Group #:

G

Final Project

Part 2

ENSC 350 2020

Last Name: Chao

SID: 3 0 1 3 1

0 6 2 4

Last Name: Kan

SID: 3 0 1 3 0

9 4 1 7

Last Name: Yonata

SID: 3 0 1 3 0

4 7 9 4

Last Name:

SID:

Table of Contents

Introduction

For this project, the main objective is to design a complete RISC-V Execution Unit that is compatible with RV64I. To achieve this, we have split the objective into four parts: designing the Logic Unit, designing the Arithmetic Unit, designing the Shift Unit, and then designing the Execution Unit. The Execution Unit consists of the Logic Unit, the Arithmetic Unit, and the Shift Unit. In this project, we will design all these different units and perform functional verification, synthesis, and timing verification to ensure that the different parts work properly. After making sure each part works properly, we will connect them all together and construct our Execution Unit circuit, performing functional verification, synthesis, and timing verification on the RISC-V Execution Unit. From this report, we will document the design entities of the individual entities, as well as the Execution Unit, and we will also discuss the observations and results from the functional and timing simulations.

Documentation for Design Entities

Logic Unit

The Logic Unit is designed in filename LogicUnit.vhd and it is inside the SourceCode folder.

Figure 1a shows the design entity of our Logic Unit. A and B are N-bit inputs of the Logic Unit, where N is set to 64-bits. Y is N-bits and it is the output of the Logic Unit. LogicFN is a 2-bit control signal that controls which result of the logical operations to output, or just passing the B value as the output.

The Logic Unit performs three logical operations on the two given operands A and B, which are both N bits. N is assigned as 64. The results of the logical operation will be passed as input of a 4-channel MUX and given the 2-bit control signal, LogicFn, the desired result will be outputted as Y. Depending on the given control signal, the operands A and B will be XOR, OR or AND together. If the given control signal is 0, the value in operand B will be outputted as Y.

```
Entity LogicUnit 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);
    -- Control signals
    LogicFN: in std_logic_vector(1 downto 0));
End Entity LogicUnit;
```

Figure 1a: Reference from LogicUnit.vhd

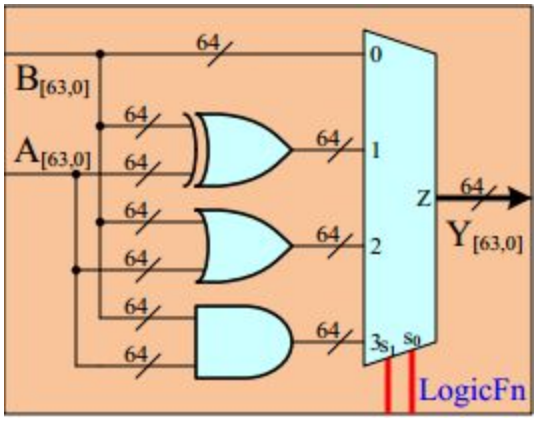


Figure 1b: Final Circuit diagram of the Logic Unit

Arithmetic Unit

The Adder in our Arithmetic Unit is designed in the file Adder.vhd. The Arithmetic Unit is designed in the file ArithUnit.vhd. Both files are inside the SourceCode folder.

Figure 2a shows the design entity of our Adder. Xin and Yin are our width-bit inputs, where width is defined to be 64. Cin is the carry in of our adder. Cout is our carry out signal and C63 is our width-1 bit output. S is the output of our Adder.

Figure 2b shows the design entity of our Arithmetic Unit. A and B are our N-bit inputs. AddY is the output of our Adder and Y is the output of our Arithmetic Unit. The Arithmetic Unit consists of three control signals: NotA, Addn'Sub, and ExtWord. NotA controls whether all the bits in input A gets flipped or not. Addn'Subcontrols whether the operation in the Adder is an addition or subtraction. ExtWord controls whether the output of the Arithmetic Unit gets sign extended or not, and this is because when this control signal is 1, the instruction is a word instruction. There are five status signals in the Arithmetic Unit. Cout is the carry out of the Adder. Ovfl shows whether the Adder operation causes an overflow or not. Zero shows whether the result of the Adder is zero or not. AltB shows whether A is less than B if A and B are both signed numbers. AltBu shows whether A is less than B if A and B are both unsigned numbers.

The Arithmetic Unit is responsible for performing arithmetic and comparison operations in the processor. The Arithmetic Unit takes in three input control signals: Addn'Sub, ExtWord and NotA. The Addn'Sub signal determines whether the two operands will be added to each other or subtracted. Subtraction is performed when Addn'Sub is 1, where the bits in B input will be flipped and Cin for our 64-bit adder will be 1, such that A-B becomes A + (2's complement of B). The resulting sum will be output as AddY. The Adder output is also passed on a 2-Channel MUX to determine whether it needs to be sign extended or not. It also goes through a NOR gate to determine whether the Zero signal should be on or off, where the

inputs are each of the bits of the Adder output. The resulting carry out (Cout) and overflow (Ovfl) goes through the AtlBu and AtlB. AltBu is determined by inverting Cout and AltB is determined by XOR operation of the sign bit of the Adder output and Ovfl. These are the status signals that determine the result of comparison operations such as SLT or SLTU. The output of the arithmetic unit is placed into Y.

```
Entity rippleadder is
    generic ( width : integer := 64 );
    port(
        Xin,Yin : in std_logic_vector(width-1 downto 0);
        Cin : in std_logic;
        S : out std_logic_vector(width-1 downto 0);
        Cout,C63 : out std_logic);
end rippleadder;
```

Figure 2a: Reference from Adder.vhd

```
Entity ArithUnit is
    Generic ( N : natural := 64 );
    Port (
        A, B : in std_logic_vector( N-1 downto 0 );
        AddY, Y : out std_logic_vector( N-1 downto 0 );
        -- Control signals
        NotA, AddnSub, ExtWord : in std_logic := '0';
        -- Status signals
        Cout, Ovfl, Zero, AltB, AltBu : out std_logic );
End Entity ArithUnit;
```

Figure 2b: Reference from ArithUnit.vhd

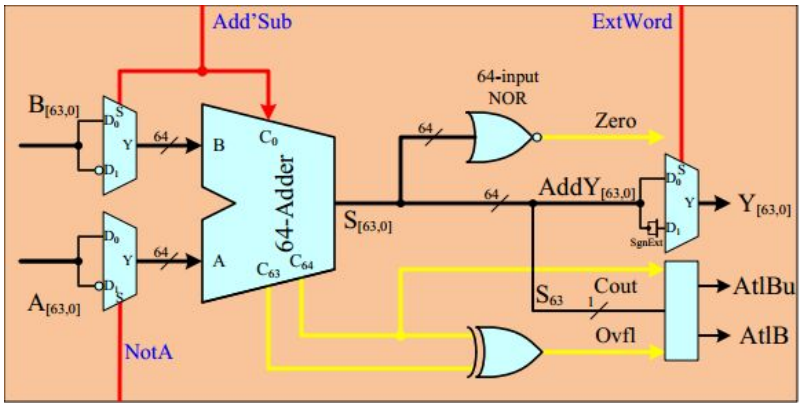


Figure 2c: Final Circuit Diagram of the Arithmetic Unit

The 64-bit Barrel Shifters:

SLL64:

The 64-bit Barrel Shifter for SLL operation is designed in the file SLL64.vhd and it is inside the SourceCode folder.

Figure 3a shows the design entity of our 64-bit Barrel Shifter for SLL operation. X is the N-bit input of the Barrel Shifter, where N is 64. Y is the output of the Barrel Shifter and ShiftCount is a 6-bit unsigned integer that determines how many bits the input is shifting left by.

Our design of the circuit for SLL operation uses barrel shifter method to perform the shifts. The input can be shifted by 63 bits to the left at max and bit shifting can be split up since the total shift amount can be split into the sum of smaller shift amounts. Our circuit is implemented with 3 levels of 4-Channel MUXes, where the first level shifts the input left by either 0, 16, 32, or 48 bits, the second level shifts the input left by either 0, 4, 8, or 12 bits, and the third level shifts the input left by either 0, 1, 2, or 3 bits. The 5th bit and the 4th bit of ShiftCount controls the shift amount of the first level shift operation. The 3rd bit and the 2nd bit of ShiftCount controls the shift amount of the second level shift operation. The 1st bit and the least significant bit of ShiftCount controls the shift amount of the third level shift operation.

```
Entity SLL64 is
Generic ( N : natural := 64 );
Port ( X : in std_logic_vector( N-1 downto 0 );
    Y : out std_logic_vector( N-1 downto 0 );
    -- Control signal
    ShiftCount : in unsigned( integer(ceil(log2(real(N))))-1 downto 0 ) );
End Entity SLL64;
```

Figure 3a: Reference from SLL64.vhd

SRL64:

The 64-bit Barrel Shifter for SRL operation is designed in the file SRL64.vhd and it is inside the SourceCode folder.

Figure 3b shows the design entity of our 64-bit Barrel Shifter for SRL operation. X is the N-bit input of the Barrel Shifter, where N is 64. Y is the output of the Barrel Shifter and ShiftCount is a 6-bit unsigned integer that determines how many bits the input is shifting right by.

Our design of the circuit for SRL operation uses barrel shifter method to perform the shifts. The input can be shifted by 63 bits to the right at max and bit shifting can be split up since the total shift amount can be split into the sum of smaller shift amounts. Our circuit is implemented with 3 levels of 4-Channel MUXes, where the first level shifts the input right by

either 0, 16, 32, or 48 bits, the second level shifts the input right by either 0, 4, 8, or 12 bits, and the third level shifts the input right by either 0, 1, 2, or 3 bits. The 5th bit and the 4th bit of ShiftCount controls the shift amount of the first level shift operation. The 3rd bit and the 2nd bit of ShiftCount controls the shift amount of the second level shift operation. The 1st bit and the least significant bit of ShiftCount controls the shift amount of the third level shift operation.

```
Entity SRL64 is
Generic ( N : natural := 64 );
Port ( X : in std_logic_vector( N-1 downto 0 );
    Y : out std_logic_vector( N-1 downto 0 );
    -- Control signal
    ShiftCount : in unsigned( integer(ceil(log2(real(N))))-1 downto 0 ) );
End Entity SRL64;
```

Figure 3b: Reference from SRL64.vhd

SRA64:

The 64-bit Barrel Shifter for SRA operation is designed in the file SRA64.vhd and it is inside the SourceCode folder.

Figure 3c shows the design entity of our 64-bit Barrel Shifter for SRL operation. X is the N-bit input of the Barrel Shifter, where N is 64. Y is the output of the Barrel Shifter and ShiftCount is a 6-bit unsigned integer that determines how many bits the input is shifting right by.

Our design of the circuit for SRA operation uses barrel shifter method to perform the shifts, together with VHDL's shift_right() function to ensure that the input gets shifted right while having the most significant bit sign extending the left bits. The input can be shifted by 63 bits to the right at max and bit shifting can be split up since the total shift amount can be split into the sum of smaller shift amounts. Our circuit is implemented with 3 levels of 4-Channel MUXes, where the first level shifts the input right by either 0, 16, 32, or 48 bits, the second level shifts the input right by either 0, 4, 8, or 12 bits, and the third level shifts the input right by either 0, 1, 2, or 3 bits. All the levels will have the most significant bit sign extending the left bits. The 5th bit and the 4th bit of ShiftCount controls the shift amount of the first level shift operation. The 3rd bit and the 2nd bit of ShiftCount controls the shift amount of the second level shift operation. The 1st bit and the least significant bit of ShiftCount controls the shift amount of the third level shift operation. The resulting output Y will be shifted right by the ShiftCount and sign extended up to 64 bits.

```
Entity SRA64 is
Generic ( N : natural := 64 );
Port ( X : in std_logic_vector( N-1 downto 0 );
    Y : out std_logic_vector( N-1 downto 0 );
```

```
    -- Control signal
    ShiftCount : in unsigned( integer(ceil(log2(real(N))))-1 downto 0 ) );
End Entity SRA64;
```

Figure 3c: Reference from SRA64.vhd

The Shift Unit

The Shift Unit is designed in the file ShiftUnit.vhd and it is inside the SourceCode folder.

Figure 4a shows the design entity of the Shift Unit. A, B, and C are N-bit inputs of the Shift Unit, where N is 64. Y is the output of the Shift Unit. There are two control signals, ShiftFN and ExtWord. ShiftFN is a 2-bit signal that controls which type of shift operation will be outputted. ShiftFN(1) usually determines whether the instruction is a shift left instruction or a shift right instruction. ShiftFN(0) usually determines which type of shift left or shift right instruction it is, depending on the bit value of ShiftFN(1). ExtWord is a control signal that controls whether the signal needs to be sign extended or not.

For the Shift Unit, A is the input that contains the values that will be shifted and the lower bits of input B will be the ShiftCount. The Extract block is a simple circuit that determines the shifting amount. If the operation is to shift a 32-bit value, then the shifting operation will only shift a value by up to 31 bits, which means Extract will only extract the least significant 5 bits of input B and then pad it to 6 bits for ShiftCount. If the operation is to shift a 64-bit value, then the shifting operation shifts a value by up to 63 bits, which means Extract will extract the least significant 6 bits of input B. The specific VHDL statement of the Extract is shown in Figure 4b. There is a 2-channel MUX for A that determines whether the output will be the input of A, or the result of input of A after it undergoes SwapWord, which replaces the upper word of A with the lower word of A and the new lower word of A is replaced with 32 bits of zeroes. The input selector of the MUX is the result of ShiftFN(1) AND ExtWord, which means that SwapWord is only necessary when we are performing shift right word instruction. The specific VHDL statement of the SwapWord MUX is shown in Figure 4c. For shift left instruction, the output of SLL64 goes into a MUX and the input selector ShiftFN(0) determines whether C is outputted or the output of SLL64 is outputted. After that MUX, another MUX will determine whether the signal needs to be sign extended or not using the signal ExtWord, which depends on whether the instruction is a word instruction or not. For shift right instructions, the output of SRL64 and SRA64 goes into a MUX and the input selector ShiftFN(0) determines whether to output the result of SRL64 or the result of SRA64. The output of that MUX then goes through another MUX which determines whether the result needs to be sign extended or not. If the result has to be sign extended, that means that it is a shift right word instruction, which means we need to move the upper word of the result back to the lower word first, and then perform sign extension. The sign extended results from both

the shift left instruction side and the shift right instructions side goes into a final MUX, and input selector ShiftFN(1) determines what signal is outputted as the output of Y.

```
Entity ShiftUnit is
Generic ( N : natural := 64 );
Port ( A, B, C : in std_logic_vector( N-1 downto 0 );
    Y : out std_logic_vector( N-1 downto 0 );
    -- Control Signals
    ShiftFN : in std_logic_vector( 1 downto 0 );
    ExtWord : in std_logic );
End Entity ShiftUnit;
```

Figure 4a: Reference from ShiftUnit.vhd

```
shiftCount <= unsigned("0" & B(4 downto 0)) when ExtWord = '1' else
              unsigned(B(5 downto 0));
```

Figure 4b: Extract block VHDL statement

```
A_s <= ShiftFn(1) AND ExtWord;
    A_Y1 <= A(31 downto 0)& x"00000000" when A_s = '1' else
            A;
```

Figure 4c: SwapWord MUX VHDL statement

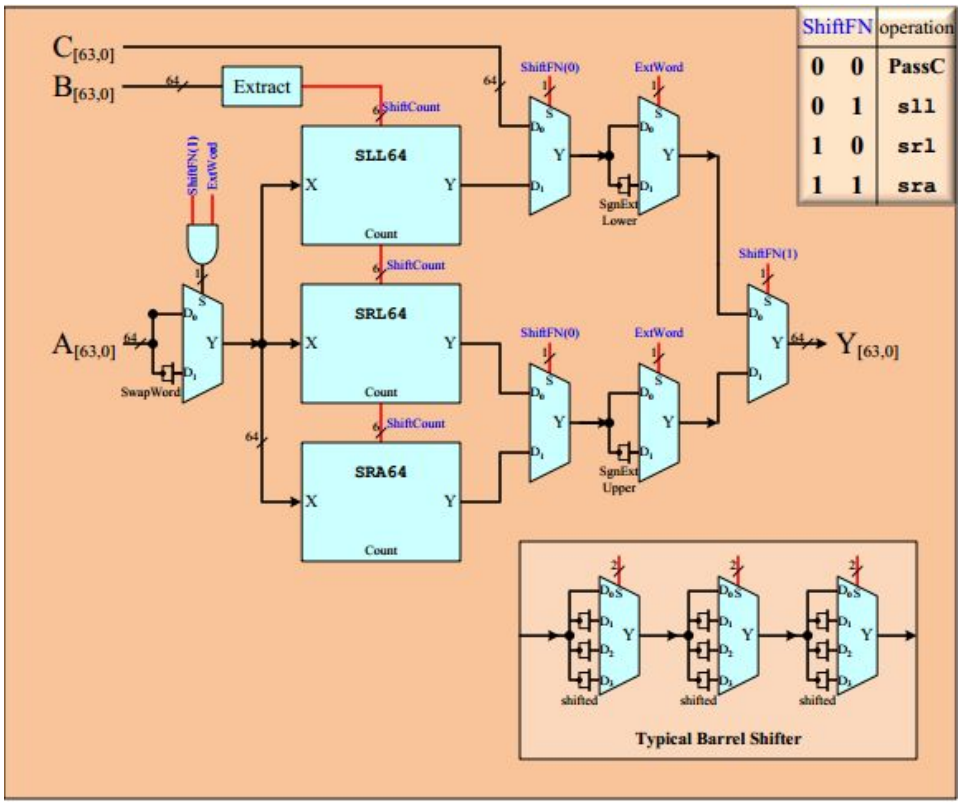


Figure 4d: Final Circuit Diagram of the Shift Unit and a Typical Barrel Shifter

Execution Unit

The Execution Unit is designed in the file ExecUnit.vhd and it is inside the SourceCode folder.

Figure 5a shows the design entity of the Shift Unit. A and B are N-bit inputs of the Execution Unit, where N is 64. Y is the output of the Shift Unit. There are six control signals: NotA, FuncClass, LogicFN, ShiftFN, Addn'Sub, and ExtWord. NotA controls whether all the bits in input A gets flipped or not. FuncClass is a 2-bit control signal that determines which type of operation it is, whether it is comparison, shift/arithmetic, or logical, and controls what to output. LogicFN controls which logical operation result is outputted. ShiftFN controls whether the arithmetic result is outputted or the shift instruction results is outputted. Addn'Sub controls whether the arithmetic operation is an addition or subtraction. ExtWord determines whether the instruction is a word instruction or not, and thus controls whether the result needs to be sign extended to 64 bits or not. There are also three status signals in the Execution Unit: Zero, AltB, and AltBu.

The Execution Unit consists of three sub-entities: the Arithmetic Unit, the Shift Unit, and the Logic Unit. The Arithmetic Unit's functional behaviour is described in the Arithmetic Unit section of the design entity documentation. The Shift Unit's functional behaviour is described in the Shift Unit section of the design entity documentation. The Logic Unit's functional behaviour is described in the Logic Unit section of the design entity documentation. Other than the three sub-entities, the Execution Unit also contains a 4-Channel MUX, where the inputs consists of the AltB output, the AltBu output, the output of the Shift Unit, and the output of the Logic Unit and the input selector FuncClass will determine what to output. The AltB and AltBu results will first be padded with zeros up to 64 bits first before outputting.

```
Entity ExecUnit 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 );
    -- Control signals
    NotA : in std_logic := '0';
    FuncClass : in std_logic_vector( 1 downto 0 );
    LogicFN : in std_logic_vector( 1 downto 0 );
    ShiftFN : in std_logic_vector( 1 downto 0 );
    AddnSub, ExtWord : in std_logic := '0';
    -- Status signals
    Zero, AltB, AltBu : out std_logic
    );
End Entity ExecUnit;
```

Figure 5a: Reference from ExecUnit.vhd

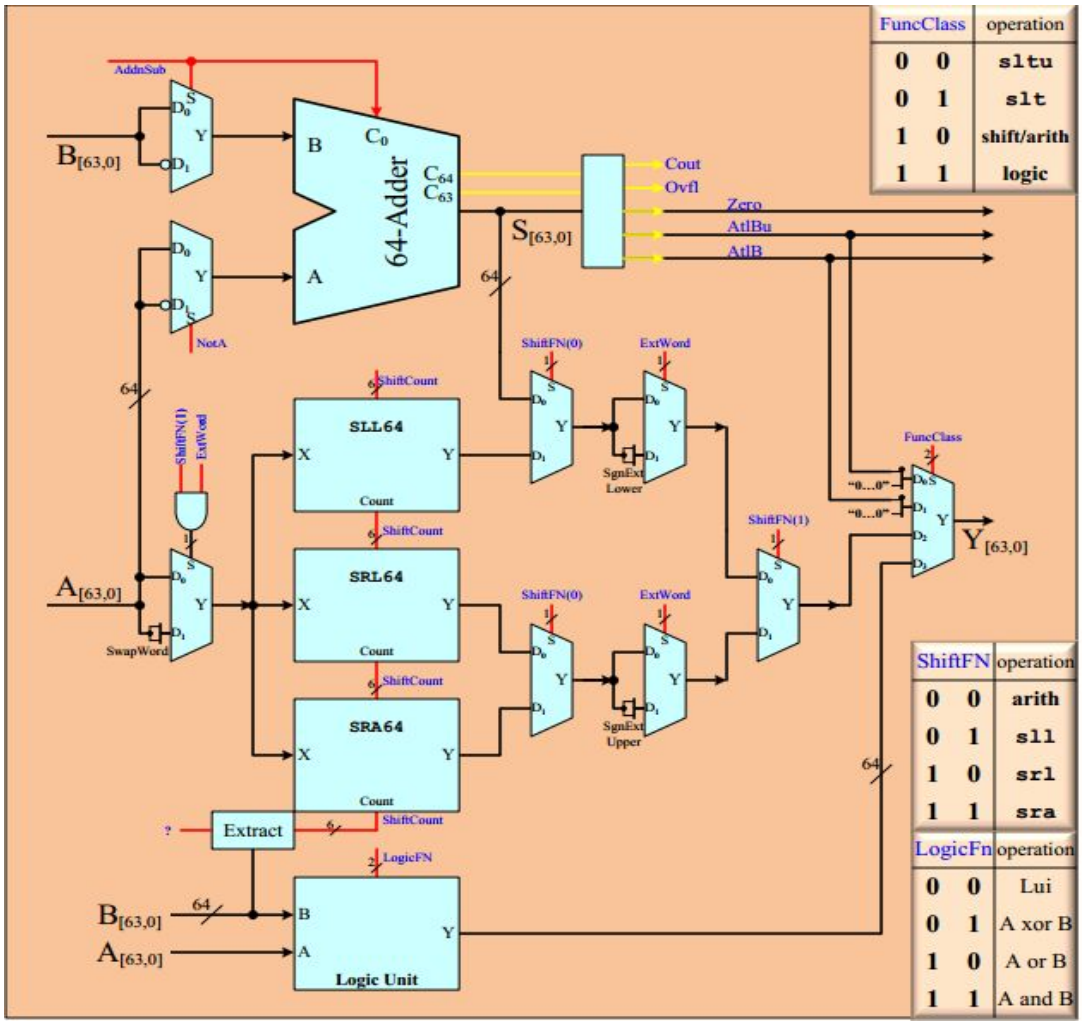
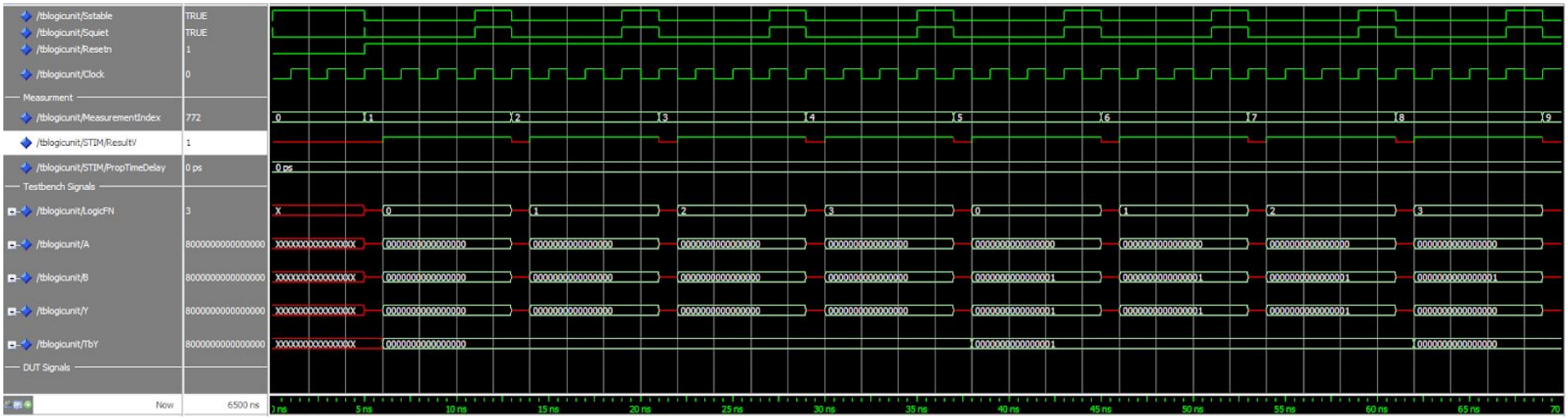


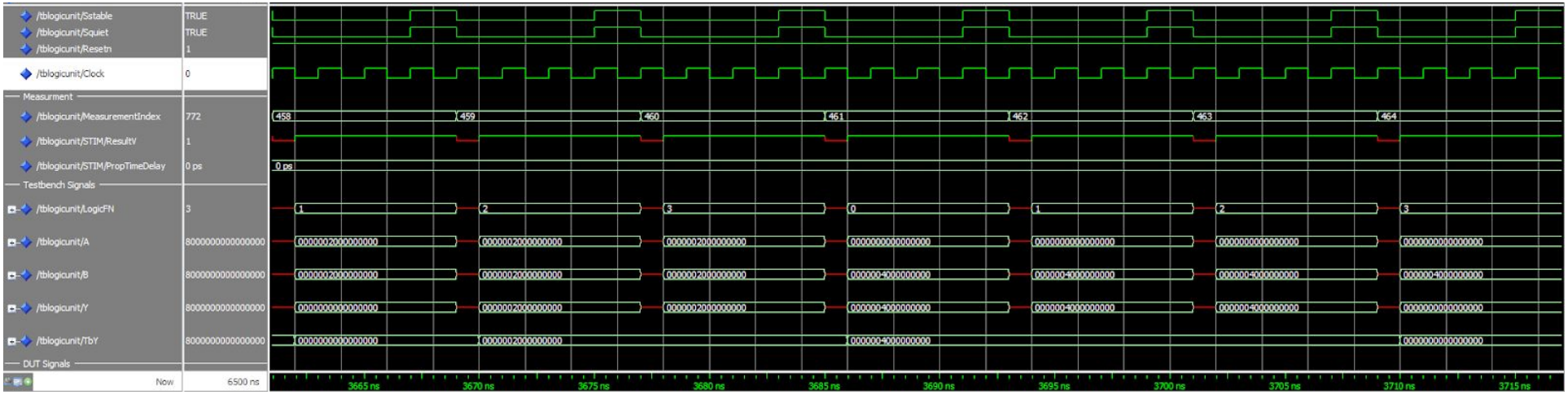
Figure 5b: Final Circuit Diagram of the Execution Unit

Functional Simulation



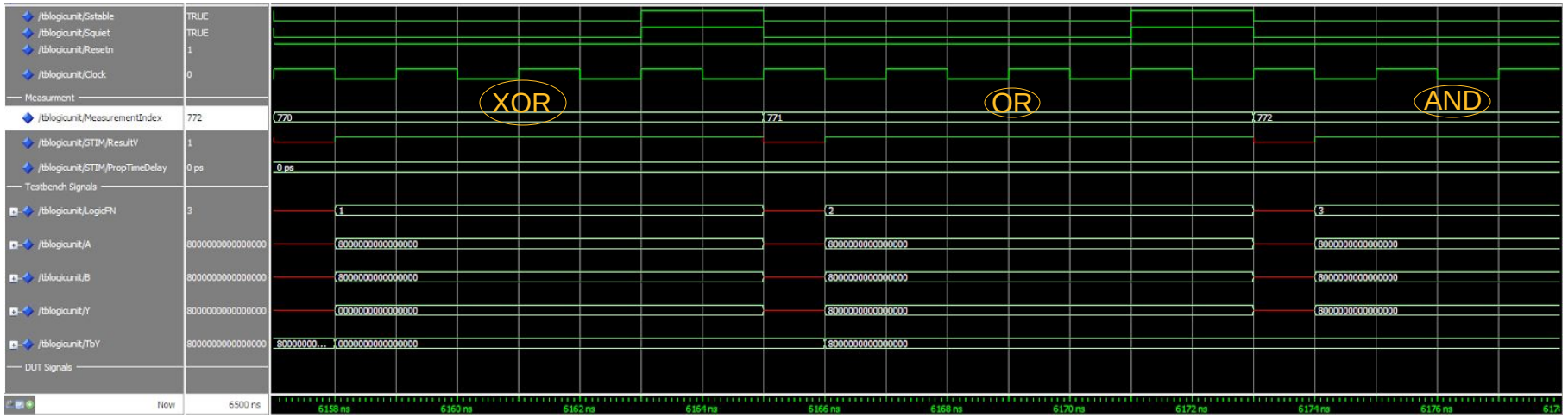
Functional simulation: Logic Unit 1

The diagram shows the values of the first 8 measurements. Each of the Y values and TbY values match for each measurement. From measurements 6-8, when LogicFN is 0, the value of B is passed as output Y. When LogicFN is 1, output of Y will be the result of a XOR operation of A and B. When LogicFN is 2, output of Y will be the result of A OR B. When LogicFN is 3, the output of Y will be the result of A AND B.



