Group #:
G 36

Final Project Part 2

ENSC 350 2020

Last Name:
Gill

SID: 3 0 1 3 0 5 9 4 9

Last Name:
Rezghighomi

SID: 3 0 1 3 1 1 4 1 7

Last Name:
Qu

SID: 3 0 1 2 9 9 4 8 9

Last Name:

SID:

Table of Contents

Abstract

This project focused on work in incremental electronic design, functional verification, synthesis and timing verification. Following this procedure, a logic unit, arithmetic unit, and shift unit were constructed. All three circuits are combined to form the RISC-V execution unit. The resulting circuit is capable of signed and unsigned comparison, logical shift left, logical shift right, arithmetic shift right, AND, OR, and XOR operations. Functional verification is performed using testbenches for all four units and comparing circuit output against expected test vector results. Also, synthesis is performed on all four units and results are analyzed. Furthermore, timing simulations are conducted and results are compared with theoretical expectations. All aspects of design, functional verification, synthesis, and timing verification are explored in-depth in this report.

Design Entities

64-bit Logic Unit

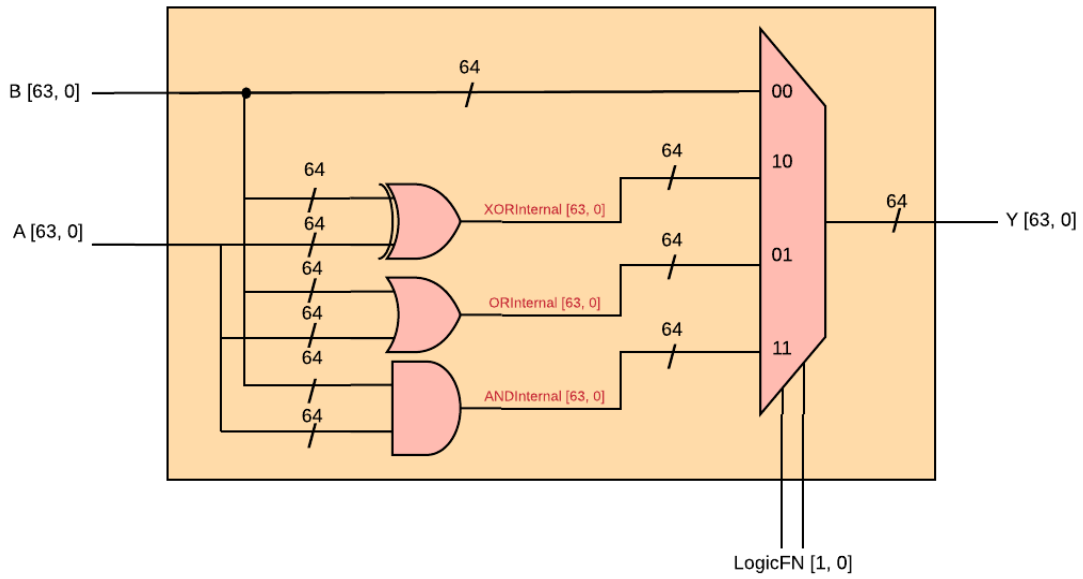

Figure 1: A circuit diagram of the Logic Unit. External signals are labeled in black, while internal signals are labeled in red. BUS size is also indicated.

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

Figure 2: The VHDL interface for the Logic Unit Entity. Signals A, B, and LogicFN are taken as input while signal Y is output.

The logic unit takes two N-bit inputs, A and B, and outputs a N-bit value, Y. Furthermore, the logic unit also possesses a 2-bit control-signal, *LogicFN*, which is used to control a N-bit, 4 channel multiplexor (MUX). All internal connections are N-bit BUSes. When the control signal, *LogicFN* is set to '10', the MUX selects the N-bit *XORInternal* internal signal, which carries the result of performing an bitwise exclusive-or operation on input signals A and B. Similarly, when the control signal, *LogicFN* is set to '01', the MUX selects the N-bit *ORInternal* internal signal, which carries the result of performing a bitwise OR operation on input signals A and B. Furthermore, when the control signal, *LogicFN* is set to '11', the MUX selects the N-bit *ANDInternal* internal signal, which carries the result of performing a bitwise AND operation on input signals A and B. The remaining channel is selected when *LogicFN* is set to '00', passing the input B through the logic unit.

64-bit Arithmetic Unit

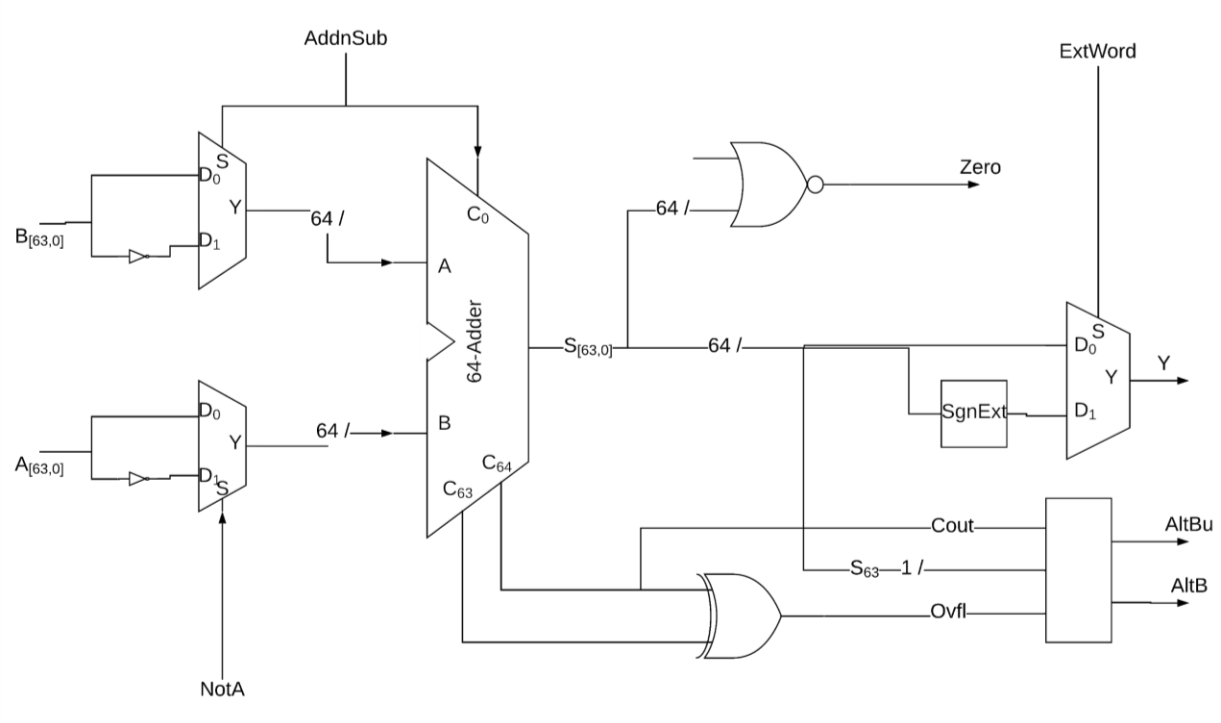

Figure 3: A circuit diagram of the ArithmeticUnit.

```
Entity ArithUnit 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, AddnSub, ExtWord : in std_logic := '0';
-- Status signals
       Cout, Ovfl, Zero, AltB, AltBu : out std_logic );
End Entity ArithUnit;
```

Figure 4: The VHDL interface for the Logic Unit Entity.VHDL comments indicate the control and status signals.

The arithmetic unit calculates arithmetic operations on two N-bit inputs, A and B, to produce a N-bit output, Y. 1-bit control signals *AddnSub* and *NotA* directs complementation of inputs A and B to facilitate signed arithmetic using an instance of the Adder entity. *AddnSub* is also used as the carry-in for the Adder. The *ExtWord* control signal allows the use of half-width inputs by allowing half-length results to be sign-extended to the full output signal length. The *Zero* status signal is calculated by nor-reducing the output from the Adder, and indicates the presence of a zero result. The arithmetic unit also outputs two status signals that indicates whether A is less than B: *AltBu* and *AltB*, respectively for cases of unsigned and signed arithmetic. *AltBu* is the complement of Adder's *Cout* output, and *AltB* is the XOR of *Ovfl* and the last bit of Y.

64-bit Ripple-Carry Adder

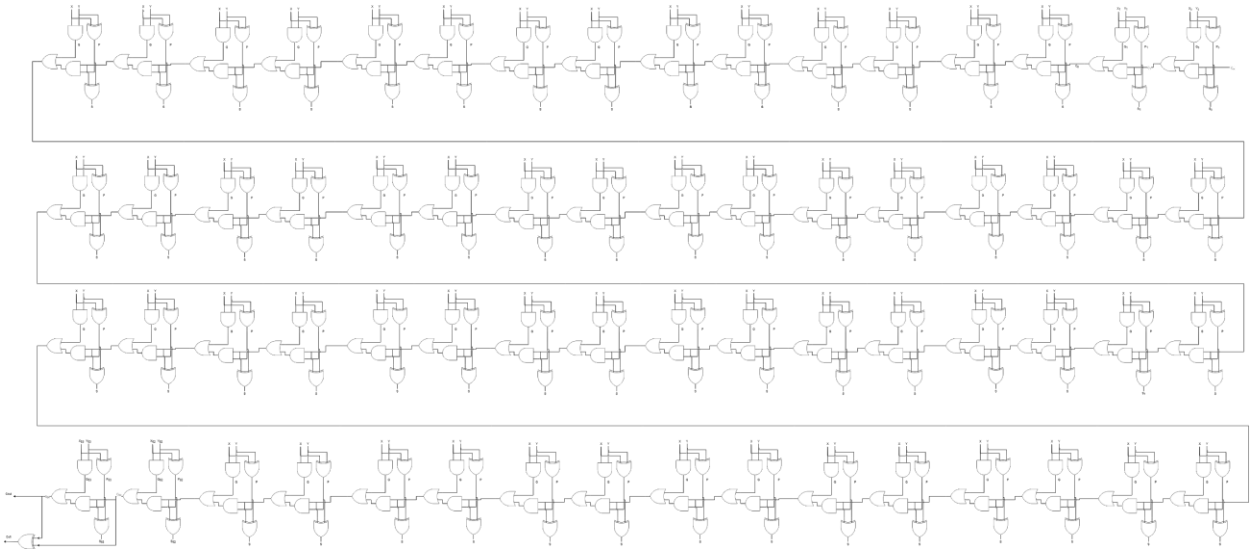

Figure 5: A circuit diagram of the ArithmeticUnit.

```
Entity Adder is
Generic ( N : natural := 64 );
Port ( A, B : in std_logic_vector( N-1 downto 0 );
       Y : out std_logic_vector( N-1 downto 0 );
-- Control signals
       Cin : in std_logic;
-- Status signals
       Cout, Ovfl : out std_logic );
End Entity Adder;
```

Figure 6: The VHDL interface for the Logic Unit Entity.

The Adder entity takes two N-bit signals, A and B, and performs addition in order to produce a N-bit output signal Y. The output is calculated using a standard ripple-carry adder implementation with 3 distinct networks: generate-propagate, carry, and sum, where each bit has 5 gates that form a full adder. The adder also has a 1-bit input signal *Cin* for a carry-in bit for the addition process. Two status signal outputs, *Cout* and *Ovfl*, respectively indicate the carry-out bit of the addition and the presence of arithmetic overflow. *Cout* is calculated by inverting the last carry bit, and *Ovfl* is calculated by XORing the last two carry bits.

64-bit Barrel Shifter Overview

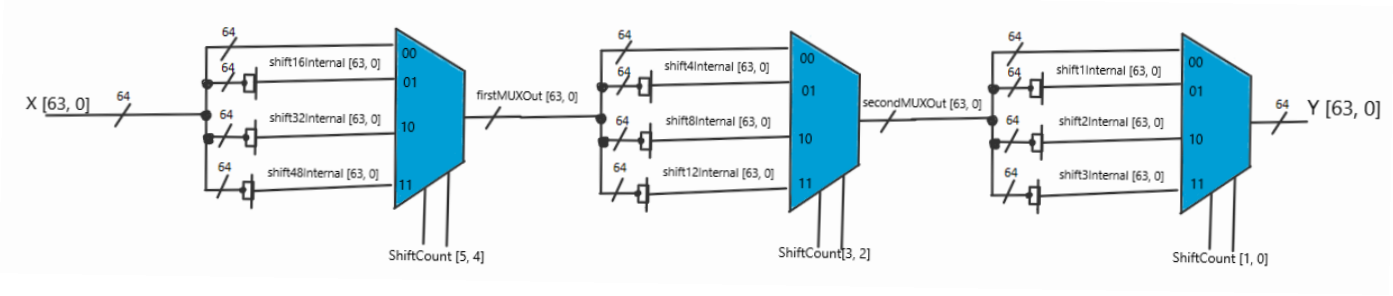

Figure 7: Circuit Diagram depicting a typical barrel shifter composed of three 64-bit, 4 channel multiplexers.

Multiplexor 1:

Control Signal ShiftCount[5, 4]	Shift Count
00	0
01	16
10	32
11	48

Multiplexor 2:

Control Signal ShiftCount[5, 4]	Shift Count
00	0
01	4
10	8
11	12

Multiplexor 3:

Control Signal ShiftCount[5, 4]	Shift Count
00	0
01	1
10	2
11	3

Figure 8: Tables depicting the shift-count for the three multiplexors contained in each of the three barrel shifters that compose the Shift Unit.

The three barrel shifters all follow the same structure with the difference between each shift operation residing in the implementation of the actual shift-circuitry being selected by each multiplexor. The shifter works on the principle of dividing a single shift operation amongst a series of intermediate shift operations, a zero- shift just selecting the value without passing through any shifting circuitry. This is illustrated in the table below:

Shift Count (Decimal)	Shift Count (Binary)	Control Bits [5, 4] for MUX1	Shift Count MUX1	Control Bits [3, 2] for MUX2	Shift Count MUX2	Control Bits [1, 0] for MUX3	Shift Count MUX3	Total Shift at Output
53	110101	11	48	01	4	01	1	48+4+1
25	011001	01	16	10	8	01	1	16+8+1
6	000110	00	0	01	4	10	2	0+4+2

Figure 9: Table demonstrating a single-shift operation being divided amongst three multiplexors.

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

Figure 10: The standard VHDL Barrel-Shifter port mapping.

Since each of the barrel shifters follows the circuit design described on the previous page, each barrel shifter has an identical port mapping. The design entity possesses a single N-bit input, *X*, declared as a standard-logic vector, a N-bit output, *Y,* declared as a standard-logic vector, and a log(N-1)-bit control signal, *ShiftCount*, declared as an unsigned value.

64-bit Shift Logical Left Barrel Shifter

```
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 );
  ShiftCount : in unsigned( integer(ceil(log2(real(N))))-1 downto 0 ) );
End Entity SLL64;
```

Figure 11: The VHDL interface for the Shift-Logical-Left Barrel Shifter Entity.

Since each of the barrel shifters follows the design template previously mentioned, the only difference lies in the implementation of the shifting circuitry. An M-bit shift logical-left occurs while filling the lower M-bits with zero. More precisely, let *i* represent the bit to be shifted, the wires are connected so that the *ith* bit is shifted to the *i* + M position, if *i* + M ≤ N-1; otherwise if *i* + M > N-1, these bits are connected to nothing and are grounded. Moreover, the lower M-bits are connected to ground. For example, when performing a logical-left shift of 5-bits on a 64-bit binary number using the 3-multiplexor barrel shifter with a 6-bit control signal, the shift operation is broken down into a 0-bit shift, 4-bit shift, and a 1-bit shift. For the 0-bit shift, the input value is selected using the upper two bits of the shift count to control the first multiplexor. Consequently, the second multiplexor selects a 4-bit shift using the middle 2-bits of the shift count. Of the 64-bit output of the first multiplexor, the wires of bits 63 to 60 are discarded and connected to ground while the wires for bits 59 to 0 are shifted left by 4 bit-positions. The wires for the lower 4-bits of the shifted number are connected to ground. Moreover, the third multiplexor selects a 1-bit shift using the lower 2-bits of the shift count; of the 64-bit output of the second multiplexor, the wire for the upper bit is discarded and connected to ground while the wires for bits 62 to 0 are shifted left by 1 bit-positions. The wire for the lower bit of the shifted number is connected to ground.

64-bit Shift Logical Right Barrel Shifter

```
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 );
  ShiftCount : in unsigned( integer(ceil(log2(real(N))))-1 downto 0 ) );
End Entity SRL64;
```

Figure 12: The VHDL interface for the Shift-Logical-Right Barrel Shifter Entity.

Similar to the left-logical shift barrel shifter described previously, the only difference from the barrel shifter design template lies in the implementation of the shifting circuitry. An M-bit logical-right shift occurs while filling the upper M-bits with zero. More precisely, let *i* represent the bit to be shifted, the wires are connected so that the *ith* bit is shifted to the *i* - M position, if *i* - M ≥ 0; otherwise if *i* - M < 0, these bits are discarded and connected to ground. Moreover, the upper M-bits are connected to ground. For example, when performing a logical-right shift of 6-bits on a 64-bit binary number using the 3-multiplexor barrel shifter with a 6-bit control signal, the shift operation is broken down into a 0-bit shift, 4-bit shift, and a 1-bit shift. For the 0-bit shift, the input value is selected using the upper two bits of the shift count to control the first multiplexor. Consequently, the second multiplexor selects a 4-bit shift using the middle 2-bits of the shift count. Of the 64-bit output of the first multiplexor, the wires for bits 63 to 4 are shifted right by 4 bit-positions while bits 3 to 0 are discarded and the wires are connected to ground. The wires for the upper 4-bits of the shifted number are connected to ground. Moreover, the third multiplexor selects a 1-bit shift using the lower 2-bits of the shift count. Of the 64-bit output of the second multiplexor, the wires for bits 63 to 1 are shifted right by 1 bit-position while the lower bit is discarded and the wire is connected to ground. The wire for the upper bit of the shifted number is connected to ground.

64-bit Shift Arithmetic Right Barrel Shifter

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

Figure 13: The VHDL interface for the Shift-Right-Arithmetic Barrel Shifter Entity.

Similar to the left-logical shift and right-logical shift barrel shifters described previously, the only difference from the barrel shifter design template lies in the implementation of the shifting circuitry. An M-bit right-arithmetic shift occurs while filling the upper M-bits with the sign bit. More precisely, let *i* represent the bit to be shifted, the wires are connected so that the *ith* bit is shifted to the *i* - M position, if *i* - M ≥ 0; otherwise if *i* - M < 0, these bits are discarded and connected to ground. Moreover, the wire of the sign bit is connected to the upper M-bits. For example, when performing a shift right-arithmetic operation of 5-bits on a 64-bit binary number using the 3-multiplexor barrel shifter design with a 6-bit control signal, the shift operation is broken down into a 0-bit shift, 4-bit shift, and a 1-bit shift. For the 0-bit shift, the input value is selected using the upper two bits of the shift count to control the first multiplexor. Consequently, the second multiplexor selects a 4-bit shift using the middle 2-bits of the shift count. Of the 64-bit number output from the first multiplexor, the wires for bits 63 to 4 are shifted right by 4 bit-positions while bits 3 to 0 are discarded and the wires are connected to ground. The wire of the sign-bit of the output of the first multiplexor is connected to the upper 4-bits of the shifted number. Moreover, the third multiplexor selects a 1-bit shift using the lower 2-bits of the shift count. Of the 64-bit output of the second multiplexor, the wires for bits 63 to 1 are shifted right by 1 bit-position while the lower bit is discarded and the wire is connected to ground. The wire of the sign-bit of the output of the second multiplexor is connected to the upper bit of the shifted number.

Shift Unit

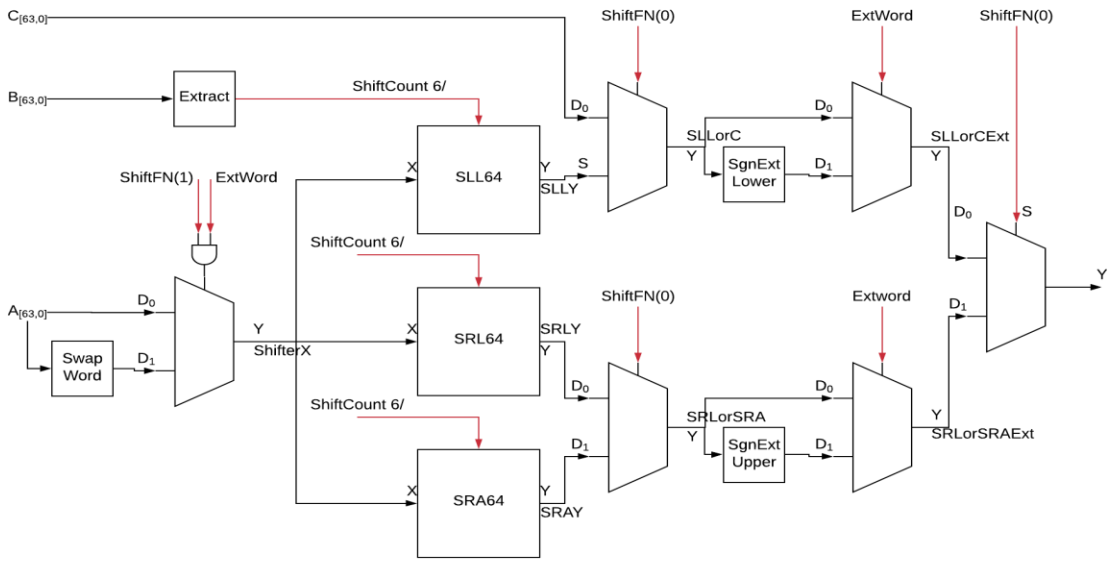

Figure 14: The circuit diagram of the shift unit entity.

```
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 );
           ShiftFN : in std_logic_vector( 1 downto 0 );
           ExtWord : in std_logic );
End Entity ShiftUnit;
```

Figure 15: The VHDL interface for the Shift Unit Entity.

ShiftFN	operation
00	C pass-through
01	Shift logical left (SLL)
10	Shift right logical (SRL)
11	Shift right arithmetic (SRA)

Figure 16: Tables depicting the different operations performed by Shift Unit based on values of ShiftFN

The shift unit produces shift operations on the 64-bit input A or pass-through 64-bit input C and outputs the result in *Y*, as controlled by the control signals *B*, *ShiftFN*, and *ExtWord*. *ShiftFN* Of "00" will cause *Y* to be assigned the value of *C* with no modifications. Other values of *ShiftFN* will cause different shift operations to be performed on *A* according to the table above, the shifting amount is determined by the last bits of *B*, the size of the shifting amount is determined by *ExtWord*. When *ExtWord* is '1' (indicating 32-bit data), the last 5 bits of *B* are taken as the shift amount, but when *ExtWord* is '0' (indicating 64-bit data), the last 6 bits of *B* are taken as the shift amount. This means the maximum shift amount is 32 and 64 bits respectively. When *ExtWord* is asserted, the shift or pass-through operation result is sign extended before the final output by 32 bits. In cases of C pass-through and SLL, the 31st bit is extended, whereas the 63rd bit is extended for SRL and SRA results. Additionally in cases of SRL and SRA operations on a 32-bit data vector, the upper and lower words of *A* will be swapped in place.

