Project 1
ECE 485: Computer Organization and Design

Alan Palayil
Professor: Won-Jae Yi

Acknowledgement: I acknowledge all works including figures, codes and writings belong to me and/or persons who are referenced. I understand if any similarity in the code, comments, customized program behavior, report writings and/or figures are found, both the helper (original work) and the requestor (duplicated/modified work) will be called for academic disciplinary action.

Electronic Signature: ***Alan Palayil*** (10/07/2022)

Contents

Abstract:

This project is used to get us familiar with VDHL programming. We must create an 8:1-bit multiplexer using both behavioral and structural model which meant simply mapping out what the selected inputs would output data based on the data input. Since the multiplexer was designed in the essence of a truth table. Only the inputs and outputs were considered, so we only cared about the function rather than design. This is the behavioral model using simple if else conditions to get the same results as the truth table.

Then I created a structural model of an 8:1 -bit multiplexer which meant I considered the uses of smaller multiplexers to combine them into the 8:1-bit multiplexer. I used two 4:1-bit multiplexer, a 2:1-bit multiplexer, and combined them to create an 8:1-bit multiplexer.

After compiling both the models, I created a testbench file that would utilize the behavioral model and the structural model and test sample input and make sure the output is correct.

Introduction:

The purpose of this Project 1 is to prepare you towards Project 2 and Project 3, which you will design and implement your own custom 32-bit RISC processor, a stripped-down MIPS processor. The goal of this project is to get familiar with the VHDL programming, as well as the simulation environment such as ModelSim or Vivado.

In this project, you will design and implement an eight-input multiplexer in both behavioral model and structural model using VHDL. After designing the both the models we test them using our own testbench.

Background:

We have discussed two different models that you could implement using VDHL. Below is the description of the approach I take for both, the structural and behavioral models, along with the testbench. I also will discuss my experience between the two environments: Vivado and ModelSim.

i. Description of a VHDL structural model

The structural model considers the gate implementation for a multiplexer to work. In my case, I used two 4:1-bit multiplexer and a 2:1-bit multiplexer to achieve the final 8:1-bit multiplexer.

ii. Description of a VHDL behavioral model

The behavioral model considers only the function (or behavior) of the multiplexer. In essence, it only cares about what's the input and what will be the output based on the input. It ignores how the actual multiplexer function. I use what case to depict the 8:1-bit multiplexer behavior.

iii. Description of a VHDL testbench

The testbench uses sample data and then inputs it into the behavioral model and the structural model and sees if the output is correct based on the sample data. To check if it is correct, we must simply look at the waveform and check it ourselves, if the correct output is there based on the inputs.

iv. Description of your VHDL simulator and environment

Vivado is an environment in which we can code in various HDL languages such as Verilog and VDHL to create circuit implementations that can be compiled, and the files can be checked to see if there are any errors to fix. We can also create simulated schematics and use the testbench files to simulate the output using waveform generators.

v. Description of the choice of simulator and the difference between Vivado and ModelSim

ModelSim is an environment in which you can code in VHDL to create circuit implementations and simulate outputs using waveform generators. I comparison to Vivado, ModelSim requires a lot more time to set up all the features. Will looking over tutorials I did see that ModelSim provides a chip simulation which I could not find in Vivado.

System Design:

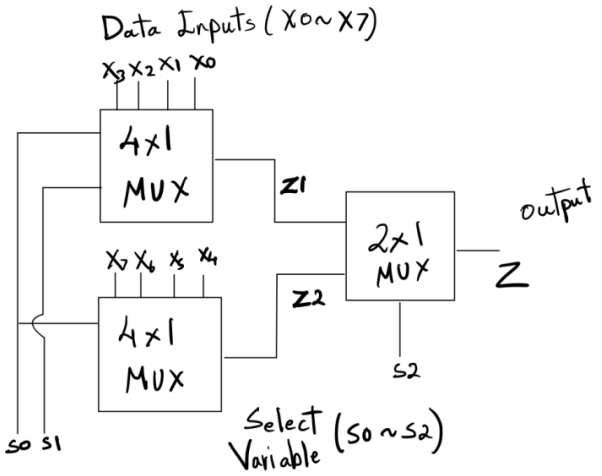
An 8:1-bit multiplexer consists of eight data inputs from (I0~I7), with three input select lines (S0~S2) and a single output line (Z). Depending on the select signal combination, the multiplexer selects the input (I).

