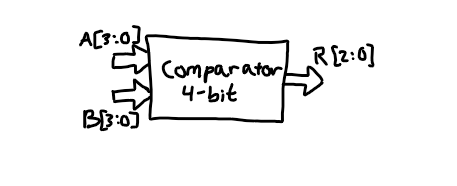
# Laboratory Report #4

**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]**

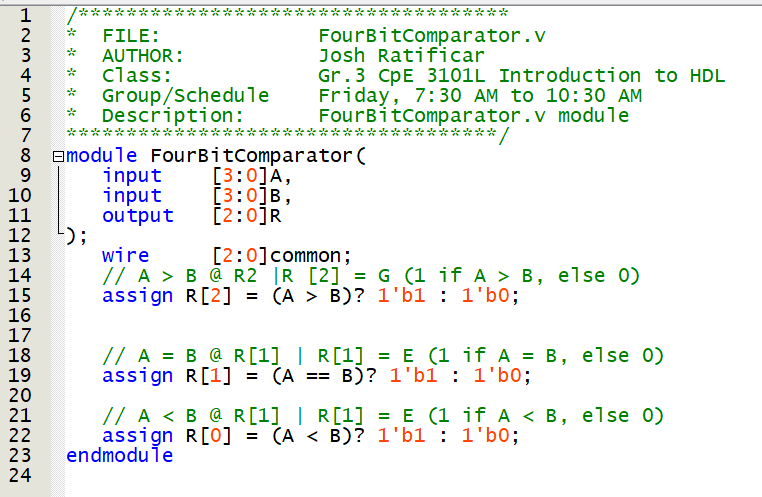
***NOTE***

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

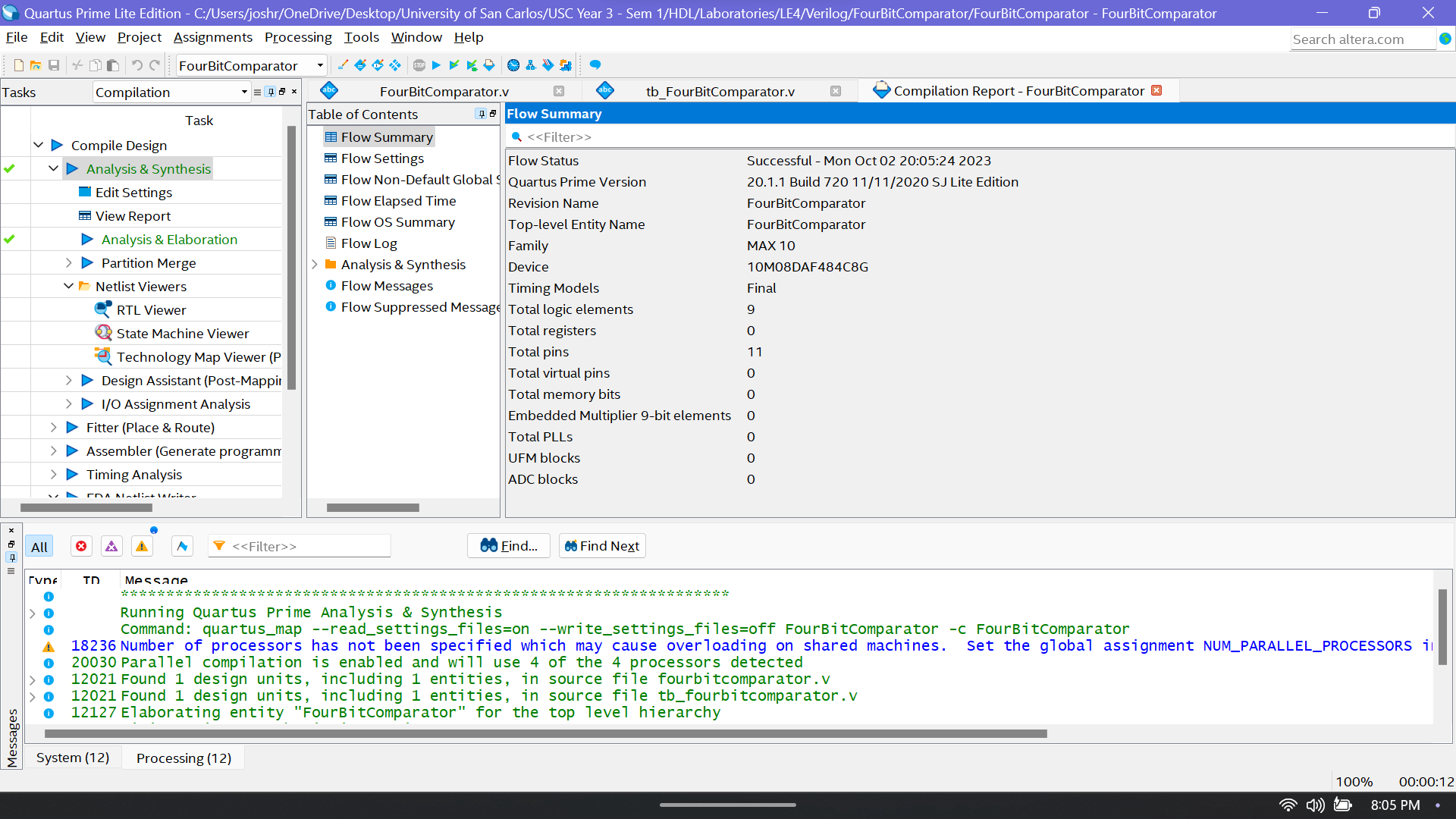
**E (“equal to” is TRUE) |** R [1] = E (1 if A = B, else 0)

