**Department of Electrical and Computer Engineering**

Author: Zainab Hussein

Title: FPGA & Logic Design Refresher

Date: 2/4/2017

Time spent: 5 Hours

**Block diagram:**



**System Verilog files:**

1. Debouncer & Level-To-Pulse converter

//-----------------------------------------------------------------------------

// Title : debounce -- Button debouncer

// Project : Lab 7

//-----------------------------------------------------------------------------

// File : debounce.sv

// Author : Jon Wallace

// Created : 15 Oct. 2015

// Last modified : 15 Oct. 2015

//-----------------------------------------------------------------------------

// Description :

// This module provides code to debounce a raw button input.

// Inputs:

// clk Clock

// button\_in Raw button input with bounce

// Outputs:

// button\_out Debounced button

// pulse Provides a one-clock pulse when button is pressed

//-----------------------------------------------------------------------------

module debounce(input logic clk, //should be synchronized

input logic button\_in,

output logic button\_out,

output logic pulse);

parameter DEBOUNCE\_TIME\_MS = 5;

parameter CLKFREQ = 100\_000\_000;

parameter WAIT\_COUNT = DEBOUNCE\_TIME\_MS\*(CLKFREQ/1000);

// States for button debouncing

logic button\_state, button\_state\_next;

// Counter for debouncing

logic [26:0] count\_reg, count\_next;

// Counter and button state register

always\_ff @(posedge clk)

begin

button\_state <= button\_state\_next;

count\_reg <= count\_next;

end

// Next-state / output logic

always\_comb

begin

// Defaults

button\_state\_next = button\_state;

count\_next = count\_reg;

pulse = 1'b0;

// Does the button input match the stored button state?

if (button\_in == button\_state)

// Yes, so just reset the counter

count\_next = 0;

else if (count\_reg == WAIT\_COUNT-1)

begin

// No, so if the counter is done, transition to the other state

button\_state\_next = ~button\_state;

count\_next = 0;

// Generate a pulse if going from 0=>1

pulse = ~button\_state;

end

else

// Have not reached wait count yet, so increment counter.

count\_next = count\_reg + 1;

end // always\_comb

assign button\_out = button\_state;

endmodule // debounce

1. Clock divider

//-----------------------------------------------------------------------------

// Title : clkdiv - parameterized clock divider

// Project : ECE 211 - Digital Circuit 1

//-----------------------------------------------------------------------------

// File : clkdiv.sv

// Author : John Nestor

// Created : 09.08.2011

// Last modified : 10.16.2015

//-----------------------------------------------------------------------------

// Description :

// This module divides the 100MHz clock on the Nexys4 board down to a lower

// frequency. Set the DIVFREQ parameter to the desired frequency in Hz.

// If a frequency lower than 1 Hz is desired, the DIVBITS bitwidth parameter

// must be increased. If a higher frequency is used, we assume that synthesis

// will trim the unused most signficant counter bits and logic.

// To use, instantiate this module whiel setting the DIVFREQ parameter to the

// desired frequency in Hz.

// For example, to generate a 1 Hz clock, instantiate a module as follows:

//

// clkdiv #(.DIVFREQ(1)) U\_DIV (clk, reset, sclk);

//

// Where:

// clk is the 50MHz system clock that arrives on an input pin of the FPGA

// (see the documentation)

// reset is a signal that resets the clock divider counter

// (connect it to zero if unused)

// sclk is the output clock - connect this to your logic

//

//-----------------------------------------------------------------------------

module clkdiv(input logic clk, input logic reset, output logic sclk);

parameter DIVFREQ = 100; // desired frequency in Hz (change as needed)

parameter DIVBITS = 26; // enough bits to divide 100MHz down to 1 Hz

parameter CLKFREQ = 100\_000\_000;

parameter DIVAMT = (CLKFREQ / DIVFREQ) / 2;

logic [DIVBITS-1:0] q;

always\_ff @(posedge clk)

if (reset)

begin

q <= 0;

sclk <= 0;

end

else if (q == DIVAMT-1) begin

q <= 0;

sclk <= ~sclk;

end

else q <= q + 1;

endmodule // clkdiv

1. Nexys4 top level

module nexys4 (

// un-comment the ports that you will use

input logic CLK100MHZ,

// input logic [15:0] SW,

input logic BTNC,

input logic BTNU, //reset

input logic BTNL,

input logic BTNR,

// input logic BTND,

// output logic [6:0] SEGS,

// output logic [7:0] AN,

// output logic DP,

output logic [15:0] led

// input logic UART\_TXD\_IN,

// input logic UART\_RTS,

// output logic UART\_RXD\_OUT,

// output logic UART\_CTS

);