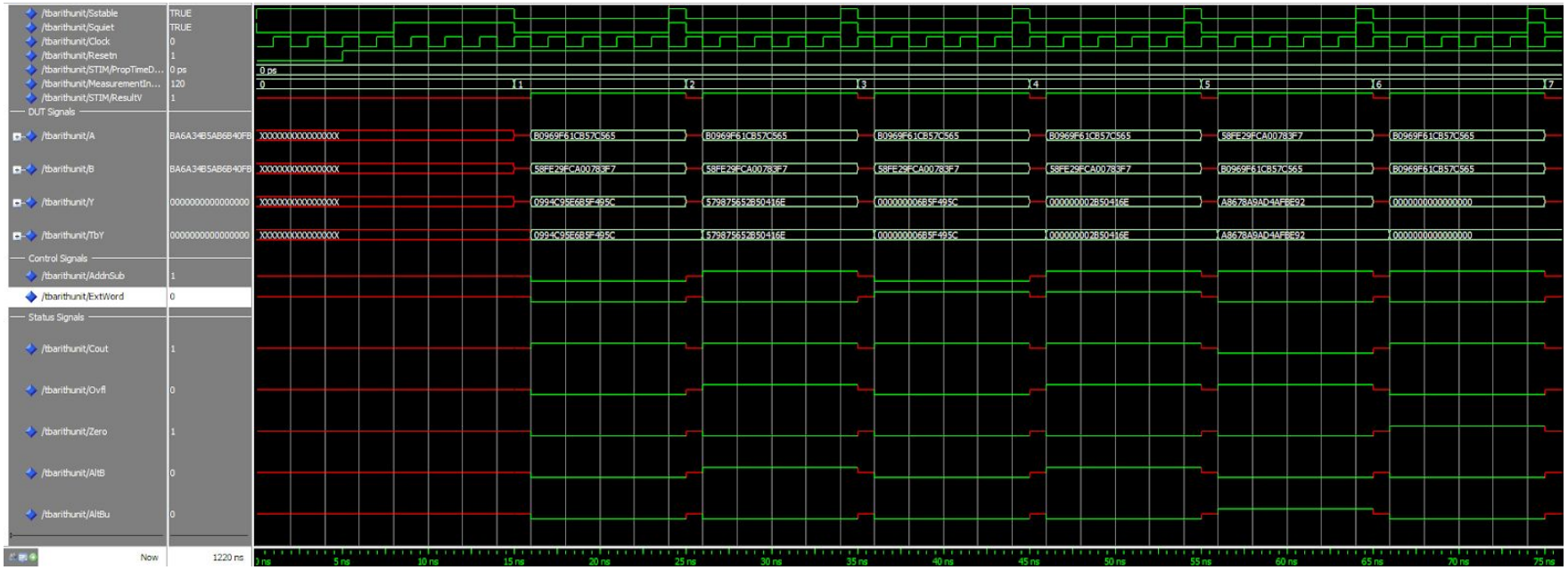
Functional simulation: Logic Unit 2

The diagram shows measurements 458 up to the end of 464. Each of the Y values and TbY values match for each measurement. From this screenshot, when LogicFN is 0, the value of B is passed as output Y. When LogicFN is 1, output of Y will be the result of A XOR B. When LogicFN is 2, output of Y will be the result of A OR B. When LogicFN is 3, the output of Y will be the result of A AND B.



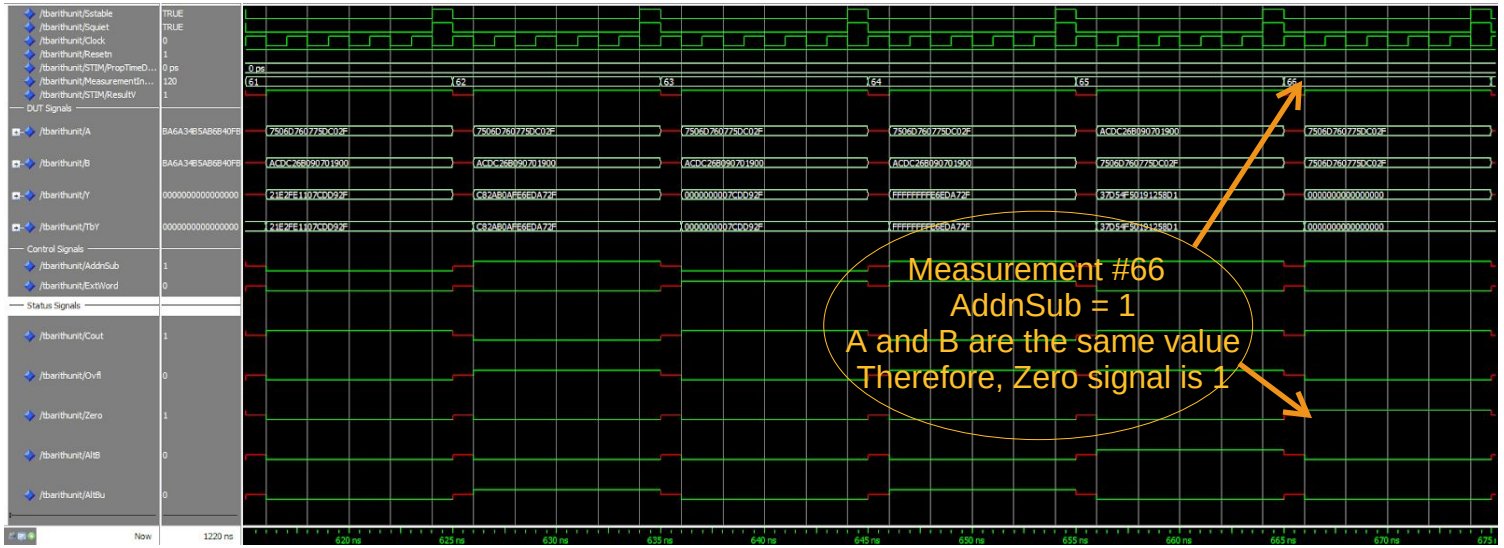
Functional simulation: Logic Unit 3

The screenshot shows the last three measurements 770, 771 and 772. When LogicFN is 0, the value of B is passed as output Y. When LogicFN is 1, output of Y will be the result of A XOR B. When LogicFN is 2, output of Y will be the result of A OR B. When LogicFN is 3, the output of Y will be the result of A AND B.



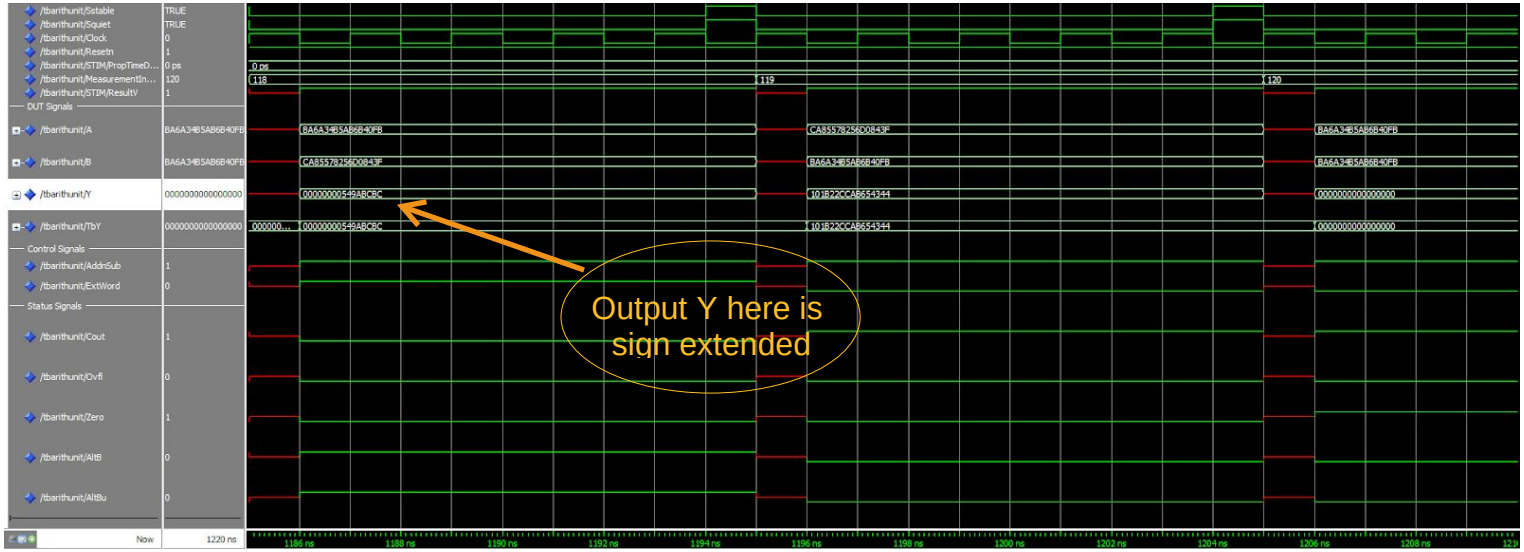
Functional simulation: Arithmetic Unit 1

The diagram shows the first 6 measurements of our Arithmetic Unit. All our output Y values match the given TbY values and the status signal outputs are correct. For example in measurement 5, when AddnSub is 1, the result of AltBu is the result of NOT Y, which is the case since Cout is 0 and AltBu is 1.



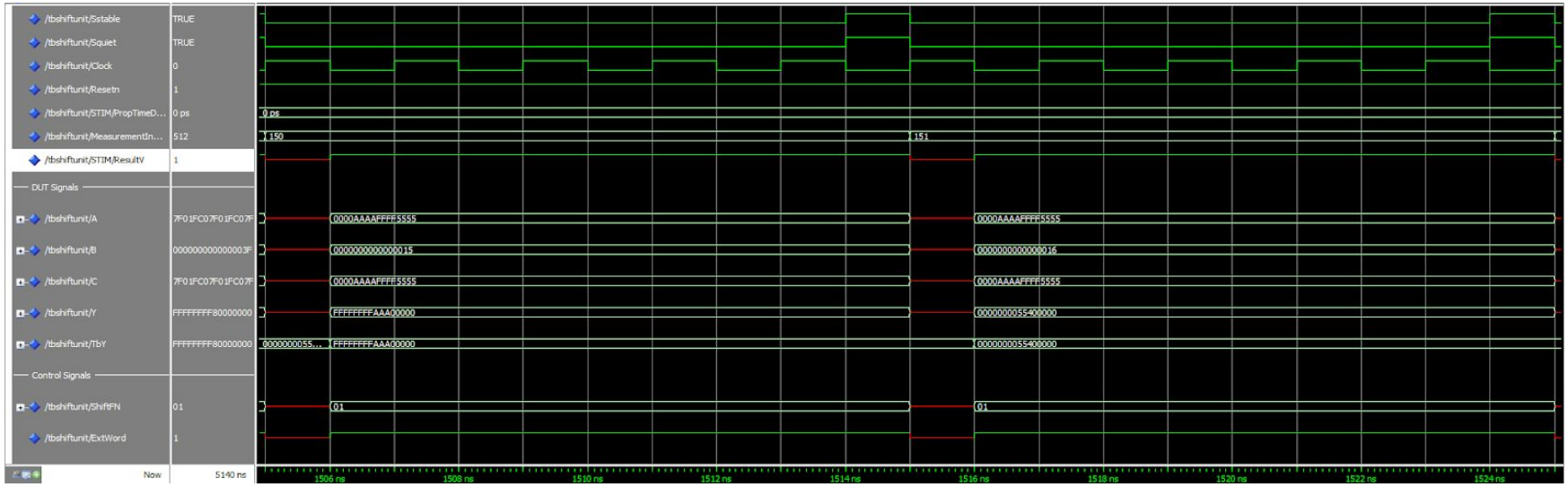
Functional simulation: Arithmetic Unit 2

The diagram shows measurements 61 up to 66 of our Arithmetic Unit. All our output Y values match the given TbY values and the status signal outputs are correct. For example in measurement 66, both operand A and B have the same value and AddnSub, thus meaning that status signal Zero should be 1 from this subtraction.



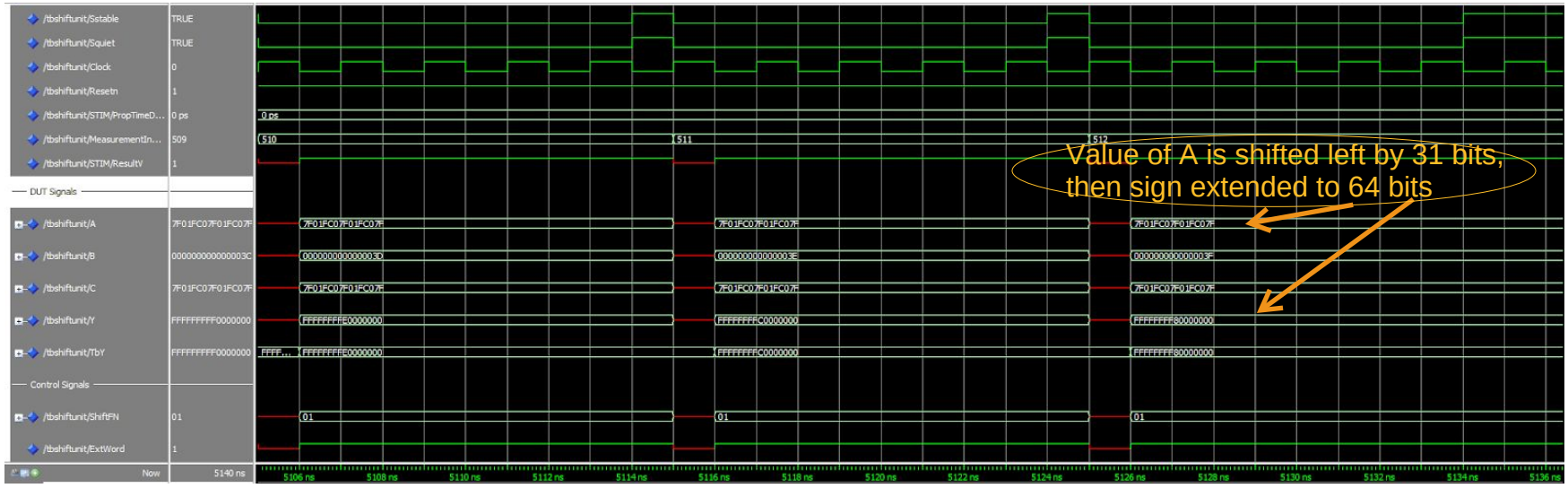
Functional simulation: Arithmetic Unit 3

The diagram shows the last three measurements of our Arithmetic Unit. Our output Y values match the TbY output values and the status signal outputs are correct, meaning that the arithmetic unit is functioning properly. For example in measurement 118, ExtWord is 1 which means that it is a word instruction, therefore the output Y is sign extended to 64 bits.



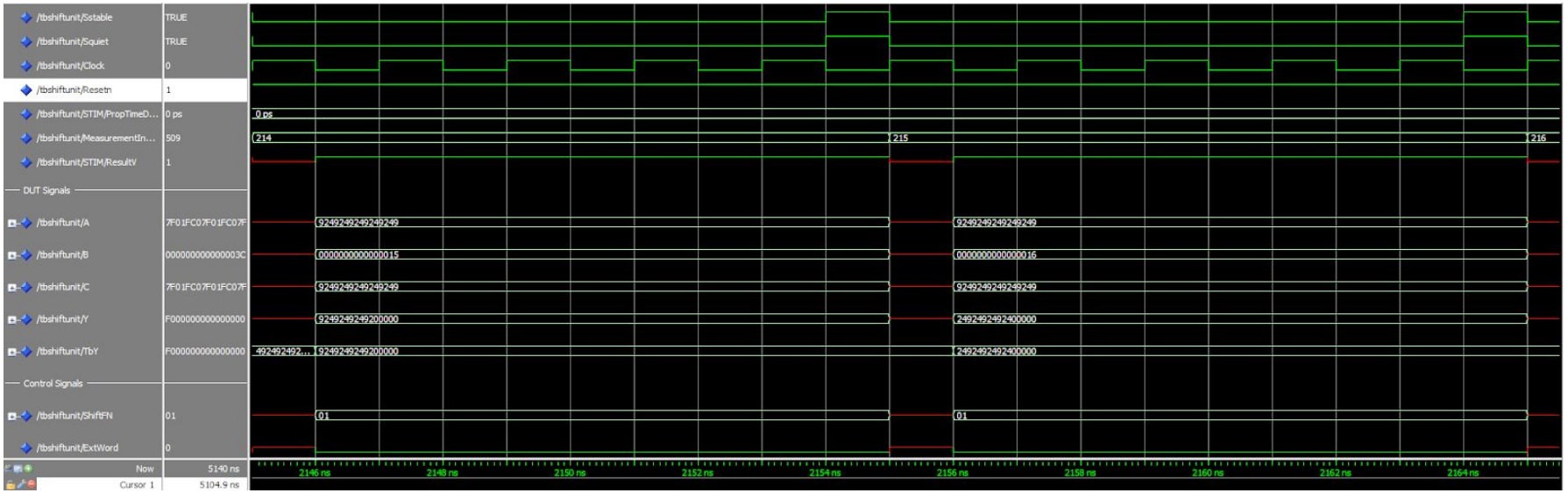
Functional simulation: Shift Unit SLL32 1

This screenshot shows the measurements of 150 and 151. The output Y value matched with the TbY value, meaning that the operation is functioning properly. For example in measurement 151, ExtWord is 1 and value of A is shifted left by 22 bits, so only the lower 32 bits of A and shift that part by 22 bits to the left. The result will be 55400000 in hexadecimal and it will be sign extended back to 64 bit.



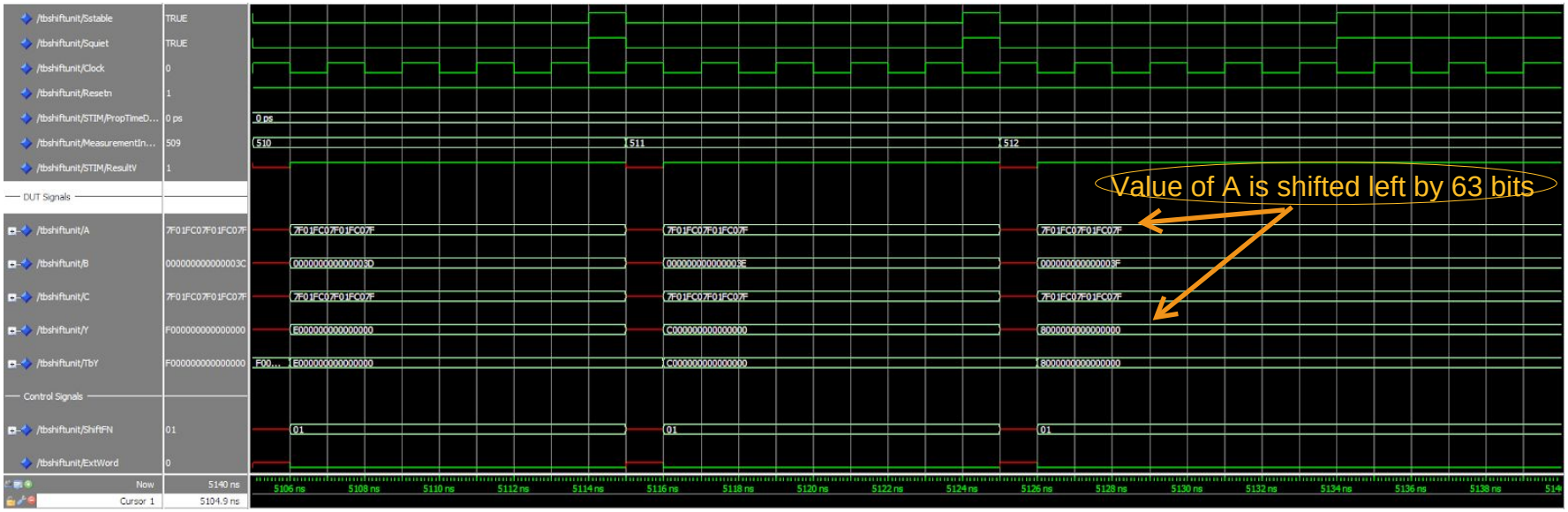
Functional simulation: Shift Unit SLL32 2

This screenshot shows the last three measurements. The output Y value matched with the TbY value, meaning that the operation is functioning properly. For example in measurement 512, ExtWord is 1 and value of A is shifted left by 31 bits, since shift left instruction will be performed on a word. The lower 32 bits of A will be shifted by 31 its to the left. The result will be 80000000 in hexadecimal and it will be sign extended back to 64 bit.



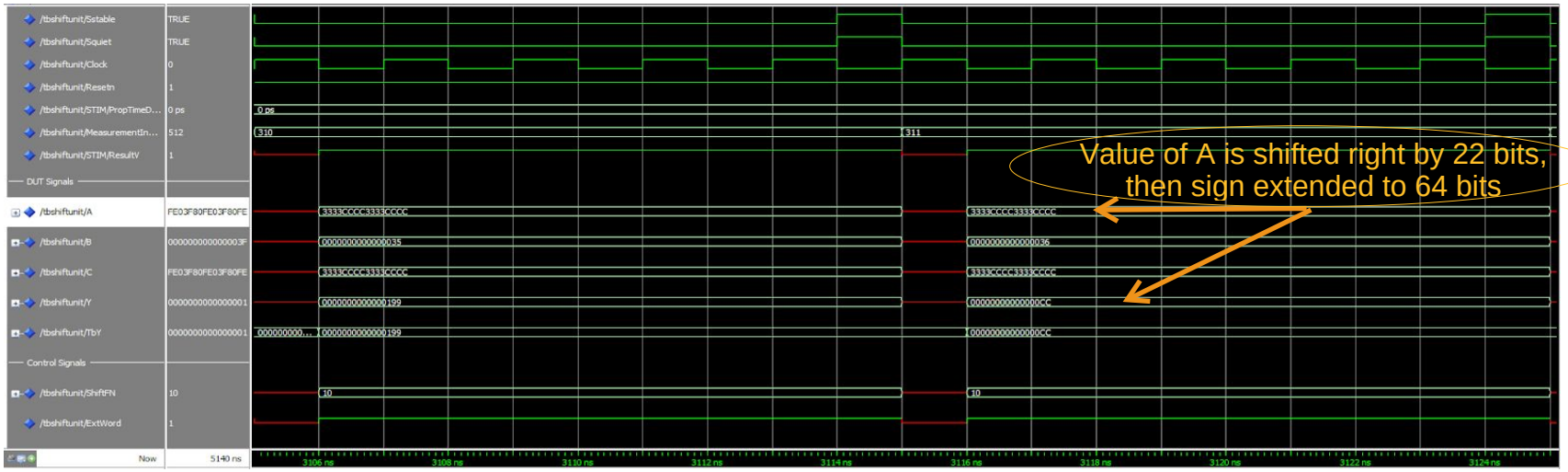
Functional simulation: Shift Unit SLL64 1

This screenshot shows the measurements of 214 and 215. The output Y value matched with the TbY value, meaning that the operation is functioning properly. For example in measurement 215, the value of A is shifted left by 22 bits. The result Y is correct and matches with the TbY value.



