ECE3300L.E01

Lab 8: RGB LED PWM Controller

(Nexys A7)

Priyanka Ravinder and Raj Gokidi

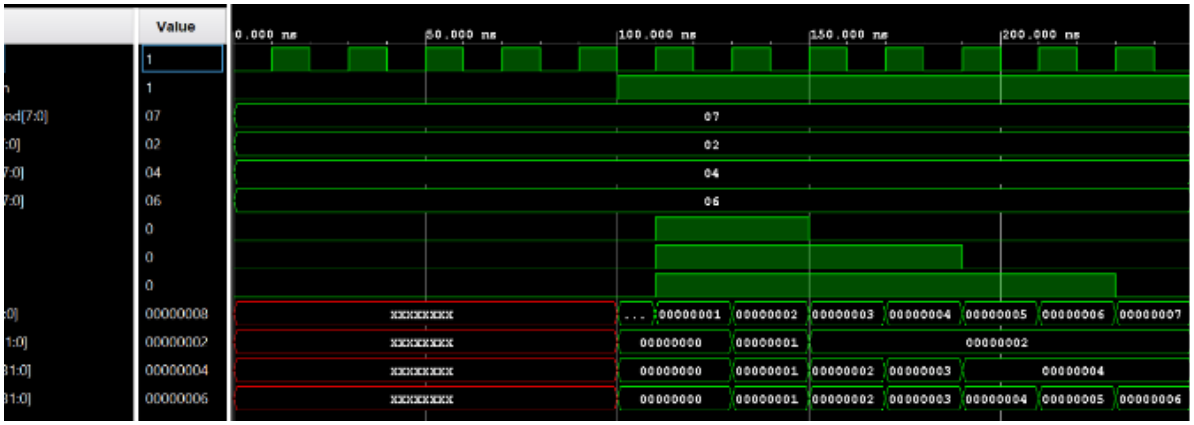
08/14/2025

Introduction:

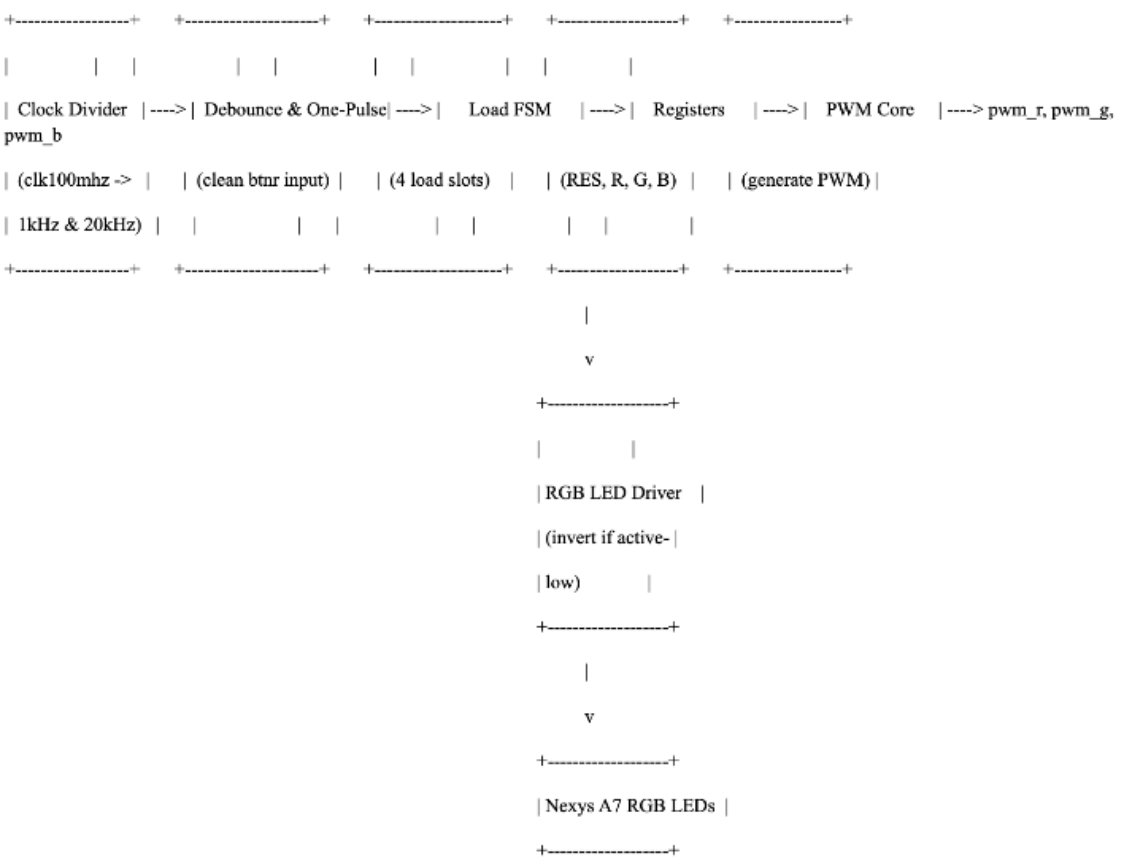
The purpose of this lab is to design and implement a hardware-controlled RGB LED controller on the Nexys A7 FPGA board. The pulse width modulation was used to control the brightness of the RGB LED, which allows for the larger range of brightness. This lab also utilizes several verilog modules, including a clock divider to generate clock frequencies, the debounce module for the circuit to smooth out noise, and the PWM core to generate the PWM signals. The final circuit is required to use all of these components, then provide a range of functionality for the RGB LED controller.