S2 S1 S0	I7	I6	I5	I4	I3	I2	I1	I0	Z
000	X	X	X	X	X	X	X	I0	I0
001	X	X	X	X	X	X	I1	X	I1
010	X	X	X	X	X	I2	X	X	I2
011	X	X	X	X	I3	X	X	X	I3
100	X	X	X	I4	X	X	X	X	I4
101	X	X	I5	X	X	X	X	X	I5
110	X	I6	X	X	X	X	X	X	I6
111	I7	X	X	X	X	X	X	X	I7

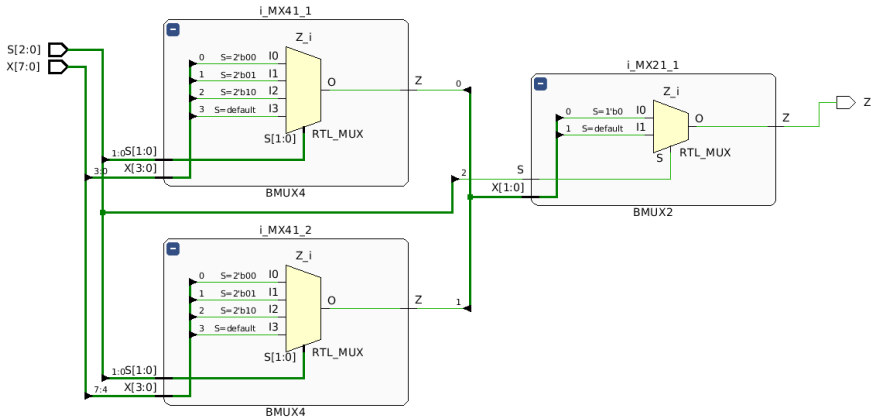
Functional Table for an 8:1-bit multiplexer

i. Description of your eight-input multiplexer using structural model

To create the structural model of the 8:1-bit multiplexer, I first started by creating an entity for the multiplexer and then assigned it 8 data input variables (X0~X7), 3 select input variables (S0~S2), and an output variable (Z). From there on, I proceeded to create an architecture for the structural based multiplexer in which I considered the overall design would implement smaller multiplexers. I first created a 2:1-bit multiplexer component and two 4:1-bit multiplexer components. I then initialized the signals that would be used in these multiplexers (basically the data, select inputs, and final output). The two 4:1-bit multiplexers contain the data inputs with two select signals. The first 4:1-bit multiplexer contains data inputs (X0~X3), select inputs (S0~S1) with the output (Z1) goes in as the first input for the 2:1-bit multiplexer. The second 4:1-bit multiplexer contains data inputs (X4~X7), select inputs (S0~S1) with the output (Z2) goes in as the second input for the 2:1-bit multiplexer. The 2:1-bit multiplexer gets data input (Z1 and Z2), select signal (S2) with the outcome (Z).



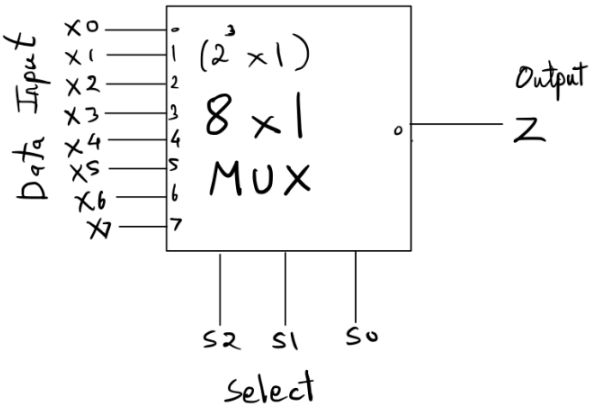
Block Diagram of the Structural Model of the 8:1-bit Multiplexer



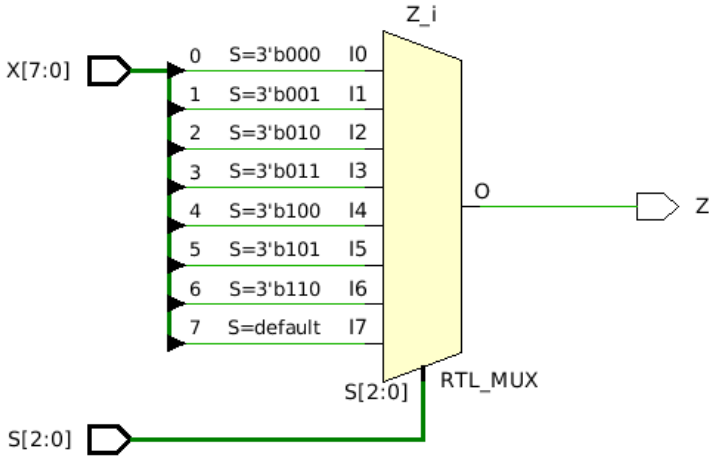
Schematic of the 8:1-bit structural multiplexer made by Vivado

