Laboratory Report #4

Name: Josh Ratificar **Date Completed:** 10-02-2023

Laboratory Exercise Title: Dataflow Modeling of Combinational Circuits

Block Diagrams:

Above includes the block diagrams for this Laboratory Exercise. Based on the diagrams, the following can be concluded:

4-Bit Comparator

8-Input Ports:

- **A[0], A[1], A[2], A[3], B[0], B[1], B[2], B[3]**

3-Output Ports:

- **R[2], R[1], R[0]**

NOTE

G ("greater than" is TRUE) | R [2] = G (1 if A > B, else 0)

E ("equal to" is TRUE) | R [1] = E (1 if A = B, else 0)

L ("less than" is TRUE) | R [0] = L (1 if A < B, else 0)

Exercise 4A:

```
1  /*************************************
2  *  FILE:              FourBitComparator.v
3  *  AUTHOR:            Josh Ratificar
4  *  Class:             Gr.3 CpE 3101L Introduction to HDL
5  *  Group/Schedule     Friday, 7:30 AM to 10:30 AM
6  *  Description:       FourBitComparator.v module
7  *************************************/
8  module FourBitComparator(
9      input     [3:0]A,
10     input     [3:0]B,
11     output    [2:0]R
12 );
13     wire      [2:0]common;
14     // A > B @ R2 |R [2] = G (1 if A > B, else 0)
15     assign R[2] = (A > B)? 1'b1 : 1'b0;
16
17
18     // A = B @ R[1] | R[1] = E (1 if A = B, else 0)
19     assign R[1] = (A == B)? 1'b1 : 1'b0;
20
21     // A < B @ R[1] | R[1] = E (1 if A < B, else 0)
22     assign R[0] = (A < B)? 1'b1 : 1'b0;
23 endmodule
24
```