Waveforms:

pwm_core_tb:



Block Diagram:



Code:

clock_divider_fixed.v

```
`timescale 1ns / 1ps
module clock_divider_fixed #(
   parameter integer INPUT_HZ = 100_000_000,
    parameter integer TICK1_HZ = 1_000,
    parameter integer PWM_HZ   = 20_000
)(
    input  wire clk_in,
    input  wire rst_n,
```

```
    output reg  clk_1k,
    output reg  clk_pwm
);
    localparam integer DIV1H  = (INPUT_HZ/TICK1_HZ)/2;
    localparam integer DIVPMH = (INPUT_HZ/PWM_HZ)/2;
    reg [$clog2(DIV1H):0]  c1;
    reg [$clog2(DIVPMH):0] c2;

    always @(posedge clk_in or negedge rst_n) begin
        if (!rst_n) begin
            c1 <= 0; clk_1k <= 0;
            c2 <= 0; clk_pwm <= 0;
        end else begin
            if (c1 == DIV1H-1) begin c1 <= 0; clk_1k <= ~clk_1k; end else c1 <= c1 + 1;
            if (c2 == DIVPMH-1) begin c2 <= 0; clk_pwm <= ~clk_pwm; end else c2 <= c2 + 1;
        end
    end
endmodule
```

debounce_onepulse.v

```
`timescale 1ns / 1ps
module debounce_onepulse #(
   parameter integer STABLE_TICKS = 20
)(
```

```
    input  wire clk,
    input  wire rst_n,
    input  wire din,
    output reg  pulse
);
    reg d0, d1;
    reg stable, stable_q;
    reg [$clog2(STABLE_TICKS+1)-1:0] cnt;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin d0 <= 0; d1 <= 0; end
        else begin d0 <= din; d1 <= d0; end
    end
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin cnt <= 0; stable <= 0; end
        else if (d1 != stable) begin
            if (cnt == STABLE_TICKS) begin stable <= d1; cnt <= 0; end
            else cnt <= cnt + 1;
        end else cnt <= 0;
    end
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin stable_q <= 0; pulse <= 0; end
        else begin pulse <= (~stable_q) & stable; stable_q <= stable; end
    end
```

```
endmodule
```