ii. Description of your eight-input multiplexer using behavioral model

To create the behavioral model of the 8:1 multiplexer, I first started by creating an entity for the multiplexer and then assigned it 8 data input variables (X0~X7), 3 select input variables (S0~S2), and an output variable (Z). From here, I proceeded to create an architecture for the behavioral based multiplexer in which I created a when statement to go through every case of the select inputs to determine what the output would be given some input data. It was basically a mapping table. The code is designed on Vivado and once we Run Synthesis, are can also visualize a schematic of the when statement into a circuit.



Block Diagram of the Behavioral Model of the 8:1-bit Multiplexer



Schematic of the 8:1-bit behavioral multiplexer made by Vivado

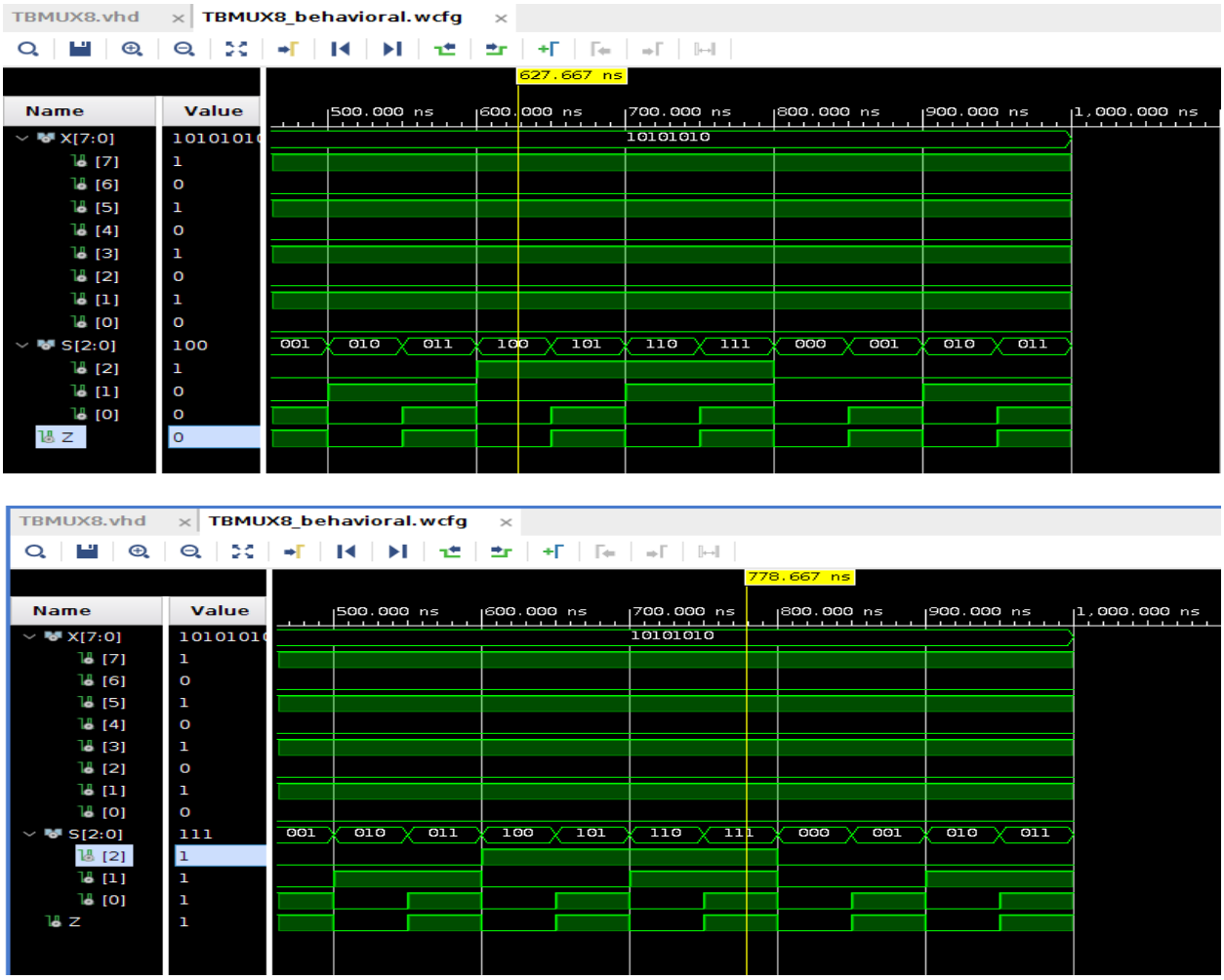
iii. Description of your VHDL testbench code

