DESIGN OF BASIC GATES AND I/O

Lab Report for ECE3270
Digital Systems Design

Submitted by

Ryan Barker

Department of Electrical & Computer Engineering
Clemson University

01/31/2015

Abstract

The goal of this experiment was to gain experience working on multi-stage VHDL projects through the Quartus II IDE. The final design was an interface that allowed for the user to spell the word “TIGER” on five of the Altera DE115 board’s seven segment displays and rotate “TIGER” through those displays by interfacing with the on-board switches. It was split into four parts, which involved engineering, simulating, and testing the following: An eight bit, two-to-one multiplexer, a five bit, three-to-one multiplexer, a three bit seven segment decoder, and the final design file. A critical part of the project was modularly designing the macros in each stage so they could be easily modified and re-used. The project required the user to become familiar with designing VHDL files, writing and simulating test benches, setting primary entities in Quartus, and using Quartus to download bitmaps generated by design files into an Altera board for physical implementation.

Introduction

The overall project in lab one was separated into four parts, each involving designing a different macro. Part one was concerned with constructing a two-to-one multiplexer with two eight bit inputs and an eight bit output. Part two took the core design from part one, and modified it slightly into a five-to-one multiplexer with three bit inputs and a three bit output. Part three shifted gears entirely into designing a three bit decoder to a low-active seven segment display, which would display the different characters of the word "TIGER" for its' first five input bit combinations, and display blanks for the remaining three inputs. Part four took the designs in parts two and three and combined them into a circuit which used five three bit five-to-one multiplexers and five 3 bit decoders to display the word "TIGER" across the first five seven segment displays on an Altera board. Further, the circuit in part four rotated "TIGER" across the displays as control switches were flipped ("TIGER" → "RTIGE" → "ERTIG", etc.). After each part of the lab was coded, a testbench file was written for it to assist with a ModelSim simulation to verify proper operation. Each circuit was then compiled into Quartus, and the resulting bitmap files were programmed into the Altera DE115 boards, tested, and demonstrated.

Section 1.A: Designing the 2-to-1 Multiplexer

The eight bit two-to-one multiplexer was to be built out of eight one bit two-one multiplexers that take two one bit inputs, x and y, and output one of the two dependent on the state of the select line, s. Figure 1.1.b shows a truth table for this logic, and Figure 1.1.a and Figure 1.1.c show a circuit diagram and macro for the multiplexers, respectively. As the lab instructions point out, the logic for these one bit two-to-one multiplexers is simply implemented in VHDL as a Boolean, data-flow architecture of output equals not s and x or s and y [1].

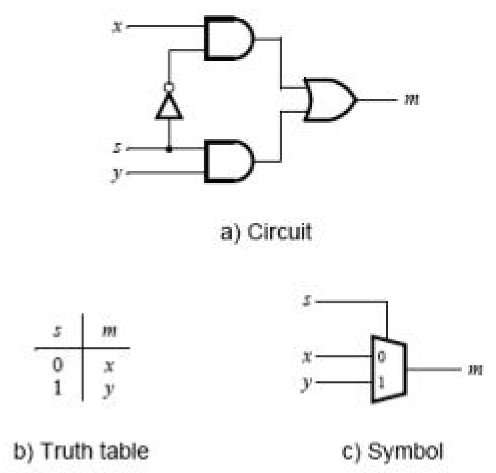


Figure 1.1. One Bit Two-to-One Multiplexer Logic. [1]

After implementing the one bit multiplexers in the code, creating the parallel eight bit variant was as simple as using a for generate statement in the overall multiplexer's architecture to port map eight, one-bit multiplexers into an eight bit entity. Figure 1.2 shows a visual representation of this coding idea. This entity was then mapped with wrapper code to physical switches and LEDs for testing purposes. All VHDL code for the eight bit, two-to-one multiplexer is seen in appendix A.1.1.

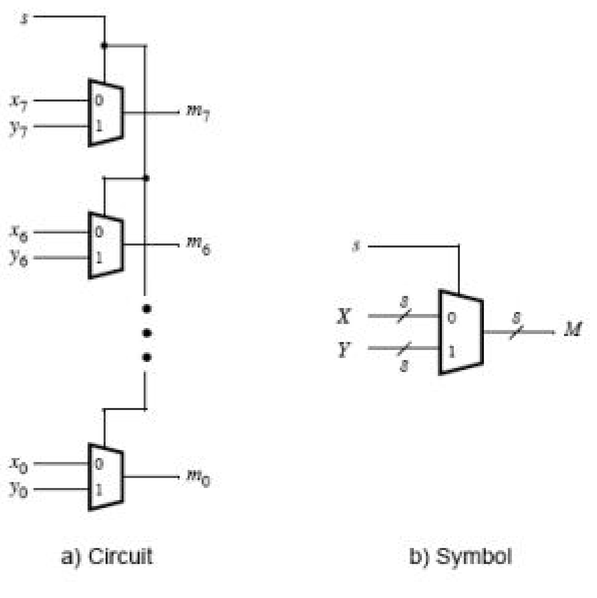


Figure 1.2. Eight Bit Two-to-One Multiplexer. [1]

Section 1.B: Testing the 2-to-1 Multiplexer

Once the eight bit multiplexer was built, writing a test bench to verify its operation was easy. To save time, this was done with Quartus II's built in "Test Bench Template Writer" to generate a .vht file. The strategy behind the test bench was to set the first eight bit input, x, to hex 0xAA and the second eight bit input, y, to hex 0x55. The control line was set to a binary zero to initially select x and changed to a one after fifty picoseconds to select y. Note that 0xAA and 0x55 are 10101010 and 01010101 in binary, respectfully, so the circuit would be deemed working if the simulation showed the output waveform starting at 10101010 and flipping polarity at the control line change. As seen below in Figure 1.3, the waveform followed this behavior, so the simulation passed and the bitmap file for the macro was sent to the board. On the board, the methods in the test bench were repeated, and the circuit was also checked to make sure the output changed as the currently selected input did. Appendix A.1.2 shows the referenced test bench.

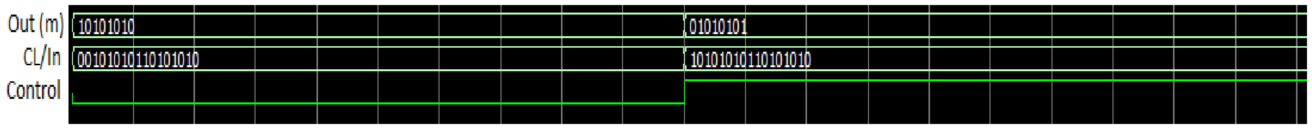


Figure 1.3. Eight Bit Two-to-One Multiplexer Simulation Wave.

Section 2.A: Designing the 5-to-1 Multiplexer

The second design was to be a five-to-one multiplexer with three bit inputs and a three bit output. Once the circuit in part one was operational, generating this design was easy and mostly involved copying the code from part one and making some slight modifications. The only major change, as seen in lines thirty through thirty-nine in the part two code, was the decision to change the Boolean architecture for the individual multiplexers from part one to a behavioral architecture that used a case when statement to describe the output in terms of the possible control line values. This was done to simplify what would have otherwise been a very long, complicated Boolean equation for the output in terms of all of the inputs. Everything else in the code follows the exact same structure as part one, just for three bits instead of eight. All VHDL code for the three bit, five-to-one multiplexer is seen in appendix A.2.1.

Section 2.B: Testing the 5-to-1 Multiplexer

This point brought a strange error in the ModelSim compiler to light: When a standard generic statement is used in an entity, and its architecture is behavioral and contains a case when statement, the ModelSim compiler generates the error “Array type case expression must be a locally static subtype” when compiling the VHDL code. Neither the lab TA nor I could explain why this happens, but fixing it is as simple as not using a generic statement and using literal values, so it was easy to overcome.

Apart from the error described above, much like designing part two followed designing part one, testing part followed testing part one. The test bench .vht file is seen in appendix A.2.2, but is very much the test bench from part one, just with five, three bit inputs instead of two, eight bit inputs. Figure 2.1 shows the relevant parts of the output waveform from part two’s simulation (Control values “000” through “100”), and shows the output waveform passed simulation, since it successfully switched to each input value as the control bit values changed. When the bitmap file for part two was sent to the actual board, it behaved in the same manner, and followed changes in its currently selected input, so it was deemed operational.

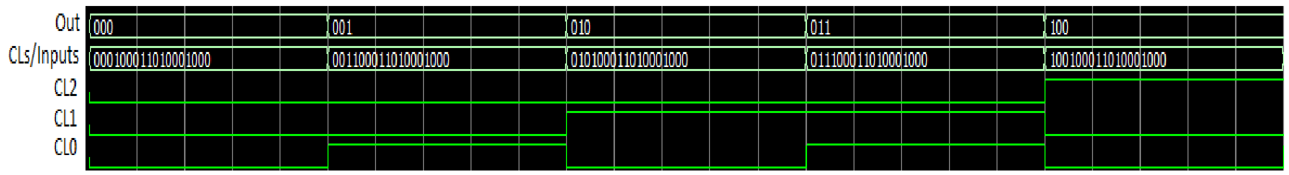


Figure 2.1. Three Bit Five-to-One Multiplexer Simulation Wave.

Section 3.A: Designing the 3 Bit Decoder

The third design in this project was a three bit decoder that was to output to a low, active seven segment display and make the characters shown in Figure 3.1 appear for each value on the input bits. Noting that a bar on a seven segment display will be on for an input value of zero on its pin, the code for this decoder was designed with a case when statement that specified each character as its binary input string to the display. The primary entity in the design was a wrapper around the decoder that mapped it physical ports on the board, so the decoder itself could be used intermediately in part four. All VHDL code for the three bit seven segment decoder is seen in appendix A.3.1.