load_fsm.v

```
`timescale 1ns / 1ps
module load_fsm(
    input  wire       clk,
    input  wire       rst_n,
    input  wire       load_pulse,
    output reg  [1:0] slot,
    output wire [3:0] slot_onehot,
    output reg        wr_res, wr_r, wr_g, wr_b
);
    assign slot_onehot = 4'b0001 << slot;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) slot <= 2'd0; else if (load_pulse) slot <= slot + 2'd1;
    end

    always @* begin
        wr_res = 0; wr_r = 0; wr_g = 0; wr_b = 0;
        case (slot)
            2'd0: wr_res = load_pulse;
            2'd1: wr_r   = load_pulse;
            2'd2: wr_g   = load_pulse;
            2'd3: wr_b   = load_pulse;
```

```
        endcase
    end
endmodule
```

pwm_core.v

```
`timescale 1ns / 1ps
module pwm_core(
    input  wire       clk,
    input  wire       rst_n,
    input  wire [7:0] period,
    input  wire [7:0] duty_r, duty_g, duty_b,
    output reg        pwm_r, pwm_g, pwm_b
);
    wire [8:0] eff_period = {1'b0, period} + 9'd1;

    function [8:0] clamp9(input [7:0] d);
        begin
            clamp9 = ({1'b0,d} >= eff_period) ? (eff_period - 9'd1) : {1'b0,d};
        end
    endfunction

    reg [8:0] cnt;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) cnt <= 0;
```

```
        else if (cnt == eff_period - 1) cnt <= 0;
        else cnt <= cnt + 1;
    end

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) {pwm_r, pwm_g, pwm_b} <= 0;
        else begin
            pwm_r <= (cnt < clamp9(duty_r));
            pwm_g <= (cnt < clamp9(duty_g));
            pwm_b <= (cnt < clamp9(duty_b));
        end
    end
endmodule
```

rgb_led_driver.v

```
`timescale 1ns / 1ps
module rgb_led_driver #(
    parameter ACTIVE_LOW = 1
)(
    input  wire pwm_r, pwm_g, pwm_b,
    output wire led_r, led_g, led_b
);
    generate
        if (ACTIVE_LOW) begin
```

```
            assign led_r = ~pwm_r;
            assign led_g = ~pwm_g;
            assign led_b = ~pwm_b;
        end else begin
            assign led_r = pwm_r;
            assign led_g = pwm_g;
            assign led_b = pwm_b;
        end
    endgenerate
endmodule
```