Execution Unit

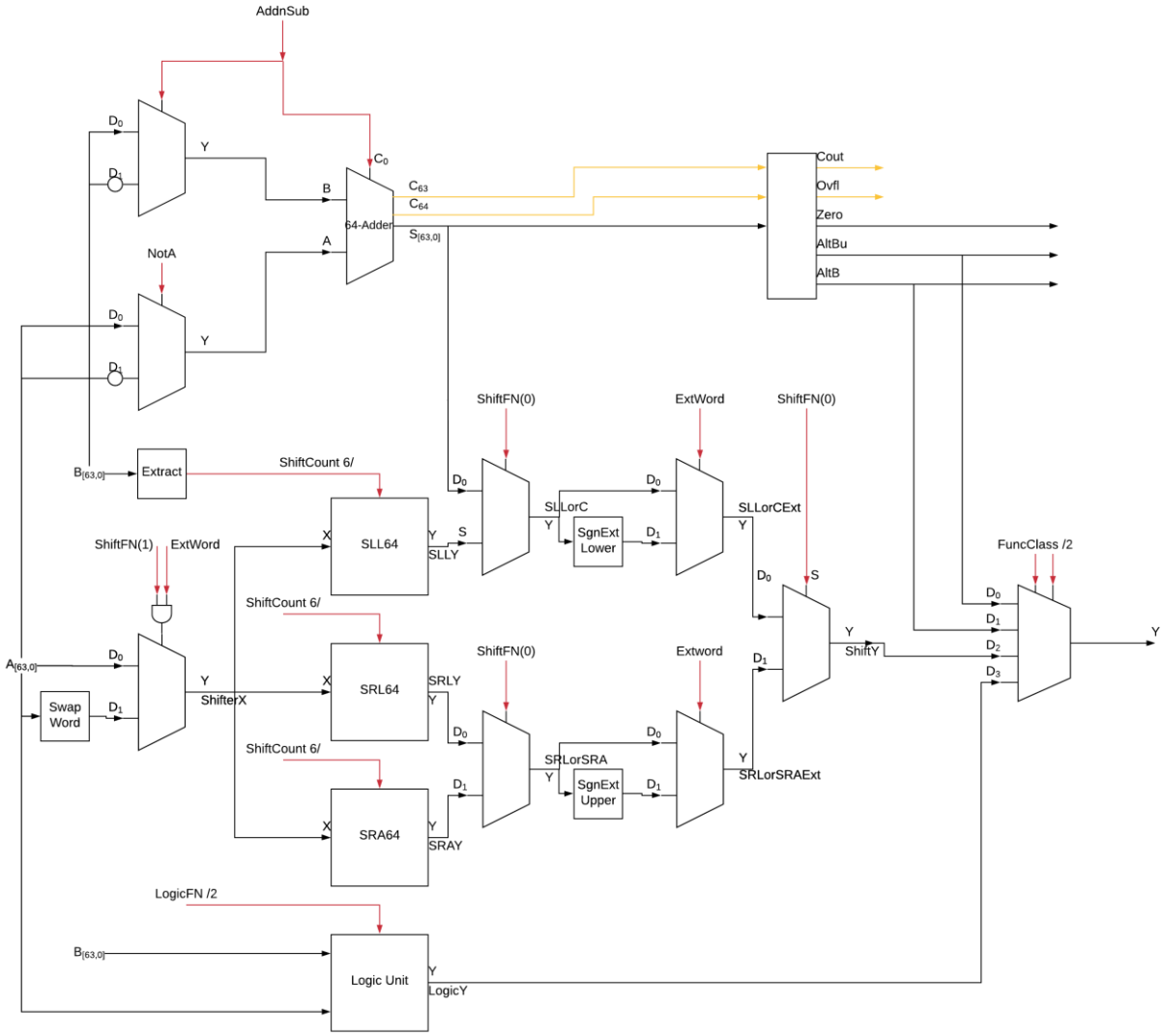

Figure 17: The circuit diagram for the completed Execution Unit.

```
Entity ExecUnit is
Generic ( N : natural := 64 );
Port ( A, B : in std_logic_vector( N-1 downto 0 );
       NotA : in std_logic := '0';
       FuncClass, LogicFN, ShiftFN : in std_logic_vector( 1 downto 0 );
       AddnSub, ExtWord : in std_logic := '0';
       Y : out std_logic_vector( N-1 downto 0 );
       Zero, AltB, AltBu : out std_logic  );
End Entity ExecUnit;
```

Figure 18: The VHDL interface for the Execution Unit Entity.

FuncClass	operation
00	Set on less than unsigned (SLTU)
01	Set on less than (SLT)
10	shift/arithmetic
11	logic

Figure 19: Tables depicting the different operations performed by Exec Unit based on values of FuncClass

The execution unit combines the arithmetic unit, shift unit, and the logic unit all into one. The choice of output is determined by the control signal *FuncClass* as listed in the table above. The arithmetic status signals *Cout* and *Ovfl* lose their importance in the combined circuit and are not featured in the Execution Unit's output ports. More notably, the arithmetic unit is now also used to facilitate comparison operations between *A* and *B* in addition to arithmetic operations. To facilitate this, the first 2 of the 4 possible outputs are used to output 0-extended single bit statuses at *Y* from the arithmetic unit of comparison results for signed and unsigned cases. Arithmetic operation result is rewired to become the C pass-through signal previously found in the Shift Unit, thus now the shift and arithmetic results are combined in a single *FuncClass* value of "10". To this end, the *ShiftFN* control signal has also been modified in order to produce the arithmetic operation result put into *Y* when it is "00". Finally the logic unit result can be selected for output at *Y* with the *FuncClass* value of "11". The Logic Unit control signal *LogicFN* is also changed to perform load upper immediate operation instead of signal *B* pass-through for value "00". Other than those changes, the rest of the individual units' functionalities have been left unaltered. The arithmetic unit is capable of input complementation (based on control signals *NotA* for *A* and *AddnSub* for *B*) and performs addition between *A* and *B*, and is used to output both arithmetic result data and comparison result signal, as well as asserting the status signals for zero output, signed comparison, and unsigned comparison. The Shift Unit is controlled by *ShiftFN* to control between results from arithmetic, SLL, SRL and SRA circuits, while input *B* is used as a control signal to determine number of bit shifts, outputs are optionally sign-extended to 64 bits to allow for half-width data input. The Logic Unit selects the logic operation based on the control signal *LogicFN* and outputs the corresponding result of the operation applied to data inputs *A* and *B*.

Functional and Simulation Waves:

Logic Unit

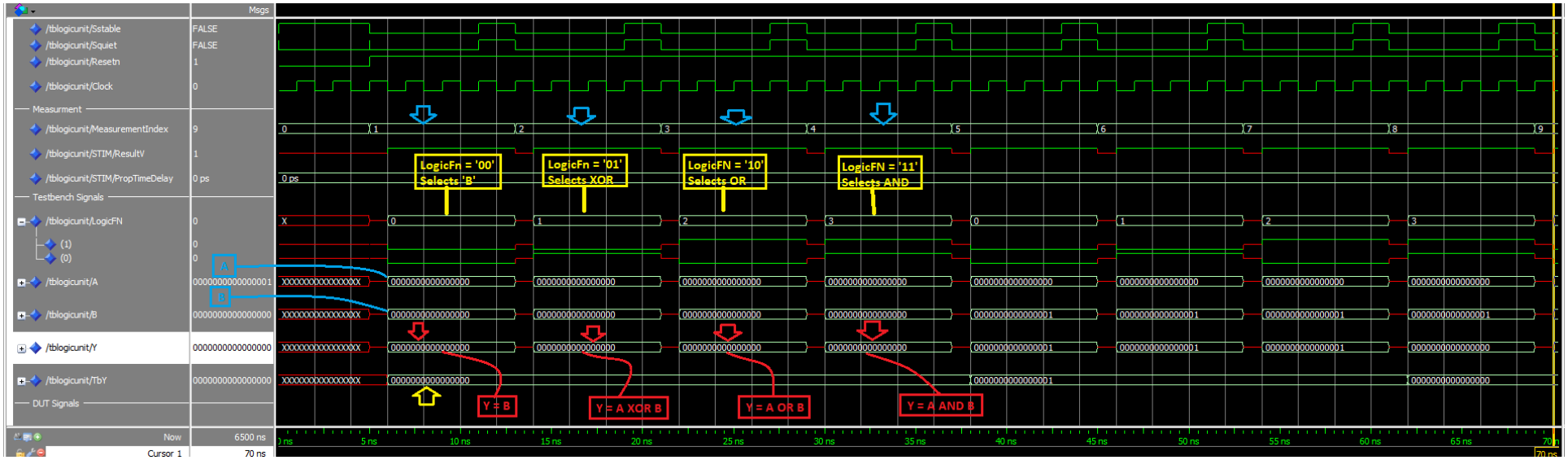

Figure 20: Functional Simulation of the Logic Unit showing signals from 0 ns – 70 ns. We can observe measurements 1 -4 (blue arrows) and compare obtained values (red arrows) with the desired value (yellow arrow) to verify the functionality of the logic unit. The control signal values are indicated in yellow, while the output values are indicated in red and the input values are indicated in blue.

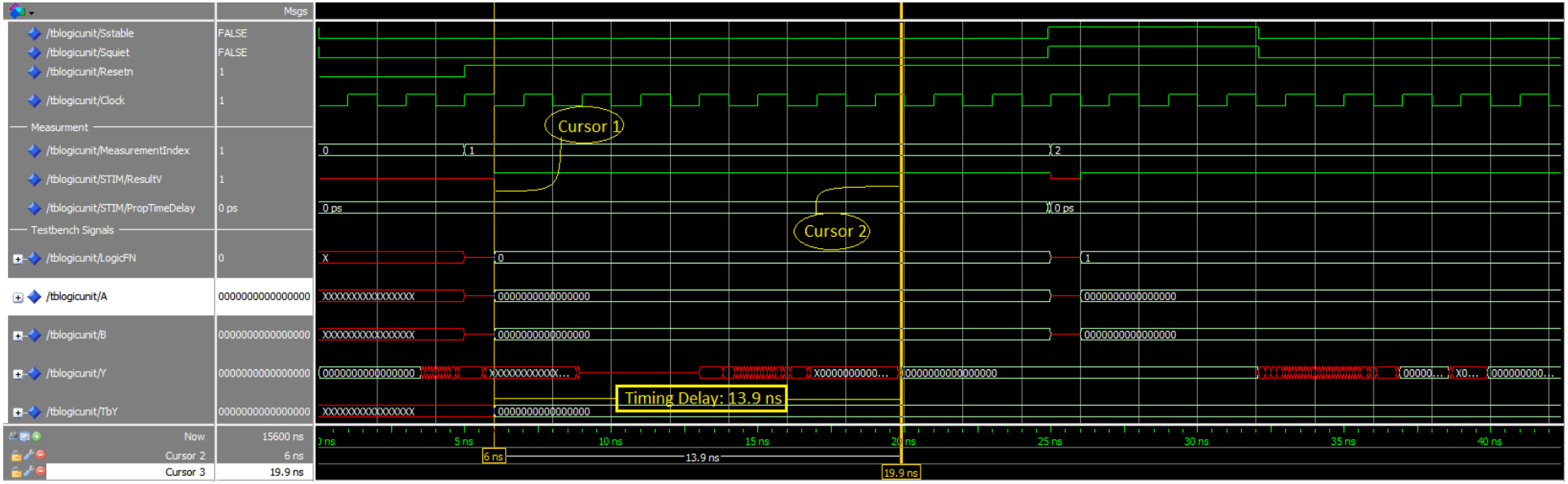

Figure 21: Timing Simulation of the Logic Unit showing the propagation delay of measurement #1. Cursor 1 indicates the beginning of measurement #1, which begins on the rising edge of the clock. Cursor 2 indicates the beginning of stable output of signal Y. The difference provides a propagation delay of t_{pd} = 13.9 ns.

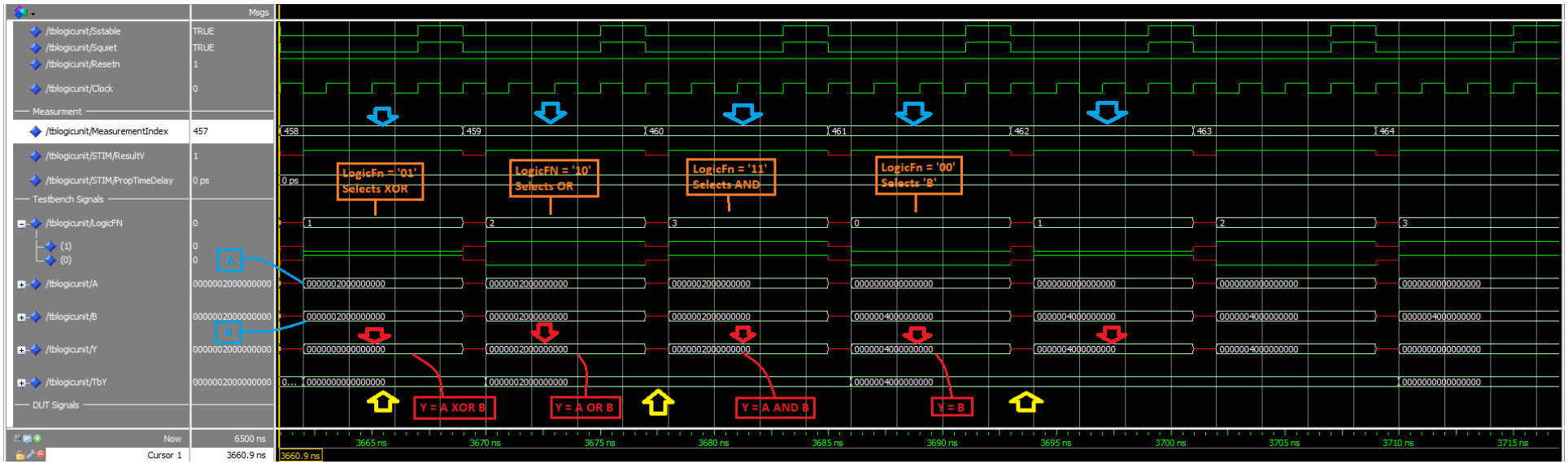

Figure 22: Functional Simulation of the Logic Unit showing signals from 3660.9ns – 3718 ns. We can observe measurements 458 -462 (blue arrows) and compare obtained values (red arrows) with the desired value (yellow arrow) to verify the functionality of the logic unit. The control signal values are indicated in blue. The input signals A and B are indicated by blue labels, while the output Y is indicated by red labels.

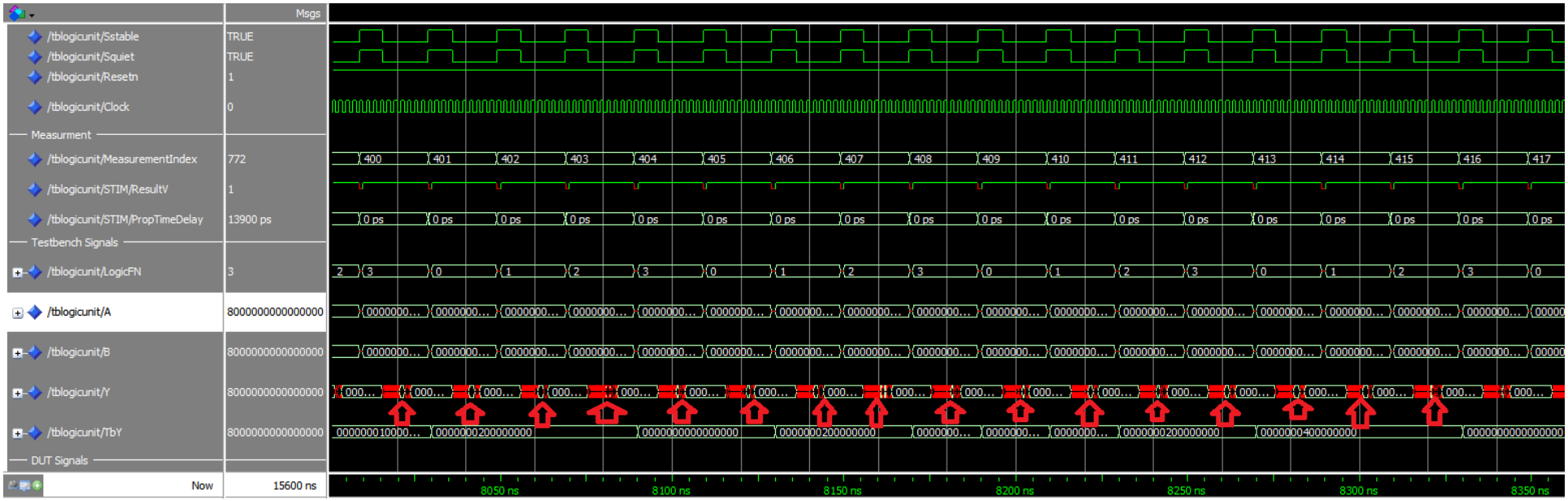

Figure 23: Timing Simulation of the Logic Unit showing measurements 400 – 416. The red portions (red arrows) indicate signal glitches before the output signal Y can stabilize.

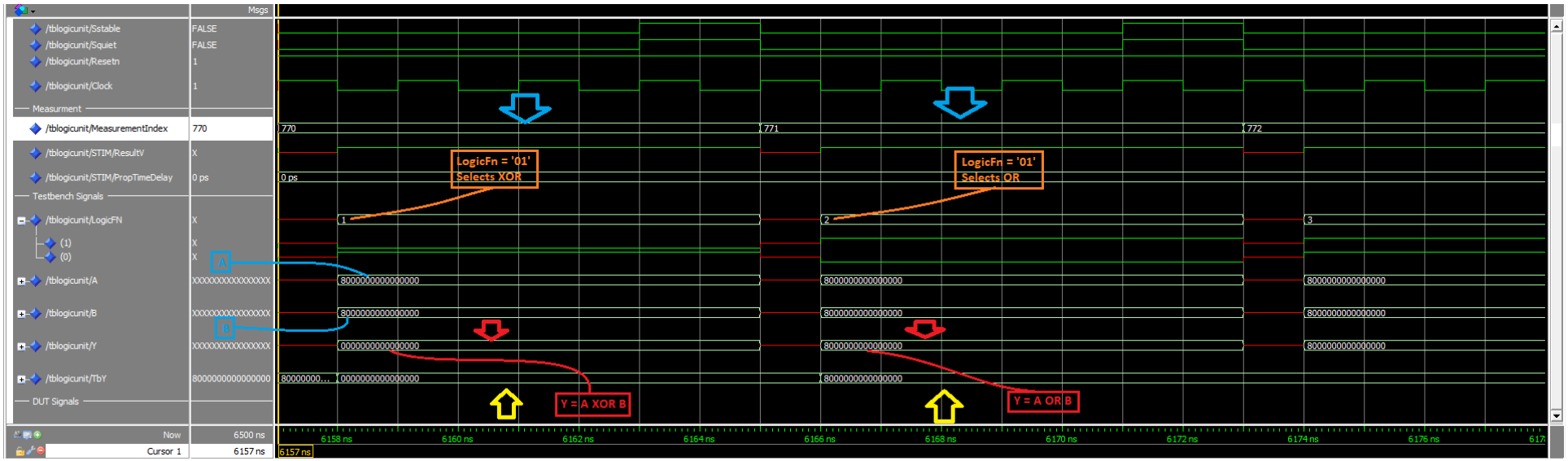

Figure 24: Functional Simulation of the Logic Unit showing signals from the beginning of measurement 458 to 5ns into measurement 772. We can verify measurements 770 and 771 (blue arrows) by comparing obtained values (red arrows) and desired values (yellow arrows). The control signal values are indicated in blue. The input signals A and B are indicated by blue labels, while the output Y is indicated by red labels.

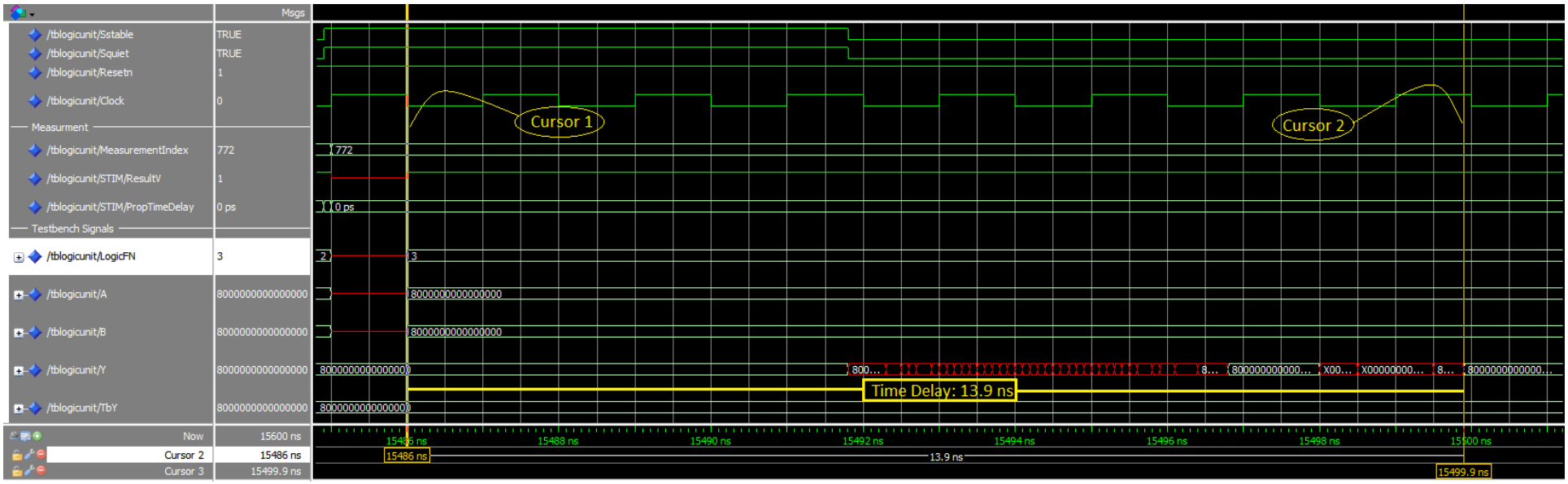

Figure 25: Timing Simulation of the Logic Unit showing the propagation delay of measurement 772. Cursor 1 indicates the beginning of measurement 772, which begins on the rising edge of the clock. Cursor 2 indicates the beginning of stable output of signal Y. The difference provides a propagation delay of t_{pd} = 13.9 ns.