Figure 1.0 – *FourBitComparator.v Script*

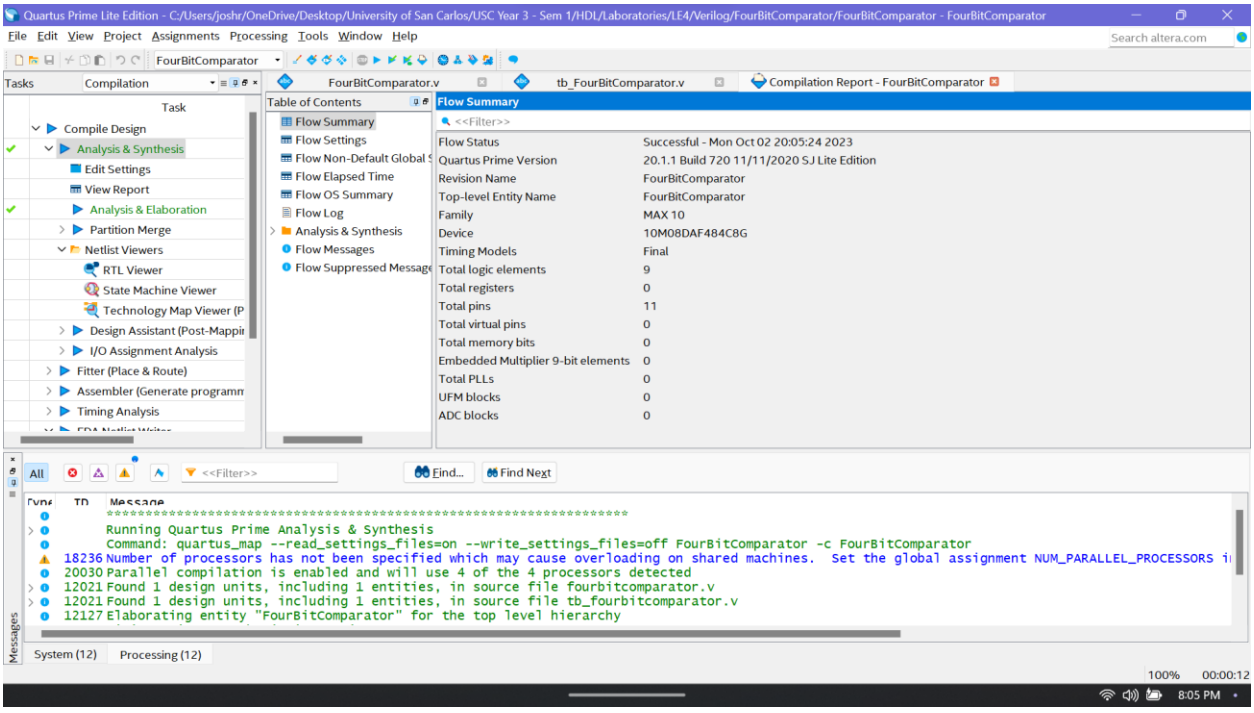

Figure 1.1 – *FourBitComparator.v Analysis and Elaboration Test Results*

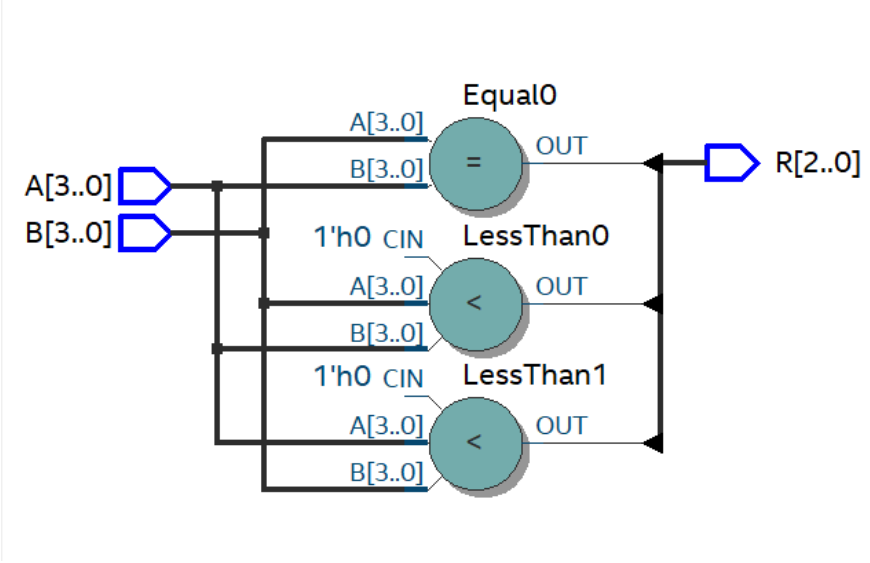

Figure 1.2 – *FourBitComparator RTL View Output*

```
/*************************************
*  FILE:              tb_FourBitComparator.v
*  AUTHOR:            Josh Ratificar
*  Class:             Gr.3 CpE 3101L Introduction to HDL
*  Group/Schedule     Friday, 7:30 AM to 10:30 AM
*  Description:       tb_FourBitComparator.v module
*************************************/
`timescale 1 ns/ 1 ps
module tb_FourBitComparator();
   reg    [3:0] A, B;
   wire   [2:0] R;
   FourBitComparator UUT(
      .A(A),
      .B(B),
      .R(R)
   );
   initial begin
      $display("Test Bench | tb_FourBitComparator...");
      // Test cases for A < B
      $display("A < B");
      A = 4'b0100; B = 4'b1010; #10; // A < B
      A = 4'b1100; B = 4'b1111; #10; // A < B
      A = 4'b0010; B = 4'b1101; #10; // A < B
      A = 4'b0011; B = 4'b1101; #10; // A < B

      // Test cases for A = B
      $display("A = B");
      A = 4'b1110; B = 4'b1110; #10; // A = B
      A = 4'b1111; B = 4'b1111; #10; // A = B
      A = 4'b1000; B = 4'b1000; #10; // A = B-
      A = 4'b0000; B = 4'b0000; #10; // A = B

      // Test cases for A > B
      $display("A > B");
      A = 4'b0001; B = 4'b0000; #10; // A > B
      A = 4'b1111; B = 4'b1110; #10; // A > B
      A = 4'b0101; B = 4'b0011; #10; // A > B
      A = 4'b0010; B = 4'b0001; #10; // A > B
      $stop;
   end
   initial begin
      $monitor("Time = %2d ns\t A = %d | [%b]\t B = %d [%b]\t R = %b", $time, A, A, B, B, R);
   end
endmodule