$c_2c_1c_0$	Character
000	T
001	I
010	G
011	E
100	R
101	
110	
111	

Figure 3.1. Table of Outputs for 3 Bit Decoder. [1]

Section 3.B: Testing the 3 Bit Decoder

Simulation for this point in the project did not seem as valuable as other points in the project, since the output waveforms would not show the characters being sent to the displays. The best that could be done was to test each input value, and verify that the respective binary output value matched the value in the case when statement in the code. This is exactly what the waveform in Figure 3.2 shows. Loading the bitmap file on the board provided a much easier and meaningful check, since the characters could be seen on HEX0 for each input value. Regardless, both tests passed, and the decoder was deemed operational. The test bench used in simulation is shown in appendix A.3.2.

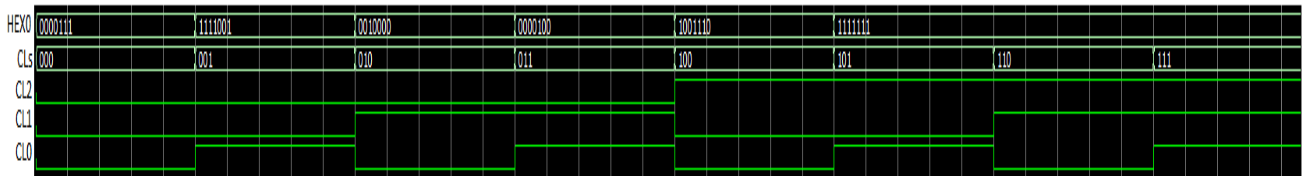


Figure 3.2. Three Bit Decoder Simulation Wave.

Section 4.A: Designing the Overall Circuit

The overall project in lab one combined the macros from parts two and three to create a circuit that could display any of the characters from the decoder on five seven segment displays. Further, each display was to start with different switches in different input positions, so the word

"TIGER" could be initially displayed and rotated across the five displays. This equated to writing port maps for five of the circuits shown in Figure 4.1 in the code, with intermediate signals for the lines between each multiplexer and decoder. All VHDL code for the overall circuit is seen in appendix A.4.1.

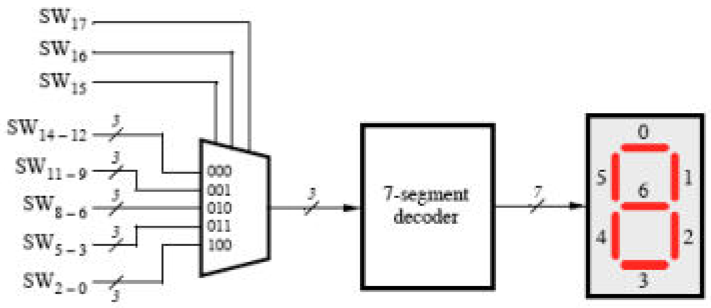


Figure 4.1. Single Component of Overall Circuit

Section 4.B: Testing the Overall Circuit

The overall project was simulated by initializing switches zero through fourteen to different input values ("T", "I", "G", "E", and "R" in that order), toggling the control switches (fifteen through seventeen) to the first five input values, and watching the bits for the word "TIGER" rotate across HEX0, HEX1, HEX2, and HEX3 in the simulation. Though it is hard to see due to the number of waveforms, the waveforms in Figure 4.1 show this behavior. Much like part three, sending the bitmap to the board was much more meaningful testing, as the characters switching across the displays could be observed in physical rather than binary form. The test bench used in simulation is shown in appendix A.4.2.

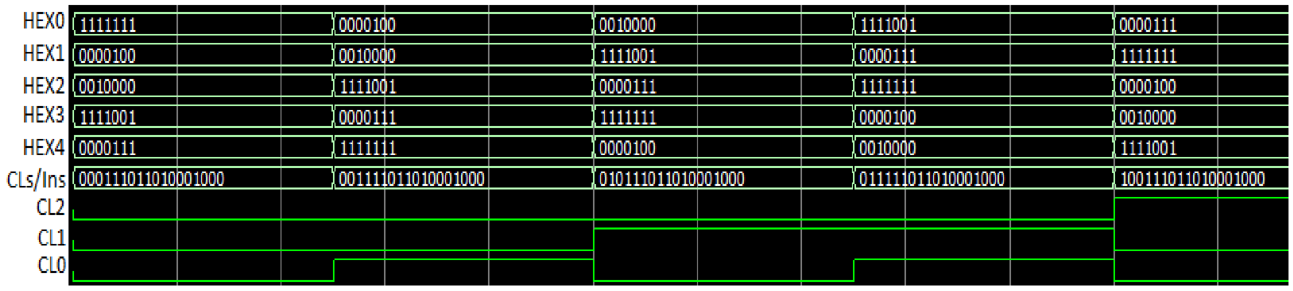


Figure 4.2. Overall Design Simulation.

Conclusions

Overall, this lab taught a great deal about using the Quartus II interface, modularly coding VHDL using wrapper entities so entities in a given VHDL file could be used in other VHDL files later, simulating waveforms with test benches and ModelSim, and generating and sending bit map files to the board. It was a great example of just how careful one needs to be using the Quartus II interface, as there is quite a bit to remember after writing code for a given design, such as setting the top level entity to that design, generating and writing a test bench, and setting Quartus up to use that test bench for simulation. It exercised one's ability to pick the most efficient architecture type (Behavioral, Dataflow, or Structural) to accomplish a given task in a given entity, and taught good coding practices, like using for generate statements to automate making similar port maps. Perhaps the biggest lesson in this lab, though, is that as far as VHDL is concerned, simpler code is better code.

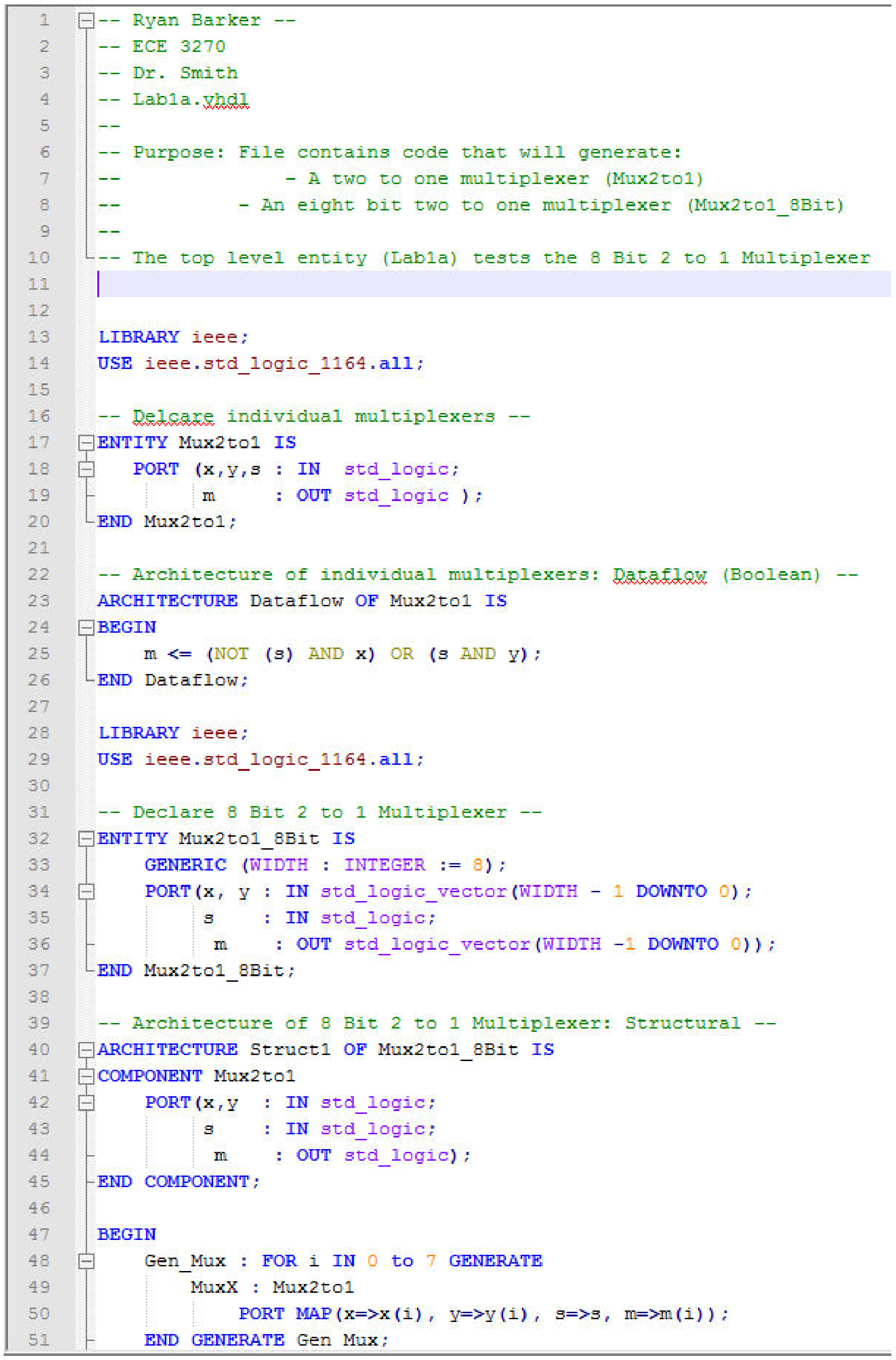
References

[1] M. Smith. *ECE327 Digital System Design, Lab 1: Design of basic gates and I/O*. [Online]. Available: https://bb.clemson.edu/bbcswebdav/pid-1872120-dt-content-rid-18310256_2/courses/smithmc-ece-327-DSD/Lab1_BASIC_GATES%282%29.pdf

