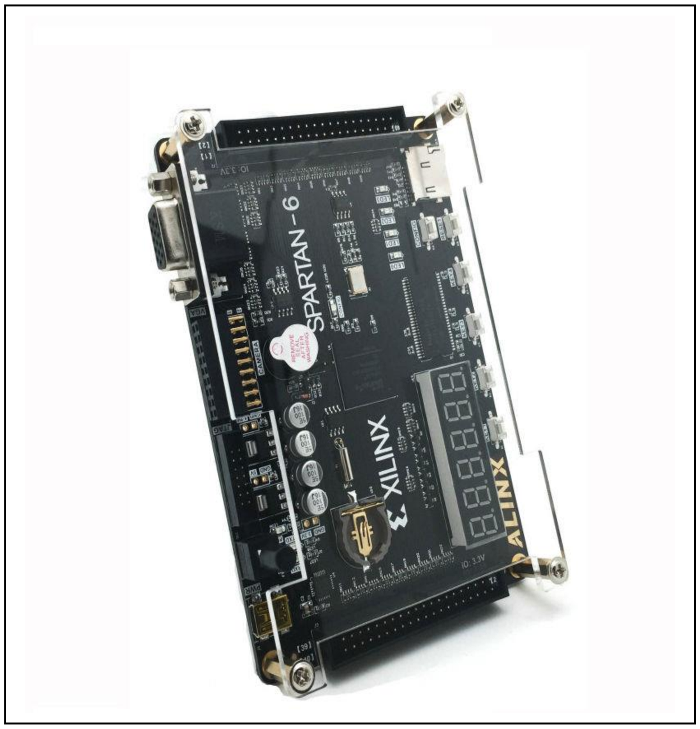
Projects we have completed demonstrate what we know - future projects decide what we will learn.

-Dr. Mohsin Tiwana

FPGA Based LED Brightness Controller Using PWM (Pulse Width Modulation)



COMSATS Institute of Information & Technology, Islamabad

Department of Electrical Engineering.

FPGA Based Brightness Controller Using PWM

Faculty Advisor:
Dr. Armaghan Mohsin
Assistant Professor, CIIT Islamabad

By

Irfan Danish	[CIIT/FA15-BEL-012/ISB]
Hammad Munir	[CIIT/FA15-BEL-034/ISB]

Submitted in partial fulfillment of the 'Digital Logic Design [EEE241]' course for Bachelor of Science in Electronics Engineering

May 30, 2017

Acknowledgments

WE WOULD LIKE TO THANK OUR INSTRUCTOR,
DR. ARMAGHAN MOHSIN,
FOR ALL HIS ADVICE AND SUPPORT IN THE ACCOMPLISHMENT OF THIS PROJECT.

Summary / Abstract

- This project was aimed to implement PWM using Verilog Hardware Descriptive Language.
- It was requested on May 05, 2017.
- Submitted in partial fulfillment of the course, Digital Logic Design.
- Project was prepared by:
  Irfan Danish [CIIT/FA15-BEL-012/ISB]
  Hammad Munir [CIIT/FA15-BEL-034/ISB]

- LED Brightness controller based on PWM (Pulse Width Modulation).
- Spartan 6 XC6SLX9 used to implement the project.
- ModelSim SE 5.7f is used for Programming & Simulation.
- Xilinx ISE 12.2 is used to implement the code on FPGA.
- Major task was to produce capricious duty cycles.
- There are total 12 Verilog programming modules for this project.
- 11 modules consisting 380 lines of Verilog code for design.
- 1 module consisting 200 lines of Verilog code for Test Bench.
- The Source Code can be accessed by following link.
  https://github.com/irfandanishdani/FPGA-Based-LED-Brightness-Controller-Using-PWM-or-PWM-Genetator-

Section **Page**

1. Introduction:

A. WHAT IS FPGA?

A field-programmable gate array (FPGA) is an integrated circuit (IC) that can be programmed in the field after manufacture. FPGAs are similar in principle to, but have vastly wider potential application than, programmable read-only memory (PROM) chips. FPGAs are used by engineers in the design of specialized ICs that can later be produced hard-wired in large quantities for distribution to computer manufacturers and end users. Ultimately, FPGAs might allow computer users to tailor microprocessors to meet their own individual needs.