```

Figure 1.3 – *tb_ FourBitComparator.v Script for Testing Module*

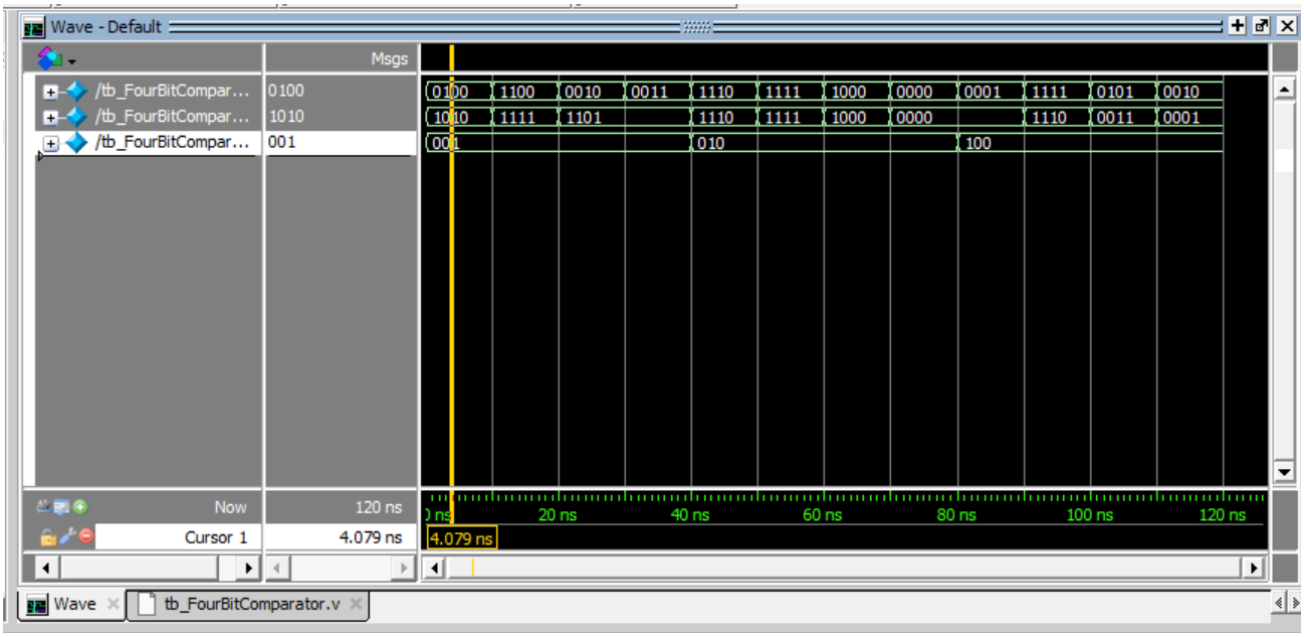

Figure 1.4 – *Four Bit Comparator RTL Simulation Output*

```
# Test Bench | tb_FourBitComparator...
# A < B
# Time =  0 ns  A =  4 | [0100]  B = 10 [1010]  R = 001
# Time = 10 ns  A = 12 | [1100]  B = 15 [1111]  R = 001
# Time = 20 ns  A =  2 | [0010]  B = 13 [1101]  R = 001
# Time = 30 ns  A =  3 | [0011]  B = 13 [1101]  R = 001
# A = B
# Time = 40 ns  A = 14 | [1110]  B = 14 [1110]  R = 010
# Time = 50 ns  A = 15 | [1111]  B = 15 [1111]  R = 010
# Time = 60 ns  A =  8 | [1000]  B =  8 [1000]  R = 010
# Time = 70 ns  A =  0 | [0000]  B =  0 [0000]  R = 010
# A > B
# Time = 80 ns  A =  1 | [0001]  B =  0 [0000]  R = 100
# Time = 90 ns  A = 15 | [1111]  B = 14 [1110]  R = 100
# Time = 100 ns  A =  5 | [0101]  B =  3 [0011]  R = 100
# Time = 110 ns  A =  2 | [0010]  B =  1 [0001]  R = 100
# ** Note: $stop    : C:/Users/joshr/OneDrive/Desktop/University of San Carlos/USC Year 3 - Sem 1/HDL/Laboratories/LE4/Verilog/FourBitComparator/tb_FourBitComparator.v(39
#    Time: 120 ns  Iteration: 0  Instance: /tb_FourBitComparator
# Break in Module tb_FourBitComparator at C:/Users/joshr/OneDrive/Desktop/University of San Carlos/USC Year 3 - Sem 1/HDL/Laboratories/LE4/Verilog/FourBitComparator/tb_Fo
parator.v line 39

VSIM 2>
```

Figure 1.5 – *Four Bit Comparator Test Bench Monitor Output (Annotations to **Figure 1.4**)*

Discussion of Results (Exercise 4A)

By observing the Test bench script (**Figure 1.3**), we expect that the respective R bit reflect the comparison between two four-bit inputs. This is the best-case desired behaviour expected which after simulating through RTL simulation (**Figure 1.4**), R appropriately and respectively changes every 40 ns. This is how the test bench was setup, and we can furthermore observe this behaviour in **Figure 1.5**. In conclusion, our design for our comparator works even when we applied Data Flow design in HDL Verilog.

Exercise 4B:

```
1  /************************************
2  *  FILE:              n_bit4x1Multiplexer.v
3  *  AUTHOR:            Josh Ratificar
4  *  Class:             Gr.3 CpE 3101L Introduction to HDL
5  *  Group/Schedule     Friday, 7:30 AM to 10:30 AM
6  *  Description:       n_bit4x1Multiplexer.v module
7  ************************************/
8  module n_bit4x1Multiplexer
9  #(parameter n = 4)
10 (
11    input [(n-1):0] A, B, C, D,
12    input [1:0] S,
13    output [(n-1):0] Y
14 );
15    assign Y = (S == 2'b00) ? A : (S == 2'b01) ? B : (S == 2'b10) ? C : (S == 2'b11) ? D : {n{1'b0}};
16 endmodule
17
```