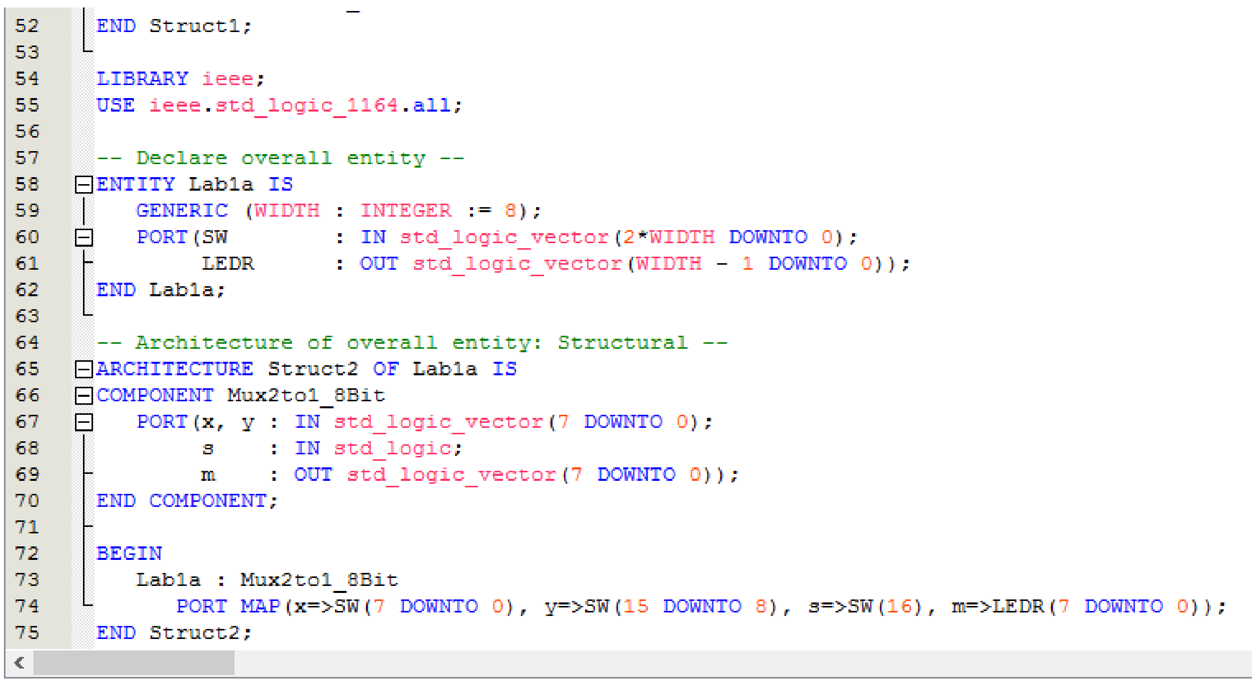
APPENDIX

Appendix A.1: 8 bit, 2-to-1 Multiplexer

A.1.1: VHDL Code

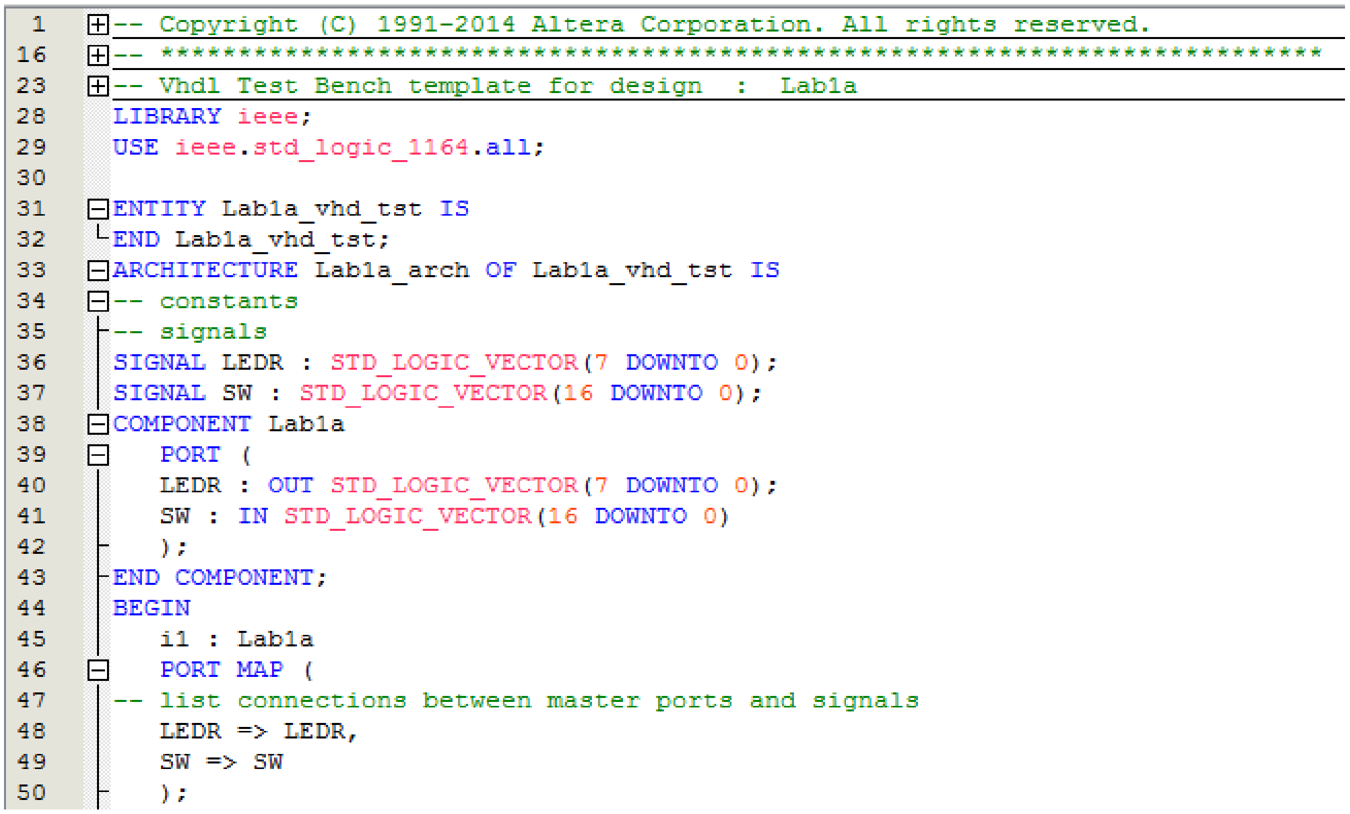


```vhdl
-- Ryan Barker --
-- ECE 3270
-- Dr. Smith
-- Lab1a.vhdl
--
-- Purpose: File contains code that will generate:
--              - A two to one multiplexer (Mux2to1)
--          - An eight bit two to one multiplexer (Mux2to1_8Bit)
--
-- The top level entity (Lab1a) tests the 8 Bit 2 to 1 Multiplexer


LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- Delcare individual multiplexers --
ENTITY Mux2to1 IS
   PORT (x,y,s : IN  std_logic;
              m     : OUT std_logic );
END Mux2to1;

-- Architecture of individual multiplexers: Dataflow (Boolean) --
ARCHITECTURE Dataflow OF Mux2to1 IS
BEGIN
    m <= (NOT (s) AND x) OR (s AND y);
END Dataflow;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- Declare 8 Bit 2 to 1 Multiplexer --
ENTITY Mux2to1_8Bit IS
    GENERIC (WIDTH : INTEGER := 8);
    PORT(x, y : IN std_logic_vector(WIDTH - 1 DOWNTO 0);
         s    : IN std_logic;
          m     : OUT std_logic_vector(WIDTH -1 DOWNTO 0));
END Mux2to1_8Bit;

-- Architecture of 8 Bit 2 to 1 Multiplexer: Structural --
ARCHITECTURE Struct1 OF Mux2to1_8Bit IS
COMPONENT Mux2to1
    PORT(x,y  : IN std_logic;
         s    : IN std_logic;
          m     : OUT std_logic);
END COMPONENT;

BEGIN
    Gen_Mux : FOR i IN 0 to 7 GENERATE
        MuxX : Mux2to1
            PORT MAP(x=>x(i), y=>y(i), s=>s, m=>m(i));
    END GENERATE Gen_Mux;
```

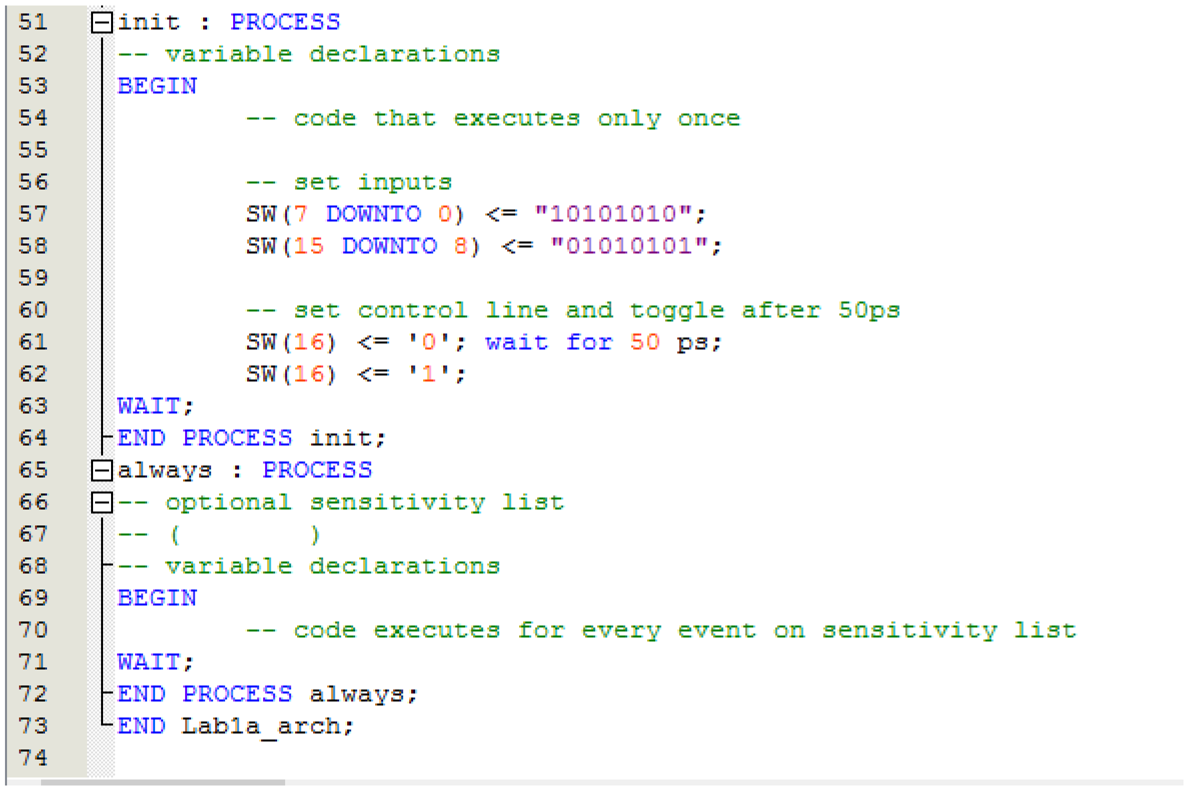


```vhdl
END Struct1;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- Declare overall entity --
ENTITY Lab1a IS
   GENERIC (WIDTH : INTEGER := 8);
   PORT(SW        : IN std_logic_vector(2*WIDTH DOWNTO 0);
        LEDR      : OUT std_logic_vector(WIDTH - 1 DOWNTO 0));
END Lab1a;

-- Architecture of overall entity: Structural --
ARCHITECTURE Struct2 OF Lab1a IS
COMPONENT Mux2to1_8Bit
   PORT(x, y : IN std_logic_vector(7 DOWNTO 0);
        s    : IN std_logic;
        m    : OUT std_logic_vector(7 DOWNTO 0));
END COMPONENT;

BEGIN
   Lab1a : Mux2to1_8Bit
      PORT MAP(x=>SW(7 DOWNTO 0), y=>SW(15 DOWNTO 8), s=>SW(16), m=>LEDR(7 DOWNTO 0));
END Struct2;
```

