Hassan Allabad
Dejan Rasuo
Gabriel Van Dreel
November 27, 2019
ECGR 2181-002

Project 2 – Fall 2019

Part 1:

For the first part of the project, a four-digit seven segment display was modeled and implemented using switches as binary inputs on a Digilent Basys 3 Artix-7 Field-Programmable Gate Array (FPGA) Board. This was done by corresponding four groups of switches as binary inputs to each of the encoder input groups needed to define the output of the four seven segment displays on the Basys 3 board. In developing the VHDL model for this part of the project, it was assumed that structural VHDL would be used with the predefined behavioral script, ssd_muxer.vhd, for the seven segment display multiplexer of the Basys 3 board and the encoder behavioral script, encoder.vhd, shown in Figure 2 for a seven segment display from project 1. These files were then specified to be imported into a Vivado project titled, “computer_assignment_3.”

In developing structural VHDL for this part of the project, a new file was created entitled “top.vhd” that is shown in Figure 1 corresponding specifically to the computer_assignment_3 Vivado project. The “top” entity was defined and the encoder and ssd_muxer components were declared. The architecture of the implementation for the four-digit seven segment display was defined using signals that mimicked wires and were accordingly named as a “w” followed by a positive integer starting from “0.” These wires mapped the connections of the seven segment display multiplexer, called mux1, to each of the four seven segment displays. To verify the functionality of the device, simple entity declarations and signal assignments were used at first. Because it was found in Dr. Jayram Moorkanikara Nageswaran's “Generate Statement” article that there existed more compact VHDL that served the same purpose, however, a for generate statement was used. Upon testing both a simple instantiation of the seven segment display components as well as an array of them with a for generate statement, it was found that both led to the same functionality of the final circuit. Therefore, the simplified but longer instantiation of the seven segment display components was commented and the for generate statement was used.

