Lab #1

ECE 4304 Spring 2021

Professor Aly

California State Polytechnic University, Pomona

Ahiezer Lopez	Joe Si	Sander Zuckerman
ahiezerlopez@cpp.edu	joesi@cpp.edu	sazuckerman@cpp.edu
Bronco ID:012715521	Bronco ID: 012636091	Bronco ID: 012644671

2/10/2021

Objective:
Students will create a 64x1 Multiplexer through VHDL utilizing certain coding formats such as “generic”, Mux2x1 network, and for loops. Demonstration requires students to utilize the textio library and create a simulation. Students will also implement a 16x1 multiplexer onto the FPGA boards. The 16bit input will be each switch on the board and the 6 bit select will be the buttons on the FPGA. Our 1 bit output will be represented through the LEDs.

Materials:
- FPGA (Nexys A7-100T)
- Vivado Software
- Computer

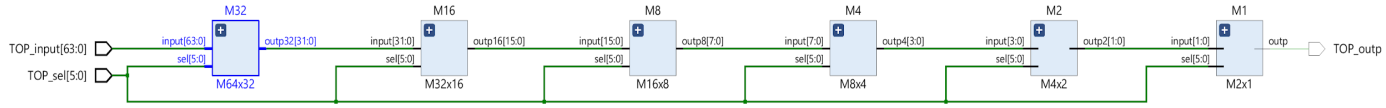
Contributions:
This lab was done with the work effort of every person in the group. For this lab, Joe was responsible for the design of the 64x1 mux, while Sander was in charge of the textio and test bench. Ahiezer was responsible for implementing the 16x1mux onto the FPGA. After having done the lab, this report was worked on together so that we could give insight to whoever is reading this about what we did in this lab.

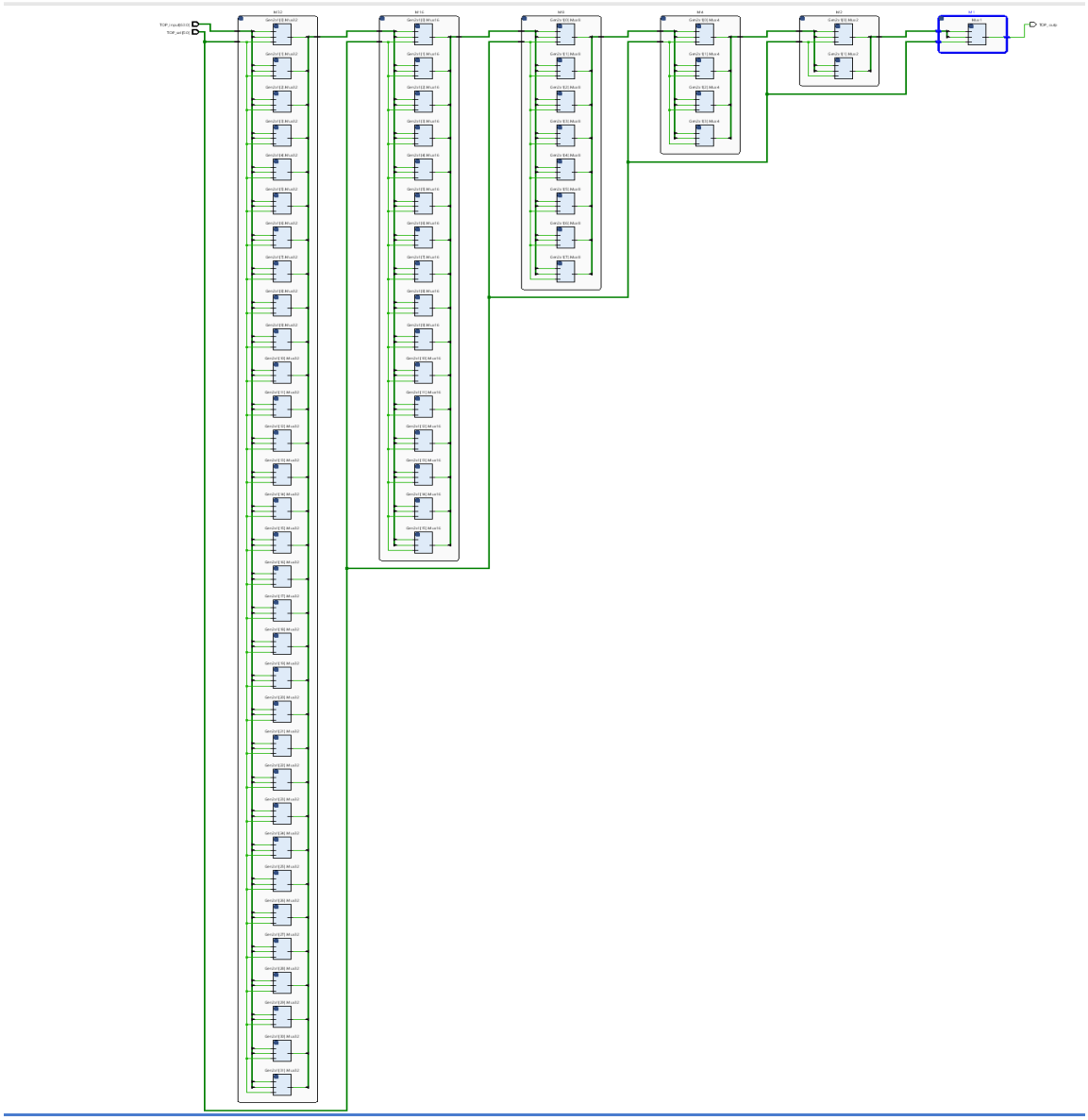
Design Process:
Using the 2x1 given to us during class, we used this as our main component between our modules. We first attempted to create one module with separate generate statements, however we could not find a way to do it. We ended up creating multiple modules each with the 2x1 multiplexer component and each having a generate statement with a generic parameter that decides the amount of bits and the amount of 2x1 multiplexers within that layer/module. With each module created we were able to instantiate each module and connect them with signals each with a determined amount of bits assisted by using the generic parameters. In the end we were able to create a 64x1 multiplexer with 17 LUT and a power consumption of 1.405W.