Arithmetic Unit:

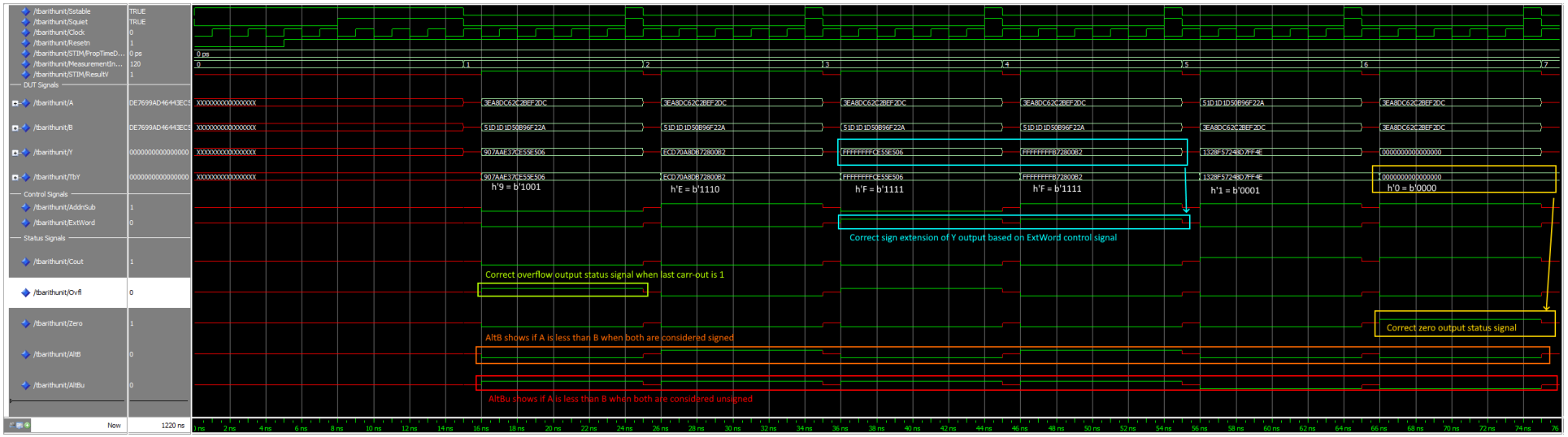

Figure 26: waveFAU1, from t = 0 to 76 ns. The waveFAU1 waveform pattern shows 6 calculations, in all cases the Y output match with the testbench reference value TbY, and the status signals are correctly assigned based on Y output. We can observe results of regular calculation, sign extension, and zero result.

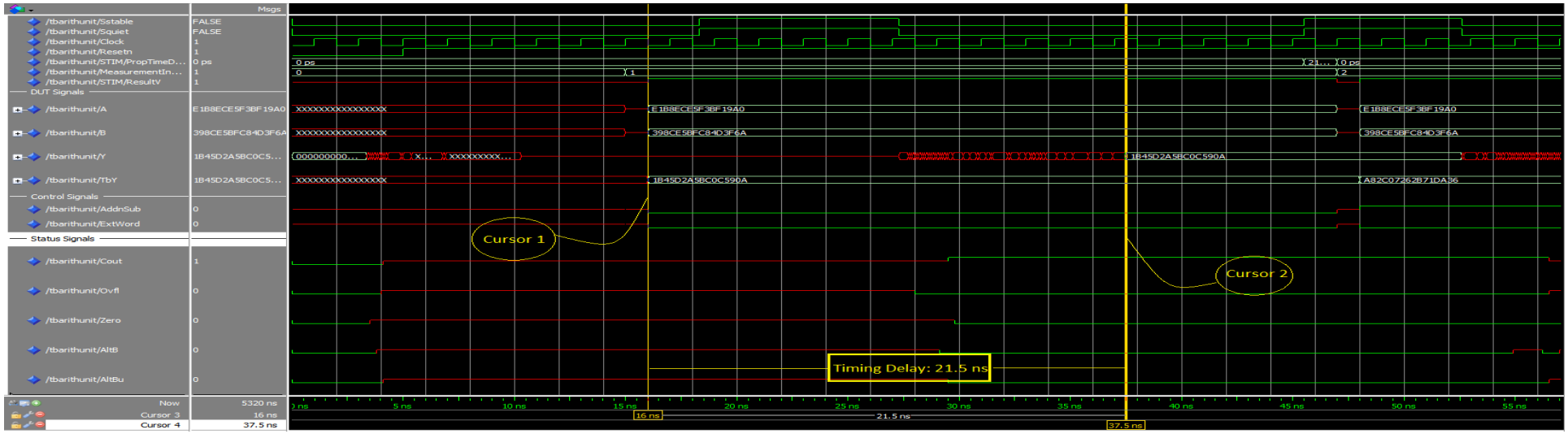

Figure 27: waveTAU1, from t = 0 to a few ns into the beginning of measurement #2. The waveTAU1 waveform pattern shows the propagation delay of the first calculation at 21.5 ns from when the X and Y inputs are valid to when Y output becomes valid.

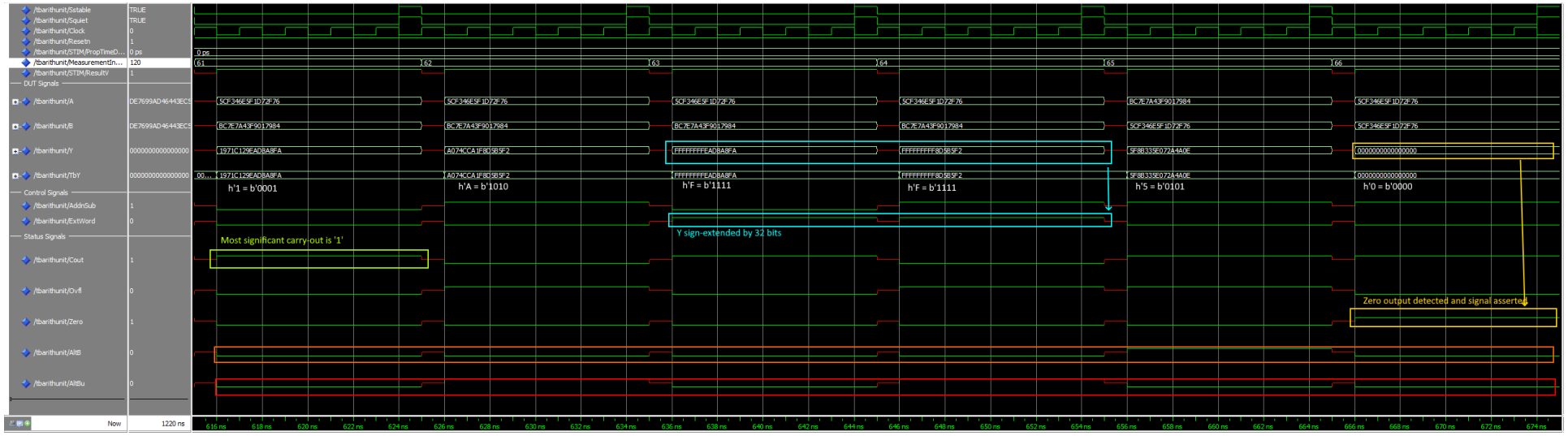

Figure 28: waveFAU2, from t = 615 to 675 ns. The waveFAU2 waveform pattern shows another 6 calculations, and again in all cases the Y output match with the testbench reference value TbY, and the status signals are correctly assigned based on Y output. We can observe results of regular calculation, negative sign extension, and zero result.

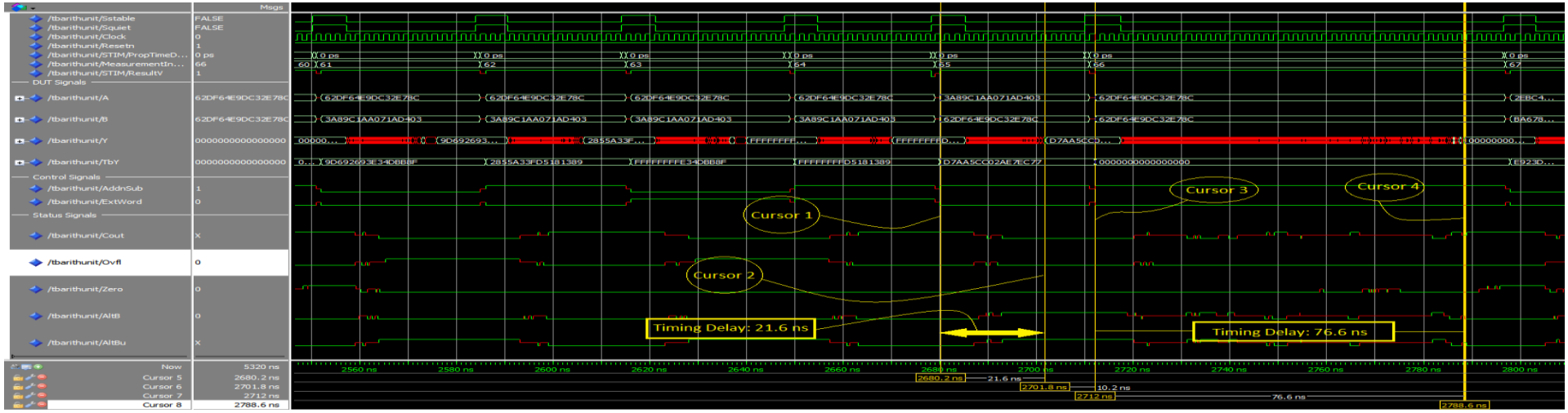

Figure 29: waveTAU2, from beginning of measurement #61 to end of measurement #66. The waveTAU2 waveform pattern shows the propagation delay of the 65th calculation at 21.6 ns from when the X and Y inputs are valid to when Y output becomes valid, as well as the propagation delay of the 66th calculation at 76.6 ns from when the X and Y inputs are valid to when Y output becomes valid. As the 66th calculation results in a zero output, the Zero status signal can only be set after running through the full 64-bit NOR-gate, which significantly delays the output of Y as compared to a non-zero Y result.

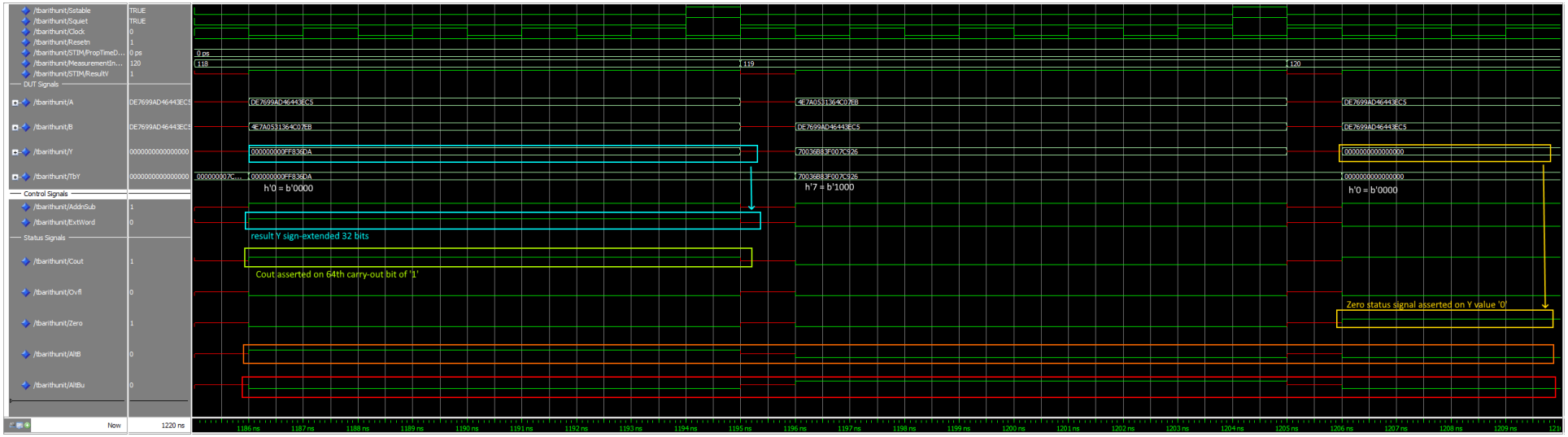

Figure 30: waveFAU3, from t = 1185 to 1210 ns. The waveFAU2 waveform pattern shows the last 3 calculations of the verification test, and again in all cases the Y output match with the testbench reference value TbY, and the status signals are correctly assigned based on Y output. We can observe results of regular calculation, positive sign extension, and zero result.

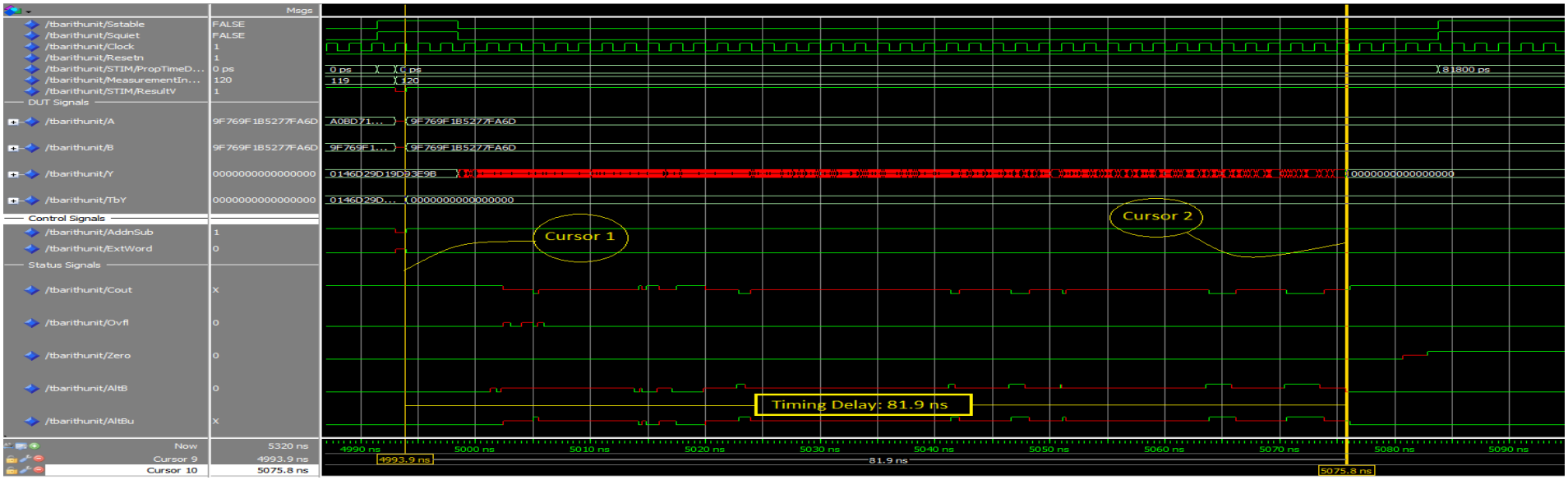

Figure 31: waveTAU3, from beginning of measurement #120 till 5 ns after the end of measurement #120. The waveTAU2 waveform pattern shows the propagation delay of the 120th calculation at 21.6 ns from when the X and Y inputs are valid to when Y output becomes valid. As the 120th calculation results in a zero output, the Zero status signal can only be set after running through the full 64-bit NOR-gate, which significantly delays the output of Y as compared to a non-zero Y result.

Shift Unit:

Shift-Left Logical 32-bit Functional Simulation Waves:

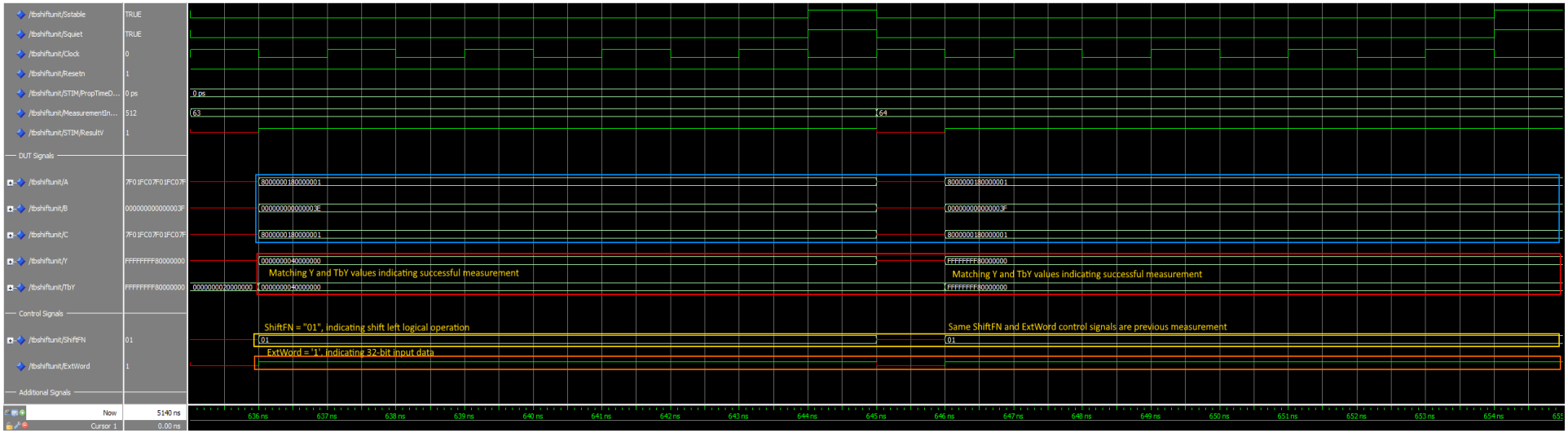

Figure 32: Image indicating simulation waves for measurements 64 and 65 for the *SLL32Unit00.tvs* test vector. Operands are labelled in blue, circuit and testbench output is labelled in red, the *ShiftFN* control signal is labelled in yellow, and the *ExtWord* control signal is labelled in orange.

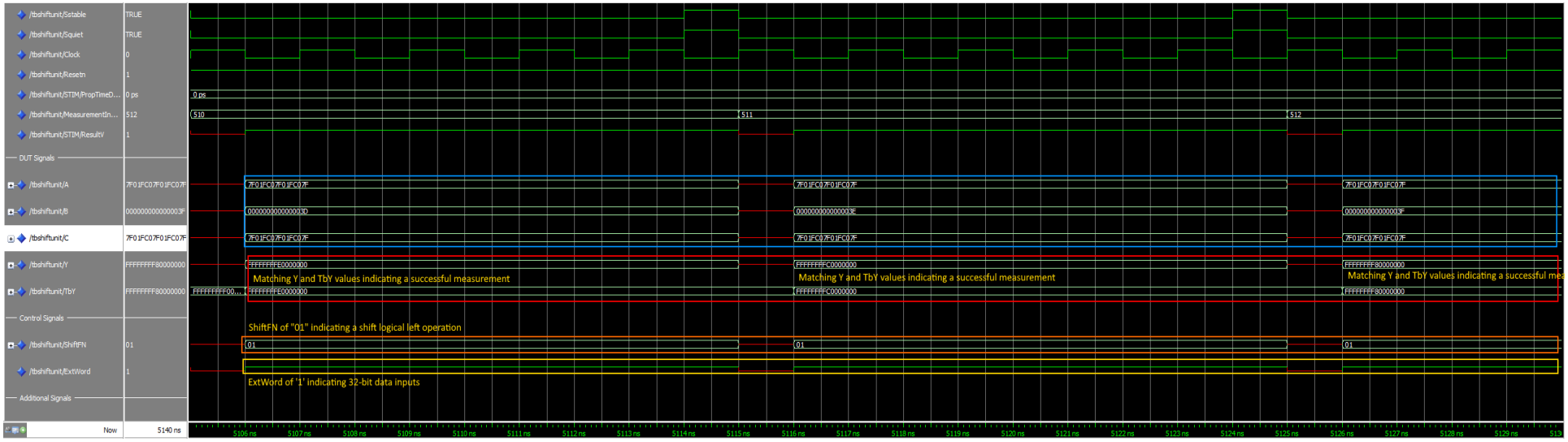

Figure 33: Image indicating simulation waves for measurements 64 and 65 for the *SLL32Unit00.tvs* test vector. Operands are labelled in blue, circuit and testbench output is labelled in red, the *ShiftFN* control signal is labelled in orange, and the *ExtWord* control signal is labelled in yellow.

Shift-Left Logical 32-bit Timing Simulation Waves:

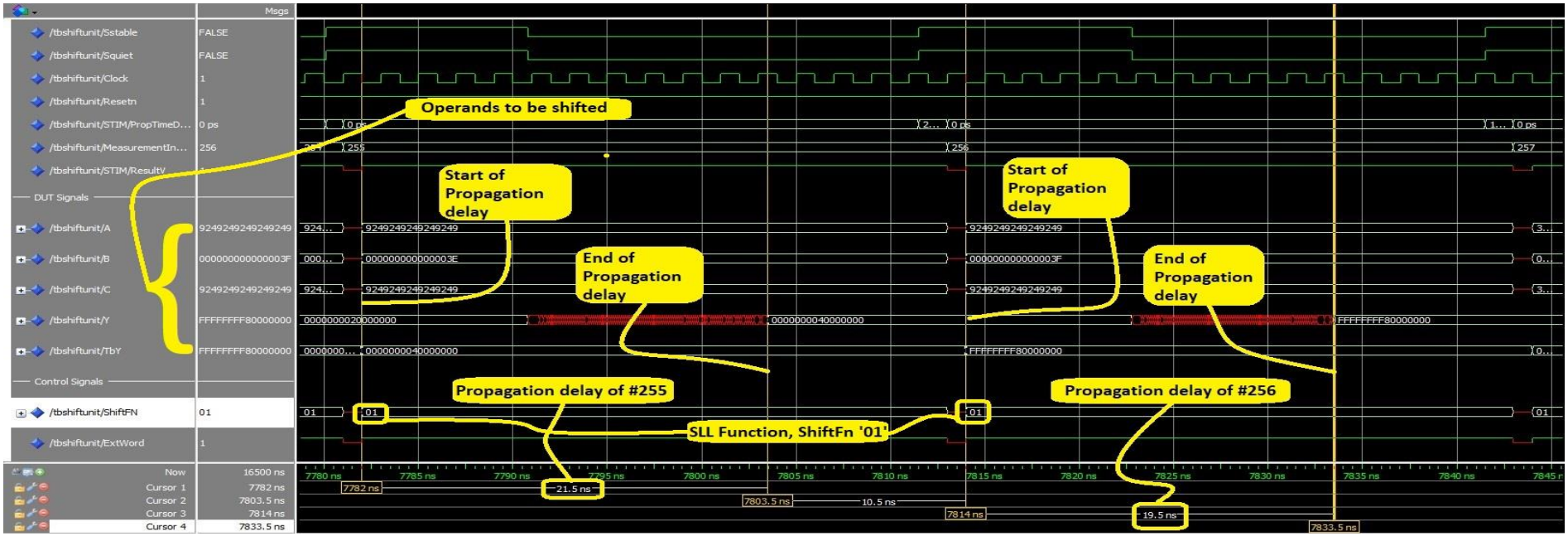

Figure 34: Image depicting the propagation delays for measurement 255, t_{pd} = 21.5 ns, and measurement 256, t_{pd} = 19.5 ns.

