Department of Electrical and Computer Engineering

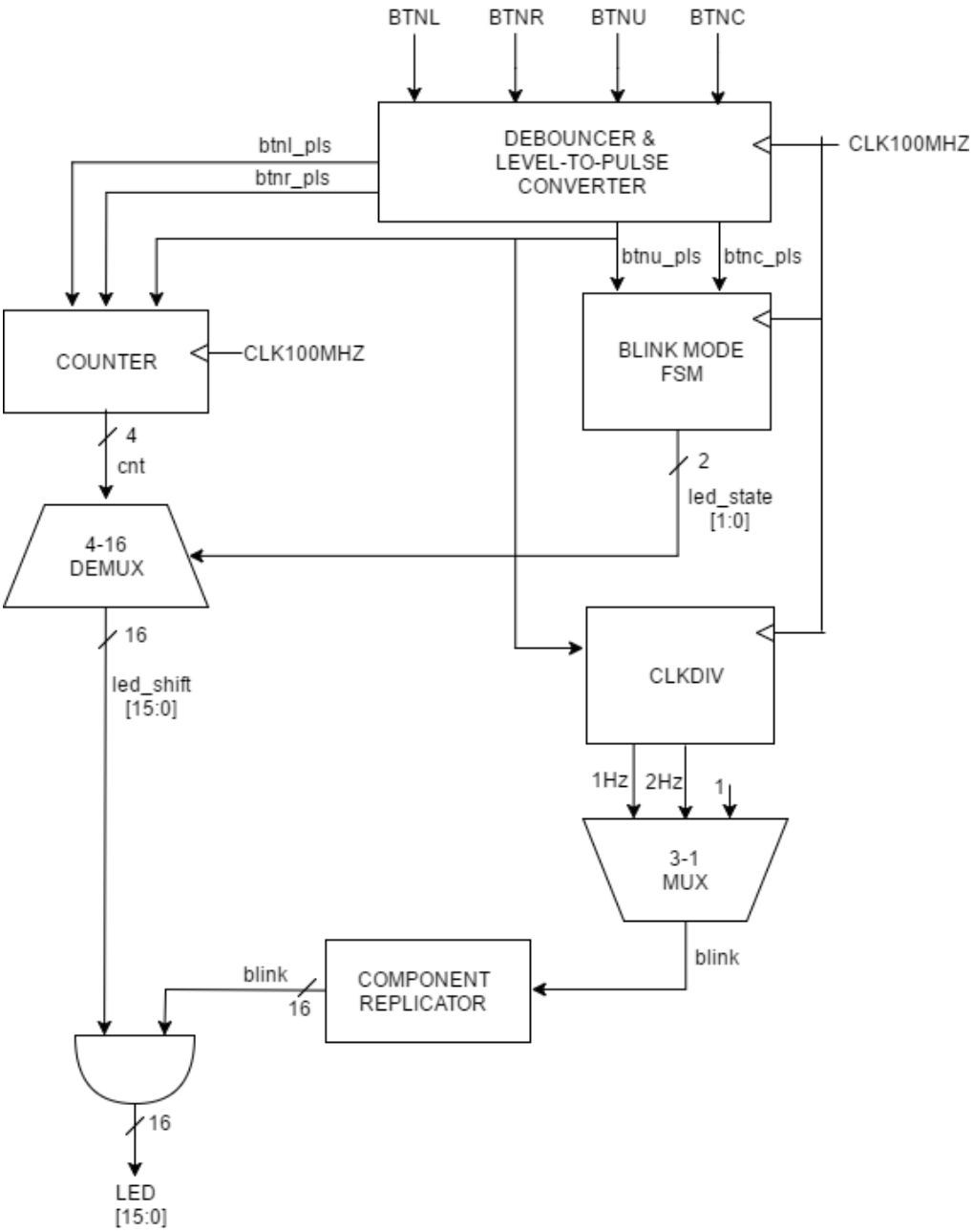
Author: Zainab Hussein

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
```

2. 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
```