//internal signals

logic btnc\_pls, btnr\_pls, btnl\_pls, btnu\_pls;

logic btnc\_dbn, btnr\_dbn, btnl\_dbn, btnu\_dbn;

// add SystemVerilog code & module instantiations here

//debouncing

debounce U\_DBN\_C( .clk(CLK100MHZ), .button\_in(BTNC), .button\_out(btnc\_dbn), .pulse(btnc\_pls) );

debounce U\_DBN\_R( .clk(CLK100MHZ), .button\_in(BTNR), .button\_out(btnr\_dbn), .pulse(btnr\_pls) );

debounce U\_DBN\_L( .clk(CLK100MHZ), .button\_in(BTNL), .button\_out(btnl\_dbn), .pulse(btnl\_pls) );

debounce U\_DBN\_U( .clk(CLK100MHZ), .button\_in(BTNU), .button\_out(btnu\_dbn), .pulse(btnu\_pls) );

//fsm logic for blink mode

logic [1:0] led\_state;

typedef enum logic [1:0] {

solid = 2'b00,

oneHz = 2'b01,

twoHz = 2'b10

} states\_t;

states\_t state, next;

always\_ff @(posedge CLK100MHZ)

if (btnu\_pls) state <= solid;

else state <= next;

always\_comb

begin

next = solid;

case( state )

solid:

begin

if (btnc\_pls) next = oneHz;

else next = solid;

end

oneHz:

begin

if (btnc\_pls) next = twoHz;

else next = oneHz;

end

twoHz:

begin

if (btnc\_pls) next = solid;

else next = twoHz;

end

default:

next = solid;

endcase

end

assign led\_state = state;

//counter for led shift

logic [3:0] cnt;

always\_ff @(posedge CLK100MHZ)

if (btnu\_pls) cnt <= 4'd0;

else if (btnr\_pls) cnt <= cnt - 1; //shift right

else if (btnl\_pls) cnt <= cnt + 1; //shift left

// use demux to get 16 bits of LED

logic [15:0] led\_shift;

//testing always comb block

`ifdef TEST

assign led\_shift = 16'b0000000000000001;

`else

always\_comb

case(cnt)

4'd0: led\_shift = 16'b0000000000000001;

4'd1: led\_shift = 16'b0000000000000010;

4'd2: led\_shift = 16'b0000000000000100;

4'd3: led\_shift = 16'b0000000000001000;

4'd4: led\_shift = 16'b0000000000010000;

4'd5: led\_shift = 16'b0000000000100000;

4'd6: led\_shift = 16'b0000000001000000;

4'd7: led\_shift = 16'b0000000010000000;

4'd8: led\_shift = 16'b0000000100000000;

4'd9: led\_shift = 16'b0000001000000000;

4'd10: led\_shift = 16'b0000100000000000;

4'd11: led\_shift = 16'b0001000000000000;

4'd12: led\_shift = 16'b0010000000000000;

4'd13: led\_shift = 16'b0100000000000000;

4'd14: led\_shift = 16'b1000000000000000;

4'd15: led\_shift = 16'b0000000000000001;

default: led\_shift = 16'b0000000000000001;

endcase

`endif

//blink rate mux

parameter BAUD = 1; // desired frequency in Hz

parameter TWICEBAUD = 2;

//clkdiv

logic oneHz\_clk, twoHz\_clk, solid\_clk;

clkdiv #(.DIVFREQ(BAUD)) U\_CD\_oneHz(.clk(CLK100MHZ), .reset(BTNU), .sclk(oneHz\_clk));

clkdiv #(.DIVFREQ(TWICEBAUD)) U\_CD\_twoHz(.clk(CLK100MHZ), .reset(BTNU), .sclk(twoHz\_clk));

// clkdiv #(.DIVFREQ(100)) U\_CD\_solid(.clk(CLK100MHZ), .reset(BTNU), .sclk(solid\_clk));

//mux for blink rate selection based on blink mode

logic blink;

//testing blink rate mux

`ifdef TEST

assign blink = 1;

`else

always\_comb

unique case (led\_state)

2'd0 : blink = 1;

2'd1 : blink = oneHz\_clk;

2'd2 : blink = twoHz\_clk;

default : blink = 0; // fills all bits with 0s

endcase // case(led\_state)

`endif

//display

//replicate blink 16 bits and wire led

assign led = ( {16{blink}} & led\_shift );

// assign led = 16'b1010101010101010; //for testing

endmodule // nexys4

**Testbench**

**Waveforms annotated**

Final design worked correctly on the FPGA board and accepted by Prof. Shmult

Making lab better - the first lab went well.