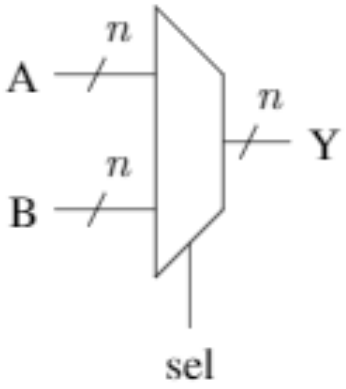
Lab 2 Report

Multiplexors

CENG 342 Digital Systems



Samuel Donovan

Samantha Pfeiffer

2021-02-01

OVERVIEW

The purpose of this lab is to become familiar with synthesizing and simplifying complex logic in VHDL, by writing and simulating simple 4-bit 2-to-1 and 8-to-1 multiplexors.

--- N-BIT X-TO-1 MULTIPLEXOR ---

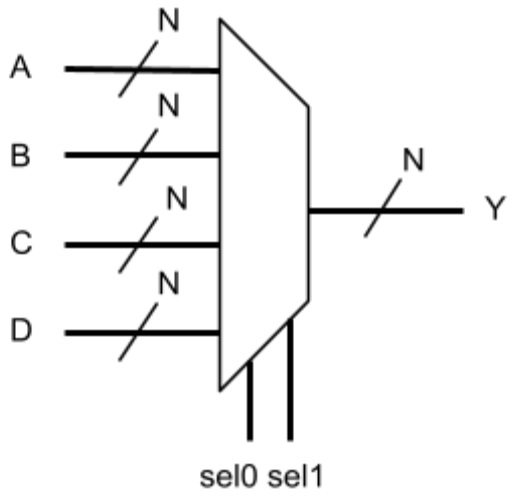
Multiplexors have a single output, **Y**, and *X* inputs, along with log_2X selector inputs. Depending on the value of the selector, a different input is shorted to **Y**. truth table for a 4-to-1 multiplexor (bus size *N* is undefined). Two **sel** inputs are The number *N* determines the bus size of the inputs and **Y**. **Table 1** shows the required as $\log_2 4=2$.

Table 1 4-to-1 Multiplexor

Inputs
A
B
C
D

sel_1	sel_0	Y
0	0	A
0	1	B
1	0	C
1	1	D

Figure 1 N-Bit 4-to-1 Multiplexer



DESIGN

All designs were made for a Nexys A7-100T (xc7a100tcsg324-1) FPGA board in VHDL using Vivado 2020.2.

--- 4-BIT 2-TO-1 MULTIPLEXOR ---

The design of 4-bit 2-to-1 multiplexor was simple. The inputs include two 4-bit standard logic vectors, A and B, and a 1-bit standard logic value, SEL. The output, Y, is a 4-bit standard logic vector as well. In the architecture a `when else` statement was used to set the value of Y.

To test the design A was set to the hexadecimal value 0x0a, and B to 0x0b. **Figure 2** shows that when SEL is 0, Y is a as expected. Similarly, Y is b when SEL is 1.

--- 4-BIT 8-TO-1 MULTIPLEXOR ---

The design of 4-bit 8-to-1 multiplexor was more complex. The inputs include eight 4-bit standard logic vectors, A to H, and a 3-bit standard logic vector, SEL. The output, Y, is a 4-bit standard logic vector as well. In the architecture `when` statements were used inside a `with select` statement to set the value of Y with respect to the value of SEL.

To test the design inputs A through H were set to constant values 1 through 8. A for loop was then used to cycle through the values of SEL. **Figure 3** shows that changing the value of SEL causes the expected changes in the output, Y.

--- CONSTRAINTS ---

To test the 2-to-1 multiplexor on our FPGA, we set the constraints file up to have all 4 bits of both bus A and B hooked to switches, with the output bus Y connected to four LEDs. SEL was also hooked to a switch.

