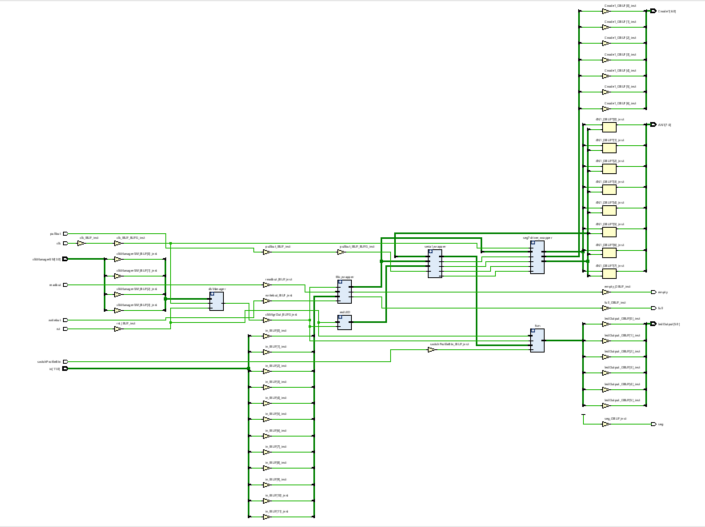
**ECE3300L Lab10 Group H Report (Sherwin Sathish & Mohamed Hamida)**

**SCHEMATIC:**

****

***Xdc for top.v:***

## This file is a general .xdc for the Nexys A7-100T

## 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 {clk}];

##Switches

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { redSW[0] }]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { redSW[1] }]; #IO\_L3N\_T0\_DQS\_EMCCLK\_14 Sch=sw[1]

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { redSW[2] }]; #IO\_L6N\_T0\_D08\_VREF\_14 Sch=sw[2]

set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports { redSW[3] }]; #IO\_L13N\_T2\_MRCC\_14 Sch=sw[3]

set\_property -dict { PACKAGE\_PIN R17 IOSTANDARD LVCMOS33 } [get\_ports { greenSW[0] }]; #IO\_L12N\_T1\_MRCC\_14 Sch=sw[4]

set\_property -dict { PACKAGE\_PIN T18 IOSTANDARD LVCMOS33 } [get\_ports { greenSW[1] }]; #IO\_L7N\_T1\_D10\_14 Sch=sw[5]

set\_property -dict { PACKAGE\_PIN U18 IOSTANDARD LVCMOS33 } [get\_ports { greenSW[2] }]; #IO\_L17N\_T2\_A13\_D29\_14 Sch=sw[6]

set\_property -dict { PACKAGE\_PIN R13 IOSTANDARD LVCMOS33 } [get\_ports { greenSW[3] }]; #IO\_L5N\_T0\_D07\_14 Sch=sw[7]

set\_property -dict { PACKAGE\_PIN T8 IOSTANDARD LVCMOS18 } [get\_ports { blueSW[0]}]; #IO\_L24N\_T3\_34 Sch=sw[8]

set\_property -dict { PACKAGE\_PIN U8 IOSTANDARD LVCMOS18 } [get\_ports { blueSW[1]}]; #IO\_25\_34 Sch=sw[9]

set\_property -dict { PACKAGE\_PIN R16 IOSTANDARD LVCMOS33 } [get\_ports { blueSW[2] }]; #IO\_L15P\_T2\_DQS\_RDWR\_B\_14 Sch=sw[10]

set\_property -dict { PACKAGE\_PIN T13 IOSTANDARD LVCMOS33 } [get\_ports { blueSW[3] }]; #IO\_L23P\_T3\_A03\_D19\_14 Sch=sw[11]

set\_property -dict { PACKAGE\_PIN H6 IOSTANDARD LVCMOS33 } [get\_ports { clkManagerSW[0] }]; #IO\_L24P\_T3\_35 Sch=sw[12]

set\_property -dict { PACKAGE\_PIN U12 IOSTANDARD LVCMOS33 } [get\_ports { clkManagerSW[1] }]; #IO\_L20P\_T3\_A08\_D24\_14 Sch=sw[13]

set\_property -dict { PACKAGE\_PIN U11 IOSTANDARD LVCMOS33 } [get\_ports { clkManagerSW[2] }]; #IO\_L19N\_T3\_A09\_D25\_VREF\_14 Sch=sw[14]

set\_property -dict { PACKAGE\_PIN V10 IOSTANDARD LVCMOS33 } [get\_ports { clkManagerSW[3] }]; #IO\_L21P\_T3\_DQS\_14 Sch=sw[15]

