Seven-Segment Display Controller Using Counters (Feb 2020)

Joshua A. Rothe, *Student, Johns Hopkins University, Whiting School of Engineering*

Table of Contents

I. Introduction

THIS task took the concepts from the previous labs (using processes to control a 7-segment display) and built on it by adding additional functionality by way of a shift register and a more advanced controller to display different scrolling values simultaneously. This was done by using pulse generators to generate the enable pulses for both the 7-segment display (toggling anodes at the specified refresh rate to display multiple different values as needed) as well as the much slower shifting of values across the display. Each module was coded and tested individually before the final design was programmed into the Nexys 4 DDR Development Board and the functionality verified.

II. Procedure

A. *pulseGenerator.vhd*

The pulse generator was coded as a simple module that took the clock signal and used a count to output a pulse that triggers only once every N clock cycles, with N being a generic that is configurable as a top-level constant. This allows the module to be reused multiple times – it is good to strive for reusability and to standardize modules whenever possible.

B. *seg7_controller.vhd*

The 7-segment controller instantiated the seg7hex.vhd module from previous labs as well as a pulse generator that generated an enable signal at a refresh rate of 1 kHz (assuming the top level clock is 100 MHz – the pulse generator divides the clock by a certain number of counts so if you simulate at a faster speed, the functionality will be the same albeit sped up for simulation purposes).

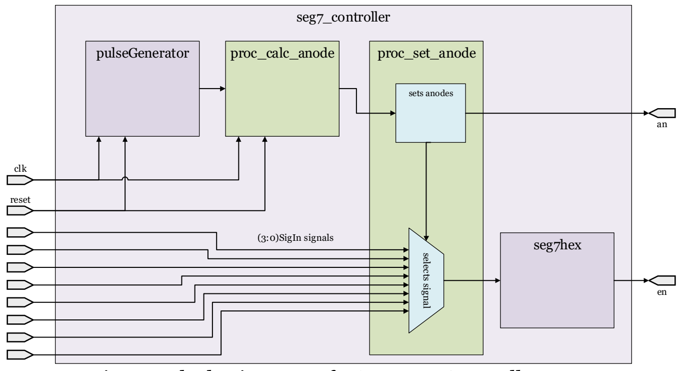

Fig. 1 – Block Diagram of 7 Segment Controller

Using the enable pulse it cycled through the different anodes and 7 segment inputs, so that all 8 different individual inputs were displayed on the 8 different displays.

C. *shiftReg*

The shift register constantly held the output values for the 7-segment controller to allow it to read and display these values on the board. Using a very slow pulse of 1 Hz (to allow for a visible scrolling effect to the human eye) it was able to take in values from switches 3 through 0 as inputs and shift them through the registers as specified in the lab prompt.

D. *lab3_top.vhd*

All of the above modules were instantiated and connected in a top level. The constant values for the generics were also initialized to allow the pulse generators to function at the proper speeds.

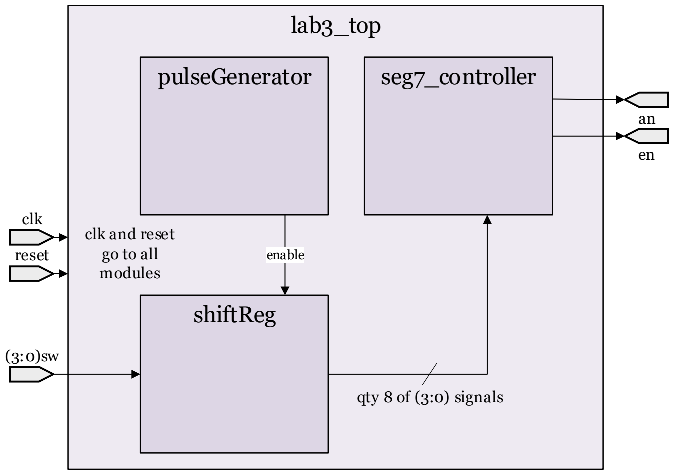

Fig. 2 – Block Diagram of Lab 3

As before, the LEDs also display all switch activity on the board (not shown in the block diagram).

III. Testing Strategy

Testbenches for both the 7-segment controller as well as the shift register were created to verify functionality of both modules before instantiating them in the top module. This ended up being helpful in debugging, as the constant declaration was incorrect which caused issues with the scrolling and display of values (using integers fixed this nicely).

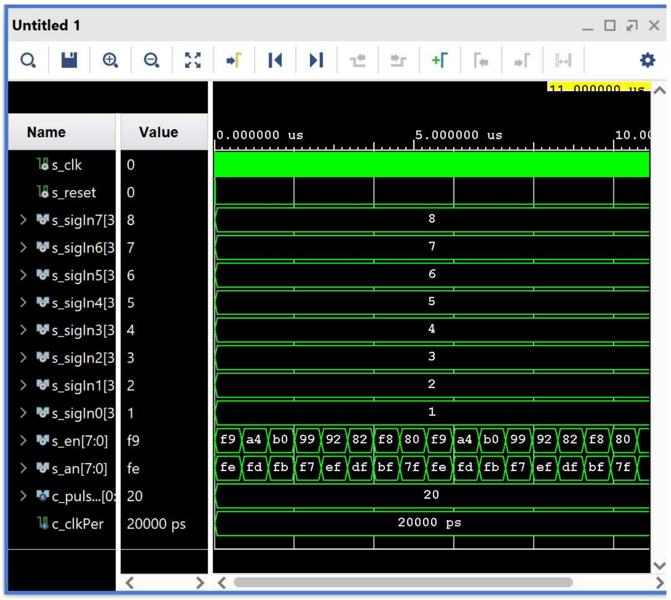

Fig. 3 – 7-Segment Display Testbench Waveform

The above testbench was simple – with different values for each input (and a sped-up pulse), the module cycled through the values for the anodes and the 7-segment display.

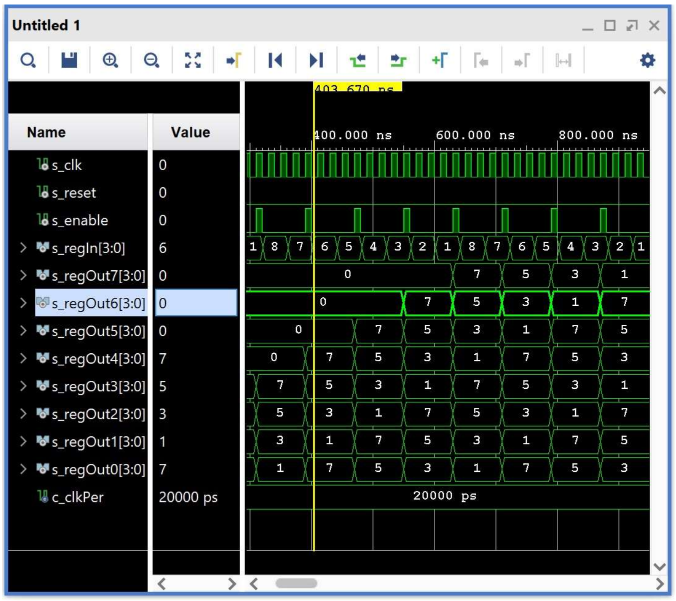

Fig 4. – Shift Register Testbench Waveform

The shift register testbench was likewise pretty simple – observe how the values shifted through the register (note the slower pulse skipped every other input value, as the regIn values were cycled through at a faster pace than the pulse was).

A top level testbench was also created to verify top level functionality (and also to debug why it was not working – it was observed that the constant values were not holding the numbers they needed to, so this was fixed).

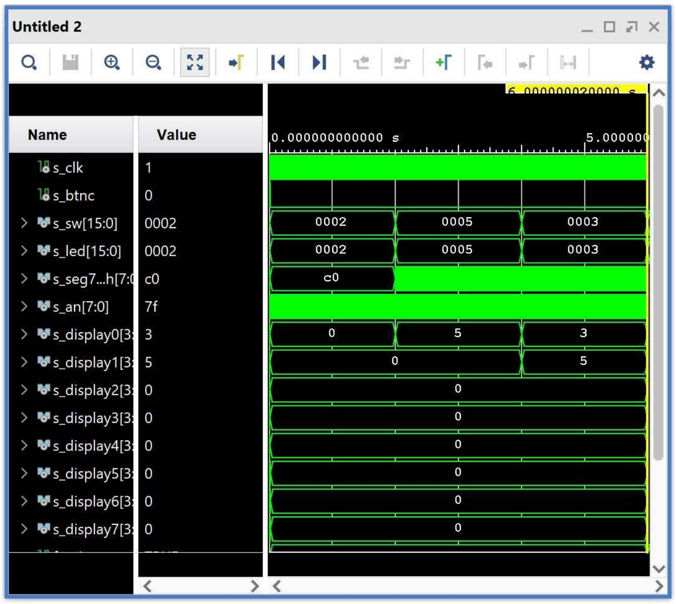

Fig. 5 – Top Level Testbench Waveform

The above waveform was only simulated for two real seconds due to the length of time simulation was taking – it was observed that the proper values were being shifted along the display as required, so the simulation was stopped in favor of hardware testing.