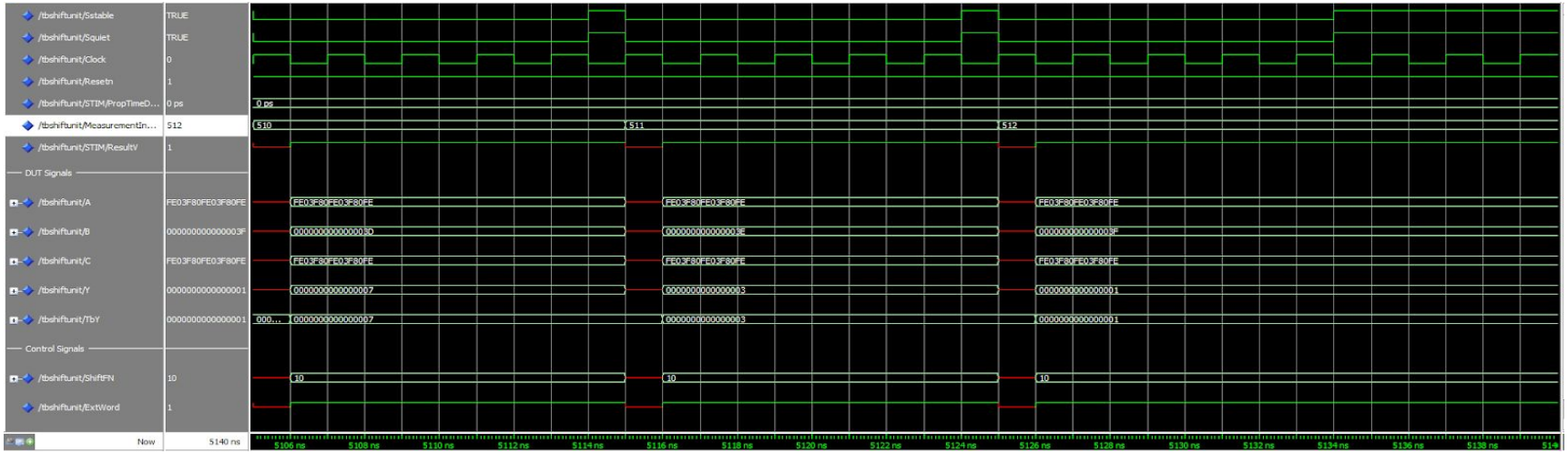
Functional simulation: Shift Unit SLL64 2

This screenshot shows the last three measurements of the SLL64 functional simulation. The output Y value matched with the TbY value, meaning that the operation is functioning properly. For example in measurement 512, the value of A is shifted left by 63 bits. The result Y is correct and matches with the TbY value.



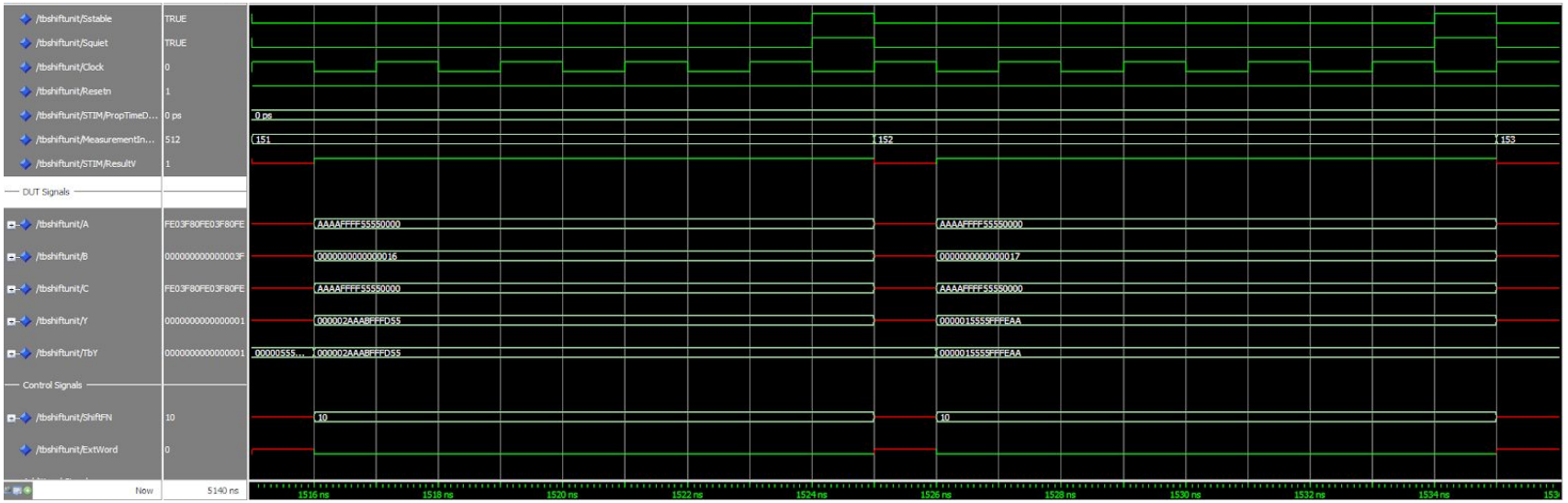
Functional simulation: Shift Unit SRL32 1

This screenshot shows the measurements of 310 and 311. The output Y value matched with the TbY value, meaning that the operation is functioning properly. For example in measurement 311, ExtWord is 1 and the value of A is shifted right by 22 bits, since shift right instruction is performed on a word. The lower 32 bits of A will be put into the upper word of 64-bits, and then shifted right by 22 bits. Output Y will contain the upper word part of that result as the lower word, and sign extended to 64 bits.



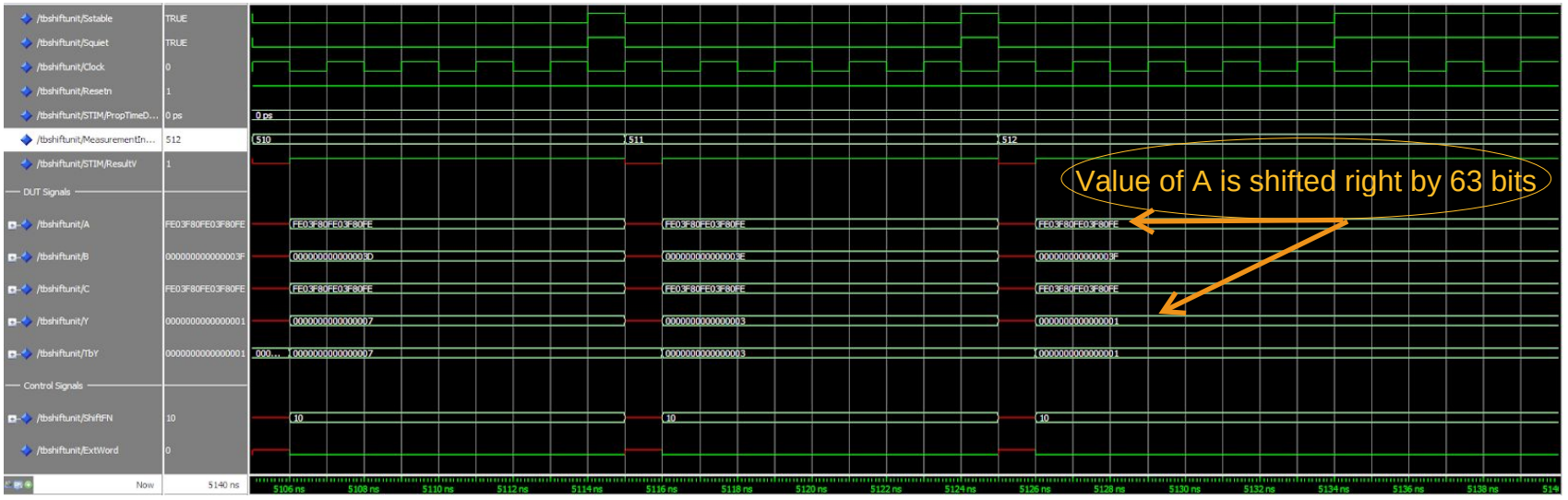
Functional simulation: Shift Unit SRL32 2

This screenshot shows the last three measurements. The output Y value matched with the TbY value, meaning that the operation is functioning properly. For example in measurement 512, ExtWord is 1 and the value of A will be shifted right by 31 bits, since shift right instruction is performed on a word. The lower 32 bits of A is put into the upper word of 64 bits, and then shifted right by 31 bits. Output Y will contain the upper word part of that result as the lower word, and sign extended to 64 bits.



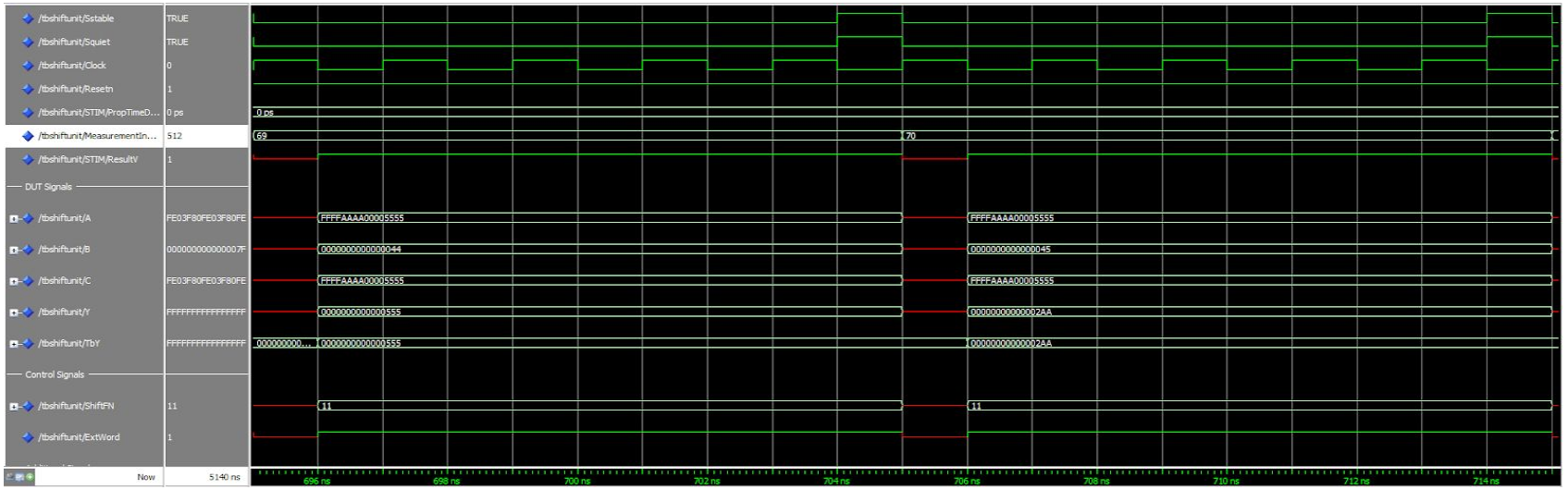
Functional simulation: Shift Unit SRL64 1

This screenshot shows the measurements of 151 and 152. The output Y value matched with the TbY value, meaning that the operation is functioning properly. For example in measurement 311, the value of A right will be shifted right by 22 bits. The result Y is correct and matches with the TbY value.



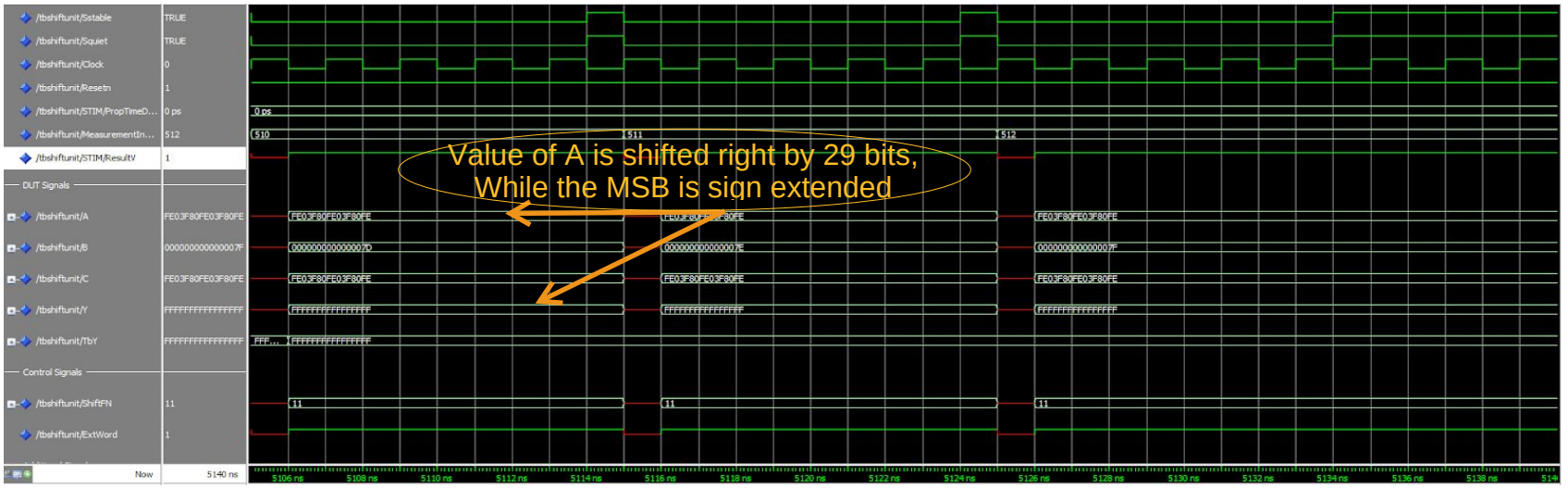
Functional simulation: Shift Unit SRL64 2

This screenshot shows the last three measurements. The output Y value matched with the TbY value, meaning that the operation is functioning properly. For example in measurement 512, the value of A is shifted right by 63 bits. The result of the shift should have the most significant bit in A in the least significant bit position of Y. The result Y is correct and matches with the TbY value.



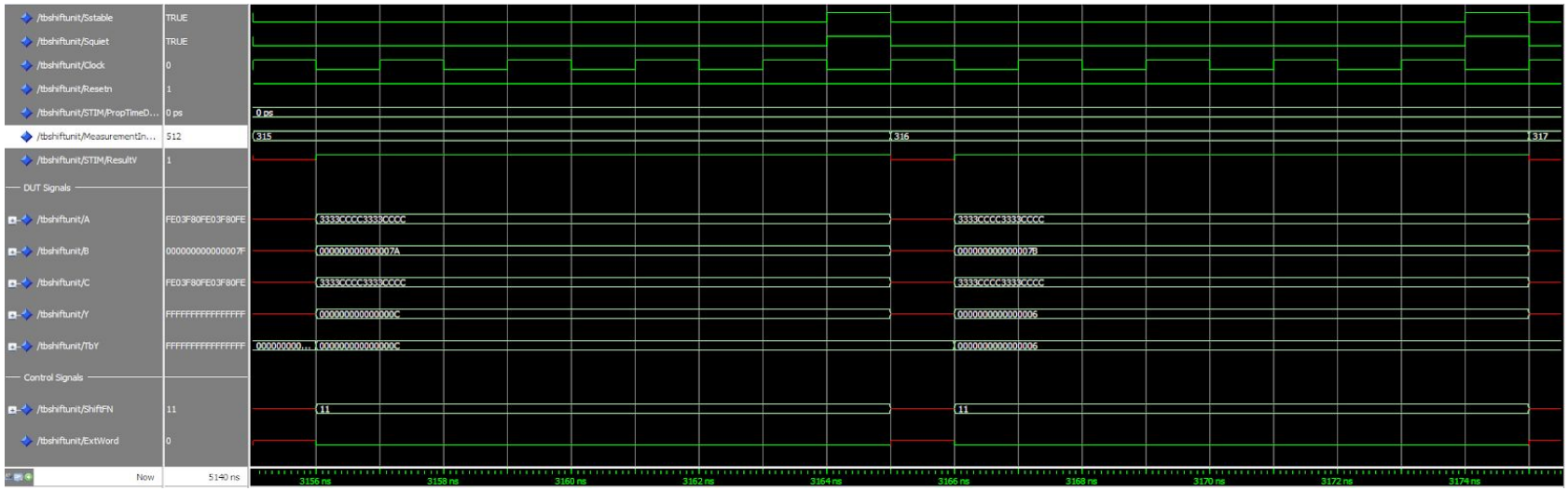
Functional simulation: Shift Unit SRA32 1

This screenshot shows the measurements of 69 and 70. The output Y value matched with the TbY value, meaning that the operation is functioning properly. For example in measurement 69, ExtWord is 1 and the instruction SRA by 4 bits, since only the least significant 5 bits of operand B is extracted as shift count. The lower 32 bits of A is put into the upper word of 64-bits, and then shifted right by 4 bits, while sign extending the most significant bit every shift. Output Y will contain the upper word part of that result as the lower word, and it will be sign extended to 64 bits.



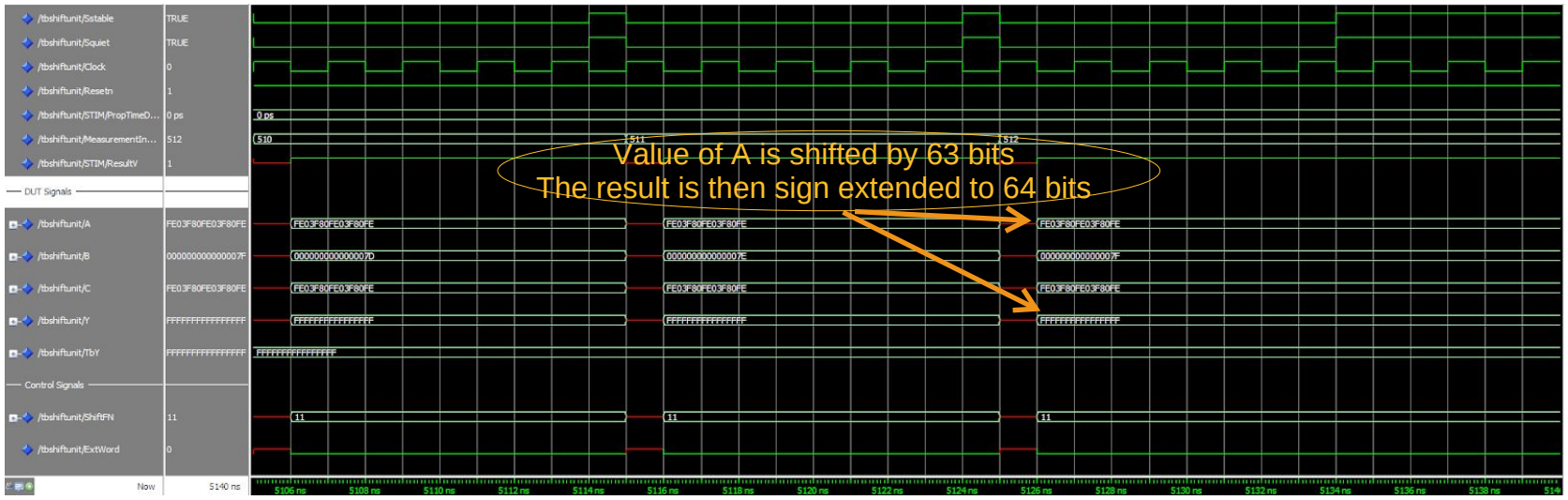
Functional simulation: Shift Unit SRA32 2

This screenshot shows the last three measurements. The output Y value matched with the TbY value, meaning that the operation is functioning properly. For example in measurement 510, ExtWord is 1 and the instruction SRA by 29 bits, since only the least significant 5 bits of operand B is extracted as shift count. The lower 32 bits of A, put this lower 32 bits into the upper word of 64-bits, and then shift the value by 29 bits to the right, while sign extending the most significant bit every shift. Output Y will contain the upper word part of that result as the lower word, and it will be sign extended to 64 bits.



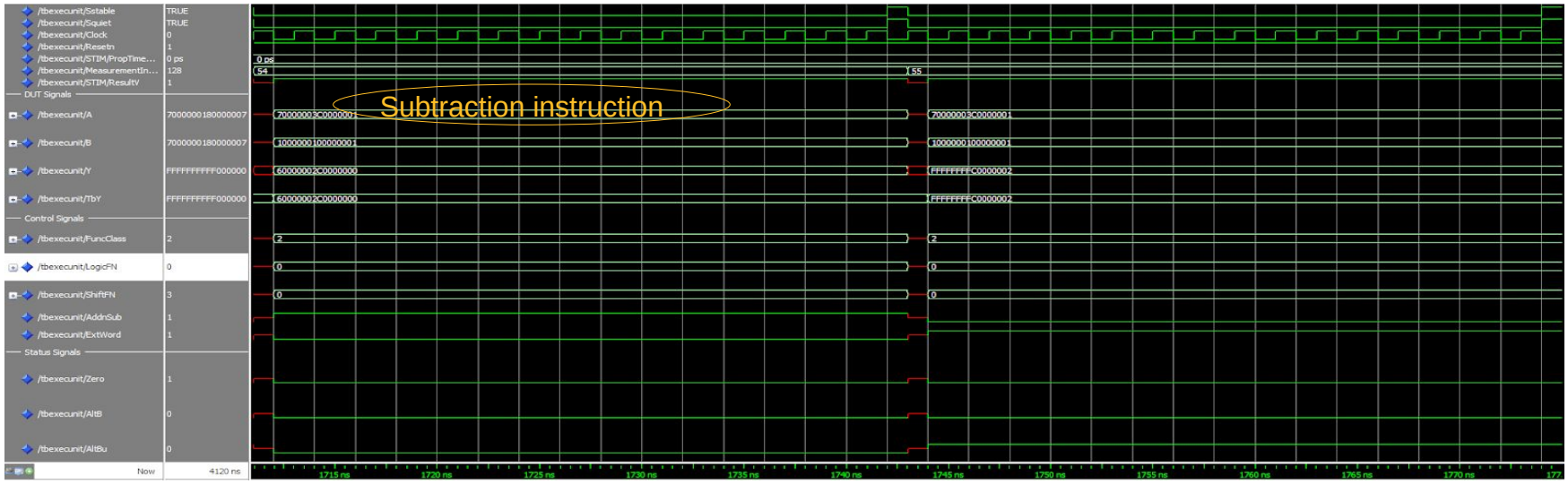
Functional simulation: Shift Unit SRA64 1

This screenshot shows the measurements of 315 and 316. The output Y value matched with the TbY value, meaning that the operation is functioning properly. For example in measurement 315, SRA is performed on the value of A by 26 bits. The result of the shift should be sign extended to 64 bits. The result Y is correct and matches with the TbY value.



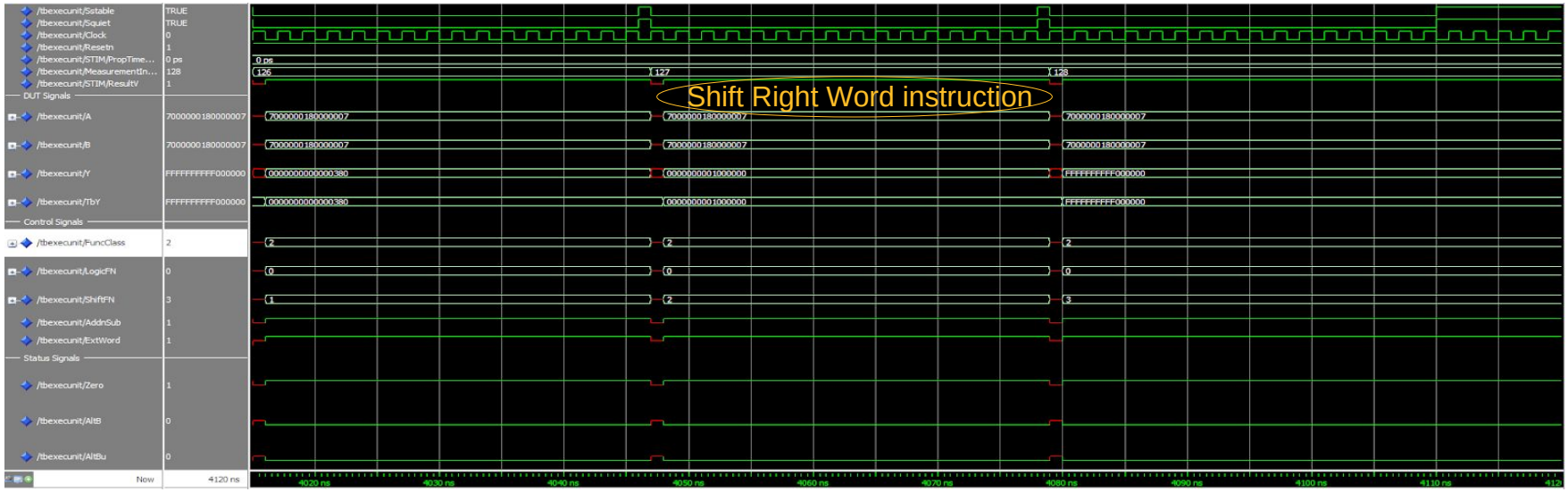
Functional simulation: Shift Unit SRA64 2

This screenshot shows the last three measurements. The output Y value matched with the TbY value, meaning that the operation is functioning properly. For example in measurement 512, SRA is performed on the value of A by 63 bits. The result of the shift should be sign extended to 64 bits. The result Y is correct and matches with the TbY value.



Functional simulation: The Execution Unit 1

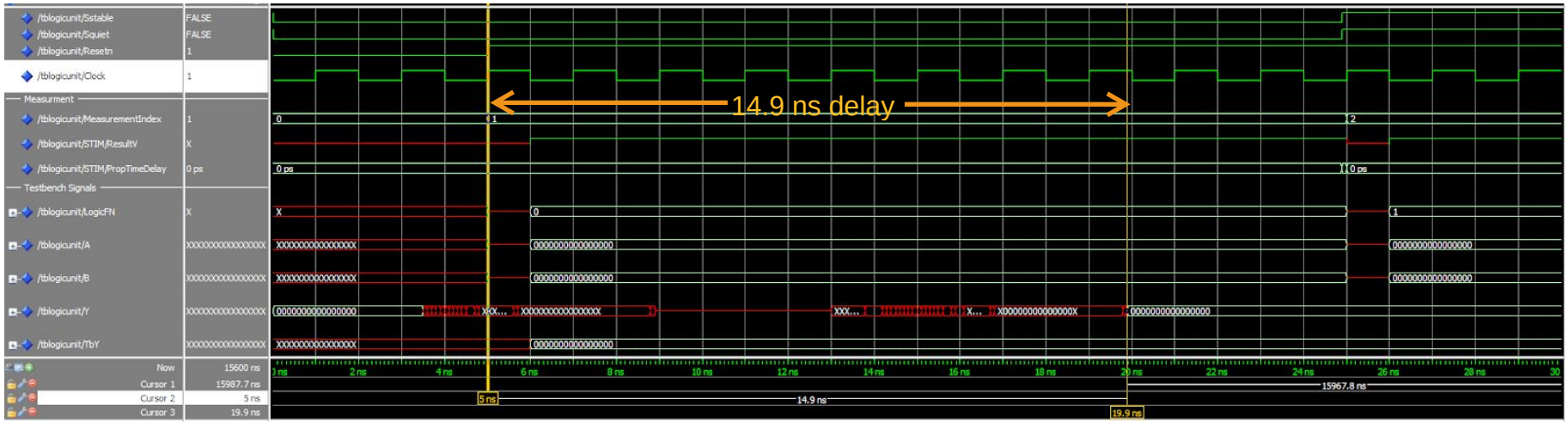
This screenshot shows the measurements of 54 and 55. The output Y value matches the TbY value and the status signals are correct, meaning that the execution unit is functioning properly. For example in measurement 54, FuncClass is 2, ShiftFn is 0, and AddnSub is 1, meaning that it is a 64-bit subtraction instruction. The output Y is the correct value, and other status signals such as Zero, AltB, and AltBu are accurate too since A is greater than B.



