Seven-Segment Display Controller Using Counters (Feb 2020)

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

Table of Contents

[I. Introduction 1](#_Toc32850491)

[II. Procedure 1](#_Toc32850492)

[A. pulseGenerator.vhd 1](#_Toc32850493)

[B. seg7\_controller.vhd 1](#_Toc32850494)

[C. shiftReg 1](#_Toc32850495)

[D. lab3\_top.vhd 1](#_Toc32850496)

[III. Testing Strategy 2](#_Toc32850497)

[IV. Analysis/Conclusion 3](#_Toc32850498)

[Appendix 3](#_Toc32850499)

[A. Lab 3 Top Module 3](#_Toc32850500)

[B. 7-Segment Controller Module 3](#_Toc32850501)

[C. Shift Register Module 4](#_Toc32850502)

[D. Pulse Generator Module 5](#_Toc32850503)

[E. Testbench – Shift Register 5](#_Toc32850504)

[F. Testbench – 7-Segment Controller 6](#_Toc32850505)

[G. Testbench – Lab 3 Top Module 6](#_Toc32850506)

[H. 7-Segment Display Simulation Decoder Module 7](#_Toc32850507)

[I. Utilization Report 7](#_Toc32850508)

[J. Constraints File 9](#_Toc32850509)

# Introduction

T

HIS 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.