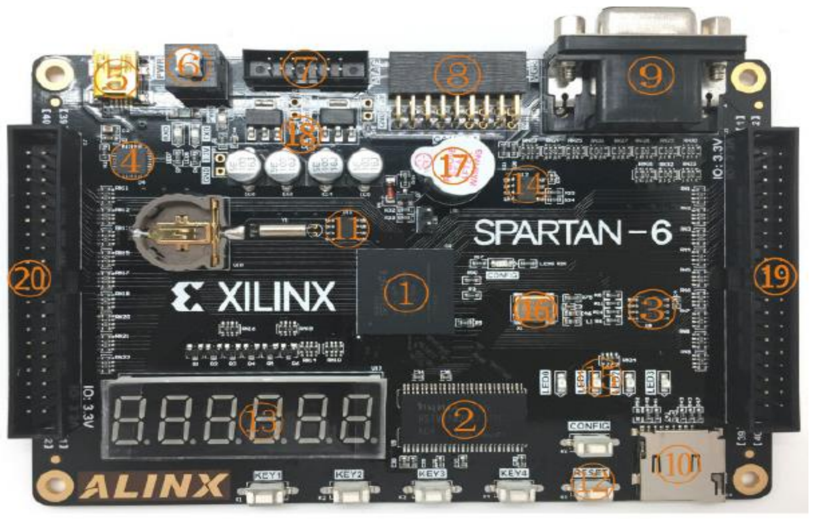


Fig1.0 Spartan 6

Tab1.Baord Source

1	Xilinx XC6SLX9	2	256Mbit SDRAM
3	FLASH 16Mbit	4	USB to UART cp2102
5	USB/USB Power	6	Power Switch
7	JTAG Port	8	Camera Port
9	VGA Port	10	SD card
11	RTC DS1302	12	Reset and 4 User Buttons
13	6-digit LED Display	14	EEPROM 24LC04
15	4 User LEDs	16	50Mhz System Clock
17	Buzzer	18	Power Circuit
19	40PIN Expansion Port	20	40PIN Expansion Port

The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit

(ASIC). (Circuit diagrams were previously used to specify the configuration, as they were for ASICs, but this is increasingly rare.) A Spartan FPGA from Xilinx FPGAs contain an array of programmable logic blocks, and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together", like many logic gates that can be inter-wired in different configurations. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND & XOR. In most FPGAs, logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

B. PULSE WIDTH MODULATION (PWM):

Pulse-width modulation (PWM), or pulse-duration modulation (PDM), is a modulation technique used to encode a message into a pulsing signal. It is achieved by changing the width of the signal & keeping the time period or frequency same.
Pulse width modulation (PWM) is a powerful technique for controlling analog circuits with digital outputs. PWM is employed in a wide variety of applications, ranging from measurement and communications to power control and conversion.

C. ADVANTAGES:

Pulse Width Modulation (PWM) uses digital signals to control power applications, as well as being fairly easy to convert back to analog with a minimum of hardware.
Analog systems, such as linear power supplies, tend to generate a lot of heat since they are basically variable resistors carrying a lot of current. Digital systems don't generally generate as much heat. Almost all the heat generated by a switching device is during the transition (which is done quickly), while the device is neither on nor off, but in between. This is because power follows the following formula:

$$\mathbf{P = E\ I}, \text{ or}$$

$$\textbf{Watts} = \textbf{Voltage} \times \textbf{Current}$$

If either voltage or current is near zero, then power will be near zero. PWM takes full advantage of this fact. PWM can have many of the characteristics of an analog control system, in that the digital signal can be freewheeling. PWM does not have to capture data, although there are exceptions to this with higher end controllers.

One of the parameters of any square wave is duty cycle. Most square waves are 50%, this is the norm when discussing them, but they don't have to be symmetrical.
The ON time can be varied completely between signal being off to being fully on, 0% to 100%, and all ranges between. Fig 1.1 is giving a detailed working picture of PWM.