Functional simulation: The Execution Unit 2

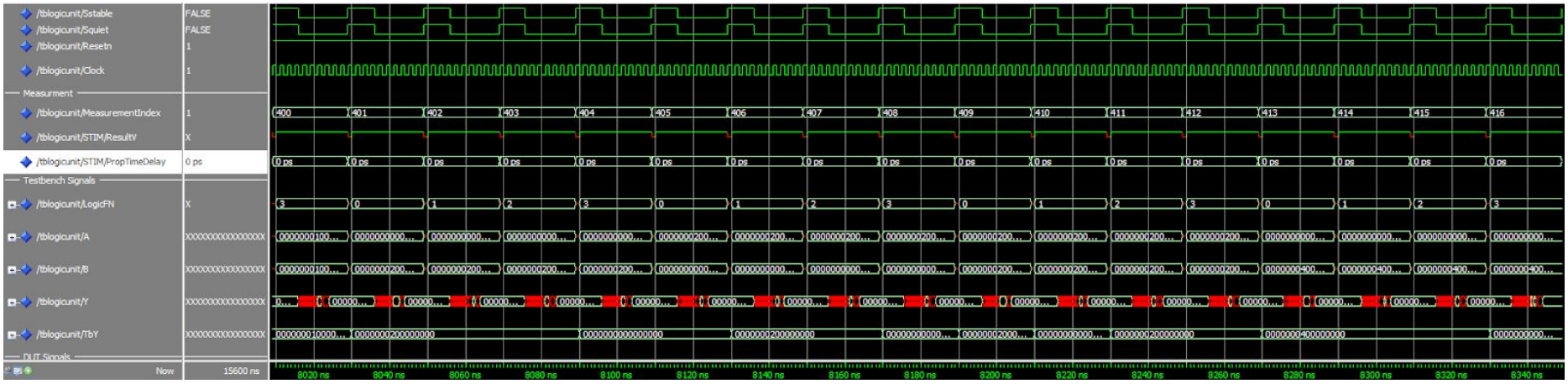
This screenshot shows the last three measurements. Output Y value matches the TbY value and the status signals are correct, meaning that the Execution Unit is functioning properly. For example in measurement 127, FuncClass is 2, ShiftFn is 2, and ExtWord is 1, meaning that it is a shift right word instruction. The lower word of A will be swapped into the upper word of A and shifted by 7 bits. Upper word of the result will then be put into the lower word of output Y and then sign extended to 64 bits.

Timing Simulation Waves



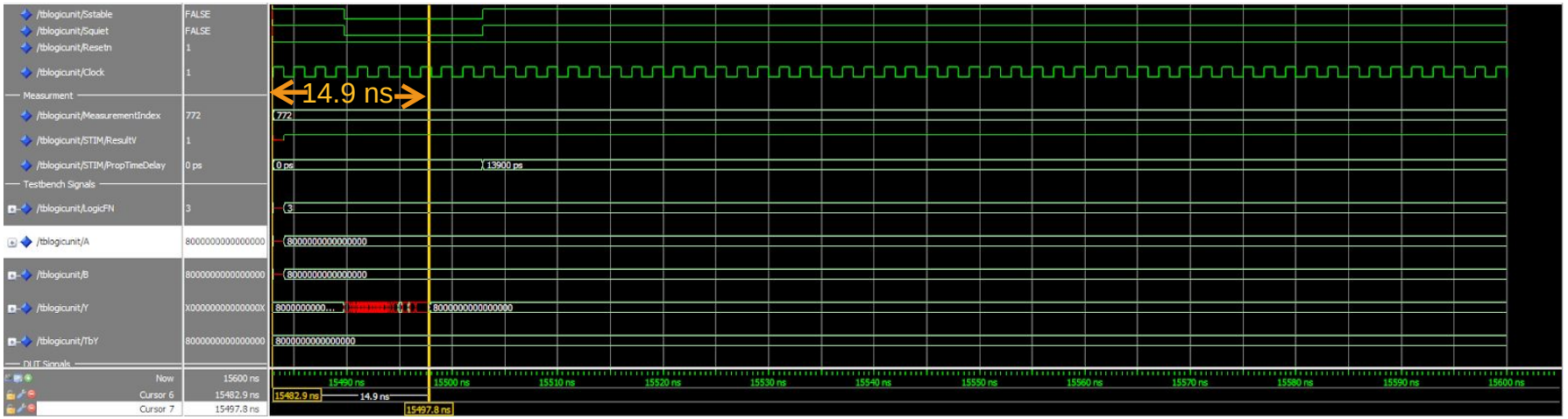
Timing simulation: Logic Unit 1

This screenshot shows the close up view of the waves of the timing simulation of measurements #0,#1,#2 for the logic unit. For measurement #1,the propagation delay is measured using the cursors. As seen at the bottom, measurement #1 starts at 5 ns and the output value appears at 19.9 ns. Which means the propagation delay is 19.9ns - 5ns = 14.9 ns



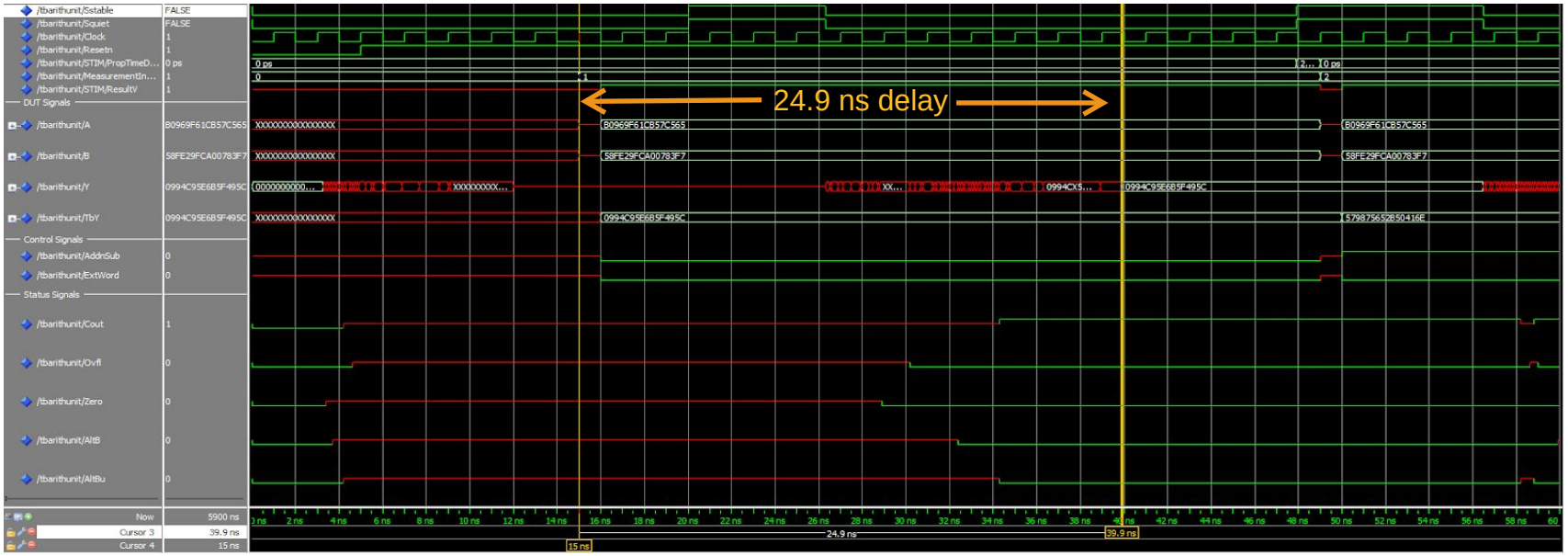
Timing simulation: Logic Unit 2

This diagram shows the close up view of the waves of the timing simulation from measurement #400 to end of #416 for the logic unit. These intermediate values show the propagation delays in between each measurement output. The red sections in between each outputs are the propagation delays.



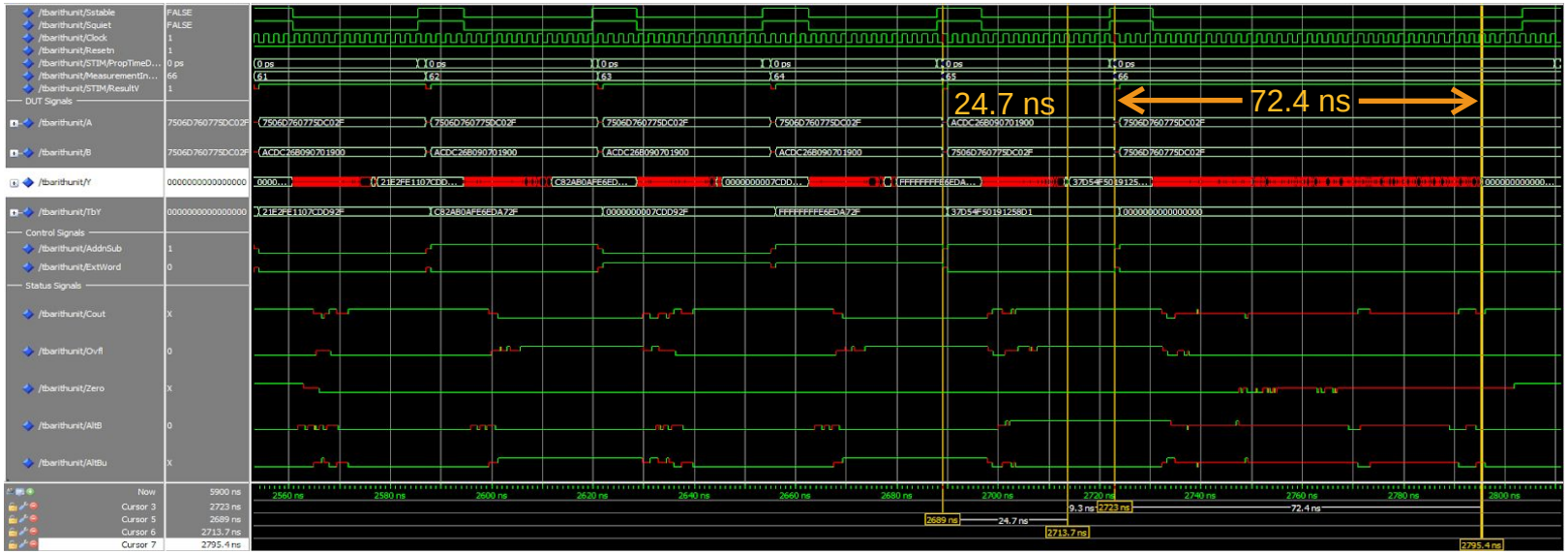
Timing simulation: Logic Unit 3

This diagram shows the close up view of the waves of the timing simulation for the last measurement of the logic unit. For our last measurement (measurement #772), the propagation delay is measured using the two cursors at the bottom. As seen at the bottom of the diagram, the measurement starts at 15482.9 ns and the output value Y appears at 15497.8 ns. Which means that the propagation delay is, 15497.8 ns - 15482.9 ns = 14.9 ns.



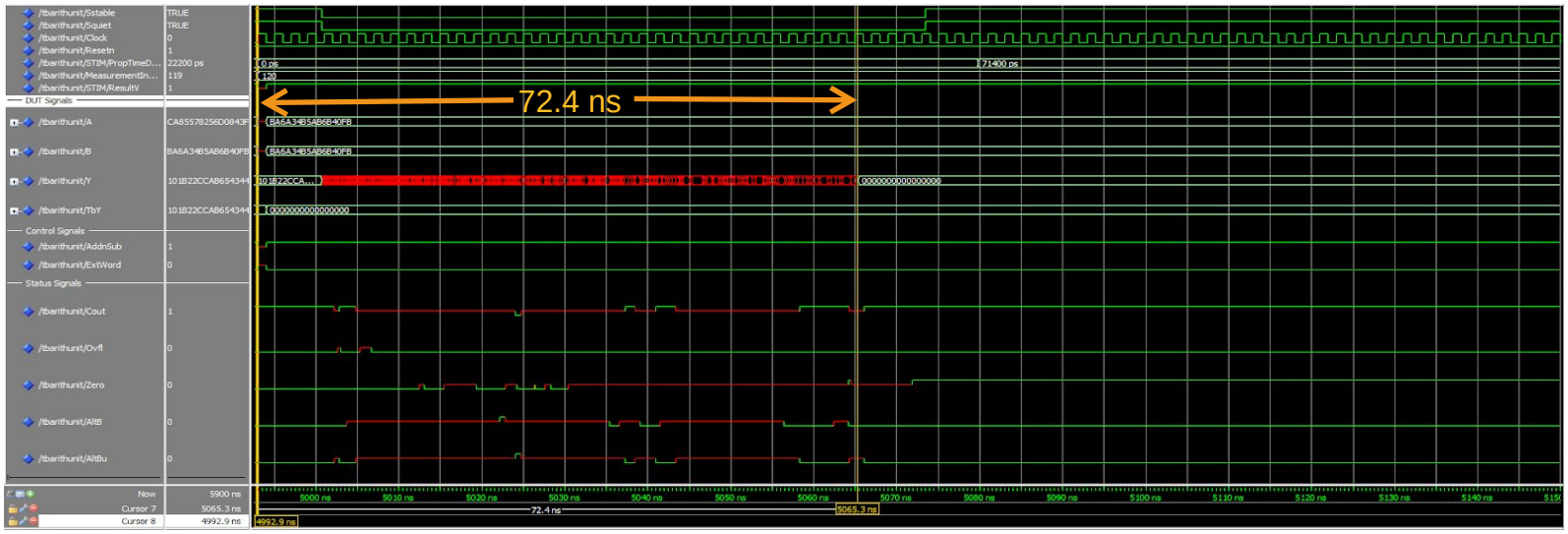
Timing simulation: Arithmetic Unit 1

This screenshot shows the close up view of the waves of the timing simulation of measurements #0,#1,#2 for the arithmetic unit. For the first measurement, the propagation delay is measured using the cursors. As seen at the bottom, measurement #1 starts at 15 ns and the output value appears at 39.9 ns. Which means the propagation delay is 39.9ns - 15ns = 24.9 ns



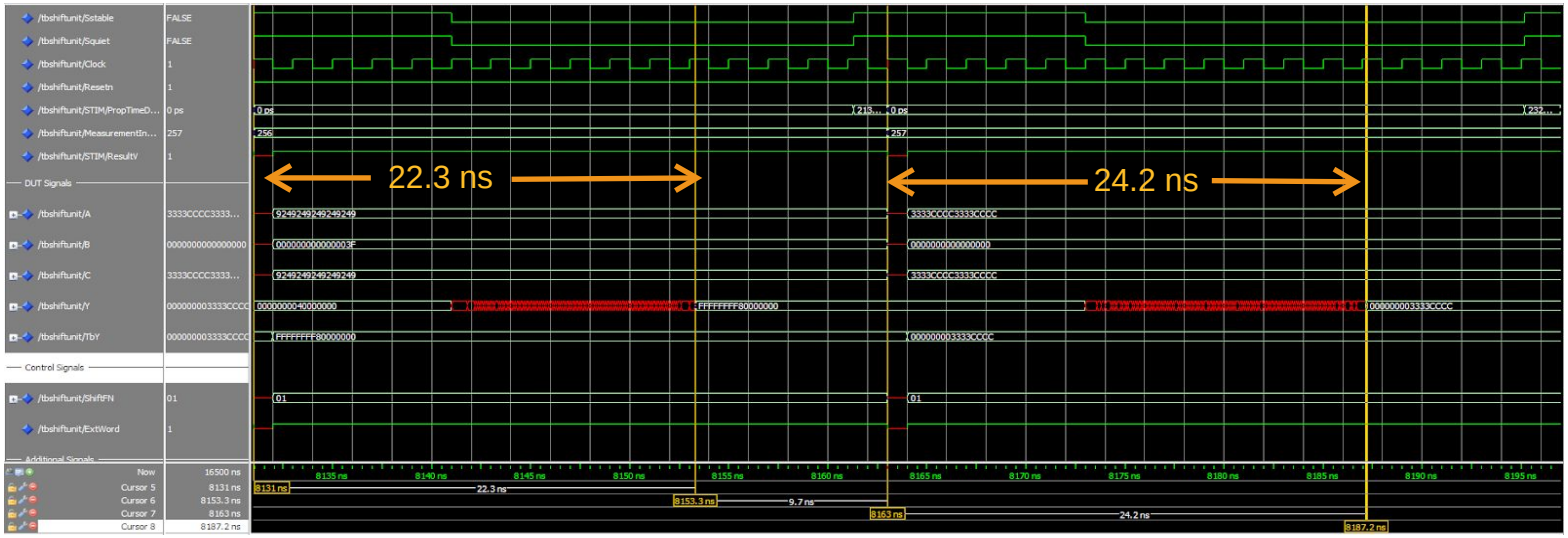
Timing simulation: Arithmetic Unit 2

The screenshot above shows the timing simulation of measurements #61 up to #66. The cursors above show the propagation delay of measurements #65 and measurements #66. The propagation delay of measurement #65 is shown to be 24.7 ns and the propagation delay for measurement #66 is 72.4 ns. In measurement #66, we observe that the delay is significantly longer whenever the operation is a subtraction of a number with itself.



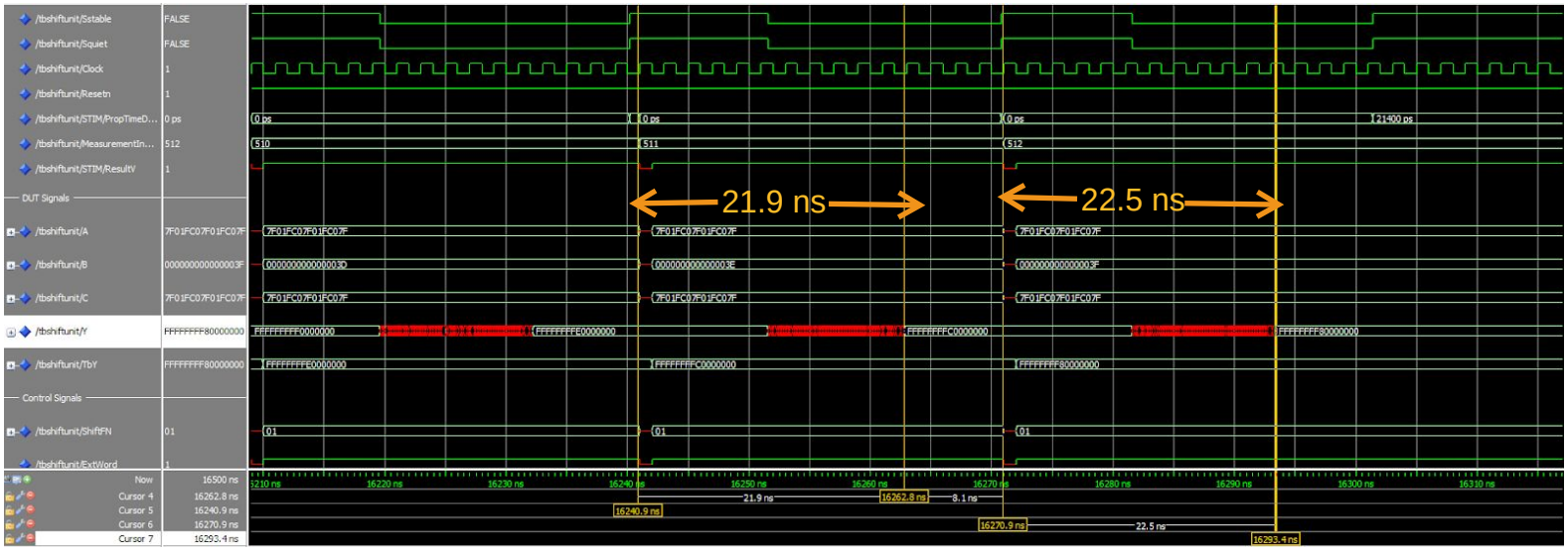
Timing simulation: Arithmetic Unit 3

The diagram shows the measurement of the propagation time delay of last measurements in the arithmetic unit. The cursors at the bottom show that the measurement starts at 4992.9 ns and the output value appears at 5065.3 ns. The propagation delay is 72.4 ns. This is also a similar case to the measurement #66 where the subtracting number with itself outputs a value of zero.



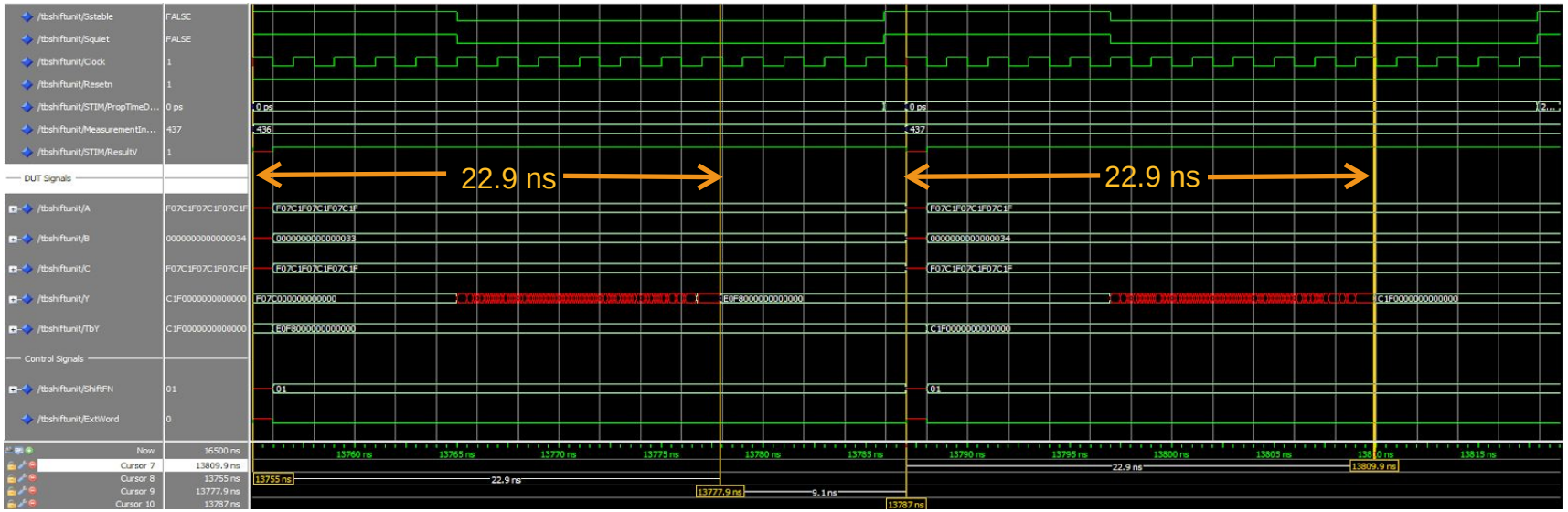
Timing simulation: Shift Unit SLL32 1

The screenshot above shows two intermediate measurements for the SLL32 test. The cursors at the bottom show the measurement delay for each measurement. Measurement #256 has a delay of 8153.3 ns - 8131 ns = 22.3 ns and measurement #257 has a propagation delay of 8187.2 ns - 8163 ns = 24.2 ns.



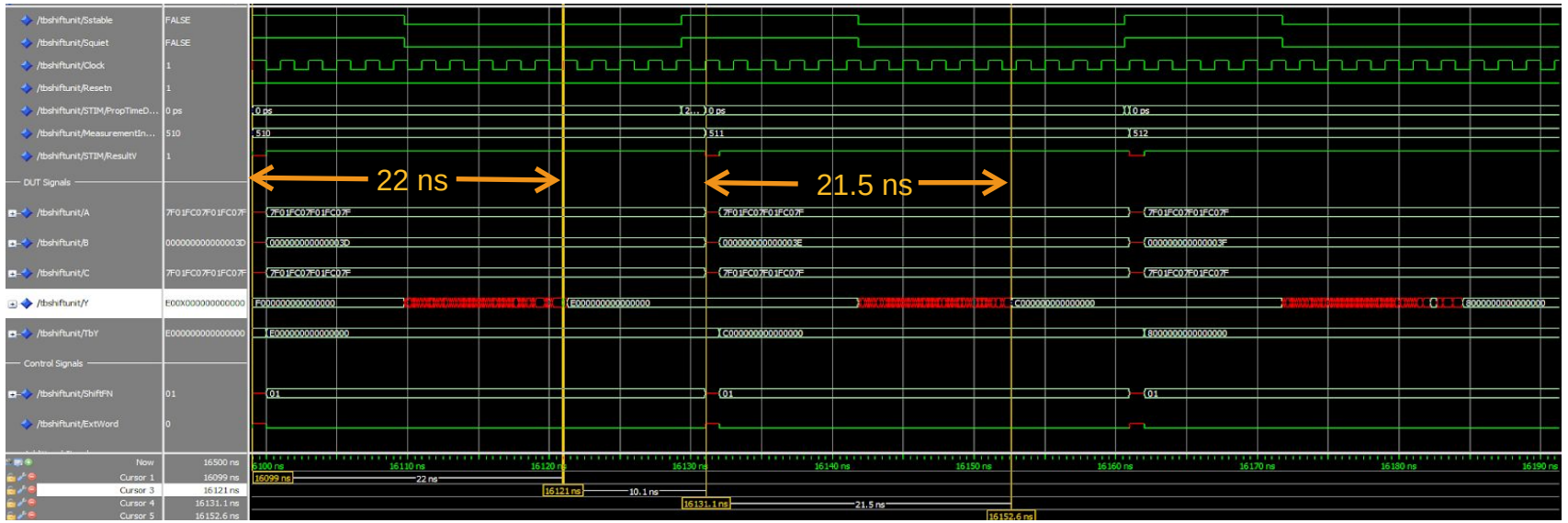
Timing simulation: Shift Unit SLL32 2

The screenshot above shows the last three measurements for the SLL32 test. The cursors shown above measures the propagation delay for measurements #511 and #512.The propagation delay shows to be 21.6 ns and 22.3 ns respectively.



