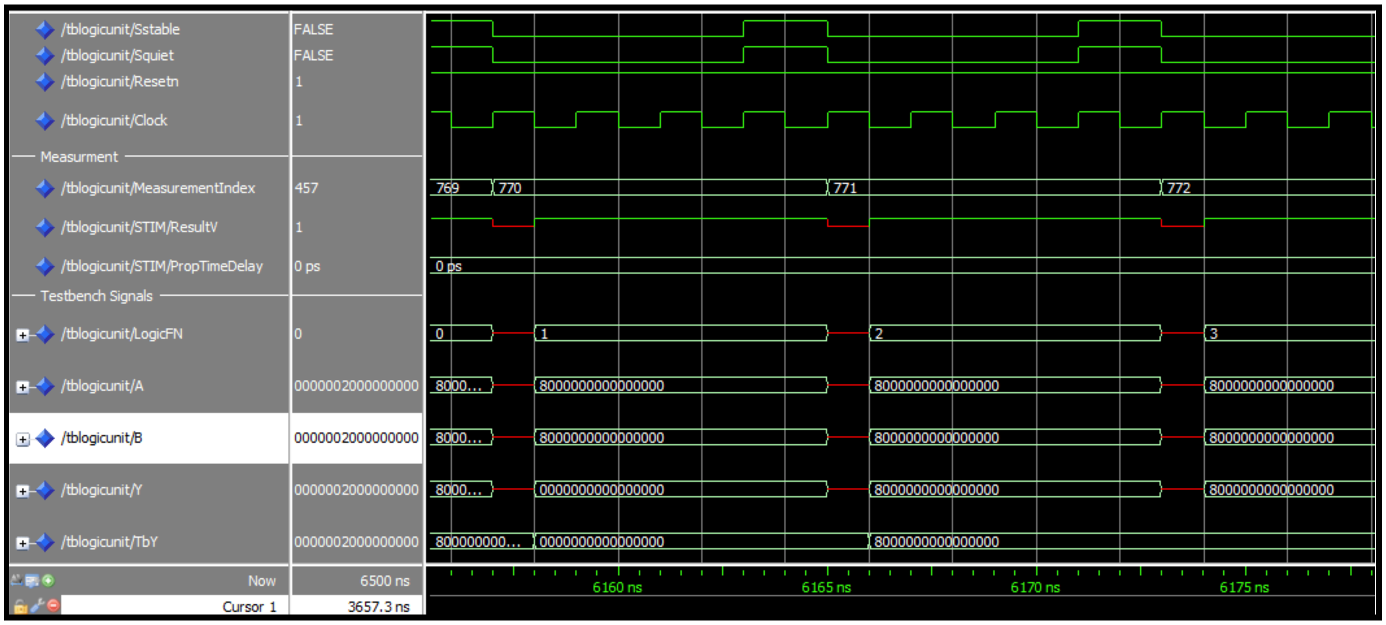


Figure 13: Waveform for t = 6156ns to 6179ns – Measurement 770 to 772 (waveFLU3.png)

2.3 Shift Unit:

The shift unit functional simulation proved extremely useful in tracking down small errors we had previously made in our circuit implementation. We were quickly able to identify that tests involving sign extension failed, and from this information resolve the issue. We realized that previously we relied too heavily on the modelsim compilation. The errors we had made were syntactically correct and thus passed compilation, however they were logically wrong, which we were only able to discover via waveform tests.

In figure 14, we display logical left shifts 64-bit of 2,3, and 4 on the input vector A. Of particular interest is measurement 132 corresponding to a shift of 3. With an odd number of shifts, the alignment of the ‘0x0AAAA’ portion of the vector corresponds to an output of ‘0x55557’. This is due to the fact that multiple repeated values of 0xA=‘1010’ when shifted to the left, corresponds to 0x5=‘0101’. This can be extended to the appearance of the 0x7=’0111’ value appearing at the output as this corresponds to the LSB of 0xA=‘101**0**’ and the 3 MSB of 0xF= ‘**111**1’. In figure 15, we test the same circuit but operate on word values (32-bit) using the ExtWord=’1’ signal.

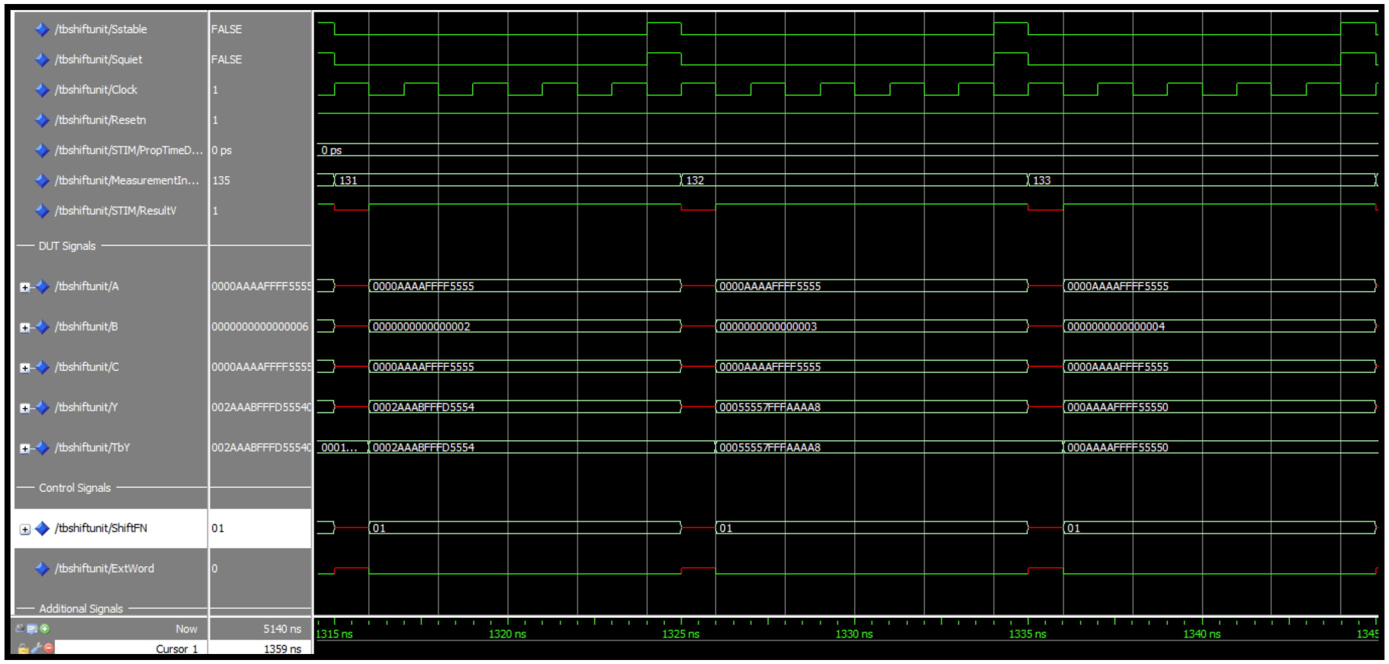


Figure 14: Waveform for t = 1315ns to 1345ns – Measurement 131 to 133 (waveFSUSLL64.png)

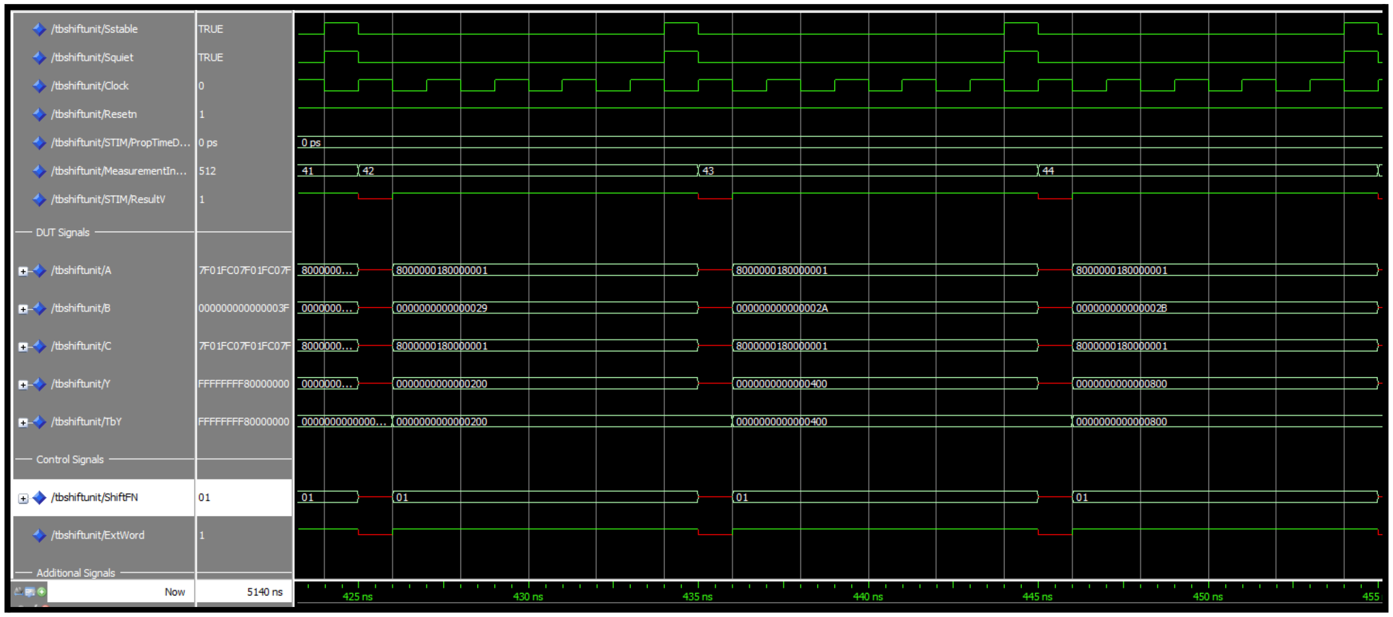
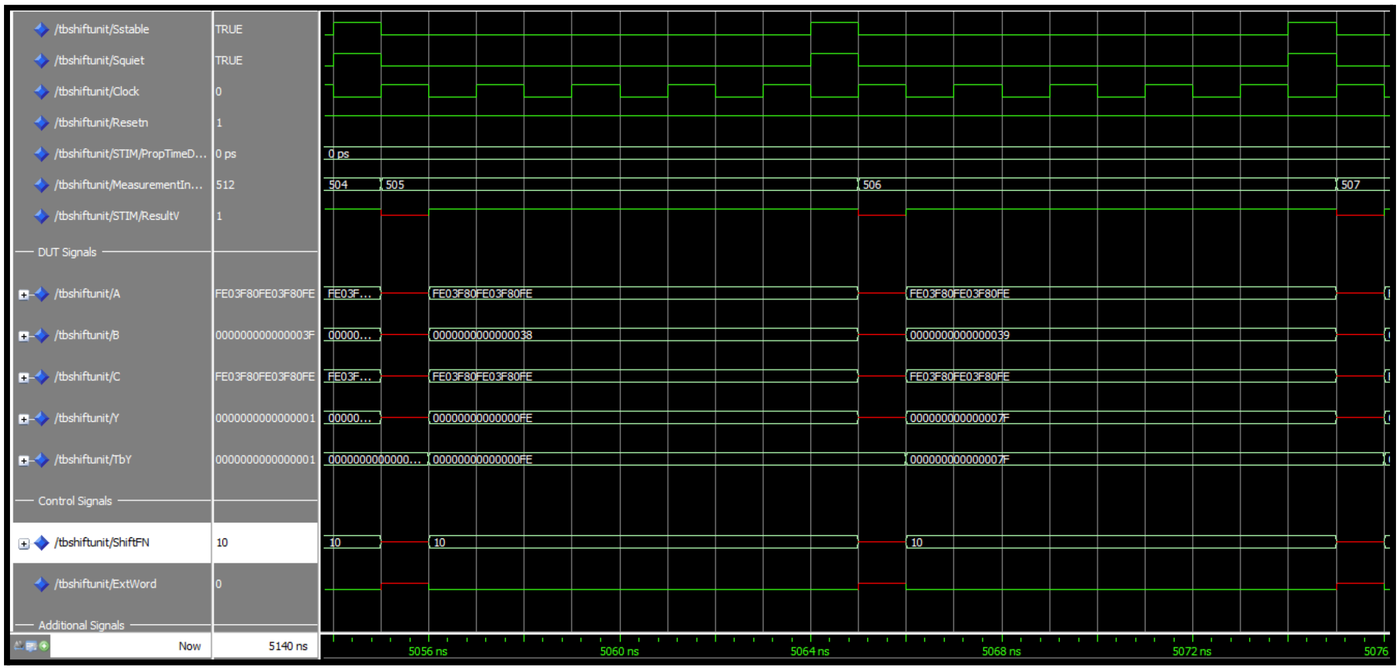


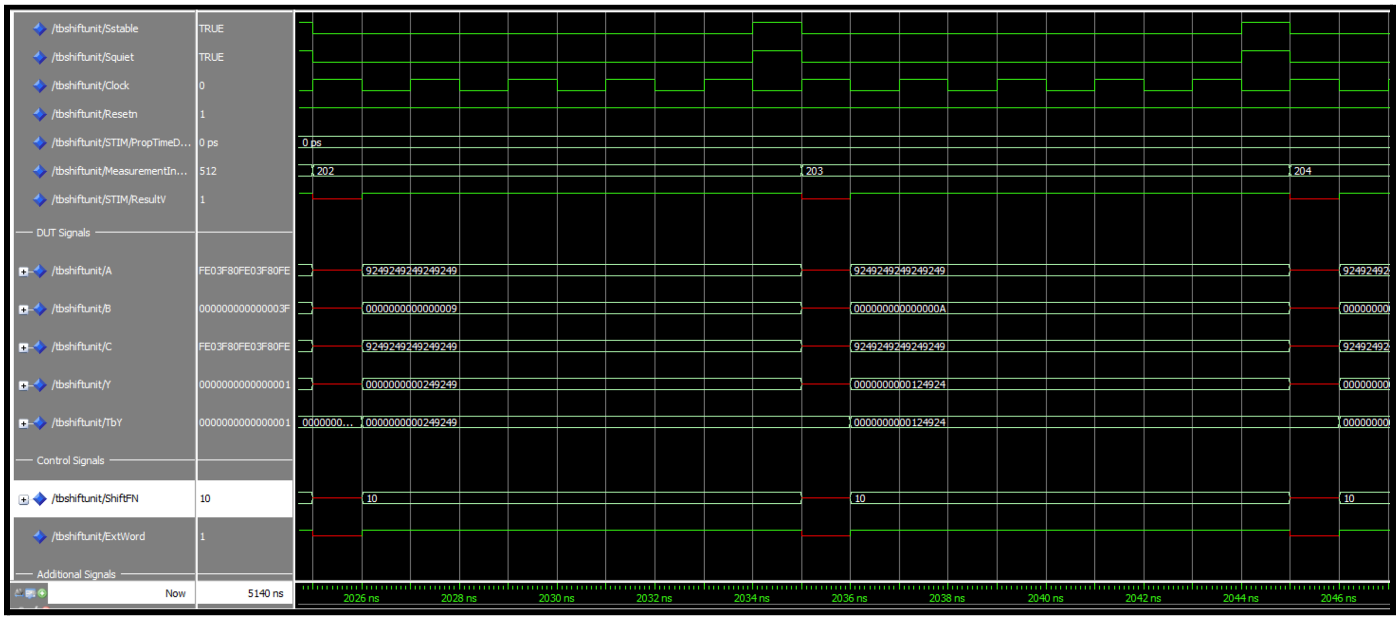
Figure 15: Waveform for t = 425ns to 455ns – Measurement 42 to 44 (waveFSUSLL32.png)

Next we discuss Shift Right Logical 64-bit in figure 16. The values within register B correspond to 0x38='00**111000**' and 0x39='00**111001**'. As we only use the lower 6 bits, we shift the value within register A a total of 48+8=56 and 48+8+1=57, respectively. Therefore we expect only 8 and 7 values of meaning in the resultant output vector due to such a large shift. We can confirm this is the case as the two test output vectors give 0xFE='11111110' and 0x7F='01111111' with the leading 0 of measurement 506 being a result of the zero extension scheme. We confirm that the zero extension operation corresponding to an arithmetic shift also works for 32-bit values as shown in figure 17.