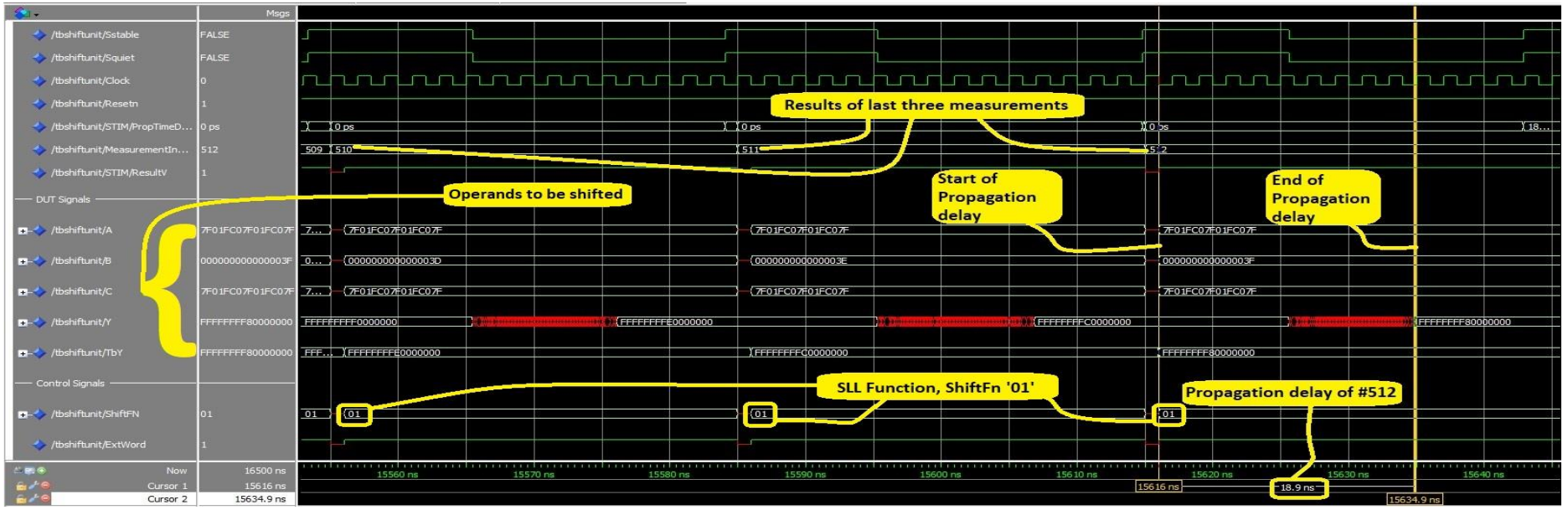

Figure 35: Image depicting the propagation delay of measurements 510, 511, and 512, where measurement 512 has a propagation delay of t_{pd} = 18.9 ns.

Shift-Right Logical 32-bit Functional Simulation Waves:

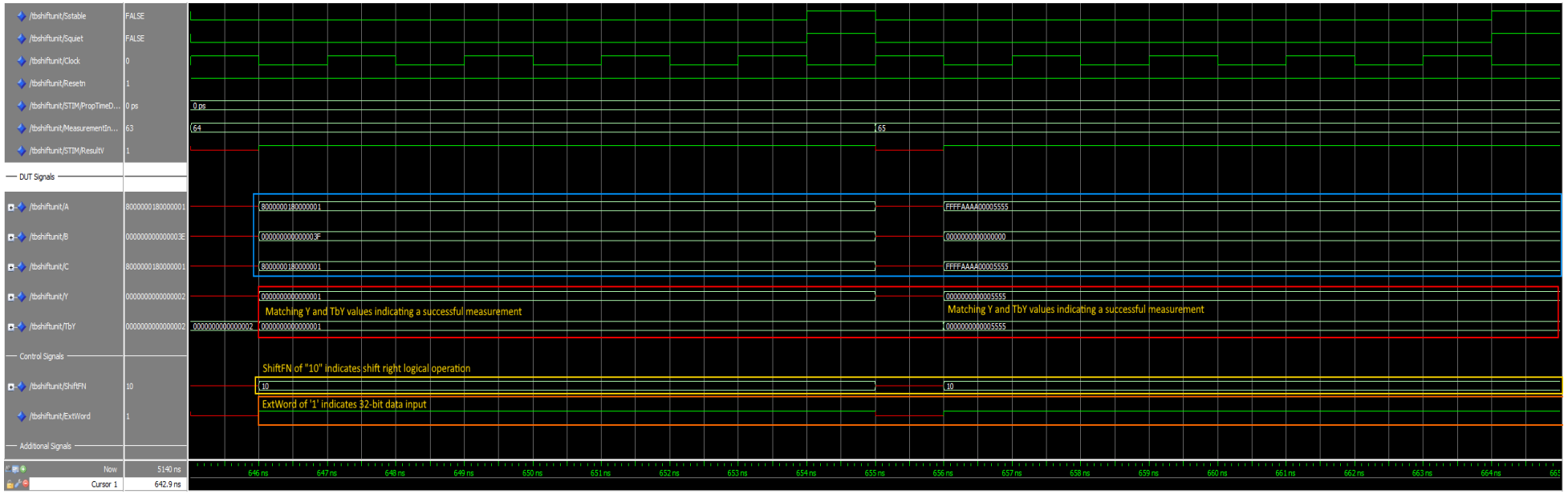

Figure 36: Image indicating simulation waves for measurements 64 and 65 for the *SRL32Unit00.tvs* test vector. Operands are labelled in blue, circuit and testbench output is labelled in red, the *ShiftFN* control signal is labelled in yellow, and the *ExtWord* control signal is labelled in orange.

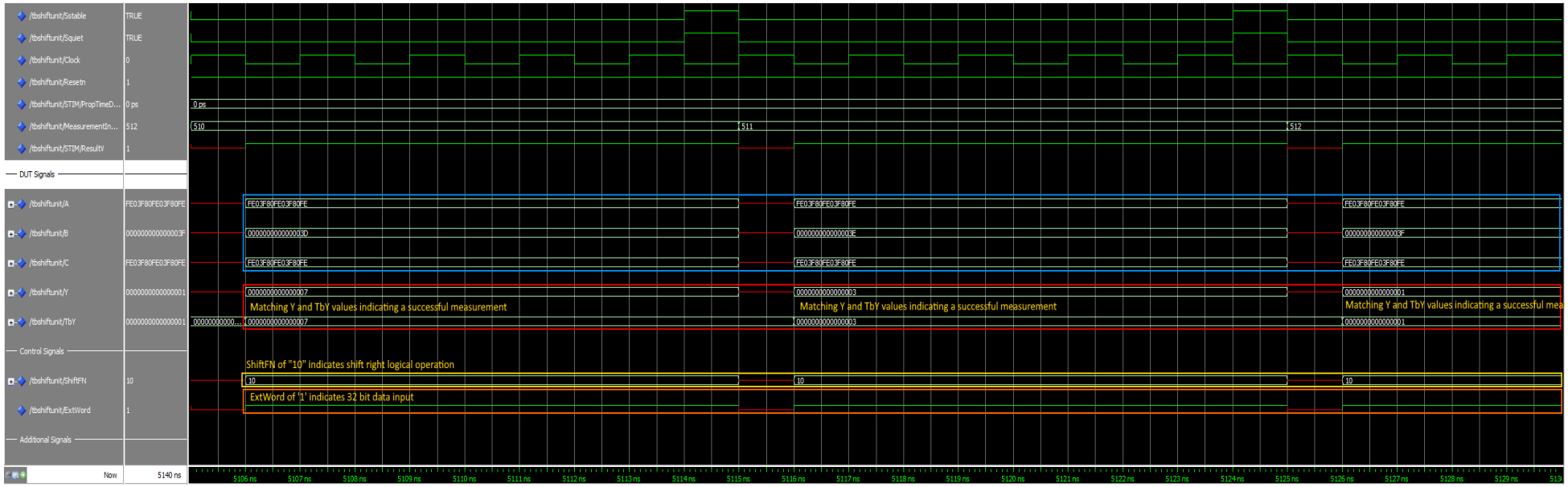

Figure 37: Image indicating simulation waves for measurements 126 to 128 for the *SRL32Unit00.tvs* test vector. Operands are labelled in blue, circuit and testbench output is labelled in red, the *ShiftFN* control signal is labelled in yellow, and the *ExtWord* control signal is labelled in orange.

Shift-Right Logical 32-bit Timing Simulation Waves:

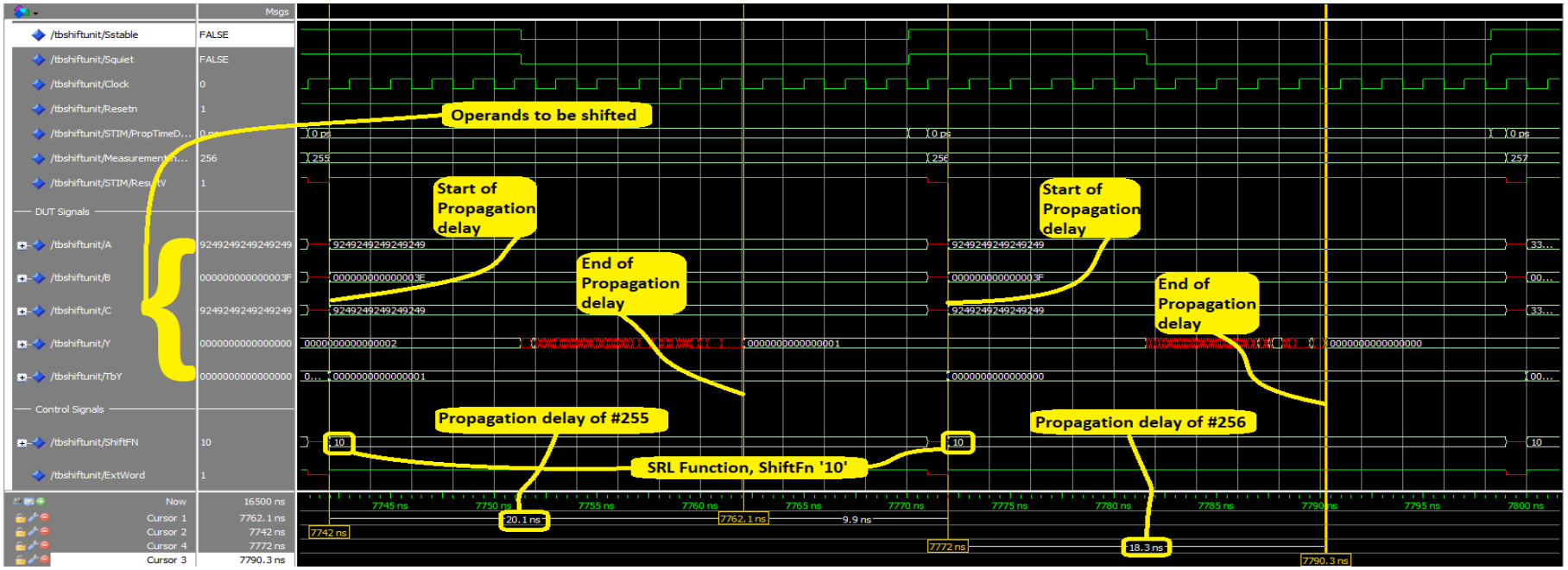

Figure 38: Image depicting the propagation delays for measurement 255, t_{pd} = 20.1 ns, and measurement 256, t_{pd} = 18.3 ns.

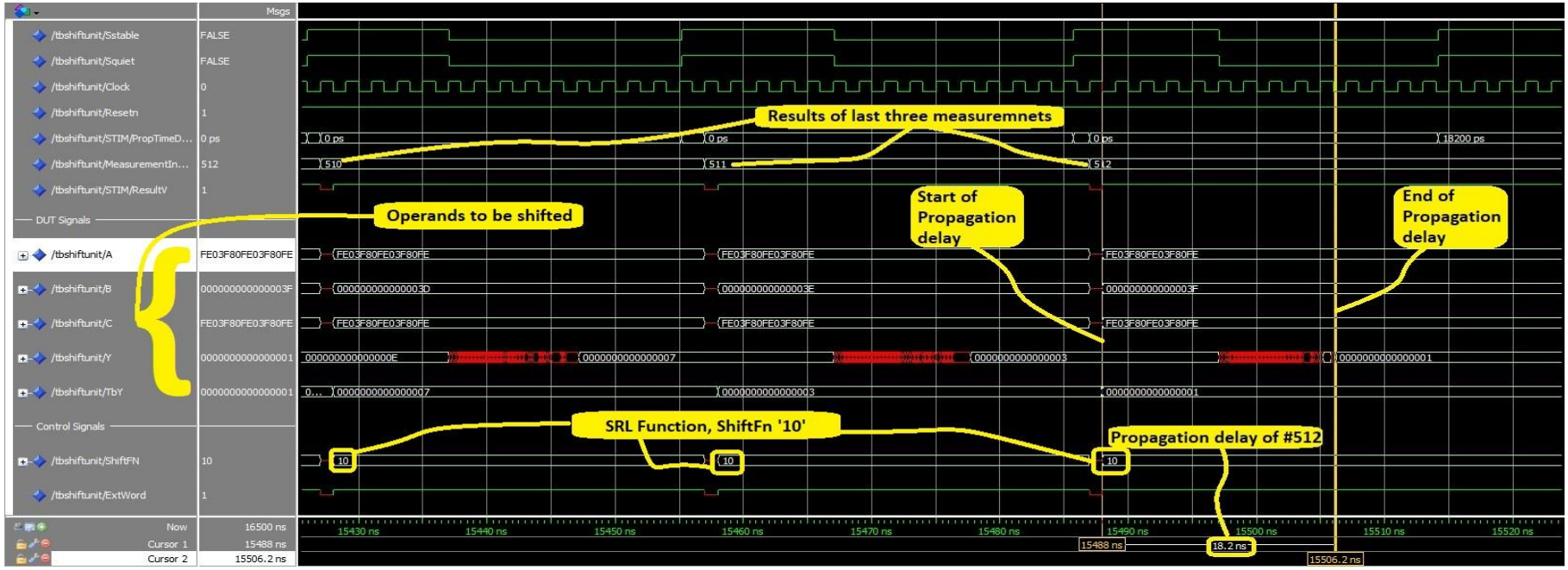

Figure 39: Image depicting the propagation delay of measurements 510, 511, and 512, where measurement 512 has a propagation delay of t_{pd} = 18.2 ns.

Shift-Right Arithmetic 32-bit Functional Simulation Waves:

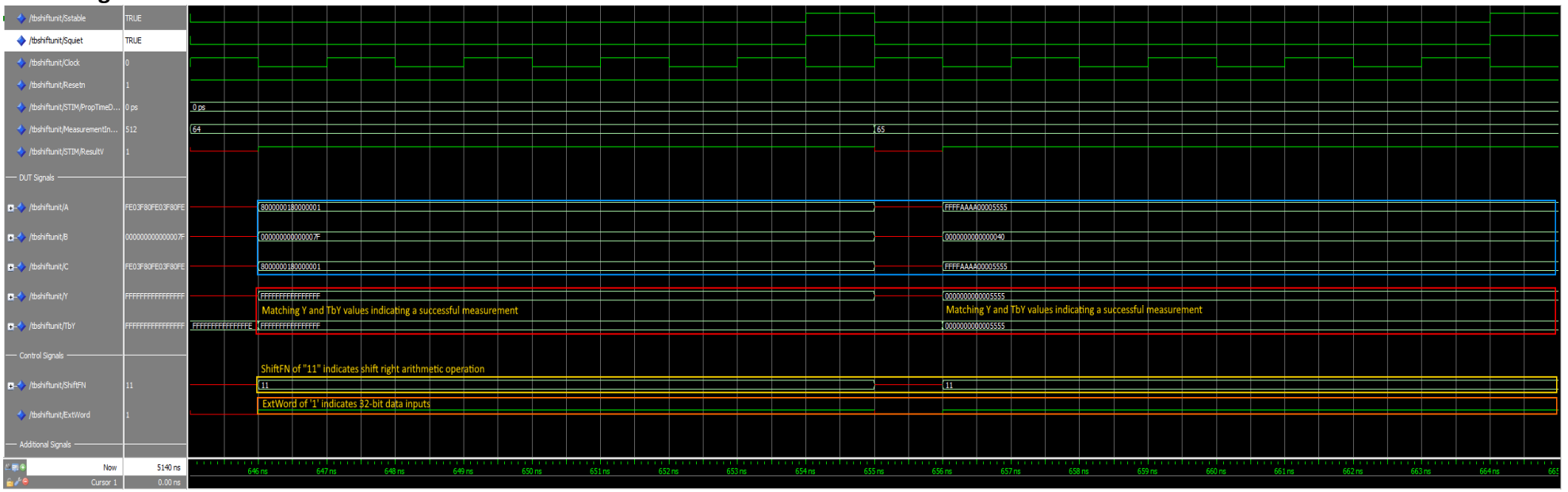

Figure 40: Image indicating simulation waves for measurements 64 and 65 for the *SRA32Unit00.tvs* test vector. Operands are labelled in blue, circuit and testbench output is labelled in red, the *ShiftFN* control signal is labelled in yellow, and the *ExtWord* control signal is labelled in orange.

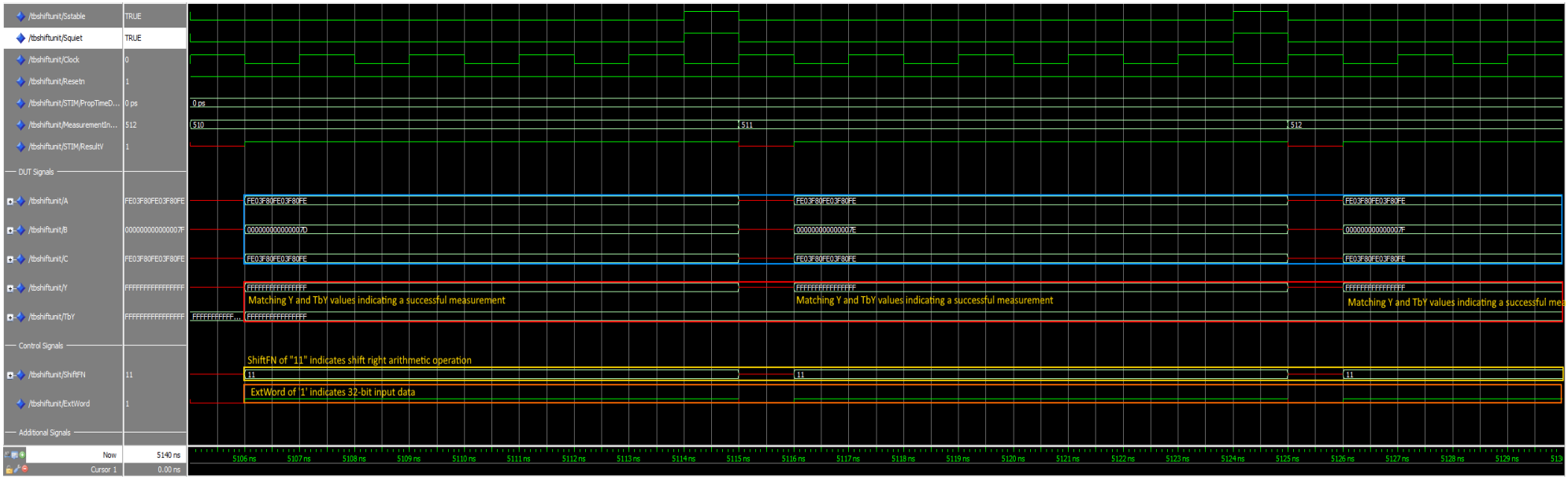

Figure 41: Image indicating simulation waves for measurements 126 to 128 for the *SRA32Unit00.tvs* test vector. Operands are labelled in blue, circuit and testbench output is labelled in red, the *ShiftFN* control signal is labelled in yellow, and the *ExtWord* control signal is labelled in orange.

Shift-Right Arithmetic 32-bit Functional Simulation Waves:

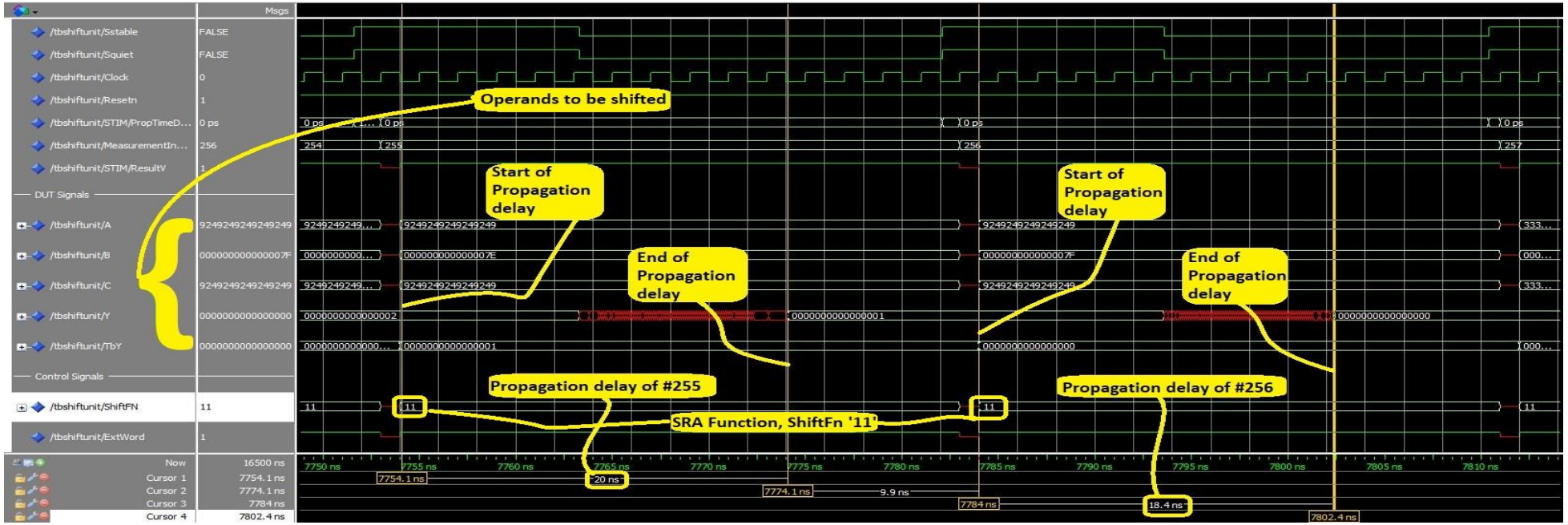

Figure 42: Image depicting the propagation delays for measurement 255, t_{pd} = 20 ns, and measurement 256, t_{pd} = 18.4 ns.

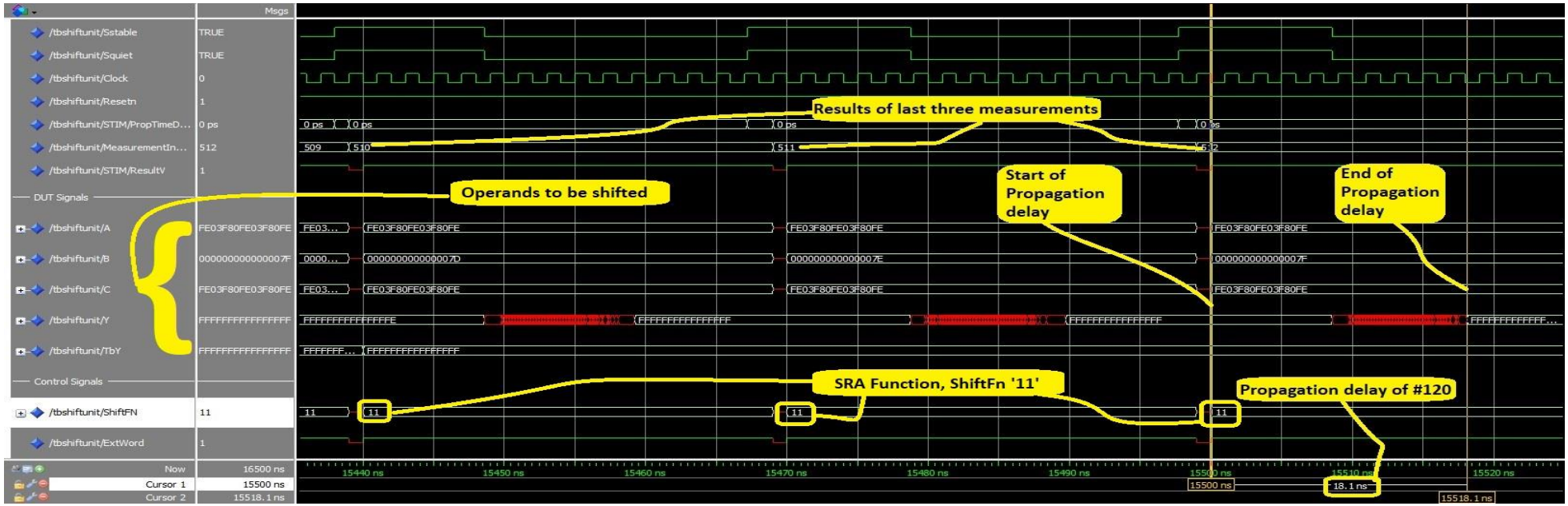

Figure 43: Image depicting the propagation delay of measurements 510, 511, and 512, where measurement 512 has a propagation delay of t_{pd} = 18.1 ns.