Figure 2.0 – *n_bit4x1Multiplexer.v Script*

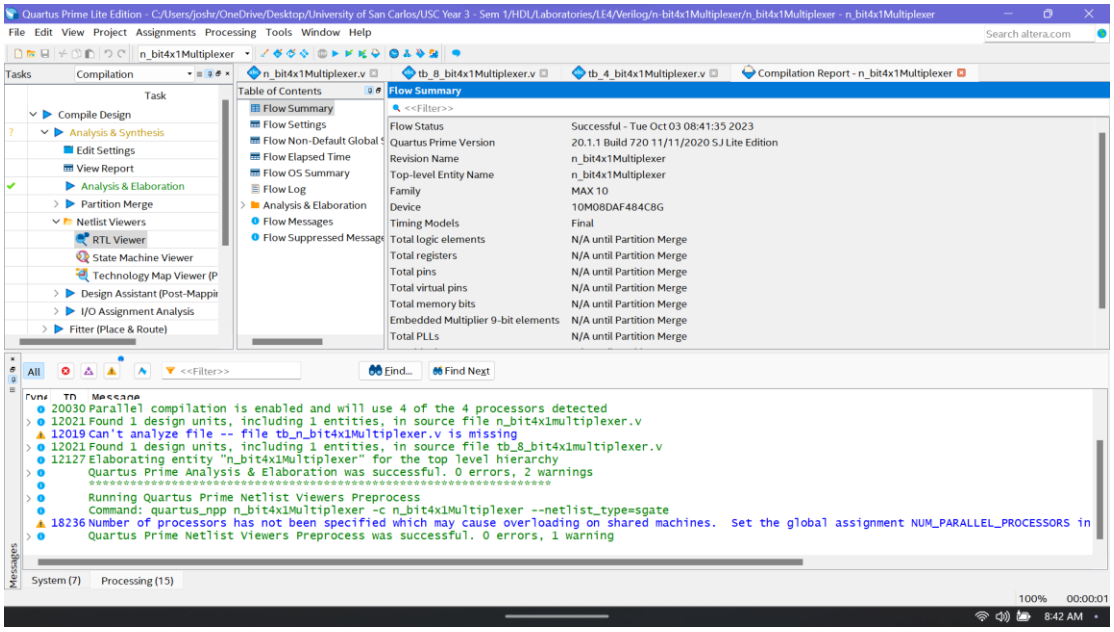

Figure 2.1 – *n_bit4x1Multiplexer.v Analysis and Elaboration Test Results*

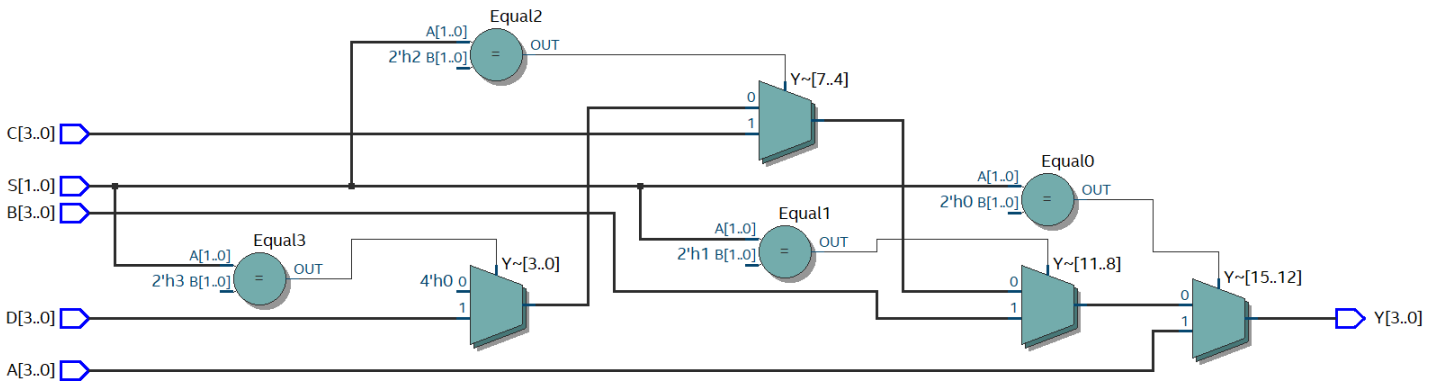

Figure 2.2 – *n_bit4x1Multiplexer RTL View Output*

```
1  ************************************
2   FILE:              tb_4_bit4x1Multiplexer.v
3   AUTHOR:            Josh Ratificar
4   Class:             Gr.3 CpE 3101L Introduction to HDL
5   Group/Schedule     Friday, 7:30 AM to 10:30 AM
6   Description:       tb_4_bit4x1Multiplexer.v test bench for n_bit4x1Multiplexer
7  ************************************/
8  dule tb_4_bit4x1Multiplexer();
9   reg  [3:0] A, B, C, D;
10  reg  [1:0] S;
11  wire [3:0] Y;
12
13  n_bit4x1Multiplexer #(.n(4)) UUT(
14      .A(A),
15      .B(B),
16      .C(C),
17      .D(D),
18      .S(S),
19      .Y(Y)
20  );
```

```
21  initial begin
22      $display("Test Bench | tb_4_bit4x1Multiplexer...");
23      $display("Test case 1: Select A");
24      A = 4'b1010; B = 4'b0110; C = 4'b1101; D = 4'b0001; S = 2'b00; #10;
25
26      $display("Test case 2: Select B");
27      A = 4'b1010; B = 4'b0110; C = 4'b1101; D = 4'b0001; S = 2'b01; #10;
28
29      $display("Test case 3: Select C");
30      A = 4'b1010; B = 4'b0110; C = 4'b1101; D = 4'b0001; S = 2'b10; #10;
31
32      $display("Test case 4: Select D");
33      A = 4'b1010; B = 4'b0110; C = 4'b1101; D = 4'b0001; S = 2'b11; #10;
34      $stop;
35  end
36  initial begin
37      $monitor("Time = %2d ns\t |A = %d [%b]|\t |B = %d [%b]|\t |C = %d [%b]|\t |D = %d [%b]|\t |S = %d [%b]|\t |Y = %d
38  end
39 dmodule
40
```