/tbshiftunit/Sstable TRUE
/tbshiftunit/Squiet TRUE
/tbshiftunit/Clock 0
/tbshiftunit/Resetn 1
/tbshiftunit/STIM/PropTimeD... 0 ps
/tbshiftunit/MeasurementIn... 512
/tbshiftunit/STIM/ResultV 1
DUT Signals
/tbshiftunit/A FE03F80FE03F80FE
/tbshiftunit/B 000000000000003F
/tbshiftunit/C FE03F80FE03F80FE
/tbshiftunit/Y 0000000000000001
/tbshiftunit/TbY 0000000000000001
Control Signals
/tbshiftunit/ShiftFN 10
/tbshiftunit/ExtWord 0
Additional Signals
Now 5140 ns
0 ps
504 505 506 507
FE03F... FE03F80FE03F80FE FE03F80FE03F80FE
00000... 0000000000000038 0000000000000039
FE03F... FE03F80FE03F80FE FE03F80FE03F80FE
00000... 00000000000000FE 000000000000007F
0000000000000... 00000000000000FE 000000000000007F
10 10 10
5056 ns 5060 ns 5064 ns 5068 ns 5072 ns 5076

Figure 16: Waveform for t = 5055ns to 5076ns – Measurement 505 to 506 (waveFSUSRL64.png)



/tbshiftunit/Sstable TRUE
/tbshiftunit/Squiet TRUE
/tbshiftunit/Clock 0
/tbshiftunit/Resetn 1
/tbshiftunit/STIM/PropTimeD... 0 ps
/tbshiftunit/MeasurementIn... 512
/tbshiftunit/STIM/ResultV 1
DUT Signals
/tbshiftunit/A FE03F80FE03F80FE
/tbshiftunit/B 000000000000003F
/tbshiftunit/C FE03F80FE03F80FE
/tbshiftunit/Y 0000000000000001
/tbshiftunit/TbY 0000000000000001
Control Signals
/tbshiftunit/ShiftFN 10
/tbshiftunit/ExtWord 1
Additional Signals
Now 5140 ns
0 ps
202 203 204
9249249249249249 9249249249249249
0000000000000009 000000000000000A
9249249249249249 9249249249249249
0000000000249249 0000000000124924
0000000... 0000000000249249 0000000000124924
10 10 10
2026 ns 2028 ns 2030 ns 2032 ns 2034 ns 2036 ns 2038 ns 2040 ns 2042 ns 2044 ns 2046 ns

Figure 17: Waveform for t = 2026ns to 2046ns – Measurement 202 to 203 (waveFSUSRL32.png)

For the simulation corresponding to figure 18, we explore the results of the Shift Right Arithmetic 64-bit. In simulation 420, we shift the input vector a total of 0x63='01**100011**' 48+2+1=51 places. Therefore, the majority of the output vector will be a result of the sign extension corresponding to the arithmetic shift scheme. Similar to the SLL64 test, due to the alignment and the use of a non-aligned shift amount of 51, the lower values of the output vector in hexadecimal representation have completely changed from the values given in the input vector. In figure 19, we again verify that the same operation successfully completes given a 32-bit operation.

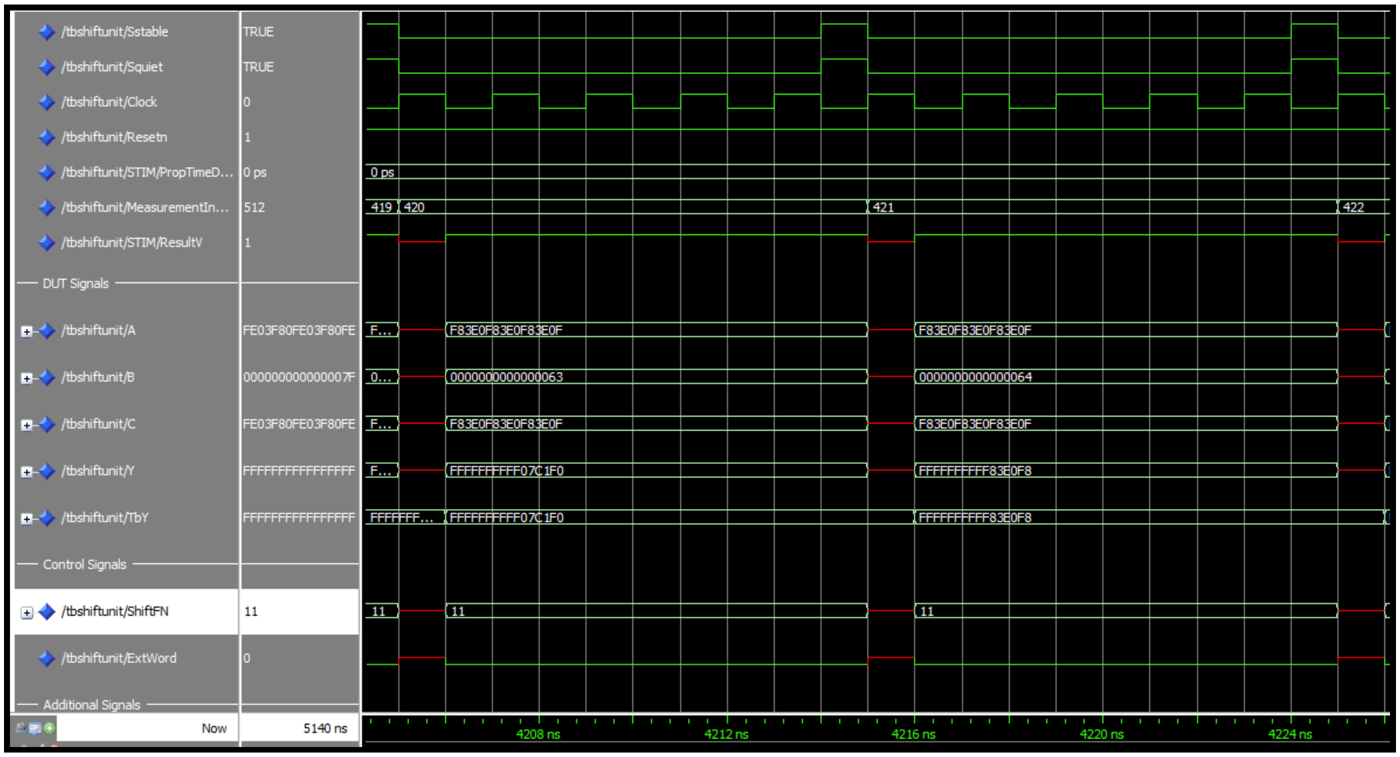


Figure 18: Waveform for t = 4209ns to 4225ns – Measurement 420 to 421 (waveFSUSRA64.png)

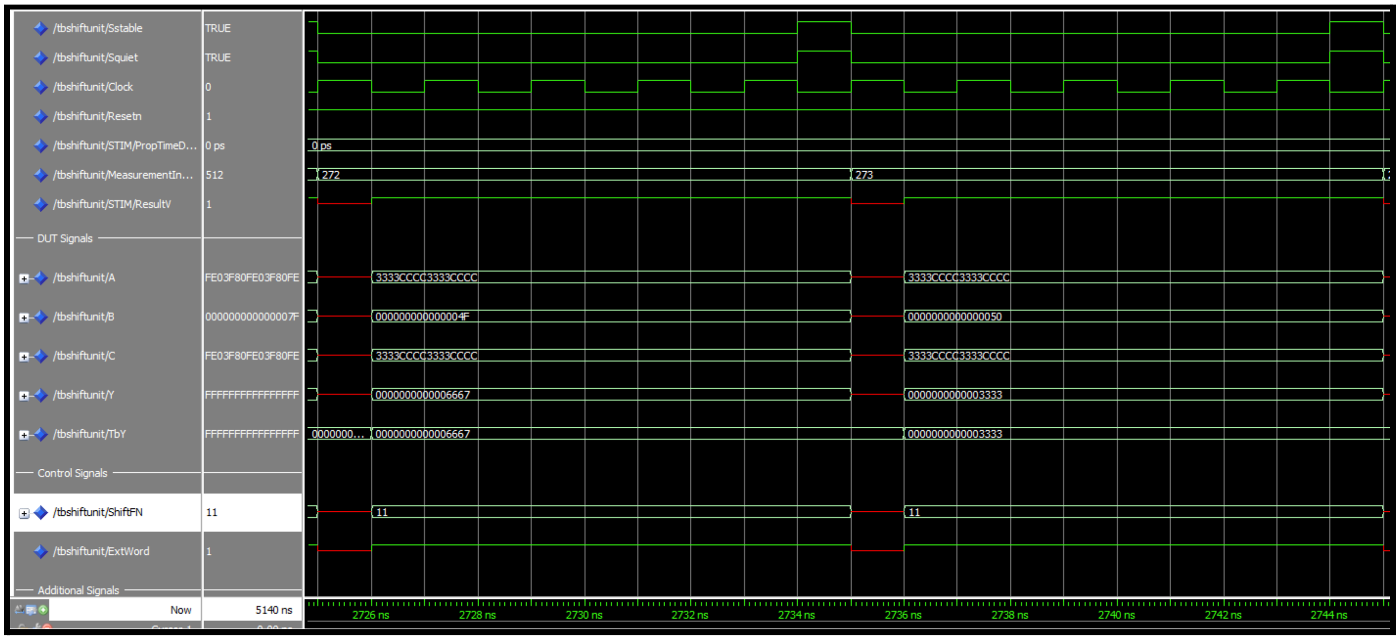


Figure 19: Waveform for t = 2725ns to 2745ns – Measurement 272 to 273 (waveFSUSRA32.png)

2.4 Execution Unit:

Finally, we end with a discussion of the larger execution unit simulation. For this simulation, we were able to confirm our implementation met the expected results for each test vector on the first attempt. We attribute this to the fact that implementing this unit is relatively straightforward as it essentially connects each individual unit together along with a final output mux.

In figure 20, we can use the current control signal values to determine the type of signal being produced. We determine that the test for test vector 20 corresponds to a logic function, operation type AND. Therefore, given the two input vectors with only a common ‘1’ value in the LSB, we would expect an output vector of all 0s except for a ‘1’ in the LSB. We confirm our output and the expected value within the test vector matches this prediction. For this type of operation, the signals AddnSub, ExtWord, and ShiftFN could have any arbitrary signal applied to them and we would still receive the same output.