Finally, the whole program was synthesized and tested on the board, where (with the proper constants) the display shifted the values as specified at an easily visible pace, and the reset functionality would re-initialize it at all zeroes but immediately begin shifting the values again according to the input switches.

IV. Analysis/Conclusion

The utilization for this lab was estimated as follows: one clock buffer (for one clock source and no timing constraints set), 50 FPGA IOs (one 16-bit input, one 16-bit output, two 8-bit outputs, and two 1-bit inputs), no registers as latches (because every if statement was controlled by a clock or enable, and is synchronous) and 113 slice registers (properly clocked/strobed registers – 8 4-bit signals going from shiftReg to seg7_controller is 32; the 7-segment controller has synchronous processes with 3, 8, and 4 bits used, respectively, which is 15; and each pulse generator holds a 32 bit count value (all integers are synthesized as 32 bits in Xilinx) as well as one bit register for the clear, making 33 registers each; adding 32, 15, 33, and 33 makes 113. All of these estimates were correct and consistent with synthesis – unsigned values could have been used instead of integers to reduce register utilization for the count values, and this will be reworked in future labs for efficiency.

Resources	Estimate Used	Actual Used
Slice Registers	113	113
Register Latches	0	0
Clock Buffers	1	1
Number of IOs	50	50

Fig. 5 – Resource Utilization

In addition, the FPGA also functioned consistently with the specification once the proper count values were loaded into the pulse generators, with the count cycling across the display and resetting with a press of BTNC. This exercise in module instantiation and reuse will be built upon in later labs, so successful functionality was vital and appreciated.

Appendix

The following sections include the source code for the lab as well as the synthesis and utilization reports. These files are also included in the lab submission as separate files for easier viewing. The seg7hex.vhd file is not displayed here since it is trivial to this report, but it is nevertheless included in the submission folder.

A. *Lab 3 Top Module*

```
  -- lab3_top.vhd, written by Josh Rothe 5 Feb 2020
  -- this module instantiates the 7 segment controller, shift
reg,
  -- and pulse generator/clk divider to shift LCD values
continuously
  -- across a display

  library ieee;

  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;
  use work.all;

  entity lab3_top is port (
   clk    : in std_logic;
   btnc     : in std_logic;               -- reset
   sw     : in std_logic_vector(15 downto 0);    --  switches
(16 switches)
   led    : out std_logic_vector(15 downto 0);  -- LEDs  (16
LEDs)
   seg7_cath : out std_logic_vector(7 downto 0); --      seg7
display signals
   an       : out std_logic_vector(7 downto 0)
   );
  end lab3_top;

  architecture behavior of lab3_top is

  -- signal instantiation
  signal s_pulse      : std_logic;
  signal s_displayChar7  : std_logic_vector(3 downto 0);
  signal s_displayChar6  : std_logic_vector(3 downto 0);
  signal s_displayChar5  : std_logic_vector(3 downto 0);
  signal s_displayChar4  : std_logic_vector(3 downto 0);
  signal s_displayChar3  : std_logic_vector(3 downto 0);
  signal s_displayChar2  : std_logic_vector(3 downto 0);
  signal s_displayChar1  : std_logic_vector(3 downto 0);
  signal s_displayChar0  : std_logic_vector(3 downto 0);

  -- constant definition
  constant c_oneHz    : integer := 100000000;  -- to convert
from 100Mhz to 1Hz
  constant c_onekHz   : integer := 100000;      -- to convert
from 100Mhz to 1kHz

  -- alias definition
  alias a_swDisplayVal is sw(3 downto 0);     -- input value
from switches

  begin

  -- instantiate components, top lvl signals on right
   pulseGen_1Hz_inst1 : entity pulseGenerator
   generic map(maxCount  => c_oneHz)
   port map (  clk       => clk,
          reset     => btnc,
          pulseOut  => s_pulse
   );

   --  shift  register  handles  the  shifting/propagation  of
display values
   shiftReg_inst1   : entity shiftReg
   port map (  clk       => clk,
          reset     => btnc,
          enable    => s_pulse,
          regOut7   => s_displayChar7,
          regOut6   => s_displayChar6,
          regOut5   => s_displayChar5,
          regOut4   => s_displayChar4,
          regOut3   => s_displayChar3,
          regOut2   => s_displayChar2,
          regOut1   => s_displayChar1,
          regOut0   => s_displayChar0,
          regIn     => a_swDisplayVal
   );

   --  controller  reads  whatever  value  the  shift  register
presents it
   seg7_controller_inst1 : entity seg7_controller
   generic map ( pulseDiv  => c_onekHz)  --    passes    1kHz
constant in
   port map  ( clk       => clk,
             reset     => btnc,
             en      => seg7_cath,
             sigIn7    => s_displayChar7,
             sigIn6    => s_displayChar6,
             sigIn5    => s_displayChar5,
             sigIn4    => s_displayChar4,
             sigIn3    => s_displayChar3,
             sigIn2    => s_displayChar2,
             sigIn1    => s_displayChar1,
             sigIn0    => s_displayChar0,
             an      => an
   );

  -- LEDs indicate switch toggle, all switches enabled
  led <= sw;

  end behavior;
```

B. *7-Segment Controller Module*

```
-- seg7_controller.vhd, written by Josh Rothe 5 Feb 2020
-- This defines a 7 segment display controller that utilizes
-- refresh rates to display 8 diff values across a display

library ieee;

use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.all;
```

```
entity seg7_controller is
  generic ( pulseDiv  : integer := 10);  -- default value of
10, configurable
  port (    clk     : in std_logic;
         reset     : in std_logic;
         sigIn7    : in std_logic_vector(3 downto 0);
         sigIn6    : in std_logic_vector(3 downto 0);
         sigIn5    : in std_logic_vector(3 downto 0);
         sigIn4    : in std_logic_vector(3 downto 0);
         sigIn3    : in std_logic_vector(3 downto 0);
         sigIn2    : in std_logic_vector(3 downto 0);
         sigIn1    : in std_logic_vector(3 downto 0);
         sigIn0    : in std_logic_vector(3 downto 0);
         en     : out std_logic_vector(7 downto 0); -- seg7
display signals
         an        : out std_logic_vector(7 downto 0)
  );
end seg7_controller;

architecture behavior of seg7_controller is

-- signal instantiation
signal s_pulse   : std_logic;--  enable  synced  with  refresh
rate of display
signal s_digit   : std_logic_vector(3 downto 0);
signal s_sel_an    : std_logic_vector(2 downto 0);--
selects anode and cathode values

begin

-- instantiate components, top lvl signals on right
  pulseGen_1kHz_inst1: entity pulseGenerator
  generic map(maxCount  => pulseDiv)
  port map (  clk       => clk,
         reset     => reset,
         pulseOut  => s_pulse
  );

  seg7hex_inst1  : entity seg7hex
  port map (  digit   => s_digit,
         seg7 => en
  );

------- calculates which s_sel_an value to use based on pulse
count -------
proc_calc_anode  : process(clk,reset)
    variable anodeCount : std_logic_vector(2 downto 0);     -
- counts through the 8 values
begin
  if (reset='1') then
    anodeCount  := (others => '0');-- asynchronous reset of
count
  elsif (rising_edge(clk)) then
    if (s_pulse='1') then      -- only triggers on a pulse
      if (anodeCount = "111") then
        anodeCount  := "000";      -- cycle from 7 back to
0
      else
        anodeCount           :=           std_logic_vector(
unsigned(anodeCount) + 1 );  -- otherwise increment
      end if;
    end if;
  end if;
  s_sel_an <= anodeCount;
end process proc_calc_anode;
-----------------------------------------------------------
----------------

-- reads s_sel_an and outputs a decoded value to anodes --
-- also selects display value to send to 7-seg encoder ---
proc_set_an : process(clk)
begin
    if (rising_edge(clk)) then
        case s_sel_an is
            when "000" =>
                an <= "11111110";
                s_digit <= sigIn0;
            when "001" =>
                an <= "11111101";
                s_digit <= sigIn1;
            when "010" =>
                an <= "11111011";
                s_digit <= sigIn2;
            when "011" =>
                an <= "11110111";
                s_digit <= sigIn3;
            when "100" =>
                an <= "11101111";
                s_digit <= sigIn4;
            when "101" =>
                an <= "11011111";
                s_digit <= sigIn5;
            when "110" =>
                an <= "10111111";
                s_digit <= sigIn6;
            when others =>
                an <= "01111111";
                s_digit <= sigIn7;
        end case;
    end if;
end process proc_set_an;
----------------------------------------------------------

end behavior;
```