No errors were found with the final circuit implementation using a for generate statement and, during testing, each switch lit a corresponding Light-Emitting Diode (LED) as was specified in the deliverables for the first part of the project. While the compiled bitstream in Vivado threw six synthesis warnings likely due to a lack of an additional constraints file to exactly match every input and output to those listed in the constraints file for the Basys 3 board, the final implemented design worked correctly upon testing of several switch combinations. A testbench was not developed for this portion of the project because it appeared unnecessary given the simple operation of the Basys 3 board for the binary inputs of the seven segment displays.

```
----------------------------------------------------------------------------------
-- Company: UNC Charlotte
-- Engineer: Dejan Rasuo, Gabriel Van Dreel, & Hassan Allabad
--
-- Create Date: 11/16/2019 04:52:58 PM
-- Design Name: Seven Segment Display Encoder
-- Module Name: top - Behavioral
-- Project Name: Project 2
-- Target Devices:
-- Tool Versions:
-- Description: This task is an implementation of four encoders to operate a 4-digit seven
    segment display given switches as inputs to the encoders
--
-- 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 leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity top is
    port(   sw: in std_logic_vector(15 downto 0);
            clk: in std_logic;
            seg: out std_logic_vector(6 downto 0);
            dp: out std_logic;
            an: out std_logic_vector(3 downto 0);
            led: out std_logic_vector(15 downto 0));
```

```
end top;

architecture Behavioral of top is

--Imports encoder
component encoder is
    port(   hex_in: in std_logic_vector(3 downto 0);
            a: out std_logic;
            b: out std_logic;
            c: out std_logic;
            d: out std_logic;
            e: out std_logic;
            f: out std_logic;
            g: out std_logic);
end component;

--Imports ssd_muxer
component ssd_muxer is
    port(   a_in: in std_logic_vector(3 downto 0);
            b_in: in std_logic_vector(3 downto 0);
            c_in: in std_logic_vector(3 downto 0);
            d_in: in std_logic_vector(3 downto 0);
            e_in: in std_logic_vector(3 downto 0);
            f_in: in std_logic_vector(3 downto 0);
            g_in: in std_logic_vector(3 downto 0);
            decp0_in: in std_logic; --Could be simplified t0 decp_in in std_logic_vector(3 downto 0)
            decp1_in: in std_logic;
            decp2_in: in std_logic;
            decp3_in: in std_logic;
            seg_out: out std_logic_vector(6 downto 0);
            dp_out: out std_logic;
            an_out: out std_logic_vector(3 downto 0);
            clk: in std_logic);
end component;

--Declares signals for components
signal w0: std_logic_vector(3 downto 0); --w0 or wire 0 is analogous to a(3), a(2), a(1), and a(0)
      augmented as shown in Figure 1, respectively
signal w1: std_logic_vector(3 downto 0);
signal w2: std_logic_vector(3 downto 0);
signal w3: std_logic_vector(3 downto 0);
signal w4: std_logic_vector(3 downto 0);
signal w5: std_logic_vector(3 downto 0);
signal w6: std_logic_vector(3 downto 0);

begin

--Connects the switches to the LEDs
led <= sw;

--Implements the seven segment display multiplexer
mux1: ssd_muxer port map(
    a_in => w0,
    b_in => w1,
    c_in => w2,
    d_in => w3,
    e_in => w4,
    f_in => w5,
    g_in => w6,
    decp0_in => '0',
    decp1_in => '0',
    decp2_in => '0',
    decp3_in => '0',
    clk => clk,
    seg_out => seg,
    dp_out => dp,
    an_out => an);

--Implements four encoders
gen_enc: for i in 0 to 3 generate
    encx: encoder port map(hex_in => sw((4 * i + 3) downto (4 * i)), a => w0(i), b => w1(i), c =>
      w2(i), d => w3(i), e => w4(i), f => w5(i), g => w6(i));
end generate gen_enc;

--enc1 is analogous to gen_enc[3]
--enc1: encoder port map( --Potentially use for generator statement
--    hex_in => sw(15 downto 12),
--    a => w0(3),
--    b => w1(3),
--    c => w2(3),
--    d => w3(3),
--    e => w4(3),
--    f => w5(3),
--    g => w6(3));

--enc2: encoder port map(
--    hex_in => sw(11 downto 8),
--    a => w0(2),
--    b => w1(2),
--    c => w2(2),
--    d => w3(2),
--    e => w4(2),
--    f => w5(2),
--    g => w6(2));

--enc3: encoder port map(
--    hex_in => sw(7 downto 4),
--    a => w0(1),
--    b => w1(1),
--    c => w2(1),
--    d => w3(1),
--    e => w4(1),
--    f => w5(1),
--    g => w6(1));

--enc4: encoder port map(
--    hex_in => sw(3 downto 0),
--    a => w0(0),
--    b => w1(0),
--    c => w2(0),
--    d => w3(0),
--    e => w4(0),
--    f => w5(0),
--    g => w6(0));

end Behavioral;
```

Figure 1: Four-Digit Seven Segment Display Encoder (top.vhd) Code

```
----------------------------------------------------------------------------------
-- Company: UNC Charlotte
-- Engineer: Dejan Rasuo, Gabriel Van Dreel, & Hassan Allabad
--
-- Create Date: 08/25/2019 04:12:54 PM
-- Design Name: Encoder
-- Module Name: encoder - Behavioral
-- Project Name: Project 2
-- Target Devices:
-- Tool Versions:
-- Description: This task acts as an encoder for a seven segment display that takes a four bit
    unsigned binary value
--
-- 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 leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity encoder is
   port(   hex_in: in std_logic_vector(3 downto 0);
          a: out std_logic;
          b: out std_logic;
          c: out std_logic;
          d: out std_logic;
          e: out std_logic;
```

```
            f: out std_logic;
            g: out std_logic);
end encoder;

architecture Behavioral of encoder is

signal output: std_logic_vector(6 downto 0);

begin

process(hex_in)
begin
    case hex_in is
        when "0000" => output <= "1111110";
        when "0001" => output <= "0110000";
        when "0010" => output <= "1101101";
        when "0011" => output <= "1111001";
        when "0100" => output <= "0110011";
        when "0101" => output <= "1011011";
        when "0110" => output <= "1011111";
        when "0111" => output <= "1110000";
        when "1000" => output <= "1111111";
        when "1001" => output <= "1111011";
        when "1010" => output <= "1110111";
        when "1011" => output <= "0011111";
        when "1100" => output <= "1001110";
        when "1101" => output <= "0111101";
        when "1110" => output <= "1001111";
        when "1111" => output <= "1000111";
        when others => output <= "0000000";
    end case;
end process;

a <= output(6);
b <= output(5);
c <= output(4);
d <= output(3);
e <= output(2);
f <= output(1);
g <= output(0);
```

```
end Behavioral;
```