## RGB LEDs

set\_property -dict { PACKAGE\_PIN R12 IOSTANDARD LVCMOS33 } [get\_ports { ledOutput[2] }]; #IO\_L5P\_T0\_D06\_14 Sch=led16\_b

set\_property -dict { PACKAGE\_PIN M16 IOSTANDARD LVCMOS33 } [get\_ports { ledOutput[1] }]; #IO\_L10P\_T1\_D14\_14 Sch=led16\_g

set\_property -dict { PACKAGE\_PIN N15 IOSTANDARD LVCMOS33 } [get\_ports { ledOutput[0] }]; #IO\_L11P\_T1\_SRCC\_14 Sch=led16\_r

set\_property -dict { PACKAGE\_PIN G14 IOSTANDARD LVCMOS33 } [get\_ports { ledOutput[5]}]; #IO\_L15N\_T2\_DQS\_ADV\_B\_15 Sch=led17\_b

set\_property -dict { PACKAGE\_PIN R11 IOSTANDARD LVCMOS33 } [get\_ports { ledOutput[4]}]; #IO\_0\_14 Sch=led17\_g

set\_property -dict { PACKAGE\_PIN N16 IOSTANDARD LVCMOS33 } [get\_ports { ledOutput[3]}]; #IO\_L11N\_T1\_SRCC\_14 Sch=led17\_r

##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 { rst }]; #IO\_L9P\_T1\_DQS\_14 Sch=btnc

set\_property -dict { PACKAGE\_PIN M18 IOSTANDARD LVCMOS33 } [get\_ports { switchProfileBtn }]; #IO\_L4N\_T0\_D05\_14 Sch=btnu

#set\_property -dict { PACKAGE\_PIN P17 IOSTANDARD LVCMOS33 } [get\_ports { rstTop }]; #IO\_L12P\_T1\_MRCC\_14 Sch=btnl

#set\_property -dict { PACKAGE\_PIN M17 IOSTANDARD LVCMOS33 } [get\_ports { ld }]; #IO\_L10N\_T1\_D15\_14 Sch=btnr

#set\_property -dict { PACKAGE\_PIN P18 IOSTANDARD LVCMOS33 } [get\_ports { btn }]; #IO\_L9N\_T1\_DQS\_D13\_14 Sch=btnd

**Fifo.v**

`timescale 1ns / 1ps

//`default\_nettype none

module fifo #(

parameter WIDTH = 12,

parameter DEPTH = 16

)(

input [WIDTH-1:0] data\_in,

input wire clk,

input wire write,

input wire read,

output reg [WIDTH-1:0] data\_out,

output wire fifo\_full,

output wire fifo\_empty,

output wire fifo\_not\_empty,

output wire fifo\_not\_full

);

integer count;

// memory will contain the FIFO data.

reg [WIDTH-1:0] memory [0:DEPTH-1];

// $clog2(DEPTH+1)-2 to count from 0 to DEPTH

reg [$clog2(DEPTH)-1:0] write\_ptr;

reg [$clog2(DEPTH)-1:0] read\_ptr;

assign fifo\_empty = ( write\_ptr == read\_ptr ) ? 1'b1 : 1'b0;

assign fifo\_full = ( write\_ptr == (DEPTH-1) ) ? 1'b1 : 1'b0;

assign fifo\_not\_empty = ~fifo\_empty;

assign fifo\_not\_full = ~fifo\_full;

always @ (posedge clk) begin

if ( write ) begin

memory[write\_ptr] <= data\_in;

end

if ( read ) begin

data\_out <= memory[read\_ptr];

end

end

always @ ( posedge clk ) begin

if ( write ) begin

write\_ptr <= write\_ptr + 1;

end

if ( read && fifo\_not\_empty ) begin

read\_ptr <= read\_ptr + 1;

end

end

endmodule

The fifo basically stores memory across a particular width at a certain depth. Write and read pointers are used to save the data into the memory array, and there is a counter for these pointers to store memory at different indexes. Flags of full and empty are used to prevent pushing data that doesn’t exist and prevent loading in too much data.

**SERIALIZER.v**

`timescale 1ns / 1ps

module SERIALIZER(

input [11:0] pull\_in,

input pull\_but,

output reg [3:0] out\_red,

output reg [3:0] out\_green,

output reg [3:0] out\_blue

);

reg state=0;

always@(pull\_but or pull\_in)

begin

if (pull\_but)

state = ~state;

else

state = state;

end

always@(pull\_but or state)

begin

case(state)

1'b0:

begin end

1'b1:

begin

out\_red <= pull\_in[3:0];

out\_green <= pull\_in[7:4];

out\_blue <= pull\_in[11:8];

end

endcase

end

endmodule

`timescale 1ns / 1ps

module SERIALIZER\_TB(

);

reg [11:0] pull\_in\_tb;

reg pull\_but\_tb;

wire [3:0] out\_red\_tb;

wire [3:0] out\_green\_tb;

wire [3:0] out\_blue\_tb;