# Procedure

## 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.

## 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.

## 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.

## 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.

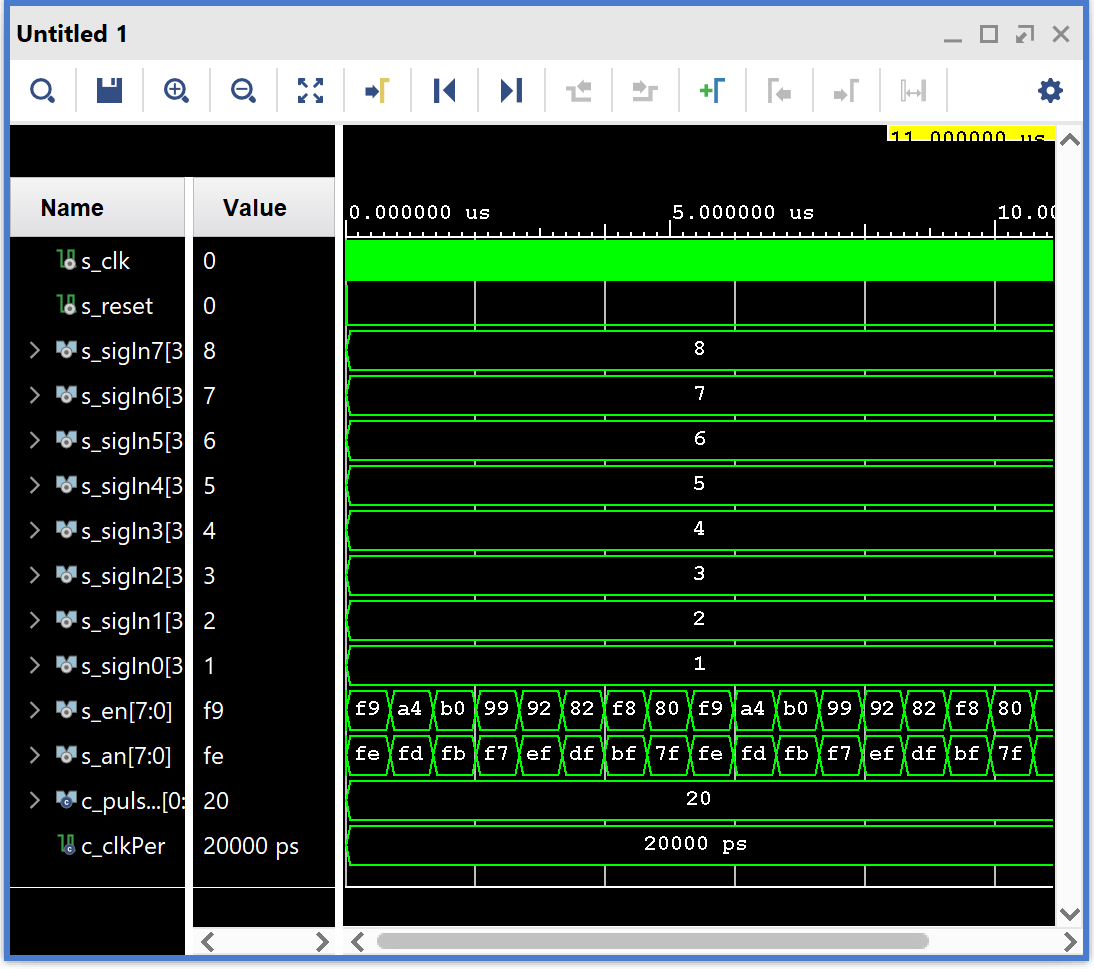


*Fig. 2 – Block Diagram of Lab 3*

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

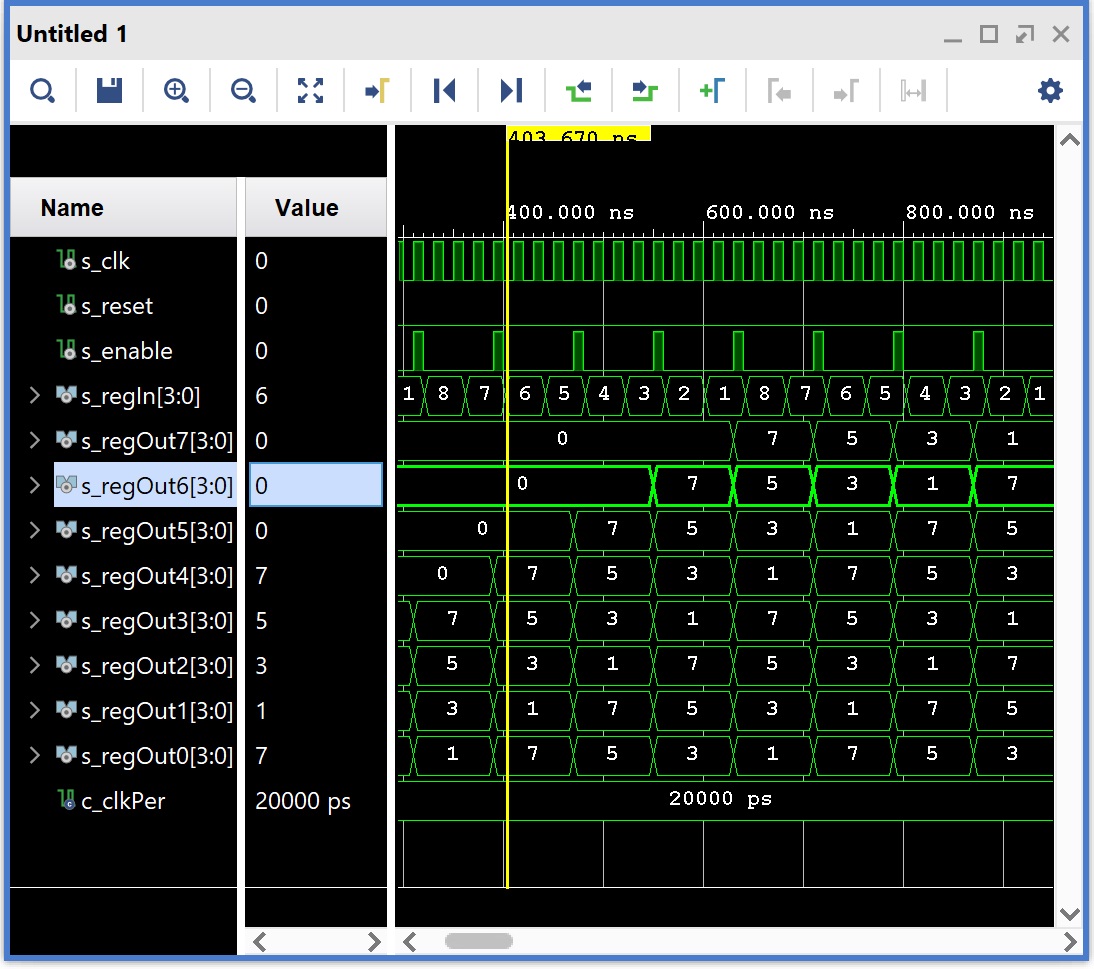
# Testing Strategy

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*

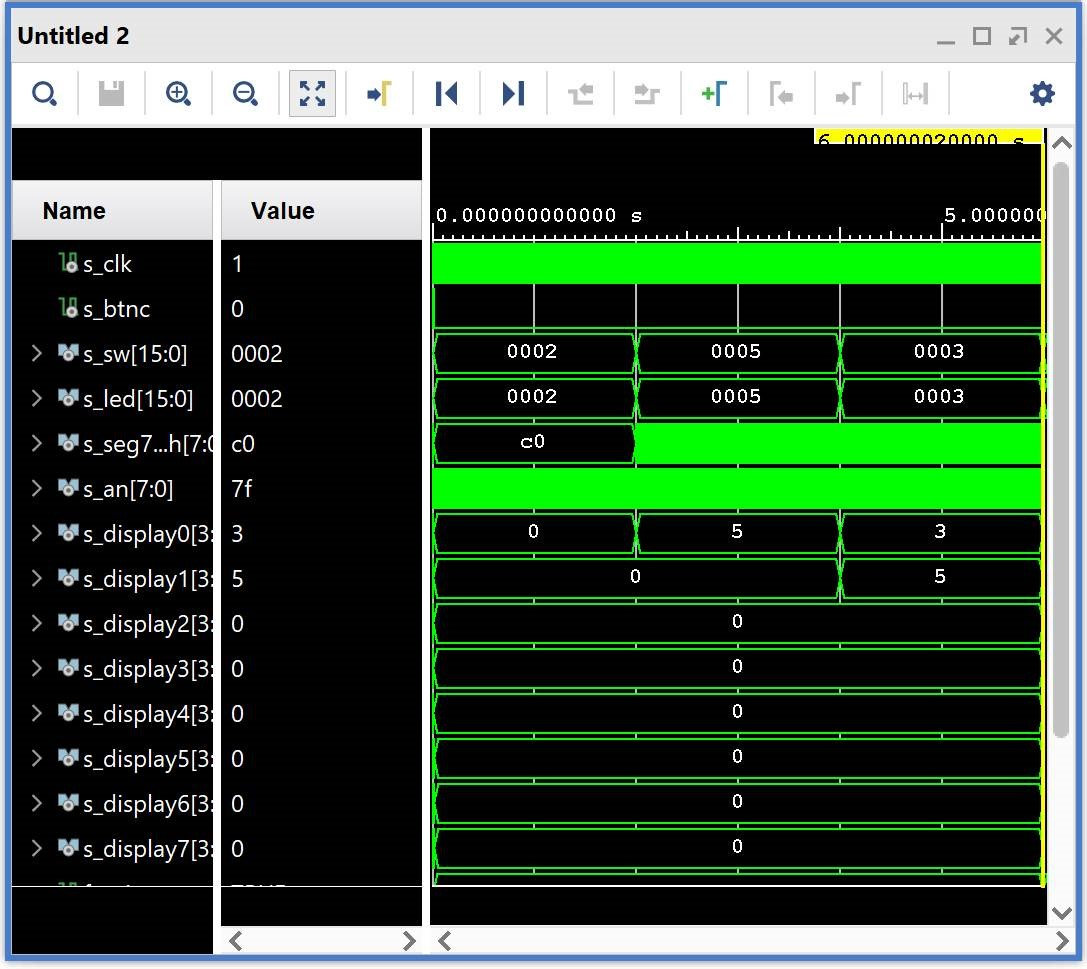
The above testbench was simple – with different values for each input (and a sped-up pulse), the module cycled through the values for the anodes and the 7-segment display.



*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.

# 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.

## 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;

## 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;

## 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;

## 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;

## 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;

## 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;

## 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;

## 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;

## 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 |

+----------+------+

## 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