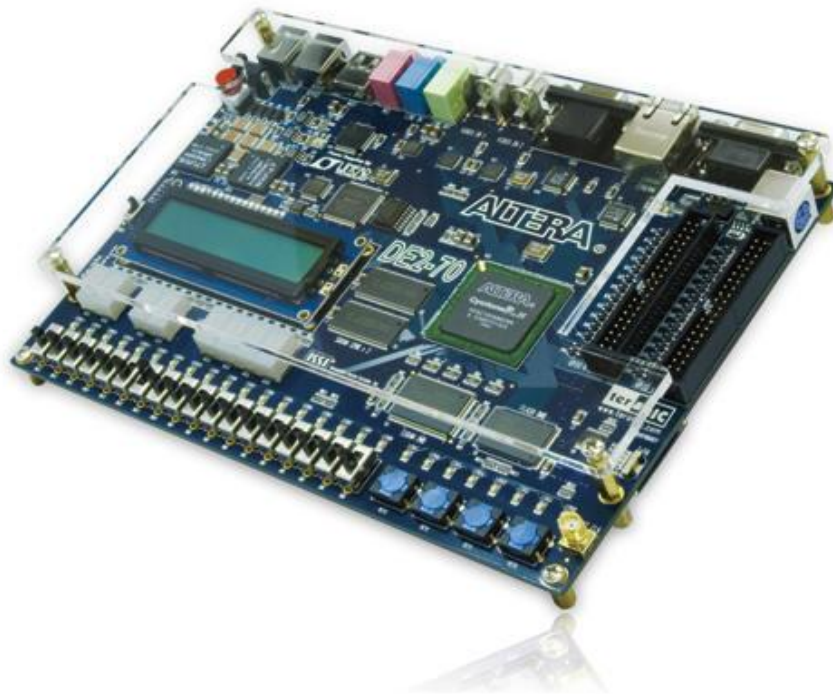




University of Jordan
Faculty of Engineering and Technology
Department of Computer Engineering
Digital Logic Laboratory 0907234

Labsheet5: Multiplexer and Demultiplexer Simulation



Name:

Student ID:

Section:

Description

In this experiment, you will design the simple two-bit TDM communication circuit shown in Figure 1.