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

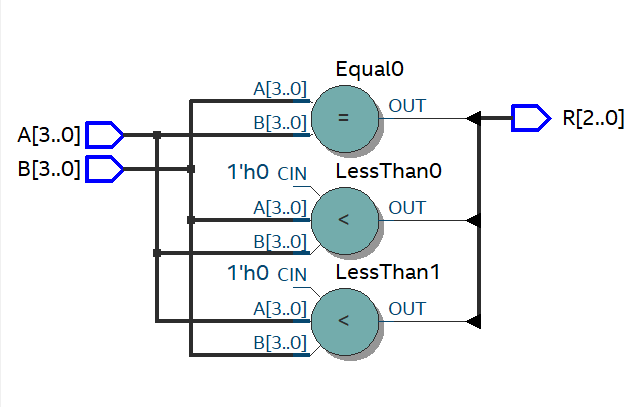
**Exercise 4A:**



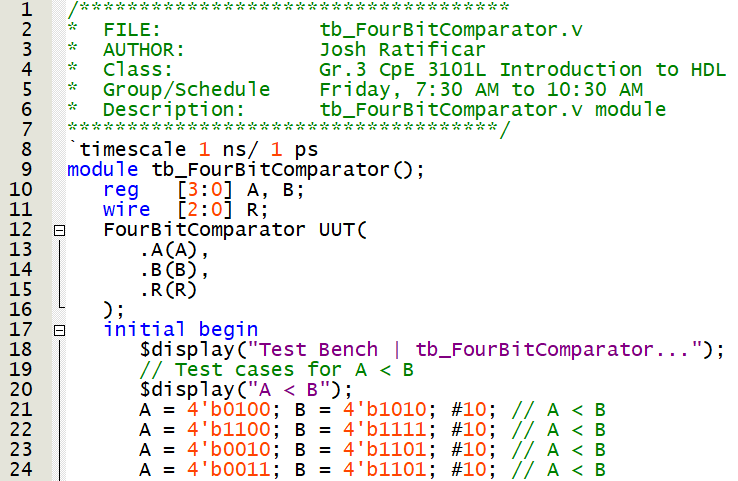
**Figure 1.0** – *FourBitComparator.v Script*