A.1.2: Test Bench



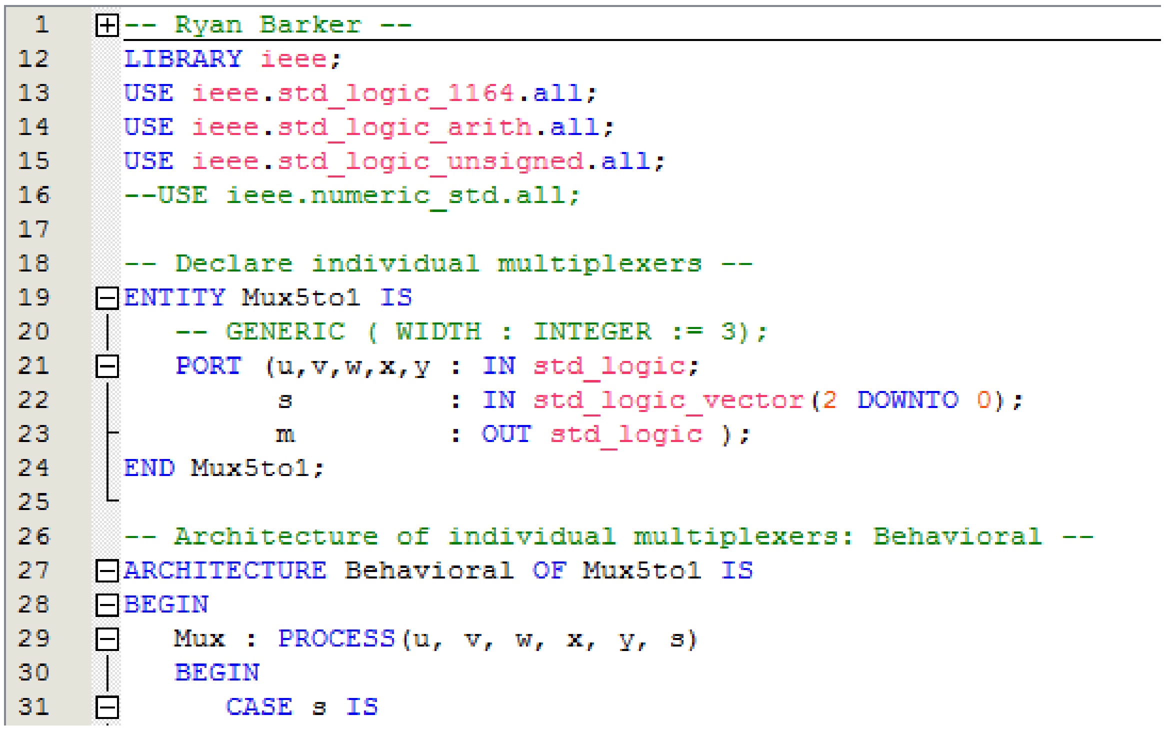
```vhdl
-- Copyright (C) 1991-2014 Altera Corporation. All rights reserved.
-- ***************************************************************************
-- Vhdl Test Bench template for design  :  Lab1a
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Lab1a_vhd_tst IS
END Lab1a_vhd_tst;
ARCHITECTURE Lab1a_arch OF Lab1a_vhd_tst IS
-- constants
-- signals
SIGNAL LEDR : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL SW : STD_LOGIC_VECTOR(16 DOWNTO 0);
COMPONENT Lab1a
   PORT (
   LEDR : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
   SW : IN STD_LOGIC_VECTOR(16 DOWNTO 0)
   );
END COMPONENT;
BEGIN
   i1 : Lab1a
   PORT MAP (
-- list connections between master ports and signals
   LEDR => LEDR,
   SW => SW
   );
```



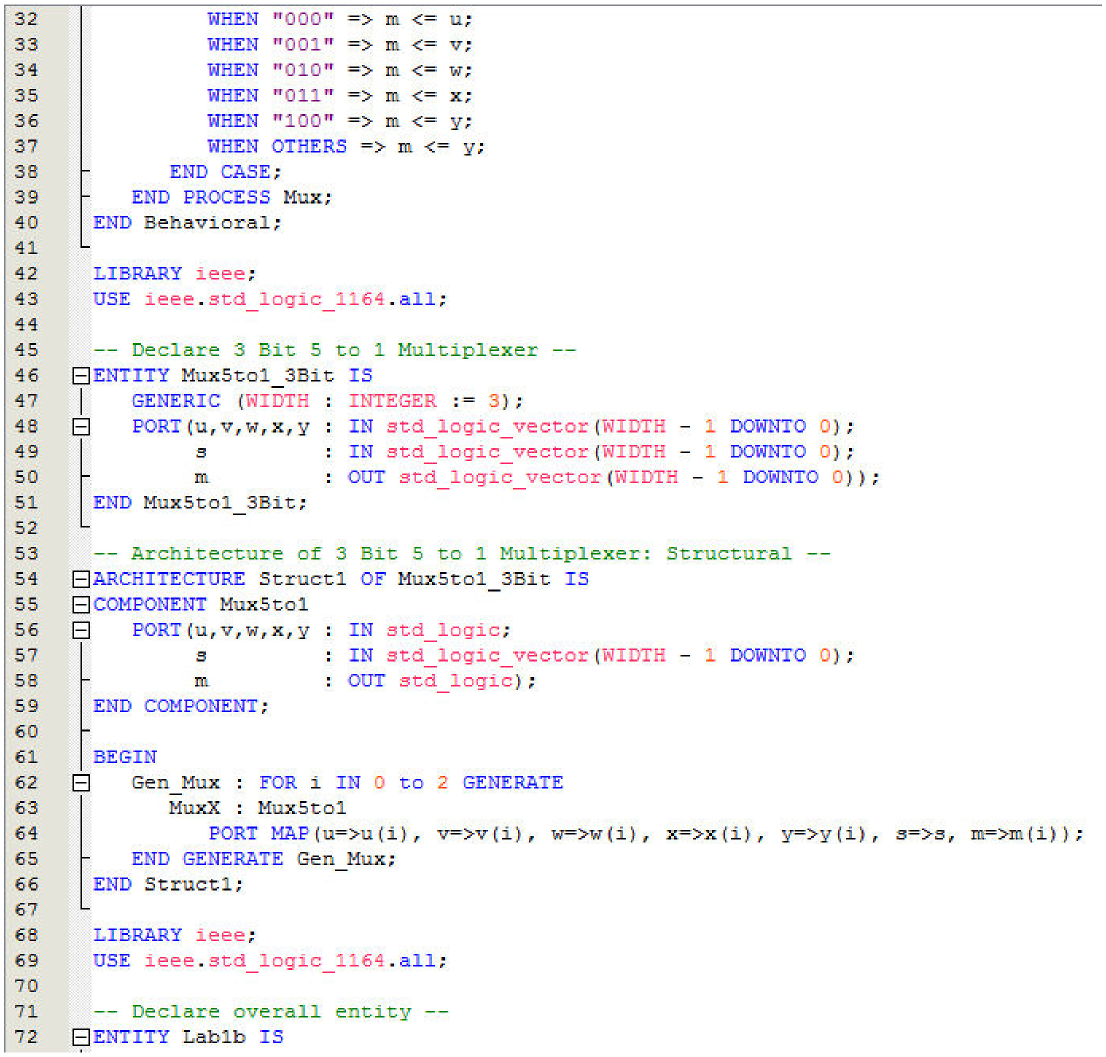
```vhdl
init : PROCESS
-- variable declarations
BEGIN
        -- code that executes only once

        -- set inputs
        SW(7 DOWNTO 0) <= "10101010";
        SW(15 DOWNTO 8) <= "01010101";

        -- set control line and toggle after 50ps
        SW(16) <= '0'; wait for 50 ps;
        SW(16) <= '1';
WAIT;
END PROCESS init;
always : PROCESS
-- optional sensitivity list
-- (         )
-- variable declarations
BEGIN
        -- code executes for every event on sensitivity list
WAIT;
END PROCESS always;
END Lab1a_arch;