Design:
We made use of the 2x1 multiplexer component and created separate layers of a determined amount of Mux’s totaling with 6 layers and 127 2x1 Multiplexers. We have an input of 64 bits into the 64x1 multiplexer and we essentially half the bits to separate input 1 and input 0 of each 2x1 multiplexer. Each layer outputs half the amount of input bits. Ex. The first layer has a 64 bit input and outputs 32 bits, then the next layer receives a 32 bit input and outputs 16 bits and so

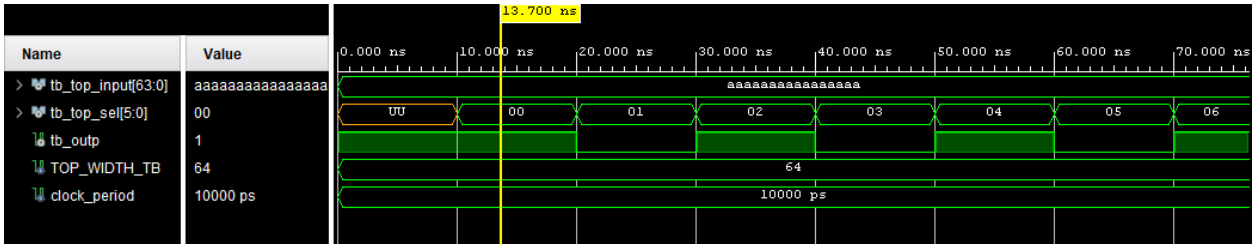
on. We have the 6 bit select input to select which bit out of the 64 bits to select. Each layer only takes one bit out of the layer. Ex. The first layer selects the least significant bit of the select, second select bit goes to the second layer and so on. We were able to create a functional 64x1 mux and output the correct bits.