Figure 2: Seven Segment Display Encoder (encoder.vhd) Code

Part 2:

The second part of the project built on the first part in that they both used much of the same structural VHDL needed to operate the seven segment display, However, it required that eight bits inputted from eight switches needed to be used for inputting a two's complement binary value that would be converted to a signed decimal value displayed on the four available seven segment displays. In addition to the ssd_muxer.vhd and encoder.vhd files used in the first part of the project, it was assumed that the prewritten script, debounce.vhd, would also be used for sampling binary values from two buttons on the Basys 3 board that were intended to either clear (clr) the current binary value being converted on the board or enter (en) the new one represented by the states of the switches on the board. This was important because it would allow the user to set his or her two's complement binary input before having the board perform a conversion of the given binary value and avoid providing distracting or unwanted decimal values on the four seven segment displays. It was also assumed that this portion of the project would be completed in a Vivado project titled, "computer_assignment_4," to prevent filename ambiguity among modified VHDL files.

Because this portion of the project required the use of two new seven segment display states to show the sign of the converted decimal, the encoder.vhd file was modified as shown in Figure 4. One output was redefined where no segments were enabled to show a positive value or unused decimal place and another output was used to show a negative sign such that only segment G, the middle-most segment, would light. The binary encodings "1110" and "1111" were selected to correspond to the listed encoder outputs, respectively, since they were convenient to type with a symmetry of leading ones and only differed by one bit with the least-significant bit as a form of gray code to indicate the sign of the decimal. In the development of the VHDL for this part of the project, the "top" entity whose script is shown in Figure 3 was defined and the encoder, ssd_muxer, and additional debounce components were also declared. The debounce components were instantiated as d1 and d2 where d1 preserved binary signals from the right-hand-side button, btnR, for the clr signal and d2 preserved binary signals from the left-hand-side-button, btnL, for the en signal. When performing the binary conversion, the IEEE.NUMERIC_STD library was used to simplify the conversion process and perform decimal arithmetic to determine the binary signals that would need to be sent to the seven segment display multiplexer. A multi-way if statement was therefore used to map the appropriate binary outputs to every seven segment display using wire signals that followed the same naming convention as that discussed in part one of project two.