C. *Shift Register Module*

```
-- shiftReg.vhd, written by Josh Rothe 5 Feb 2020
-- This shifts values across outputs to be read
-- by a 7-segment controller

library ieee;

use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.all;

entity shiftReg is
  port (    clk     : in std_logic;
         reset     : in std_logic;
         enable    : in std_logic;
         regIn     : in std_logic_vector(3 downto 0);
         regOut7   : out std_logic_vector(3 downto 0);
         regOut6   : out std_logic_vector(3 downto 0);
         regOut5   : out std_logic_vector(3 downto 0);
         regOut4   : out std_logic_vector(3 downto 0);
         regOut3   : out std_logic_vector(3 downto 0);
         regOut2   : out std_logic_vector(3 downto 0);
         regOut1   : out std_logic_vector(3 downto 0);
         regOut0   : out std_logic_vector(3 downto 0)
  );
end shiftReg;

architecture behavior of shiftReg is

-- signal instantiation for output regs
signal s_reg7  : std_logic_vector(3 downto 0);
signal s_reg6  : std_logic_vector(3 downto 0);
signal s_reg5  : std_logic_vector(3 downto 0);
signal s_reg4  : std_logic_vector(3 downto 0);
signal s_reg3  : std_logic_vector(3 downto 0);
signal s_reg2  : std_logic_vector(3 downto 0);
signal s_reg1  : std_logic_vector(3 downto 0);
signal s_reg0  : std_logic_vector(3 downto 0);

begin

------- shifts each data input in based on enable pulse ----
---
proc_shift_reg : process(clk,reset)
begin
  if (reset='1') then     -- asynchronous reset of outputs
    s_reg7  <= (others => '0');
    s_reg6  <= (others => '0');
    s_reg5  <= (others => '0');
    s_reg4  <= (others => '0');
    s_reg3  <= (others => '0');
    s_reg2  <= (others => '0');
    s_reg1  <= (others => '0');
    s_reg0  <= (others => '0');
  elsif (rising_edge(clk)) then
    if (enable='1') then  -- enable input from pulse
      s_reg7 <= s_reg6;       --  when  enabled,  shift  all
values
      s_reg6 <= s_reg5;       -- with input entering least
sig reg
      s_reg5 <= s_reg4;
      s_reg4 <= s_reg3;
      s_reg3 <= s_reg2;
      s_reg2 <= s_reg1;
      s_reg1 <= s_reg0;
      s_reg0 <= regIn;
    end if;
  end if;
end process proc_shift_reg;
-----------------------------------------------------------
----

regOut7 <= s_reg7; -- continuously assign signals to outputs
regOut6 <= s_reg6;
regOut5 <= s_reg5;
regOut4 <= s_reg4;
regOut3 <= s_reg3;
regOut2 <= s_reg2;
regOut1 <= s_reg1;
regOut0 <= s_reg0;

end behavior;
```

D. Pulse Generator Module

```
-- pulseGenerator.vhd, written by Josh Rothe 5 Feb 2020
-- derived from Johns Hopkins EN.525.642.82.SP20, Module 3F
lecture
-- Pulse counter acts as a clk divider for a configurable
number of cycles

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.all;

entity pulseGenerator is
  generic ( maxCount: integer);--  default  value  of  10,
configurable
  port (    clk    : in std_logic;
          reset  : in std_logic;
          pulseOut: out std_logic);
end pulseGenerator;

architecture behavioral of pulseGenerator is
  signal pulseCnt  : integer;
  signal clear: std_logic;
begin
  -- Pulse Generator logic
  process(clk,reset)
  begin
    if (reset='1') then
      pulseCnt <= 0; -- reset signal to 0s
    elsif (rising_edge(clk)) then
      if (clear='1') then       -- when pulse goes high,
        pulseCnt <= 0; -- reset to 0 after one cycle
        clear <= '0';
      else                -- otherwise increment the count
        pulseCnt <= pulseCnt + 1;
        if (PulseCnt = maxCount) then
            clear <= '1';
        end if;
      end if;
    end if;
  end process;
  -- clear and pulseOut only go high at peak of count
  pulseOut <= clear;

end behavioral;
```

E. Testbench – Shift Register

```
-- tb_seg7_controller.vhd, written by Josh Rothe 6 Feb 2020
-- This testbench verifies the sim functionality of the
-- 7-segment controller written for lab 3. This tb simply
--  verifies  the  anodes  work,  and  the  values  are  read
appropriately

library ieee;

use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.all;

entity tb_shiftReg is
end tb_shiftReg;

architecture behavior of tb_shiftReg is

-- define constants
constant c_clkPer  : time := 20 ns;   -- 100 MHz clk

component shiftReg is
  port (    clk     : in std_logic;
          reset     : in std_logic;
          enable    : in std_logic;
          regIn     : in std_logic_vector(3 downto 0);
          regOut7   : out std_logic_vector(3 downto 0);
          regOut6   : out std_logic_vector(3 downto 0);
          regOut5   : out std_logic_vector(3 downto 0);
          regOut4   : out std_logic_vector(3 downto 0);
          regOut3   : out std_logic_vector(3 downto 0);
          regOut2   : out std_logic_vector(3 downto 0);
          regOut1   : out std_logic_vector(3 downto 0);
          regOut0   : out std_logic_vector(3 downto 0)
  );
end component;

-- signal instantiation
signal s_clk  : std_logic;
signal s_reset     : std_logic;
signal s_enable  : std_logic;
signal s_regIn   : std_logic_vector(3 downto 0);
signal s_regOut7 : std_logic_vector(3 downto 0);
signal s_regOut6 : std_logic_vector(3 downto 0);
signal s_regOut5 : std_logic_vector(3 downto 0);
signal s_regOut4 : std_logic_vector(3 downto 0);
signal s_regOut3 : std_logic_vector(3 downto 0);
signal s_regOut2 : std_logic_vector(3 downto 0);
signal s_regOut1 : std_logic_vector(3 downto 0);
signal s_regOut0 : std_logic_vector(3 downto 0);

begin
-- instantiate the unit under test, top lvl signals on right
uut : shiftReg
port map (  clk    => s_clk,
       reset  => s_reset,
       enable => s_enable,
       regIn  => s_regIn,
       regOut7=> s_regOut7,
       regOut6=> s_regOut6,
       regOut5=> s_regOut5,
       regOut4=> s_regOut4,
       regOut3=> s_regOut3,
       regOut2=> s_regOut2,
       regOut1=> s_regOut1,
       regOut0=> s_regOut0
     );

-- clock process, repeats indefinitely
proc_clock  : process
begin
  s_clk <= '0';
  wait for c_clkPer/2;
  s_clk <= '1';
  wait for c_clkPer/2;
end process;

-- reset high at start (initialize)
proc_reset  : process
begin
    s_reset <= '1';
    wait for c_clkPer;
    s_reset <= '0';
    wait;
end process;

-- enable pulse - set to 40 ns for faster sim
proc_pulse  : process
begin
  s_enable <= '0';
  wait for 70 ns;
  s_enable <= '1';
  wait for 10 ns;
end process;

-- signal input values test
proc_sig  : process
begin
  s_regIn <= "1000";
  wait for 41 ns;
  s_regIn <= "0111";
  wait for 40 ns;
  s_regIn <= "0110";
  wait for 40 ns;
  s_regIn <= "0101";
  wait for 40 ns;
  s_regIn <= "0100";
  wait for 40 ns;
  s_regIn <= "0011";
  wait for 40 ns;
  s_regIn <= "0010";
  wait for 40 ns;
  s_regIn <= "0001";
  wait for 40 ns;
end process;

end behavior;
```