Figure 3.0 – *tb_4_bit4x1Multiplexer.v Script for Testing Module*

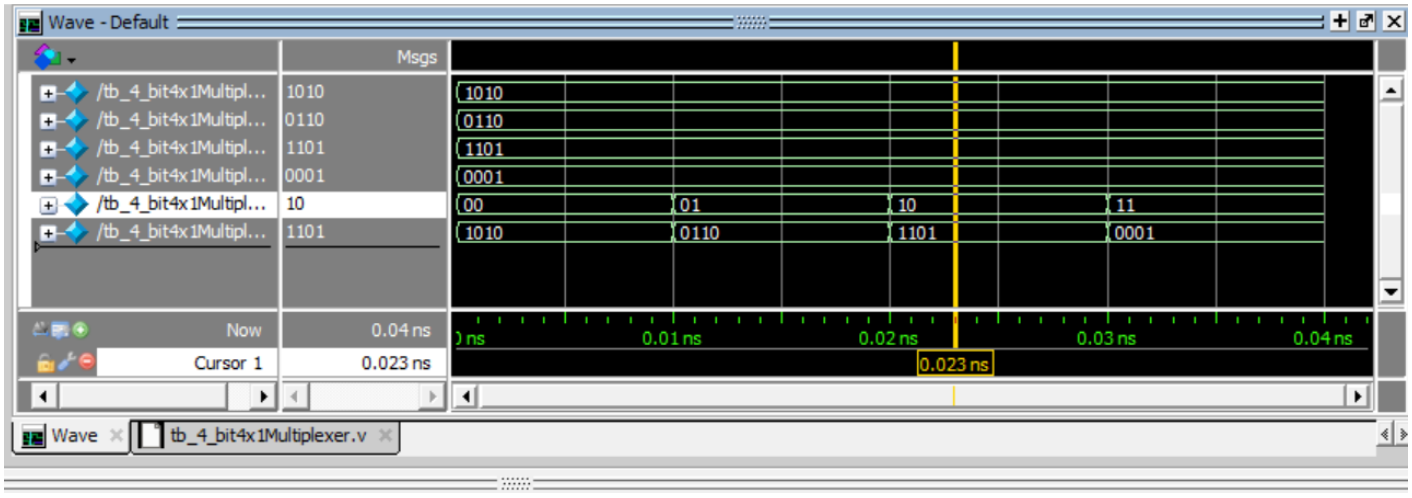

Figure 3.1 – *tb_4_bit4x1Multiplexer RTL Simulation Output*

```
# Test Bench | tb_4_bit4x1Multiplexer...
# Test case 1: Select A
# Time =  0 ns  |A = 10 [1010]|  |B =  6 [0110]|  |C = 13 [1101]|  |D =  1 [0001]|  |S = 0 [00]|  |Y = 10 [1010]|
# Test case 2: Select B
# Time = 10 ns  |A = 10 [1010]|  |B =  6 [0110]|  |C = 13 [1101]|  |D =  1 [0001]|  |S = 1 [01]|  |Y =  6 [0110]|
# Test case 3: Select C
# Time = 20 ns  |A = 10 [1010]|  |B =  6 [0110]|  |C = 13 [1101]|  |D =  1 [0001]|  |S = 2 [10]|  |Y = 13 [1101]|
# Test case 4: Select D
# Time = 30 ns  |A = 10 [1010]|  |B =  6 [0110]|  |C = 13 [1101]|  |D =  1 [0001]|  |S = 3 [11]|  |Y =  1 [0001]|
# ** Note: $stop    : C:/Users/joshr/OneDrive/Desktop/University of San Carlos/USC Year 3 - Sem 1/HDL/Laboratories/LE4/Verilog/n-bit4x1Multiplexer/tb
_4_bit4x1Multiplexer.v(34)
#    Time: 40 ps  Iteration: 0  Instance: /tb_4_bit4x1Multiplexer
# Break in Module tb_4_bit4x1Multiplexer at C:/Users/joshr/OneDrive/Desktop/University of San Carlos/USC Year 3 - Sem 1/HDL/Laboratories/LE4/Verilog/
n-bit4x1Multiplexer/tb_4_bit4x1Multiplexer.v line 34
```

Figure 3.2 – *tb_4_bit4x1Multiplexer Test Bench Monitor Output (Annotations to **Figure 3.1**)*

```
1  /***************************************
2  *  FILE:              tb_8_bit4x1Multiplexer.v
3  *  AUTHOR:            Josh Ratificar
4  *  Class:             Gr.3 CpE 3101L Introduction to HDL
5  *  Group/Schedule     Friday, 7:30 AM to 10:30 AM
6  *  Description:       tb_8_bit4x1Multiplexer.v test bench for n_bit4x1Multiplexer
7  ***************************************/
8  module tb_8_bit4x1Multiplexer();
9     reg  [7:0] A, B, C, D;
10    reg  [1:0] S;
11    wire [7:0] Y;
12
13    n_bit4x1Multiplexer #(.n(8)) UUT(
14       .A(A),
15       .B(B),
16       .C(C),
17       .D(D),
18       .S(S),
19       .Y(Y)
20    );
```

```
21    initial begin
22       $display("Test Bench | tb_8_bit4x1Multiplexer...");
23       $display("Test case 1: Select A");
24       A = 8'b10101010; B = 8'b01100110; C = 8'b11011101; D = 8'b00010001; S = 2'b00; #10;
25       $display("Test case 2: Select B");
26       A = 8'b10101010; B = 8'b01100110; C = 8'b11011101; D = 8'b00010001; S = 2'b01; #10;
27       $display("Test case 3: Select C");
28       A = 8'b10101010; B = 8'b01100110; C = 8'b11011101; D = 8'b00010001; S = 2'b10; #10;
29       $display("Test case 4: Select D");
30       A = 8'b10101010; B = 8'b01100110; C = 8'b11011101; D = 8'b00010001; S = 2'b11; #10;
31       $stop;
32    end
33    initial begin
34       $monitor("Time = %2d ns\t |A = %d [%b]|\t |B = %d [%b]|\t |C = %d [%b]|\t |D = %d [%b]|\t |S = %d [%b]|\t |Y = %
35    end
36 endmodule
37
```