When implementing each of the seven segment displays, it was observed that a for generate statement was not practical due to the consolidation of the hexadecimal encoder inputs that were used for computation as the signals h0, h1, h2, and h3. The bitstream was successfully generated with minimal constraint errors and the implementation on the Basys 3 board performed correctly. No testbench files were developed because the operation of the Basys 3 board was available for testing and debugging the VHDL implementation for this part of the project.

```
----------------------------------------------------------------------------------
-- Company: UNC Charlotte
-- Engineer: Dejan Rasuo, Gabriel Van Dreel, & Hassan Allabad
--
-- Create Date: 11/16/2019 06:05:22 PM
-- Design Name: Two's Complement Binary Converter
-- Module Name: top - Behavioral
-- Project Name: Project 2
-- Target Devices:
-- Tool Versions:
-- Description: This is an implementation of four encoders that display the signed decimal
    equivalent of a two's complement binary value inputted using eight switches for input and
    two buttons that either enter or clear the binary input
--
-- 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 leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity top is
   port(   sw: in std_logic_vector(7 downto 0);
           btnR: in std_logic;
           btnL: in std_logic;
           clk: in std_logic;
           seg: out std_logic_vector(6 downto 0);
```

```
            dp: out std_logic;
            an: out std_logic_vector(3 downto 0);
            led: out std_logic_vector(7 downto 0));
end top;

architecture Behavioral of top is

--Imports encoder
component encoder is
    port(   hex_in: in std_logic_vector(3 downto 0);
            a: out std_logic; --Could simplify signal and I/O declarations
            b: out std_logic;
            c: out std_logic;
            d: out std_logic;
            e: out std_logic;
            f: out std_logic;
            g: out std_logic);
end component;

--Imports ssd_muxer
component ssd_muxer is
    port(   a_in: in std_logic_vector(3 downto 0);
            b_in: in std_logic_vector(3 downto 0);
            c_in: in std_logic_vector(3 downto 0);
            d_in: in std_logic_vector(3 downto 0);
            e_in: in std_logic_vector(3 downto 0);
            f_in: in std_logic_vector(3 downto 0);
            g_in: in std_logic_vector(3 downto 0);
            decp0_in: in std_logic; --Could be simplified t0 decp_in in std_logic_vector(3 downto 0)
            decp1_in: in std_logic;
            decp2_in: in std_logic;
            decp3_in: in std_logic;
            seg_out: out std_logic_vector(6 downto 0);
            dp_out: out std_logic;
            an_out: out std_logic_vector(3 downto 0);
            clk: in std_logic);
end component;

--Imports debounce
component debounce is
    port(   clk_100m: in std_logic;
            sw: in std_logic;
            sglPulse: out std_logic;
            sig: out std_logic);
end component;

--Declares signals for entering and clearing input
signal clr: std_logic; --Could simplify signal and I/O declarations
signal en: std_logic;

--Declares signals for computation
signal accumulator: std_logic_vector(7 downto 0);
signal h0: std_logic_vector(3 downto 0);
signal h1: std_logic_vector(3 downto 0);
signal h2: std_logic_vector(3 downto 0);
signal h3: std_logic_vector(3 downto 0);

--Declares signals for components
signal w0: std_logic_vector(3 downto 0);
signal w1: std_logic_vector(3 downto 0);
signal w2: std_logic_vector(3 downto 0);
signal w3: std_logic_vector(3 downto 0);
signal w4: std_logic_vector(3 downto 0);
signal w5: std_logic_vector(3 downto 0);
signal w6: std_logic_vector(3 downto 0);

begin

--Implements two debounces
d1: debounce port map(
    clk_100m => clk, --Can be condensed
    sw => btnR,
    sglPulse => clr,
    sig => open);

d2: debounce port map(
    clk_100m => clk,
    sw => btnL,
    sglPulse => en,
    sig => open);

--Connects the switches to the LEDs
```

```
led <= sw;

--Resets input or transfers input
process(clr, en)
begin
    if(clr = '1') then
        accumulator <= "00000000";
    elsif(en = '1') then
        accumulator <= sw;
    end if;
end process;

--2s complement conversion
process(accumulator)
variable tempNum, a, b, c: integer := 0;
begin
    tempNum := abs(to_integer(signed(accumulator)));

    a := tempNum / 100; --Seven segment hundreds place
    b := tempNum mod 100 / 10; --Seven segment tens place
    c := tempNum mod 100 mod 10 / 1; --Seven segment ones place

    if(accumulator(7) = '1') then --Negative number (-)
        if(a = 0 and b = 0) then
            h0 <= "1110";
            h1 <= "1110";
            h2 <= "1111";
        elsif(a = 0) then
            h0 <= "1110";
            h1 <= "1111";
            h2 <= std_logic_vector(to_unsigned(b, 4));
        else
            h0 <= "1111";
            h1 <= std_logic_vector(to_unsigned(a, 4));
            h2 <= std_logic_vector(to_unsigned(b, 4));
        end if;
    else --Positive number (+)
        h0 <= "1110";
        if(a = 0) then
            h1 <= "1110";
        else
```

```
            h1 <= std_logic_vector(to_unsigned(a, 4));
        end if;

        if(a = 0 and b = 0) then
            h2 <= "1110";
        else
            h2 <= std_logic_vector(to_unsigned(b, 4));
        end if;
    end if;

    h3 <= std_logic_vector(to_unsigned(c, 4));

end process;

--Implements the seven segment display multiplexer
mux1: ssd_muxer port map(
    a_in => w0,
    b_in => w1,
    c_in => w2,
    d_in => w3,
    e_in => w4,
    f_in => w5,
    g_in => w6,
    decp0_in => '0',
    decp1_in => '0',
    decp2_in => '0',
    decp3_in => '0',
    clk => clk,
    seg_out => seg,
    dp_out => dp,
    an_out => an);

--Implements four encoders
enc1: encoder port map( --Potentially use for generator statement
    hex_in => h0, --To make for generator statement h0, h1, h2, and h3 need to be augmented as
      one signal
    a => w0(3),
    b => w1(3),
    c => w2(3),
    d => w3(3),
    e => w4(3),
    f => w5(3),
    g => w6(3));

enc2: encoder port map(
    hex_in => h1,
    a => w0(2),
    b => w1(2),
    c => w2(2),
    d => w3(2),
    e => w4(2),
    f => w5(2),
    g => w6(2));

enc3: encoder port map(
    hex_in => h2,
    a => w0(1),
    b => w1(1),
    c => w2(1),
    d => w3(1),
    e => w4(1),
    f => w5(1),
    g => w6(1));

enc4: encoder port map(
    hex_in => h3,
    a => w0(0),
    b => w1(0),
    c => w2(0),
    d => w3(0),
    e => w4(0),
    f => w5(0),
    g => w6(0));

end Behavioral;
```