F. Testbench – 7-Segment Controller

```
-- tb_seg7_controller.vhd, written by Josh Rothe 6 Feb 2020
-- This testbench verifies the sim functionality of the
-- 7-segment controller written for lab 3. This tb simply
--  verifies  the  anodes  work,  and  the  values  are  read
appropriately

library ieee;

use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.all;

entity tb_seg7_controller is
end tb_seg7_controller;

architecture behavior of tb_seg7_controller is

-- define constants
constant c_pulseDiv : unsigned := "100000";-- set for 1kHz
constant c_clkPer  : time := 20 ns;   -- 100 MHz clk

component seg7_controller is
  generic ( pulseDiv  : unsigned := "10");  --   pull   from
above, overrides default
  port (    clk    : in std_logic;
         reset     : in std_logic;
         sigIn7    : in std_logic_vector(3 downto 0);
         sigIn6    : in std_logic_vector(3 downto 0);
         sigIn5    : in std_logic_vector(3 downto 0);
         sigIn4    : in std_logic_vector(3 downto 0);
         sigIn3    : in std_logic_vector(3 downto 0);
         sigIn2    : in std_logic_vector(3 downto 0);
         sigIn1    : in std_logic_vector(3 downto 0);
         sigIn0    : in std_logic_vector(3 downto 0);
         en     : out std_logic_vector(7 downto 0); -- seg7
display signals
         an       : out std_logic_vector(7 downto 0)     --
annodes for activation
  );
end component;

-- signal instantiation
signal s_clk: std_logic;
signal s_reset   : std_logic;
signal s_sigIn7  : std_logic_vector(3 downto 0);
signal s_sigIn6  : std_logic_vector(3 downto 0);
signal s_sigIn5  : std_logic_vector(3 downto 0);
signal s_sigIn4  : std_logic_vector(3 downto 0);
signal s_sigIn3  : std_logic_vector(3 downto 0);
signal s_sigIn2  : std_logic_vector(3 downto 0);
signal s_sigIn1  : std_logic_vector(3 downto 0);
signal s_sigIn0  : std_logic_vector(3 downto 0);
signal s_en   : std_logic_vector(7 downto 0);
signal s_an : std_logic_vector(7 downto 0);

begin
-- instantiate the unit under test, top lvl signals on right
uut : seg7_controller
generic map(pulseDiv  => c_pulseDiv)
port map (  clk      => s_clk,
          reset       => s_reset,
      sigIn7    => s_sigIn7,
      sigIn6    => s_sigIn6,
      sigIn5    => s_sigIn5,
      sigIn4    => s_sigIn4,
      sigIn3    => s_sigIn3,
      sigIn2    => s_sigIn2,
      sigIn1    => s_sigIn1,
      sigIn0    => s_sigIn0,
      en     => s_en,
      an     => s_an
    );

-- clock process, repeats indefinitely
proc_clock  : process
begin
  s_clk <= '0';
  wait for c_clkPer/2;
  s_clk <= '1';
  wait for c_clkPer/2;
end process;

-- reset high at start (initialize)
proc_reset  : process
begin
    s_reset <= '1';
    wait for c_clkPer;
    s_reset <= '0';
    wait; -- does not repeat
end process;

-- signal input values test
proc_sig  : process
begin
  s_sigIn7 <= "1000";
  s_sigIn6 <= "0111";
  s_sigIn5 <= "0110";
  s_sigIn4 <= "0101";
  s_sigIn3 <= "0100";
  s_sigIn2 <= "0011";
  s_sigIn1 <= "0010";
  s_sigIn0 <= "0001";
  wait;
end process;

end behavior;
```

G. Testbench – Lab 3 Top Module

```
-- tb_lab3_top.vhd, written by Josh Rothe 10 Feb 2020
-- Revised 17 Feb 2020 to incorporate decoder
-- This testbench verifies the sim functionality of the
-- entire lab 3 design

library ieee;

use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.all;

entity tb_lab3_top is
end tb_lab3_top;

architecture behavior of tb_lab3_top is

-- define constants
constant c_clkPer  : time := 20 ns;   -- 100 MHz clk = 20 ns

component lab3_top is
  port ( clk    : in std_logic;
      btnc     : in std_logic;    -- reset
      sw     : in std_logic_vector(15 downto 0);
      led    : out std_logic_vector(15 downto 0);
      seg7_cath : out std_logic_vector(7 downto 0);
      an       : out std_logic_vector(7 downto 0)
  );
end component;

-- decoder for 7 segment display (sim only)
component seg7_interface_sim is
  port ( cathodes  : in std_logic_vector (7 downto 0);
      anodes    : in std_logic_vector (7 downto 0);
      display0  : out std_logic_vector (3 downto 0);
      display1  : out std_logic_vector (3 downto 0);
      display2  : out std_logic_vector (3 downto 0);
      display3  : out std_logic_vector (3 downto 0);
      display4  : out std_logic_vector (3 downto 0);
      display5  : out std_logic_vector (3 downto 0);
      display6  : out std_logic_vector (3 downto 0);
      display7  : out std_logic_vector (3 downto 0));
end component;

-- signal instantiation
signal s_clk   : std_logic;
signal s_btnc    : std_logic;
signal s_sw    : std_logic_vector(15 downto 0);
signal s_led   : std_logic_vector(15 downto 0);
signal s_seg7_cath : std_logic_vector(7 downto 0);
signal s_an      : std_logic_vector(7 downto 0);
signal s_display0  : std_logic_vector(3 downto 0);
signal s_display1  : std_logic_vector(3 downto 0);
signal s_display2  : std_logic_vector(3 downto 0);
signal s_display3  : std_logic_vector(3 downto 0);
signal s_display4  : std_logic_vector(3 downto 0);
signal s_display5  : std_logic_vector(3 downto 0);
signal s_display6  : std_logic_vector(3 downto 0);
signal s_display7  : std_logic_vector(3 downto 0);

-- exit and initialization flags
signal f_exit    : boolean := false;
signal f_initialize: boolean := false;

begin
-- instantiate the unit under test, top lvl signals on right
uut : lab3_top
port map (  clk       => s_clk,
       btnc    => s_btnc,
       sw      => s_sw,
       led       => s_led,
       seg7_cath => s_seg7_cath,
       an      => s_an);

-- instantiate decoder
decode_uut : seg7_interface_sim
port map (  cathodes  => s_seg7_cath,
       anodes    => s_an,
       display0  => s_display0,
       display1  => s_display1,
       display2  => s_display2,
       display3  => s_display3,
       display4  => s_display4,
       display5  => s_display5,
       display6  => s_display6,
       display7  => s_display7);

-- clock process, repeats until exit flag
proc_clock  : process
begin
    while f_exit = false loop
        s_clk <= '0';
        wait for c_clkPer/2;
        s_clk <= '1';
        wait for c_clkPer/2;
  end loop;
end process;

-- reset high at start (initialize)
proc_reset  : process
begin
    s_btnc <= '1';
    wait for c_clkPer;
    s_btnc <= '0';
  wait for c_clkPer;
  f_initialize <= true;
    wait;
end process;

-- signal input values test - stimulus process
proc_sw: process
begin
  s_sw <= "0000000000000010";
  wait for c_clkPer*100000000;
  s_sw <= "0000000000000101";
  wait for c_clkPer*100000000;
  s_sw <= "0000000000000011";
  wait for c_clkPer*100000000;
  f_exit <= true;       -- exit flag triggered at end
end process;

end behavior;
```

H. 7-Segment Display Simulation Decoder Module

```
-- seg7_interface_sim.vhd, written by Josh Rothe 17 Feb 2020
-- This module decodes and checks the values going to the 7
segment display
-- derived from sample code given in module 4E, Johns Hopkins
EN.525.642.82.SP20

library ieee;

use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.all;

entity seg7_interface_sim is
  Port ( cathodes  : in std_logic_vector (7 downto 0);
       anodes    : in std_logic_vector (7 downto 0);
       display0  : out std_logic_vector (3 downto 0);
       display1  : out std_logic_vector (3 downto 0);
       display2  : out std_logic_vector (3 downto 0);
       display3  : out std_logic_vector (3 downto 0);
       display4  : out std_logic_vector (3 downto 0);
       display5  : out std_logic_vector (3 downto 0);
       display6  : out std_logic_vector (3 downto 0);
       display7  : out std_logic_vector (3 downto 0));
end seg7_interface_sim;

architecture behavior of seg7_interface_sim is

signal digit_decode : std_logic_vector(3 downto 0);
signal char0        : std_logic_vector(3 downto 0);
signal char1        : std_logic_vector(3 downto 0);
signal char2        : std_logic_vector(3 downto 0);
signal char3        : std_logic_vector(3 downto 0);
signal char4        : std_logic_vector(3 downto 0);
signal char5        : std_logic_vector(3 downto 0);
signal char6        : std_logic_vector(3 downto 0);
signal char7        : std_logic_vector(3 downto 0);

begin

--decoder
with cathodes select
digit_decode <= X"0" when "11000000",
          X"1" when "11111001",
          X"2" when "10100100",
          X"3" when "10110000",
          X"4" when "10011001",
          X"5" when "10010010",
          X"6" when "10000010",
          X"7" when "11111000",
          X"8" when "10000000",
          X"9" when "10010000",
          X"A" when "10001000",
          X"B" when "10000011",
          X"C" when "11000110",
          X"D" when "10100001",
          X"E" when "10000110",
          X"F" when others;

--Capture decoded character for each anode low signal
(LATCHES! DO NOT MAKE THESE FOR HARDWARE DESIGNS!)
char0 <= digit_decode when anodes(0) = '0' else char0;
char1 <= digit_decode when anodes(1) = '0' else char1;
char2 <= digit_decode when anodes(2) = '0' else char2;
char3 <= digit_decode when anodes(3) = '0' else char3;
char4 <= digit_decode when anodes(4) = '0' else char4;
char5 <= digit_decode when anodes(5) = '0' else char5;
char6 <= digit_decode when anodes(6) = '0' else char6;
char7 <= digit_decode when anodes(7) = '0' else char7;

display0 <= char0;
display1 <= char1;
display2 <= char2;
display3 <= char3;
display4 <= char4;
display5 <= char5;
display6 <= char6;
display7 <= char7;

end behavior;
```