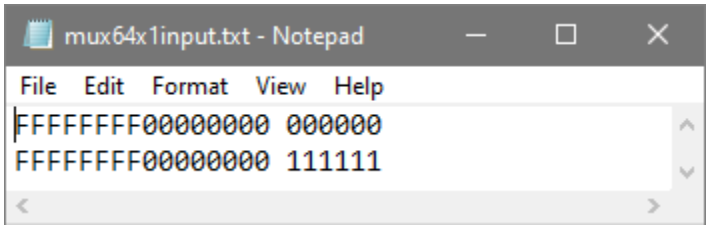
Circuit Diagram:

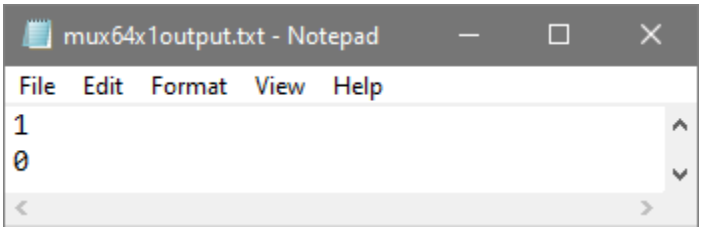




Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF
✓ synth_1	constrs_1	synth_design Complete!								17	0
✓ impl_1	constrs_1	route_design Complete!	NA	NA	NA	NA	NA	1.405	0	17	0