To create the testbench, I created the entity of testbench and then added the components of the behavioral and structural models separately. I initialized all the signals (data inputs and select inputs) as '0'. Then I created a port map in which I mapped each signal input to the data inputs for the model (behavioral or structural). After which I created a sample data inputs which were randomly chosen to be 1s or 0s for each input. Then I went through each select variation and had it wait for 50 ns every time. The purpose of this was so that the waveform would show the

outputs clearly given these inputs. The select variations simply counted in binary from 000 to 111. Then the output would be whatever data input was given for each count in the binary.

Simulation Results and Discussion:

i. Screenshots of your test cases



Random Marker on 8:1-bit Multiplexer-Behavioral Model. (Figure 1.)

Random Marker on 8:1-bit Multiplexer-Structural Model. (Figure 2.)

ii. Descriptions of each screenshot

- The First two sets of screenshots are the test case for the Behavioral model. I chose random data input with the alteration between 0 and 1 for the inputs while the selects count from 0 to 8 in binary. As you can see for the Z output at the bottom, it fluctuates between 0 and 1 depending on the input.
- The second set of screenshots are the test case for the Structural model. I chose the same random data input with the alteration between 0 and 1 for the inputs while the selects count from 0 to 8 in binary. As the selects counted to 8 in binary, you can see that the Z output at the bottom changed based on the input.

iii. Discussion of any issues faced and improvements that could be made

- While writing the VDHL code for the 8:1-bit multiplexer, I can write the structural model in various other ways such as building it completely from gates like AND, OR, and INV to create the multiplexer.
- I can also make use of other ways to create a behavioral model for the 8:1-bit multiplexer using if-else statements, cases, and straight up assignments.

Conclusion:

I learned how to use the two simulators ModelSim and Vivado with VHDL to create an 8:1-bit multiplexer both behavioral and structural models. I also learned how to perform a simulation using the waveform to test my code in the simulators.

List of References:

- VHDL Tutorial by van der Spiegel
- VHDL Tutorial by Ashenden
- ModelSim Tutorial
- Vivado Tutorial
- https://technobyte.org/verilog-multiplexer-4x1/
- https://stackoverflow.com/questions/53329810/vhdl-4-bit-multiplier-based-on-4-bit-adder
- https://www.youtube.com/playlist?list=PLwxoxZ8sTlocIF6Ezqgs1S_rAflMjd6IV

Appendix:

i. Entire Source Code of project with comments

Behavioral Model code of 8:1-bit Multiplexer:

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

--Create the entity for behavioral model initializing inputs and outputs.
entity BMUX8 is
    Port ( X: in STD_LOGIC_VECTOR (7 downto 0); --Input terminals
           S: in STD_LOGIC_VECTOR (2 downto 0); --Select (2^3 = 8 for 8 inputs
)
           Z : out STD_LOGIC); --output terminal
end BMUX8;

--Creating the architecture for how the multiplexer will operate
architecture Behavioral of BMUX8 is
begin
    --Declare what the variables are being used in the process
    with S select
        Z<= X(0) when "000",
            X(1) when "001",
            X(2) when "010",
```

```
            X(3) when "011",
            X(4) when "100",
            X(5) when "101",
            X(6) when "110",
            X(7) when others;

end Behavioral;
```

Structural Model code of 8:1-bit Multiplexer:

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

--Create the entity for structural model initializing inputs and outputs.
entity SMUX8 is
    Port ( X: in STD_LOGIC_VECTOR (7 downto 0); --Input terminals
           S: in STD_LOGIC_VECTOR (2 downto 0); --Select (2^3 = 8 for 8 inputs
)
           Z : out STD_LOGIC); --output terminal
end SMUX8;

--Creating the architecture for how the multiplexer will operate
architecture Structural of SMUX8 is

--2:1 Multiplexer component
component BMUX2
    Port (
           X: in STD_LOGIC_VECTOR (1 downto 0); --Input terminals
           S: in STD_LOGIC; --Select
           Z : out STD_LOGIC); --output terminal
end component;

--4:1 Multiplexer component
component BMUX4
    Port (
           X: in STD_LOGIC_VECTOR (3 downto 0); --Input terminals
           S: in STD_LOGIC_VECTOR (1 downto 0); --Select
           Z : out STD_LOGIC); --output terminal
end component;

--Signals
signal Z_s: STD_LOGIC_VECTOR (1 downto 0);

begin

i_MX41_1: BMUX4 port map (
```

```
    X(0)=> X(0),
    X(1)=> X(1),
    X(2)=> X(2),
    X(3)=> X(3),
    S(0)=> S(0),
    S(1)=> S(1),
    Z=> Z_s(0)
);

i_MX41_2: BMUX4 port map (
    X(0)=> X(4),
    X(1)=> X(5),
    X(2)=> X(6),
    X(3)=> X(7),
    S(0)=> S(0),
    S(1)=> S(1),
    Z=> Z_s(1)
);
i_MX21_1: BMUX2 port map (
    X(0)=> Z_s(0),
    X(1)=> Z_s(1),
    S=> S(2),
    Z=> Z
);

end Structural;
```