I. Utilization Report

```
Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.
----------------------------------------------------------
------------------------------------------------
| Tool Version : Vivado v.2019.2 (win64) Build 2708876 Wed
Nov  6 21:40:23 MST 2019
| Date         : Mon Feb 10 22:47:03 2020
| Host         : DESKTOP-OJ146FQ running 64-bit major release
(build 9200)
|   Command              :   report_utilization   -file
lab3_top_utilization_synth.rpt                         -pb
lab3_top_utilization_synth.pb
| Design       : lab3_top
| Device       : 7a100tcsg324-1
| Design State : Synthesized
----------------------------------------------------------
------------------------------------------------

Utilization Design Information

Table of Contents
-----------------
1. Slice Logic
1.1 Summary of Registers by Type
2. Memory
3. DSP
```

```
4. IO and GT Specific
5. Clocking
6. Specific Feature
7. Primitives
8. Black Boxes
9. Instantiated Netlists

1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-----
--+
|        Site Type        | Used | Fixed | Available | Util%
|
+-------------------------+------+-------+-----------+-----
--+
| Slice LUTs*             |   97 |     0 |     63400 |  0.15
|
|   LUT as Logic          |   97 |     0 |     63400 |  0.15
|
|   LUT as Memory         |    0 |     0 |     19000 |  0.00
|
| Slice Registers         |  113 |     0 |    126800 |  0.09
|
|   Register as Flip Flop |  113 |     0 |    126800 |  0.09
|
|   Register as Latch     |    0 |     0 |    126800 |  0.00
|
| F7 Muxes                |    4 |     0 |     31700 |  0.01
|
| F8 Muxes                |    0 |     0 |     15850 |  0.00
|
+-------------------------+------+-------+-----------+-----
--+
* Warning! The Final LUT count, after physical optimizations
and full implementation, is typically lower. Run opt_design
after synthesis, if not already completed, for a more
realistic count.


1.1 Summary of Registers by Type
--------------------------------

+-------+--------------+-------------+--------------+
| Total | Clock Enable | Synchronous | Asynchronous |
+-------+--------------+-------------+--------------+
| 0     |            _ |           - |            - |
| 0     |            _ |           - |          Set |
| 0     |            _ |           - |        Reset |
| 0     |            _ |         Set |            - |
| 0     |            _ |       Reset |            - |
| 0     |          Yes |           - |            - |
| 0     |          Yes |           - |          Set |
| 99    |          Yes |           - |        Reset |
| 8     |          Yes |         Set |            - |
| 6     |          Yes |       Reset |            - |
+-------+--------------+-------------+--------------+


2. Memory
---------

+----------------+------+-------+-----------+-------+
|    Site Type   | Used | Fixed | Available | Util% |
+----------------+------+-------+-----------+-------+
| Block RAM Tile |    0 |     0 |       135 |  0.00 |
|   RAMB36/FIFO* |    0 |     0 |       135 |  0.00 |
|   RAMB18       |    0 |     0 |       270 |  0.00 |
+----------------+------+-------+-----------+-------+
* Note: Each Block RAM Tile only has one FIFO logic available
and therefore can accommodate only one FIFO36E1 or one
FIFO18E1. However, if a FIFO18E1 occupies a Block RAM Tile,
that tile can still accommodate a RAMB18E1


3. DSP
------

+-----------+------+-------+-----------+-------+
| Site Type | Used | Fixed | Available | Util% |
+-----------+------+-------+-----------+-------+
| DSPs      |    0 |     0 |       240 |  0.00 |
+-----------+------+-------+-----------+-------+


4. IO and GT Specific
---------------------

+-----------------------------+------+-------+-----------+-
------+
|          Site Type          | Used | Fixed | Available |
Util% |
+-----------------------------+------+-------+-----------+-
------+
| Bonded IOB                  |   50 |     0 |       210 |
23.81 |
| Bonded IPADs                |    0 |     0 |         2 |
0.00 |
| PHY_CONTROL                 |    0 |     0 |         6 |
0.00 |
| PHASER_REF                  |    0 |     0 |         6 |
0.00 |
| OUT_FIFO                    |    0 |     0 |        24 |
0.00 |
| IN_FIFO                     |    0 |     0 |        24 |
0.00 |
| IDELAYCTRL                  |    0 |     0 |         6 |
0.00 |
| IBUFDS                      |    0 |     0 |       202 |
0.00 |
| PHASER_OUT/PHASER_OUT_PHY   |    0 |     0 |        24 |
0.00 |
| PHASER_IN/PHASER_IN_PHY     |    0 |     0 |        24 |
0.00 |
| IDELAYE2/IDELAYE2_FINEDELAY |    0 |     0 |       300 |
0.00 |
| ILOGIC                      |    0 |     0 |       210 |
0.00 |
| OLOGIC                      |    0 |     0 |       210 |
0.00 |
+-----------------------------+------+-------+-----------+-
------+


5. Clocking
-----------

+------------+------+-------+-----------+-------+
|  Site Type | Used | Fixed | Available | Util% |
+------------+------+-------+-----------+-------+
| BUFGCTRL   |    1 |     0 |        32 |  3.13 |
| BUFIO      |    0 |     0 |        24 |  0.00 |
| MMCME2_ADV |    0 |     0 |         6 |  0.00 |
| PLLE2_ADV  |    0 |     0 |         6 |  0.00 |
| BUFMRCE    |    0 |     0 |        12 |  0.00 |
| BUFHCE     |    0 |     0 |        96 |  0.00 |
| BUFR       |    0 |     0 |        24 |  0.00 |
+------------+------+-------+-----------+-------+


6. Specific Feature
-------------------

+-------------+------+-------+-----------+-------+
|  Site Type  | Used | Fixed | Available | Util% |
+-------------+------+-------+-----------+-------+
| BSCANE2     |    0 |     0 |         4 |  0.00 |
| CAPTUREE2   |    0 |     0 |         1 |  0.00 |
| DNA_PORT    |    0 |     0 |         1 |  0.00 |
| EFUSE_USR   |    0 |     0 |         1 |  0.00 |
| FRAME_ECCE2 |    0 |     0 |         1 |  0.00 |
| ICAPE2      |    0 |     0 |         2 |  0.00 |
| PCIE_2_1    |    0 |     0 |         1 |  0.00 |
| STARTUPE2   |    0 |     0 |         1 |  0.00 |
| XADC        |    0 |     0 |         1 |  0.00 |
+-------------+------+-------+-----------+-------+


7. Primitives
-------------

+----------+------+---------------------+
| Ref Name | Used | Functional Category |
+----------+------+---------------------+
| FDCE     |   99 |        Flop & Latch |
| LUT2     |   72 |                 LUT |
| OBUF     |   32 |                  IO |
| LUT6     |   20 |                 LUT |
| IBUF     |   18 |                  IO |
| CARRY4   |   16 |          CarryLogic |
| LUT4     |    8 |                 LUT |
| FDSE     |    8 |        Flop & Latch |
| FDRE     |    6 |        Flop & Latch |
| MUXF7    |    4 |               MuxFx |
| LUT1     |    3 |                 LUT |
| LUT5     |    2 |                 LUT |
| LUT3     |    1 |                 LUT |
| BUFG     |    1 |               Clock |
+----------+------+---------------------+


8. Black Boxes
--------------

+----------+------+
| Ref Name | Used |
+----------+------+


9. Instantiated Netlists
------------------------

+----------+------+
| Ref Name | Used |
+----------+------+
```