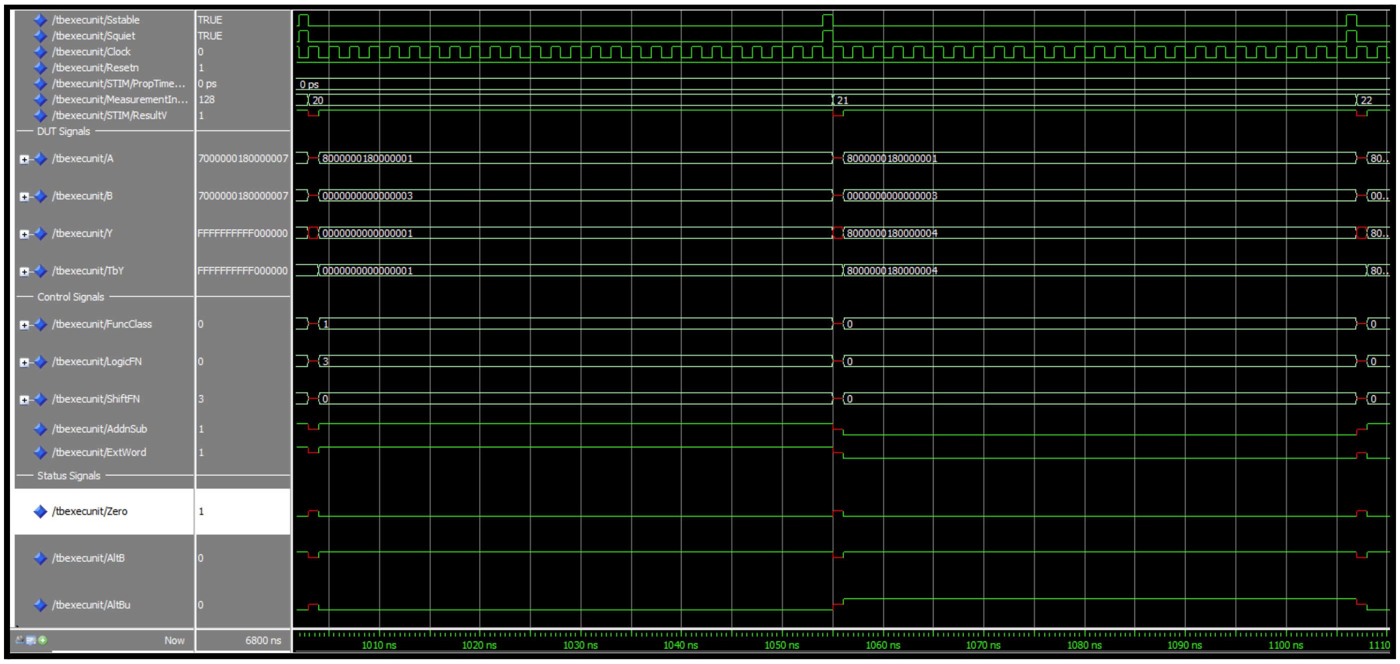
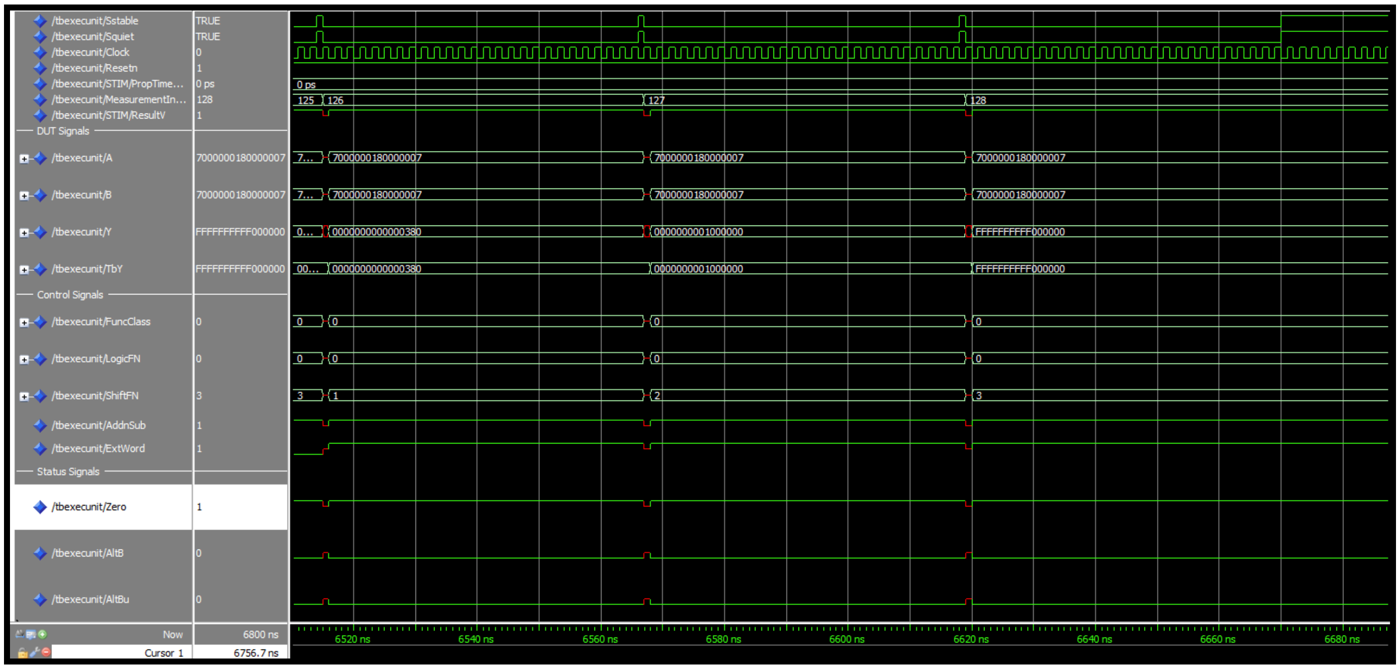


Figure 20: Waveform for t = 1009ns to 1110ns – Measurement 20 to 21 (waveFEU1.png)

In figure 21, we confirm that a separate operation type also works. In this case, the test at measurement index 127 corresponds to a SRL word operation. As we are operating on a word, we are shifting the vector 0x800000007 a total of 0x07=’000**00111**’, 4+3=7 places. As a result we receive an output vector with a single ‘1’ bit and all other bits corresponding to ‘0’.



```
/tbexecunit/Sstable            TRUE
/tbexecunit/Squiet             TRUE
/tbexecunit/Clock              0
/tbexecunit/Resetn             1
/tbexecunit/STIM/PropTime...   0 ps
/tbexecunit/MeasurementIn...   128
/tbexecunit/STIM/ResultV       1
DUT Signals
/tbexecunit/A                  7000000180000007
/tbexecunit/B                  7000000180000007
/tbexecunit/Y                  FFFFFFFFFF000000
/tbexecunit/TbY                FFFFFFFFFF000000
Control Signals
/tbexecunit/FuncClass          0
/tbexecunit/LogicFN            0
/tbexecunit/ShiftFN            3
/tbexecunit/AddnSub            1
/tbexecunit/ExtWord            1
Status Signals
/tbexecunit/Zero               1
/tbexecunit/AltB               0
/tbexecunit/AltBu              0
Now                            6800 ns
Cursor 1                       6756.7 ns
```

Figure 21: Waveform for t = 6519ns to 6680ns – Measurement 126 to 128 (waveFEU2.png)

Part 3: Synthesis

3.1 Arithmetic Unit:

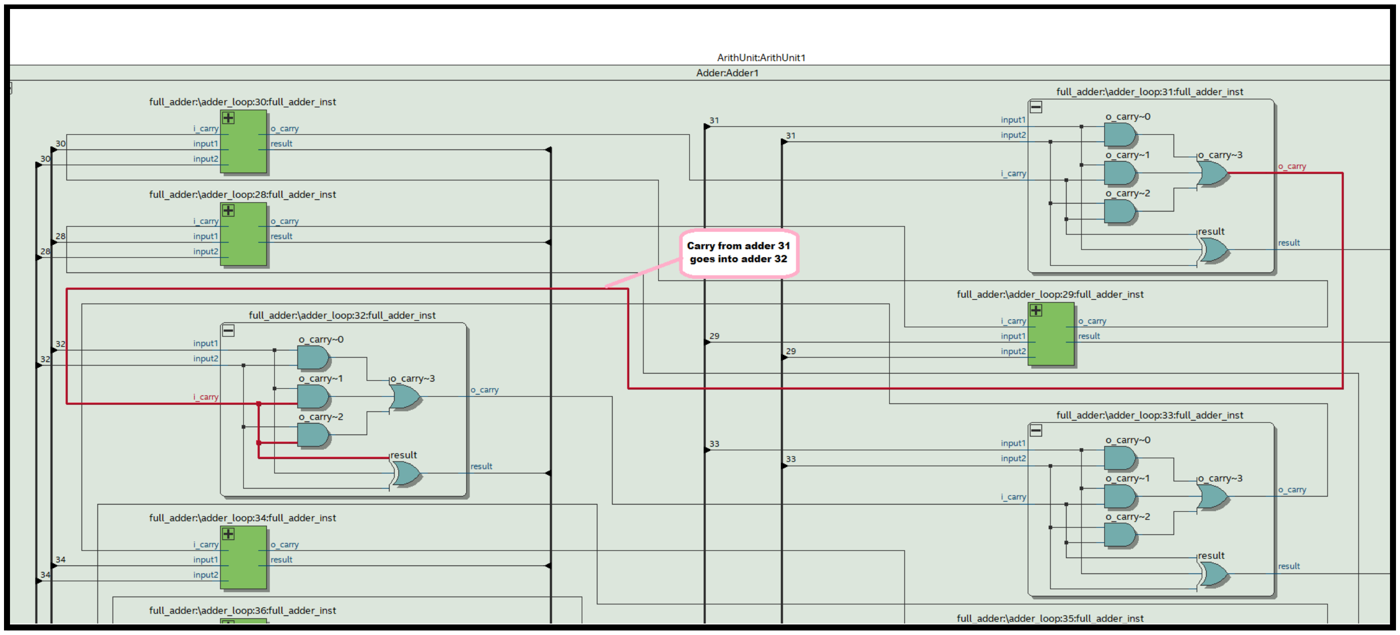

ArithUnit:ArithUnit1
Adder:Adder1
full_adder:\adder_loop:30:full_adder_inst
full_adder:\adder_loop:28:full_adder_inst
full_adder:\adder_loop:32:full_adder_inst
full_adder:\adder_loop:34:full_adder_inst
full_adder:\adder_loop:36:full_adder_inst
full_adder:\adder_loop:31:full_adder_inst
full_adder:\adder_loop:29:full_adder_inst
full_adder:\adder_loop:33:full_adder_inst
full_adder:\adder_loop:35:full_adder_inst
Carry from adder 31 goes into adder 32

Figure 22: Arithmetic unit ripple carry adder for bits 31 and 32

The image above demonstrates the functionality of a ripple adder. The carry out from the previous addition flows into the carry in of the next addition. Just like the circuit diagram in part 1, the synthesized full adder is implemented using logic as follows to determine the carry: (input1 AND input2) OR (carry_in AND input1) OR (carry_in AND input2) as well as a three input XOR gate to determine the result as shown below. The output synthesis matches our expectations.

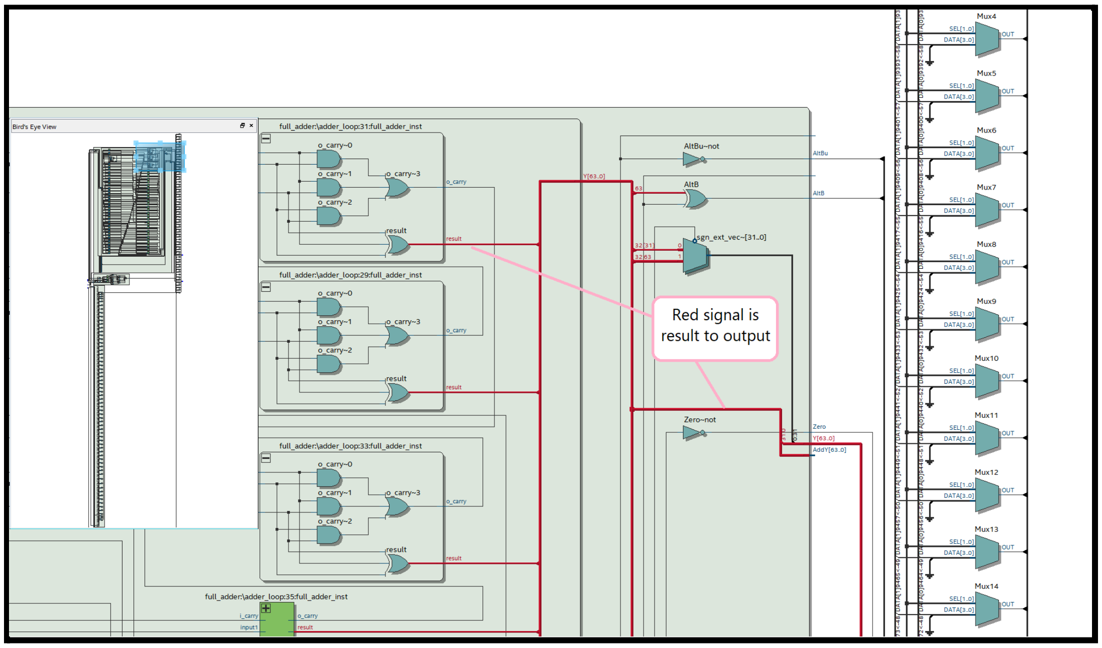

Bird's Eye View
full_adder:\adder_loop:31:full_adder_inst
full_adder:\adder_loop:29:full_adder_inst
full_adder:\adder_loop:33:full_adder_inst
full_adder:\adder_loop:35:full_adder_inst
AltBu~not
AltB
sgn_ext_vec~[31..0]
Zero~not
Red signal is result to output
Mux4
Mux5
Mux6
Mux7
Mux8
Mux9
Mux10
Mux11
Mux12
Mux13
Mux14

Figure 23: The output of the arithmetic unit via the individual xor gates

3.2 Logic Unit:

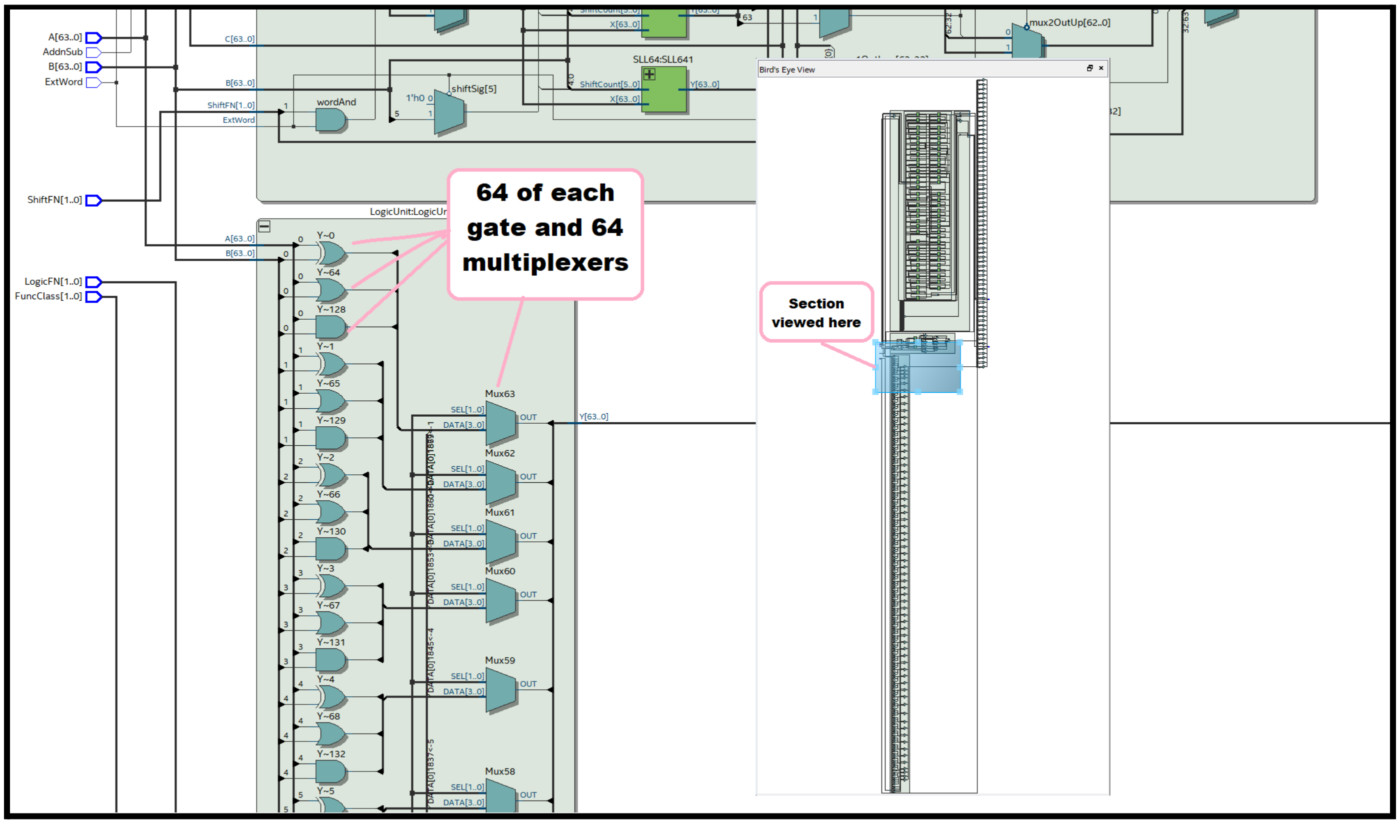


Figure 24: The logic unit synthesis including each basic gate and a mux for every bit

The logic unit essentially has 64 copies of AND, OR and XOR gates in order to perform 64 bit operations on the input vectors. Compared to our previous circuit diagram, the circuit is equivalent with the sole difference being the singular 64 bit gates are expanded into their more accurate form of 64 individual 1 bit gates for the bitwise operation. The output is then selected from the 64 multiplexers resulting in the output vector being passed.