Behavioral Model code of 2:1-bit Multiplexer:

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

entity BMUX2 is
    Port (
           X: in STD_LOGIC_VECTOR (1 downto 0); --Input terminals
           S: in STD_LOGIC; --Select
           Z : out STD_LOGIC); --output terminal
end BMUX2;

architecture Behavioral of BMUX2 is
begin
    with S select
        Z<= X(0) when '0',
            X(1) when others;

end Behavioral;
```

Behavioral Model code of 4:1-bit Multiplexer:

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

entity BMUX4 is
Port (
           X: in STD_LOGIC_VECTOR (3 downto 0); --Input terminals
           S: in STD_LOGIC_VECTOR (1 downto 0); --Select
           Z : out STD_LOGIC); --output terminal
end BMUX4;

architecture Behavioral of BMUX4 is
begin
    with S select
        Z<= X(0) when "00",
            X(1) when "01",
            X(2) when "10",
            X(3) when others;

end Behavioral;
```

ii. Entire Testbench Code with comments used to verify design

Testbench Code of Behavioral Model:

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

entity TBMUX8 is
end TBMUX8;

architecture Testbench of TBMUX8 is

--8:1 Multiplexer component
component BMUX8 --For Behavioral Model
    Port ( X: in STD_LOGIC_VECTOR (7 downto 0); --Input terminals
               S: in STD_LOGIC_VECTOR (2 downto 0); --
Select (2^3 = 8 for 8 inputs )
               Z : out STD_LOGIC); --output terminal
end component;

--Signals
signal X: STD_LOGIC_VECTOR (7 downto 0):= (others=> '0');
signal S: STD_LOGIC_VECTOR (2 downto 0):= (others=> '0');
signal Z: STD_LOGIC;
```

```
begin

uut: BMUX8 port map ( --Change to SMUX8 for Structural Model
    X=> X,
    S=> S,
    Z=> Z
);

process_S: process
    begin
        X <= "10101010";
        wait for 1000 ns;
    end process;

process_S: process
    begin
        S <= "000";
        wait for 50 ns;
        S <= "001";
        wait for 50 ns;
        S <= "010";
        wait for 50 ns;
        S <= "011";
        wait for 50 ns;
        S <= "100";
        wait for 50 ns;
        S <= "101";
        wait for 50 ns;
        S <= "110";
        wait for 50 ns;
        S <= "111";
        wait for 50 ns;
    end process;

end Testbench;
```

Testbench Code of Structural Model:

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

entity TBMUX8 is
end TBMUX8;

architecture Testbench of TBMUX8 is
```

```
--8:1 Multiplexer component
component SMUX8 --For Structural Model
    Port ( X: in STD_LOGIC_VECTOR (7 downto 0); --Input terminals
                S: in STD_LOGIC_VECTOR (2 downto 0); --
Select (2^3 = 8 for 8 inputs )
                Z : out STD_LOGIC); --output terminal
end component;

--Signals
signal X: STD_LOGIC_VECTOR (7 downto 0):= (others=> '0');
signal S: STD_LOGIC_VECTOR (2 downto 0):= (others=> '0');
signal Z: STD_LOGIC;

begin

uut: SMUX8 port map ( --Change to SMUX8 for Structural Model
    X=> X,
    S=> S,
    Z=> Z
);

process_X: process
    begin
        X <= "10101010";
        wait for 1000 ns;
    end process;

process_S: process
    begin
        S <= "000";
        wait for 50 ns;
        S <= "001";
        wait for 50 ns;
        S <= "010";
        wait for 50 ns;
        S <= "011";
        wait for 50 ns;
        S <= "100";
        wait for 50 ns;
        S <= "101";
        wait for 50 ns;
        S <= "110";
        wait for 50 ns;
        S <= "111";
```

```
        wait for 50 ns;
    end process;

end Testbench;
```