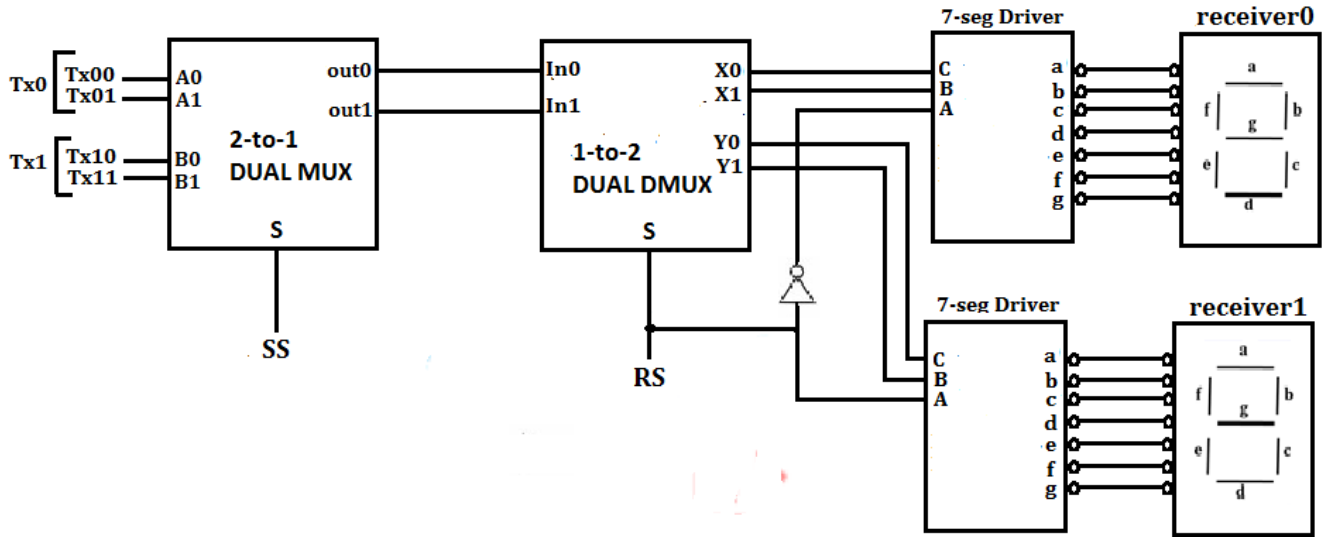


Figure1. 2- Bit TDM Communication Circuit

On the sender side, a 2-to-1 dual multiplexer is used to determine which transmitter (Tx0 or Tx1) is allowed to send data. On the receiver side, a 1-to-2 dual demultiplexer is used to determine which port (X which is connected to receiver0 or Y which is connected to receiver1) is used to receive data. Two 7-segment displays are connected to the receivers to display the value received (in decimal) if the receiver is selected. If the receiver is not selected, the (-) sign should be displayed on its 7-segment display.

The experiment is divided into four parts. First, you will implement the 2-to-1 dual multiplexer. Second, you will implement the 1-to-2 dual demultiplexer. In the third part, you will implement the 7-segment driver. Finally, you will instantiate these components to implement the overall circuit.

Part1:2-to-1 Dual MUX implementation.

a) Fill-in the truth table for the 2-to-1 (1-bit) MUX shown in Figure 2.

| S | I1 | I0 | out |
|---|----|----|-----|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

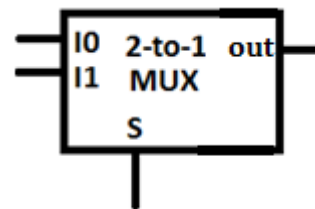


Figure 2.2-to-1 MUX

b) Determine the output equation for the 2-to-1 MUX.

out=

- c) Implement the **2-to-1 MUX** using Verilog code **structurally** in a module named *mux1* inside the “*mux1.v*” file. This module should have 3 inputs (I1, I0, and S) and one output (out).
- d) Implement the **2-to-1 dual MUX** using Verilog code **structurally** in a module named *dualmux* inside the “*dualmux.v*” file by instantiating two instances from the module *mux1* according to Figure 3. This module should have 5 inputs (A1, A0, B1, B0, and S) and 2 outputs (out1, out0).

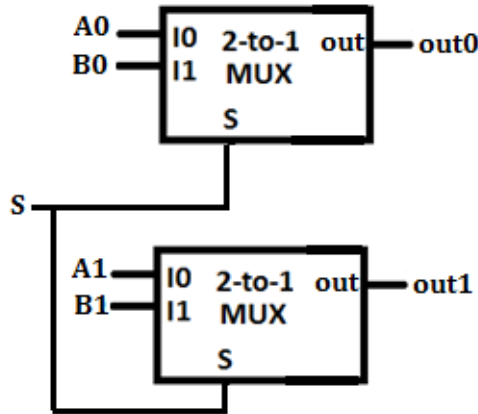


Figure 3.2-to-1 Dual MUX

Part2:1-to-2 Dual DMUX implementation.

- a) Fill-in the truth table for the 1-to-2 (1-bit)DMUX shown in Figure 4.

| S | D | Y | X |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

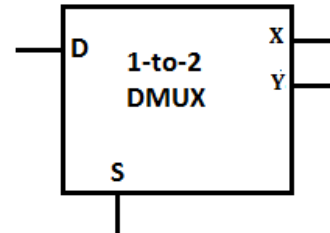


Figure 4.1-to-2DMUX

- b) Find the equations of outputs X and Y for the 1-to-2 DMUX module.

| |
|----|
| X= |
| Y= |

- c) Implement the **1-to-2 DMUX** using Verilog code **structurally** in a module named *dmux1* inside the “*dmux1.v*” file. This module should have 2 inputs (D and S) and 2 outputs (X and Y).
- d) Implement the **1-to-2 dual DMUX** using Verilog code **structurally** in a module named *dualdmux* inside the “*dualdmux.v*” file by instantiating two instances from the module *dmux1* according to Figure 5. This module should have 3 inputs (In1, In0, and S) and 4 outputs (Y1, Y0, X1, and X0).