3.3 Shift Unit:

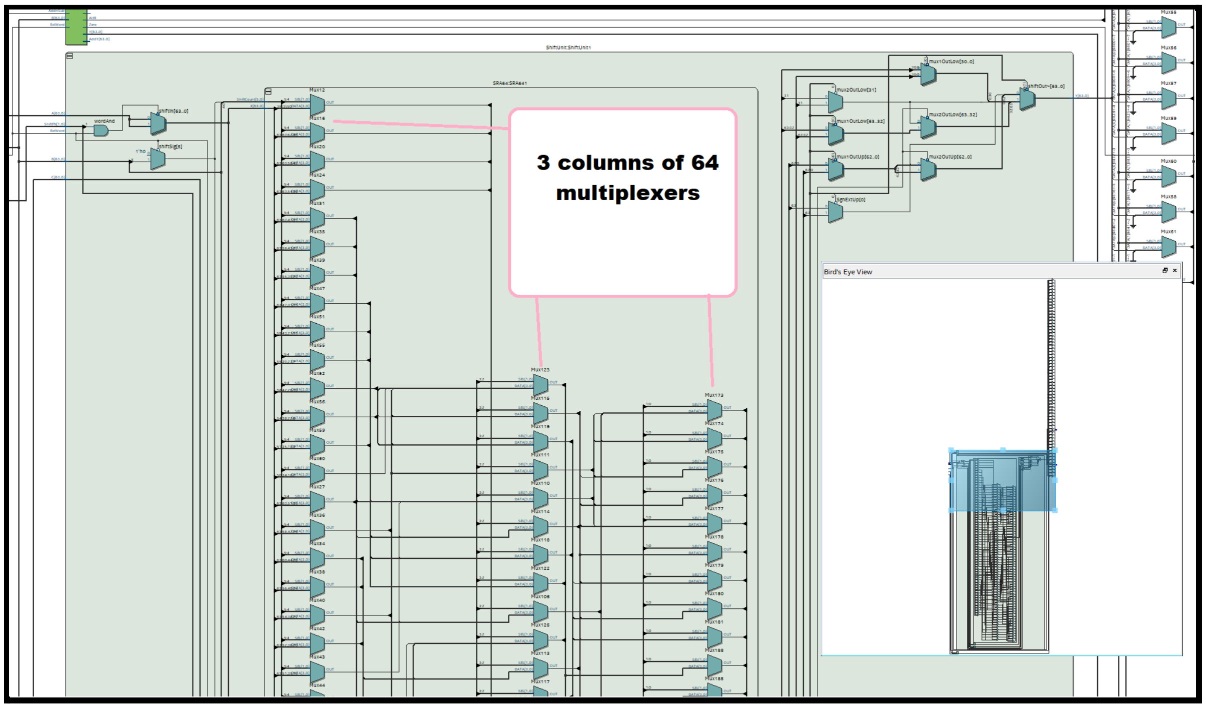


Figure 25: The shift unit made up of several levels of muxes

The individual barrel shifter for shift-right-arithmetic as shown above demonstrates the three layers of multiplexers used in order to perform the shifting operation. Similarly to our circuit diagram above, with the difference being the 64 bit 4-1 multiplexers are expanded. With this circuit there will be 3 levels of delay due to the sets of multiplexers in series.

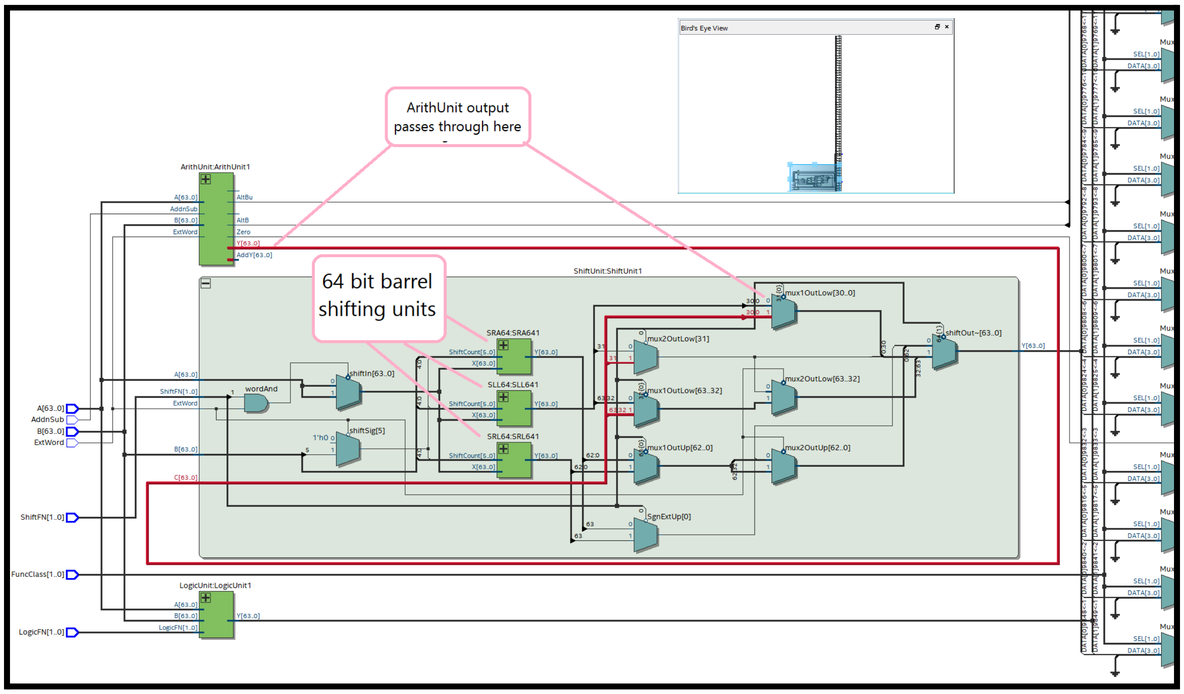


Figure 26: The output of the arithmetic unit feeding into the shift unit

The overall shift unit is as expected, with signals running to and from the barrel shifting units and then through to the output multiplexers. Something of note is the output from the arithmetic unit being passed through the circuit along with the three barrel shifter outputs. This is for the case of ShiftFN = ‘00’, meaning an arithmetic operation was performed instead of a shifting operation and therefore the arithmetic operation must be forwarded through the shifting unit to the end of the execution unit. Synthesized result is very similar to our circuit diagram.

3.4 Execution Unit:

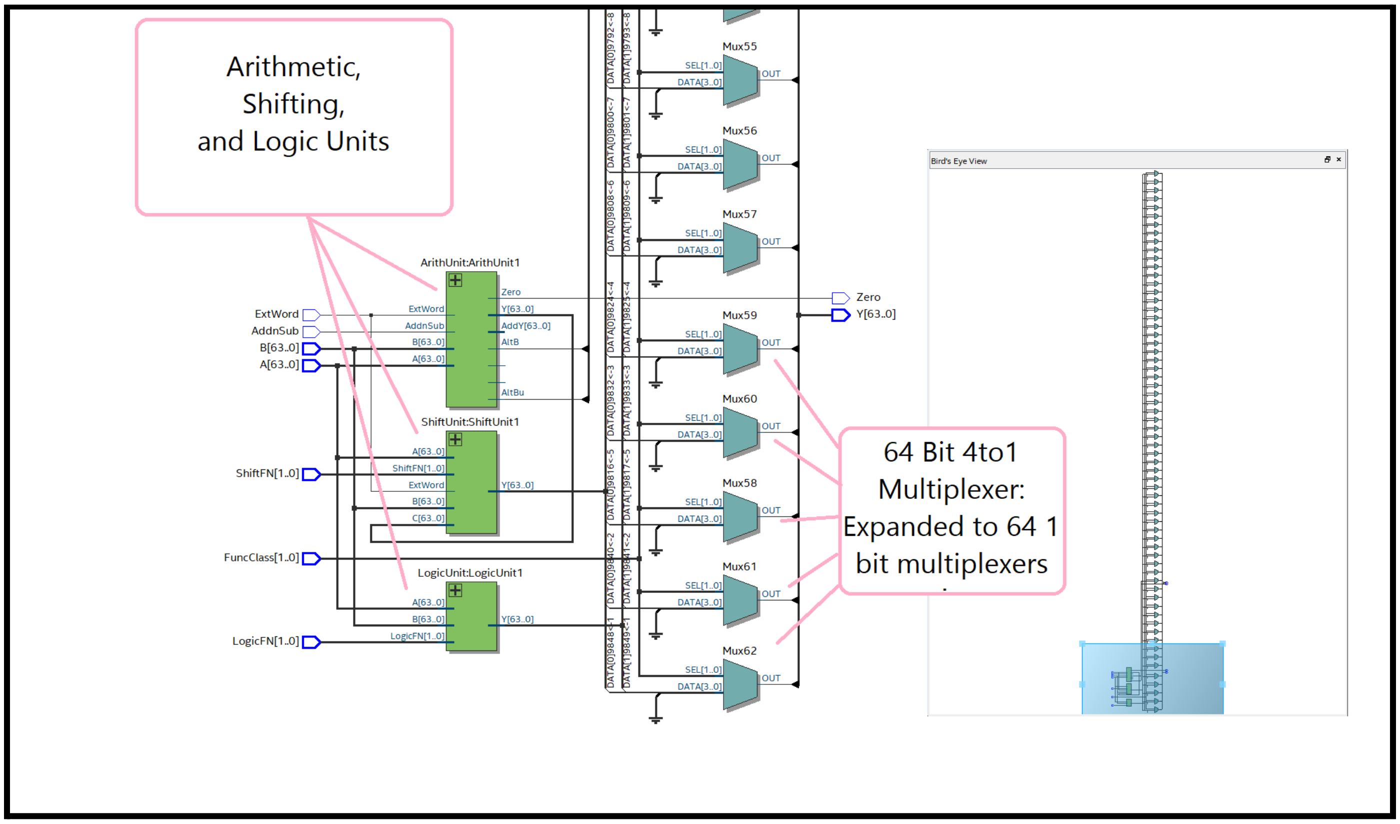


Figure 27: A general overview of the execution circuit with the chain of output muxes

The synthesized execution unit is the same as our diagram with the only exception being the 64 bit multiplexor is expanded into 64 1 bit multiplexers for each bit in the output vectors. Its job is essentially to ferry signals to each of the operational units and decide which of their outputs will be selected and passed to the execution unit output.

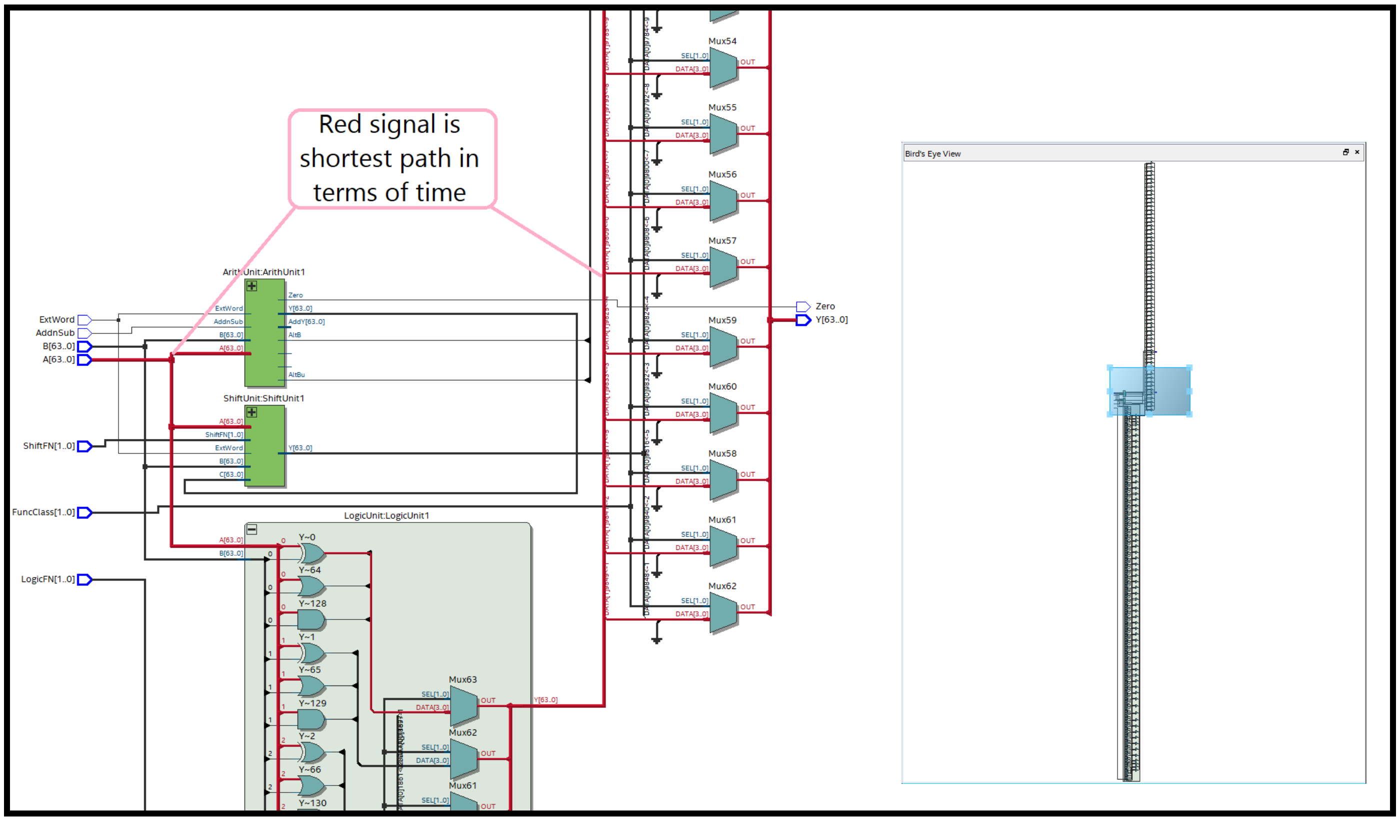


Figure 28: Shortest path (delay) through the execution unit circuit

The arithmetic unit runs addition and subtraction using a slow ripple adder, whereas the barrel shifters use multiple layers of multiplexers to perform the operation. The logic unit simply has 64 logic gates in parallel as well as 64 multiplexers in parallel, along with another set of 64 multiplexers right before exiting the execution unit and therefore has the shortest (delay) path possible.