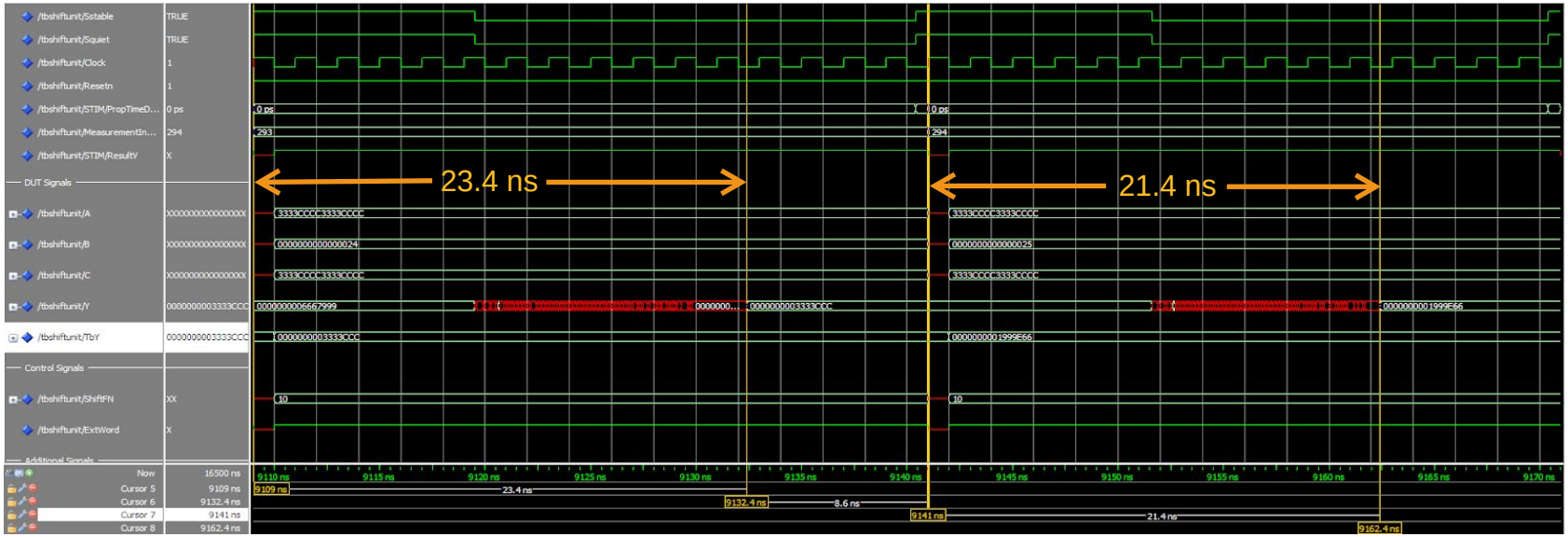
Timing simulation: Shift Unit SLL64 1

The screenshot above shows two intermediate measurements for the SLL64 test. The cursors at the bottom show the measurement delay for each measurement. Measurement #436 has a delay of 13777.9 ns - 13755 ns = 22.9 ns and measurement #437 has a propagation delay of 13809.9 ns - 13787 ns = 22.9 ns.



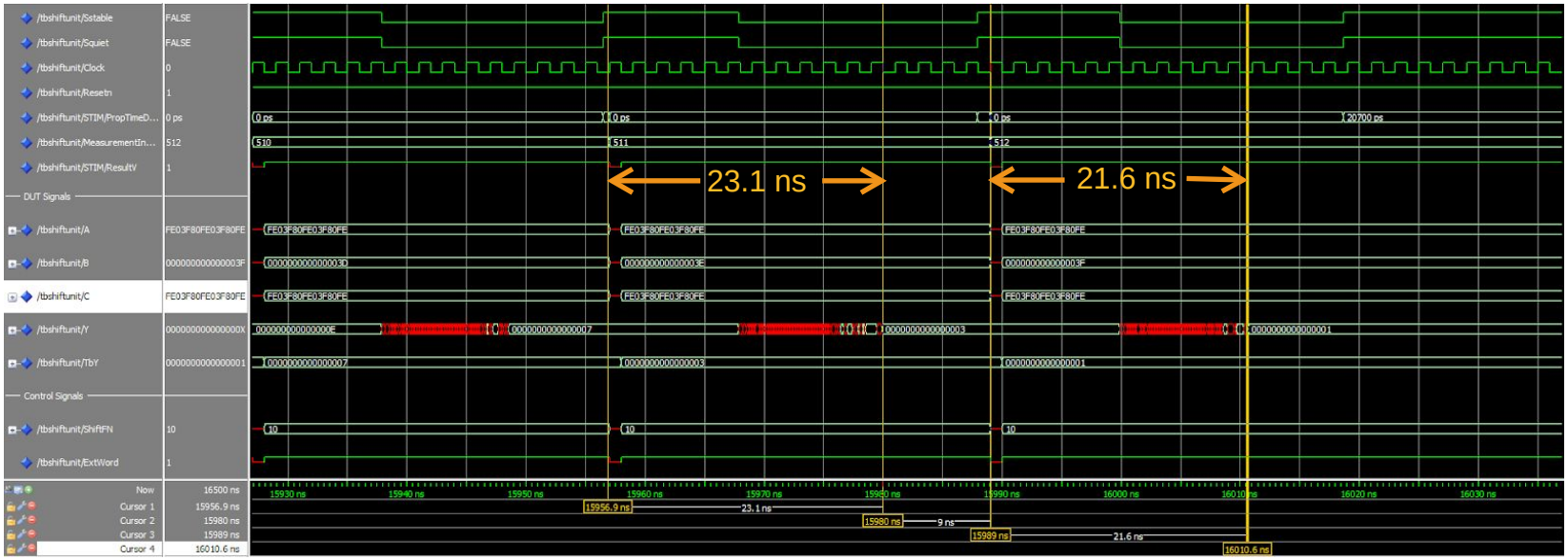
Timing simulation: Shift Unit SLL64 2

The screenshot above shows the last three measurements for the SLL64 test. The cursors at the bottom show the measurement delay for each measurement. Measurement #510 has a delay of 16121 ns - 16099 ns = 22 ns and measurement #511 has a propagation delay of 16152.6 ns - 16131.1 ns = 21.5 ns.



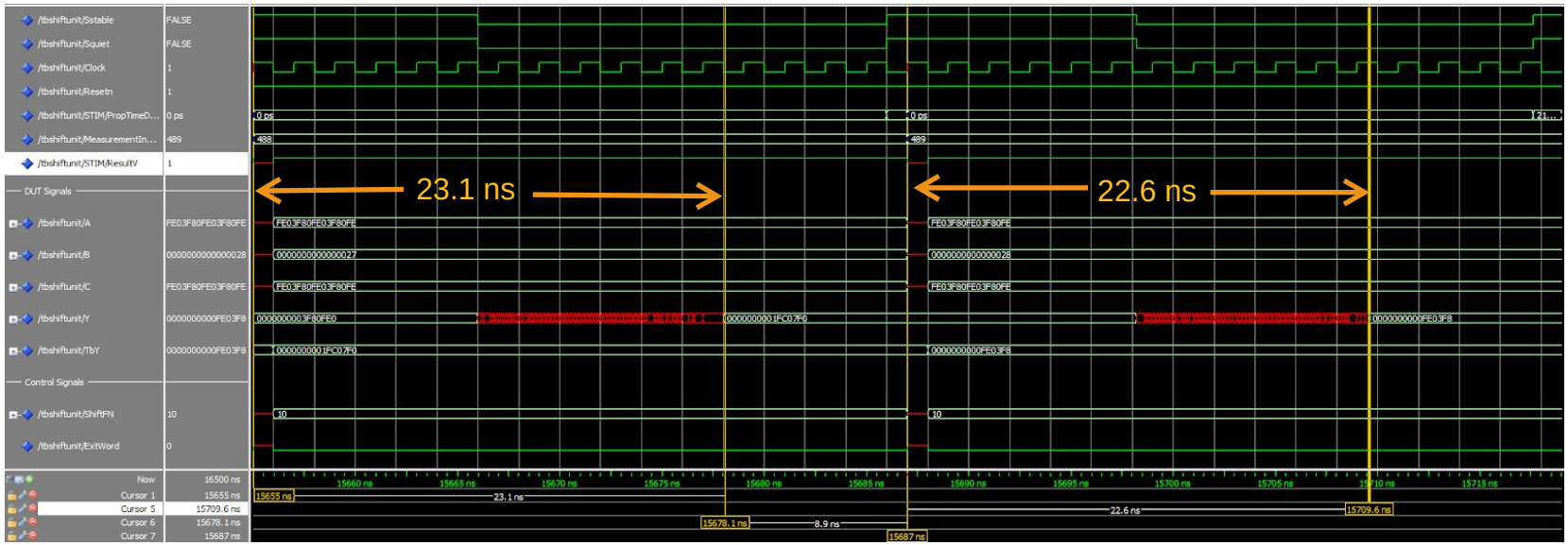
Timing simulation: Shift Unit SRL32 1

The screenshot above shows two intermediate measurements for the SRL32 test. The cursors at the bottom show the measurement delay for each measurement. Measurement #293 has a delay of 9132.4 ns - 9109 ns = 23.4 ns and measurement #294 has a propagation delay of 9162.4 ns -9141 ns = 21.4 ns.



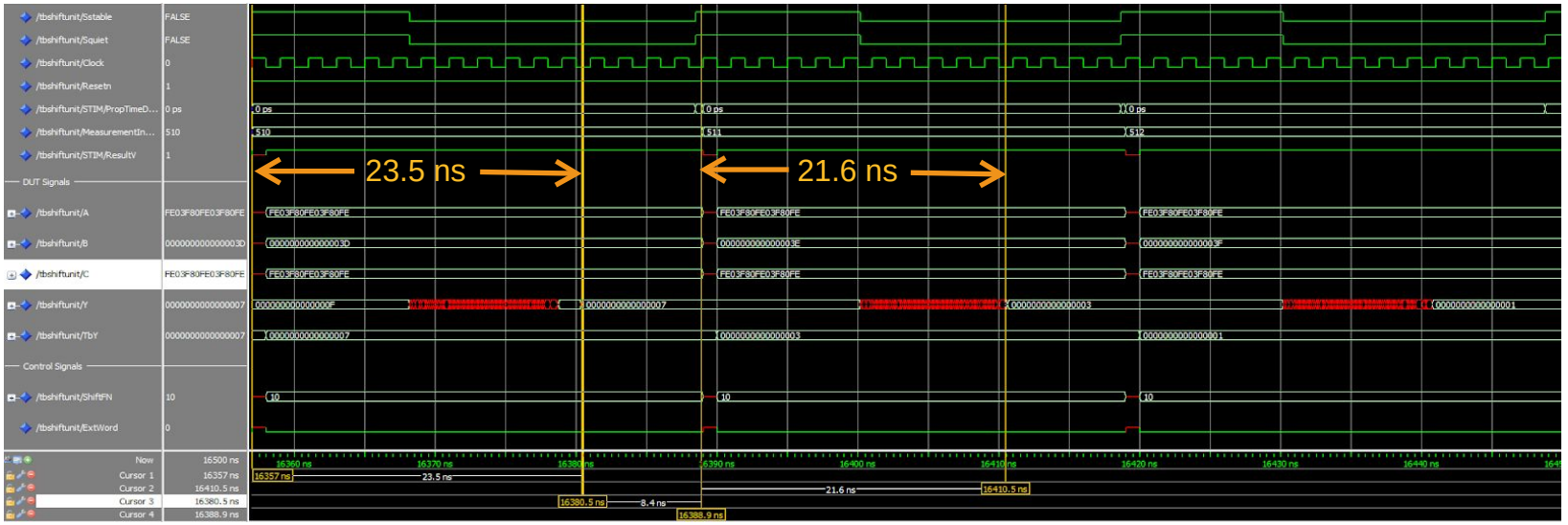
Timing simulation: Shift Unit SRL32 2

The screenshot above shows the last three measurements for the SRL32 test. The cursors at the bottom show the measurement delay for each measurement. Measurement #510 has a delay of 15948.6 ns - 15927 ns = 21.6 ns and measurement #511 has a propagation delay of 15979.9 ns - 15956.9 ns = 23 ns.



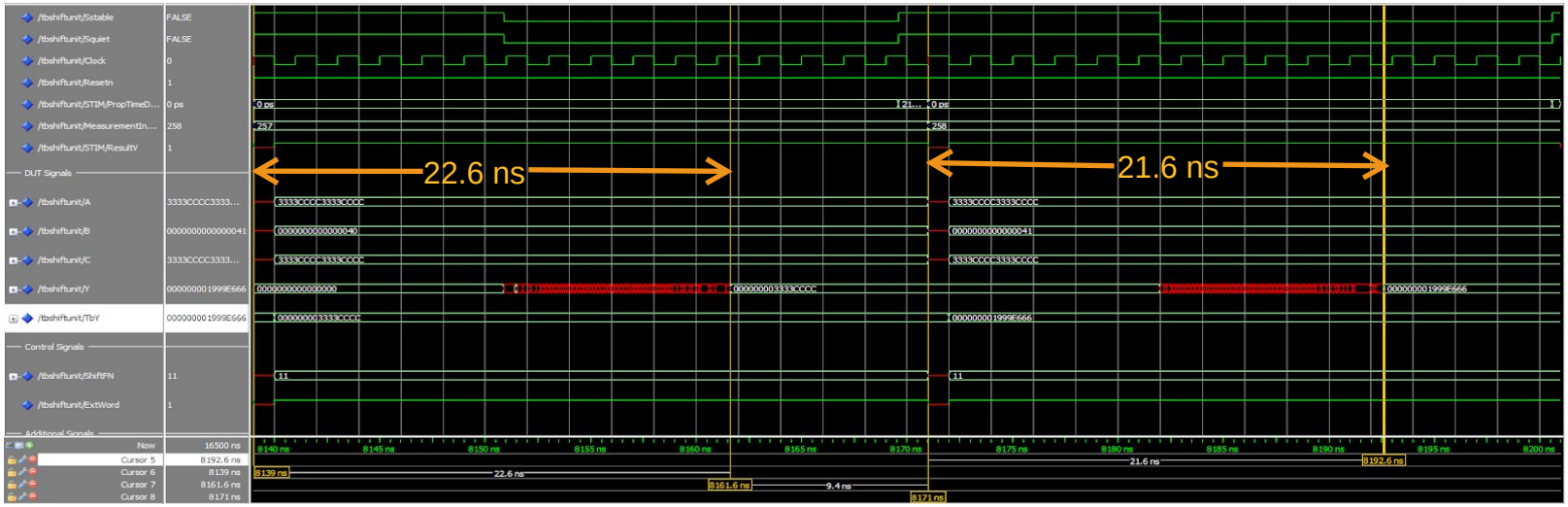
Timing simulation: Shift Unit SRL64 1

The screenshot above shows two intermediate measurements for the SRL64 test. The cursors at the bottom show the measurement delay for each measurement. Measurement #488 has a delay of 15678.1 ns - 15655 ns = 23.1 ns and measurement #489 has a propagation delay of 15709.6 ns - 15687 ns = 22.6 ns.



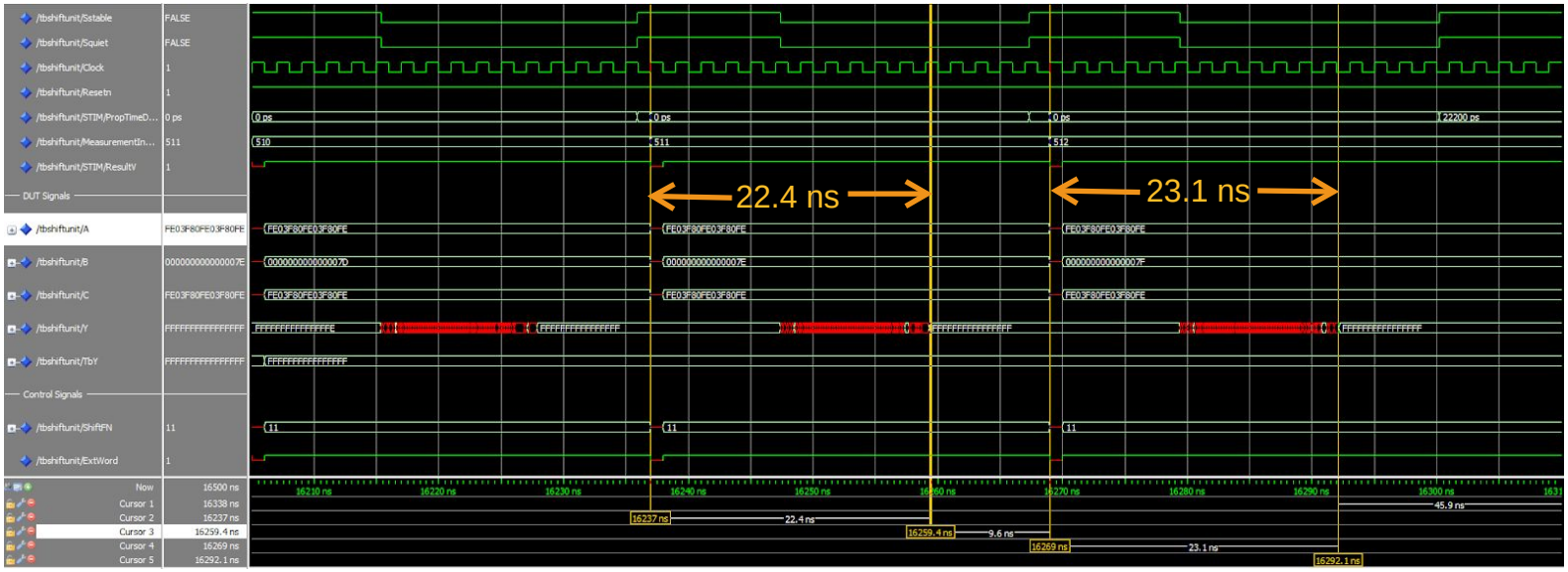
Timing simulation: Shift Unit SRL64 2

The screenshot above shows the last three measurements for the SRL64 test. The cursors at the bottom show the measurement delay for each measurement. Measurement #510 has a delay of 16380.5 ns - 16357 ns = 23.5 ns and measurement #511 has a propagation delay of 16410.5 ns - 16388.9 ns = 21.6 ns.



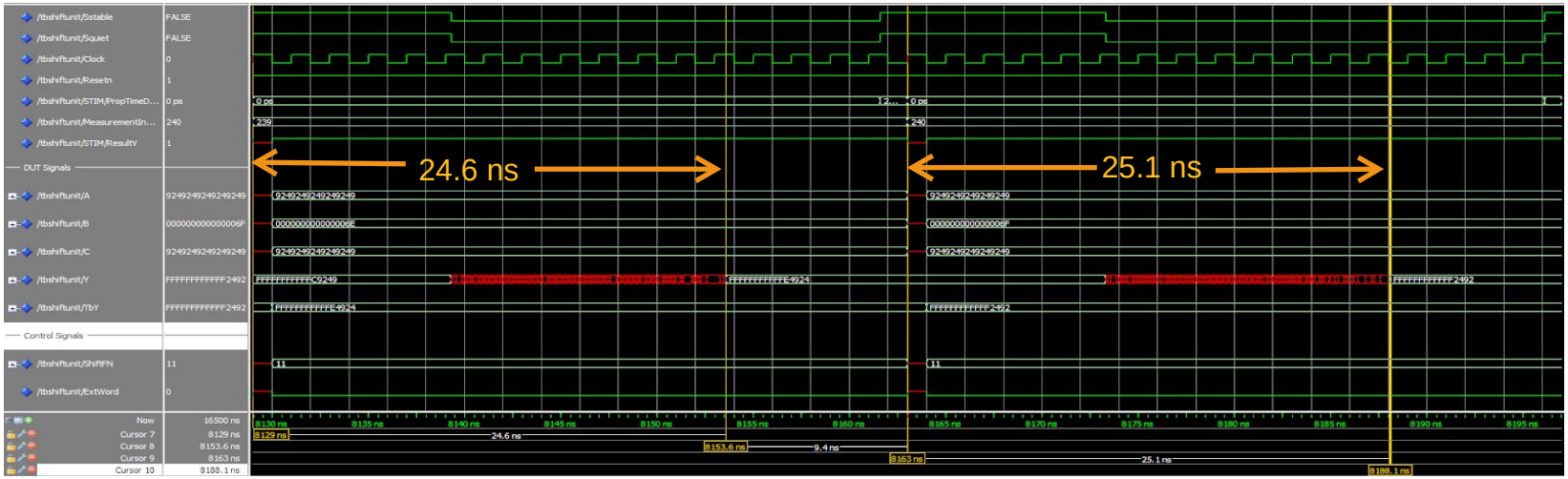
Timing simulation: Shift Unit SRA32 1

The screenshot above shows two intermediate measurements for the SRA32 test. The cursors at the bottom show the measurement delay for each measurement. Measurement #257 has a delay of 8161.6 ns - 8139 ns = 22.6 ns and measurement #258 has a propagation delay of 8192.6 ns - 8171 ns = 21.6 ns.



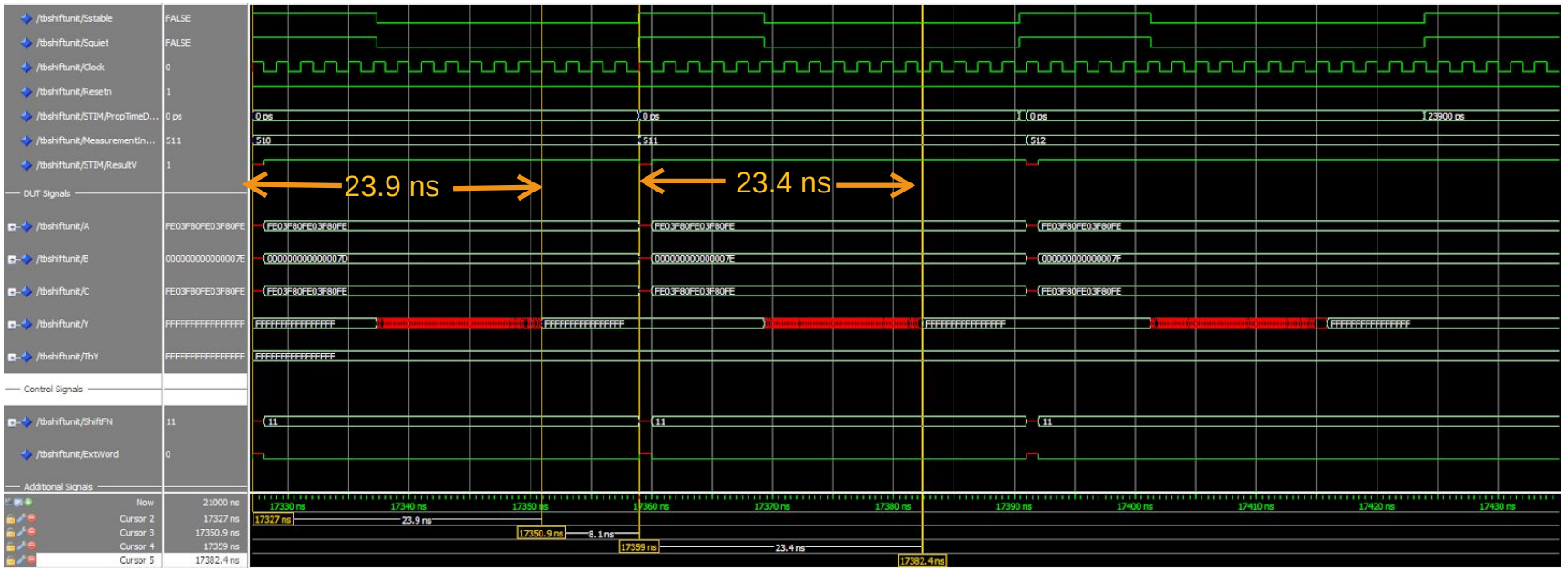
Timing simulation: Shift Unit SRA32 2

The screenshot above shows the last three measurements for the SRA32 test. The cursors at the bottom show the measurement delay for each measurement. Measurement #510 has a delay of 16227.9 ns - 16205 ns = 22.9 ns and measurement #511 has a propagation delay of 16259.3 ns - 16237 ns = 22.3 ns.



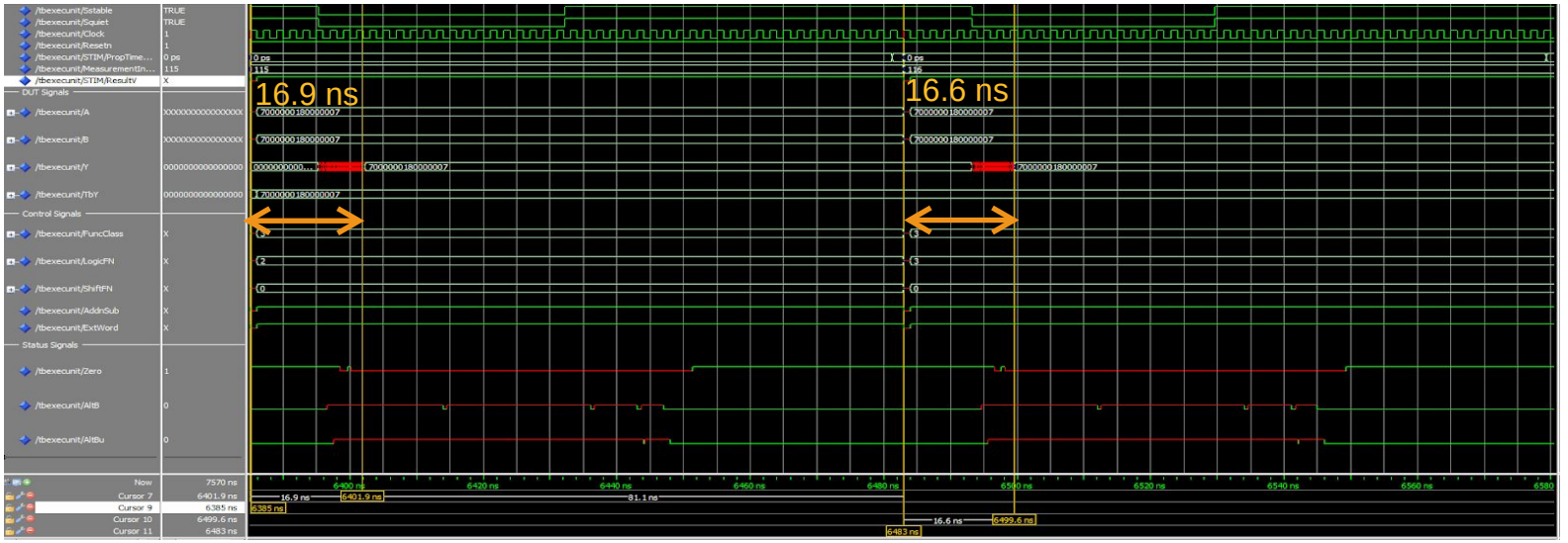
Timing simulation: Shift Unit SRA64 1

The screenshot above shows the last three measurements for the SRA64 test. The cursors at the bottom show the measurement delay for each measurement. Measurement #239 has a delay of 8153.6 ns - 8129 ns = 24.6 ns and measurement #240 has a propagation delay of 8188.1 ns - 8163 ns = 25.1 ns.