Figure 3: Two’s Complement Binary Converter (top.vhd) Code

```
----------------------------------------------------------------------------------
-- Company: UNC Charlotte
-- Engineer: Dejan Rasuo, Gabriel Van Dreel, & Hassan Allabad
--
-- Create Date: 08/25/2019 04:12:54 PM
-- Design Name: Modified Encoder
-- Module Name: encoder - Behavioral
-- Project Name: Project 2
-- Target Devices:
-- Tool Versions:
-- Description: This task acts as a modified encoder for a seven segment display that takes a four
    bit unsigned binary value
--
-- 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 leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity encoder is
    port(   hex_in: in std_logic_vector(3 downto 0);
            a: out std_logic;
            b: out std_logic;
            c: out std_logic;
            d: out std_logic;
            e: out std_logic;
            f: out std_logic;
            g: out std_logic);
end encoder;

architecture Behavioral of encoder is

signal output: std_logic_vector(6 downto 0);

begin

process(hex_in)
begin
    case hex_in is
        when "0000" => output <= "1111110";
        when "0001" => output <= "0110000";
        when "0010" => output <= "1101101";
        when "0011" => output <= "1111001";
        when "0100" => output <= "0110011";
        when "0101" => output <= "1011011";
        when "0110" => output <= "1011111";
        when "0111" => output <= "1110000";
        when "1000" => output <= "1111111";
        when "1001" => output <= "1111011";
        when "1010" => output <= "1110111";
        when "1011" => output <= "0011111";
        when "1100" => output <= "1001110";
        when "1101" => output <= "0111101";
        when "1110" => output <= "0000000"; --Modified to output no sign on the leading decimal
      place
        when "1111" => output <= "0000001"; --Modified to output a negative sign on the leading
      decimal place
        when others => output <= "0000000";
    end case;
end process;

a <= output(6);
b <= output(5);
c <= output(4);
d <= output(3);
e <= output(2);
f <= output(1);
```

```
g <= output(0);

end Behavioral;
```

Figure 4: Modified Seven Segment Display Encoder (encoder.vhd) Code

Part 3:

The third and final part of the second project focused on using VHDL to program the Basys 3 board to perform a binary guessing game where the user of the board could set an eight-bit binary value using the switches of the board. The game could then be played by pressing a button to set the given binary value as the value against which guesses may be checked and passing the board off to another user that can input guesses for the binary value using the switches of the board and check them by pressing another button on the board that will only light up the fifteenth LED in the event of a matching binary value. In implementing this game on the Basys 3 board, structural VHDL was used only with the ssd_muxer and debounce components since the seven segment displays were not needed. It was assumed that the structural script “top.vhd” shown in Figure 5 would be developed in a project titled, “computer_assignment_5.” Accordingly, the “top” entity was defined and the ssd_muxer and debounce components were declared.