```

Appendix A.2: 3 bit, 5-to-1 Multiplexer

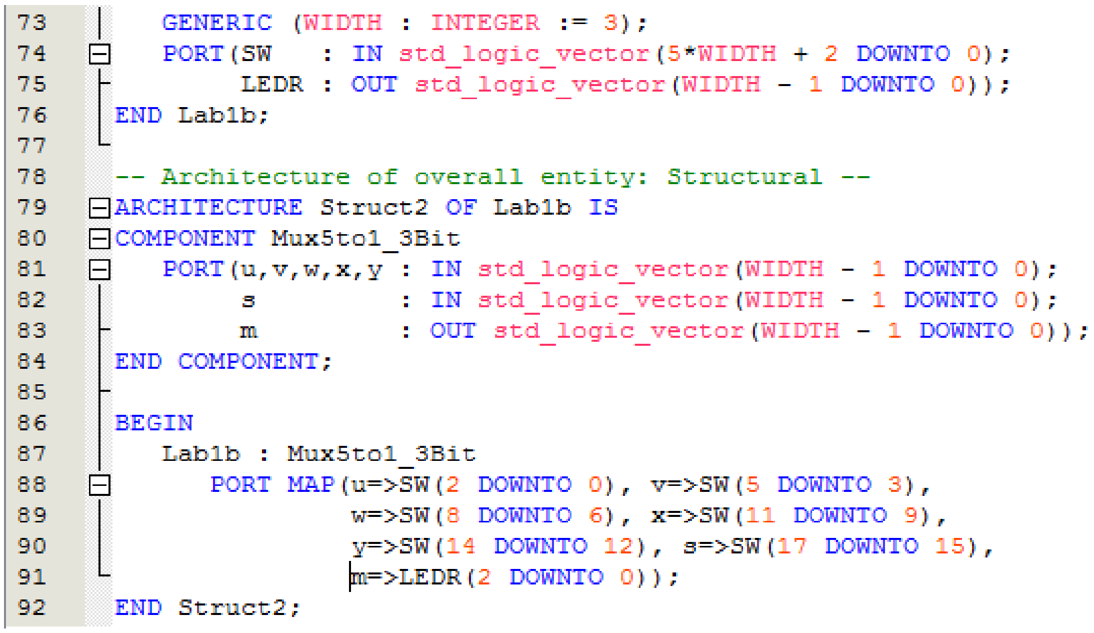
A.2.1: VHDL Code



```vhdl
-- Ryan Barker --
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
--USE ieee.numeric_std.all;

-- Declare individual multiplexers --
ENTITY Mux5to1 IS
   -- GENERIC ( WIDTH : INTEGER := 3);
   PORT (u,v,w,x,y : IN std_logic;
         s         : IN std_logic_vector(2 DOWNTO 0);
         m         : OUT std_logic );
END Mux5to1;

-- Architecture of individual multiplexers: Behavioral --
ARCHITECTURE Behavioral OF Mux5to1 IS
BEGIN
   Mux : PROCESS(u, v, w, x, y, s)
   BEGIN
      CASE s IS
```

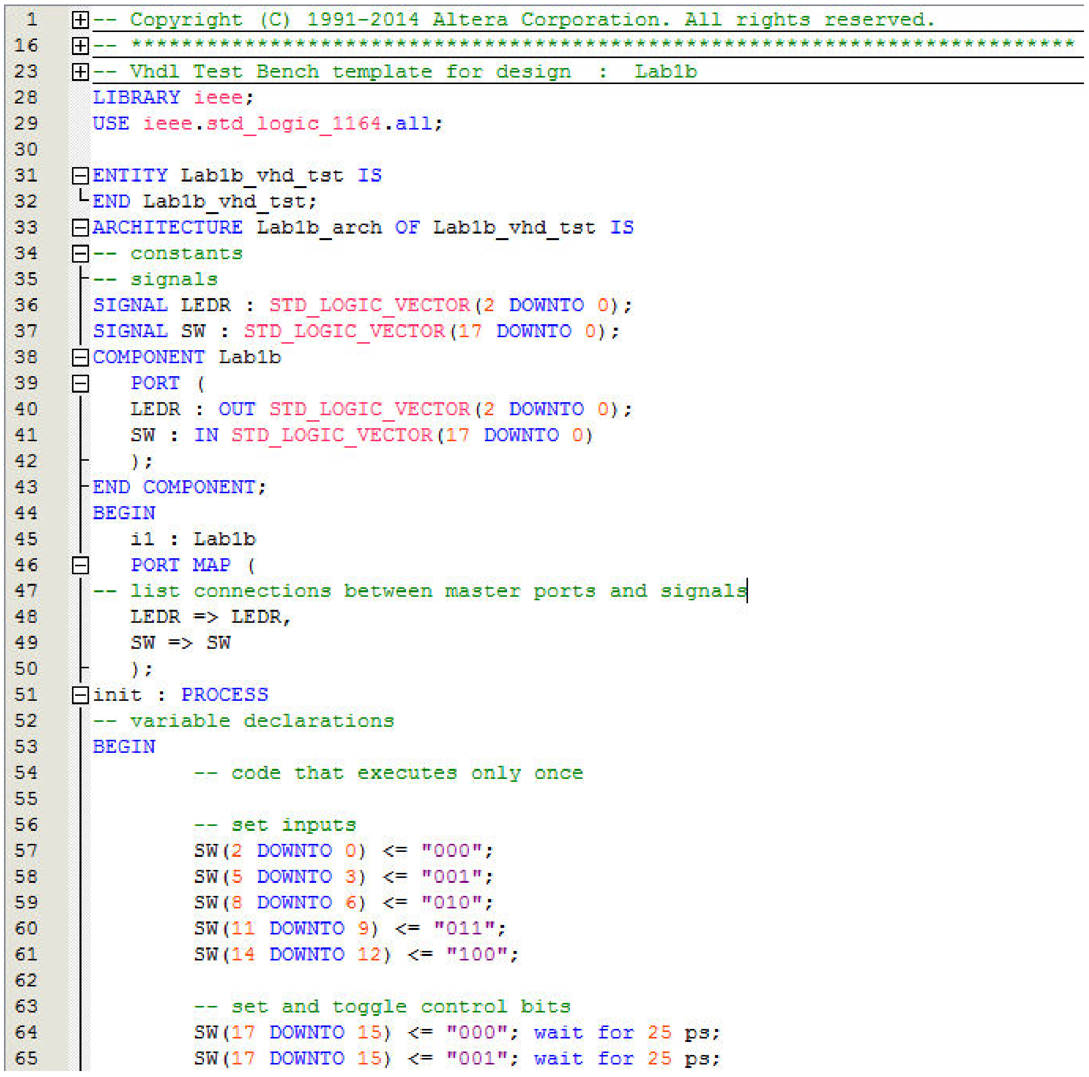


```
         WHEN "000" => m <= u;
         WHEN "001" => m <= v;
         WHEN "010" => m <= w;
         WHEN "011" => m <= x;
         WHEN "100" => m <= y;
         WHEN OTHERS => m <= y;
      END CASE;
   END PROCESS Mux;
END Behavioral;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- Declare 3 Bit 5 to 1 Multiplexer --
ENTITY Mux5to1_3Bit IS
   GENERIC (WIDTH : INTEGER := 3);
   PORT(u,v,w,x,y : IN std_logic_vector(WIDTH - 1 DOWNTO 0);
        s         : IN std_logic_vector(WIDTH - 1 DOWNTO 0);
        m         : OUT std_logic_vector(WIDTH - 1 DOWNTO 0));
END Mux5to1_3Bit;

-- Architecture of 3 Bit 5 to 1 Multiplexer: Structural --
ARCHITECTURE Struct1 OF Mux5to1_3Bit IS
COMPONENT Mux5to1
   PORT(u,v,w,x,y : IN std_logic;
        s         : IN std_logic_vector(WIDTH - 1 DOWNTO 0);
        m         : OUT std_logic);
END COMPONENT;

BEGIN
   Gen_Mux : FOR i IN 0 to 2 GENERATE
      MuxX : Mux5to1
         PORT MAP(u=>u(i), v=>v(i), w=>w(i), x=>x(i), y=>y(i), s=>s, m=>m(i));
   END GENERATE Gen_Mux;
END Struct1;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- Declare overall entity --
ENTITY Lab1b IS
```

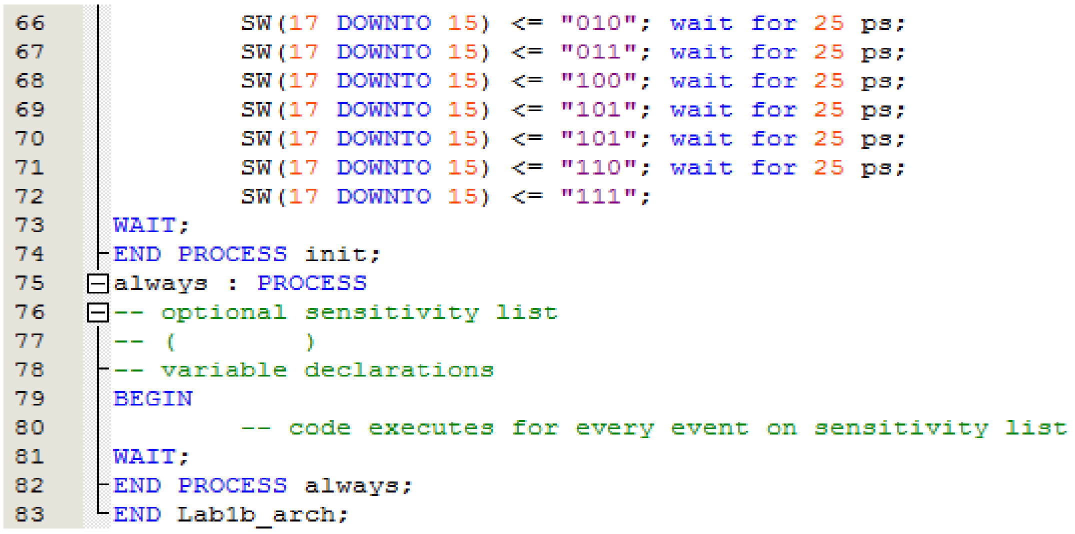


```
   GENERIC (WIDTH : INTEGER := 3);
   PORT(SW   : IN std_logic_vector(5*WIDTH + 2 DOWNTO 0);
        LEDR : OUT std_logic_vector(WIDTH - 1 DOWNTO 0));
END Lab1b;

-- Architecture of overall entity: Structural --
ARCHITECTURE Struct2 OF Lab1b IS
COMPONENT Mux5to1_3Bit
   PORT(u,v,w,x,y : IN std_logic_vector(WIDTH - 1 DOWNTO 0);
        s         : IN std_logic_vector(WIDTH - 1 DOWNTO 0);
        m         : OUT std_logic_vector(WIDTH - 1 DOWNTO 0));
END COMPONENT;

BEGIN
   Lab1b : Mux5to1_3Bit
      PORT MAP(u=>SW(2 DOWNTO 0), v=>SW(5 DOWNTO 3),
               w=>SW(8 DOWNTO 6), x=>SW(11 DOWNTO 9),
               y=>SW(14 DOWNTO 12), s=>SW(17 DOWNTO 15),
               m=>LEDR(2 DOWNTO 0));
END Struct2;
```