Shift Left-Logical 64-bit Functional Simulation Waves:

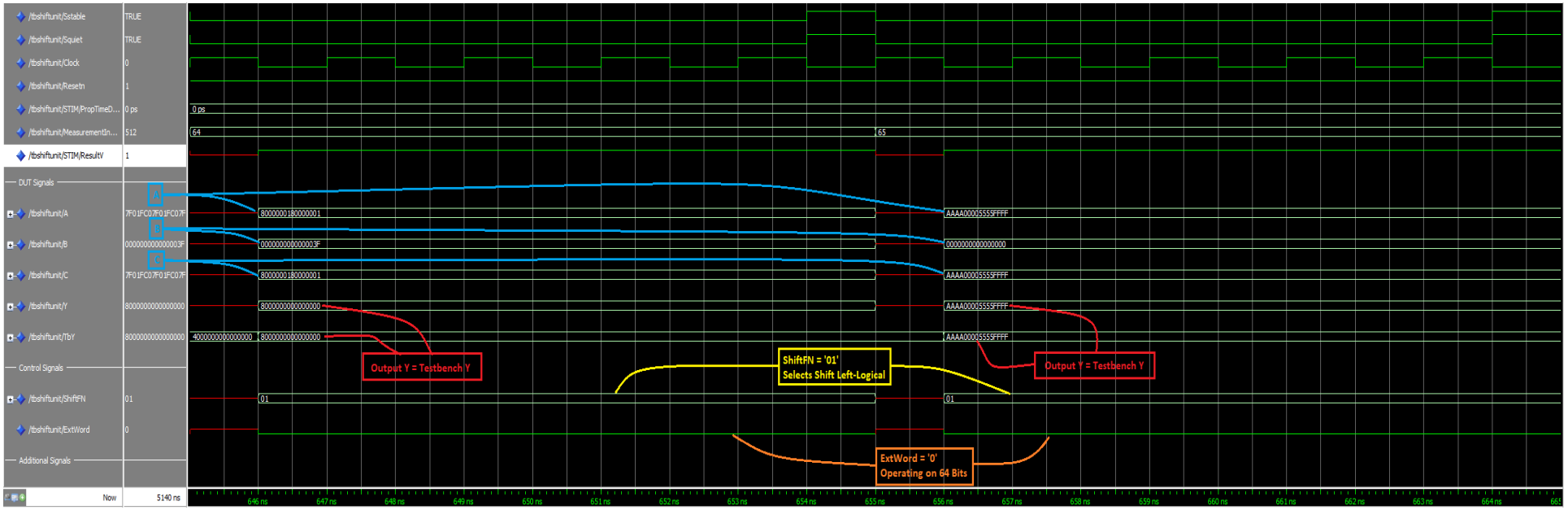

Figure 44: Image indicating simulation waves for measurements 64 and 65 for the *SLL64Unit00.tvs* test vector. Operands are labelled in blue, circuit and testbench output is labelled in red, the *ShiftFN* control signal is labelled in yellow, and the *ExtWord* control signal is labelled in orange.

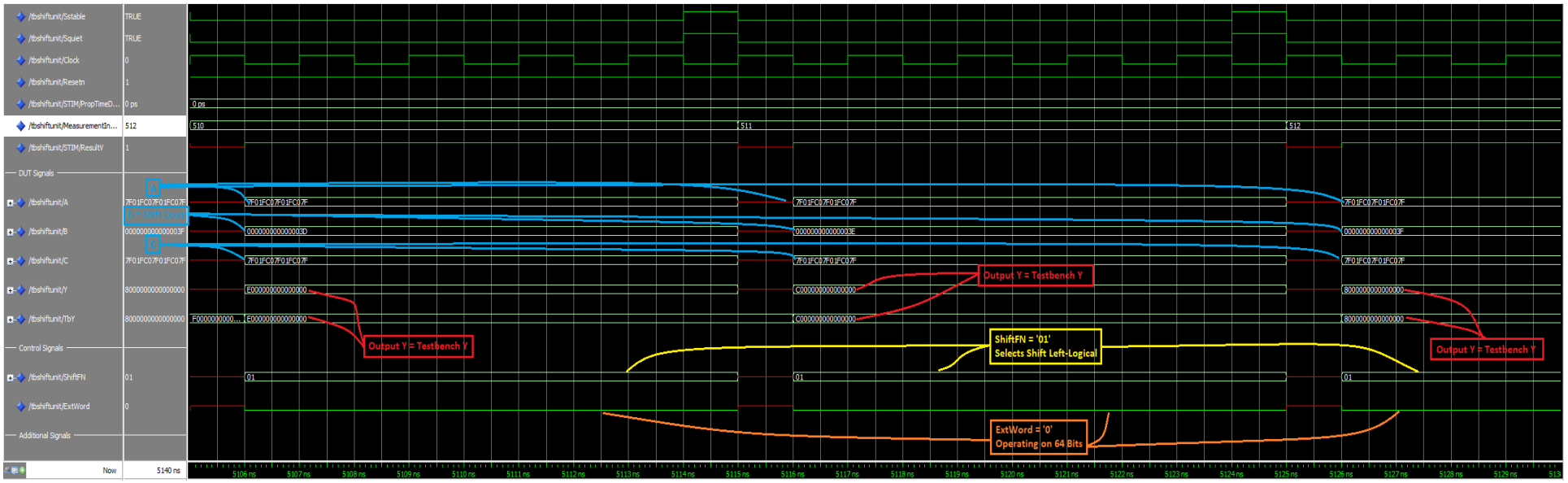

Figure 45: Image indicating simulation waves for measurements 510, 511, and 512. Operands are labelled in blue, circuit and testbench output is labelled in red, the *ShiftFN* control signal is labelled in yellow, and the *ExtWord* control signal is labelled in orange.

Shift Left-Logical 64-bit Timing Simulation Waves:

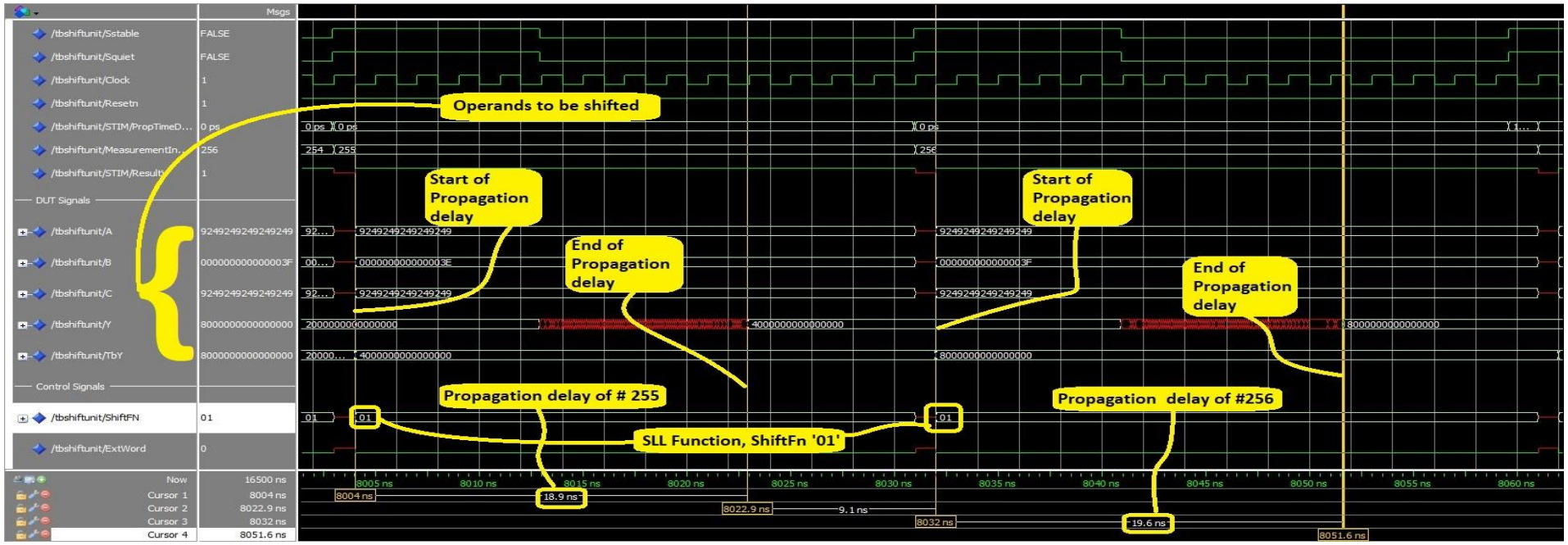

Figure 46: Image depicting the propagation delays for measurement 255, t_{pd} = 18.9 ns, and measurement 256, t_{pd} = 19.6 ns.

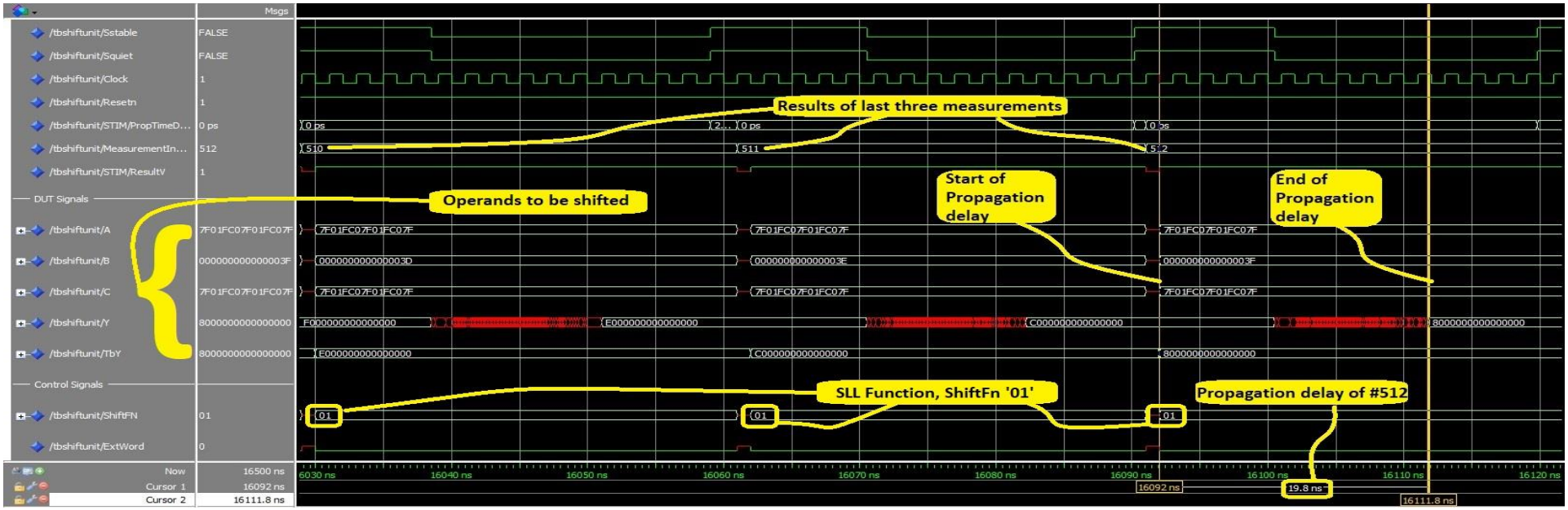

Figure 47: Image depicting the propagation delay of measurements 510, 511, and 512, where measurement 512 has a propagation delay of t_{pd} = 19.8 ns.

Shift Right-Logical 64-bit Functional Simulation Waves:

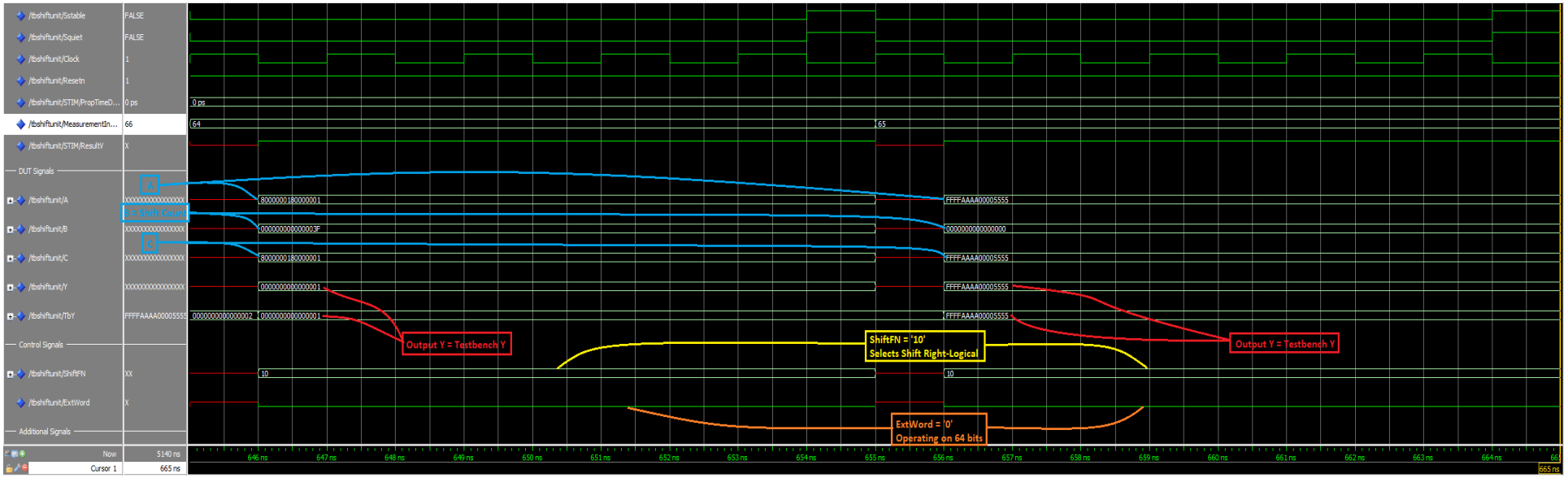

Figure 48: Image indicating simulation waves for measurements 64 and 65 for the *SRL64Unit00.tvs* test vector. Operands are labelled in blue, circuit and testbench output is labelled in red, the *ShiftFN* control signal is labelled in yellow, and the *ExtWord* control signal is labelled in orange.

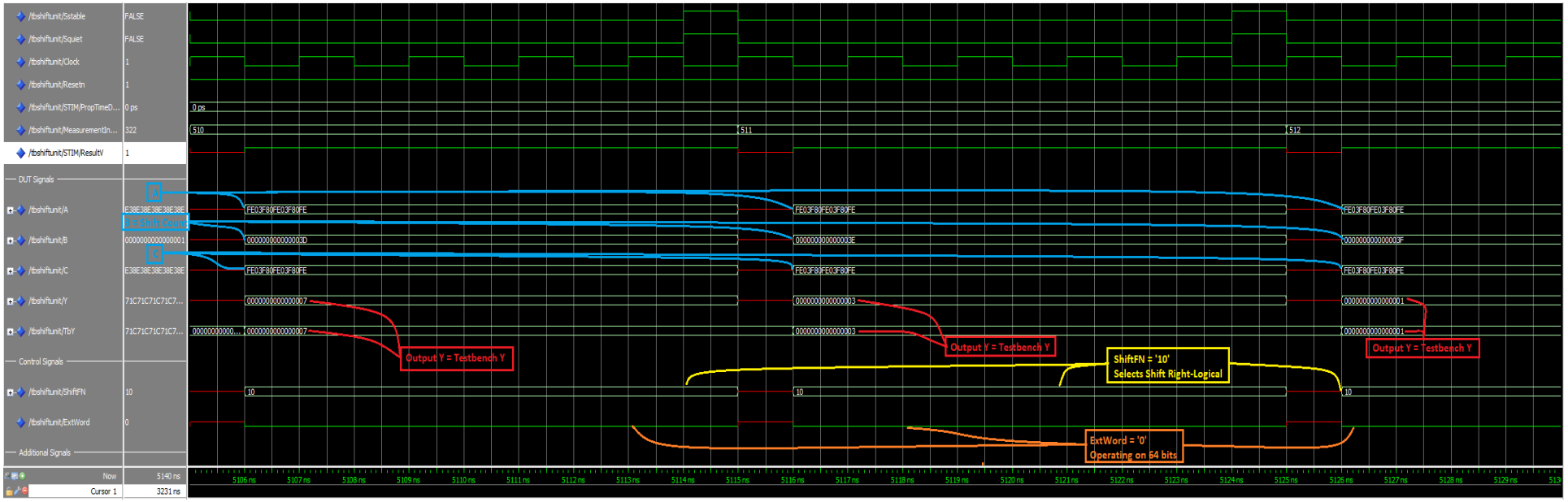

Figure 49: Image indicating simulation waves for measurements 510, 511, and 512. Operands are labelled in blue, circuit and testbench output is labelled in red, the *ShiftFN* control signal is labelled in yellow, and the *ExtWord* control signal is labelled in orange.

Shift Right-Logical 64-bit Timing Simulation Waves:

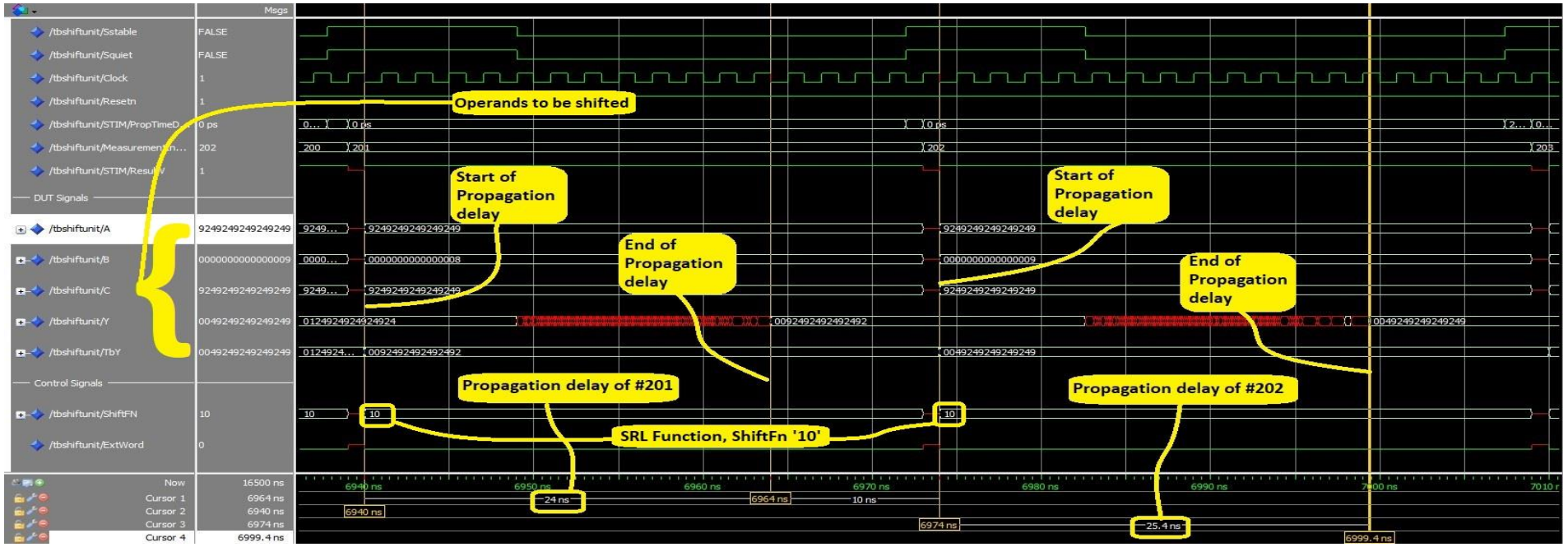

Figure 50: Image depicting the propagation delays for measurement 201, t_{pd} = 24 ns, and measurement 202, t_{pd} = 24 ns.

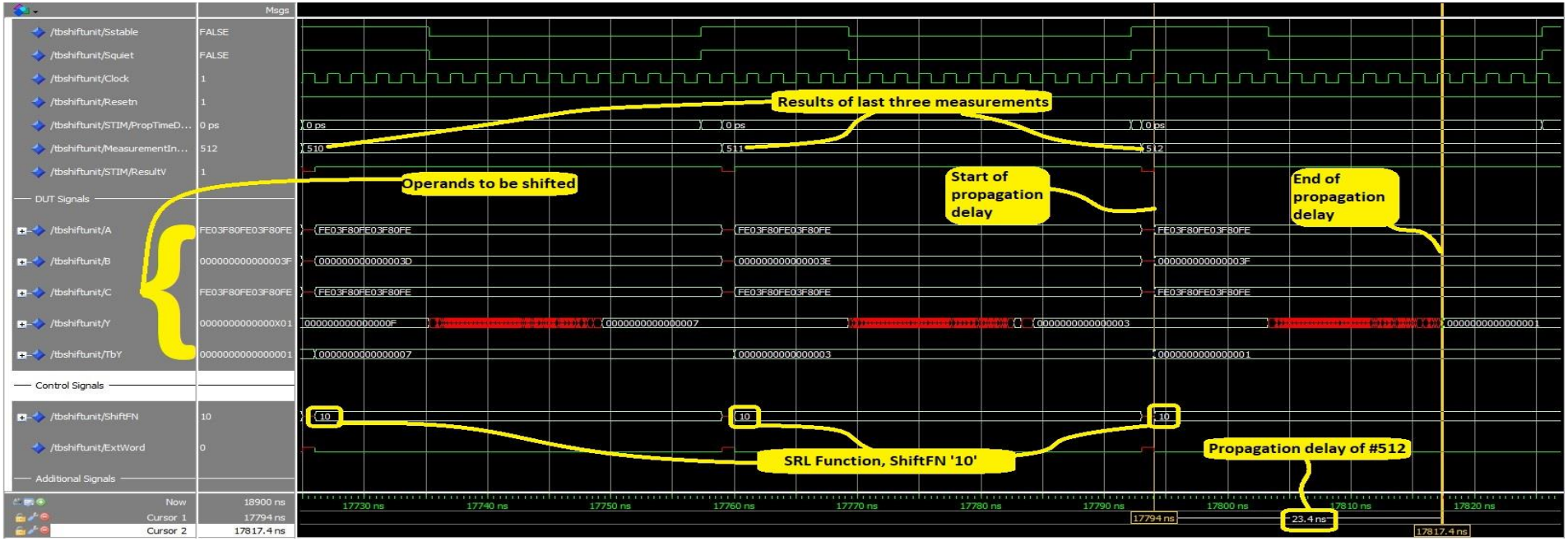

Figure 51: Image depicting the propagation delay of measurements 510, 511, and 512, where measurement 512 has a propagation delay of t_{pd} = 23.4 ns.