Part 4: Timing Simulation

4.1 Arithmetic Unit:

As previously mentioned, we expect the arithmetic unit to have a higher delay associated with its operations compared with the other units due to the ripple carry adder. We measure the delay associated with the test and determine that for measurement index 1, we get a propagation delay of 22.9ns. We can see on the graph in hexadecimal values that due to the carry propagation, certain bits of the final value take longer than others. Figure 30 shows this gradual change.

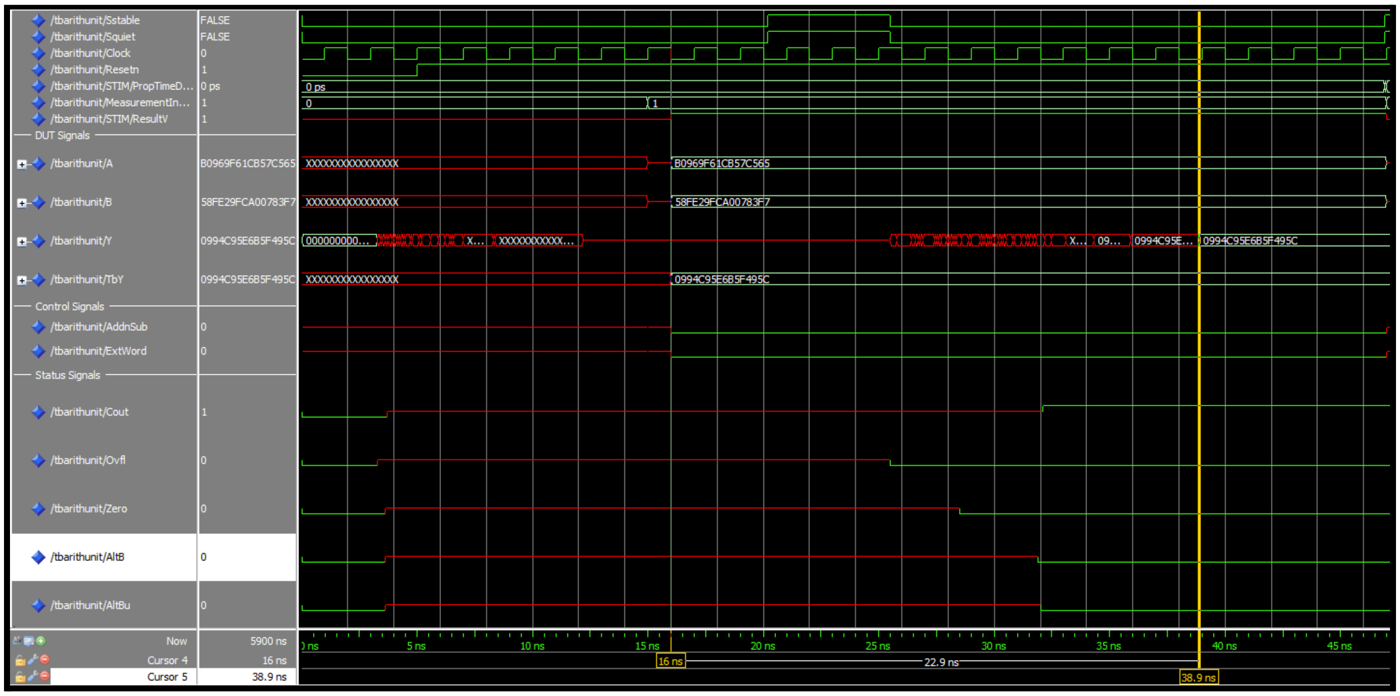

/tbarithunit/Sstable FALSE
/tbarithunit/Squiet FALSE
/tbarithunit/Clock 0
/tbarithunit/Resetn 1
/tbarithunit/STIM/PropTimeD... 0 ps
/tbarithunit/MeasurementIn... 1
/tbarithunit/STIM/ResultV 1
DUT Signals
/tbarithunit/A B0969F61CB57C565
/tbarithunit/B 58FE29FCA00783F7
/tbarithunit/Y 0994C95E6B5F495C
/tbarithunit/TbY 0994C95E6B5F495C
Control Signals
/tbarithunit/AddnSub 0
/tbarithunit/ExtWord 0
Status Signals
/tbarithunit/Cout 1
/tbarithunit/Ovfl 0
/tbarithunit/Zero 0
/tbarithunit/AltB 0
/tbarithunit/AltBu 0
Now 5900 ns
Cursor 4 16 ns
Cursor 5 38.9 ns
22.9 ns

Figure 29: Waveform for t = 0 to 45ns – Measurement 1, propagation delay of 22.9ns (waveTAU1.png)

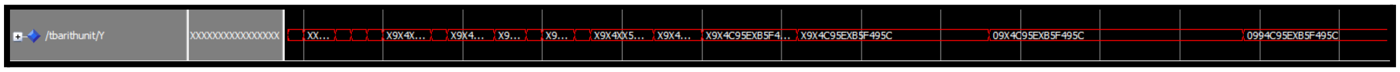


Figure 30: The propagation of the carry bit as the signal settles

Next, we highlight the difference in timing based upon input values. As shown in figure 31, we can see that measurement 55 and measurement 56 have significantly different propagation delays of 17.7ns and 65.5ns respectively. This large difference can be attributed to the input carry bit of the twos complement representation. When the two registers contain the same value and we perform a subtraction of the two, the carry-in bit results in a carry-in/out bit for every single full adder as it ripples through the circuit. This therefore is the largest possible delay in the arithmetic circuit. Should this design be implemented in real life and a data sheet created alongside it, this measurement would likely be used to characterize the propagation delay of the circuit as we use the worst case delay.

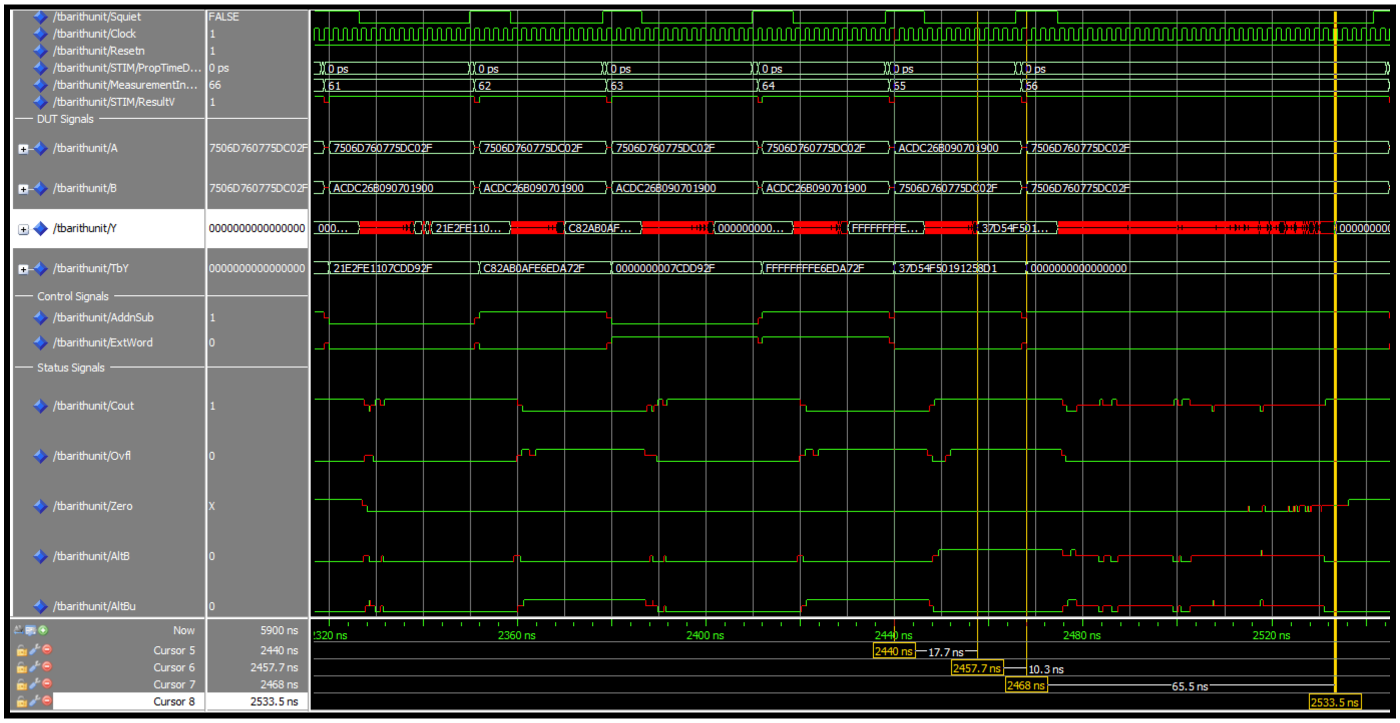


Figure 31: Waveform for t = 2320ns to 2535ns – Measurement 61 to 66, Propagation delays 17.7ns and 65.6 ns for measurements 65 and 66 respectively (waveTAU2.png)

Finally, on measurement 120, we can confirm our earlier finding in the previous waveform as we are again performing a subtraction of two identical numbers. As seen in figure 32, we measure the propagation delay to be 65.5ns.

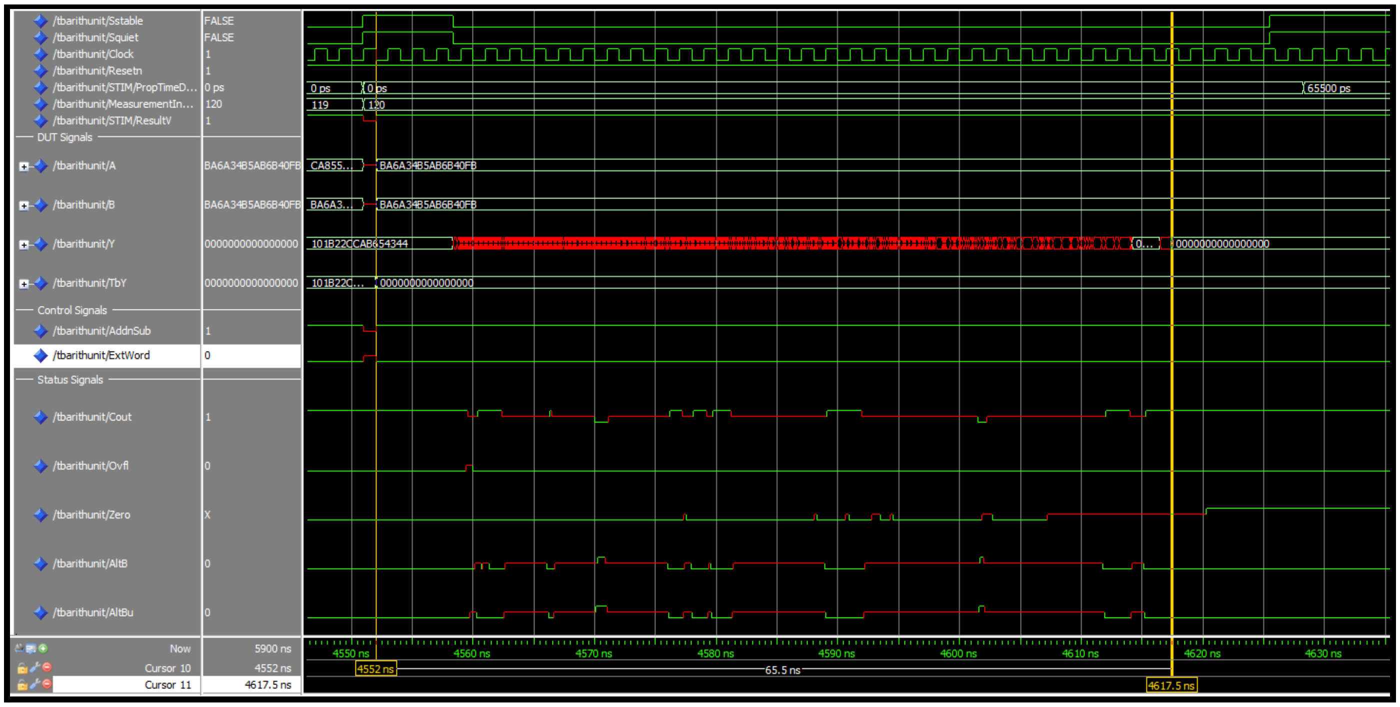
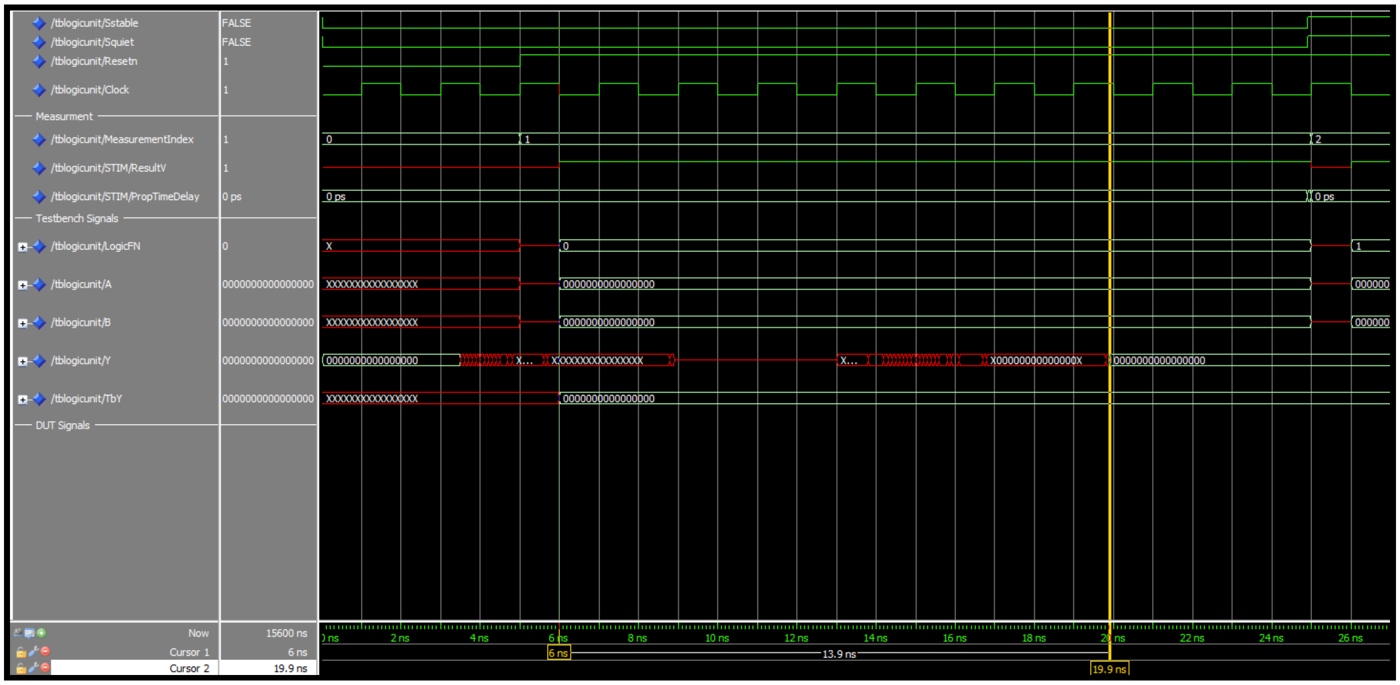