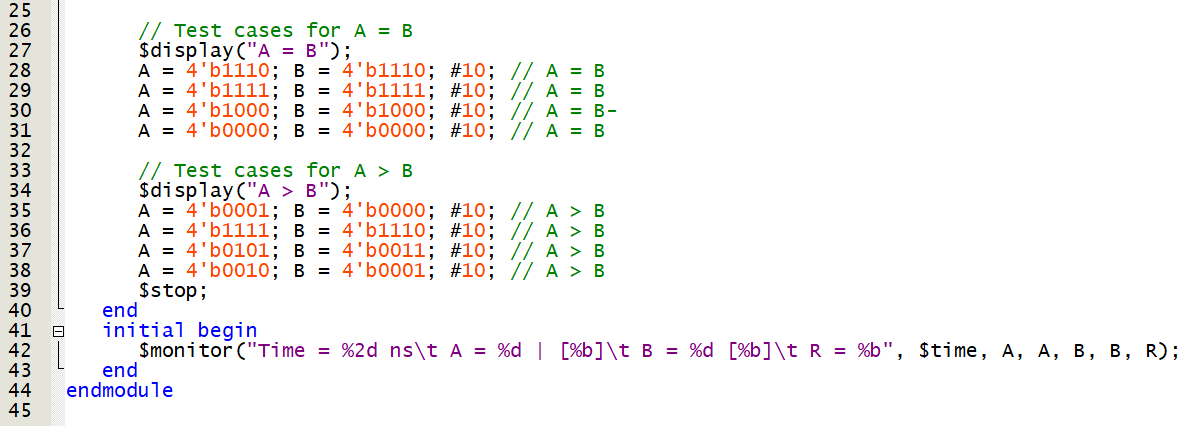
**

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

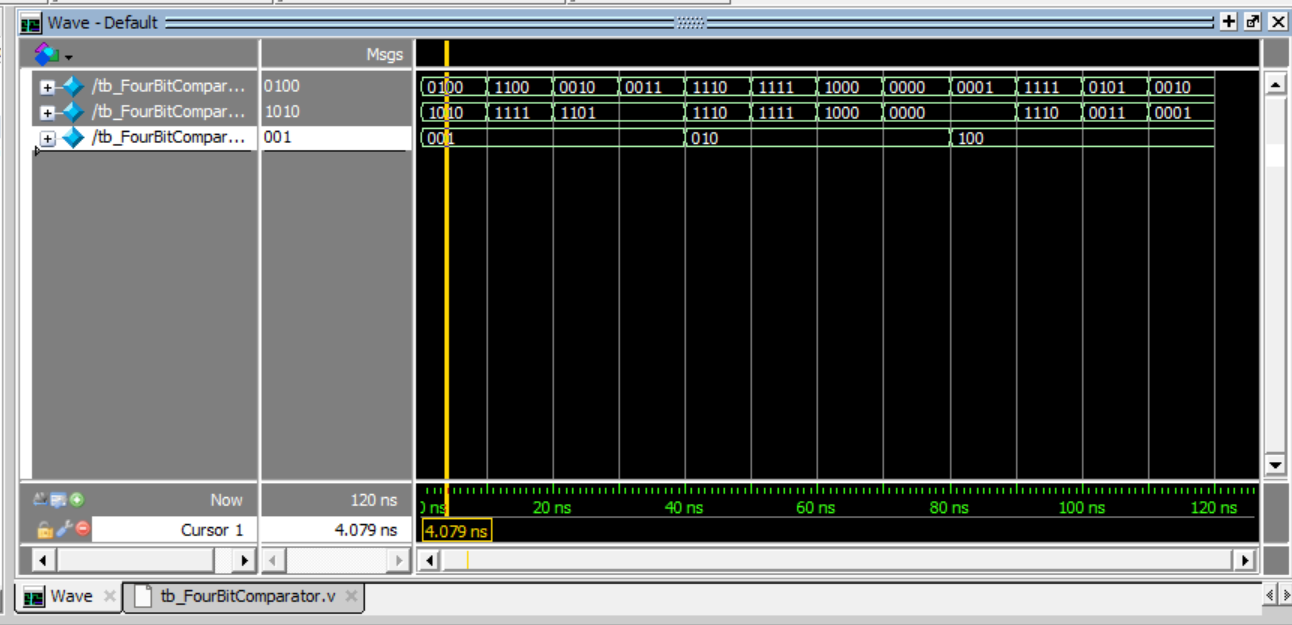


**Figure 1.2** – *FourBitComparator RTL View Output*

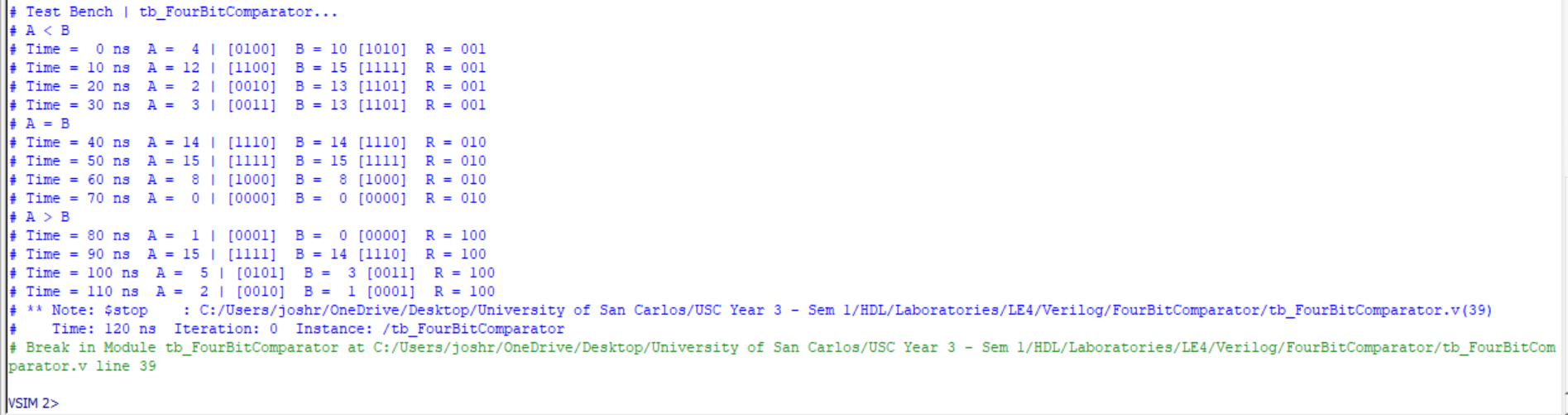




**Figure 1.3** – *tb\_* *FourBitComparator.v Script for Testing