Shift Right-Arithmetic 64-bit Functional Simulation Waves:

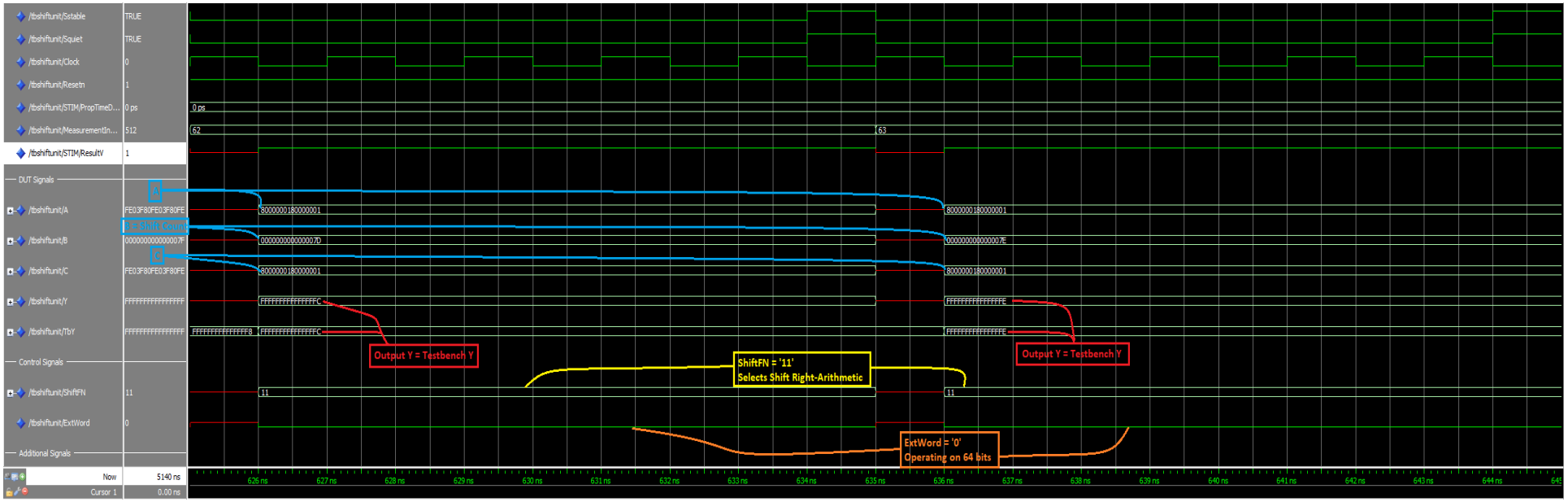

Figure 52: Image indicating simulation waves for measurements 64 and 65 for the SRA64Unit00.tvs test vector. Operands are labelled in blue, circuit and testbench output is labelled in red, the ShiftFN control signal is labelled in yellow, and the ExtWord control signal is labelled in orange.

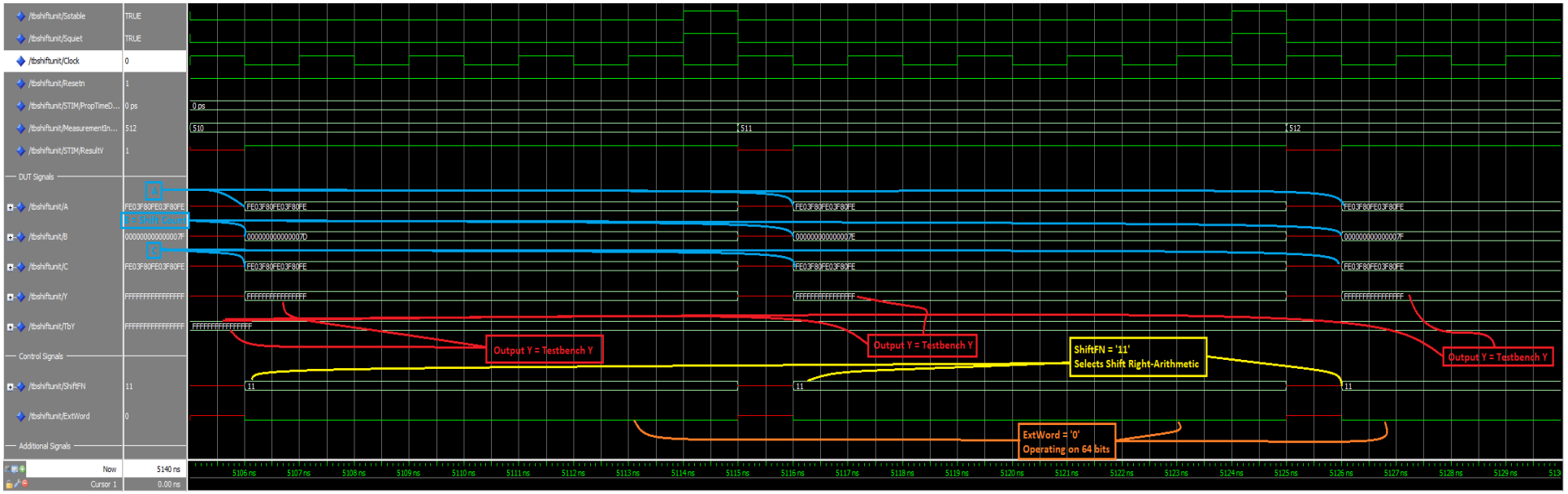

Figure 53: Image indicating simulation waves for measurements 510, 511, and 512. Operands are labelled in blue, circuit and testbench output is labelled in red, the *ShiftFN* control signal is labelled in yellow, and the *ExtWord* control signal is labelled in orange.

Shift Right-Arithmetic 64-bit Timing Simulation Waves:

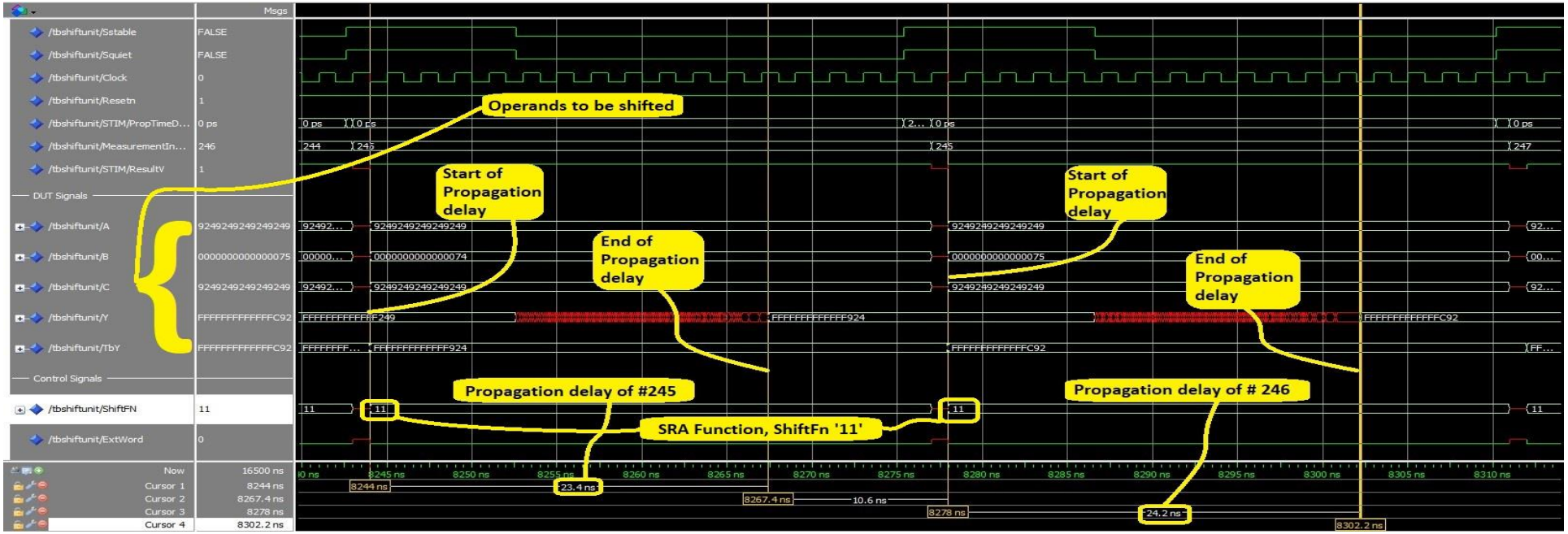

Figure 54: Image depicting the propagation delays for measurement 245, t_{pd} = 23.4 ns, and measurement 246, t_{pd} = 24.2 ns.

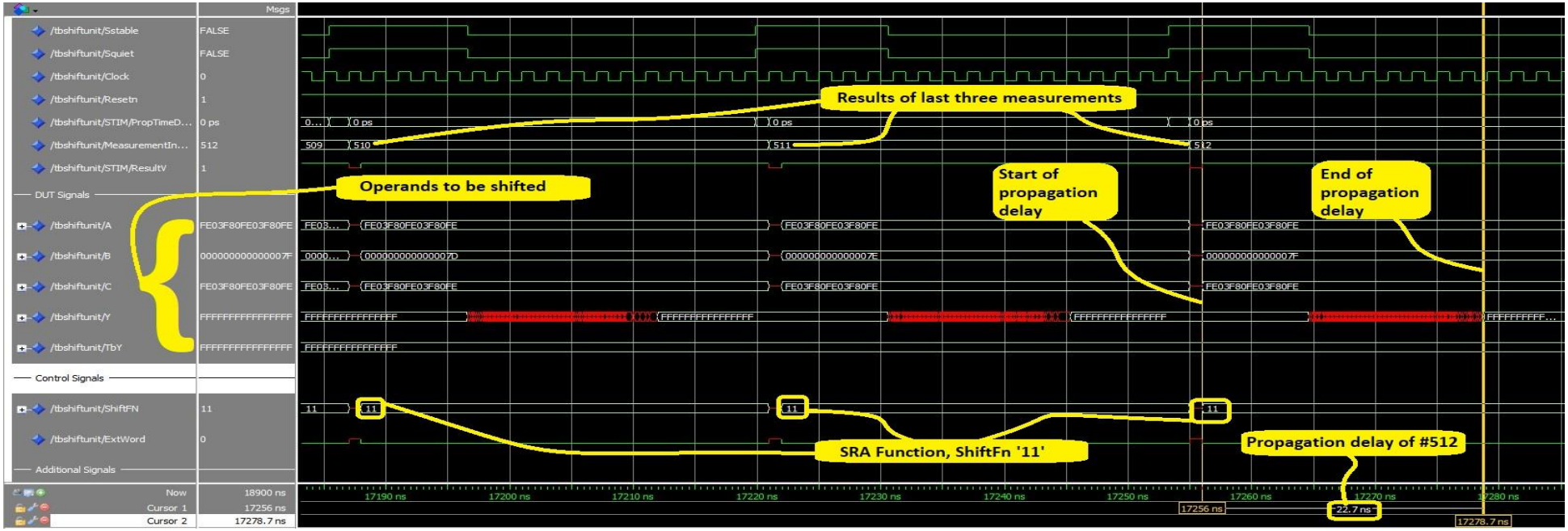

Figure 55: Image depicting the propagation delay of measurements 510, 511, and 512, where measurement 512 has a propagation delay of t_{pd} = 22.7 ns.

Execution Unit:

Functional Verification:

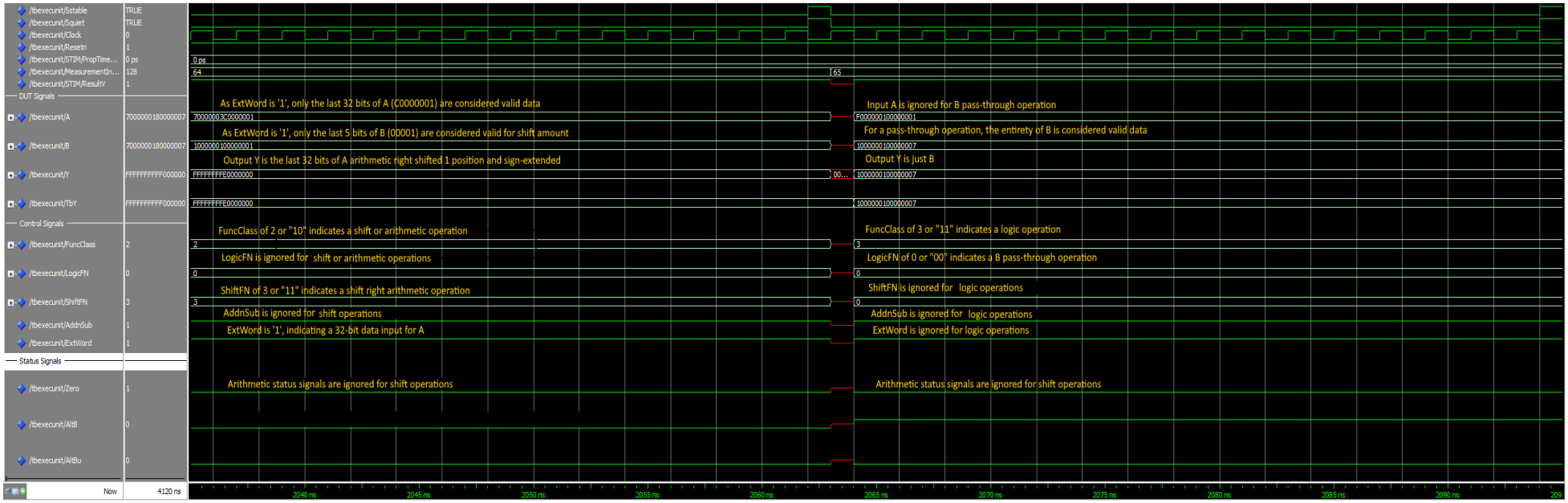

Figure 56: Image indicating functional simulation waves for measurements 64 and 65 for the ExexUnit00.tvs test vector

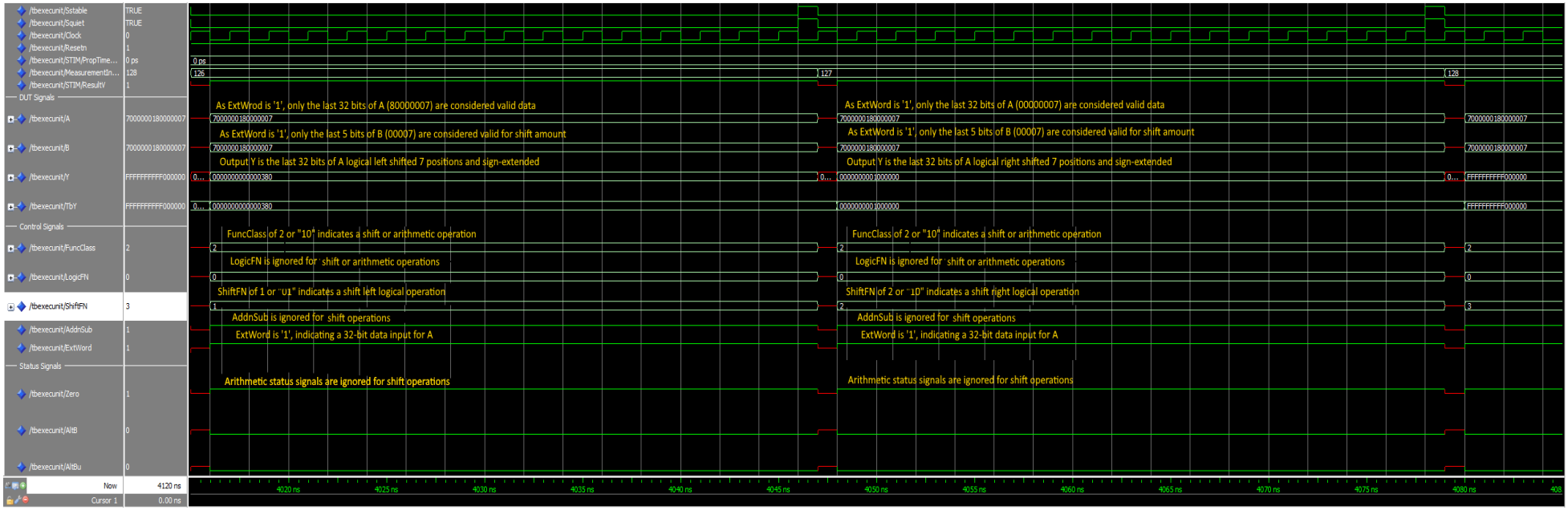

Figure 57: Image indicating functional simulation waves for measurements 126 to 128 for the ExexUnit00.tvs test vector

Timing Simulation:

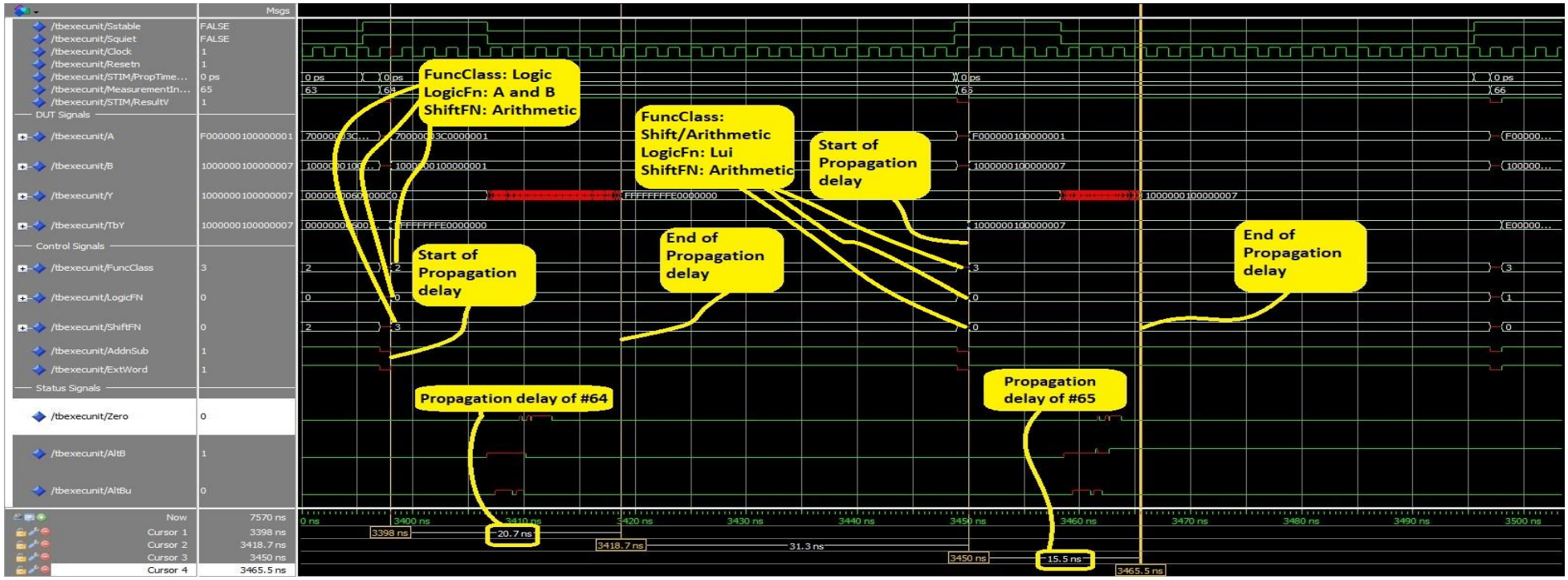

Figure 58: Image depicting the propagation delays for measurement 64, t_{pd} = 20.7 ns, and measurement 65, t_{pd} = 15.5 ns.

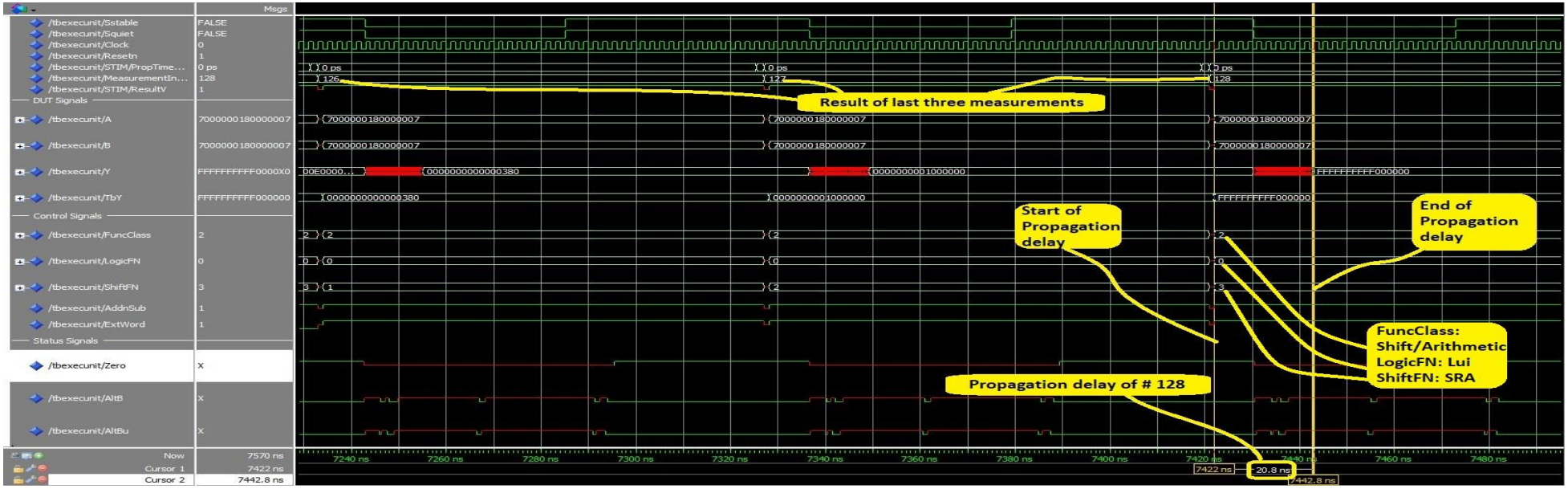

Figure 59: Image depicting the propagation delay of measurements 126, 127, and 128, where measurement 128 has a propagation delay of t_{pd} = 20.8 ns.

Simulation Results

Logic Unit

Functional Verification

To perform functional verification of the logic unit entity, we use the *LogicUnit00.tvs* test vector which supplies two 64-bit input values, a value for the *LogicFn* control signal, and the correct corresponding output of performing the specified operation on the two input operands. To verify the functionality of the Logic Unit circuit, we can observe wave functions for each of the 772 measurements provided by the test vector. However, this can create very tedious work; we are able to automate this process by using the provided testbench, *TBLogicUnit.vhd.* Upon performing functional simulation, we are able to conclude that our Logic Unit circuit works correctly by viewing that no errors exist in the transcript file, *FuncLogicUnitTranscript.txt*. However, it is still important to view the corresponding functional simulation waves. Three waveforms have been included in the report; one from the beginning of the simulation, one near the middle of the simulation, and one at the end of the simulation. The input values at the beginning of the simulation, indicated in figure 20, are all zero values; therefore, observing the results of these measurements does not yield any useful results as all operations would appear the same. However, the functional simulation wave in figure 22**,** yields more practical comparisons. We are able to view four measurements; a measurement each for *XOR*, *OR*, *AND*, and *B passthrough*, selected by the *LogicFn* control signal values specified by the test vector. The Logic Unit's output signal,*Y*, will show values of *Y = A XOR B, Y = A OR B, Y = A AND B,* and, *Y = B*, respectively. Consequently, we are able to compare the output signal with the correct output provided by the test vector, *TbY*; by ensuring that these values are the same, we can verify the functionality of our circuit.