A.2.2: Test Bench



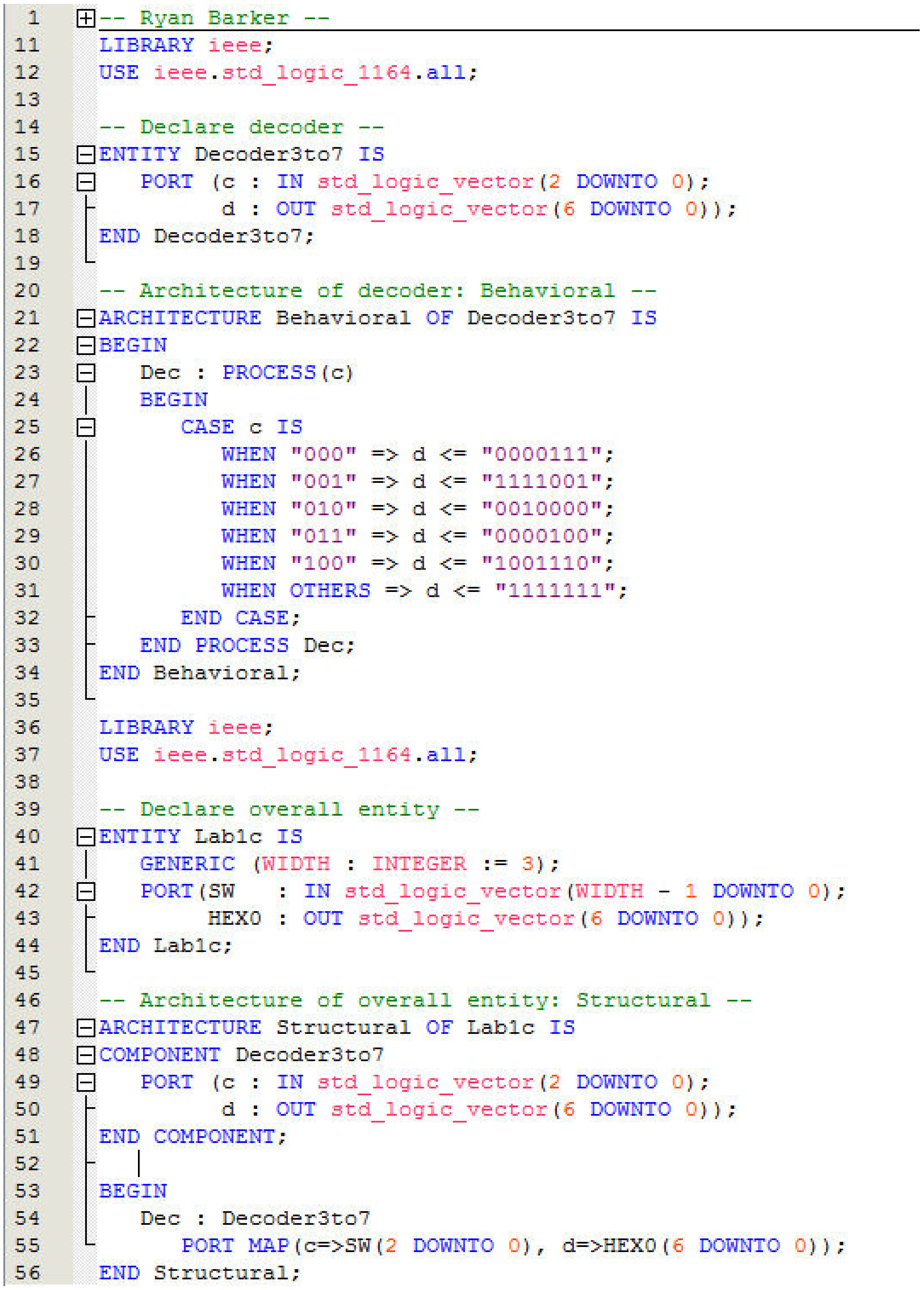
```
-- Copyright (C) 1991-2014 Altera Corporation. All rights reserved.
-- ***************************************************************************
-- Vhdl Test Bench template for design  :  Lab1b
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Lab1b_vhd_tst IS
END Lab1b_vhd_tst;
ARCHITECTURE Lab1b_arch OF Lab1b_vhd_tst IS
-- constants
-- signals
SIGNAL LEDR : STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL SW : STD_LOGIC_VECTOR(17 DOWNTO 0);
COMPONENT Lab1b
   PORT (
   LEDR : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
   SW : IN STD_LOGIC_VECTOR(17 DOWNTO 0)
   );
END COMPONENT;
BEGIN
   i1 : Lab1b
   PORT MAP (
-- list connections between master ports and signals
   LEDR => LEDR,
   SW => SW
   );
init : PROCESS
-- variable declarations
BEGIN
        -- code that executes only once

        -- set inputs
        SW(2 DOWNTO 0) <= "000";
        SW(5 DOWNTO 3) <= "001";
        SW(8 DOWNTO 6) <= "010";
        SW(11 DOWNTO 9) <= "011";
        SW(14 DOWNTO 12) <= "100";

        -- set and toggle control bits
        SW(17 DOWNTO 15) <= "000"; wait for 25 ps;
        SW(17 DOWNTO 15) <= "001"; wait for 25 ps;
```



```
        SW(17 DOWNTO 15) <= "010"; wait for 25 ps;
        SW(17 DOWNTO 15) <= "011"; wait for 25 ps;
        SW(17 DOWNTO 15) <= "100"; wait for 25 ps;
        SW(17 DOWNTO 15) <= "101"; wait for 25 ps;
        SW(17 DOWNTO 15) <= "101"; wait for 25 ps;
        SW(17 DOWNTO 15) <= "110"; wait for 25 ps;
        SW(17 DOWNTO 15) <= "111";
WAIT;
END PROCESS init;
always : PROCESS
-- optional sensitivity list
-- (          )
-- variable declarations
BEGIN
        -- code executes for every event on sensitivity list
WAIT;
END PROCESS always;
END Lab1b_arch;
```