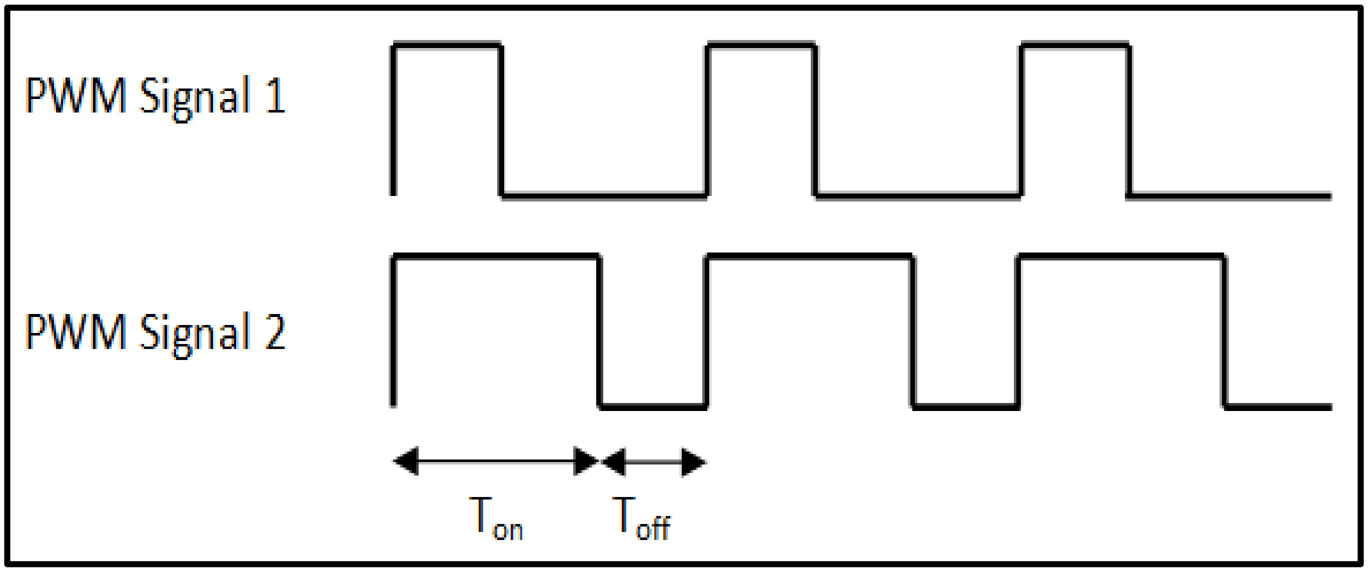


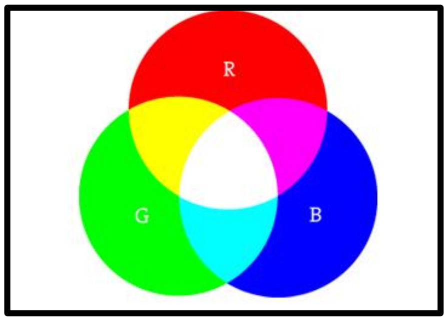
Fig. 1.1

The reason PWM is popular is simple. Many loads, such as resistors, integrate the power into a number matching the percentage. Conversion into its analog equivalent value is straightforward.
LEDs are very nonlinear in their response to current, give an LED half its rated current and you still get more than half the light the LED can produce. With PWM the light level produced by the LED is very linear. Motors, are also very responsive to PWM.

D. APPLICATIONS OF PWM:

We can control the brightness of an LED by adjusting the duty cycle.

With an RGB (red green blue) LED, you can control how much of each of the three colors you want in the mix of color by dimming them with various amounts.



Basics of color mixing

PWM used to control LED brightness

If all three are on in equal amounts, the result will be white light of varying brightness. Blue equally mixed with green will get teal. As slightly more complex example, try turning red fully on, and green 50% duty cycle and blue fully off to get an orange color.

The frequency of the square wave does need to be sufficiently high enough when controlling LEDs to get the proper dimming effect. A 20% duty cycle wave at 1 Hz will be obvious that it's turning on and off to your eyes meanwhile, 20% duty cycle at 100 Hz or above will just look dimmer than fully on. Essentially, the period cannot be too large if you're aiming for a dimming effect with the LEDs.

You can also use pulse width modulation to control the angle of a servo motor attached to something mechanical like a robot arm. Servos have a shaft that turns to specific position based on its control line.
Our servo motors have a range of about 180 degrees.

Frequency/period are specific to controlling a specific servo. A typical servo motor expects to be updated every 20 ms with a pulse between 1 ms and 2 ms, or in other words, between a 5 and 10% duty cycle on a 50 Hz waveform. With a 1.5

ms pulse, the servo motor will be at the natural 90-degree position. With a 1 ms pulse, the servo will be at the 0-degree position, and with a 2 ms pulse, the servo will be at 180 degrees. You can obtain the full range of motion by updating the servo with a value in between.

E. DUTY CYCLE:

Duty cycle is the proportion of time during which a component, device, or system is operated. The duty cycle can be expressed as a ratio or as a percentage. Suppose a disk drive operates for 1 second, then is shut off for 99 seconds, then is run for 1 second again, and so on.

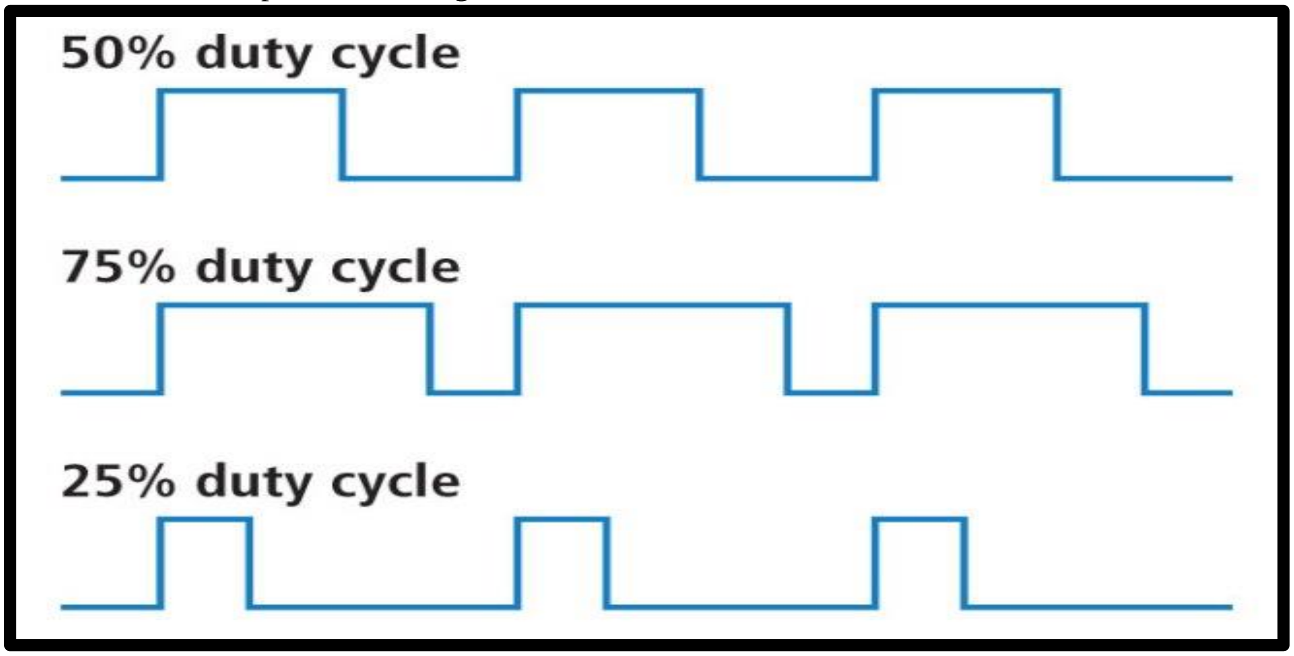
The drive runs for one out of 100 seconds, or 1/100 of the time, and its duty cycle is therefore 1/100, or 1 percent. A duty cycle is the fraction of one period in which a signal or system is active. Duty cycle is commonly expressed as a percentage or a ratio. A period is the time it takes for a signal to complete an on-and-off cycle. As a formula, a duty cycle (%) may be expressed as:

$$D = PW/T \times 100\%$$

Equally, a duty cycle (ratio) may be expressed as:

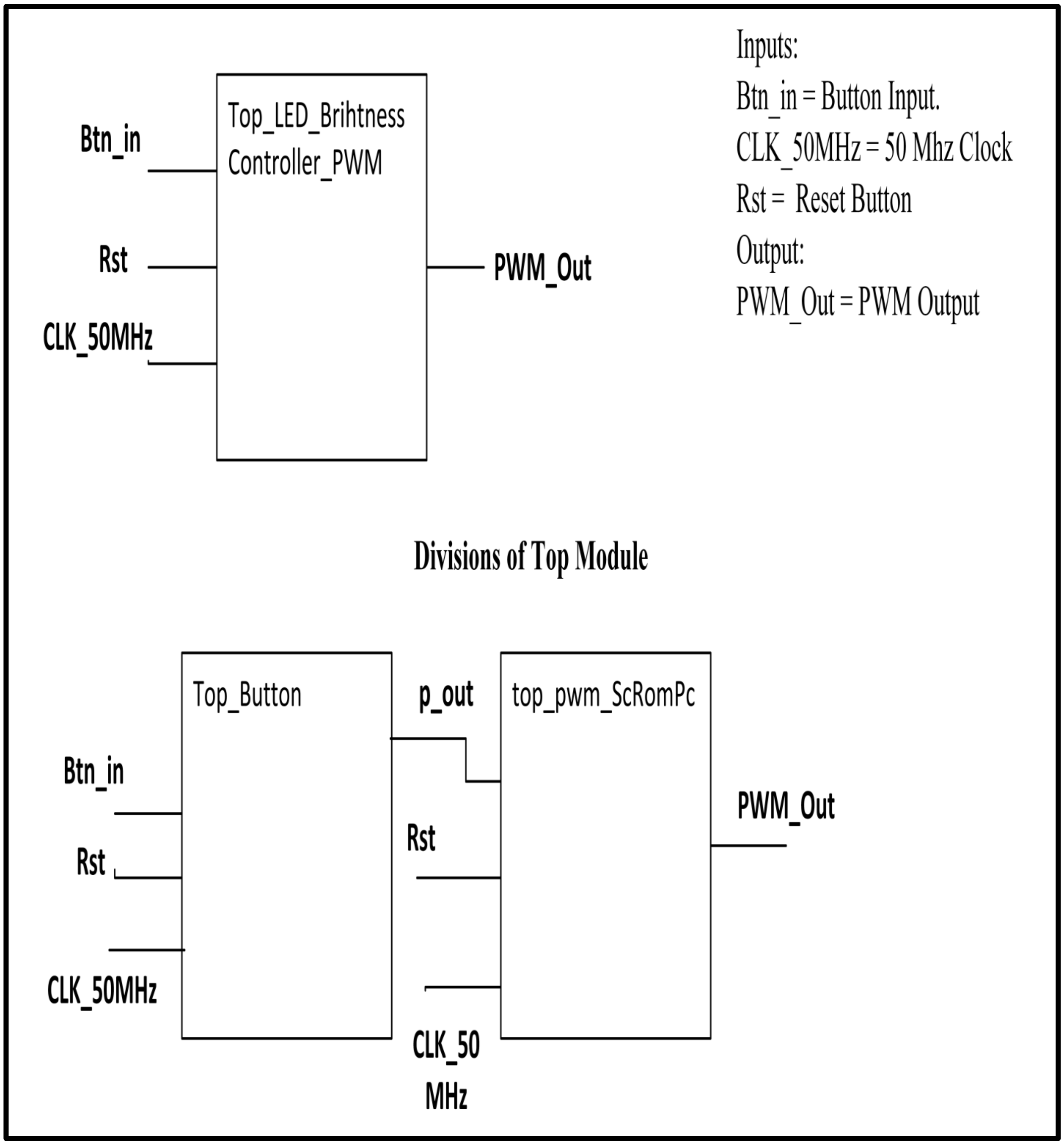
$$D = PW/T$$

Where, D is Duty Cycle. PW is the pulse width (pulse active time), and T is the total period of the signal.