SERIALIZER COMP(

.pull\_in(pull\_in\_tb),

.pull\_but(pull\_but\_tb),

.out\_red(out\_red\_tb),

.out\_green(out\_green\_tb),

.out\_blue(out\_blue\_tb)

);

initial

begin

pull\_in\_tb = 56;

pull\_but\_tb = 1;

#10

pull\_in\_tb = 56;

pull\_but\_tb = 0;

#100

$finish;

end

endmodule

The serializer pulls an 12-bit number from the fifo using a pull button, and splits it into 3 4-bit numbers that go into the red, green, and blue PWM respectively.

**Top.v**

`timescale 1ns / 1ps

module top

(

input[3:0] clkManagerSW,

input[11:0] in,

input clk,

input rst,

input writebut,

input readbut,

input pullbut,

input switchProfileBtn,

output full,

output empty,

output [6:0] Cnode1,

output [7:0] AN1,

output seg,

output[5:0] ledOutput

);

wire clkMgrOut;

CLKMANAGER clkManager(

.clk(clk),

.rst(rst),

.SW(clkManagerSW),

.clkout(clkMgrOut)

);

fifo fifo\_wrapper(

.data\_in(in),

.clk(clkMgrOut),

.write(writebut),

.read(readbut),

.data\_out(seg7Digit[23:12]),

.fifo\_full(full),

.fifo\_empty(empty)

);

wire [31:0] seg7Digit;

SERIALIZER serial\_wrapper (

.pull\_in(seg7Digit[23:12]),

.pull\_but(pullbut),

.out\_red(seg7Digit[3:0]),

.out\_green(seg7Digit[7:4]),

.out\_blue(seg7Digit[11:8])

);

wire tmpEn = 1'b1;

wire [2:0] tmpLEDOUT;

PWM redLED(

.inputSW(seg7Digit[3:0]),

.clk(clkMgrOut),

.rst(rst),

.en(tmpEn),

.result(tmpLEDOUT[0])

);

PWM greenLED(

.inputSW(seg7Digit[7:4]),

.clk(clkMgrOut),

.rst(rst),

.en(tmpEn),

.result(tmpLEDOUT[1])

);

PWM blueLED(

.inputSW(seg7Digit[11:8]),

.clk(clkMgrOut),

.rst(rst),

.en(tmpEn),

.result(tmpLEDOUT[2])

);

FSM fsm(

.sel(tmpLEDOUT),

.fsmBTN(switchProfileBtn),

.rst(rst),

.clk(clkMgrOut),

.led1(ledOutput[2:0]),

.led2(ledOutput[5:3])

);

SEGDRIVE seg7driver\_wrapper(

.nexysCLK(clk),

.inDigit(seg7Digit),

.Cnode(Cnode1),

.dp(seg),

.AN(AN1)

);

endmodule

Our top file instantiated the PWM redLed, PWM blueLed, and PWM greenLed, the fifo, serializer, and the segdriver. These three PWM modules output to the FSM module which then outputs to our XDC output. We also have a clock manager which is creating an artificial clock to input into each module. The fifo stores user input into a 2-D array then sends it off to a serializer which splits the data into 3 parts that each go into the input of a PWM.

**CLKMANAGER.v**



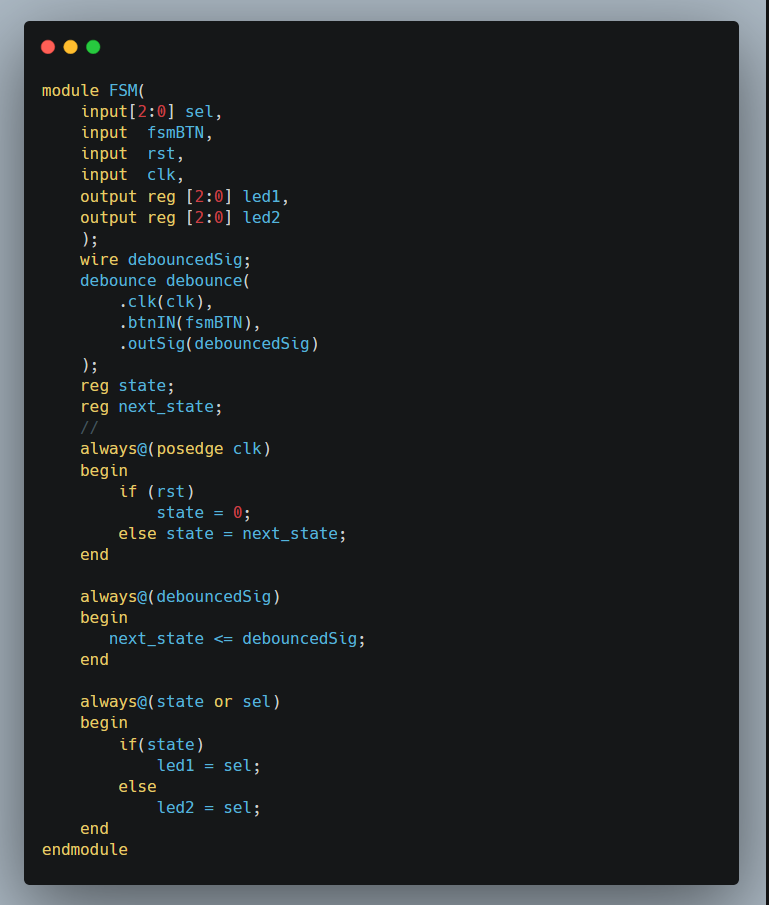
We used a 32 bit clock manager to feed variable frequencies into our PWM LED modules, and our FSM module.