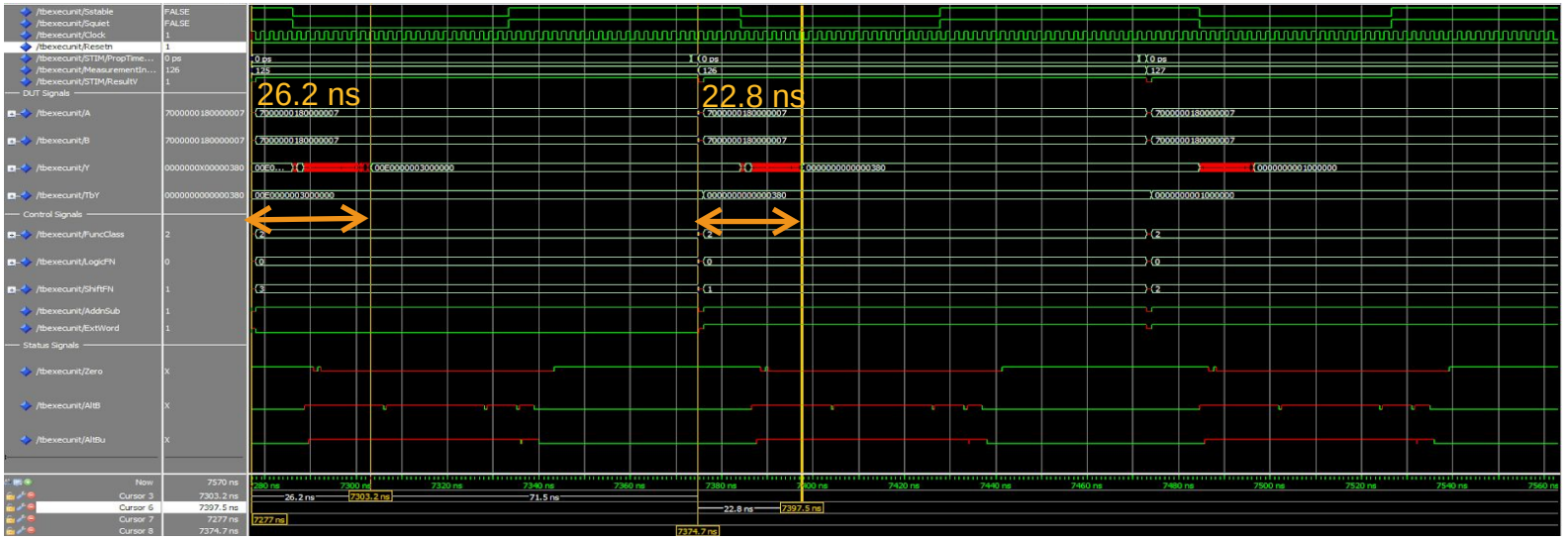
Timing simulation: Shift Unit SRA64 2

The screenshot above shows the last three measurements for the SRL64 test. The cursors at the bottom show the measurement delay for each measurement. Measurement #510 has a delay of 17350.9 ns - 17327 ns = 23.9 ns and measurement #511 has a propagation delay of 17382.4 ns - 17359 ns = 23.4 ns.



Timing Simulation: Execution Unit 1

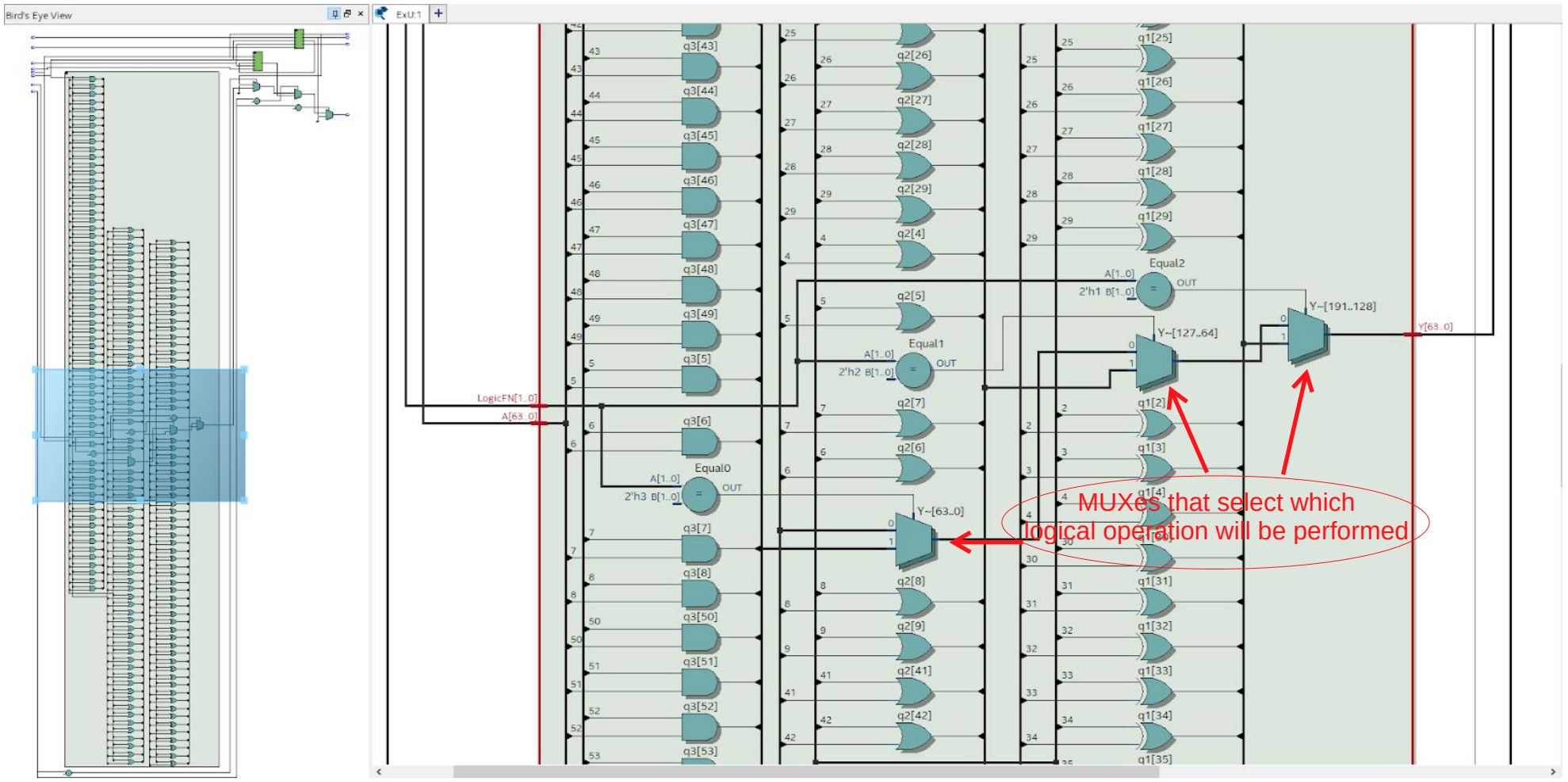
The screenshot above shows two intermediate values in the execution unit. Measurements #115 has a propagation delay of 16.9 ns and #116 has a propagation delay of 16.6 ns as shown by the cursors above. The two measurements above show FuncClass = 3, LogicFN= 2 and 3 . Which means this is a logic operation of OR and AND.



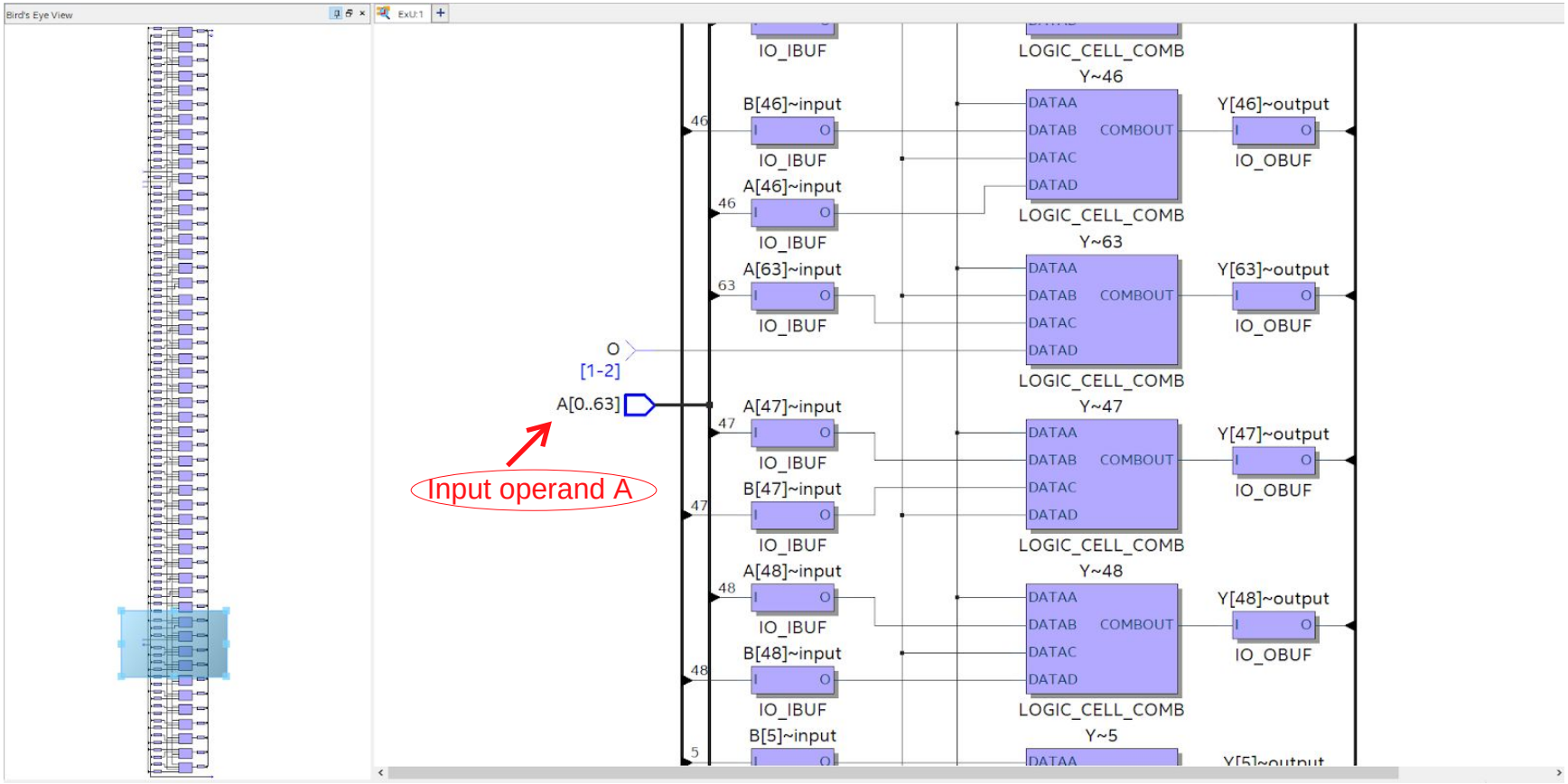
Timing Simulation: Execution Unit 2

The screenshot shown above shows the last three measurements in the execution unit. The cursors show the propagation delay of measurements #125 and #126. The propagation delays are 26.2 ns and 22.8 ns respectively. The signals are FuncClass = 2, (which means that it's a shift/arith operation) ShiftFN = 3 and 1. Therefore measurements #125 and #126 are SRA and SLL operations.

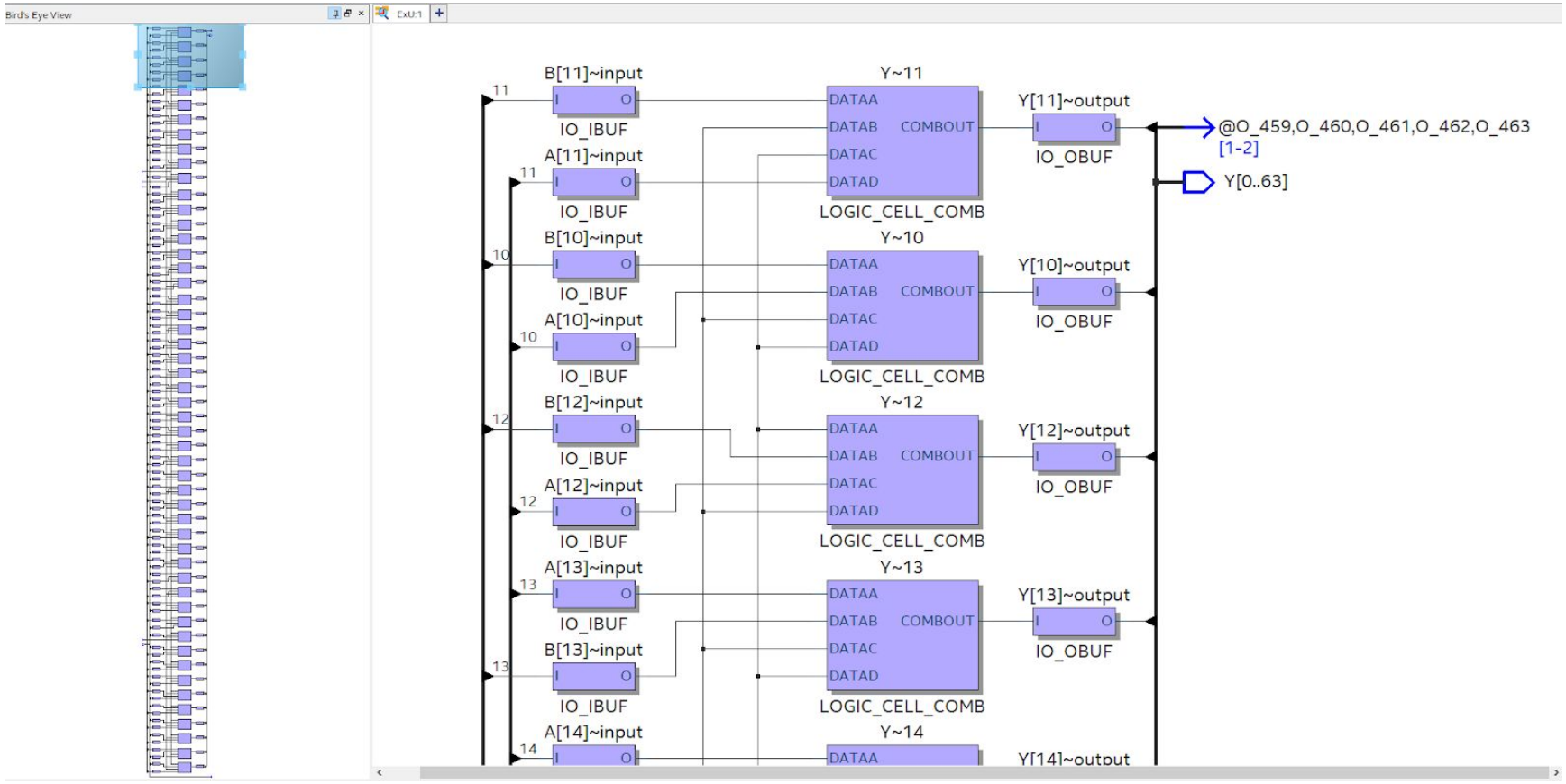
Circuit Screenshots



Logic Unit RTL:
The zoomed in picture shows the input LogicFN signal with 64-bit input operand A, into the overall general structure of the circuit. The result is shown to pass on to a MUX into output Y at the end.

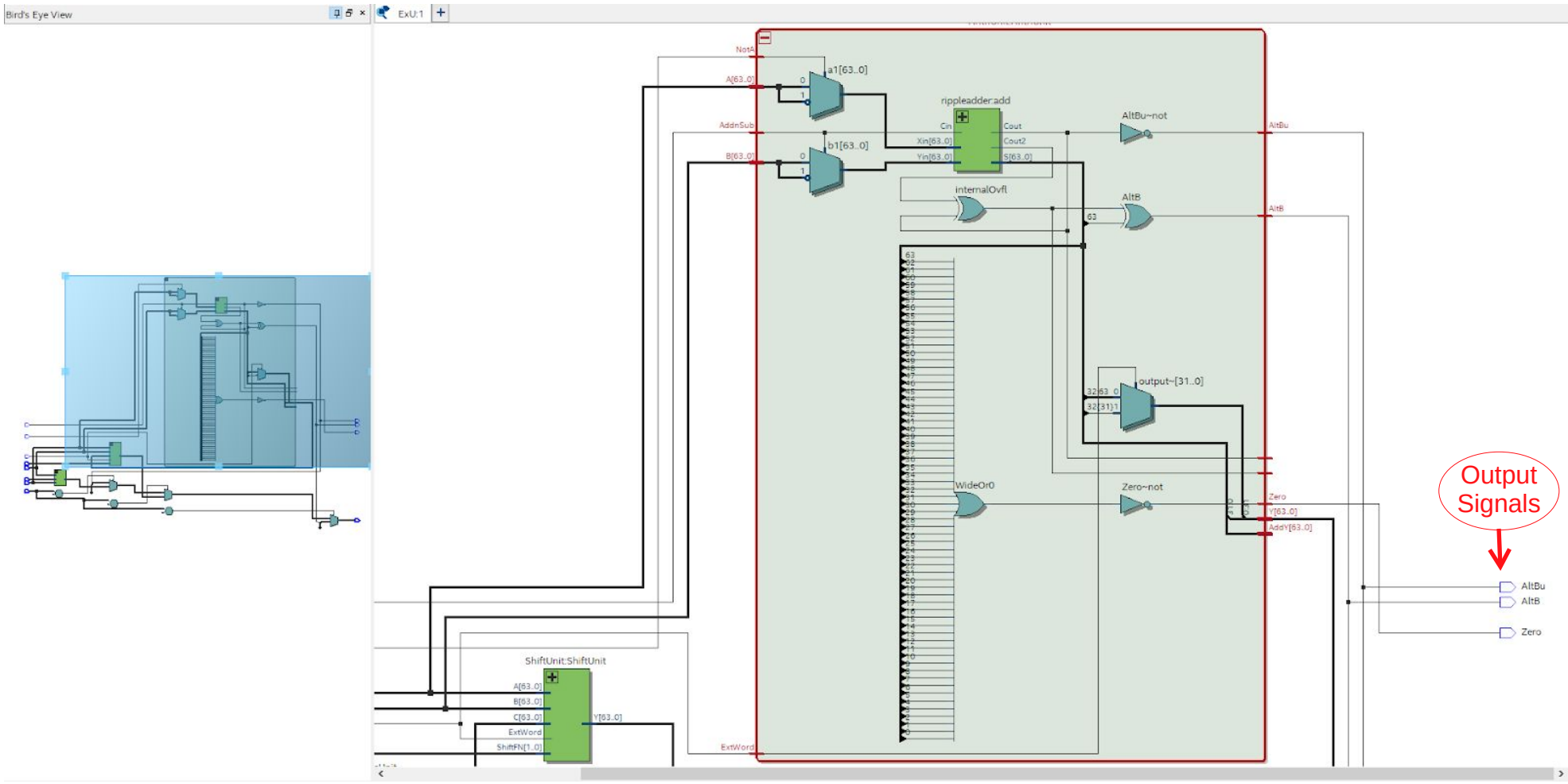


Logic Unit Post Fit:
Shows the 64-bit input of operand A into the logic unit and control signal input.



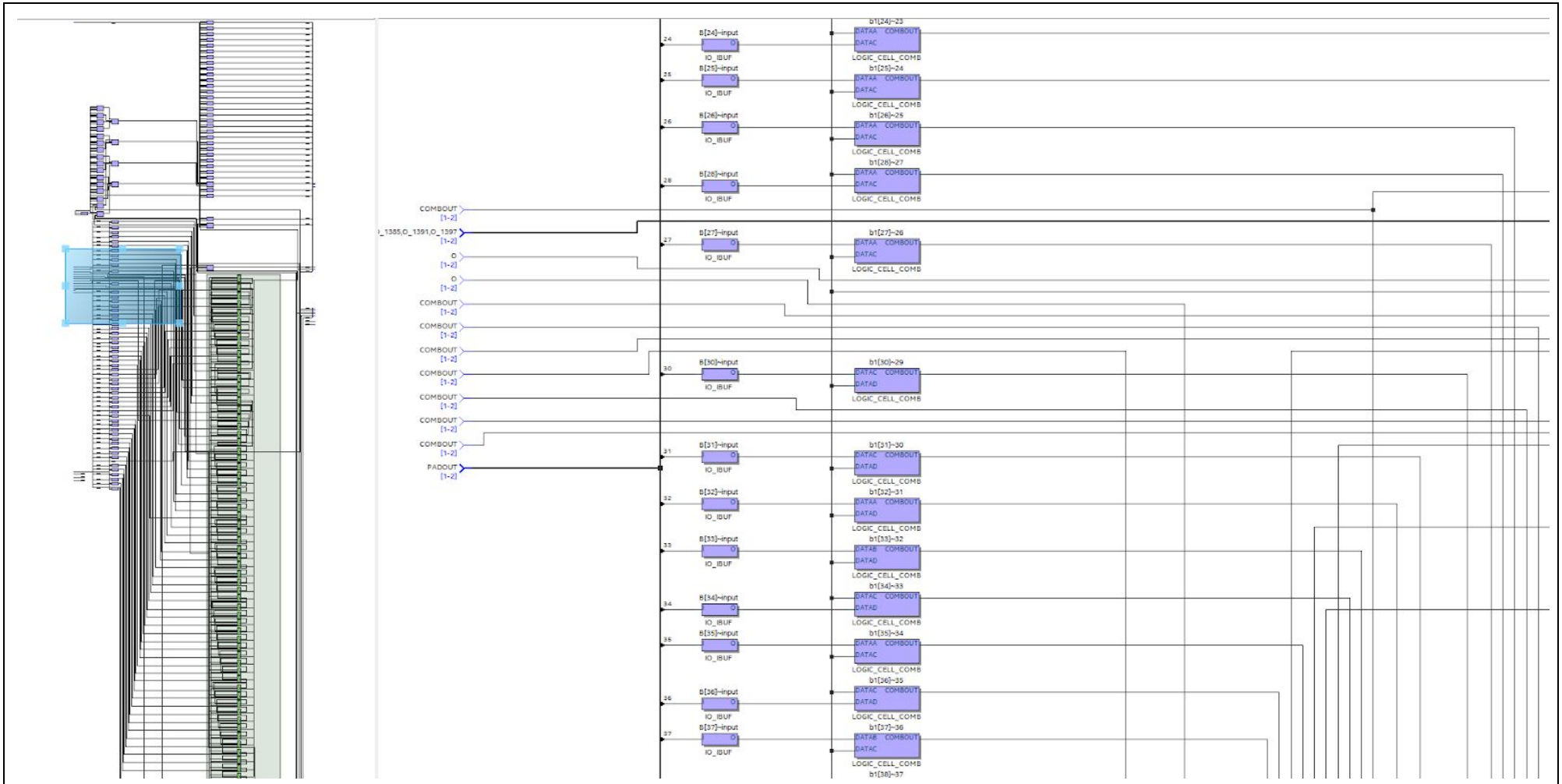
Logic Unit Post Fit:

The diagram above shows the general structure of the circuit and the resulting output Y at the top right.



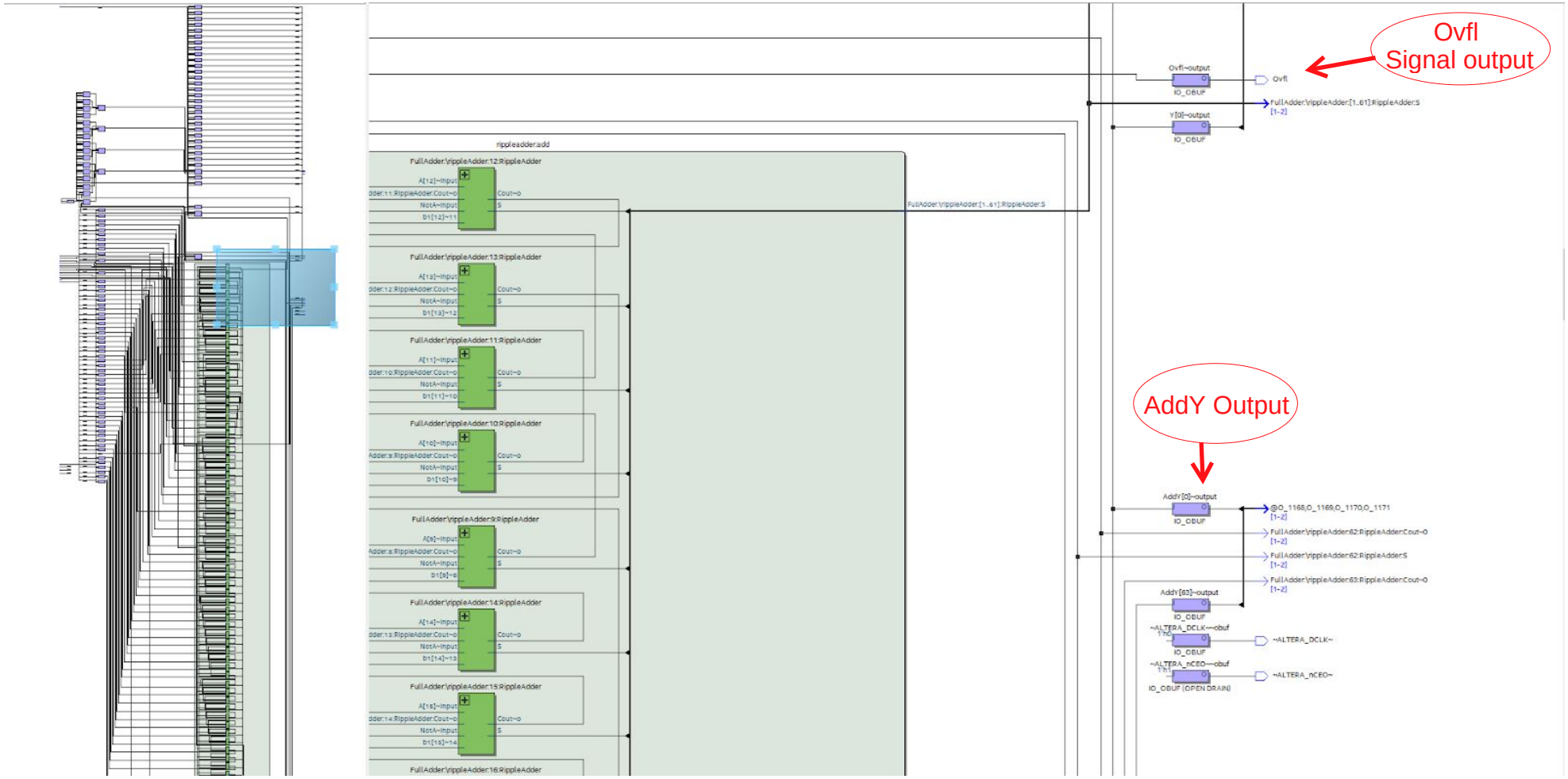
Arithmetic Unit RTL viewer:

The above screenshot shows the overall structure of the arithmetic unit. The two signals NotA and Addn'Sub enters with the two input operands A and B. NotA and Addn'Sub controls the two MUXes at the beginning to determine if the operands need to be changed for a subtraction.The results then enter the ripple adder shown above. The output of the ripple adder will then determine all the output signals: AltBu, AltB, Cout and Ovfl. The ExtWord signal determines whether the result needs to be sign extended or not. Performing an OR operation and into a NOT operation will determine the Zero signal shown.