3. Nexys4toplevel

```
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;
```

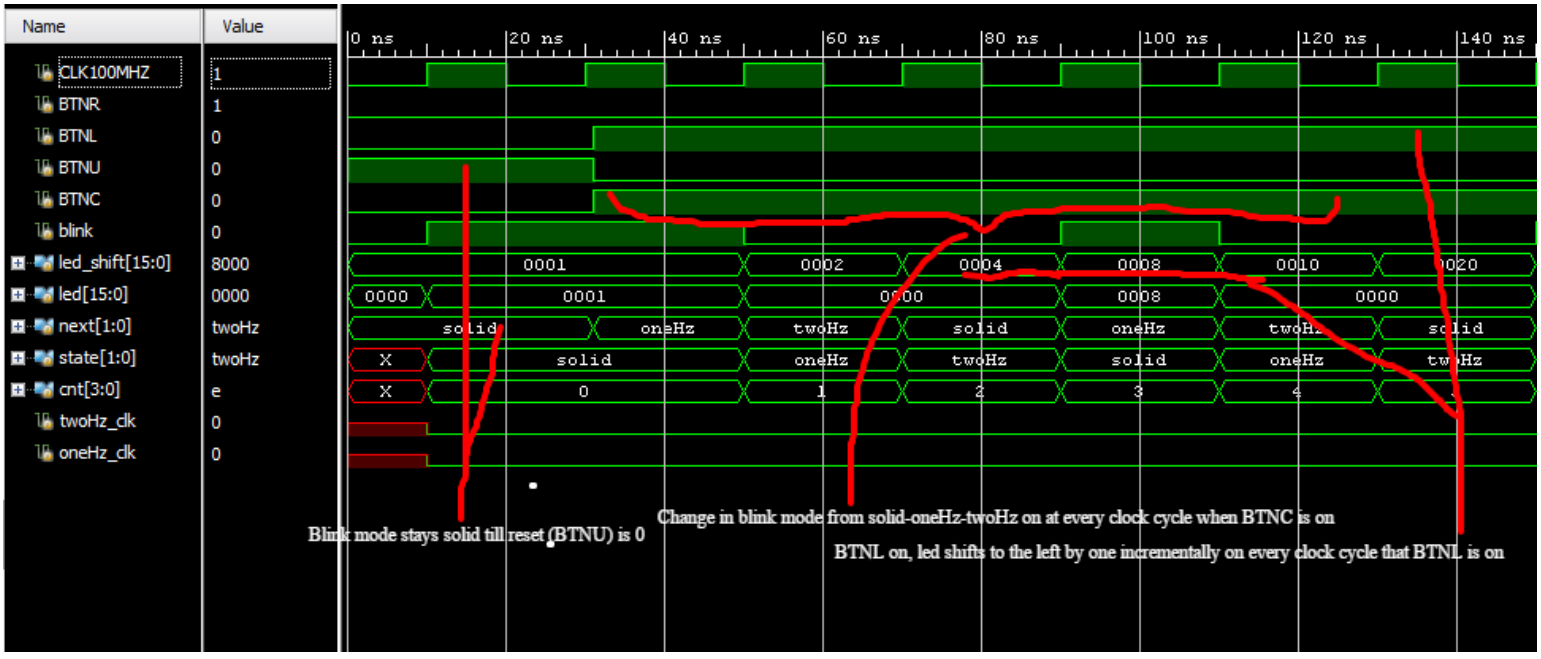
```
    4'd12: led_shift = 16'b0010000000000000;
    4'd13: led_shift = 16'b0100000000000000;
    4'd14: led_shift = 16'b1000000000000000;
    4'd15: led_shift = 16'b0000000000000001;
    default: led_shift = 16'b0000000000000001;
 endcase

 `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
```