J. Constraints File

```
  ## This file is a general .xdc for the Nexys4 DDR Rev. C
  ## To use it in a project:
  ## - uncomment the lines corresponding to used pins
  ## - rename the used ports (in each line, after get_ports)
according to the top level signal names in the project

  ## Clock signal
  set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33
} [get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
  #create_clock  -add  -name  sys_clk_pin  -period  10.00  -
waveform {0 5} [get_ports {CLK100MHZ}];


  ##Switches

  set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33
} [get_ports { sw[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
  set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33
} [get_ports { sw[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
  set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33
} [get_ports { sw[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
  set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33
} [get_ports { sw[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
  set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33
} [get_ports { sw[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
  set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33
} [get_ports { sw[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
  set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33
} [get_ports { sw[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
  set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33
} [get_ports { sw[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
  set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18
} [get_ports { sw[8] }]; #IO_L24N_T3_34 Sch=sw[8]
  set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18
} [get_ports { sw[9] }]; #IO_25_34 Sch=sw[9]
  set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33
}   [get_ports   {   sw[10]   }];   #IO_L15P_T2_DQS_RDWR_B_14
Sch=sw[10]
  set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33
} [get_ports { sw[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
  set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33
} [get_ports { sw[12] }]; #IO_L24P_T3_35 Sch=sw[12]
  set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33
} [get_ports { sw[13] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
  set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33
}   [get_ports   {   sw[14]   }];   #IO_L19N_T3_A09_D25_VREF_14
Sch=sw[14]
  set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33
} [get_ports { sw[15] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]


  ## LEDs

  set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33
} [get_ports { led[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
  set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33
} [get_ports { led[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
  set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33
} [get_ports { led[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
  set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33
} [get_ports { led[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
  set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33
} [get_ports { led[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
  set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33
} [get_ports { led[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
  set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33
} [get_ports { led[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
  set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33
} [get_ports { led[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
  set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33
} [get_ports { led[8] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
  set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33
} [get_ports { led[9] }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
  set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33
} [get_ports { led[10] }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
  set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33
}  [get_ports  {  led[11]  }];  #IO_L15N_T2_DQS_DOUT_CSO_B_14
Sch=led[11]
  set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33
} [get_ports { led[12] }]; #IO_L16P_T2_CSI_B_14 Sch=led[12]
  set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33
} [get_ports { led[13] }]; #IO_L22N_T3_A04_D20_14 Sch=led[13]
  set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33
} [get_ports { led[14] }]; #IO_L20N_T3_A07_D23_14 Sch=led[14]
  set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33
}   [get_ports   {   led[15]   }];   #IO_L21N_T3_DQS_A06_D22_14
Sch=led[15]

  #set_property -dict { PACKAGE_PIN R12   IOSTANDARD LVCMOS33
} [get_ports { LED16_B }]; #IO_L5P_T0_D06_14 Sch=led16_b
  #set_property -dict { PACKAGE_PIN M16   IOSTANDARD LVCMOS33
} [get_ports { LED16_G }]; #IO_L10P_T1_D14_14 Sch=led16_g
  #set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33
} [get_ports { LED16_R }]; #IO_L11P_T1_SRCC_14 Sch=led16_r
  #set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33
}   [get_ports   {   LED17_B   }];   #IO_L15N_T2_DQS_ADV_B_15
Sch=led17_b
  #set_property -dict { PACKAGE_PIN R11   IOSTANDARD LVCMOS33
} [get_ports { LED17_G }]; #IO_0_14 Sch=led17_g
  #set_property -dict { PACKAGE_PIN N16   IOSTANDARD LVCMOS33
} [get_ports { LED17_R }]; #IO_L11N_T1_SRCC_14 Sch=led17_r


  ##7 segment display

  set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33
} [get_ports { seg7_cath[0] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
  set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33
} [get_ports { seg7_cath[1] }]; #IO_25_14 Sch=cb
  set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33
} [get_ports { seg7_cath[2] }]; #IO_25_15 Sch=cc
  set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33
} [get_ports { seg7_cath[3] }]; #IO_L17P_T2_A26_15 Sch=cd
  set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33
} [get_ports { seg7_cath[4] }]; #IO_L13P_T2_MRCC_14 Sch=ce
  set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33
} [get_ports { seg7_cath[5] }]; #IO_L19P_T3_A10_D26_14 Sch=cf
  set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33
} [get_ports { seg7_cath[6] }]; #IO_L4P_T0_D04_14 Sch=cg
  set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33
}  [get_ports  {  seg7_cath[7]  }];  #IO_L19N_T3_A21_VREF_15
Sch=dp

  set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33
} [get_ports { an[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
  set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33
} [get_ports { an[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
  set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33
} [get_ports { an[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
  set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33