SIMULATION RESULTS

Figure 2 4-bit 2x1 Multiplexor Testbench Simulation Result

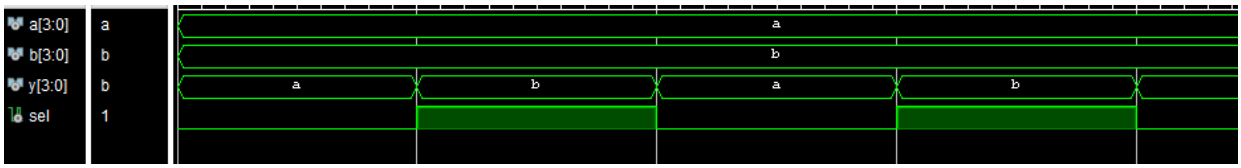


Figure 3 4-bit 8x1 Multiplexor Testbench Simulation Result

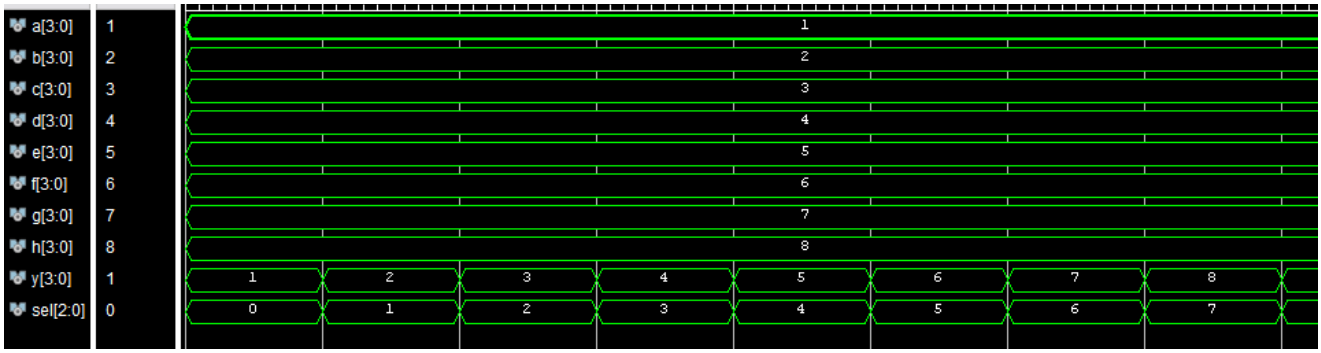
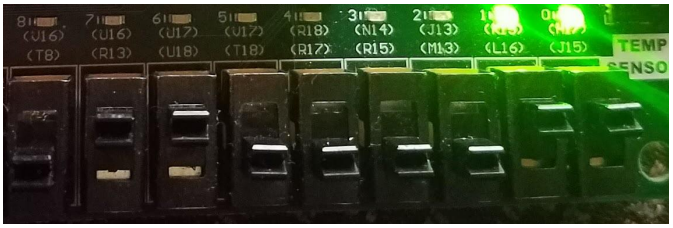


Figure 4 Picture of Programmed Board with switches SEL, B, A from left to right

SEL = ‘1’



SEL = ‘0’

ISSUES ENCOUNTERED

We were initially confused on how to test our code on the physical FPGA as we thought a USB cable was missing from our assigned kit. However, after contacting the instructor and finding a spare micro-USB cable to connect the device, we were able to push our code to it.

We also encountered an issue with the constraints in our XDC file resulting from us forgetting to declare a physical switch for the SEL input, causing a failure when generating the bitstream. The corrected XDC file, which can be found in **Appendix E**, allowed the device to work as expected, as seen in **Figure 4**.

CONCLUSIONS

The lengthy logic of the multiplexors was able to be vastly simplified by using VHDL's `when` statements (For example, see **Appendix A**). This was especially useful for the 8-to-1 multiplexer, as the raw logic equation for such a device is extremely long for a relatively simple task.

While the simplification of logic enables maintainability and legibility of our code, we question whether there is extra silicon being used or an increase in propagation delay as a result of mis-optimization.