**PWM.v**



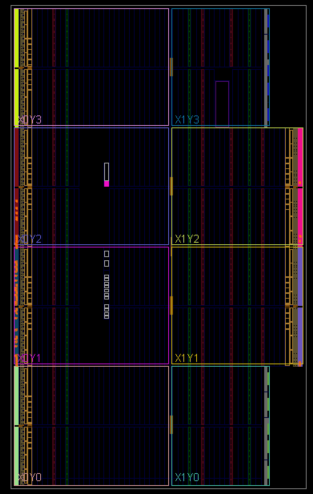
This is the generic PWM file which instantiates into the redLed, blueLed, and greenLed to send off those values to the FSM and eventually to the dedicated output led on our Nexys A7 board. It uses the UPCOUNT and comparator modules to compare the values from the switches to the counter and based off those values it will send the appropriate signals to the FSM for outputting.

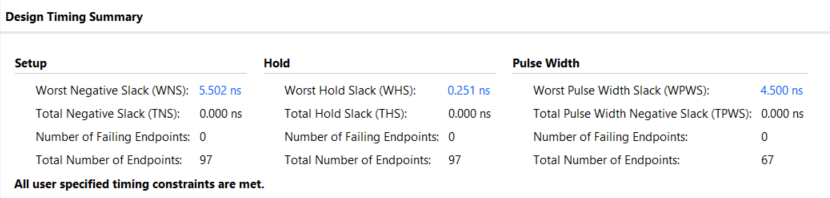
**FSM.v**

****

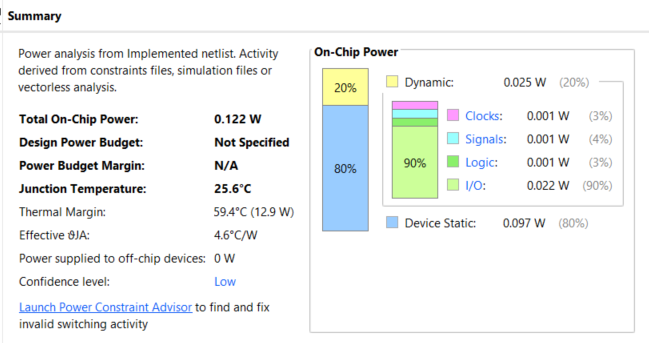
This simple FSM (finite state machine) module will iterate between two states depending on the input button. Depending on the button output (which is debounced using the button debouncer) it will either fill the led1 values or led2 values.

**IMPLEMENTED DESIGN/ TIMING SUMMARY:**

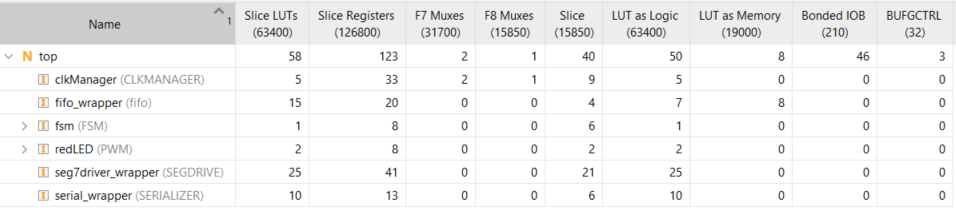
****

****

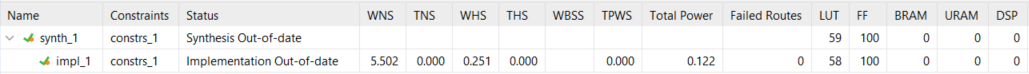
**POWER SUMMARY:**

****

**UTILIZATION:**

****

**RESOURCE USAGE:**

****