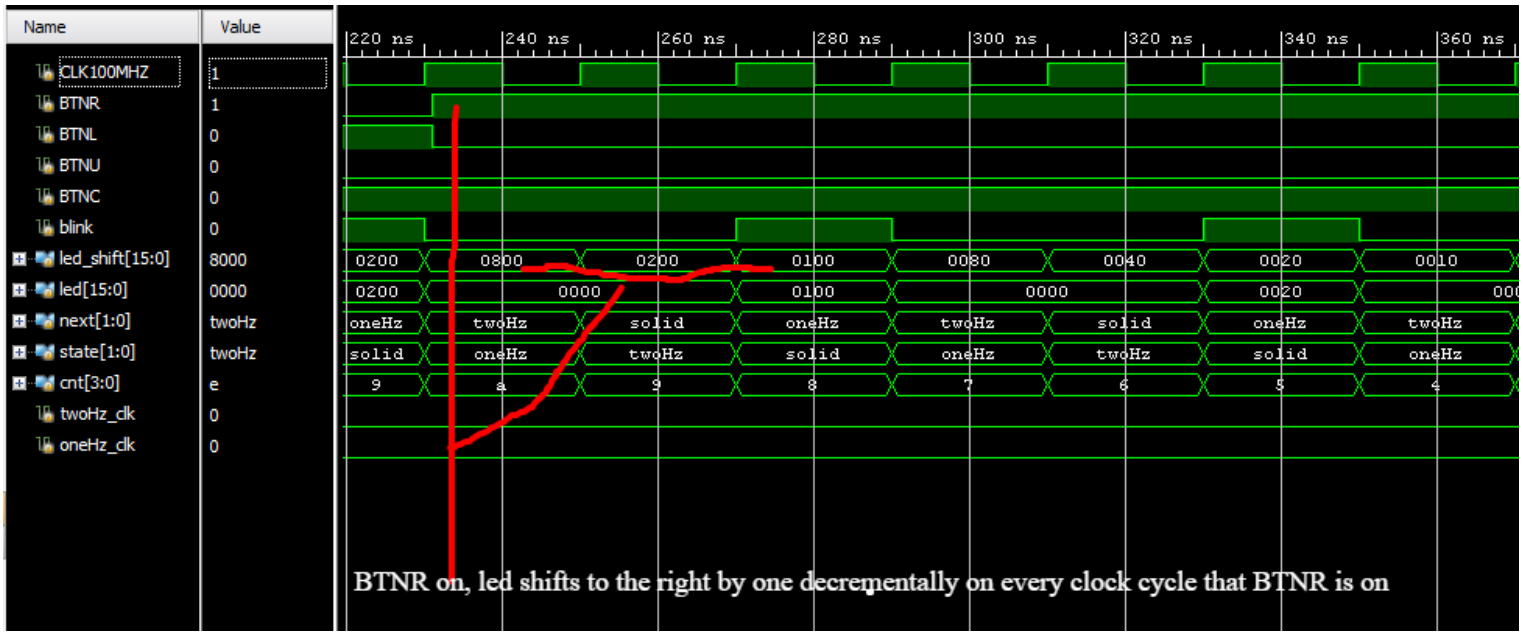
Testbench

```
`timescale 1ns / 1ps
module Nexys4Test();
    // inputs
    logic CLK100MHZ;
    logic BTNR;
    logic BTNL;
    logic BTNU;
    logic BTNC;
    //output
    logic  [15:0] led;

    // instantiate device under verification (nexys4DDR)
    nexys4DDR  DUV(.CLK100MHZ, .BTNC, .BTNU, .BTNL, .BTNR, .led );

    // clock generator with period=20 time units
    always begin
        CLK100MHZ = 0;  #10;
        CLK100MHZ = 1;  #10;
     end

    //stimulus
      initial begin
            // Initialize Inputs
            CLK100MHZ= 0;
            BTNU = 1;   //reset
            BTNC = 0;
            BTNL = 0;
            BTNR = 0;
```

```
        // Wait 100 ns for global reset to finish
        #10;
        @(posedge CLK100MHZ) #1;
        BTNU = 0;
        BTNC = 1;   //change in flash mode
        BTNL = 1;   //lsb led move left 1
        BTNR = 0;
        repeat(10) @(posedge CLK100MHZ) #1;
    BTNL = 0;    //lsb led move left 1
    BTNR = 1;
    repeat(10) @(posedge CLK100MHZ) #1;
        BTNC = 0;
        repeat(10) @(posedge CLK100MHZ) #1;
        BTNU = 1;   //reset led
        repeat(5) @(posedge CLK100MHZ) #1;
         $stop;
end

endmodule
```

Waveforms annotated





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

Making lab better – the first lab went well.