Module*

****

**Figure 1.4** – *Four Bit Comparator RTL Simulation Output*

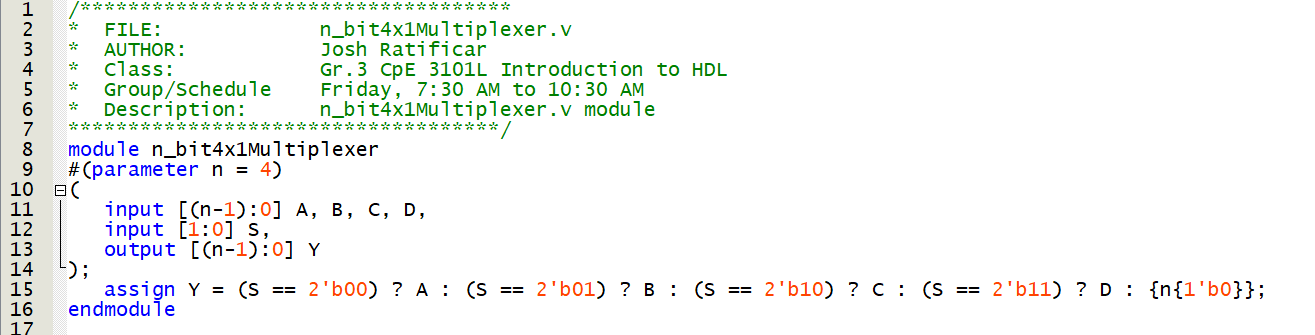
**

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

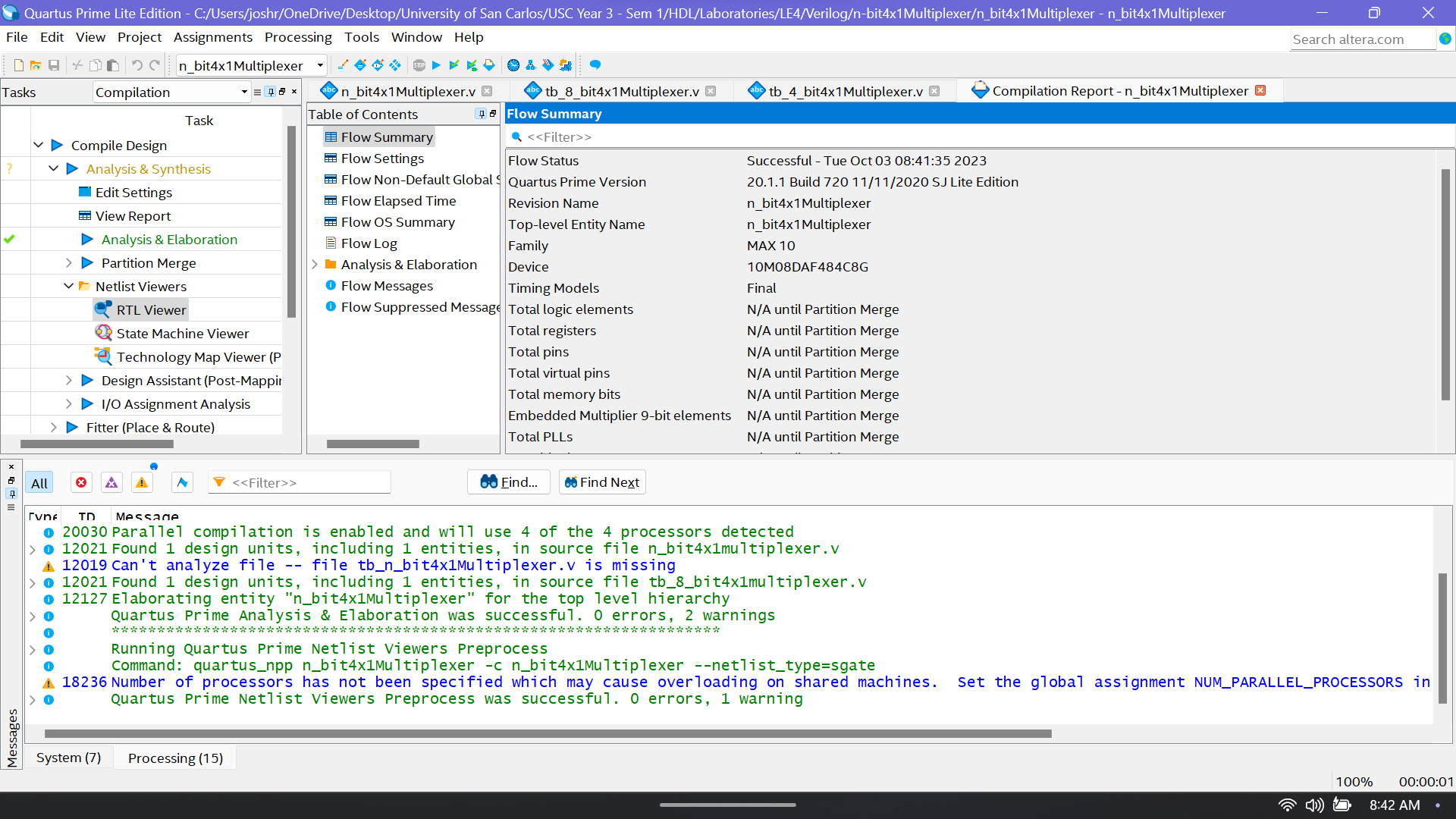
**Discussion of Results (Exercise 4A)**

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.

