**EENG 2910 Project III – Digital System Design**

Design Project 2

Nathan A Ruprecht

Due 28 September 2016

# Introduction:

For our second project, we were required to design, implement, and test an 8-bit wide, 8-to-1 multiplexer and demultiplexer.

The multiplexer (MUX) is a combinational logic circuit designed to switch one of several input lines through to a single common output line by using a control signal to “select” which input is sent through. A demultiplexer (DEMUX) is a combinational logic circuit that is the opposite of the MUX. It takes a single input signal and selects one of the many data-output-lines.

# Theory of Operation and Explanation of the Design:

The 3 things needed for a MUX is an input, output, and select. The 2-to-1 MUX discussed in the PowerPoint lecture shows that basic thought of a MUX.

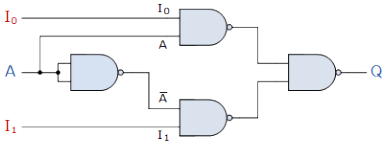
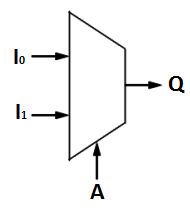


Figure : MUX Symbol

Figure : MUX Logic Circuit

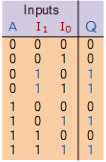


Table : MUX Truth Table

With “I” being the input, “A” selects which part of I is passed through to the output. We had an in-class assignment to code a MUX from scratch to drive home what is happening and how to implement one. One way to do it is to make a NAND gate component to make the above circuit for a 2-to-1 MUX. As shown, when A is 0, the output reflects I1. While A being 1, I0 is passed through to Q. When coding, I took advantage of a case statement that depended on the select variable. However many inputs I have, I find n in 2^n = (number of inputs) to know how long the select variable needs to be. This assignment required 8 inputs so the “A” will be a 3-bit number.

# Experimental Results:

The code itself was not too terrible. After the in-class exercise, we had a good idea on how a MUX operated. We were also given example code of a 4-to-1 MUX and DEMUX. I used this example code and just expanded to show an 8-to-1 MUX, then DEMUX.

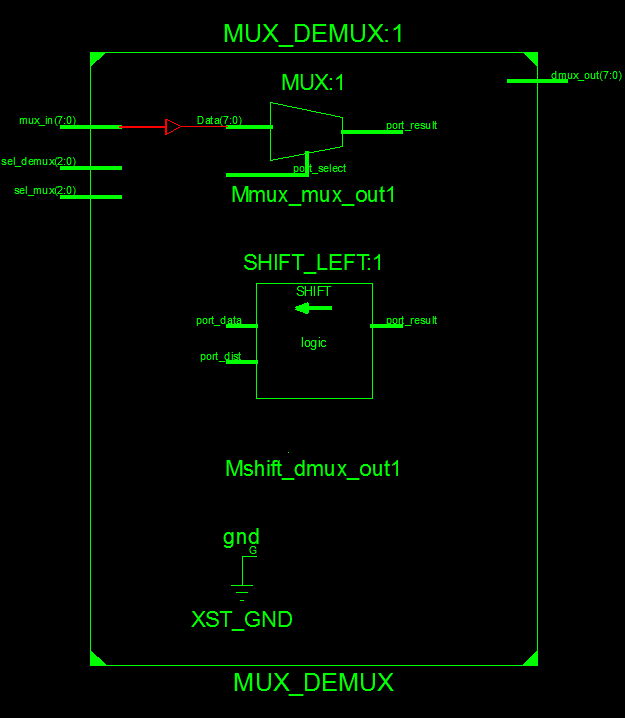


Figure : MUX/DEMUX Schematic

Figure 3 shows the schematic of what the code is doing. The input (mux\_in) is an 8-bit number, MUX and DEMUX select variables (sel\_mux and sel\_demux) are each 3 bits, and the output being an 8-bit number. Using the examples 4-to-1 code, this expanded code does the same thing by taking the 8-bit input, selecting a certain bit, and outputting an 8-bit number with all 0 except for the selected bit in its bit position.

For the text bench, I hard coded examples to simulate sending an 8-bit wide number completely through a MUX and a DEMUX. I tested it numerous times and grabbed screenshots twice to show very different inputs to show the same end result.

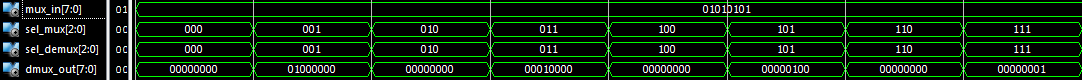


Figure 4: Test 1

My first test was sending “01010101” through the MUX and DEMUX. I went down the line and had the select variables choose each bit of the input. The output showed that selected bit value, in its correct bit position, with the rest of the 8-bit output being 0. “000” selecting the MSB down to “111” being the LSB.

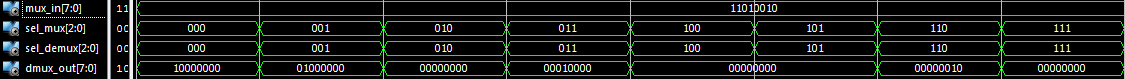


Figure 5: Test 2

I chose “11010010” for the second test since there was not a pattern in the values. Similar as test 1, I went down the line and had the select variables choose each bit of the input. Again, the output of the DEMUX showed the correct bit choice, in the correct bit position, with the rest of the output values of 0.

Problems I had at first was the select variable of the MUX being different than the select variable on the DEMUX. Or having the test bench coded for the DEMUX select to equal the MUX select. Although the output came out with the correct values, it was shifted by a bit. Simple enough fix to hard-code values for each select variable.

