Lab 2

PWM and LED dimmer  
Cory Cook  
Garrett Albers

Contents

[Introduction 1](#_Toc380344527)

[Design 1](#_Toc380344528)

[Main 1](#_Toc380344529)

[Lab2.v 2](#_Toc380344530)

[Pulse Width Modulation 2](#_Toc380344531)

[pwm\_counter.v 2](#_Toc380344532)

[Seven Segment Display Driver 2](#_Toc380344533)

[disp\_hex\_mux.v 3](#_Toc380344534)

[Tests 4](#_Toc380344535)

[Lab 2 Test 4](#_Toc380344536)

[PWM Test 5](#_Toc380344537)

[Conclusion 5](#_Toc380344538)

# Introduction

Pulse width modulation (PWM) is a technique for controlling the switching frequency of a device. The duty cycle associated with the PWM controls the percent “ON” time of the device. The resolution of the PWM is the number of unique duty cycles available. For instance, four-bit resolution of a binary PWM would represent 2^4 = 16 steps between zero and one-hundred percent.

It is possible to control the perceived brightness of a light by controlling the percent “ON” time of the light using PWM.

# Design

## Main

The main module of the application will reference the inputs from the board including the first four switches, the clock, and the reset button. It will also reference the outputs which are the seven segment displays and the seven segment enable output.

The main module will instantiate an instance of the pulse width modulation module passing the clock and switches as inputs. The switches will be mapped to the duty cycle (count\_amount) of the PWM, the clock to the clock (clk\_input). The output of the pwm (counter\_output) will be mapped to a local wire and passed to the display driver.

The main module will instantiate an instance of the display driver module passing the clock, reset, switches, and a constant four bits binary valued at fifteen. The clock (clk) maps to the clock (clk), the reset (reset) maps to the reset (reset), the switches (sw) map to all of the hex inputs (hex0:hex3), the constant (4’b1111) maps to the decimal point input (dp\_in). The output of the display driver (an, sseg) is mapped directly to the corresponding ports on the device.

### Lab2.v

module lab2(input [3:0] sw, input clk, input reset, output [7:0] sseg, output [3:0] an);

wire pwm\_on;

pwm pwm\_counter(.clk\_input(clk), .count\_amount(sw), .counter\_output(pwm\_on));

disp\_hex\_mux display\_hex(.clk(clk), .reset(reset), .hex3(sw), .hex2(sw), .hex1(sw),

.hex0(sw), .pwm\_on(pwm\_on), .dp\_in(4’b1111), .an(an), .sseg(sseg));

endmodule

## Pulse Width Modulation

The pulse width modulation module can be instantiated with different sizes. The size corresponds to the resolution of the pulse width. The size directly affects the bit width of the duty cycle (count\_amount) input.

The PWM accepts a clock (clk\_input) and a duty cycle (count\_amount) as input and outputs a one bit output indicating the state of the pwm.

The PWM module is a Moore-style state device with current state (current\_count) and a next state (next\_count). The next state is one plus the current state. The current state is set to the value of the next state at the positive edge trigger of the clock. The output (counter\_output) is assigned a bit value indicating whether the current state is less than or equal to the duty cycle.

### pwm\_counter.v

module pwm #(parameter SIZE=4)

(input clk\_input, input[SIZE-1:0] count\_amount, output wire counter\_output);

reg [SIZE-1:0] current\_count = 0;

wire [SIZE-1:0] next\_count;

assign counter\_output = current\_count <= count\_amount;

assign next\_count = current\_count + 1;

always @(posedge clk\_input) current\_count <= next\_count;

endmodule

## Seven Segment Display Driver

The seven segment display driver accepts a clock (clk), a reset, a pwm state (pwm\_on), four hex values (hex0:hex3), and a four-bit decimal point indicator. The driver outputs a seven segment display code (sseg) and a seven segment enable code (an) indicating which display is active.

The local parameter N is assigned to set a static timer that selects when a particular display is active.

The driver uses two Moore-style state machines: one drives the counter that determines the timing and the other determines the currently selected hex and decimal point values.

For this example the last three seven segment displays are disabled so they forced high for their enable bits.

The driver outputs the active enable bits and the static display code for the currently active hex value.