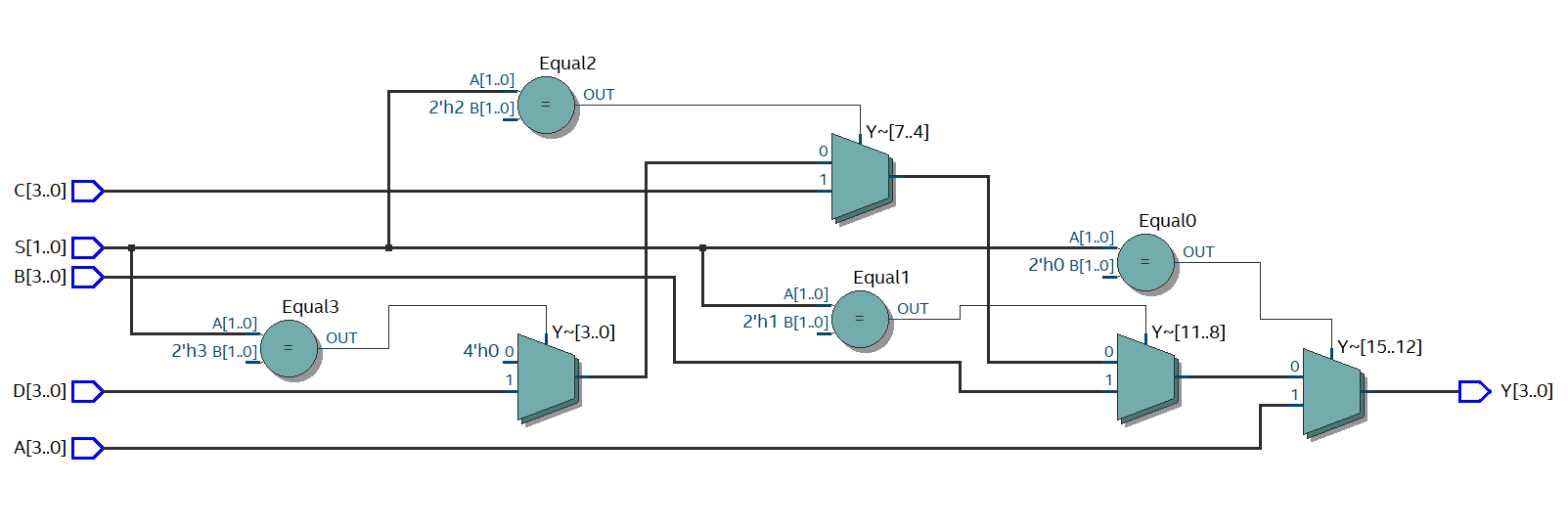
**Exercise 4B:**

****

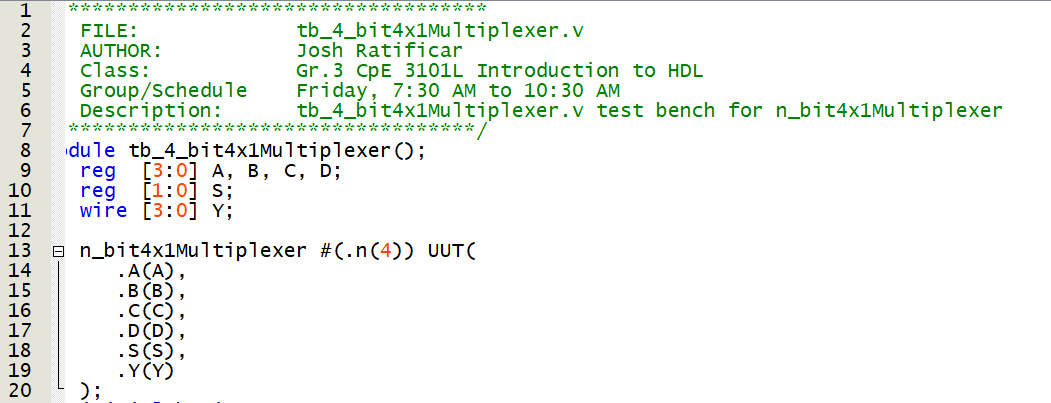
**Figure 2.0** – *n\_bit4x1Multiplexer.v Script*