Figure 32: Waveform for t = 4550ns to 4630ns – Measurement 120, Propagation delay of 65.5ns (waveTAU3.png)

4.2 Logic Unit:

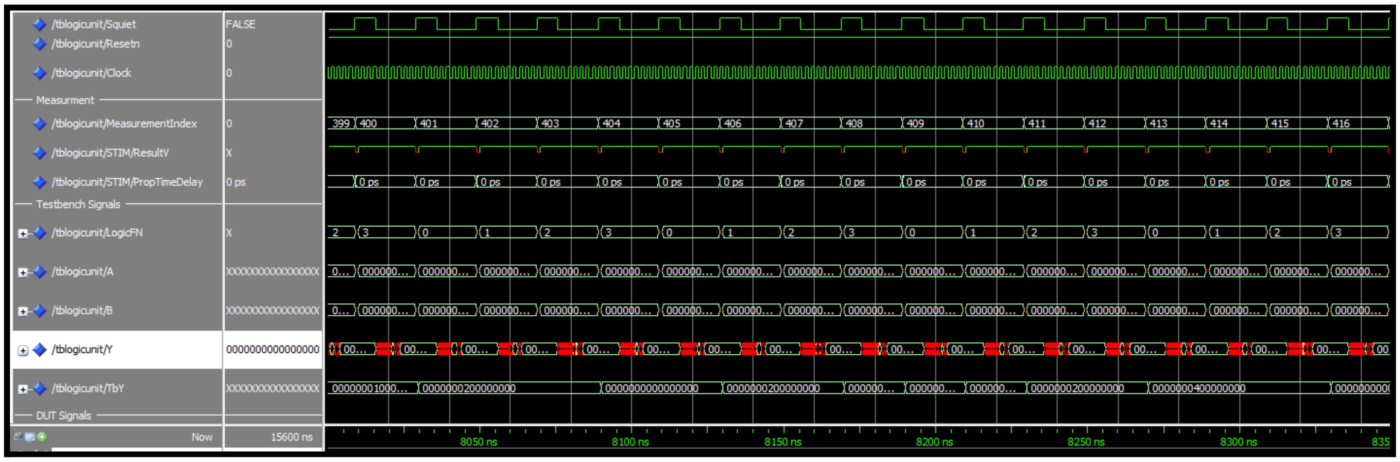
We can see from the below waveform captures that the logicUnit has properties separate from the other units that make it unique. As we can clearly see in figure 33, this circuit has the lowest propagation delay of all execution unit components due to the relatively low number of logic units required for synthesis. We measure a propagation delay of 13.9ns.



/tblogicunit/Sstable FALSE
/tblogicunit/Squiet FALSE
/tblogicunit/Resetn 1
/tblogicunit/Clock 1
Measurment
/tblogicunit/MeasurementIndex 1
/tblogicunit/STIM/ResultV 1
/tblogicunit/STIM/PropTimeDelay 0 ps
Testbench Signals
/tblogicunit/LogicFN 0
/tblogicunit/A 0000000000000000
/tblogicunit/B 0000000000000000
/tblogicunit/Y 0000000000000000
/tblogicunit/TbY 0000000000000000
DUT Signals
Now 15600 ns
Cursor 1 6 ns
Cursor 2 19.9 ns
13.9 ns

Figure 33: Waveform for t = 0 to 26ns – Measurement 1, Propagation delay of 13.9ns (waveTLU1.png)

Looking at figure 34, we can now see a larger pattern emerging. Regardless of the input, the output for every test vector appears to have roughly the same propagation delay. This can again be explained by the simple nature of the logic unit. Regardless of control signals, the logic unit will only ever have the input vector pass through a single logic gate, therefore in theory giving the same propagation delay for every input to output.



/tblogicunit/Squiet FALSE
/tblogicunit/Resetn 0
/tblogicunit/Clock 0
Measurement
/tblogicunit/MeasurementIndex 0
/tblogicunit/STIM/ResultV X
/tblogicunit/STIM/PropTimeDelay 0 ps
Testbench Signals
/tblogicunit/LogicFN X
/tblogicunit/A XXXXXXXXXXXXXXXX
/tblogicunit/B XXXXXXXXXXXXXXXX
/tblogicunit/Y 0000000000000000
/tblogicunit/TbY XXXXXXXXXXXXXXXX
DUT Signals
Now 15600 ns
8050 ns
8100 ns
8150 ns
8200 ns
8250 ns
8300 ns

Figure 34: Waveform for t = 8010ns to 8350ns – Measurement 400 to 416 (waveTLU2.png)

Finally, we can visually confirm the final test vector passes and the propagation delay is consistent with 13.9ns in figure 35 below.

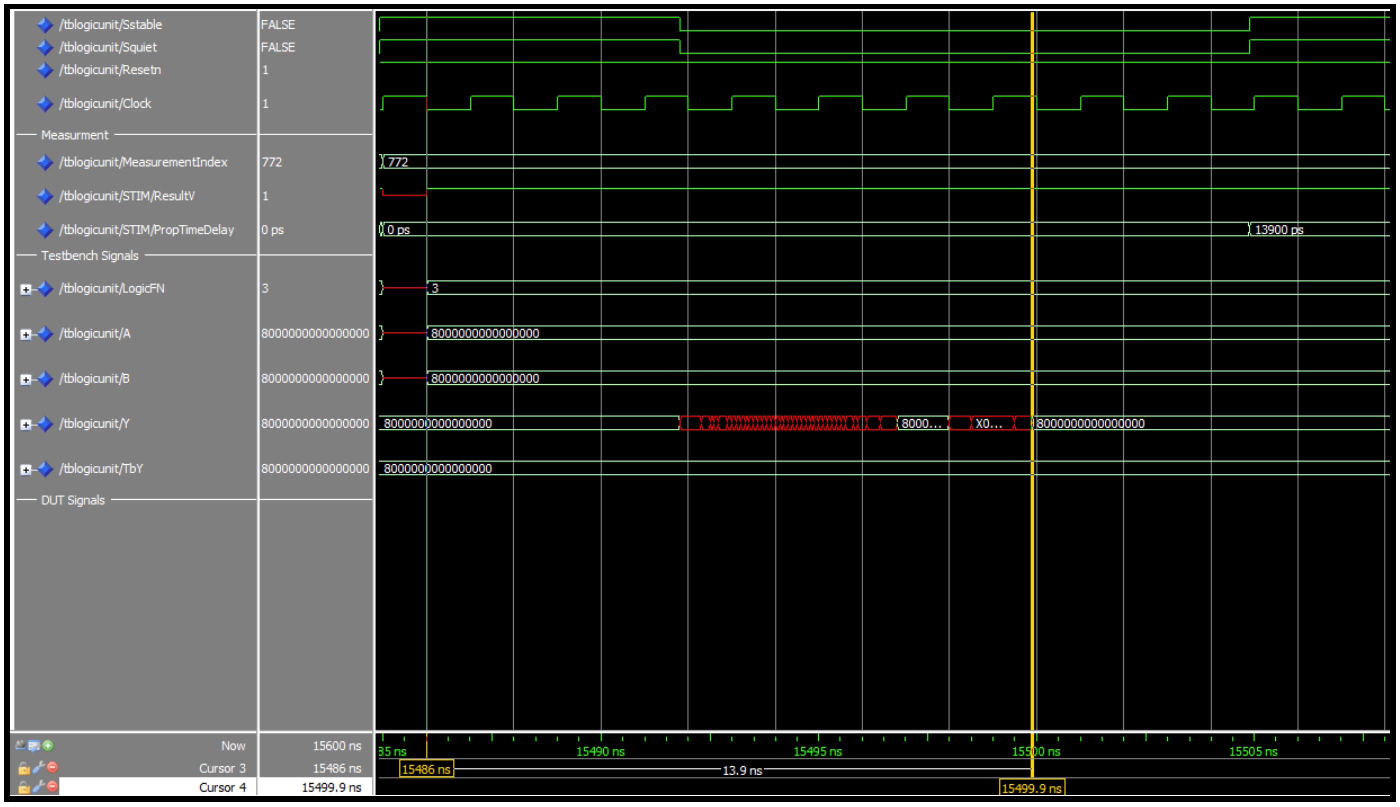


Figure 35: Waveform for t = 15486ns to 15505ns – Measurement 772, Propagation delay of 13.9ns (waveTLU3.png)

4.3 Shift Unit:

The shift unit in practice is a collection of muxes chained together. Therefore, we would expect a fairly similar propagation delay associated with any two arbitrary shift functions. As we see from the timing simulation, we can confirm this result and specify how there may be slight variance in the delay measurements.

First we will compare the two 64-bit and 32-bit shift units. We can observe a measured 21.7ns delay for the 64-bit circuit in figure 36, and a 22.6ns delay for the 32-bit circuit in figure 37. Based upon this measurement alone, it would appear that the 32-bit circuit has a larger delay. However, we must take into account the fact that the 64-bit circuit is only shifting the value within register A once, whereas the 32-bit circuit is shifting the register a total of 16+8=24 places. We will look to the next set of vectors to determine if the shift amount affects the circuit propagation delay.

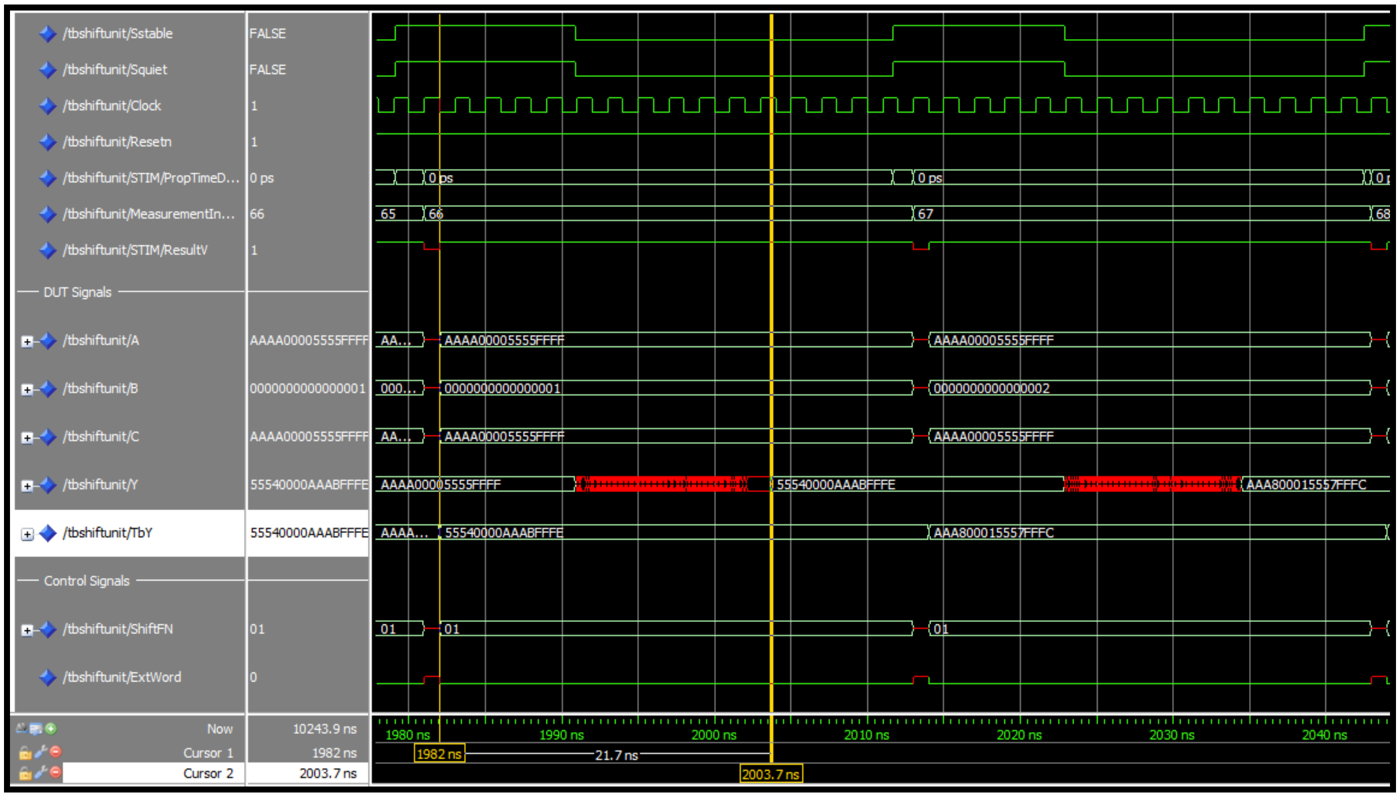

/tbshiftunit/Sstable FALSE
/tbshiftunit/Squiet FALSE
/tbshiftunit/Clock 1
/tbshiftunit/Resetn 1
/tbshiftunit/STIM/PropTimeD... 0 ps
/tbshiftunit/MeasurementIn... 66
/tbshiftunit/STIM/ResultV 1
DUT Signals
/tbshiftunit/A AAAA00005555FFFF
/tbshiftunit/B 0000000000000001
/tbshiftunit/C AAAA00005555FFFF
/tbshiftunit/Y 55540000AAABFFFE
/tbshiftunit/TbY 55540000AAABFFFE
Control Signals
/tbshiftunit/ShiftFN 01
/tbshiftunit/ExtWord 0
Now 10243.9 ns
Cursor 1 1982 ns
Cursor 2 2003.7 ns
1980 ns
1990 ns
2000 ns
2010 ns
2020 ns
2030 ns
2040 ns
1982 ns
21.7 ns
2003.7 ns