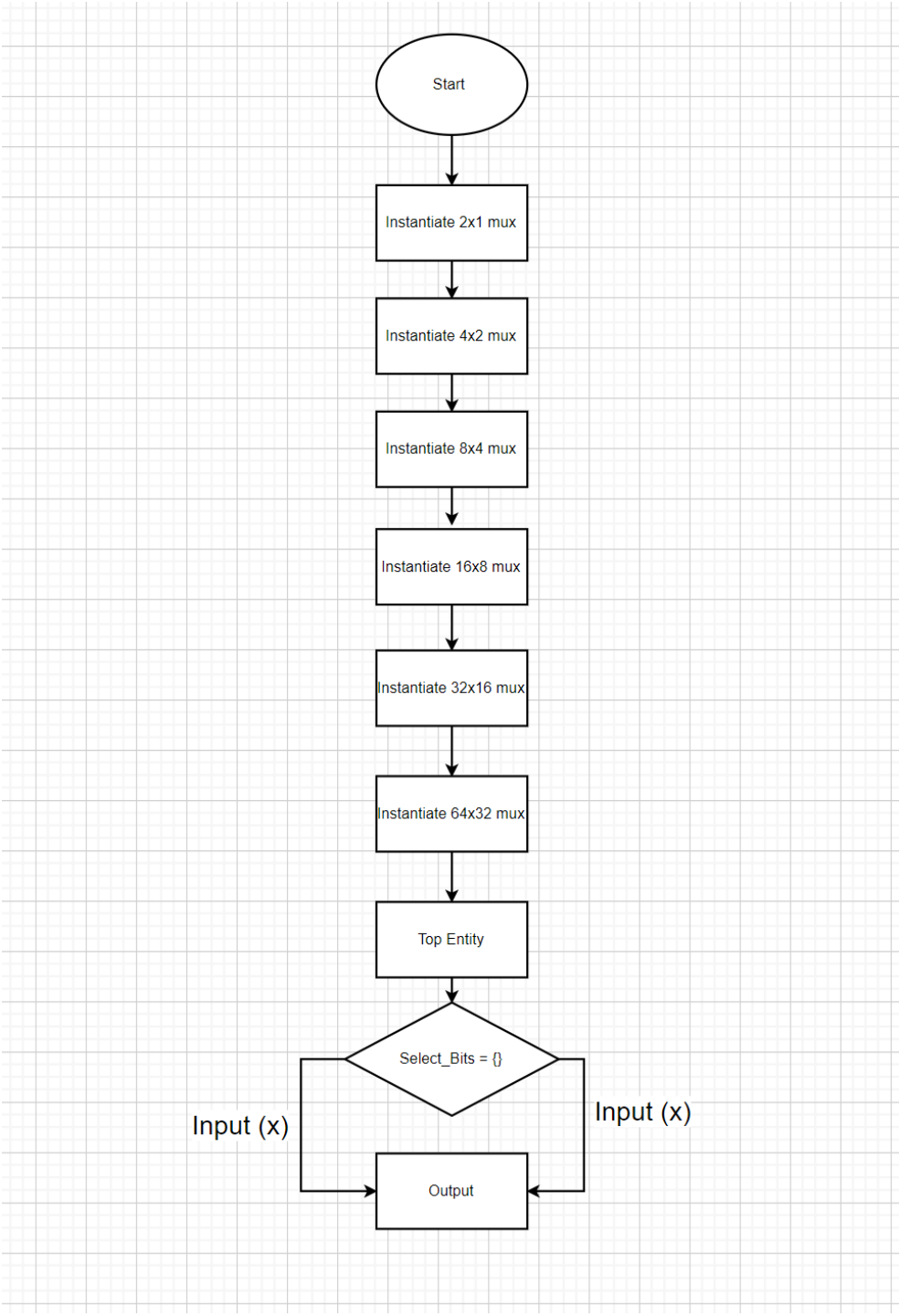

13.700 ns
Name
Value
tb_top_input[63:0]
aaaaaaaaaaaaaaaa
tb_top_sel[5:0]
00
tb_outp
1
TOP_WIDTH_TB
64
clock_period
10000 ps
0.000 ns
10.000 ns
20.000 ns
30.000 ns
40.000 ns
50.000 ns
60.000 ns
70.000 ns
aaaaaaaaaaaaaaaa
UU
00
01
02
03
04
05
06
64
10000 ps


mux64x1input.txt - Notepad
File Edit Format View Help
FFFFFFFF00000000 000000
FFFFFFFF00000000 111111


mux64x1output.txt - Notepad
File Edit Format View Help
1
0

Implementation:

Logic Flow Diagram



Design:

Top Module

```
34  entity Mux64x1 is
35          generic  (
36                      TOP_WIDTH:integer:=64
37                  );
38        Port (
39              TOP_input: in std_logic_vector(TOP_WIDTH-1 downto 0);
40              TOP_sel    : in std_logic_vector(5 downto 0);
41              TOP_outp   : out std_logic
42             );
43  end Mux64x1;
44
45  architecture Behavioral of Mux64x1 is
46
47  component M64x32
48          generic (
49                      WIDTH1:integer:=64
50                  );
51        Port (
52              input    : in std_logic_vector(WIDTH1-1 downto 0);
53              sel      : in std_logic_vector(5 downto 0);
54              outp32   : out std_logic_vector(WIDTH1/2-1 downto 0)
55             );
56  end component;
57
58  component M32x16
59          generic (
60                      WIDTH2:integer:=32
61                  );
62        Port (
63              input    : in std_logic_vector(WIDTH2-1 downto 0);
64              sel      : in std_logic_vector(5 downto 0);
65              outp16   : out std_logic_vector(WIDTH2/2-1 downto 0)
66             );
67  end component;
68
69  component M16x8
70          generic (
71                      WIDTH3:integer:=16
72                  );
73        Port (
74              input    : in std_logic_vector(WIDTH3-1 downto 0);
75              sel      : in std_logic_vector(5 downto 0);
76              outp8   : out std_logic_vector(WIDTH3/2-1 downto 0)
77             );
```

```
78  end component;
79
80  component M8x4
81          generic (
82                      WIDTH4:integer:=8
83                  );
84        Port (
85               input    : in std_logic_vector(WIDTH4-1 downto 0);
86               sel      : in std_logic_vector(5 downto 0);
87               outp4    : out std_logic_vector(WIDTH4/2-1 downto 0)
88              );
89  end component;
90
91  component M4x2
92          generic (
93                      WIDTH5:integer:=4
94                  );
95        Port (
96               input    : in std_logic_vector(WIDTH5-1 downto 0);
97               sel      : in std_logic_vector(5 downto 0);
98               outp2    : out std_logic_vector(WIDTH5/2-1 downto 0)
99              );
100 end component;
101
102 component M2x1
103         generic (
104                     WIDTH6:integer:=2
105                 );
106       Port (
107              input    : in std_logic_vector(WIDTH6-1 downto 0);
108              sel      : in std_logic_vector(5 downto 0);
109              outp     : out std_logic
110             );
111 end component;
112
113 signal temp_outp1: std_logic_vector(TOP_WIDTH/2-1 downto 0);
114 signal temp_outp2: std_logic_vector(TOP_WIDTH/4-1 downto 0);
115 signal temp_outp3: std_logic_vector(TOP_WIDTH/8-1 downto 0);
116 signal temp_outp4: std_logic_vector(TOP_WIDTH/16-1 downto 0);
117 signal temp_outp5: std_logic_vector(TOP_WIDTH/32-1 downto 0);
118
119
120 begin
121
```

```
M32: M64x32     generic map (WIDTH1 => TOP_WIDTH)
             port map(
                 input   => TOP_input,
                 sel     => TOP_sel,
                 outp32  => temp_outp1
                     );

M16: M32x16 generic map (WIDTH2 => TOP_WIDTH/2)
         port map(
                 input   => temp_outp1,
                 sel     => TOP_sel,
                 outp16  => temp_outp2
                     );

M8: M16x8 generic map (WIDTH3 => TOP_WIDTH/4)
         port map(
                 input   => temp_outp2,
                 sel     => TOP_sel,
                 outp8  => temp_outp3
                     );

M4: M8x4 generic map (WIDTH4 => TOP_WIDTH/8)
         port map(
                 input   => temp_outp3,
                 sel     => TOP_sel,
                 outp4  => temp_outp4
                     );

M2: M4x2 generic map (WIDTH5 => TOP_WIDTH/16)
         port map(
                 input   => temp_outp4,
                 sel     => TOP_sel,
                 outp2   => temp_outp5
                     );

M1: M2x1 generic map (WIDTH6 => TOP_WIDTH/32)
          port map(
                 input   => temp_outp5,
                 sel     => TOP_sel,
                 outp  => TOP_outp
                     );
end Behavioral;
```