Figure 4.0 – *tb_8_bit4x1Multiplexer.v Script for Testing Module*

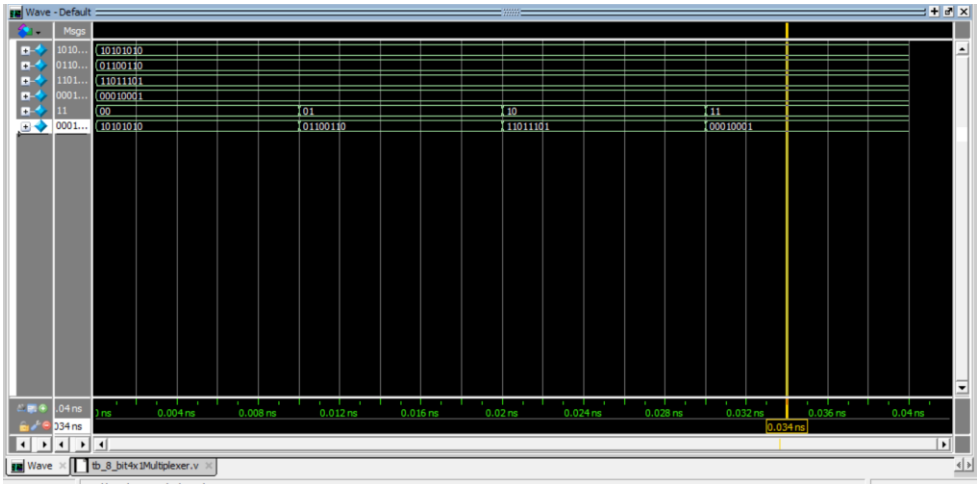

Figure 4.1 – *tb_4_bit4x1Multiplexer RTL Simulation Output*

```
# Test Bench | tb_8_bit4x1Multiplexer...
# Test case 1: Select A
# Time =  0 ns  |A = 170 [10101010]|  |B = 102 [01100110]|  |C = 221 [11011101]|  |D =  17 [00010001]|  |S = 0 [00]|  |Y = 170 [10101010]|
# Test case 2: Select B
# Time = 10 ns  |A = 170 [10101010]|  |B = 102 [01100110]|  |C = 221 [11011101]|  |D =  17 [00010001]|  |S = 1 [01]|  |Y = 102 [01100110]|
# Test case 3: Select C
# Time = 20 ns  |A = 170 [10101010]|  |B = 102 [01100110]|  |C = 221 [11011101]|  |D =  17 [00010001]|  |S = 2 [10]|  |Y = 221 [11011101]|
# Test case 4: Select D
# Time = 30 ns  |A = 170 [10101010]|  |B = 102 [01100110]|  |C = 221 [11011101]|  |D =  17 [00010001]|  |S = 3 [11]|  |Y =  17 [00010001]|
# ** Note: $stop    : C:/Users/joshr/OneDrive/Desktop/University of San Carlos/USC Year 3 - Sem 1/HDL/Laboratories/LE4/Verilog/n-bit4x1Multiplexer/tb_8_bi
t4x1Multiplexer.v(31)
#    Time: 40 ps  Iteration: 0  Instance: /tb_8_bit4x1Multiplexer
# Break in Module tb_8_bit4x1Multiplexer at C:/Users/joshr/OneDrive/Desktop/University of San Carlos/USC Year 3 - Sem 1/HDL/Laboratories/LE4/Verilog/n-bit
4x1Multiplexer/tb_8_bit4x1Multiplexer.v line 31
```

Figure 4.2 – *tb_4_bit4x1Multiplexer Test Bench Monitor Output (Annotations to **Figure 4.1**)*

Discussion of Results (Exercise 4B)

PART 1:

In **Figure 3.0**, this test bench is setup for a 4-bit 4 to 1 Mux. We are testing every possible combination of S to validate our circuit's behaviour. In our test-bench RTL simulation (**Figure 3.1**), we can observe that we tested four separate cases to select either A, B, C, or D. This behaviour is apparent every 10 ns, where the output across Y reflects its respective selection's four bits. To further strengthen this finding, we utilized the "$monitor" and "$display" commands to visually validate the behaviour which can be seen in **Figure 3.2**. Thus, our flexible n-bit 4 to 1 Mux operates as expected when n-bits is set to 4.

PART 2:

Likewise, **Figure 4.0** is setup for an 8-bit 4 to 1 Mux. The test-benches are set-up similarly, but there were as well additional input/output ports needed to accommodate the additional bits. In **Figure 4.1**, we can see that the same behaviour as described in **Figure 3.1** can be observed. Complementing this finding, **Figure 4.2** monitors which 8-bits are selected across Y, and our four test-cases reflect that only one of the 8-bits can be selected depending on S inputs. From this, we can conclude that our n-bits 4 to 1 Mux as well operates expectedly when n-bits is set to 8.