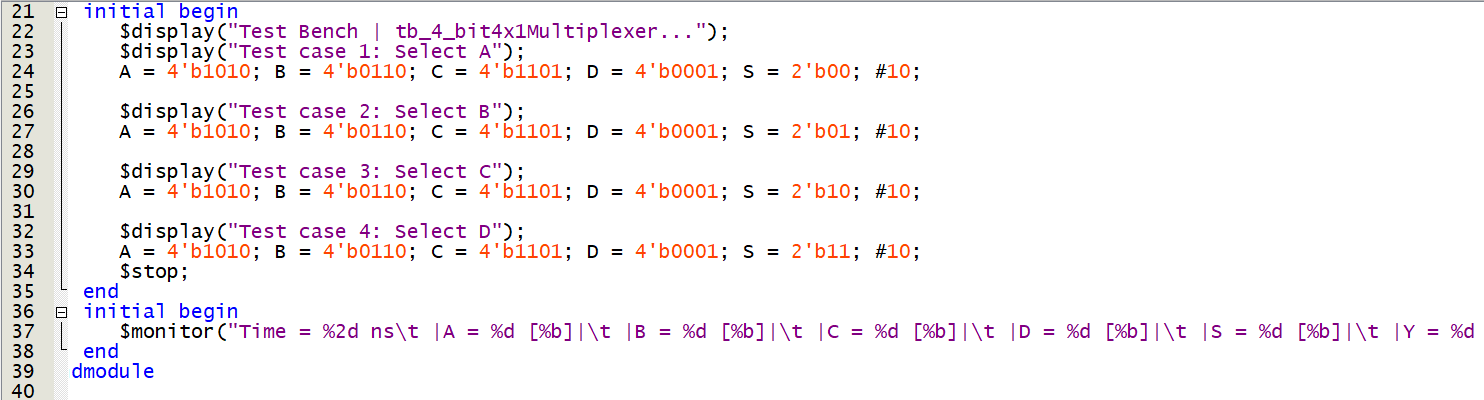
**

**Figure 2.1** – *n\_bit4x1Multiplexer.v Analysis and Elaboration Test Results*

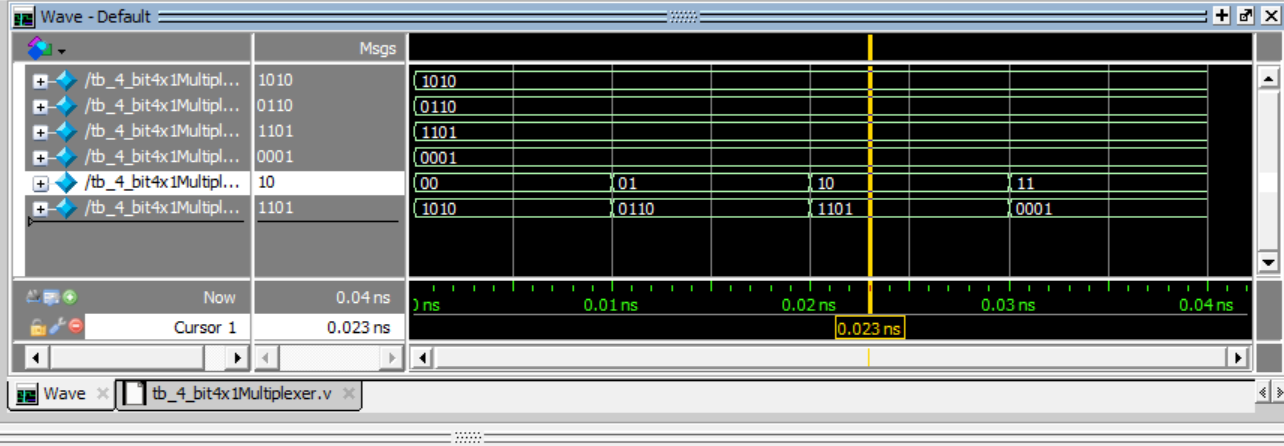
**

**Figure 2.2** – *n\_bit4x1Multiplexer RTL View Output*

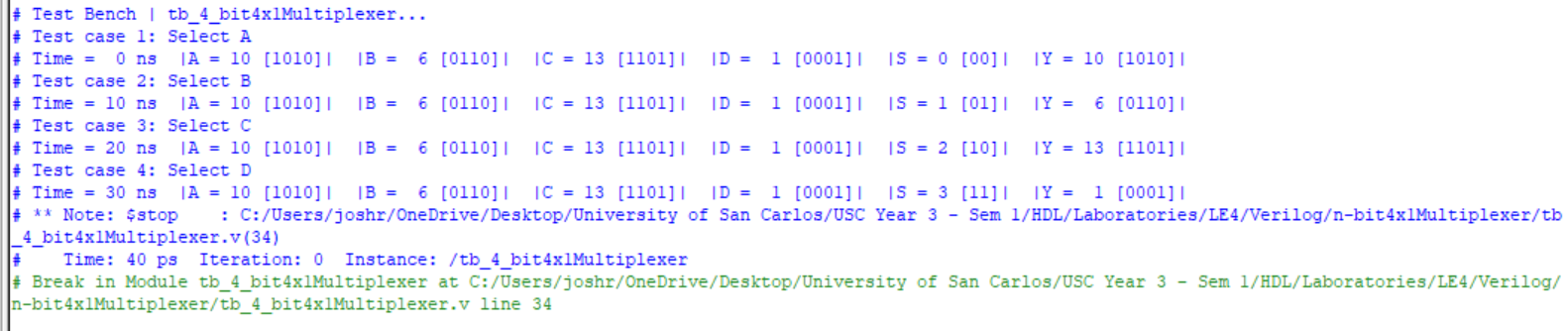




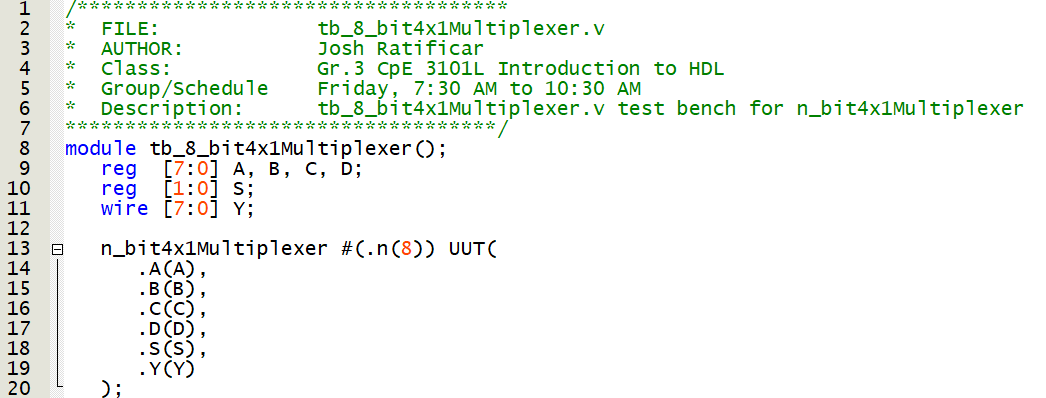
**Figure 3.0** – *tb\_4\_bit4x1Multiplexer.v Script for Testing Module*