Layer Modules

```
34  entity M64x32 is
35          generic (
36                      WIDTH1:integer:=64
37                  );
38        Port (
39               input    : in std_logic_vector(WIDTH1-1 downto 0);
40               sel      : in std_logic_vector(5 downto 0);
41               outp32   : out std_logic_vector(WIDTH1/2-1 downto 0)
42              );
43  end M64x32;
44
45  architecture Behavioral of M64x32 is
46
47  component Mux2x1
48      Port (
49             inp_0  : in std_logic;
50             inp_1  : in std_logic;
51             sel    : in std_logic;
52             outp   : out std_logic
53            );
54  end component;
55
56  signal temp_out: std_logic_vector(WIDTH1/2-1 downto 0);
57  begin
58
59
60  Gen2x1: for i in 0 to WIDTH1/2-1 generate
61
62              Mux32: Mux2x1 port map (
63                                      inp_0 => input(2*i),
64                                      inp_1 => input(2*i+1),
65                                      sel   => sel(0),
66                                      outp  => temp_out(i)
67                                       );
68         end generate;
69
70  outp32 <= temp_out;
71
72
73  end Behavioral;
```

```
34  entity M32x16 is
35          generic (
36                      WIDTH2:integer:=32
37                  );
38        Port (
39               input    : in std_logic_vector(WIDTH2-1 downto 0);
40               sel      : in std_logic_vector(5 downto 0);
41               outp16   : out std_logic_vector(WIDTH2/2-1 downto 0)
42              );
43  end M32x16;
44
45  architecture Behavioral of M32x16 is
46
47  component Mux2x1
48      Port (
49             inp_0  : in std_logic;
50             inp_1  : in std_logic;
51             sel    : in std_logic;
52             outp   : out std_logic
53            );
54  end component;
55
56  signal temp_out: std_logic_vector(WIDTH2/2-1 downto 0);
57  begin
58
59
60  Gen2x1: for i in 0 to WIDTH2/2-1 generate
61
62              Mux16: Mux2x1 port map (
63                                      inp_0 => input(2*i),
64                                      inp_1 => input(2*i+1),
65                                      sel   => sel(1),
66                                      outp  => outp16(i)
67                                       );
68          end generate;
69
70
71
72  end Behavioral;
```

```
34  entity M16x8 is
35          generic (
36                      WIDTH3:integer:=16
37                   );
38        Port (
39               input    : in std_logic_vector(WIDTH3-1 downto 0);
40               sel      : in std_logic_vector(5 downto 0);
41               outp8    : out std_logic_vector(WIDTH3/2-1 downto 0)
42              );
43  end M16x8;
44
45  architecture Behavioral of M16x8 is
46
47  component Mux2x1
48      Port (
49             inp_0  : in std_logic;
50             inp_1  : in std_logic;
51             sel    : in std_logic;
52             outp   : out std_logic
53            );
54  end component;
55
56  signal temp_out: std_logic_vector(WIDTH3/2-1 downto 0);
57  begin
58
59
60  Gen2x1: for i in 0 to WIDTH3/2-1 generate
61
62              Mux8: Mux2x1 port map (
63                                      inp_0 => input(2*i),
64                                      inp_1 => input(2*i+1),
65                                      sel   => sel(2),
66                                      outp  => temp_out(i)
67                                       );
68          end generate;
69
70  outp8 <= temp_out;
71
72
73  end Behavioral;
```

```
34  entity M8x4 is
35          generic (
36                      WIDTH4:integer:=8
37                   );
38        Port (
39               input    : in std_logic_vector(WIDTH4-1 downto 0);
40               sel      : in std_logic_vector(5 downto 0);
41               outp4    : out std_logic_vector(WIDTH4/2-1 downto 0)
42              );
43  end M8x4;
44
45  architecture Behavioral of M8x4 is
46
47  component Mux2x1
48      Port (
49             inp_0  : in std_logic;
50             inp_1  : in std_logic;
51             sel    : in std_logic;
52             outp   : out std_logic
53            );
54  end component;
55
56  signal temp_out: std_logic_vector(WIDTH4/2-1 downto 0);
57  begin
58
59
60  Gen2x1: for i in 0 to WIDTH4/2-1 generate
61
62              Mux4: Mux2x1 port map (
63                                      inp_0 => input(2*i),
64                                      inp_1 => input(2*i+1),
65                                      sel   => sel(3),
66                                      outp  => temp_out(i)
67                                       );
68         end generate;
69
70  outp4 <= temp_out;
71
72
73  end Behavioral;
```

```
34  entity M4x2 is
35          generic (
36                      WIDTH5:integer:=4
37                    );
38        Port (
39               input    : in std_logic_vector(WIDTH5-1 downto 0);
40               sel      : in std_logic_vector(5 downto 0);
41               outp2    : out std_logic_vector(WIDTH5/2-1 downto 0)
42              );
43  end M4x2;
44
45  architecture Behavioral of M4x2 is
46
47  component Mux2x1
48      Port (
49             inp_0  : in std_logic;
50             inp_1  : in std_logic;
51             sel    : in std_logic;
52             outp   : out std_logic
53            );
54  end component;
55
56  signal temp_out: std_logic_vector(WIDTH5/2-1 downto 0);
57  begin
58
59
60  Gen2x1: for i in 0 to WIDTH5/2-1 generate
61
62              Mux2: Mux2x1 port map (
63                                      inp_0 => input(2*i),
64                                      inp_1 => input(2*i+1),
65                                      sel   => sel(4),
66                                      outp  => temp_out(i)
67                                       );
68         end generate;
69
70  outp2 <= temp_out;
71
72
73  end Behavioral;
```

```
34  entity M2x1 is
35          generic (
36                      WIDTH6:integer:=2
37                  );
38        Port (
39               input    : in std_logic_vector(WIDTH6-1 downto 0);
40               sel      : in std_logic_vector(5 downto 0);
41               outp    : out std_logic
42              );
43  end M2x1;
44
45  architecture Behavioral of M2x1 is
46
47  component Mux2x1
48      Port (
49             inp_0  : in std_logic;
50             inp_1  : in std_logic;
51             sel    : in std_logic;
52             outp   : out std_logic
53            );
54  end component;
55
56  signal temp_out: std_logic;
57  begin
58
59              Mux1: Mux2x1 port map (
60                                      inp_0 => input(0),
61                                      inp_1 => input(1),
62                                      sel   => sel(5),
63                                      outp  => temp_out
64                                      );
65
66  outp <= temp_out;
67
68
69  end Behavioral;
```