# Conclusion:

The goal of the lab was to design, implement, and test an 8-bit wide, 8-to-1 multiplexer and demultiplexer. In the design process, I understood how a MUX/DEMUX was made using logic gates. How to make a simple MUX using a case statement. I expanded on given code to make an 8-bit wide, 8-to-1 MUX/DEMUX. I understood and implement the MUX/DEMUX to show an output the correctly selected the bit value of an input and successfully makes the rest of the 8-bit number 0 while holding that input value the same.

# References:

Figure 2 and Table 1 came from the lecture slides.

Figure 1 was made in paint.

Figures 3-5 are screenshots.

# Source Code:

----------------------------------------------------------------------------------

-- Company:

-- Engineer: Nathan A Ruprecht

--

-- Create Date: 11:46:09 09/21/2016

-- Design Name:

-- Module Name: MUX\_DEMUX - Behavioral

-- Project Name:

-- Target Devices:

-- Tool versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity MUX\_DEMUX is

port(

mux\_in : in STD\_LOGIC\_VECTOR(7 downto 0);

sel\_mux : in STD\_LOGIC\_VECTOR(2 downto 0);

--mux\_out : inout STD\_LOGIC;

--dmux\_in : inout STD\_LOGIC;

sel\_demux : in STD\_LOGIC\_VECTOR(2 downto 0);

dmux\_out : out STD\_LOGIC\_VECTOR(7 downto 0)

);

end MUX\_DEMUX;

architecture Behavioral of MUX\_DEMUX is

signal mux\_out,dmux\_in : STD\_LOGIC;

begin

mux : process (mux\_in,sel\_mux) is

begin

case sel\_mux is

when "000" => mux\_out <= mux\_in(7);

when "001" => mux\_out <= mux\_in(6);

when "010" => mux\_out <= mux\_in(5);

when "011" => mux\_out <= mux\_in(4);

when "100" => mux\_out <= mux\_in(3);

when "101" => mux\_out <= mux\_in(2);

when "110" => mux\_out <= mux\_in(1);

when others => mux\_out <= mux\_in(0);

end case;

end process mux;

dmux\_in<=mux\_out;

demux : process (dmux\_in,sel\_demux) is

begin

case sel\_demux is

when "000" => dmux\_out <= dmux\_in & "0000000";

when "001" => dmux\_out <= '0' & dmux\_in & "000000";

when "010" => dmux\_out <= "00" & dmux\_in & "00000";

when "011" => dmux\_out <= "000" & dmux\_in & "0000";

when "100" => dmux\_out <= "0000" & dmux\_in & "000";

when "101" => dmux\_out <= "00000" & dmux\_in & "00";

when "110" => dmux\_out <= "000000" & dmux\_in & '0';

when others => dmux\_out <= "0000000" & dmux\_in;

end case;

end process demux;

end Behavioral;

--------------------------------------------------------------------------------

-- Company:

-- Engineer: Nathan A Ruprecht

--

-- Create Date: 10:26:37 09/28/2016

-- Design Name:

-- Module Name: E:/2910/DP2/Code/MUX\_DEMUX/tb\_MUX\_DEMUX.vhd

-- Project Name: MUX\_DEMUX

-- Target Device:

-- Tool versions:

-- Description:

--

-- VHDL Test Bench Created by ISE for module: MUX\_DEMUX

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

-- Notes:

-- This testbench has been automatically generated using types std\_logic and

-- std\_logic\_vector for the ports of the unit under test. Xilinx recommends

-- that these types always be used for the top-level I/O of a design in order

-- to guarantee that the testbench will bind correctly to the post-implementation

-- simulation model.

--------------------------------------------------------------------------------

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--USE ieee.numeric\_std.ALL;

ENTITY tb\_MUX\_DEMUX IS

END tb\_MUX\_DEMUX;

ARCHITECTURE behavior OF tb\_MUX\_DEMUX IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT MUX\_DEMUX

PORT(

mux\_in : IN std\_logic\_vector(7 downto 0);

sel\_mux : IN std\_logic\_vector(2 downto 0);

sel\_demux : IN std\_logic\_vector(2 downto 0);

dmux\_out : OUT std\_logic\_vector(7 downto 0)

);

END COMPONENT;

--Inputs

signal mux\_in : std\_logic\_vector(7 downto 0) := (others => '0');

signal sel\_mux : std\_logic\_vector(2 downto 0) := (others => '0');

signal sel\_demux : std\_logic\_vector(2 downto 0) := (others => '0');

--Outputs

signal dmux\_out : std\_logic\_vector(7 downto 0);

-- No clocks detected in port list. Replace <clock> below with

-- appropriate port name

BEGIN

-- Instantiate the Unit Under Test (UUT)

uut: MUX\_DEMUX PORT MAP (

mux\_in => mux\_in,

sel\_mux => sel\_mux,

sel\_demux => sel\_demux,

dmux\_out => dmux\_out

);

-- Clock process definitions

-- Stimulus process

stim\_proc: process

begin

mux\_in <= "11010010";

sel\_mux <= "000";

sel\_demux <= "000";

wait for 100 ns;

sel\_mux <= "001";

sel\_demux <= "001";

wait for 100 ns;

sel\_mux <= "010";

sel\_demux <= "010";

wait for 100 ns;

sel\_mux <= "011";

sel\_demux <= "011";

wait for 100 ns;

sel\_mux <= "100";

sel\_demux <= "100";

wait for 100 ns;

sel\_mux <= "101";

sel\_demux <= "101";

wait for 100 ns;

sel\_mux <= "110";

sel\_demux <= "110";

wait for 100 ns;

sel\_mux <= "111";

sel\_demux <= "111";

wait for 100 ns;

sel\_mux <= "000";

sel\_demux <= "000";

wait;

end process;

END;