Timing Verification

By performing timing verification, propagation delays that arise with the logic unit circuitry can be observed and compared with the results expected from theoretical analysis. Firstly, a 6-level gate delay is expected when a *XOR, OR,* or *AND*, operation is performed. Therefore, since this is only a 6-level gate delay, we would expect a relatively small propagation delay compared to the other entities contained in the execution unit. A propagation delay of t_{pd} = 13.9ns is observed in both figure 21 and figure 25; in both figures, two different logic unit operations are being performed while possessing the same propagation delay. Consequently, since the results from the timing simulations correspond to expected results from theoretical analysis, it can be concluded that the logic unit is functioning correctly.

Arithmetic Unit

Functional Verification

To perform functional verification of the arithmetic unit entity, we use the *ArithmeticUnit00.tvs* test vector which supplies two 64-bit input values *A* and *B*, a value for *AddnSub* and *ExtWord* control signals, and the correct corresponding output *Y* of performing the specified operation on the two input operands. To verify the functionality of the Logic Unit circuit, we can observe wave functions for each of the 120 measurements provided by the test vector. However, this can create very tedious work; we are able to automate this process by using the provided testbench, *TBArithUnit.vhd.* Upon performing functional simulation, we are able to conclude that our Logic Unit circuit works correctly by viewing that no errors exist in the transcript file, *FuncArithUnitTranscript.txt*. However, it is still important to view the corresponding functional simulation waves. Three waveforms have been included in the report; one from the beginning of the simulation, one near the middle of the simulation, and one at the end of the simulation. In all three waveforms, various combinations of the control signals *AddnSub* and *ExtWord* are shown in order to get a complete picture of all possible cases of positive/negative numbers and 32/64 bit data. Consequently, we are able to compare the output signal with the correct output provided by the test vector, *TbY*; by ensuring that these values are the same, we can verify the functionality of our circuit.

Timing Verification

By performing timing verification, propagation delays that arise with the arithmetic unit circuitry can be observed and compared with the results expected from theoretical analysis. In the first and second timing simulation waveforms, we see a propagation delay of about t_{pd} = 37.5ns for measurements where the data inputs A and B are not identical. In the second and third timing simulation waveforms, we see that in measurements where A and B are identical, the propagation delay is significantly higher, at t_{pd} = 76.6ns for measurement 66 and propagation delay of t_{pd} = 81.9ns for measurement 120. As the carry network used in the Arithmetic Unit is a simple ripple-carry network with zero skipping wiring, cases like this where all bits must be evaluated using the previous carry-out bit is understandably slow. Consequently, since the results from the timing simulations correspond to expected results from theoretical analysis, it can be concluded that the arithmetic unit is functioning correctly.

Shift Unit

Functional Verification:

When verifying the functionality of the Shift Unit, it was important to note that the shift unit works on 64-bit values as well as 32-bit values. Due to the portability of the Shift Unit, verifying functionality verifies the functionality of a series of features: *SwapWord*, *Sign-Extension Upper, Sign-Extension Lower, Shift-Right Logical 64-bit and 32-bit, Shift-Left Logical 64-bit and 32-bit, and Shift-Right Arithmetic 64-bit and 32-bit.* Therefore, to test so many features, a series of test vectors are required: *SLL32Unit00.tvs, SLL64Unit00.tvs, SRL32Unit00.tvs, SRL64Unit00.tvs, SRA32Unit00.tvs,* and *SRA64Unit00.tvs*. Each test vector supplies three 64-bit input operands, two which are operated upon, and one which specifies the shift-count. Moreover, the test vectors also provide the corresponding control signals to ensure that the desired operation is performed, and also the correct output after performing the operation is supplied. Similar to functional verification performed on the logic and arithmetic units, we are able to automate this process using a testbench which can test all six of the provided test vectors to ensure correct functionality of the shift unit. However, viewing the simulation waves provides some insight as well. For example, we are able to verify the functionality of the shift left-logical, shift-right-logical, and shift-right arithmetic 64-bit barrel shifters by observing figure 44, 46 and 48 respectively. In figure 45, a left-logical shift of 63-bits can be observed; the lower bits are all filled with zero values, while the upper bits that are shifted left and discarded. By observing these figures, we are able to verify the functionality of the 64-bit barrel shifter. However, as mentioned previously, the shift unit provides 32-bit functionality as well. In figure 36 a 30-bit shift right-logical operation can be observed. The input operand is shifted right by 32-bits and upper bits are filled with zeros. Consequently, it is concluded that the shift unit displays correct-functioning behaviour.

Timing Verification

Similar to functional verification, timing verification ensures correct implementation of a series of features due to the portability of the Shift Unit. To verify that the shift unit is functioning as expected, timing simulation results can be compared to theoretical expectations of propagation delays in output results. Firstly, it is noted that the shift unit has more gate delays than the logic unit, but fewer gates than the arithmetic unit, due to the ripple-adder that the arithmetic unit possesses. Therefore, a propagation delay between the logic units, t_{pd} = 13.9ns, and the arithmetic units, t_{pd} = 81.9ns, is expected, with the delay being closer to that of the logic unit due to the vast number of gates possessed by the ripple adder. For example, in figures 55, and 51 propagation delays of t_{pd} = 22.7ns and t_{pd} = 23.4ns for shift right-logical 64-bit and shift right-arithmetic 64-bit are observed, respectively. Consequently, it is concluded that timing for the 64-bit shift operations is similar to the results expected by theoretical analysis. Furthermore, the same method of timing verification is extended to the 32-bit operations. Shift right-logical 32-bit and shift right-arithmetic 32-bit timing simulation results can be viewed in figure 39 and 43, with propagation delays of t_{pd} = 18.2ns and t_{pd} = 18.1ns respectively; similarly, these values fall between the propagation delays of the logic unit and the arithmetic unit. Consequently, it is concluded that timing simulations for 32-bit shift operations yield similar results to those induced by theoretical analysis. Thus, we are able to conclude that the timing results of the shift unit are as expected.

Execution Unit

Functional Verification

As the execution unit features all previously seen circuits, it is important to recognize that not all control signals will be used to produce the final output, and not all status signals will be valid output for the chosen operation. To perform functional verification of the execution unit entity, we use the *ExecUnit00.tvs* test vector which supplies two 64-bit input values *A* and *B*, and values for these control signals: *AddnSub*, *NotA*, *ExtWord*, *FuncClass*, *ShiftFN*, *LogicFN*, along with the correct corresponding output *Y* of performing the specified operation on the two input operands. To verify the functionality of the Logic Unit circuit, we can observe wave functions for each of the 128 measurements provided by the test vector. However, this can create very tedious work; we are able to automate this process by using the provided testbench, *TBExecUnit.vhd.* Upon performing functional simulation, we are able to conclude that our Logic Unit circuit works correctly by viewing that no errors exist in the transcript file, *FuncExecUnitTranscript.txt*. However, it is still important to view the corresponding functional simulation waves. Two waveforms have been included in the report: one near the middle of the simulation, and one at the end of the simulation. In the first waveform, we observe a 32-bit SRA operation and a B pass-through operation; in both cases we can conclude that some control signals are irrelevant to the operation dictated by FuncClass, and all of the status signals are invalid since neither operations are arithmetic. In the second waveform we can see measurements on SLL, SRL, SRA operations. Again for all three of those operations, LogicFN is ignored, and the status signals at the bottom of the waveform should be discarded. Since the testbench reported no error during simulation, we are able to compare the output signal with the correct output provided by the test vector, *TbY*; by ensuring that these values are the same, we can thus verify the functionality of our circuit.

Timing Verification

By performing timing verification, propagation delays that arise with the execution unit circuitry can be observed and compared with the results expected from theoretical analysis. In the simulation waveforms we see a logic and two arithmetic operations. The logic operation has a delay of t_{pd} = 20.7ns, while the arithmetic operations have t_{pd} = 15.5ns and t_{pd} = 20.8ns. Comparing the logic operation delay found here to the logic operation delay found for the LogicUnit, we can see an increase due to additional multiplexores placed within the execution unit to direct output selection between the different sub circuits. A similar delay can be seen with the two arithmetic calculations, whereby the propagation delay is a few nanosecond longer that if the measurement was taken in the ShiftUnit test bench instead.Consequently, since the results from the timing simulations correspond to expected results from theoretical analysis, it can be concluded that the arithmetic unit is functioning correctly.

Fitting and Synthesis

Logic Unit

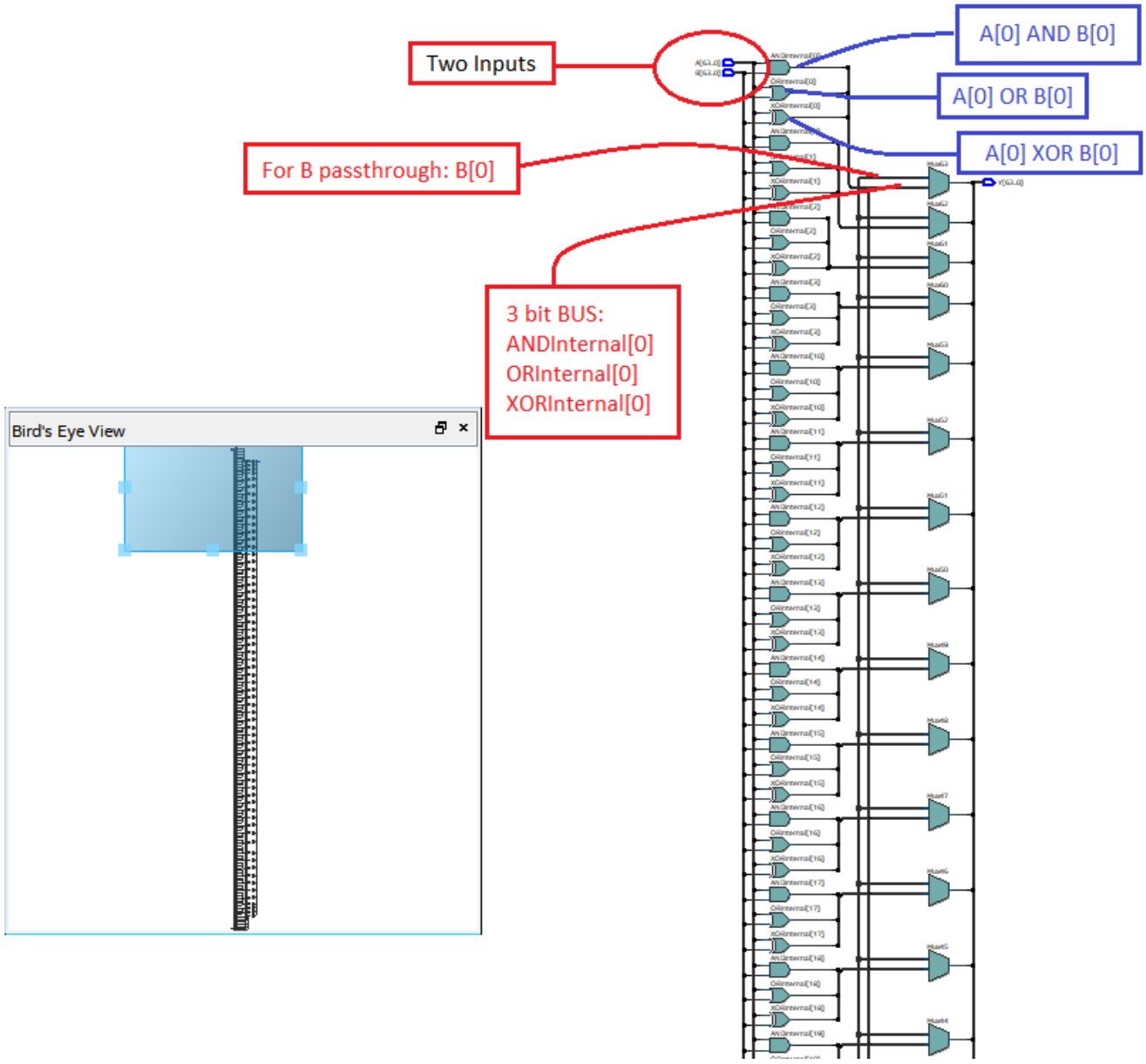

Figure 60: RTL-Netlist viewer of the Logic Unit. In the bottom-left corner, the entire 64-bit circuit can be observed. However, focusing and zooming on a smaller portion of the circuit is beneficial to view individual connections in greater detail. We are able to observe three, 1-bit logic gates (labelled in blue) taking A[0] and B[0] as input. The outputs of the three logic gates are displayed as a 3-bit BUS (labelled in red). A 4-channel, 1-bit multiplexor is also shown (labeled in red).

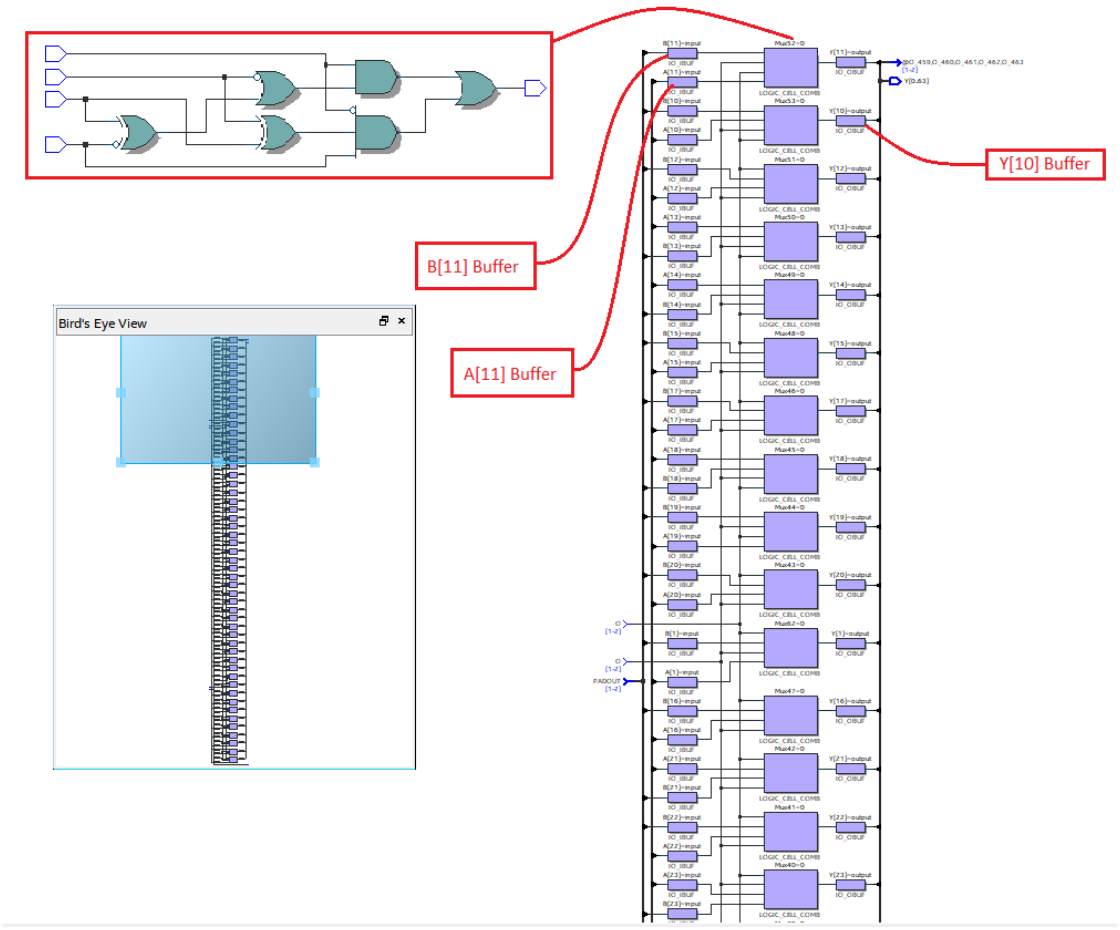

Figure 61: Similarly, in the Post-fitting Netlist viewer, the birds-eye view of the entire logic unit can be observed in the bottom-left corner, while a focused and zoomed-in image is observed. This image differs slightly from Figure 60. Firstly, we are able to observe three, 1-bit buffers, each for A, B, and Y (the buffers for the 11th bit are labeled in red). Moreover, each of the larger, purple rectangles (shown zoomed-in in the red box) are observed to be a circuit with four inputs: A[n], B[n], LogicFN[0] and LogicFN[1], where n is the n-th bit.

Arithmetic Unit

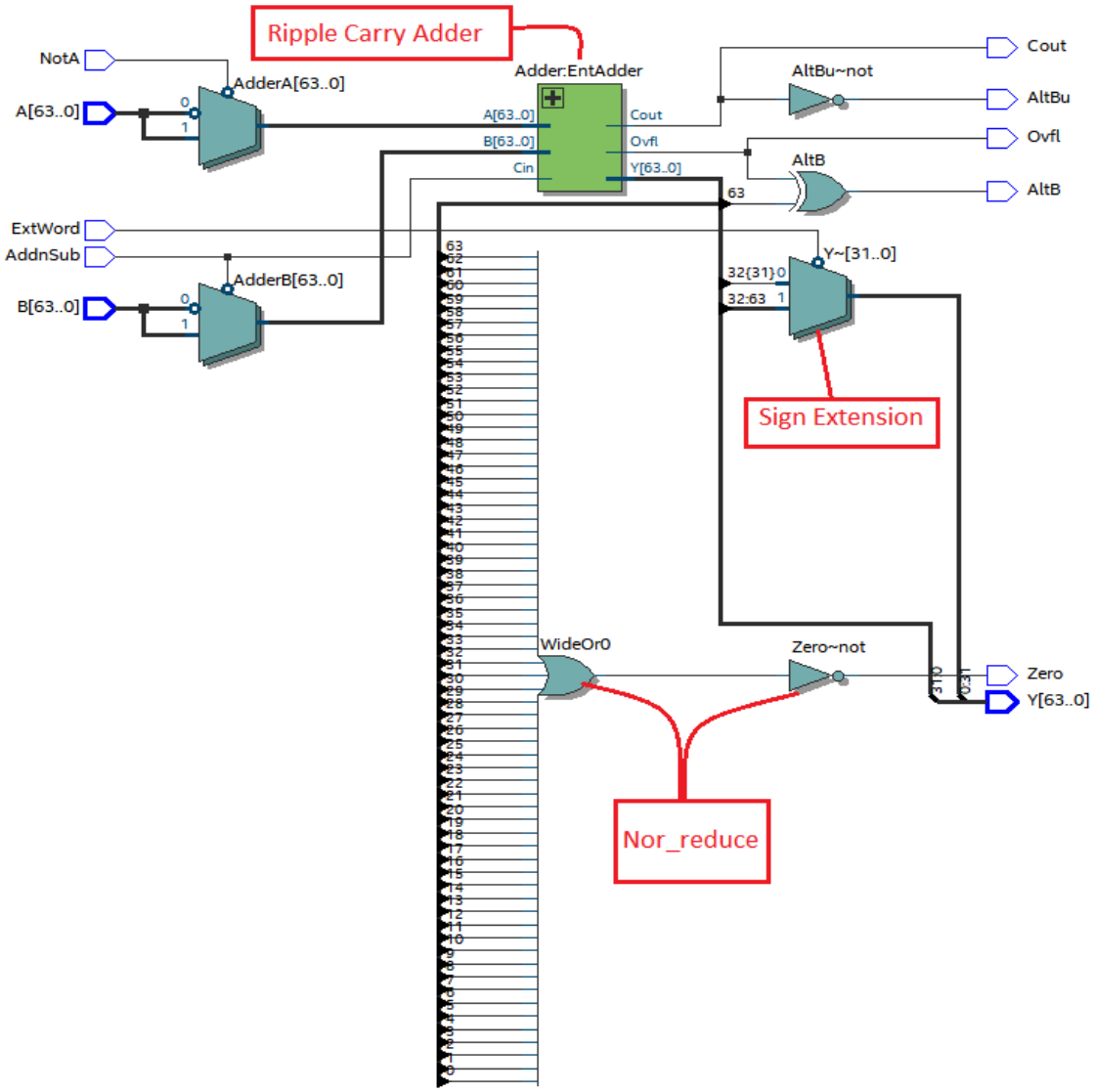

Figure 62: RTL-Netlist viewer of the Logic Unit. Focusing and zooming on a smaller portion of the circuit is beneficial to view individual connections in greater detail. At the top left, we can observe the 2 MUXes used to select between the input operands and their complementation. Moving further right, we can observe the 64-bit adder that will perform the chosen arithmetic operation through addition. On its right, we can see the assignment of 4 status signals (from top to bottom): carry-out, A less than B unsigned, arithmetic overflow, A less than B signed. Below those 4 we can see another set of MUXes used for sign-extending 32-bit results into 64-bit, the MUX output feeds into the result output Y. Just above the Y output label, is the Zero status signal label, we can trace its assignment back to the massive 64-bit NOR-gate located at the center.

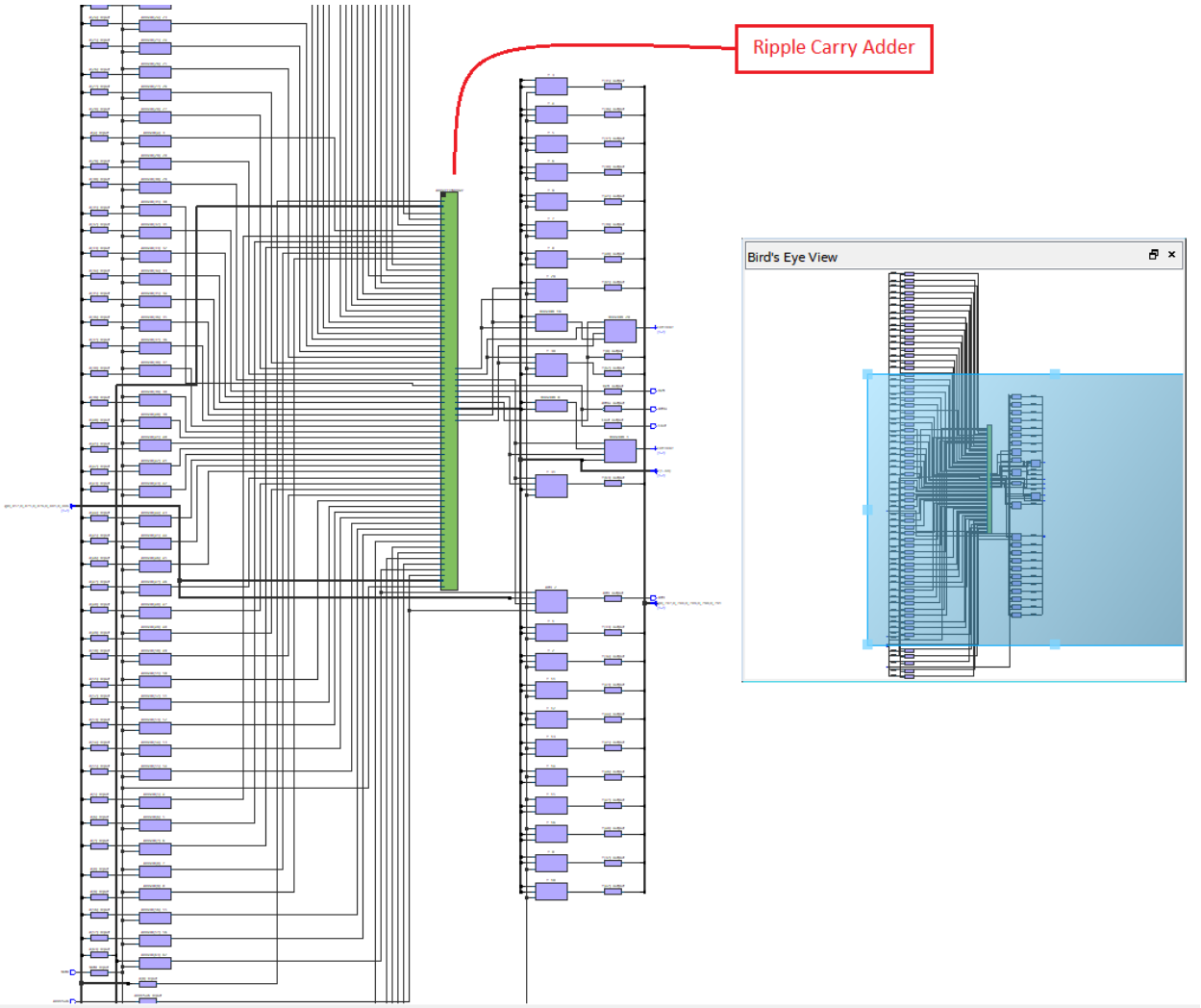

Figure 63: Similarly, in the Post-fitting Netlist viewer, the birds-eye view of the entire logic unit can be observed on the right, while a focused and zoomed-in image is observed on the left. This image differs slightly from Figure 62. From the left, we can see for each bit of the inputs a column of buffers, followed by a column of MUXes selecting between the inputs and their negation. Then the data is fed into the Adder instance at the center. After which, various status signals are calculated in the cluster of MUXes in the center-right of the Netlist view, and sign extension is performed by MUXes above and below the status signal cluster.