Figure 36: Waveform for t = 1980ns to 2040ns – Measurement 66 to 67, Propagation delay of 21.7ns (waveTSUSLL64.png)

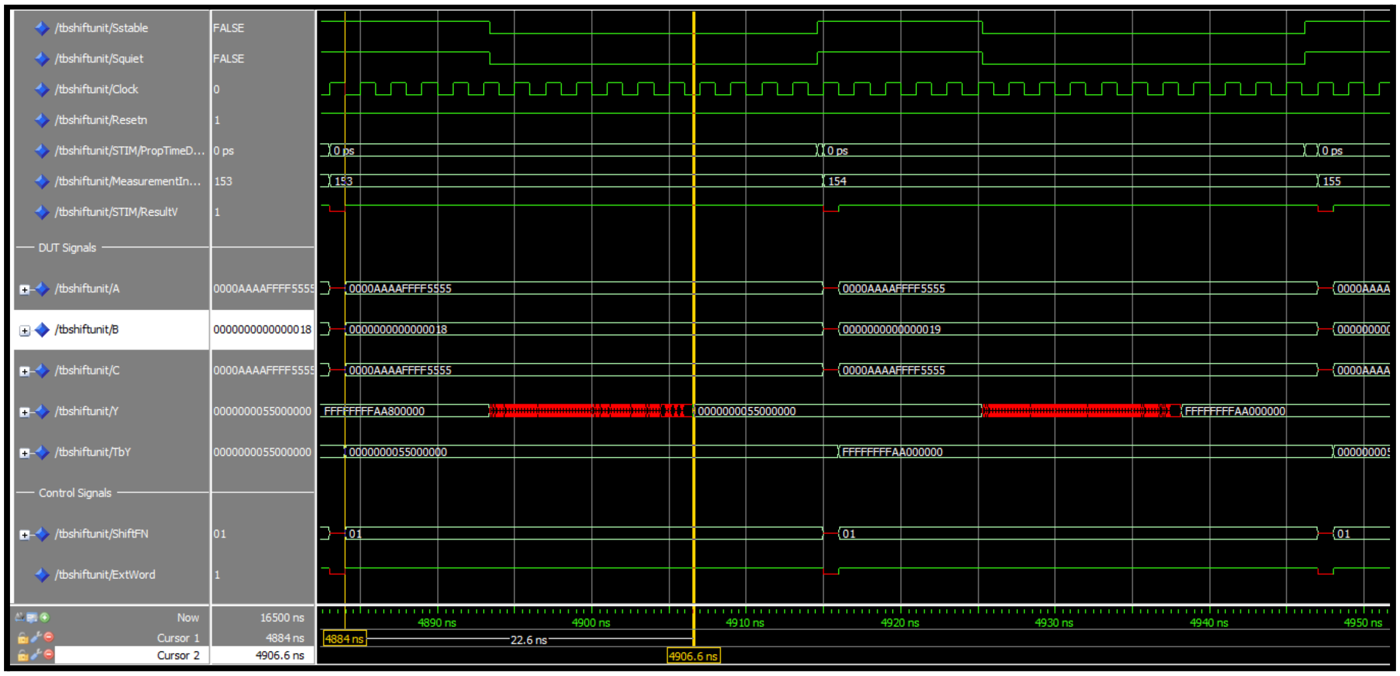

/tbshiftunit/Sstable FALSE
/tbshiftunit/Squiet FALSE
/tbshiftunit/Clock 0
/tbshiftunit/Resetn 1
/tbshiftunit/STIM/PropTimeD... 0 ps
/tbshiftunit/MeasurementIn... 153
/tbshiftunit/STIM/ResultV 1
DUT Signals
/tbshiftunit/A 0000AAAAFFFF5555
/tbshiftunit/B 0000000000000018
/tbshiftunit/C 0000AAAAFFFF5555
/tbshiftunit/Y 0000000055000000
/tbshiftunit/TbY 0000000055000000
Control Signals
/tbshiftunit/ShiftFN 01
/tbshiftunit/ExtWord 1
Now 16500 ns
Cursor 1 4884 ns
Cursor 2 4906.6 ns
4890 ns
4900 ns
4910 ns
4920 ns
4930 ns
4940 ns
4950 ns
4884 ns
22.6 ns
4906.6 ns

Figure 37: Waveform for t = 4886ns to 4950ns – Measurement 153 to 154, Propagation delay of 22.6ns (waveTSUSLL32.png)

For figure 38 and figure 39, we compare the 64-32 and 32-bit Shift Right Arithmetic circuits. For this test, we compare two measurements with the same shift value. We can determine that the propagation delay for the 64-bit circuit is 23.8ns, whereas the 32-bit circuit involves a propagation delay of only 20ns. As we are not basing any logic on the values within the vectors, we can dismiss any idea that the values affect the propagation delay and state that the propagation delay is only affected by the shift amount and sign extension.

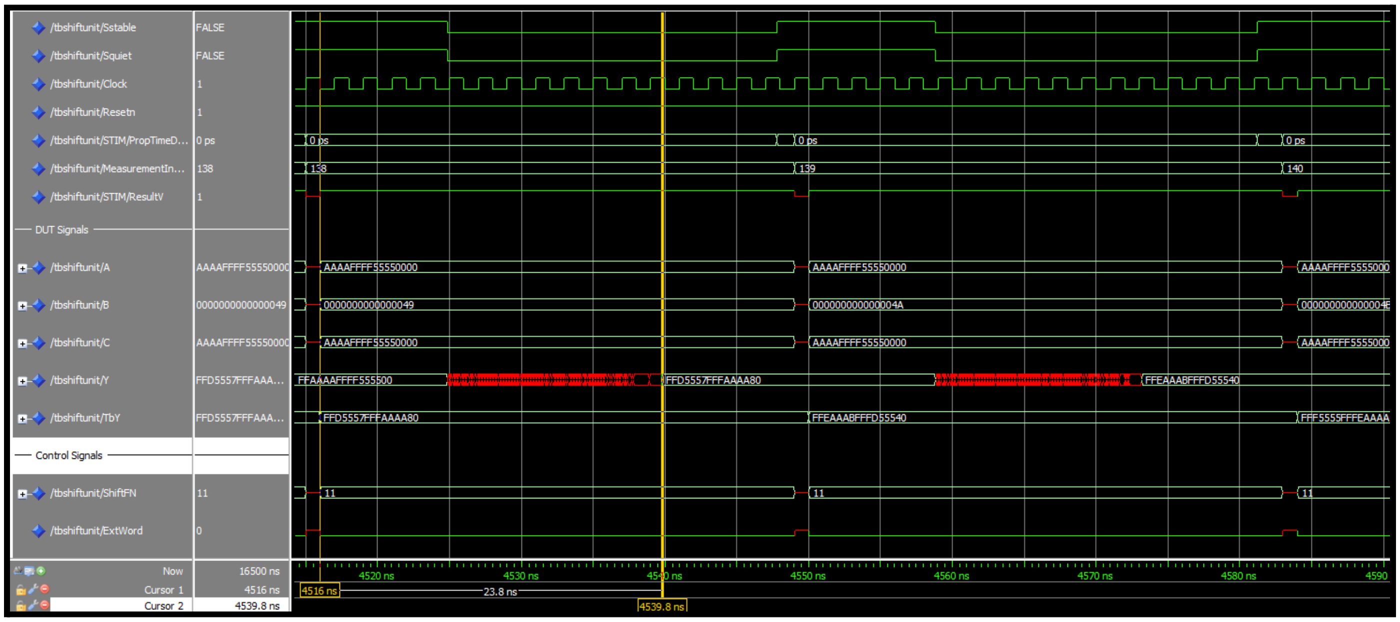


Figure 38: Waveform for t = 4516ns to 4590ns – Measurement 138 to 140, Propagation delay of 23.8ns (waveTSUSRA64.png)

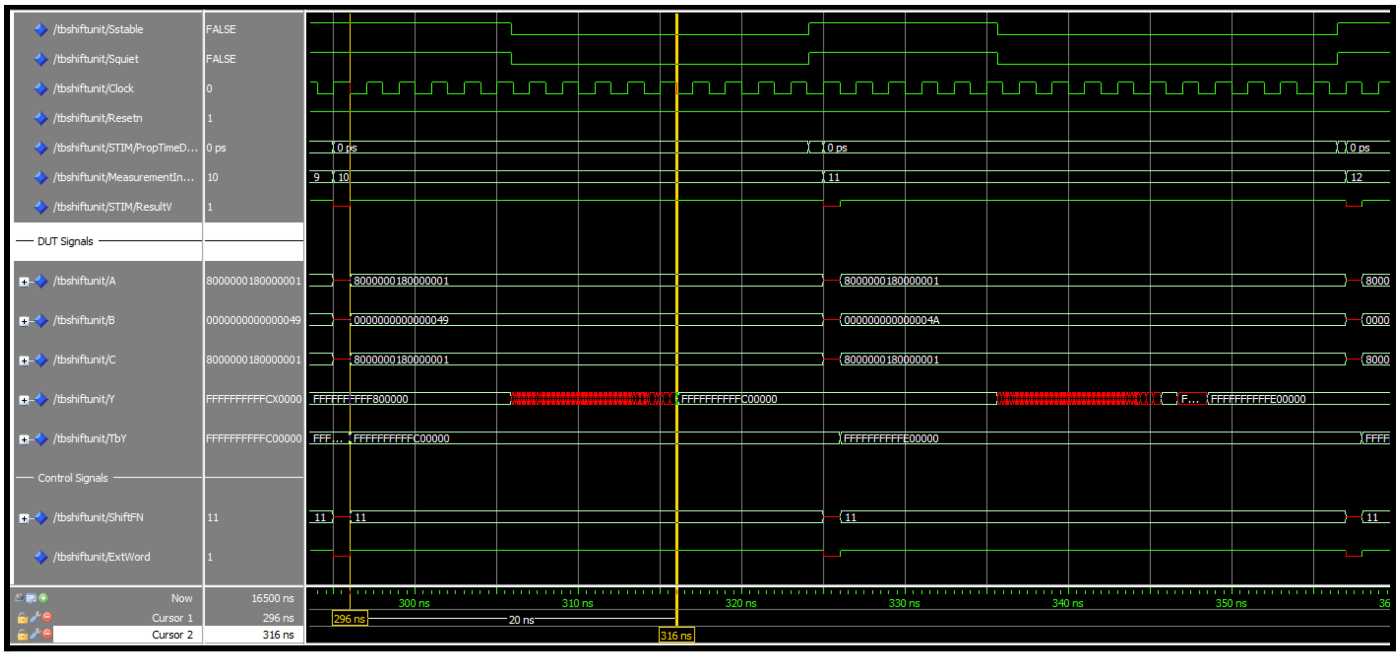


Figure 39: Waveform for t = 296ns to 360ns – Measurement 10 to 11, Propagation delay of 20ns (waveTSUSRA32.png)

Finally, we confirm that the 64-bit Shift Right Logical circuit involves a larger propagation delay compared with the 32-bit SRL circuit. In figure 40, we measure the 64-bit circuit to have a delay of 23.8ns whereas the 32-bit circuit in figure 41 involves a delay of 21.7ns.