} [get_ports { an[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
  set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33
} [get_ports { an[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
  set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33
} [get_ports { an[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
  set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33
} [get_ports { an[6] }]; #IO_L23P_T3_35 Sch=an[6]
  set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33
} [get_ports { an[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]


  ##Buttons

  #set_property -dict { PACKAGE_PIN C12   IOSTANDARD LVCMOS33
}   [get_ports   {   CPU_RESETN   }];   #IO_L3P_T0_DQS_AD1P_15
Sch=cpu_resetn

  set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33
} [get_ports { btnc }]; #IO_L9P_T1_DQS_14 Sch=btnc
```

```
  #set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33
} [get_ports { BTNU }]; #IO_L4N_T0_D05_14 Sch=btnu
  #set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33
} [get_ports { BTNL }]; #IO_L12P_T1_MRCC_14 Sch=btnl
  #set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33
} [get_ports { BTNR }]; #IO_L10N_T1_D15_14 Sch=btnr
  #set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33
} [get_ports { BTND }]; #IO_L9N_T1_DQS_D13_14 Sch=btnd


  ##Pmod Headers


  ##Pmod Header JA

  #set_property -dict { PACKAGE_PIN C17   IOSTANDARD LVCMOS33
} [get_ports { JA[1] }]; #IO_L20N_T3_A19_15 Sch=ja[1]
  #set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33
} [get_ports { JA[2] }]; #IO_L21N_T3_DQS_A18_15 Sch=ja[2]
  #set_property -dict { PACKAGE_PIN E18   IOSTANDARD LVCMOS33
} [get_ports { JA[3] }]; #IO_L21P_T3_DQS_15 Sch=ja[3]
  #set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33
} [get_ports { JA[4] }]; #IO_L18N_T2_A23_15 Sch=ja[4]
  #set_property -dict { PACKAGE_PIN D17   IOSTANDARD LVCMOS33
} [get_ports { JA[7] }]; #IO_L16N_T2_A27_15 Sch=ja[7]
  #set_property -dict { PACKAGE_PIN E17   IOSTANDARD LVCMOS33
} [get_ports { JA[8] }]; #IO_L16P_T2_A28_15 Sch=ja[8]
  #set_property -dict { PACKAGE_PIN F18   IOSTANDARD LVCMOS33
} [get_ports { JA[9] }]; #IO_L22N_T3_A16_15 Sch=ja[9]
  #set_property -dict { PACKAGE_PIN G18   IOSTANDARD LVCMOS33
} [get_ports { JA[10] }]; #IO_L22P_T3_A17_15 Sch=ja[10]


  ##Pmod Header JB

  #set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33
} [get_ports { JB[1] }]; #IO_L1P_T0_AD0P_15 Sch=jb[1]
  #set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33
} [get_ports { JB[2] }]; #IO_L14N_T2_SRCC_15 Sch=jb[2]
  #set_property -dict { PACKAGE_PIN G16   IOSTANDARD LVCMOS33
} [get_ports { JB[3] }]; #IO_L13N_T2_MRCC_15 Sch=jb[3]
  #set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33
} [get_ports { JB[4] }]; #IO_L15P_T2_DQS_15 Sch=jb[4]
  #set_property -dict { PACKAGE_PIN E16   IOSTANDARD LVCMOS33
} [get_ports { JB[7] }]; #IO_L11N_T1_SRCC_15 Sch=jb[7]
  #set_property -dict { PACKAGE_PIN F13   IOSTANDARD LVCMOS33
} [get_ports { JB[8] }]; #IO_L5P_T0_AD9P_15 Sch=jb[8]
  #set_property -dict { PACKAGE_PIN G13   IOSTANDARD LVCMOS33
} [get_ports { JB[9] }]; #IO_0_15 Sch=jb[9]
  #set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33
} [get_ports { JB[10] }]; #IO_L13P_T2_MRCC_15 Sch=jb[10]


  ##Pmod Header JC

  #set_property -dict { PACKAGE_PIN K1    IOSTANDARD LVCMOS33
} [get_ports { JC[1] }]; #IO_L23N_T3_35 Sch=jc[1]
  #set_property -dict { PACKAGE_PIN F6    IOSTANDARD LVCMOS33
} [get_ports { JC[2] }]; #IO_L19N_T3_VREF_35 Sch=jc[2]
  #set_property -dict { PACKAGE_PIN J2    IOSTANDARD LVCMOS33
} [get_ports { JC[3] }]; #IO_L22N_T3_35 Sch=jc[3]
  #set_property -dict { PACKAGE_PIN G6    IOSTANDARD LVCMOS33
} [get_ports { JC[4] }]; #IO_L19P_T3_35 Sch=jc[4]
  #set_property -dict { PACKAGE_PIN E7    IOSTANDARD LVCMOS33
} [get_ports { JC[7] }]; #IO_L6P_T0_35 Sch=jc[7]
  #set_property -dict { PACKAGE_PIN J3    IOSTANDARD LVCMOS33
} [get_ports { JC[8] }]; #IO_L22P_T3_35 Sch=jc[8]
  #set_property -dict { PACKAGE_PIN J4    IOSTANDARD LVCMOS33
} [get_ports { JC[9] }]; #IO_L21P_T3_DQS_35 Sch=jc[9]
  #set_property -dict { PACKAGE_PIN E6    IOSTANDARD LVCMOS33
} [get_ports { JC[10] }]; #IO_L5P_T0_AD13P_35 Sch=jc[10]


  ##Pmod Header JD

  #set_property -dict { PACKAGE_PIN H4    IOSTANDARD LVCMOS33
} [get_ports { JD[1] }]; #IO_L21N_T3_DQS_35 Sch=jd[1]
  #set_property -dict { PACKAGE_PIN H1    IOSTANDARD LVCMOS33
} [get_ports { JD[2] }]; #IO_L17P_T2_35 Sch=jd[2]
  #set_property -dict { PACKAGE_PIN G1    IOSTANDARD LVCMOS33
} [get_ports { JD[3] }]; #IO_L17N_T2_35 Sch=jd[3]
  #set_property -dict { PACKAGE_PIN G3    IOSTANDARD LVCMOS33
} [get_ports { JD[4] }]; #IO_L20N_T3_35 Sch=jd[4]
  #set_property -dict { PACKAGE_PIN H2    IOSTANDARD LVCMOS33
} [get_ports { JD[7] }]; #IO_L15P_T2_DQS_35 Sch=jd[7]
  #set_property -dict { PACKAGE_PIN G4    IOSTANDARD LVCMOS33
} [get_ports { JD[8] }]; #IO_L20P_T3_35 Sch=jd[8]
  #set_property -dict { PACKAGE_PIN G2    IOSTANDARD LVCMOS33
} [get_ports { JD[9] }]; #IO_L15N_T2_DQS_35 Sch=jd[9]
  #set_property -dict { PACKAGE_PIN F3    IOSTANDARD LVCMOS33
} [get_ports { JD[10] }]; #IO_L13N_T2_MRCC_35 Sch=jd[10]


  ##Pmod Header JXADC

  #set_property  -dict  {  PACKAGE_PIN  A14    IOSTANDARD  LVDS
} [get_ports { XA_N[1] }]; #IO_L9N_T1_DQS_AD3N_15 Sch=xa_n[1]
  #set_property  -dict  {  PACKAGE_PIN  A13    IOSTANDARD  LVDS
} [get_ports { XA_P[1] }]; #IO_L9P_T1_DQS_AD3P_15 Sch=xa_p[1]
  #set_property  -dict  {  PACKAGE_PIN  A16    IOSTANDARD  LVDS
} [get_ports { XA_N[2] }]; #IO_L8N_T1_AD10N_15 Sch=xa_n[2]
  #set_property  -dict  {  PACKAGE_PIN  A15    IOSTANDARD  LVDS
} [get_ports { XA_P[2] }]; #IO_L8P_T1_AD10P_15 Sch=xa_p[2]
  #set_property  -dict  {  PACKAGE_PIN  B17    IOSTANDARD  LVDS
} [get_ports { XA_N[3] }]; #IO_L7N_T1_AD2N_15 Sch=xa_n[3]
  #set_property  -dict  {  PACKAGE_PIN  B16    IOSTANDARD  LVDS
} [get_ports { XA_P[3] }]; #IO_L7P_T1_AD2P_15 Sch=xa_p[3]
  #set_property  -dict  {  PACKAGE_PIN  A18    IOSTANDARD  LVDS
} [get_ports { XA_N[4] }]; #IO_L10N_T1_AD11N_15 Sch=xa_n[4]
  #set_property  -dict  {  PACKAGE_PIN  B18    IOSTANDARD  LVDS
} [get_ports { XA_P[4] }]; #IO_L10P_T1_AD11P_15 Sch=xa_p[4]


  ##VGA Connector

  #set_property -dict { PACKAGE_PIN A3    IOSTANDARD LVCMOS33
} [get_ports { VGA_R[0] }]; #IO_L8N_T1_AD14N_35 Sch=vga_r[0]
  #set_property -dict { PACKAGE_PIN B4    IOSTANDARD LVCMOS33
} [get_ports { VGA_R[1] }]; #IO_L7N_T1_AD6N_35 Sch=vga_r[1]
  #set_property -dict { PACKAGE_PIN C5    IOSTANDARD LVCMOS33
} [get_ports { VGA_R[2] }]; #IO_L1N_T0_AD4N_35 Sch=vga_r[2]
  #set_property -dict { PACKAGE_PIN A4    IOSTANDARD LVCMOS33
} [get_ports { VGA_R[3] }]; #IO_L8P_T1_AD14P_35 Sch=vga_r[3]

  #set_property -dict { PACKAGE_PIN C6    IOSTANDARD LVCMOS33
} [get_ports { VGA_G[0] }]; #IO_L1P_T0_AD4P_35 Sch=vga_g[0]
  #set_property -dict { PACKAGE_PIN A5    IOSTANDARD LVCMOS33
}   [get_ports   {   VGA_G[1]   }];   #IO_L3N_T0_DQS_AD5N_35
Sch=vga_g[1]
  #set_property -dict { PACKAGE_PIN B6    IOSTANDARD LVCMOS33