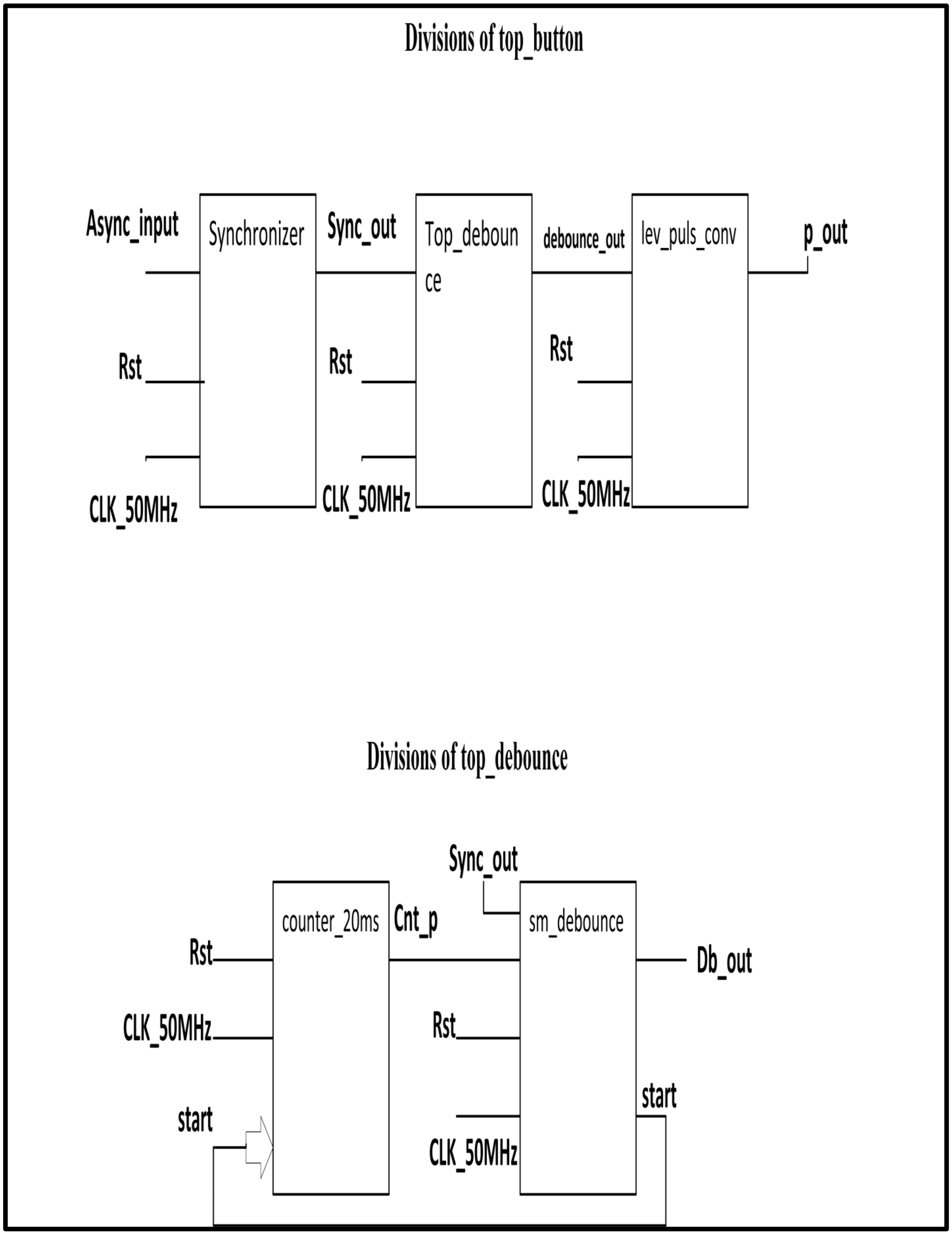


2. **Block Diagrams:**

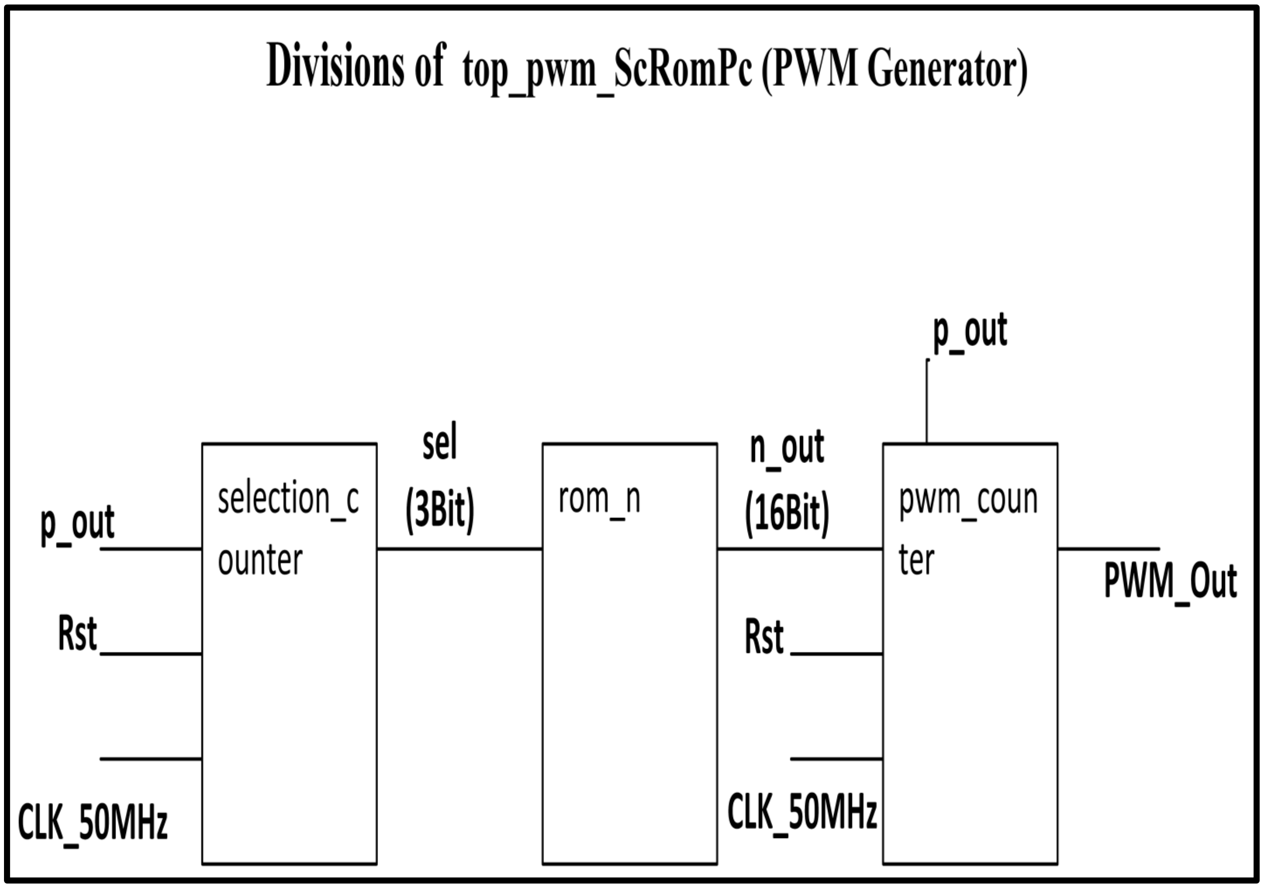
Block diagram of top module and its division:



Block diagram of top button and its divisions:



Divisions of PWM generator:



3. Modules & Working:

There are total elven modules used in this project. Which are divided as:

main_led_brightness_controller

a. Top_button
   i. Synchronizer
   ii. Top_debounce
      1. Counter
      2. Sm_debounce
   iii. Levl_puls_conv

b. Top_pwm_Generator
   i. Selection_counter
   ii. N_rom
   iii. Pwm_Generator

a) Main Module:

Main module has three inputs Clock, Reset & Button & one output PWM. Clock on FPGA have frequency of 50Mhz (Time period of 20 Nano seconds) the frequency of our output must be 1KHz (Time period of 1 mille second).

Calculation of total value for counter:

$$\text{Total} = \frac{50\ MHz}{1\ KHz}$$

$$\text{Total} = \frac{50 \times 10^{6} Hz}{1 \times 10^{3} KHz}$$

Total = 50,000

We have to divide this total into four equal parts which are:

12500, 25,000, 37,500 & 50,000. Output turns on and off so quickly so that eyes take this phenomenon as variance in intensity of LED light.