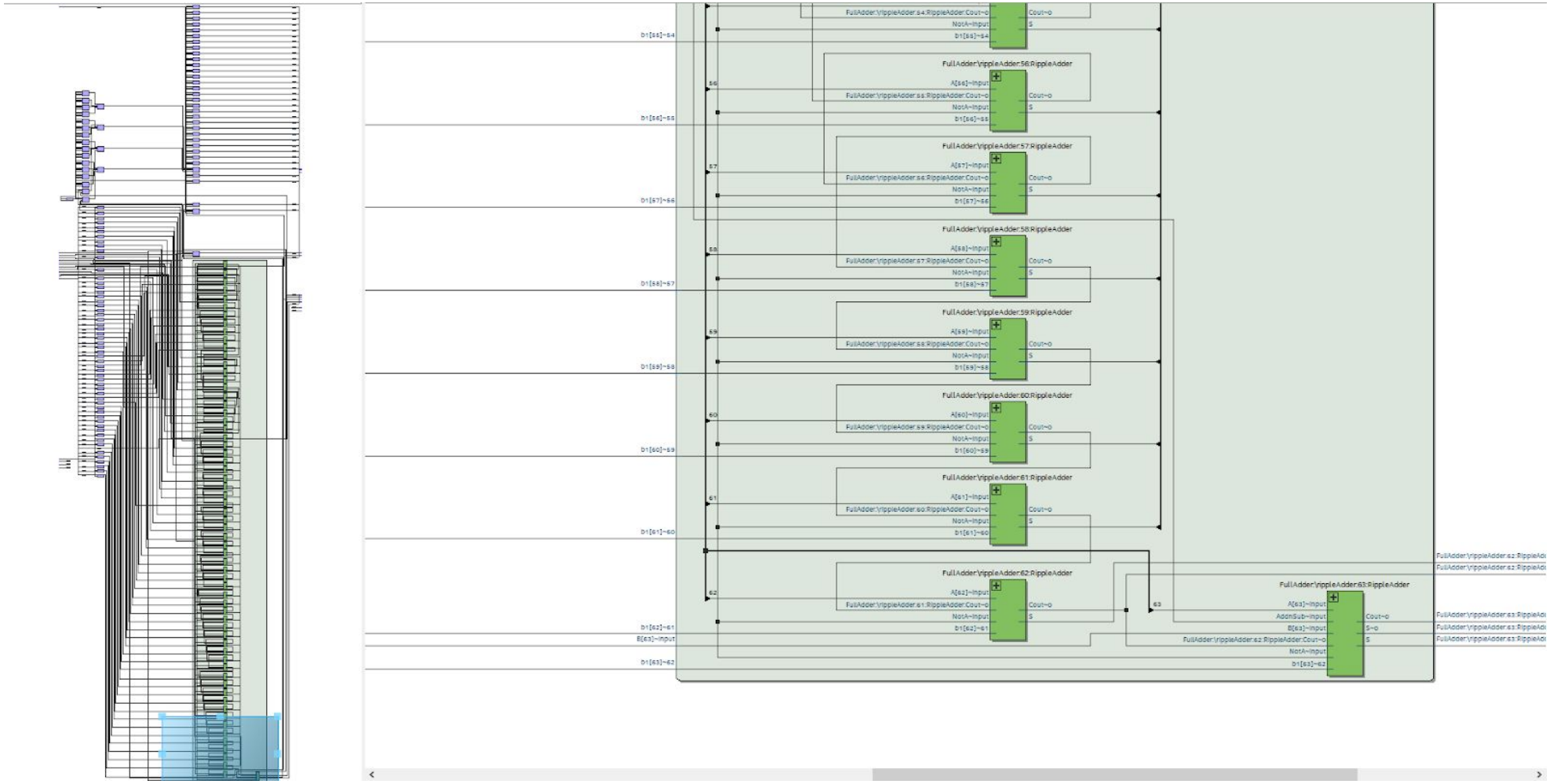
Arithmetic Unit Post Fit:

The screenshot shown above shows the bird's eye view of the post fit circuit on the left side and a zoomed in view of the input of the circuit on the right side.



Arithmetic Unit Post Fit:

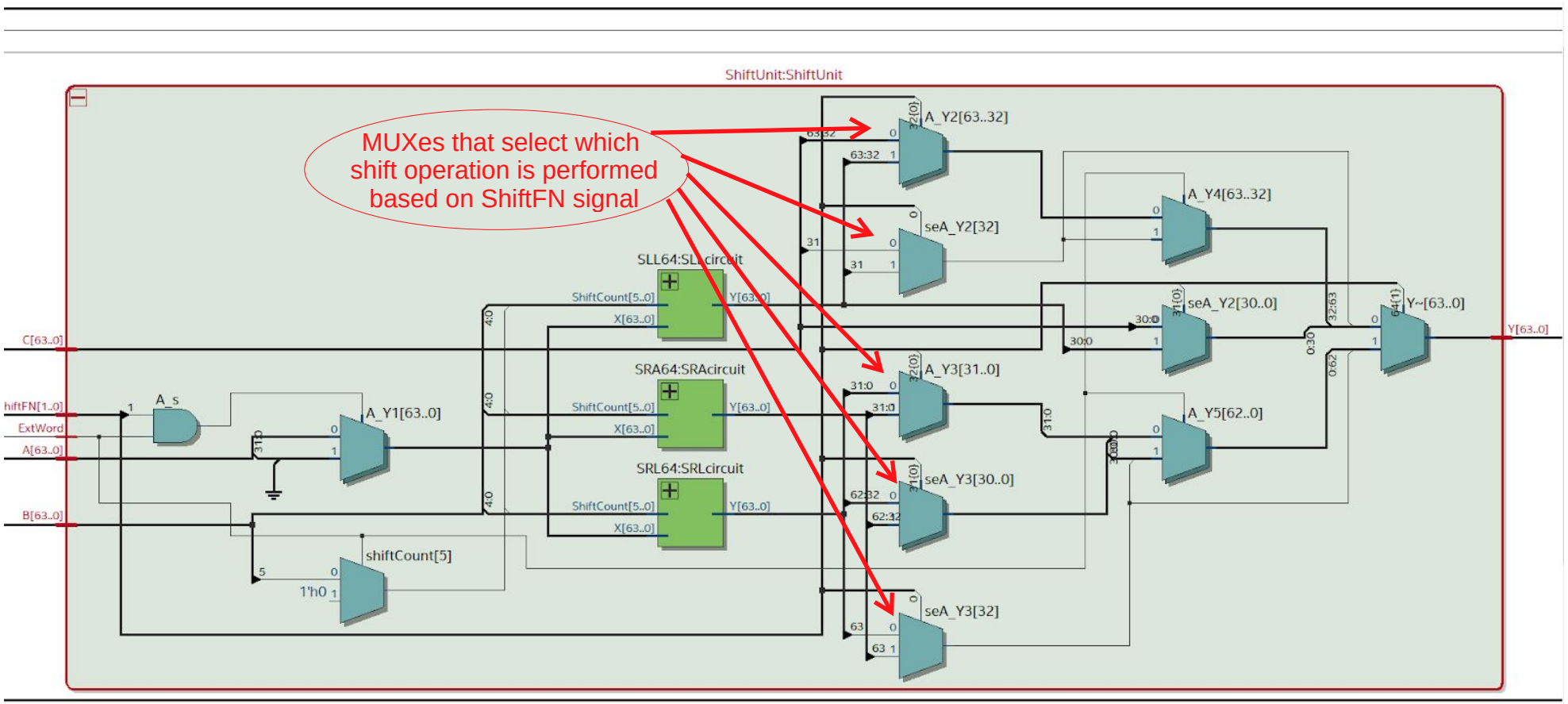
The screenshot above shows the output of the circuit. As annotated, the AddY output signal has been added to the arithmetic unit along with the other two outputs of the ripple adder (Cout and Sum). Right above the output AddY, there is also the Ovfl signal output.



Arithmetic Unit Post Fit:

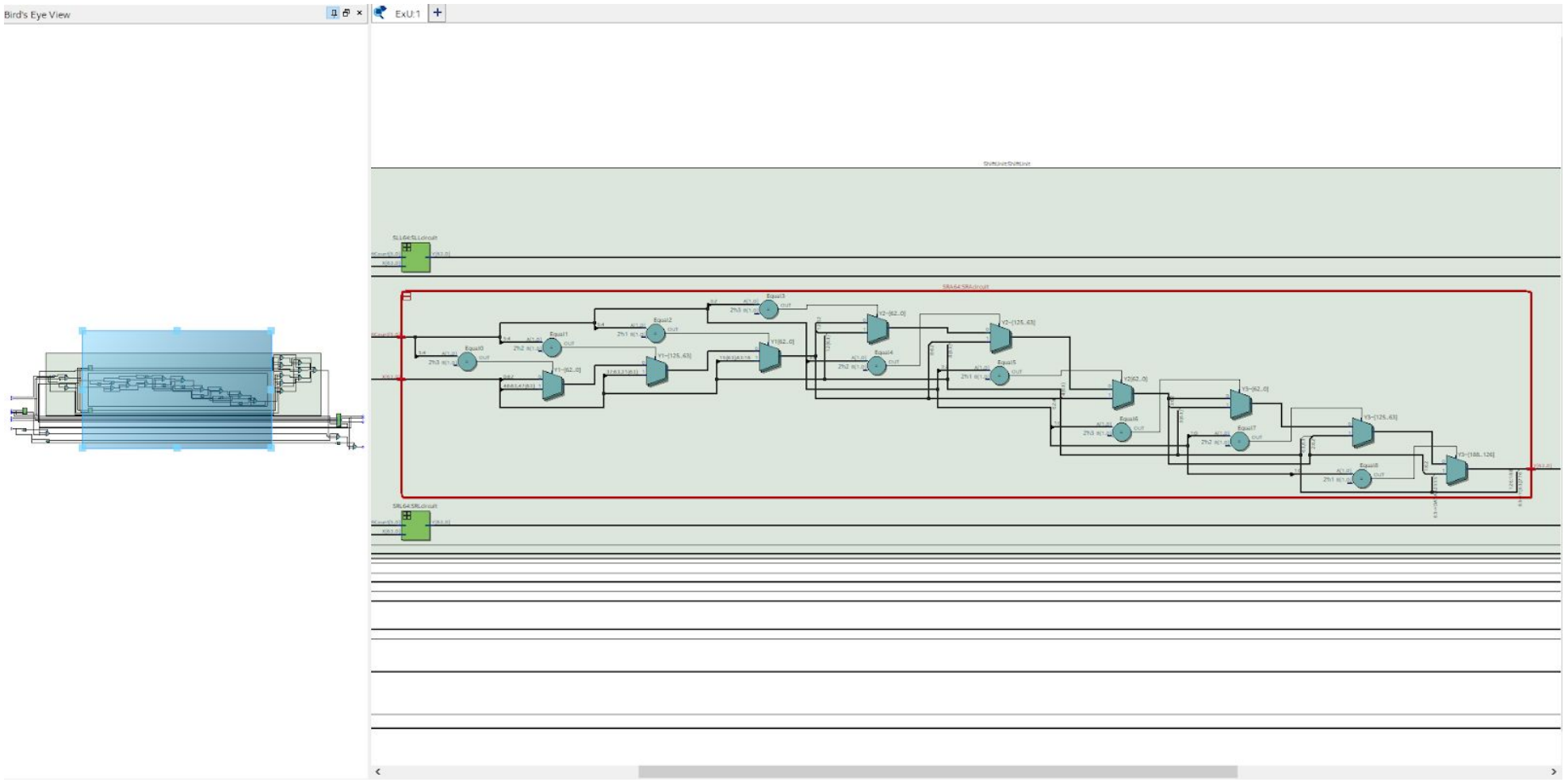
The screenshot above shows the overall structure of the ripple adder circuit. Each box represents a full adder. The final result of the adder comes out of the circuit at the bottom right as shown above.

Arithmetic Unit Post Fit:
This screenshot shows the output zero signal in the circuit. From the adder, the output bits are OR with each other and then go through a NOT in order to determine the Zero signal output here.



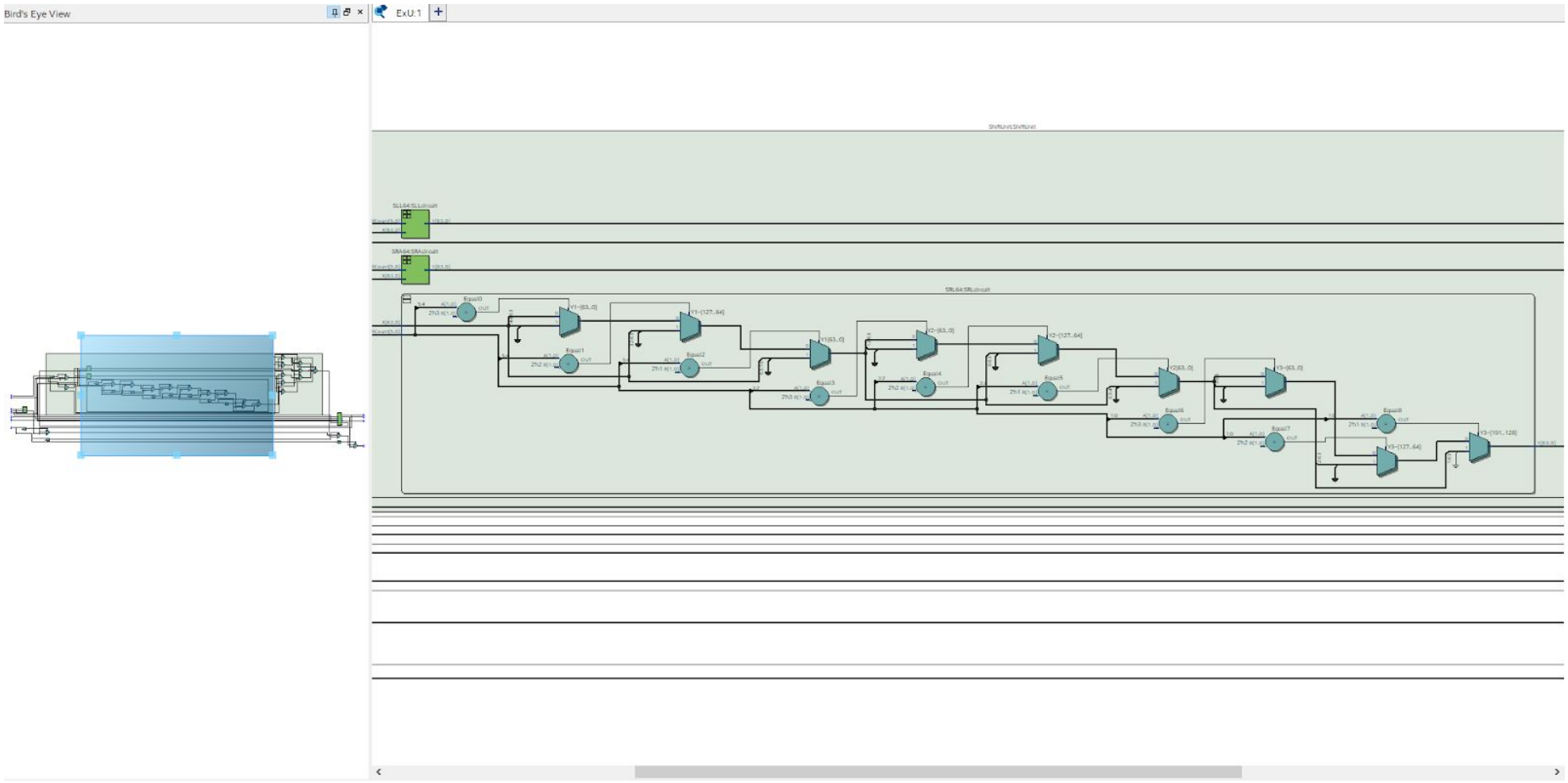
ShiftUnit RTL:

This screenshot shows the overall structure of the Shift Unit. The inputs of this circuit are operands A and B, ShiftFn signal and ExtWord Signal as shown on the left side of the circuit. Then, it will enter either SLL, SRA or SRL, depending on the ShiftFN signal. The resulting output of the shift will then go towards the MUXes to decide whether it needs to be sign extended or not based on the ExtWord input control signal.



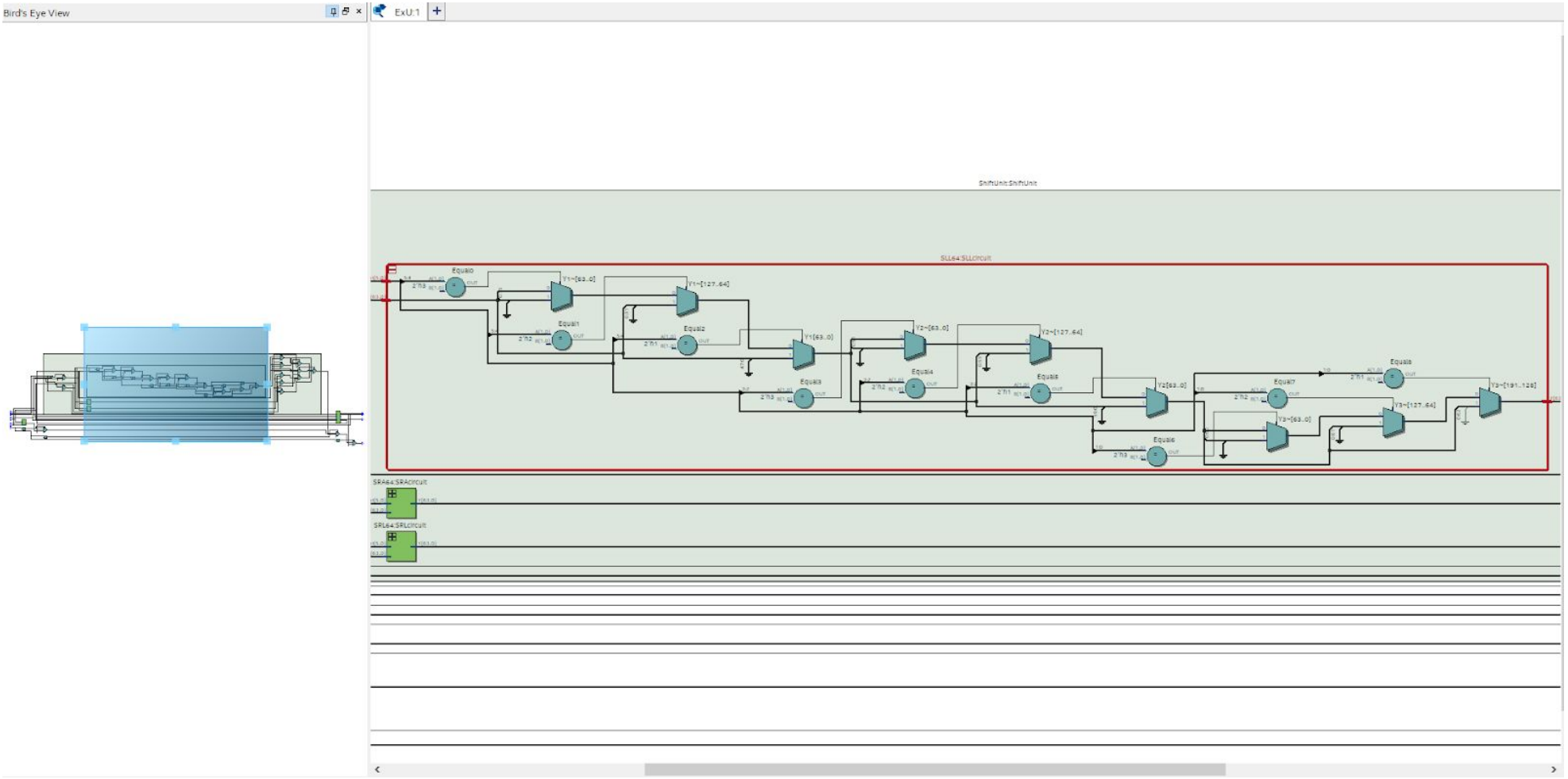
ShiftUnit RTL SRA64:

The screenshot above shows the overall shift right arithmetic unit. Our implementation of the SRA circuit uses multiple levels of MUXes. The first three MUXes determine if it needs to be shifted by 0,16,32, 48. The middle three MUXes determine if it needs to be shifted by 0,4,8,12. And the last three MUXes determine if it needs to be shifted by 0,1,2,3.



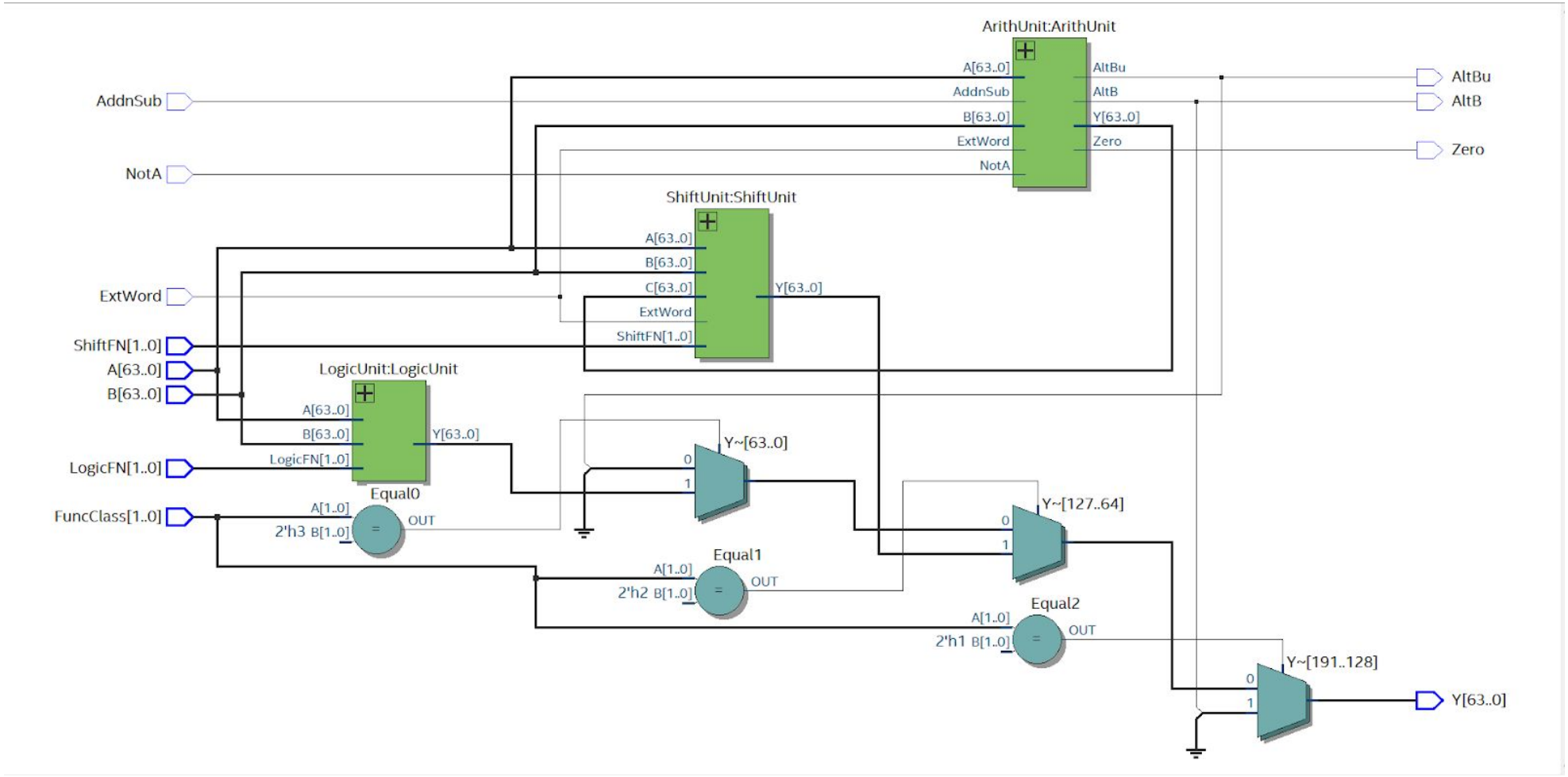
ShiftUnit RTL SRL64:

The screenshot above shows the overall shift right logic unit. Our implementation of the SRL circuit uses multiple levels of MUXes. The first three MUXes determine if it needs to be shifted by 0,16,32, 48. The middle three MUXes determine if it needs to be shifted by 0,4,8,12. And the last three MUXes determine if it needs to be shifted by 0,1,2,3.