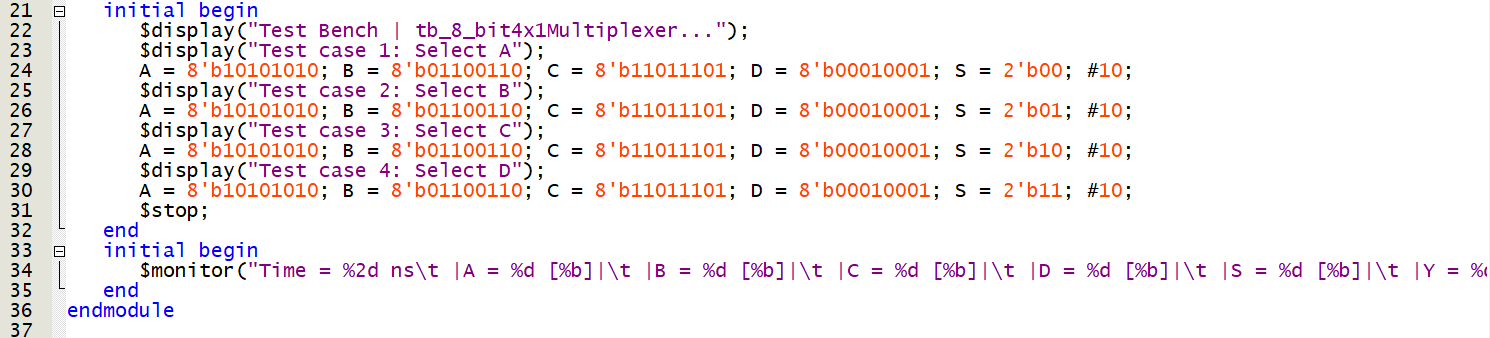
**

**Figure 3.1** – *tb\_4\_bit4x1Multiplexer RTL Simulation Output*

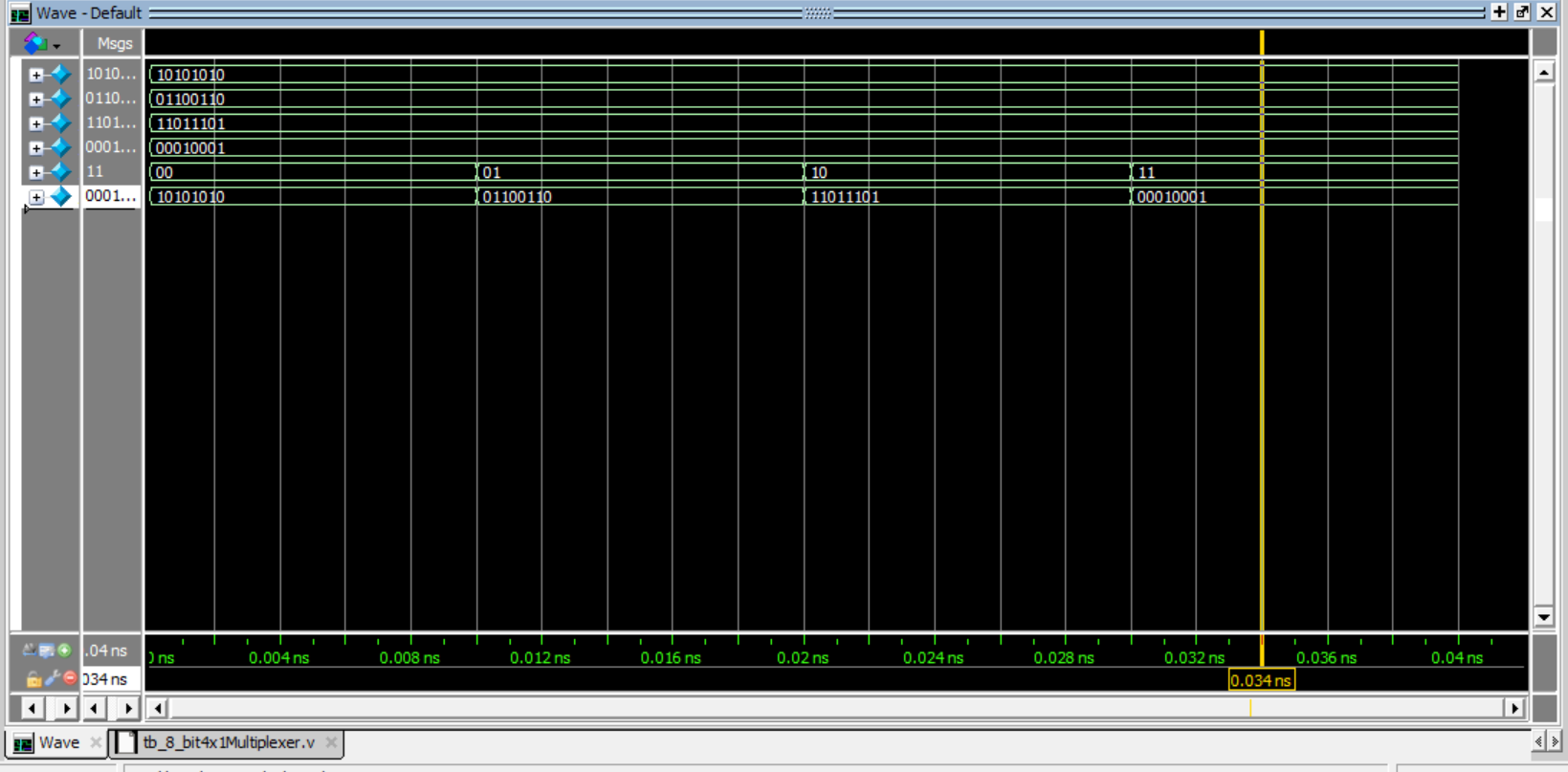
**

**Figure 3.2** – *tb\_4\_bit4x1Multiplexer Test Bench Monitor Output (Annotations to* ***Figure 