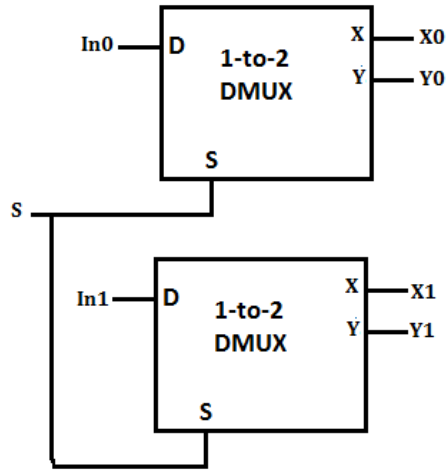


Figure 5. 1-to-2Dual DMUX

Part3: 7-segment Driver implementation.

The 7-segment driver of this circuit has three inputs: A, B, and C. Input A represents a control signal which identifies whether the receiver represented by the corresponding display is selected to receive data or not.

- If A=0, then regardless of the values of B and C, a(-) should be displayed to indicate that this receiver is not receiving data.
- If A=1, then the decimal value corresponding to the received bits (B and C) should be displayed.

a) Fill-in the truth table for **a common anode 7-segment driver** according to these specifications:

| A | B | C | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

b) Implement the driver using Verilog code **behaviorally** in a module named *segdriver* inside the "*segdriver.v*" file. This module should have 3-inputs: A, B and C (where A is the MSB) and 7-outputs (a, b, c, d, e, f, and g).

Part4: Final Circuit Implementation.

a) Implement the whole circuit which is shown in Figure 1 using Verilog code **structurally** in a module named *circuit1* inside the “**circuit1.v**” file. This module should have:

- 6-inputs (Tx01, Tx00, Tx11, Tx10, selection lines SS and RS) which are connected to switches.
- 14-outputs:
 - ❖ 7: (a, b, c, d, e, f, g)for(receiver 0) 7-segment display.
 - ❖ 7: (a1, b1, c1, d1, e1, f1, g1) for (receiver 1) 7-segment display.

b) Make the pins assignment for your inputs and outputs (as shown below).

| 7-Segment pins: Hex3 (RECEIVER 0): | | |
|--|------------|---------|
| a1 | oHEX3_D[0] | PIN_P6 |
| b1 | oHEX3_D[1] | PIN_P4 |
| c1 | oHEX3_D[2] | PIN_N10 |
| d1 | oHEX3_D[3] | PIN_N7 |
| e1 | oHEX3_D[4] | PIN_M8 |
| f1 | oHEX3_D[5] | PIN_M7 |
| g1 | oHEX3_D[6] | PIN_M6 |
| 7-Segment pins: Hex 4 (RECEIVER 1): | | |
| a | oHEX4_D[0] | PIN_P1 |
| b | oHEX4_D[1] | PIN_P2 |
| c | oHEX4_D[2] | PIN_P3 |
| d | oHEX4_D[3] | PIN_N2 |
| e | oHEX4_D[4] | PIN_N3 |
| f | oHEX4_D[5] | PIN_M1 |
| g | oHEX4_D[6] | PIN_M2 |

| Input Switch: | | |
|----------------------|--------|----------|
| Tx00 | iSW[1] | PIN_AB26 |
| Tx01 | iSW[2] | PIN_AB25 |
| Tx10 | iSW[5] | PIN_AC24 |
| Tx11 | iSW[6] | PIN_AC23 |
| SS | iSW[8] | PIN_AD24 |
| RS | iSW[9] | PIN_AE27 |

c) Download your Verilog program on FPGA and test it.