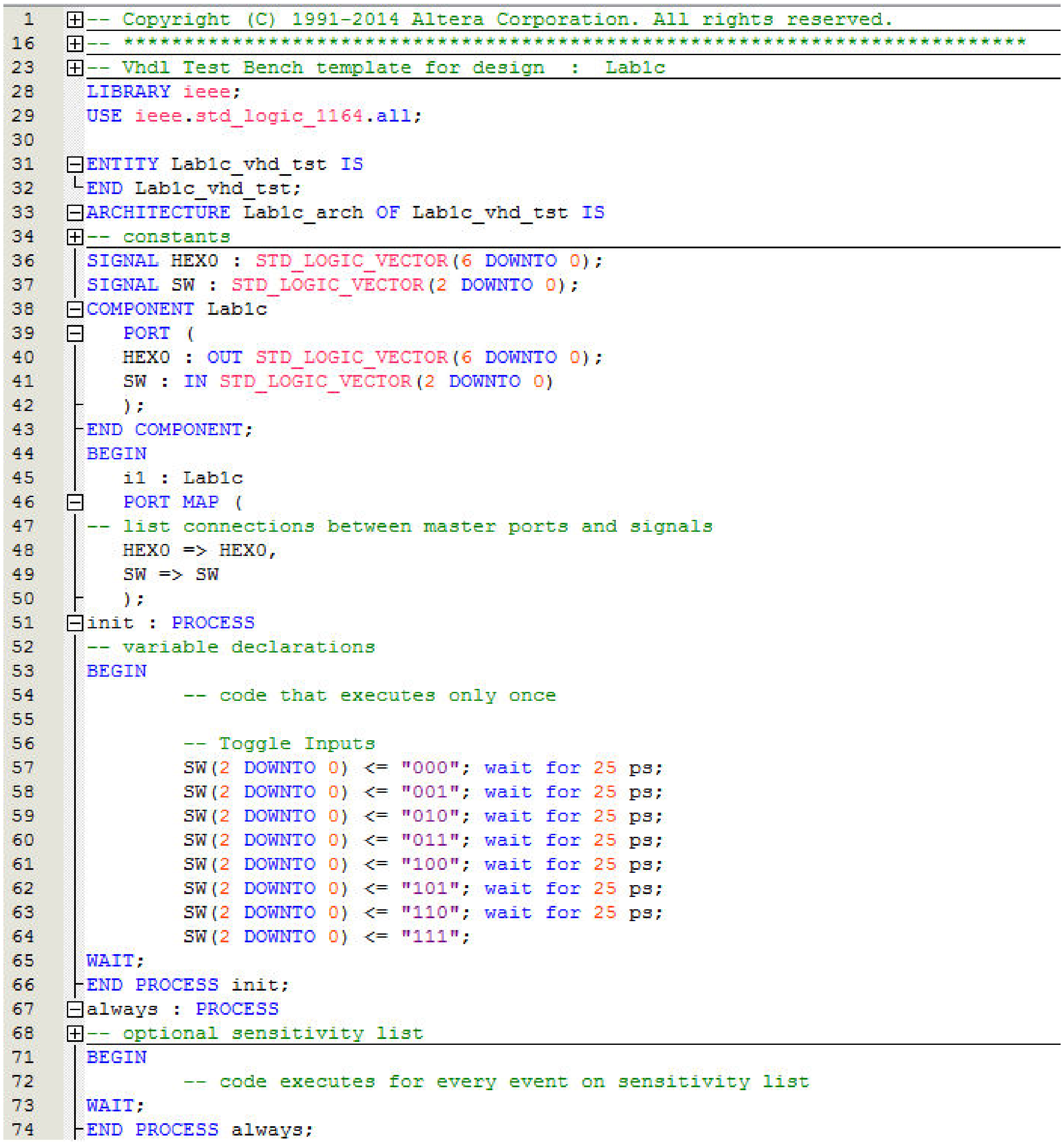
Appendix A.3: 3 bit Decoder

A.3.1: VHDL Code



```
-- Ryan Barker --
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- Declare decoder --
ENTITY Decoder3to7 IS
   PORT (c : IN std_logic_vector(2 DOWNTO 0);
         d : OUT std_logic_vector(6 DOWNTO 0));
END Decoder3to7;

-- Architecture of decoder: Behavioral --
ARCHITECTURE Behavioral OF Decoder3to7 IS
BEGIN
   Dec : PROCESS(c)
   BEGIN
      CASE c IS
         WHEN "000" => d <= "0000111";
         WHEN "001" => d <= "1111001";
         WHEN "010" => d <= "0010000";
         WHEN "011" => d <= "0000100";
         WHEN "100" => d <= "1001110";
         WHEN OTHERS => d <= "1111111";
      END CASE;
   END PROCESS Dec;
END Behavioral;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- Declare overall entity --
ENTITY Lab1c IS
   GENERIC (WIDTH : INTEGER := 3);
   PORT(SW    : IN std_logic_vector(WIDTH - 1 DOWNTO 0);
        HEX0 : OUT std_logic_vector(6 DOWNTO 0));
END Lab1c;

-- Architecture of overall entity: Structural --
ARCHITECTURE Structural OF Lab1c IS
COMPONENT Decoder3to7
   PORT (c : IN std_logic_vector(2 DOWNTO 0);
         d : OUT std_logic_vector(6 DOWNTO 0));
END COMPONENT;

BEGIN
   Dec : Decoder3to7
      PORT MAP(c=>SW(2 DOWNTO 0), d=>HEX0(6 DOWNTO 0));
END Structural;
```

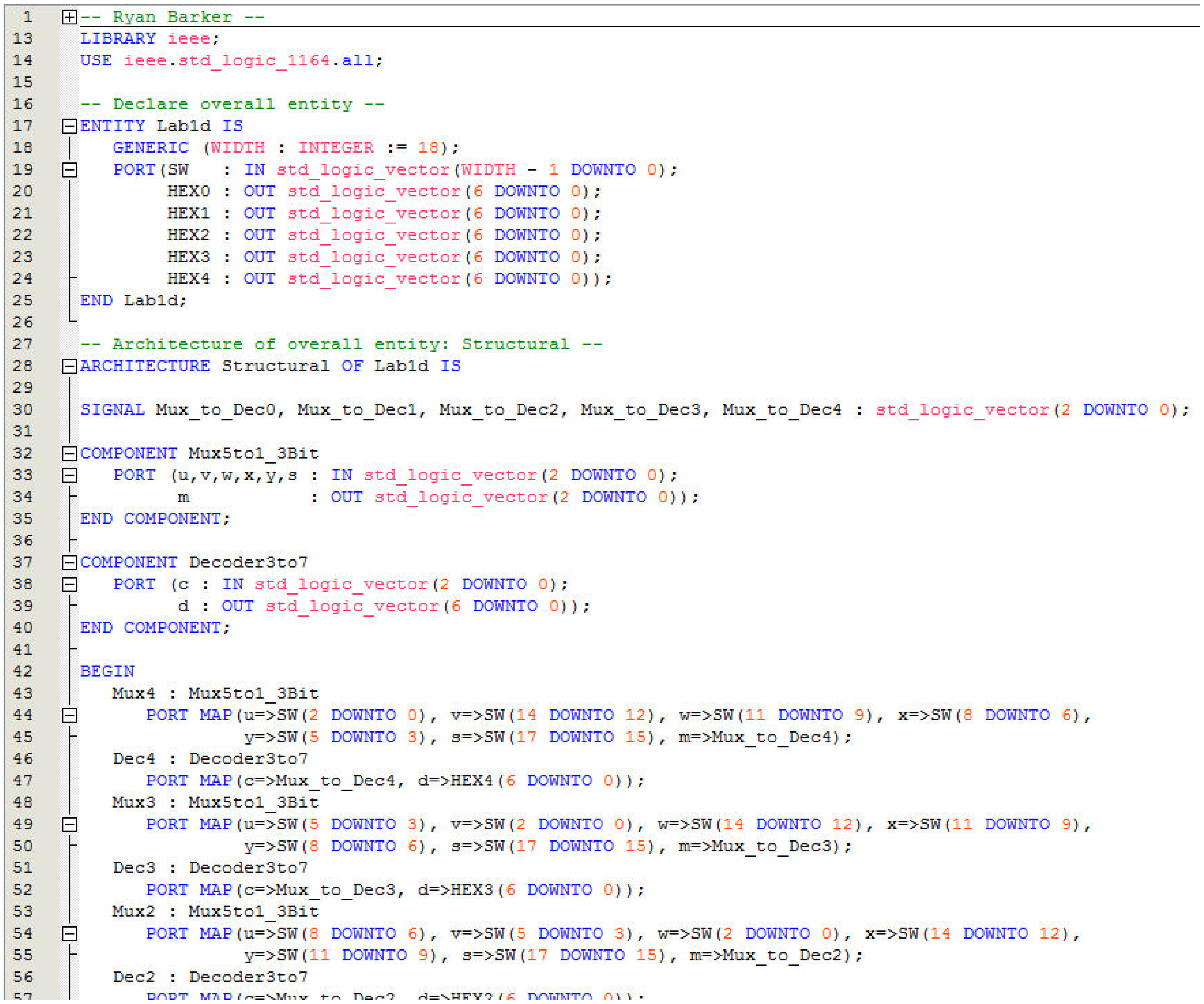
A.3.2: Test Bench



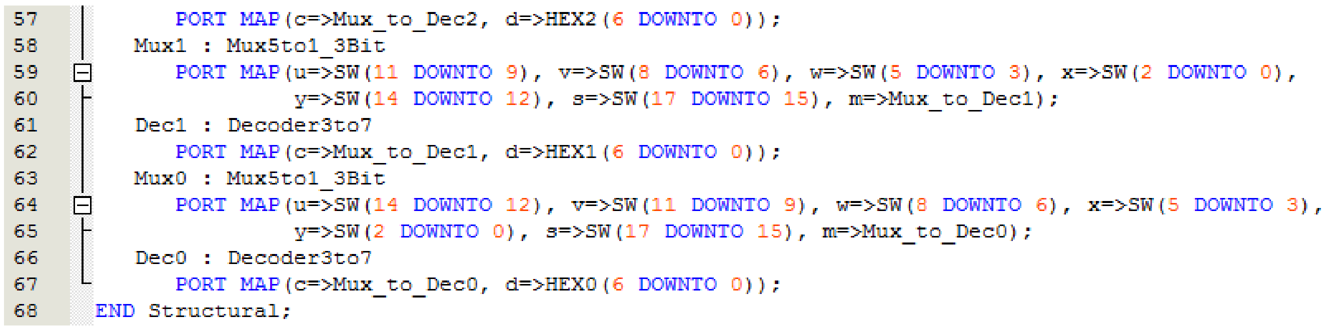
```vhdl
-- Copyright (C) 1991-2014 Altera Corporation. All rights reserved.
-- ****************************************************************************
-- Vhdl Test Bench template for design  :  Lab1c
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Lab1c_vhd_tst IS
END Lab1c_vhd_tst;
ARCHITECTURE Lab1c_arch OF Lab1c_vhd_tst IS
-- constants
SIGNAL HEX0 : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SW : STD_LOGIC_VECTOR(2 DOWNTO 0);
COMPONENT Lab1c
   PORT (
   HEX0 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
   SW : IN STD_LOGIC_VECTOR(2 DOWNTO 0)
   );
END COMPONENT;
BEGIN
   i1 : Lab1c
   PORT MAP (
-- list connections between master ports and signals
   HEX0 => HEX0,
   SW => SW
   );
init : PROCESS
-- variable declarations
BEGIN
        -- code that executes only once

        -- Toggle Inputs
        SW(2 DOWNTO 0) <= "000"; wait for 25 ps;
        SW(2 DOWNTO 0) <= "001"; wait for 25 ps;
        SW(2 DOWNTO 0) <= "010"; wait for 25 ps;
        SW(2 DOWNTO 0) <= "011"; wait for 25 ps;
        SW(2 DOWNTO 0) <= "100"; wait for 25 ps;
        SW(2 DOWNTO 0) <= "101"; wait for 25 ps;
        SW(2 DOWNTO 0) <= "110"; wait for 25 ps;
        SW(2 DOWNTO 0) <= "111";
WAIT;
END PROCESS init;
always : PROCESS
-- optional sensitivity list
BEGIN
        -- code executes for every event on sensitivity list
WAIT;
END PROCESS always;
```