3.1****)*

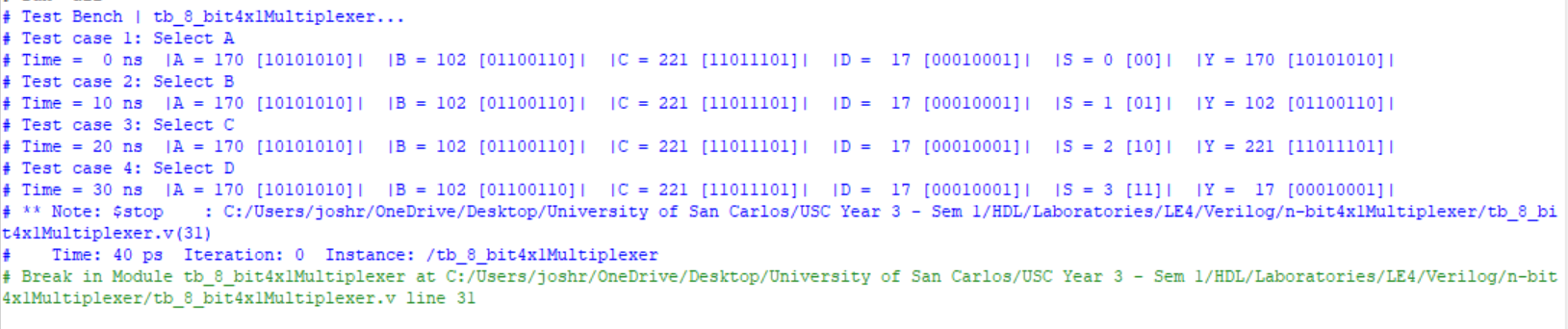




**Figure 4.0** – *tb\_8\_bit4x1Multiplexer.v Script for Testing Module*

**

**Figure 4.1** – *tb\_4\_bit4x1Multiplexer RTL Simulation Output*

**

**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.