APPENDIX A: 4-bit 2x1 Multiplexor VHDL Code (mux_2x1.vhdl)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux_2x1 is
     Port (
          SEL: in std_logic;
          A, B: in std_logic_vector(3 downto 0);
          Y: out std_logic_vector(3 downto 0)
     );
end mux_2x1;

architecture Behavioral of mux_2x1 is

begin
     Y <= A when SEL='0' else
     B;
end Behavioral;
```

APPENDIX B: 4-bit 2x1 Multiplexor Testbench VHDL Code (mux_2x1_testbench.vhdl)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;


entity mux_2x1_testbench is
end mux_2x1_testbench;

architecture Behavioral of mux_2x1_testbench is
     signal a, b, y: std_logic_vector(3 downto 0);
     signal sel: std_logic;
begin
     uut: entity work.mux_2x1 (behavioral)
     port map(A=>a, B=>b, Y=>y, SEL=>sel);

test: process
    variable i, j: integer;
     begin
          sel <= '0';
          a <= "0000";
          b <= "0000";
          for i in 1 to 15 loop
              for j in 1 to 15 loop
                  b <= std_logic_vector(TO_UNSIGNED(j,4));
                  wait for 20ns;
              end loop;
              a <= std_logic_vector(TO_UNSIGNED(i,4));
              wait for 20ns;
          end loop;

          sel <= '1';
          for i in 1 to 15 loop
              for j in 1 to 15 loop
                  b <= std_logic_vector(TO_UNSIGNED(j,4));
                  wait for 20ns;
              end loop;
              a <= std_logic_vector(TO_UNSIGNED(i,4));
```

```
            wait for 20ns;
        end loop;
    end process;
end Behavioral;
```

APPENDIX C: 4-bit 8x1 Multiplexor VHDL Code (mux_8x1.vhdl)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity mux_8x1 is
     Port (
          SEL: in std_logic_vector(2 downto 0);
          A,B,C,D,E,F,G,H: in std_logic_vector(3 downto 0);
          Y: out std_logic_vector(3 downto 0)
     );
end mux_8x1;

architecture Behavioral of mux_8x1 is
begin

with SEL select
    Y <= A when "000",
         B when "001",
         C when "010",
         D when "011",
         E when "100",
         F when "101",
         G when "110",
         H when others;

end Behavioral;
```

APPENDIX D: 4-bit 8x1 Multiplexor Testbench VHDL Code (mux_8x1_testbench.vhdl)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity mux_8x1_testbench is
end mux_8x1_testbench;

architecture Behavioral of mux_8x1_testbench is
     signal a,b,c,d,e,f,g,h,y: std_logic_vector(3 downto 0);
     signal sel: std_logic_vector(2 downto 0);
     begin
     uut: entity work.mux_8x1 (behavioral)
         port
     map(A=>a,B=>b,C=>c,D=>d,E=>e,F=>f,G=>g,H=>h,Y=>y,SEL=>sel)
     ;

     test: process
     variable i: integer;
     begin
          a <= "0001";
          b <= "0010";
          c <= "0011";
          d <= "0100";
          e <= "0101";
          f <= "0110";
          g <= "0111";
          h <= "1000";
          sel <= "000";
          for i in 0 to 8 loop
              sel <= std_logic_vector(TO_UNSIGNED(i,3));
              wait for 20ns;
          end loop;
     end process;
end Behavioral;
```

APPENDIX E: Constraints (constraints.xdc)

```
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 }
[get_ports { A[0] }];
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 }
[get_ports { A[1] }];
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 }
[get_ports { A[2] }];
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 }
[get_ports { A[3] }];
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 }
[get_ports { B[0] }];
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 }
[get_ports { B[1] }];
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 }
[get_ports { B[2] }];
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 }
[get_ports { B[3] }];
set_property -dict { PACKAGE_PIN T8   IOSTANDARD LVCMOS33 }
[get_ports { SEL }];


set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 }
[get_ports { Y[0] }];
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 }
[get_ports { Y[1] }];
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 }
[get_ports { Y[2] }];
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 }
[get_ports { Y[3] }];
```