Test Bench:

```
34  entity Mux64x1_tb is
35          generic(TOP_WIDTH_TB:integer:=64);
36  --  Port ( );
37  end Mux64x1_tb;
38
39  architecture Behavioral of Mux64x1_tb is
40
41  component Mux64x1
42          generic (
43                      TOP_WIDTH:integer:=64
44                  );
45        Port (
46               TOP_input: in std_logic_vector(TOP_WIDTH-1 downto 0);
47               TOP_sel    : in std_logic_vector(5 downto 0);
48               TOP_outp   : out std_logic
49              );
50  end component;
51
52  signal tb_top_input: std_logic_vector(TOP_WIDTH_TB-1 downto 0);
53  signal tb_top_sel: std_logic_vector(5 downto 0);
54  signal tb_outp: std_logic;
55
56  constant clock_period:time:=10 ns;
57
58  begin
59
60  MUX_1: Mux64x1
61                  generic map (TOP_WIDTH => TOP_WIDTH_TB)
62                  port map (
63                          TOP_input => tb_top_input,
64                          TOP_sel   => tb_top_sel,
65                          TOP_outp  => tb_outp
66                          );
67
68
69  GEN_TB_SIGNALS: process
70                  begin
71                      tb_top_input <= x"AAAAAAAAAAAAAAAA";
72                      wait for clock_period;
73                      tb_top_sel   <=  (others =>'0');
74
75                      wait for clock_period;
76                      tb_top_sel   <= b"000001";
77
78                      wait for clock_period;
```