} [get_ports { VGA_G[2] }]; #IO_L2N_T0_AD12N_35 Sch=vga_g[2]
  #set_property -dict { PACKAGE_PIN A6    IOSTANDARD LVCMOS33
}   [get_ports   {   VGA_G[3]   }];   #IO_L3P_T0_DQS_AD5P_35
Sch=vga_g[3]

  #set_property -dict { PACKAGE_PIN B7    IOSTANDARD LVCMOS33
} [get_ports { VGA_B[0] }]; #IO_L2P_T0_AD12P_35 Sch=vga_b[0]
  #set_property -dict { PACKAGE_PIN C7    IOSTANDARD LVCMOS33
} [get_ports { VGA_B[1] }]; #IO_L4N_T0_35 Sch=vga_b[1]
  #set_property -dict { PACKAGE_PIN D7    IOSTANDARD LVCMOS33
} [get_ports { VGA_B[2] }]; #IO_L6N_T0_VREF_35 Sch=vga_b[2]
  #set_property -dict { PACKAGE_PIN D8    IOSTANDARD LVCMOS33
} [get_ports { VGA_B[3] }]; #IO_L4P_T0_35 Sch=vga_b[3]

  #set_property -dict { PACKAGE_PIN B11   IOSTANDARD LVCMOS33
} [get_ports { VGA_HS }]; #IO_L4P_T0_15 Sch=vga_hs
  #set_property -dict { PACKAGE_PIN B12   IOSTANDARD LVCMOS33
} [get_ports { VGA_VS }]; #IO_L3N_T0_DQS_AD1N_15 Sch=vga_vs


  ##Micro SD Connector

  #set_property -dict { PACKAGE_PIN E2    IOSTANDARD LVCMOS33
} [get_ports { SD_RESET }]; #IO_L14P_T2_SRCC_35 Sch=sd_reset
  #set_property -dict { PACKAGE_PIN A1    IOSTANDARD LVCMOS33
} [get_ports { SD_CD }]; #IO_L9N_T1_DQS_AD7N_35 Sch=sd_cd
  #set_property -dict { PACKAGE_PIN B1    IOSTANDARD LVCMOS33
} [get_ports { SD_SCK }]; #IO_L9P_T1_DQS_AD7P_35 Sch=sd_sck
  #set_property -dict { PACKAGE_PIN C1    IOSTANDARD LVCMOS33
} [get_ports { SD_CMD }]; #IO_L16N_T2_35 Sch=sd_cmd
  #set_property -dict { PACKAGE_PIN C2    IOSTANDARD LVCMOS33
} [get_ports { SD_DAT[0] }]; #IO_L16P_T2_35 Sch=sd_dat[0]
  #set_property -dict { PACKAGE_PIN E1    IOSTANDARD LVCMOS33
} [get_ports { SD_DAT[1] }]; #IO_L18N_T2_35 Sch=sd_dat[1]
  #set_property -dict { PACKAGE_PIN F1    IOSTANDARD LVCMOS33
} [get_ports { SD_DAT[2] }]; #IO_L18P_T2_35 Sch=sd_dat[2]
  #set_property -dict { PACKAGE_PIN D2    IOSTANDARD LVCMOS33
}   [get_ports   {   SD_DAT[3]   }];   #IO_L14N_T2_SRCC_35
Sch=sd_dat[3]


  ##Accelerometer
  #set_property -dict { PACKAGE_PIN E15   IOSTANDARD LVCMOS33
} [get_ports { ACL_MISO }]; #IO_L11P_T1_SRCC_15 Sch=acl_miso
  #set_property -dict { PACKAGE_PIN F14   IOSTANDARD LVCMOS33
} [get_ports { ACL_MOSI }]; #IO_L5N_T0_AD9N_15 Sch=acl_mosi
  #set_property -dict { PACKAGE_PIN F15   IOSTANDARD LVCMOS33
} [get_ports { ACL_SCLK }]; #IO_L14P_T2_SRCC_15 Sch=acl_sclk
  #set_property -dict { PACKAGE_PIN D15   IOSTANDARD LVCMOS33
} [get_ports { ACL_CSN }]; #IO_L12P_T1_MRCC_15 Sch=acl_csn
  #set_property -dict { PACKAGE_PIN B13   IOSTANDARD LVCMOS33
}   [get_ports   {   ACL_INT[1]   }];   #IO_L2P_T0_AD8P_15
Sch=acl_int[1]
  #set_property -dict { PACKAGE_PIN C16   IOSTANDARD LVCMOS33
}   [get_ports   {   ACL_INT[2]   }];   #IO_L20P_T3_A20_15
Sch=acl_int[2]


  ##Temperature Sensor

  #set_property -dict { PACKAGE_PIN C14   IOSTANDARD LVCMOS33
} [get_ports { TMP_SCL }]; #IO_L1N_T0_AD0N_15 Sch=tmp_scl
  #set_property -dict { PACKAGE_PIN C15   IOSTANDARD LVCMOS33
} [get_ports { TMP_SDA }]; #IO_L12N_T1_MRCC_15 Sch=tmp_sda
  #set_property -dict { PACKAGE_PIN D13   IOSTANDARD LVCMOS33
} [get_ports { TMP_INT }]; #IO_L6N_T0_VREF_15 Sch=tmp_int
  #set_property -dict { PACKAGE_PIN B14   IOSTANDARD LVCMOS33
} [get_ports { TMP_CT }]; #IO_L2N_T0_AD8N_15 Sch=tmp_ct

  ##Omnidirectional Microphone

  #set_property -dict { PACKAGE_PIN J5    IOSTANDARD LVCMOS33
} [get_ports { M_CLK }]; #IO_25_35 Sch=m_clk
  #set_property -dict { PACKAGE_PIN H5    IOSTANDARD LVCMOS33
} [get_ports { M_DATA }]; #IO_L24N_T3_35 Sch=m_data
  #set_property -dict { PACKAGE_PIN F5    IOSTANDARD LVCMOS33
} [get_ports { M_LRSEL }]; #IO_0_35 Sch=m_lrsel


  ##PWM Audio Amplifier

  #set_property -dict { PACKAGE_PIN A11   IOSTANDARD LVCMOS33
} [get_ports { AUD_PWM }]; #IO_L4N_T0_15 Sch=aud_pwm
  #set_property -dict { PACKAGE_PIN D12   IOSTANDARD LVCMOS33
} [get_ports { AUD_SD }]; #IO_L6P_T0_15 Sch=aud_sd


  ##USB-RS232 Interface

  #set_property -dict { PACKAGE_PIN C4    IOSTANDARD LVCMOS33
}   [get_ports   {   UART_TXD_IN   }];   #IO_L7P_T1_AD6P_35
Sch=uart_txd_in
  #set_property -dict { PACKAGE_PIN D4    IOSTANDARD LVCMOS33
}   [get_ports   {   UART_RXD_OUT   }];   #IO_L11N_T1_SRCC_35
Sch=uart_rxd_out
  #set_property -dict { PACKAGE_PIN D3    IOSTANDARD LVCMOS33
} [get_ports { UART_CTS }]; #IO_L12N_T1_MRCC_35 Sch=uart_cts
  #set_property -dict { PACKAGE_PIN E5    IOSTANDARD LVCMOS33
} [get_ports { UART_RTS }]; #IO_L5N_T0_AD13N_35 Sch=uart_rts

  ##USB HID (PS/2)

  #set_property -dict { PACKAGE_PIN F4    IOSTANDARD LVCMOS33
} [get_ports { PS2_CLK }]; #IO_L13P_T2_MRCC_35 Sch=ps2_clk
  #set_property -dict { PACKAGE_PIN B2    IOSTANDARD LVCMOS33
} [get_ports { PS2_DATA }]; #IO_L10N_T1_AD15N_35 Sch=ps2_data


  ##SMSC Ethernet PHY

  #set_property -dict { PACKAGE_PIN C9    IOSTANDARD LVCMOS33
} [get_ports { ETH_MDC }]; #IO_L11P_T1_SRCC_16 Sch=eth_mdc
  #set_property -dict { PACKAGE_PIN A9    IOSTANDARD LVCMOS33
} [get_ports { ETH_MDIO }]; #IO_L14N_T2_SRCC_16 Sch=eth_mdio
  #set_property -dict { PACKAGE_PIN B3    IOSTANDARD LVCMOS33
} [get_ports { ETH_RSTN }]; #IO_L10P_T1_AD15P_35 Sch=eth_rstn
  #set_property -dict { PACKAGE_PIN D9    IOSTANDARD LVCMOS33
} [get_ports { ETH_CRSDV }]; #IO_L6N_T0_VREF_16 Sch=eth_crsdv
  #set_property -dict { PACKAGE_PIN C10   IOSTANDARD LVCMOS33
}   [get_ports   {   ETH_RXERR   }];   #IO_L13N_T2_MRCC_16
Sch=eth_rxerr
  #set_property -dict { PACKAGE_PIN C11   IOSTANDARD LVCMOS33
}   [get_ports   {   ETH_RXD[0]   }];   #IO_L13P_T2_MRCC_16
Sch=eth_rxd[0]
  #set_property -dict { PACKAGE_PIN D10   IOSTANDARD LVCMOS33
}   [get_ports   {   ETH_RXD[1]   }];   #IO_L19N_T3_VREF_16
Sch=eth_rxd[1]
  #set_property -dict { PACKAGE_PIN B9    IOSTANDARD LVCMOS33
} [get_ports { ETH_TXEN }]; #IO_L11N_T1_SRCC_16 Sch=eth_txen
  #set_property -dict { PACKAGE_PIN A10   IOSTANDARD LVCMOS33
}   [get_ports   {   ETH_TXD[0]   }];   #IO_L14P_T2_SRCC_16
Sch=eth_txd[0]
  #set_property -dict { PACKAGE_PIN A8    IOSTANDARD LVCMOS33
}   [get_ports   {   ETH_TXD[1]   }];   #IO_L12N_T1_MRCC_16
Sch=eth_txd[1]
  #set_property -dict { PACKAGE_PIN D5    IOSTANDARD LVCMOS33
}   [get_ports   {   ETH_REFCLK   }];   #IO_L11P_T1_SRCC_35
Sch=eth_refclk
  #set_property -dict { PACKAGE_PIN B8    IOSTANDARD LVCMOS33
} [get_ports { ETH_INTN }]; #IO_L12P_T1_MRCC_16 Sch=eth_intn


  ##Quad SPI Flash

  #set_property -dict { PACKAGE_PIN K17   IOSTANDARD LVCMOS33
}   [get_ports   {   QSPI_DQ[0]   }];   #IO_L1P_T0_D00_MOSI_14
Sch=qspi_dq[0]
  #set_property -dict { PACKAGE_PIN K18   IOSTANDARD LVCMOS33
}   [get_ports   {   QSPI_DQ[1]   }];   #IO_L1N_T0_D01_DIN_14
Sch=qspi_dq[1]
  #set_property -dict { PACKAGE_PIN L14   IOSTANDARD LVCMOS33
}   [get_ports   {   QSPI_DQ[2]   }];   #IO_L2P_T0_D02_14
Sch=qspi_dq[2]
  #set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33
}   [get_ports   {   QSPI_DQ[3]   }];   #IO_L2N_T0_D03_14
Sch=qspi_dq[3]
  #set_property -dict { PACKAGE_PIN L13   IOSTANDARD LVCMOS33
} [get_ports { QSPI_CSN }]; #IO_L6P_T0_FCS_B_14 Sch=qspi_csn
```