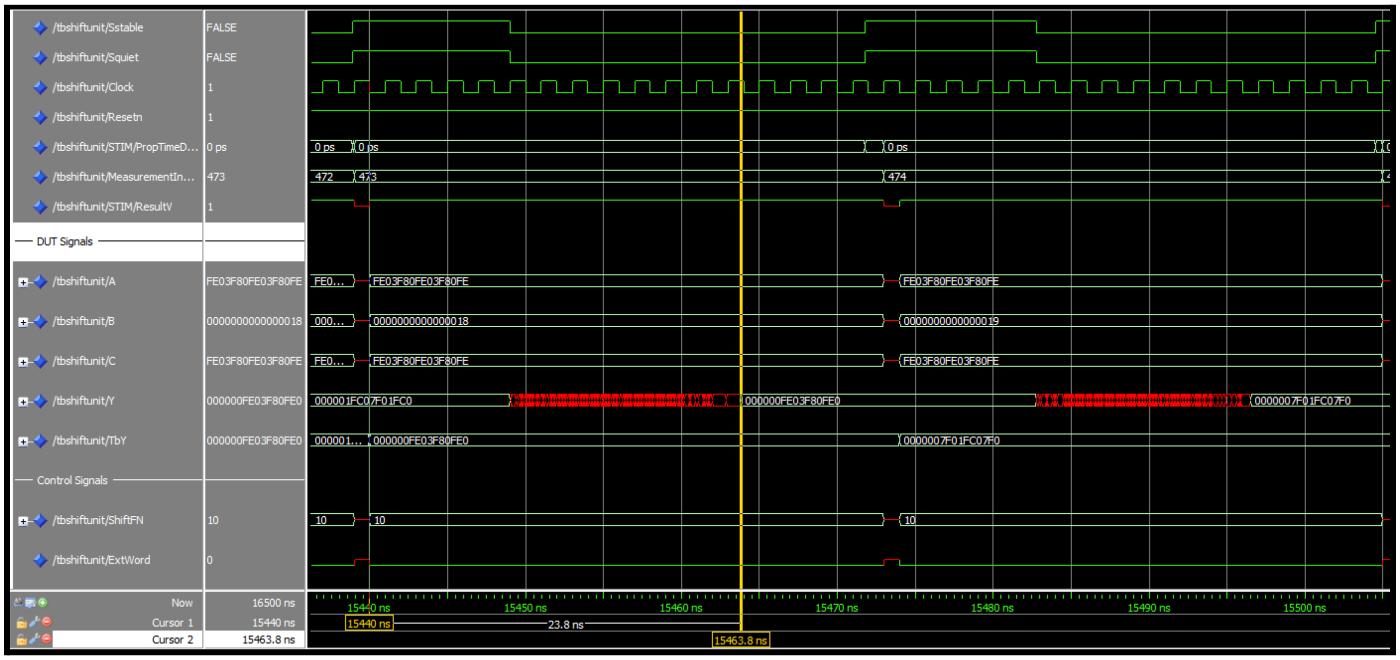

/tbshiftunit/Sstable FALSE
/tbshiftunit/Squiet FALSE
/tbshiftunit/Clock 1
/tbshiftunit/Resetn 1
/tbshiftunit/STIM/PropTimeD... 0 ps
/tbshiftunit/MeasurementIn... 473
/tbshiftunit/STIM/ResultV 1
DUT Signals
/tbshiftunit/A FE03F80FE03F80FE
/tbshiftunit/B 0000000000000018
/tbshiftunit/C FE03F80FE03F80FE
/tbshiftunit/Y 000000FE03F80FE0
/tbshiftunit/TbY 000000FE03F80FE0
Control Signals
/tbshiftunit/ShiftFN 10
/tbshiftunit/ExtWord 0
Now 16500 ns
Cursor 1 15440 ns
Cursor 2 15463.8 ns
23.8 ns

Figure 40: Waveform for t = 15440ns to 15504ns – Measurement 473 to 474, Propagation delay of 23.8ns (waveTSUSRL64.png)

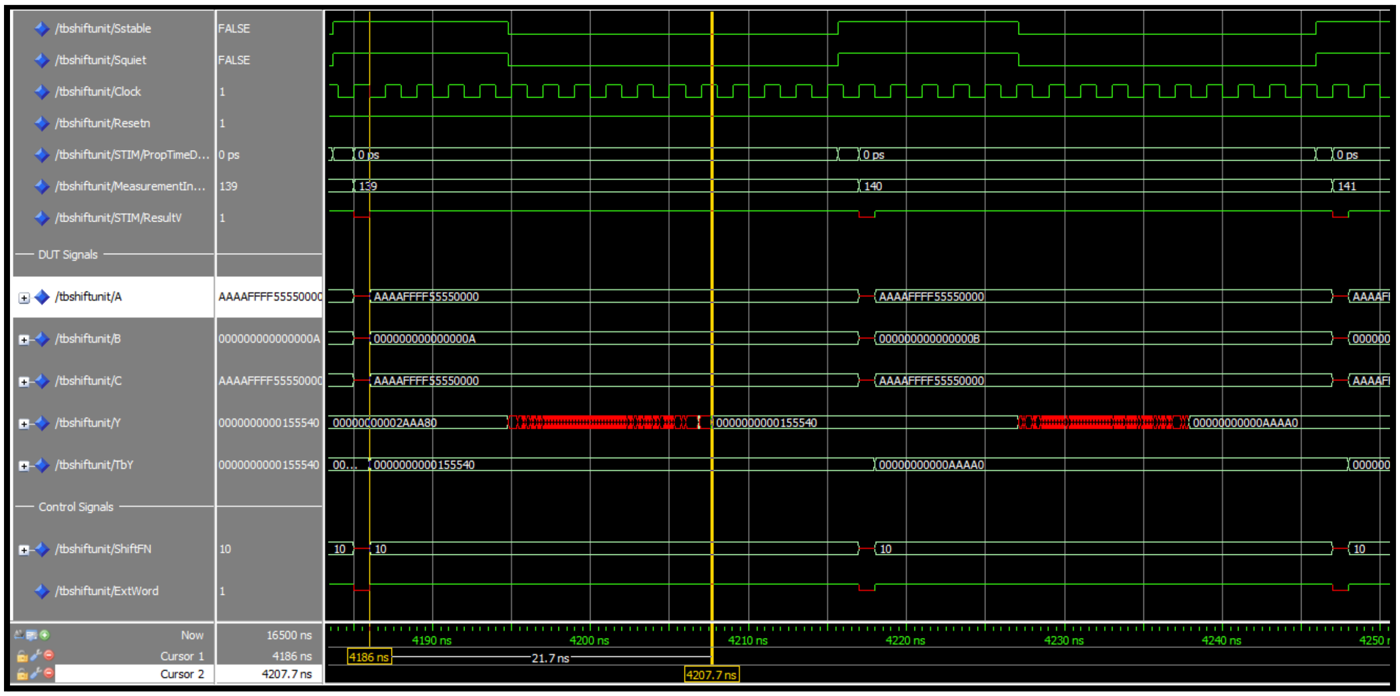

/tbshiftunit/Sstable FALSE
/tbshiftunit/Squiet FALSE
/tbshiftunit/Clock 1
/tbshiftunit/Resetn 1
/tbshiftunit/STIM/PropTimeD... 0 ps
/tbshiftunit/MeasurementIn... 139
/tbshiftunit/STIM/ResultV 1
DUT Signals
/tbshiftunit/A AAAAFFFF55550000
/tbshiftunit/B 000000000000000A
/tbshiftunit/C AAAAFFFF55550000
/tbshiftunit/Y 0000000000155540
/tbshiftunit/TbY 0000000000155540
Control Signals
/tbshiftunit/ShiftFN 10
/tbshiftunit/ExtWord 1
Now 16500 ns
Cursor 1 4186 ns
Cursor 2 4207.7 ns
21.7 ns

Figure 41: Waveform for t = 4183ns to 4250ns – Measurement 139 to 140, Propagation delay of 21.7ns (waveTSUSRL32.png)

In conclusion, we can determine that in general the 32-bit circuit will have a lower delay than the equivalent 64-bit circuit. We believe the cause of this to be that during 64 bit operation, the first multiplexor shifts between 0 16 32 and 48 bits and must be 4-1, but during a 32 bit operation the circuit only has to choose between two choices therefore lowing the time delay. Another hypothesis is that the shift_right and shift_left functions only shift one bit at a time, (ex. A shift_right(3) would shift to the right once three times sequentially) therefore resulting in a longer shift time for larger numbers of shifts.

4.4 Execution Unit:

In the final unit, we highlight the difference in timing delay between the hypothesized worst-case delay (subtraction of two identical numbers) and the best case delay of a simple logic unit operation. As we can see in figure 42, the propagation delay for the aforementioned subtraction operation results in a delay of 41.3ns. Contrasting this delay with the logic operation in figure 43, which has a delay of 16.1ns, we see that the worst case has more than twice the delay. Based on this observation, we conclude that this processor would be most efficient as a multi-cycle processor (as it currently is). If this same circuit were to be implemented in a single-cycle processor, this specific edge-case subtraction operation would significantly hinder the performance as the fastest possible clock would be based upon this operation.

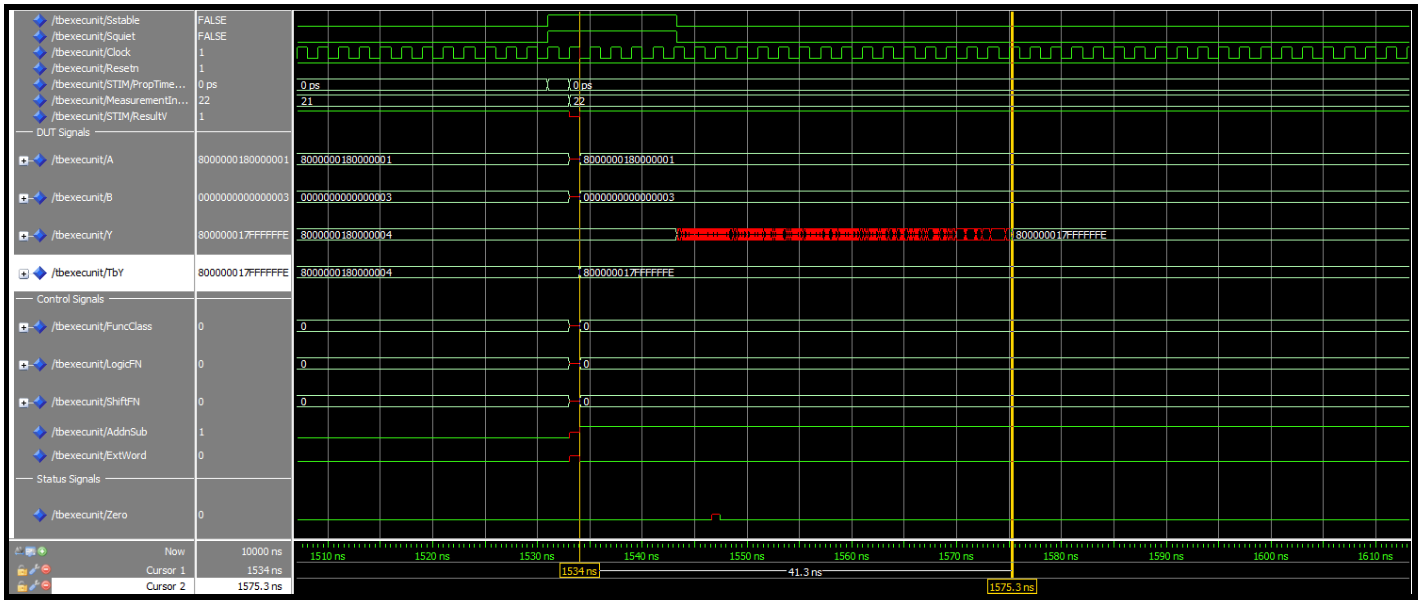


Figure 42: Waveform for t = 1509ns to 1610ns – Measurement 21 to 22, Propagation delay of 41.3ns (waveTEU1.png)

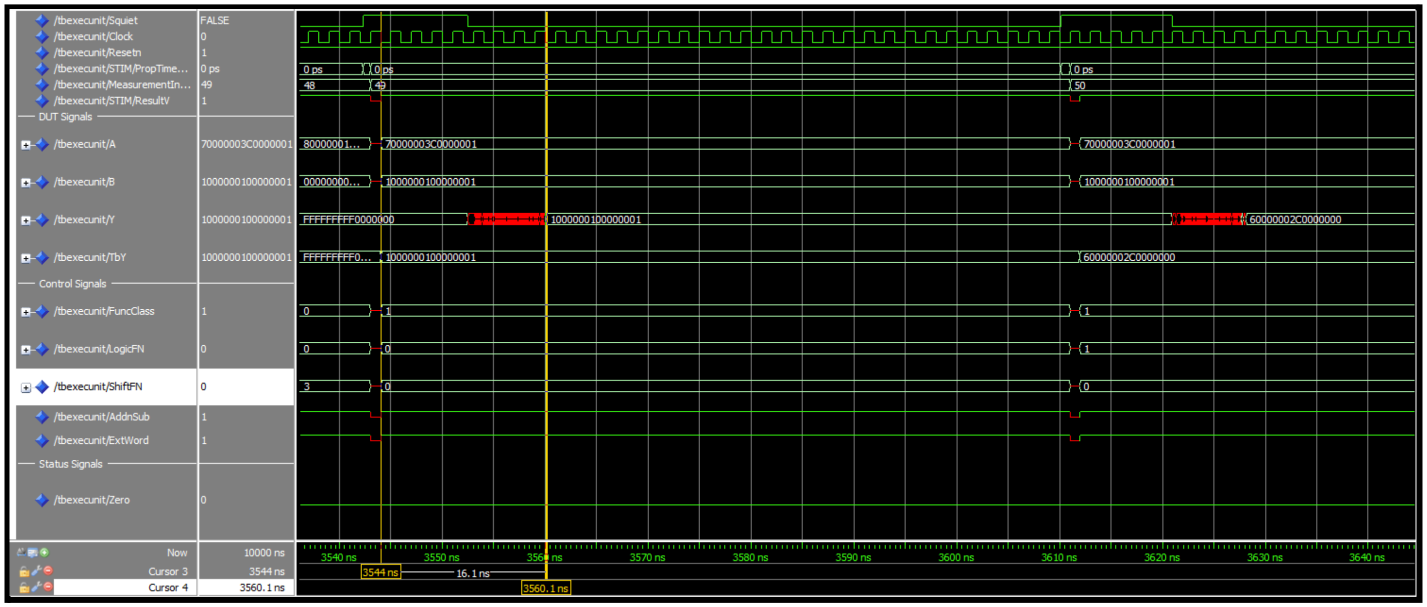


Figure 43: Waveform for t = 9710ns to 10065ns – Measurement 126 to 128, Propagation delays of 23.3ns and 20.5ns for Measurements 126 and 1127 respectively (waveTEU2.png)

Part 5: Conclusion

In this project design task, we created an execution unit in VHDL. We learned the importance of abstraction when creating the lower-level circuit components first and then instantiating them for the creation of the overall execution unit. This methodology also showed the necessity for testing components individually as errors in sub entities could be difficult to figure out when instantiated in a top-level entity. The use of scripts made testing very efficient and allowed redesigns to be tested efficiently if there was an issue with synthesizing or timing constraints. The timing testing demonstrated the importance of using efficient implementations of circuit elements especially when taking into consideration that some circuit elements will be used a large number of times in the overall design. Another benefit of having timing testing separate from functional testing was that it made it easier to determine any issues with the VHDL without having to deal with propagation and contamination delay.

Appendix

Source code	External: completeListing.pdf
Transcript File	External: summaryFile.pdf
Activity Log	External: FP-Log-G17-350-1221.xlsx
Build Archive	External: FP-Build-G17-350-1221.zip