Shift Unit

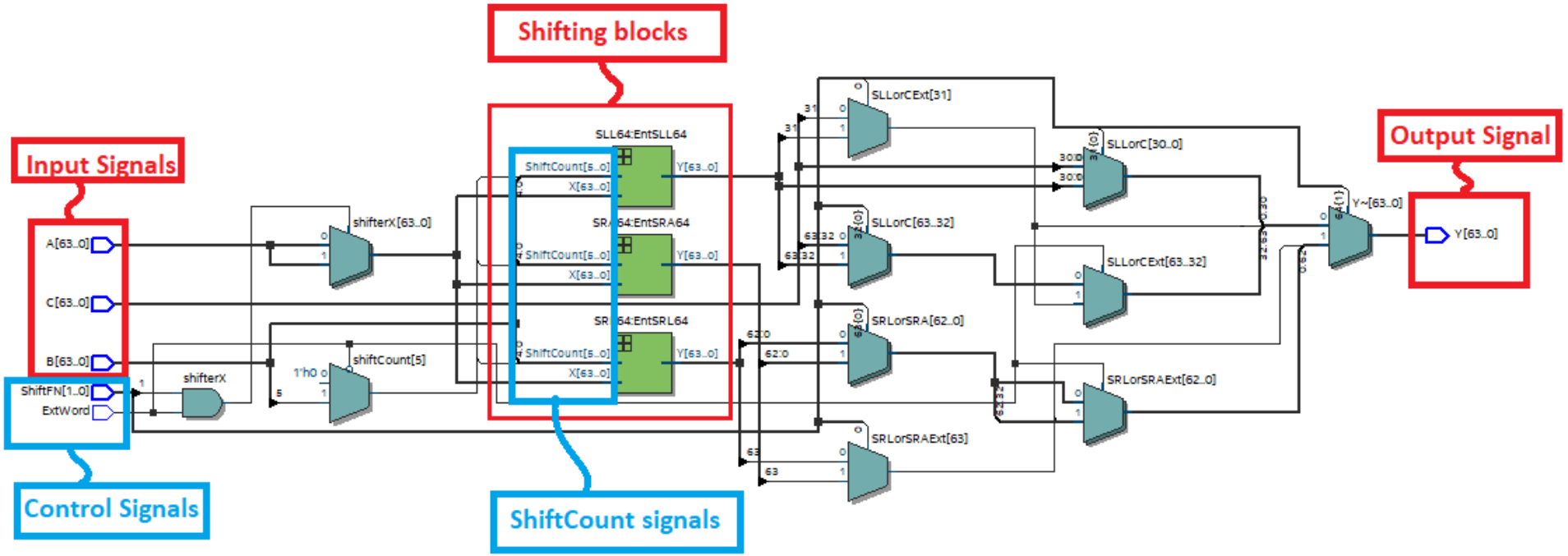

Figure 64: RTL Netlist-viewer of the Shift Unit. On the left-hand side of the image 64-bit input signals *A, B,* and*, C* are indicated in red. Also, control signals *ShiftFN* and *ExtWord* are indicated in blue. Furthermore, near the centre of the circuit three rectangles can be observed; each containing the circuitry of a barrel shifter. On the right-side of the image, the multiplexors controlled by the control signals can also be observed.

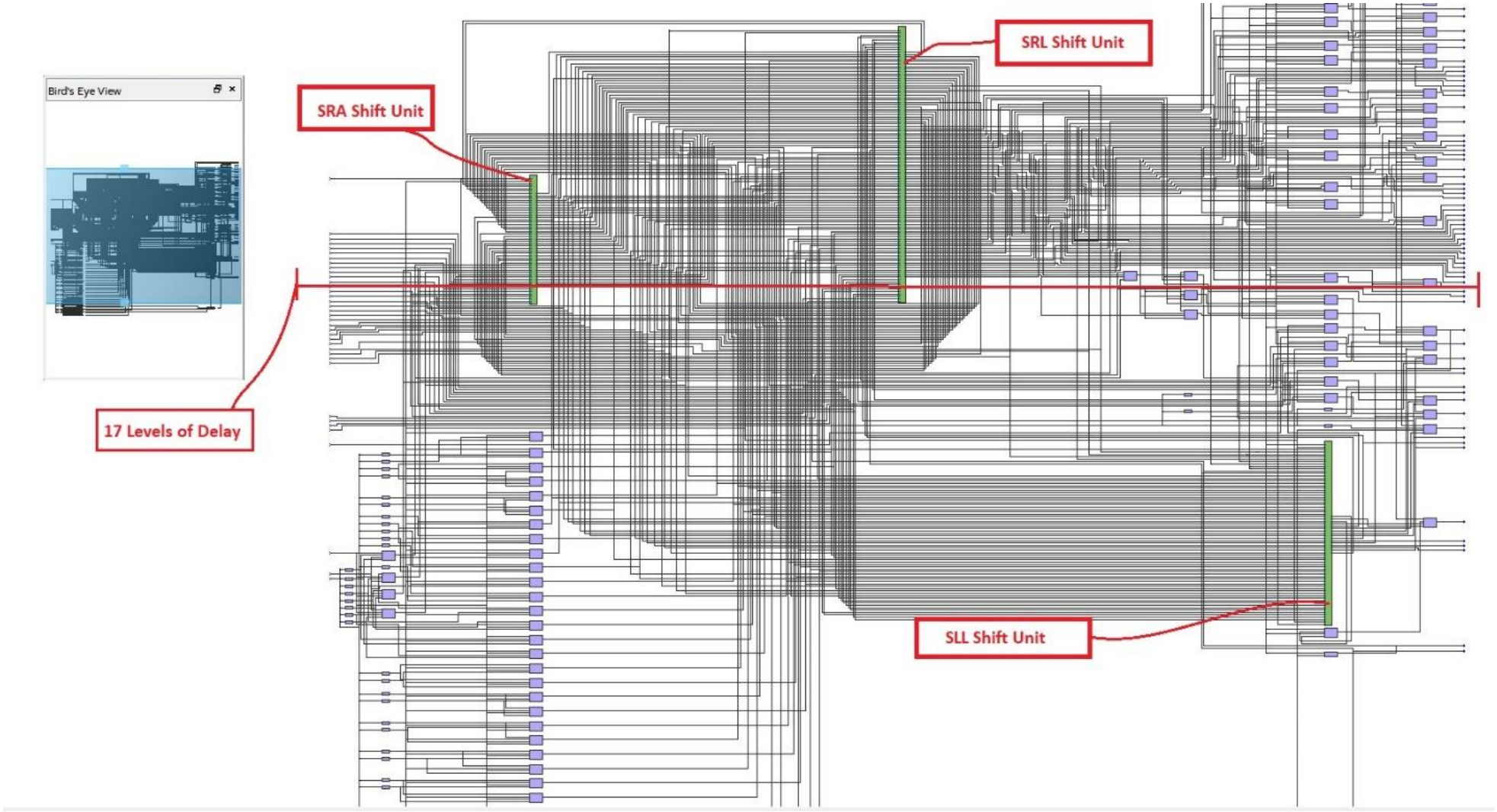

Figure 65: The post-fit of the Netlist-viewer for the shift unit. In the top-left corner, the ‘Bird’s Eye’ view can be observed. However, zooming focus on the part of the circuit which contains the three barrel-shifter provides some useful observations. Mainly, the shift unit gate-delay can be observed upon expanding a barrel shifter; it is found to be 17 levels.

Execution Unit

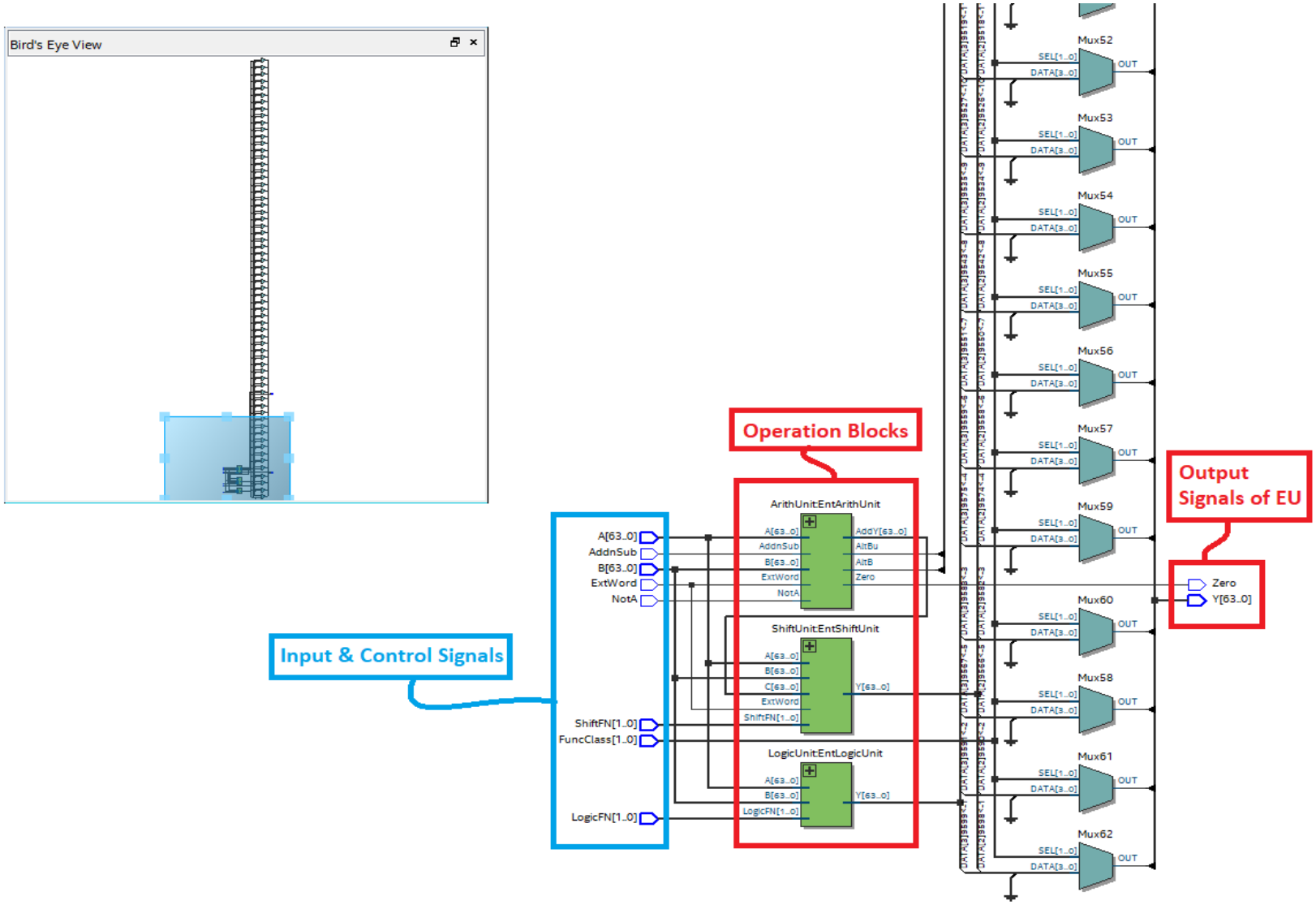

Figure 66: RTL-Netlist viewer of the Execution Unit. The blue box on the left shows all the inputs, both data and control signals. From there, they are fed into the three sub-circuits: arithmetic unit, shift unit, logic unit. These sub-circuits perform data manipulation operations based on the appropriate control signals. To their right are 64 MUXes selecting the output from the 3 sub-circuits. This is necessary as all sub-circuits output data every time and Exec Unit must choose the valid output. Finally at the right of the MUXes are the outputs for data and status signals. Only Zero is shown here but AltB and AltBu are placed at this level as well.

Figure 67: In the Post-fitting Netlist viewer, the birds-eye view of the entire logic unit can be observed on the left, while a focused and zoomed-in image is observed on the right. In this overview of the Post Fit layout, we can still identify the sub-circuits of the Logic Unit, the Arithmetic Unit, and the Shift Unit.

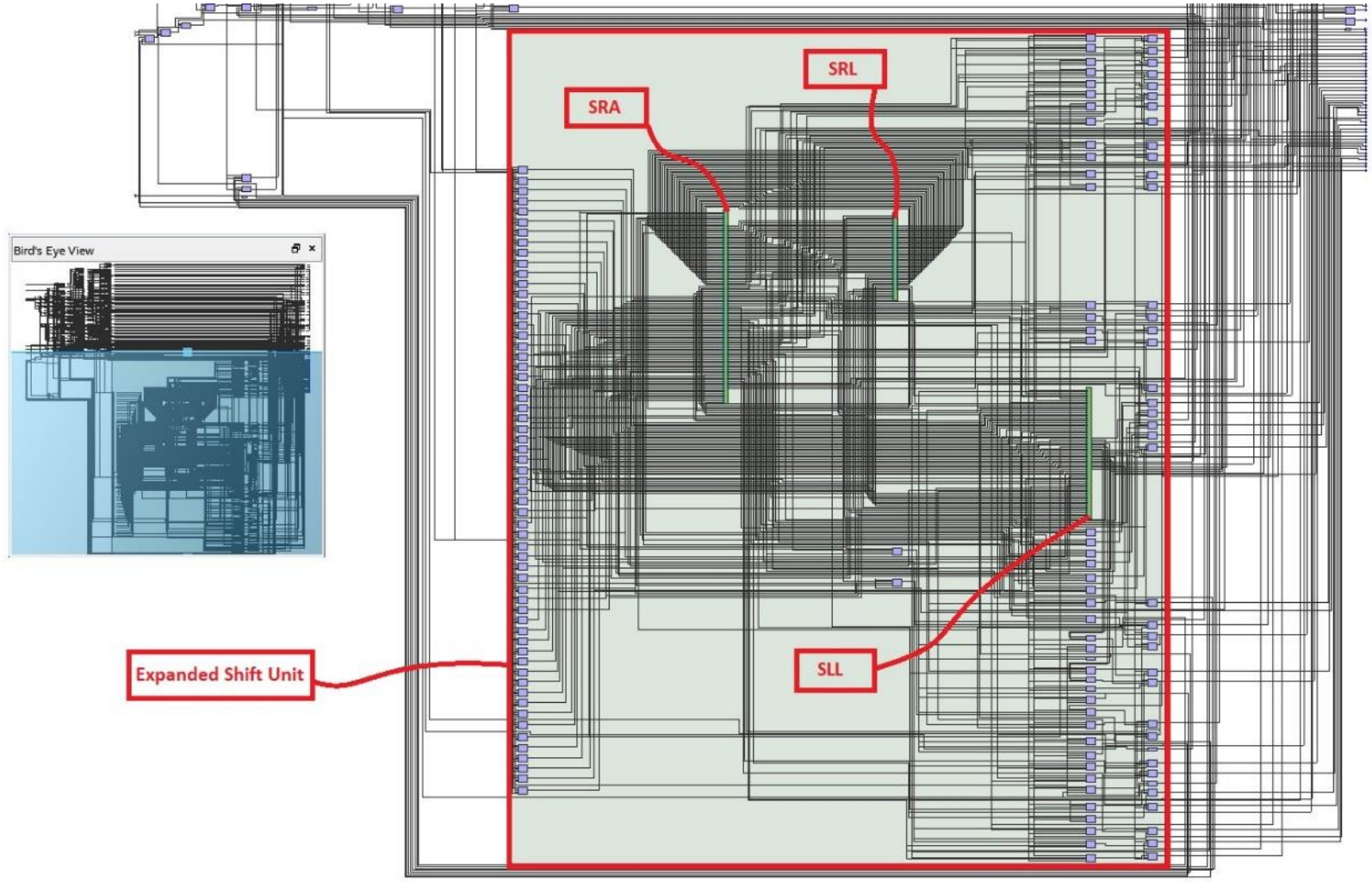

Figure 68: Here we see the expanded Shift Unit . As expected, it contains logic for output selection, in addition to three sub-circuits for each of the shift operations: shift left logical (SLL), shift right logical (SRL), and shift right arithmetic (SRA). This is familiar sight compared to the post fit layout of the standalone shift unit.

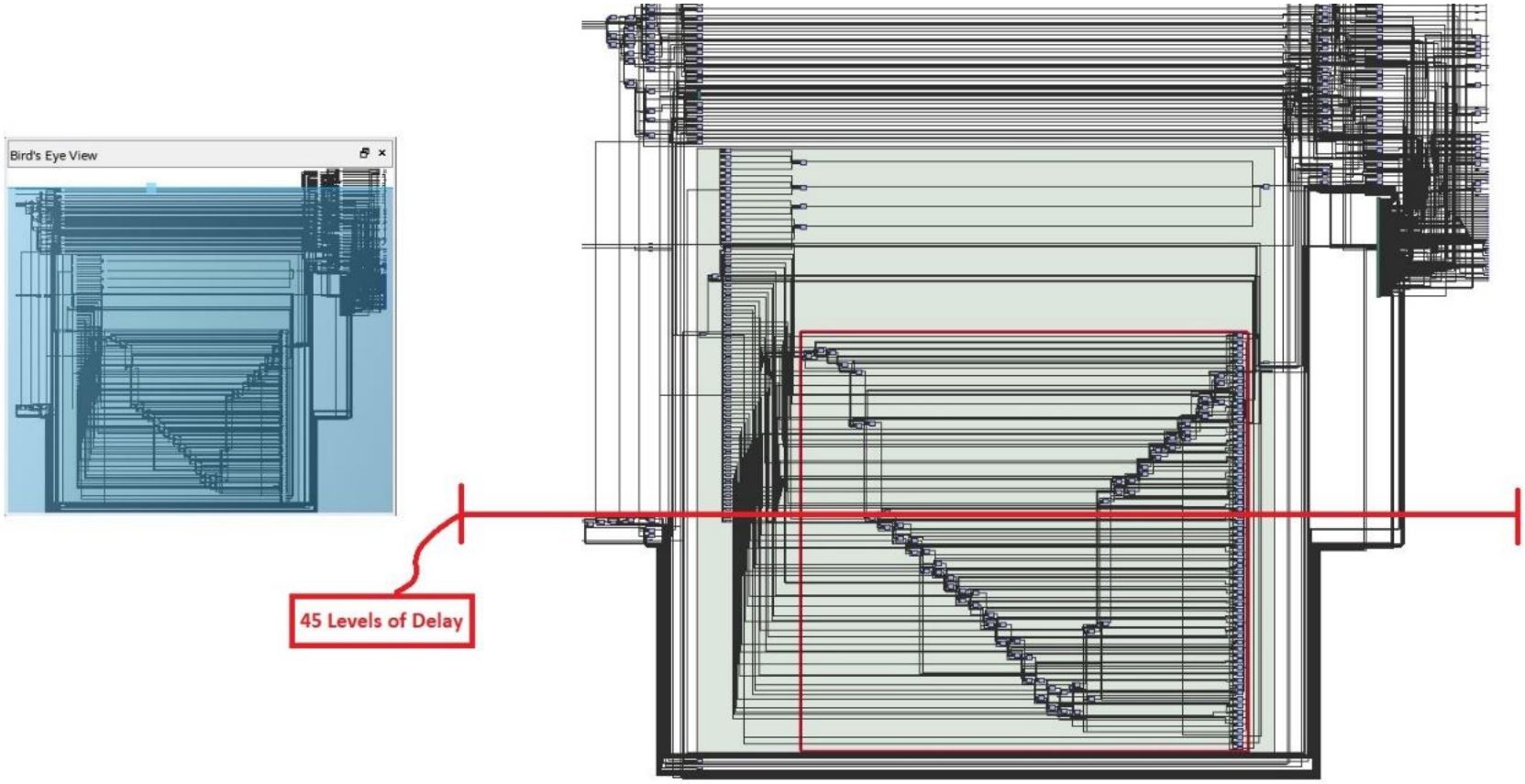

Figure 69: Here we consider the maximum possible level of delay experienced by the Execution Unit. As logic and shift operations can be easily parallelized with respect to bits, the arithmetic unit will cause the highest level of delay. The ripple-carry adder design of the Adder within the arithmetic unit will cause a single level of delay for each two bits of the input data. When we count those delays and add them to the delays found outside of the arithmetic unit, we concluded the maximum possible delay is 45 levels for the Execution Unit.

Concluding Remarks

Throughout this incremental project, design, functional verification, synthesis, and timing simulations were performed while constructing the execution unit of a RISC-V processor. This allowed in-depth knowledge building of important aspects of the engineering design cycle. In the beginning stages of the project, the logic unit and arithmetic unit were designed, synthesized, and verified. This provided opportunities to further understanding in the processes of functional and timing verification on a relatively smaller scale. Moreover, by viewing RTL Netlist diagrams of each unit, a deeper understanding of how Quartus synthesizes logic circuitry was developed by being able to observe how portions of circuitry were translated to lookup tables on the FPGA.

In the second half of the project, three barrel shifters were designed and pieced together to form the shift unit. Finally, the execution unit was constructed by combining the shift unit, arithmetic unit, and logic unit. Together, the combination of these smaller circuitry components is much larger than the RTL Netlist diagrams that were previously viewed in part 1. This provided an opportunity to understand how Quartus implements a comparatively larger circuit on the FPGA. Furthermore, with circuitry components composed of smaller components, trouble-shooting errors found in functional simulations was more difficult.

The entirety of this project provided opportunities to develop a much further understanding of digital design and was extremely rewarding.