### disp\_hex\_mux.v

// Listing 4.15

module disp\_hex\_mux

   (

    input wire clk, reset,

    input wire [3:0] hex3, hex2, hex1, hex0,  // hex digits

    input wire [3:0] dp\_in,             // 4 decimal points

input wire pwm\_on,

    output reg [3:0] an,  // enable 1-out-of-4 asserted low

    output reg [7:0] sseg // led segments

   );

   // constant declaration

   // refreshing rate around 800 Hz (50 MHz/2^16)

   localparam N = 18;

   // internal signal declaration

   reg [N-1:0] q\_reg;

   wire [N-1:0] q\_next;

   reg [3:0] hex\_in;

   reg dp;

   // N-bit counter

   // register

   always @(posedge clk, posedge reset)

      if (reset)

         q\_reg <= 0;

      else

         q\_reg <= q\_next;

   // next-state logic

   assign q\_next = q\_reg + 1;

   // 2 MSBs of counter to control 4-to-1 multiplexing

   // and to generate active-low enable signal

   always @\*

      case (q\_reg[N-1:N-2])

         2'b00:

            begin

if(pwm\_on) begin

an =  4'b1110;

end

else begin

an = 4'b1111;

end

               hex\_in = hex0;

               dp = dp\_in[0];

            end

         2'b01:

            begin

               an =  4'b1111; //Set to off for lab 2

               hex\_in = hex1;

               dp = dp\_in[1];

            end

         2'b10:

            begin

               an =  4'b1111;  //Set to off for lab 2

               hex\_in = hex2;

               dp = dp\_in[2];

            end

         default:

            begin

               an =  4'b1111;  //Set to off for lab 2

               hex\_in = hex3;

               dp = dp\_in[3];

            end

       endcase

   // hex to seven-segment led display

   always @\*

   begin

      case(hex\_in)

         4'h0: sseg[6:0] = 7'b0000001;

         4'h1: sseg[6:0] = 7'b1001111;

         4'h2: sseg[6:0] = 7'b0010010;

         4'h3: sseg[6:0] = 7'b0000110;

         4'h4: sseg[6:0] = 7'b1001100;

         4'h5: sseg[6:0] = 7'b0100100;

         4'h6: sseg[6:0] = 7'b0100000;

         4'h7: sseg[6:0] = 7'b0001111;

         4'h8: sseg[6:0] = 7'b0000000;

         4'h9: sseg[6:0] = 7'b0000100;

         4'ha: sseg[6:0] = 7'b0001000;

         4'hb: sseg[6:0] = 7'b1100000;

         4'hc: sseg[6:0] = 7'b0110001;

         4'hd: sseg[6:0] = 7'b1000010;

         4'he: sseg[6:0] = 7'b0110000;

         default: sseg[6:0] = 7'b0111000;  //4'hf

     endcase

     sseg[7] = dp;

   end

endmodule

# Tests

## Lab 2 Test

Figure 1 shows the test results of the main module. The clock (clk) alternates every two nanoseconds. The enable and disable time is determined by the PWM using the switches (sw) value as the duty cycle. The output value (sseg) is the display code corresponding with the switches value. In the test bench the switch value increments every fifty cycles. The cycle parameter is specific to the test bench.

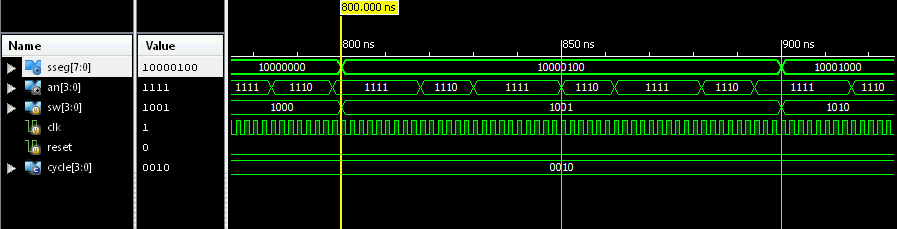


Figure 1: Lab2.v test

## PWM Test

Figure 2 shows the test results for the PWM. The duty cycle (count\_amount) increases once every 32 clock cycles. The clock cycle is set to two nanoseconds for the purposes of this test bench.

You can see that as the duty cycle increases, the active time for the pwm output (counter\_output) increases.

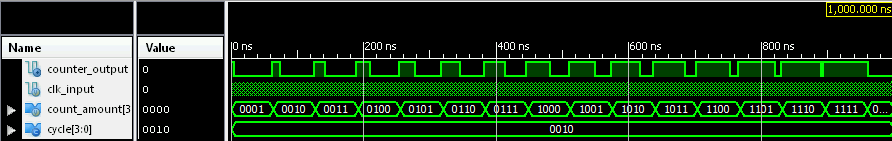


Figure 2: pwm.v test

# Conclusion

Pulse width modulation allows for controlled power output from digital logic control. Then technique can be utilized in cases where a physical device needs to be controlled by a digital circuit. Knowing how PWM is implemented will allow us to recreate or properly use this technology in the future.