Main module connects the top modules ‘Top_button’ & ‘top_PWM_Generator’.

b) Top Button:

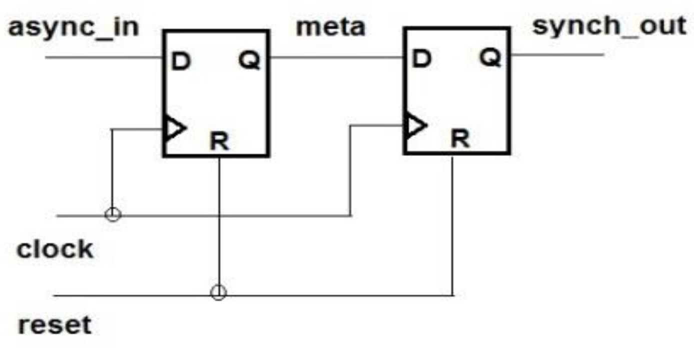
This module connects the ‘synchronizer’, ‘top debounce’ & ‘level pulse converter’.

c) Top PWM Generator:

This module connects ‘Selection Counter’, ‘Rom’ & ‘PWM Generator’.

d) Synchronizer:

As our finger is not synchronized to the clock, so the button is assumed to be an asynchronous signal. If the input changes close to the clock it could violate setup or hold time requirements of the D-Flip Flop (DFF). This can cause the output of DFF to be unpredictable (metastable), even a non-logical level. A DFF going metastable is kind of like a coin landing on its side when you flip it. It should be 0 or 1 but actually, in rare cases, it isn't. The solution of problem is just two DFFs like above. This adds a laitance of two clock cycles to the input signal, but this delay is trivial in your application.



sync_ss

So what our code does it take asynchronous input along with clock and reset inputs. We created two variables to register values at positive edge of clock. First we register value of asynchronous input to first variable and then at the same time value is registered to second variable and at the end it is assigned to variable that we’ve created for our output. In this way we achieve synchronization of input.

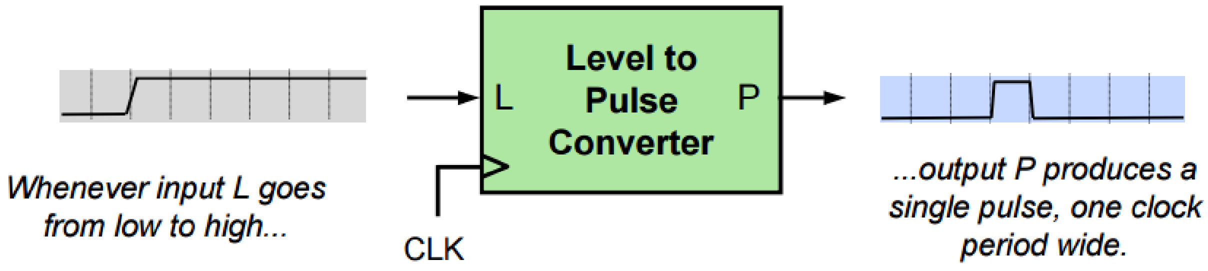
e) Debounce:

When a button is pressed, it may not immediately make a complete connection. In fact, it may make contact on one side - then both and then the other side until it finally settles down. This making and breaking

contact is called bouncing. It is not a manufacturing defect of the button bouncing is implicit in most physical switches.
Bouncing happens in order of milliseconds (5-20 ms) but our system is moving so fast that it will detect a transition between two states every time the button bounces. The solution of the problem is as follows.
Initially set up a count value to zero, Using a timer set up a sampling event with a period (say 20 ms) on the sample event if switch signal high then set 'count=0' set internal switch state released else increment count to a max of 20, end if. If count is 10 then set internal switch state to pressed and then end if.

f) Level to Pulse Converter:

A level-to-pulse converter produces a single cycle pulse each time its input goes high.
It's a synchronous rising-edge detector.


Level to Pulse Converter
L
P
CLK
Whenever input L goes from low to high...
...output P produces a single pulse, one clock period wide.

g) Selection Counter:

In this step we have to decode each input of our system. Every button press must have unique code to change the level of PWM. As we have to generate 5 different duty cycles so the program have to generate only five different codes. So we use a binary counter and take the button press as input.

First Press	000
Second Press	001
Third Press	010
Fourth Press	011
Fifth Press	100

& after fifth press counter should reset to 000.

h) ROM (Time Value Storage):

This module stores different values that is why it is named as ROM. It stores different values of time for which counters have to run. It send's specific value of time when it receives a code from 'Selection Counter'.