Appendix A.4: Overall Design

A.4.1: VHDL Code

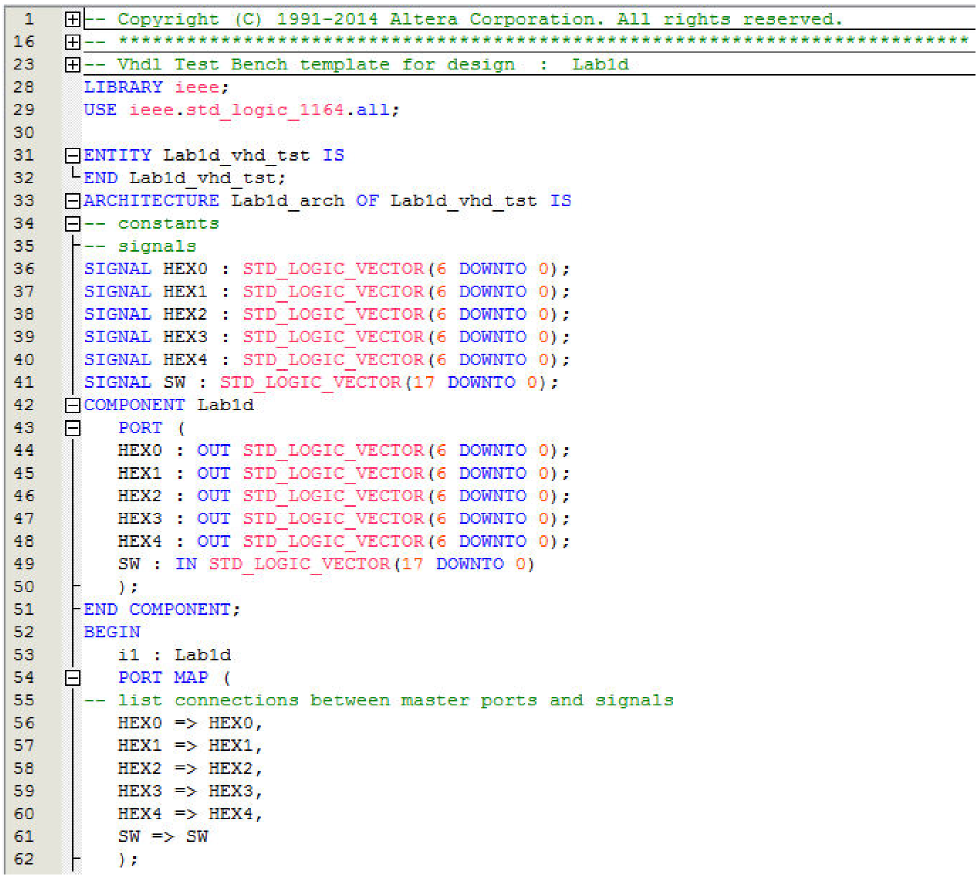


```vhdl
-- Ryan Barker --
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- Declare overall entity --
ENTITY Lab1d IS
   GENERIC (WIDTH : INTEGER := 18);
   PORT(SW    : IN std_logic_vector(WIDTH - 1 DOWNTO 0);
        HEX0 : OUT std_logic_vector(6 DOWNTO 0);
        HEX1 : OUT std_logic_vector(6 DOWNTO 0);
        HEX2 : OUT std_logic_vector(6 DOWNTO 0);
        HEX3 : OUT std_logic_vector(6 DOWNTO 0);
        HEX4 : OUT std_logic_vector(6 DOWNTO 0));
END Lab1d;

-- Architecture of overall entity: Structural --
ARCHITECTURE Structural OF Lab1d IS

SIGNAL Mux_to_Dec0, Mux_to_Dec1, Mux_to_Dec2, Mux_to_Dec3, Mux_to_Dec4 : std_logic_vector(2 DOWNTO 0);

COMPONENT Mux5to1_3Bit
   PORT (u,v,w,x,y,s : IN std_logic_vector(2 DOWNTO 0);
         m           : OUT std_logic_vector(2 DOWNTO 0));
END COMPONENT;

COMPONENT Decoder3to7
   PORT (c : IN std_logic_vector(2 DOWNTO 0);
         d : OUT std_logic_vector(6 DOWNTO 0));
END COMPONENT;

BEGIN
   Mux4 : Mux5to1_3Bit
      PORT MAP(u=>SW(2 DOWNTO 0), v=>SW(14 DOWNTO 12), w=>SW(11 DOWNTO 9), x=>SW(8 DOWNTO 6),
               y=>SW(5 DOWNTO 3), s=>SW(17 DOWNTO 15), m=>Mux_to_Dec4);
   Dec4 : Decoder3to7
      PORT MAP(c=>Mux_to_Dec4, d=>HEX4(6 DOWNTO 0));
   Mux3 : Mux5to1_3Bit
      PORT MAP(u=>SW(5 DOWNTO 3), v=>SW(2 DOWNTO 0), w=>SW(14 DOWNTO 12), x=>SW(11 DOWNTO 9),
               y=>SW(8 DOWNTO 6), s=>SW(17 DOWNTO 15), m=>Mux_to_Dec3);
   Dec3 : Decoder3to7
      PORT MAP(c=>Mux_to_Dec3, d=>HEX3(6 DOWNTO 0));
   Mux2 : Mux5to1_3Bit
      PORT MAP(u=>SW(8 DOWNTO 6), v=>SW(5 DOWNTO 3), w=>SW(2 DOWNTO 0), x=>SW(14 DOWNTO 12),
               y=>SW(11 DOWNTO 9), s=>SW(17 DOWNTO 15), m=>Mux_to_Dec2);
   Dec2 : Decoder3to7
```

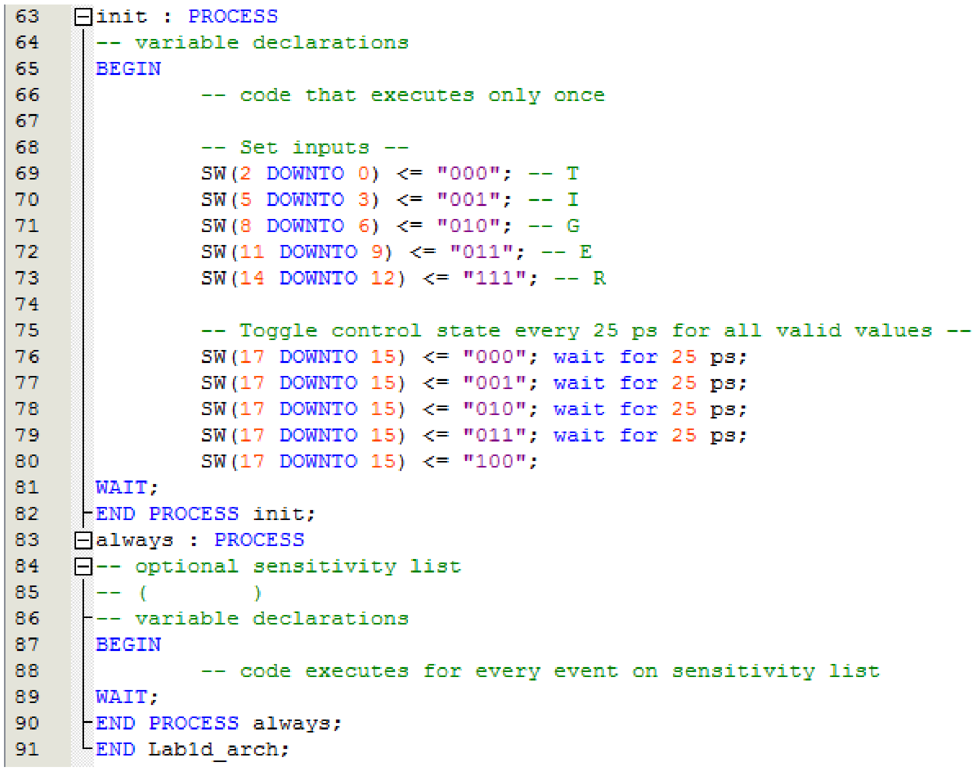


```vhdl
      PORT MAP(c=>Mux_to_Dec2, d=>HEX2(6 DOWNTO 0));
   Mux1 : Mux5to1_3Bit
      PORT MAP(u=>SW(11 DOWNTO 9), v=>SW(8 DOWNTO 6), w=>SW(5 DOWNTO 3), x=>SW(2 DOWNTO 0),
               y=>SW(14 DOWNTO 12), s=>SW(17 DOWNTO 15), m=>Mux_to_Dec1);
   Dec1 : Decoder3to7
      PORT MAP(c=>Mux_to_Dec1, d=>HEX1(6 DOWNTO 0));
   Mux0 : Mux5to1_3Bit
      PORT MAP(u=>SW(14 DOWNTO 12), v=>SW(11 DOWNTO 9), w=>SW(8 DOWNTO 6), x=>SW(5 DOWNTO 3),
               y=>SW(2 DOWNTO 0), s=>SW(17 DOWNTO 15), m=>Mux_to_Dec0);
   Dec0 : Decoder3to7
      PORT MAP(c=>Mux_to_Dec0, d=>HEX0(6 DOWNTO 0));
END Structural;
```

A.4.2: Test Bench



```vhdl
-- Copyright (C) 1991-2014 Altera Corporation. All rights reserved.
-- ************************************************************************
-- Vhdl Test Bench template for design  :  Lab1d
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Lab1d_vhd_tst IS
END Lab1d_vhd_tst;
ARCHITECTURE Lab1d_arch OF Lab1d_vhd_tst IS
-- constants
-- signals
SIGNAL HEX0 : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL HEX1 : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL HEX2 : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL HEX3 : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL HEX4 : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SW : STD_LOGIC_VECTOR(17 DOWNTO 0);
COMPONENT Lab1d
   PORT (
   HEX0 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
   HEX1 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
   HEX2 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
   HEX3 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
   HEX4 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
   SW : IN STD_LOGIC_VECTOR(17 DOWNTO 0)
   );
END COMPONENT;
BEGIN
   i1 : Lab1d
   PORT MAP (
-- list connections between master ports and signals
   HEX0 => HEX0,
   HEX1 => HEX1,
   HEX2 => HEX2,
   HEX3 => HEX3,
   HEX4 => HEX4,
   SW => SW
   );
```



```vhdl
init : PROCESS
-- variable declarations
BEGIN
        -- code that executes only once

        -- Set inputs --
        SW(2 DOWNTO 0) <= "000"; -- T
        SW(5 DOWNTO 3) <= "001"; -- I
        SW(8 DOWNTO 6) <= "010"; -- G
        SW(11 DOWNTO 9) <= "011"; -- E
        SW(14 DOWNTO 12) <= "111"; -- R

        -- Toggle control state every 25 ps for all valid values --
        SW(17 DOWNTO 15) <= "000"; wait for 25 ps;
        SW(17 DOWNTO 15) <= "001"; wait for 25 ps;
        SW(17 DOWNTO 15) <= "010"; wait for 25 ps;
        SW(17 DOWNTO 15) <= "011"; wait for 25 ps;
        SW(17 DOWNTO 15) <= "100";
WAIT;
END PROCESS init;
always : PROCESS
-- optional sensitivity list
-- (          )
-- variable declarations
BEGIN
        -- code executes for every event on sensitivity list
WAIT;
END PROCESS always;
END Lab1d_arch;
```