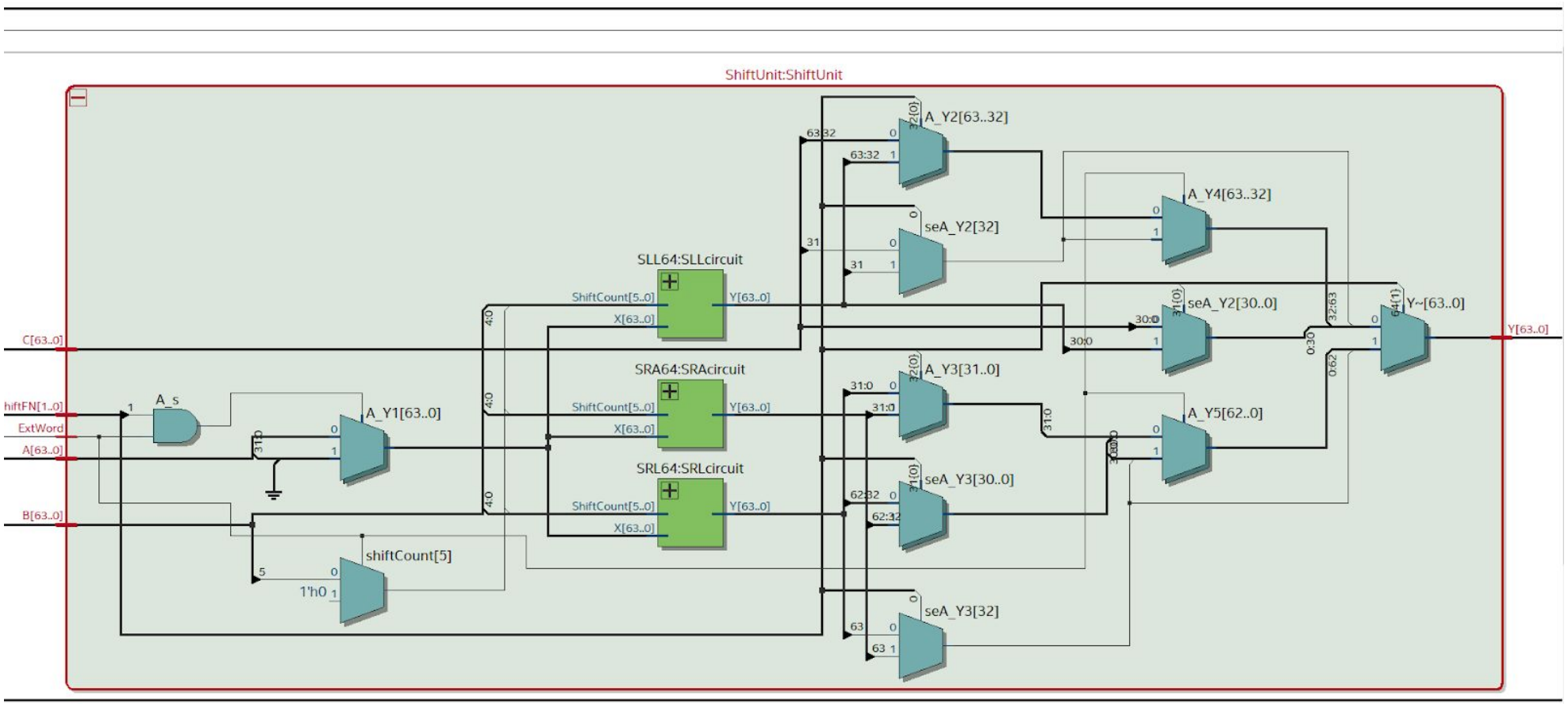
ShiftUnit RTL SLL64

The screenshot above shows the overall shift left logic unit. Our implementation of the SRL circuit uses multiple levels of MUXes. The first three MUXes determine if it needs to be shifted by 0,16,32, 48. The middle three MUXes determine if it needs to be shifted by 0,4,8,12. And the last three MUXes determine if it needs to be shifted by 0,1,2,3.



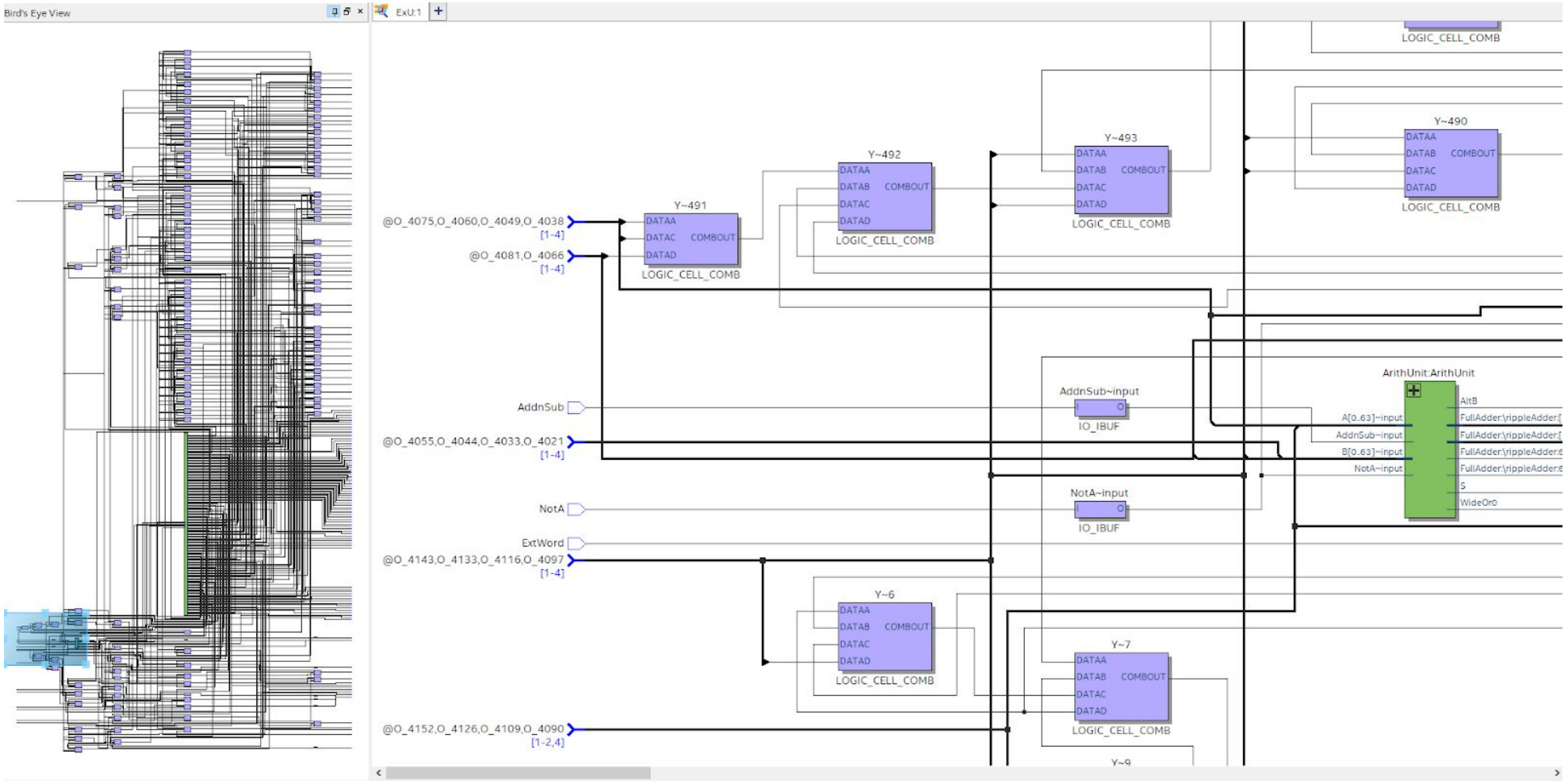
ExecUnit RTL:

This screenshot shows the overall structure of the Execution Unit. There are three subentities: ArithUnit, ShiftUnit, and LogicUnit. Operands A and B are shown on the left side of the diagram. FuncClass controls the three muxes to determine which sub entity output will be assigned to the Execution Unit output.

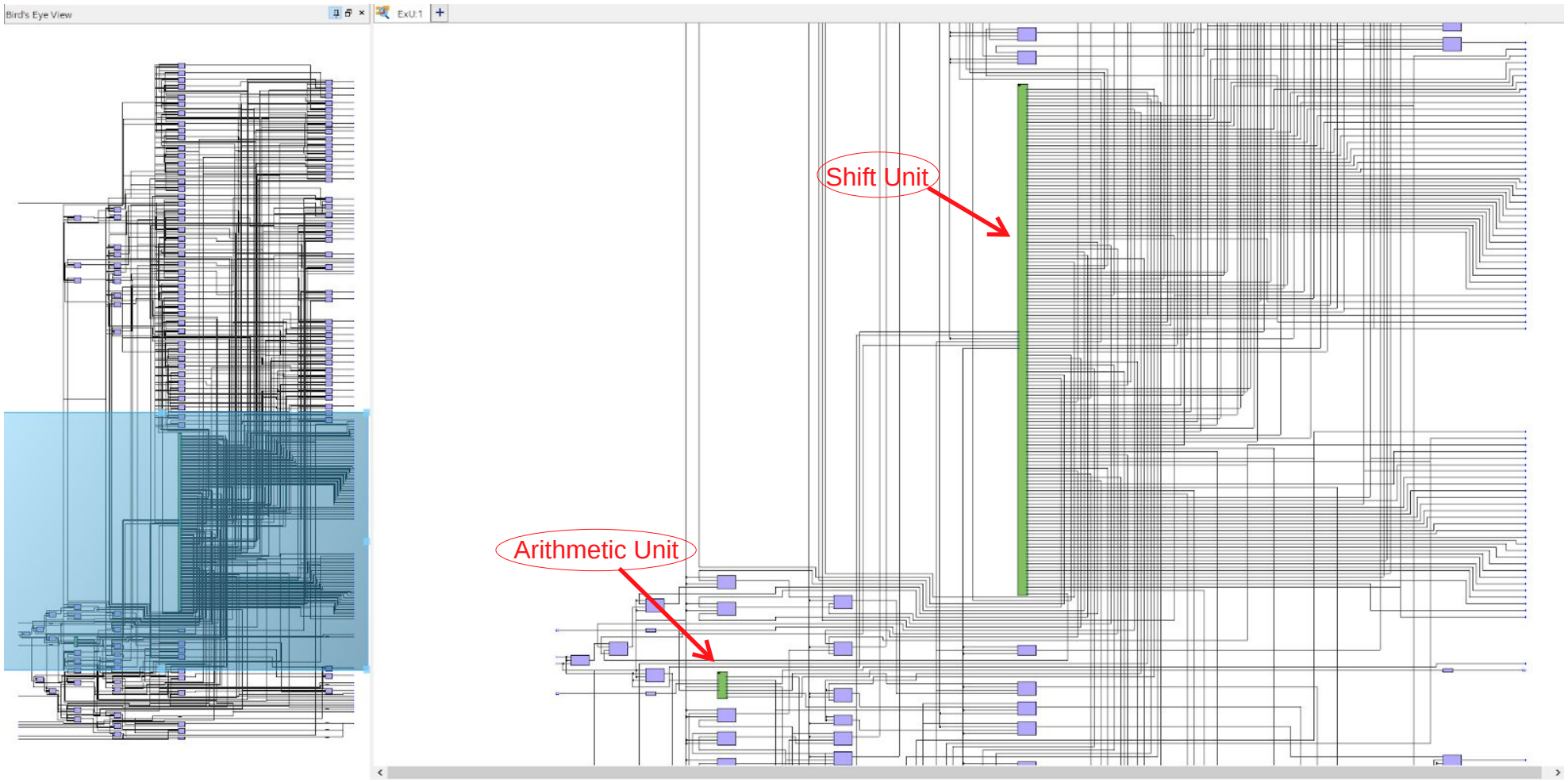


ExecUnit RTL Shift Unit:

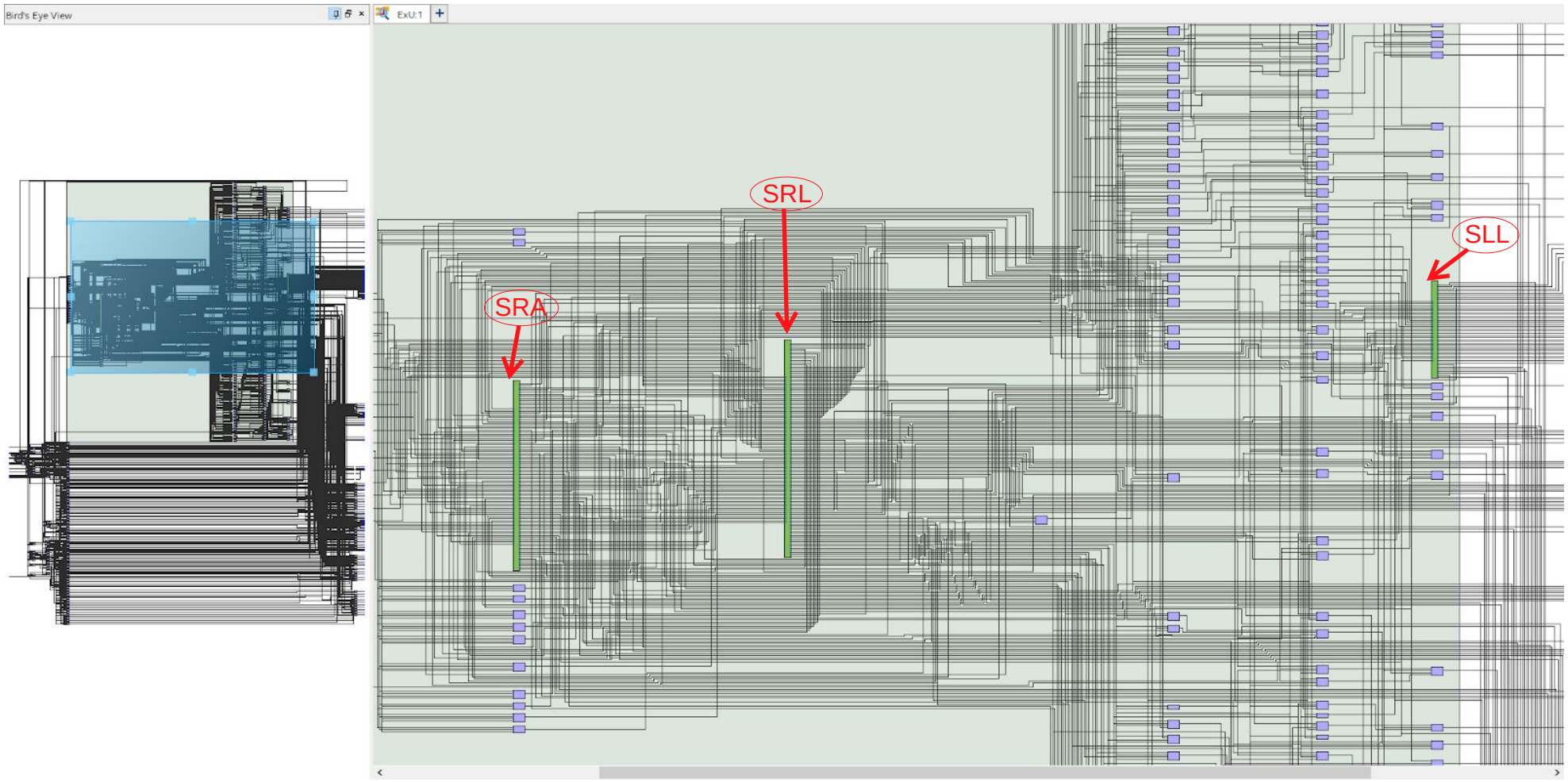
This screenshot shows the overall structure of the Shift Unit. The diagram shown above is exactly the same as the diagram shown in Shift Unit RTL, which shows that the shift unit is implemented correctly within the execution unit.



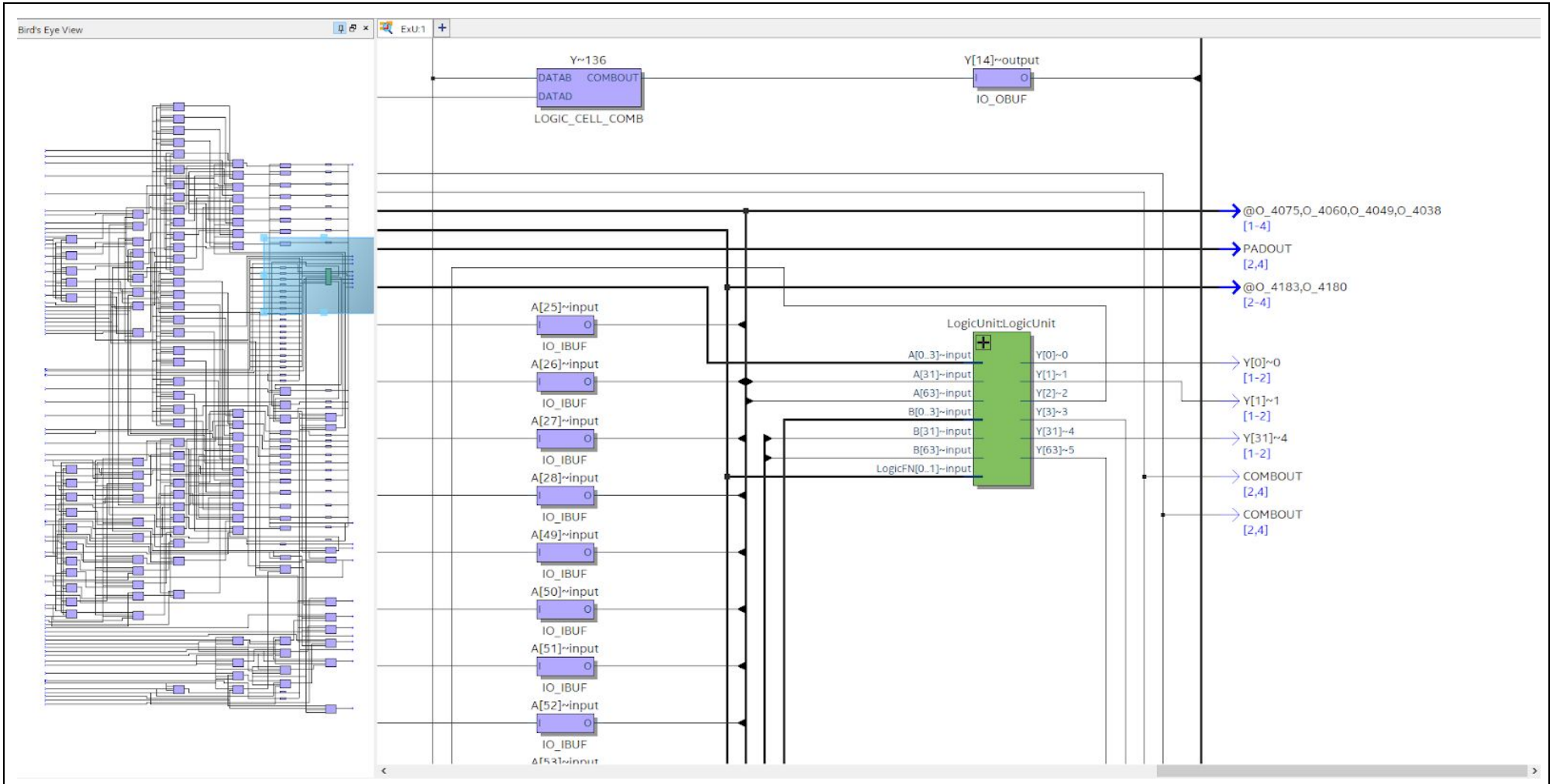
ExecUnit Post-Fit:
The screenshot above shows the signals for the arithmetic unit inside the execution unit.



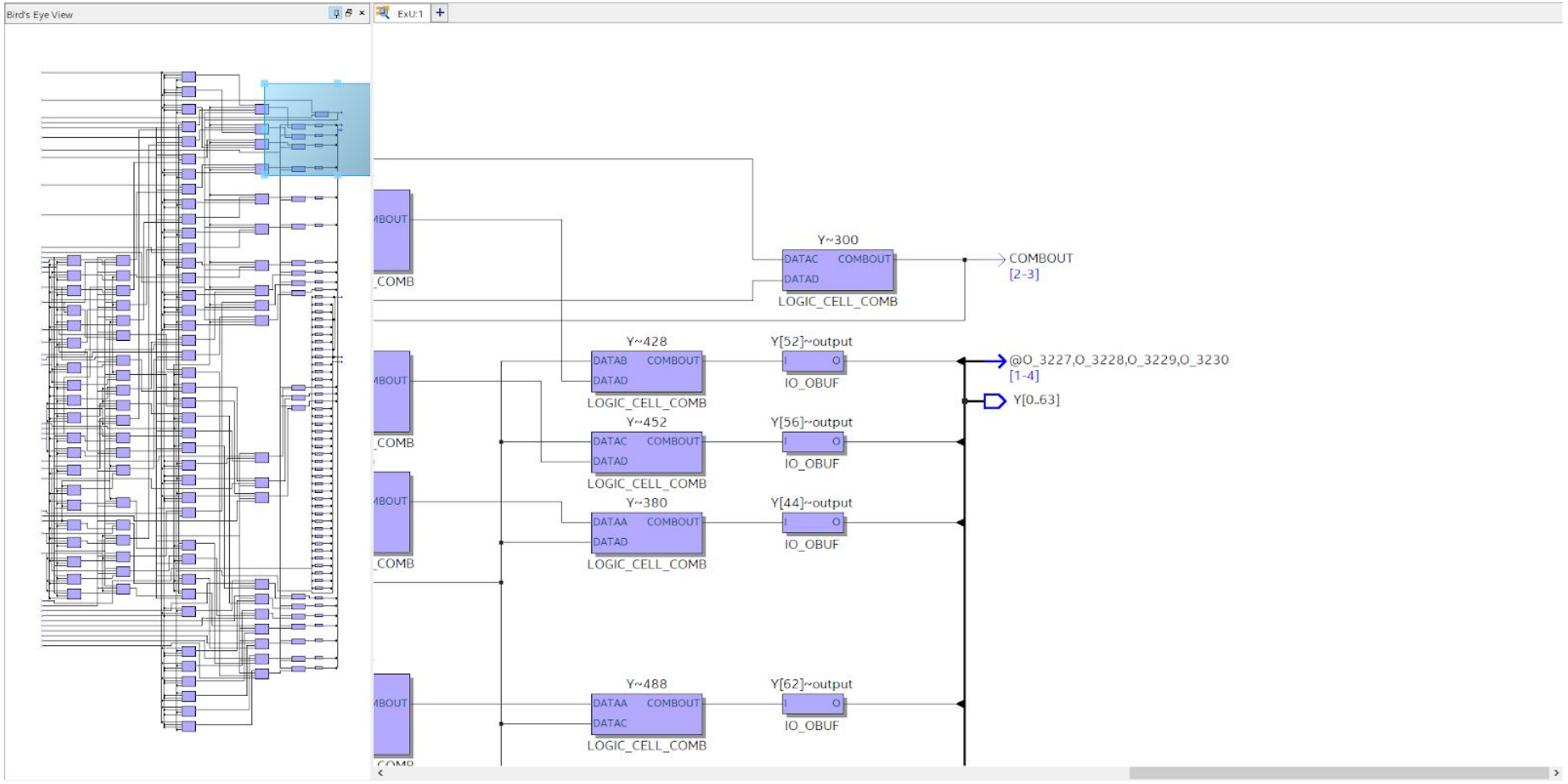
ExecUnit Post-Fit Shift Unit:
This screenshot shows two of the three subentities, Arithmetic Unit (Left) and Shift Unit (Right), inside the execution unit.



ExecUnit Post-Fit Shift Unit Expanded:
This screenshot shows the expanded shift unit, containing SRA, SRL, SLL (Left to right).



ExecUnit Post-Fit: Logic Unit:
This screenshot shows one of the three sub entities of the execution unit, which is the logic unit.



ExecUnit Post-Fit: Output Signal Y

The screenshot above shows the location of the final output Y value in the execution unit.

Functional Simulation Results and Discussion

After running the functional verification and capturing screenshots, from the wave screenshots for all the entities, it seems like the output Y matches the testbench output Y value and it seems like the status signals are correct according to the given input A and B. For the Logic Unit, the correct logical operation is performed depending on the control signal. For the Arithmetic Unit, the result from the Adder and the status signals corresponds to the operation that it is performing. Sign extension is performed correctly for the output Y for word instructions when ExtWord is '1'. For the Shift Unit, the output Y value is selected correctly based on the control signal ShiftFN, and all the outputs of a word instruction are sign extended correctly. Not only that, all the shift right instructions for 32 bits is performed correctly and the output is properly sign extended. For the Execution Unit, after combining all of the sub-entities, the Arithmetic Unit, the Logic Unit, and the Shift Unit, and linking their outputs to the 4-channel MUX, the whole Execution Unit functions properly. For example, the output Y matches the testbench output Y depending on which operation we are performing. The status signals for the Execution Unit is also calculated correctly depending on what the operation is. For example, for instructions where the resulting output Y is zero, the Zero status signal is 1. Control signal ExtWord also ensures that the results that require sign extension are properly handled.

Looking at our design for the RISC-V Execution Unit, it seems like it is definitely compatible with RV64I, with all the outputs and status signals computed correctly and the Execution Unit design following RISC-V standard.

Timing Simulation Results and Discussion

From completing the timing simulations and capturing the screenshots, we are able to verify that the Y value and the TbY value match, with propagation delay for the Y value due to the number of logic gates it must pass through. For the Logic Unit, we observed that the propagation delay is 14.9 ns for each measurement throughout the timing simulation. For the Arithmetic Unit, we observed that the propagation delay varies based on which operation is performed. We observed the longest propagation delay is when the input operands A and B are equal and are subtracted from each other. For the Shift Unit, the control signal ShiftFN selects which shifter to use. We can see that the propagation delay varies depending on how many bits we shift by. Finally for the Execution Unit, all three sub-entities, Arithmetic Unit, Logic Unit, and Shift Unit, are combined, where a 4-channel MUX determines the output of the execution unit with the control signal FuncClass. When the FuncClass selects a logical operation to be performed, the propagation delay, like in the

timing simulation of just the Logic Unit, is the same for all logical operations, although the propagation delay is slightly longer due to the extra 4-channel MUX at the end of the Execution Unit. For arithmetic operations and shift operations, the propagation delay varies based on which operation is needed.

Conclusion

After completing the design of each sub entity and verifying its functionality, connecting the Logic Unit, Arithmetic Unit and Shift Unit together results in a working Execution Unit. By designing each individual sub entity's functionality, verifying it, synthesizing and performing timing simulation, we were able to make sure that each unit worked as its intended purpose. The Execution Unit utilizes a 4-channel MUX to select which operation result will be outputted. Depending on the input control signal given, the MUX in the execution unit will select whether it will use the arithmetic, shift or the logic unit. The resulting propagation delay of the operation will correspond with the propagation delay of the unit that it uses. This completed Execution Unit computes all the outputs and status signals correctly and follows the RISC-V standards and is compatible with RV64I.