top_lab8.v

```
`timescale 1ns / 1ps
module top_lab8(
    input  wire        clk100mhz,
    input  wire        btnc,
    input  wire        btnr,
    input  wire [7:0]  sw,
    output wire [3:0]  led,
    output wire        rgb_r, rgb_g, rgb_b
);

    wire rst_n = ~btnc;
```

```
wire clk_1k, clk_pwm;
clock_divider_fixed u_div(
    .clk_in(clk100mhz), .rst_n(rst_n), .clk_1k(clk_1k), .clk_pwm(clk_pwm)
);

wire load_pulse;
debounce_onepulse #(.STABLE_TICKS(20)) u_db(
    .clk(clk_1k), .rst_n(rst_n), .din(btnr), .pulse(load_pulse)
);

wire [1:0] slot;
wire [3:0] slot_oh;
wire wr_res, wr_r, wr_g, wr_b;
load_fsm u_fsm(
    .clk(clk_1k), .rst_n(rst_n), .load_pulse(load_pulse),
    .slot(slot), .slot_onehot(slot_oh),
    .wr_res(wr_res), .wr_r(wr_r), .wr_g(wr_g), .wr_b(wr_b)
);
assign led = slot_oh;

reg [7:0] reg_res, reg_r, reg_g, reg_b;
always @(posedge clk_1k or negedge rst_n) begin
    if (!rst_n) begin
```

```
        reg_res <= 8'd63;
        reg_r   <= 8'd0;
        reg_g   <= 8'd0;
        reg_b   <= 8'd0;
    end else begin
        if (wr_res) reg_res <= sw;
        if (wr_r)   reg_r   <= sw;
        if (wr_g)   reg_g   <= sw;
        if (wr_b)   reg_b   <= sw;
    end
end
reg [7:0] res_q1,res_q2,r_q1,r_q2,g_q1,g_q2,b_q1,b_q2;
always @(posedge clk_pwm or negedge rst_n) begin
    if (!rst_n) begin
        res_q1<=0; res_q2<=0;
        r_q1<=0;   r_q2<=0;
        g_q1<=0;   g_q2<=0;
        b_q1<=0;   b_q2<=0;
    end else begin
        res_q1<=reg_res; res_q2<=res_q1;
        r_q1  <=reg_r;   r_q2  <=r_q1;
        g_q1  <=reg_g;   g_q2  <=g_q1;
        b_q1  <=reg_b;   b_q2  <=b_q1;
```

```
        end
    end

    wire pwm_r, pwm_g, pwm_b;
    pwm_core u_pwm(
        .clk(clk_pwm), .rst_n(rst_n),
        .period(res_q2), .duty_r(r_q2), .duty_g(g_q2), .duty_b(b_q2),
        .pwm_r(pwm_r), .pwm_g(pwm_g), .pwm_b(pwm_b)
    );

    rgb_led_driver #(.ACTIVE_LOW(1)) u_led(
        .pwm_r(pwm_r), .pwm_g(pwm_g), .pwm_b(pwm_b),
        .led_r(rgb_r), .led_g(rgb_g), .led_b(rgb_b)
    );
endmodule
```

pwm_coretb.v

```
`timescale 1ns / 1ps
module pwm_core_tb;
reg clk = 0, rst_n = 0;
always #10 clk = ~clk;
reg [7:0] period = 8'd7;
reg [7:0] dr = 8'd2, dg = 8'd4, db = 8'd6;
wire pr, pg, pb;
```

```
pwm_core dut(.clk(clk), .rst_n(rst_n), .period(period),
.duty_r(dr), .duty_g(dg), .duty_b(db),
.pwm_r(pr), .pwm_g(pg), .pwm_b(pb));
integer i, hr, hg, hb;
initial begin
#100 rst_n = 1;
hr = 0; hg = 0; hb = 0;
for (i = 0; i < 8; i = i + 1) begin
@(posedge clk);
hr = hr + pr; hg = hg + pg; hb = hb + pb;
end
$display("R=%0d/8 G=%0d/8 B=%0d/8 (expected: ~2,4,6)", hr, hg, hb);
$finish;
end
endmodule
```

nexysa7.xdc

```
# Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports {
clk100mhz }]; #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]
## 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 N15   IOSTANDARD LVCMOS33 } [get_ports { rgb_r }]; #IO_L11P_T1_SRCC_14 Sch=led16_r

set_property -dict { PACKAGE_PIN M16   IOSTANDARD LVCMOS33 } [get_ports { rgb_g }]; #IO_L10P_T1_D14_14 Sch=led16_g

set_property -dict { PACKAGE_PIN R12   IOSTANDARD LVCMOS33 } [get_ports { rgb_b }]; #IO_L5P_T0_D06_14 Sch=led16_b

##Buttons

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

set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { btnc_n }]; #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

Contributions:

Priyanka Ravinder: Code and Report

Raj Gokidi: Demo

Conclusion:

Overall, this lab gives insight into the important principles of digital logic design, using a hardware-based project. By using Verilog to code the various modules, we were able to

successfully show the range of the RGB LED's color and brightness. The PWM core was ultimately responsible for this brightness control, whereas the debounce and clock divider was essential in creating reliable inputs from the user. The load FSM was primarily used for cycling through each of the four parameters using a singular button, and this can be seen in the final demo video.