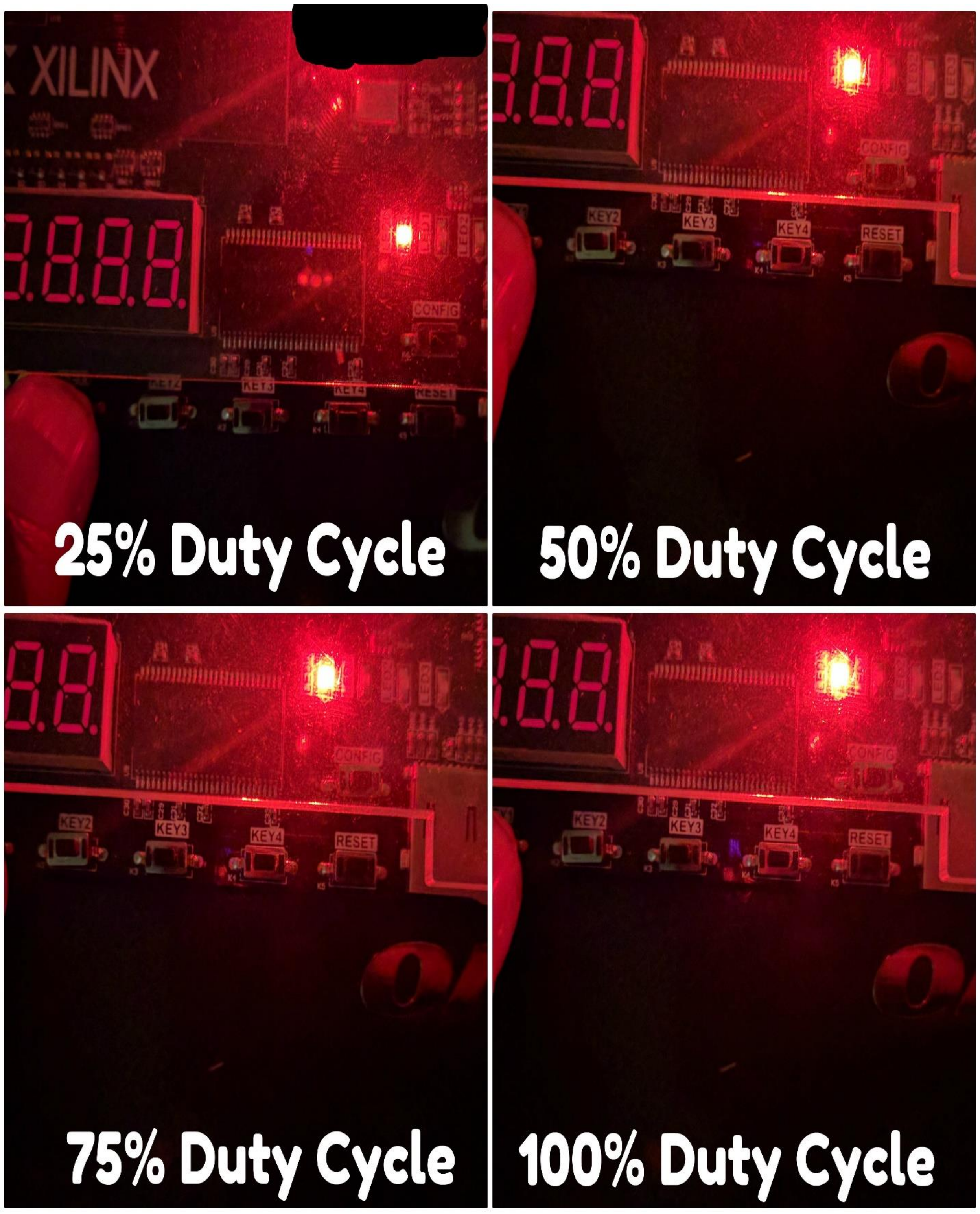
Button Press	**Code**	**Value for PWM Counter (N)**
First Press	000	0
Second Press	001	12,500
Third Press	010	25,000
Fourth Press	011	37,500
Fifth Press	100	50,000

i) PWM Counter:

PWM counter has two counters one counter counts for how much time output remains **High** i.e. **1** & the other counter counts for how much time output remains **Low** i.e. **0.** First performs the counting for the given value which ROM provides and turn the output **High** i.e. **1** for that time & after this second counter perform counting for (Total Value – Given Value) turn the output **Low** i.e. **0**. In this PWM is generated.

High Time	**Low Time**	**% of Duty Cycle**
0	50,000	0 %
12,500	37,500	25 %
25,000	25,000	50 %
37,500	12,500	75 %
50,000	0	100 %

4. Outputs:



[redacted]

5. <u>Bibliography:</u>

i. **Source:** MIT
URL: http://web.mit.edu/6.111/www/s2004/LECTURES/l6.pdf

ii. **Source:** Allaboutcircuits.com
URL: https://www.allaboutcircuits.com/textbook/semiconductors/chpt-11/pulse-width-modulation/
Book: [Vol.3-Semiconductors, Chapter 11-DC Motor Drives, Pulse Width Modulation]

iii. **Source:** electroSome
URL: https://electrosome.com/switch-debouncing/

iv. **Source:** Programming Electronics
URL: https://programmingelectronics.com/tutorial-19-debouncing-a-button-with-arduino-old-version/

v. **Source:** David Fong's ASIC Architecture, Design, Verification and DFT Blog
URL: https://daffy1108.wordpress.com/2014/06/08/synchronizers-for-asynchronous-signals/

vi. **Source:** Techtarget.com
URL: http://whatis.techtarget.com/definition/duty-cycle
Writer: Margaret Rouse; Sep 2005

vii. **Source:** Wikipedia
URL: https://en.wikipedia.org/wiki/Duty_cycle

viii. **Source:** Wikipedia
URL: https://en.wikipedia.org/wiki/Field-programmable_gate_array

ix. **Source:** Techtarget.com
URL: http://whatis.techtarget.com/definition/field-programmable-gate-array-FPGA Writer: Margaret Rouse; Sep 2005

x. **Source:** MIT
URL: http://web.mit.edu/6.111/www/f2007/handouts/L07.pdf