In the development of this part of the project, it was decided that the default binary guessing value would be that of all low input values since it allowed for simple preliminary testing of the LED. Unlike the other parts of the project, a clock signal was explicitly used to sample inputs from the buttons when operating the guessing game on the Basys 3 board. While this was not necessary, it experimentally proved to work. To make the operation of the board as intuitive as possible, the right-hand-side button, btnR, was used to enter the sentinel binary value for the comparison while the left-hand-side button, btnL, was used to check the new input against the sentinel value since it was physically closer to the LED. During testing, however, a fatal error was encountered with the led output of the “top” entity that caused the bitstream generation to fail and likely occurred because it could not be assigned as a std_logic type to the available LED port on the board without an explicit constraint file mapping the led output to the board. Therefore, to assist with the automated assignment of inputs and outputs to the Basys 3 board using only the provided constraint file, the led was redefined as the std_logic_vector type with the bit range 15 downto 15. Once this modification was performed, the assignment of the available LED output to some value in the range of the possible LED ports on the board worked with a minimal number of errors.

The final implementation of the third part of the project was successful and had no operational issues. A testbench was not developed since the Basys 3 hardware was available for testing and debugging the VHDL developed in this part of the project.

```
----------------------------------------------------------------------------------
-- Company: UNC Charlotte
-- Engineer: Dejan Rasuo, Gabriel Van Dreel, & Hassan Allabad
--
-- Create Date: 11/16/2019 08:16:15 PM
-- Design Name: Binary Guessing Game
-- Module Name: top - Behavioral
-- Project Name: Project 2
-- Target Devices:
-- Tool Versions:
-- Description: This is an implementation of two buttons that work with eight switches as binary
      inputs for a guessing game that outputs high on the fifteenth LED when the correct
      combination is guessed
--
-- 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 leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity top is
   port(   sw: in std_logic_vector(7 downto 0);
           btnR: in std_logic;
           btnL: in std_logic;
           clk: in std_logic;
           led: out std_logic_vector(15 downto 15)); --Does not run unless it is defined as a
   standard_logic_vector
end top;

architecture Behavioral of top is

--Imports debounce
component debounce is
  port(   clk_100m: in std_logic;
          sw: in std_logic;
          sglPulse: out std_logic;
          sig: out std_logic);
end component;

--Declares signal for computation
signal num: std_logic_vector(7 downto 0) := "00000000";

--Declares signals for components
signal right: std_logic := '0'; --Can simplify declarations
signal left: std_logic := '0';
signal output: std_logic_vector(15 downto 15) := "0"; --Datatype fixed

begin

--Implements two debounces
d1: debounce port map(
  clk_100m => clk,
  sw => btnR,
  sglPulse => right,
  sig => open);

d2: debounce port map(
  clk_100m => clk,
  sw => btnL,
  sglPulse => left,
  sig => open);

led <= output;

process(clk)
begin
  if(rising_edge(clk)) then
```

```
        if(right = '1') then
            num <= sw;
        elsif(left = '1') then
            if(sw = num) then
                output <= "1"; --Datatype fixed
            else
                output <= "0"; --Datatype fixed
            end if;
        end if;
    end if;
end process;

end Behavioral;
```

Figure 5: Binary Guessing Game (top.vhd) Code

References

1) Nageswaran, J. M. (2009, October 4). Generate Statement. Retrieved November 16, 2019, from https://www.ics.uci.edu/~jmoorkan/vhdlref/generate.html.