TextIO test bench:

```
75  TEXTIO_GEN: process
76              variable v_ILINE: line;
77              variable v_OLINE: line;
78              variable v_INPUT: std_logic_vector(TOP_WIDTH_TB-1 downto 0);
79              variable v_SEL: std_logic_vector(5 downto 0);
80              variable v_space: character;
81
82              begin
83              file_open(file_vectors, "D:\Users\Sander\Documents\Vivado\mux64x1input.txt" , read_mode);
84              file_open(file_output, "D:\Users\Sander\Documents\Vivado\mux64x1output.txt" , write_mode);
85
86              while not endfile(file_vectors) loop
87                  readline(file_vectors, v_ILINE);
88                  hread(v_ILINE, V_INPUT);
89                  read(v_ILINE, v_space);
90                  read(v_ILINE, V_SEL);
91
92                  tb_top_input <= V_INPUT;
93                  tb_top_sel <= V_SEL;
94
95                  wait for 2*clock_period;
96
97                  write (v_OLINE, tb_outp);
98                  writeline(file_output, v_OLINE);
99
100             end loop;
101             file_close(file_vectors);
102             file_close(file_output);
103
104             wait;
105             end process;
```

Analysis:

In this lab, we were able to create a combinational circuit utilizing multiple layers of smaller 2x1 multiplexers to create one bigger multiplexer, in this case a 64x1. By connecting the outputs of the smaller multiplexers to the inputs of the next multiplexers, and by connecting the select bits to all the multiplexers in the right fashion, this circuit is able to select between 64 inputs and provide 1 output.

As the waveform in the circuit diagram portion of this report shows, the circuit is successfully able to select through a 64-bit input. The provided test bench ticks up every clock cycle from 0 upward, and the output as shown alternates between high and low. Since the input itself alternates between high and low (where AAAAAAAAAAAAAAAA in hex is 10101010...), the output starts from the most significant bit and works its way to the right toward the least significant bit as the select bit grows higher. This is also demonstrated on the textio testbench,

where two cases were considered: With an input of FFFFFFFF00000000, and the select bit set to 0 and 63, the output is 1 and 0 respectively.

In the context of this lab, we were not able to identify any corner cases to consider.

Lastly, after having implemented the 16x1 mux onto the FPGA board, we were able to see how our code works. After selecting a certain input through the select bits, we witnessed the LED turn on if the input were high or not.

Conclusion:

In this lab, we successfully created a 64x1 multiplexer by creating a “waterfall” of smaller, serially connected 2x1 multiplexers. This was a great start into VHDL, as the design is very straightforward, and the implementation is simple. This was a good hands-on experience to learn how to create basic systems using VHDL; much like every programmer’s “hello world” program to start learning a new language, this was an effective way to start learning VHDL. We were able to create and test our 64x1 multiplexer using a testbench, a textio, and an FPGA to verify our design, and we are able to conclude that our code works entirely as intended.