PWM Generator with Variable Duty Cycle Using FPGA

Submission File:

Team A:

Devanshi Chauhan

Jainam Shah

Harsh Maghnani

Siddhpura Hemangi

Table Of Content

Acknowledgement

The success and final outcome of any project or task requires a lot of guidance and assistance from many people and we are extremely privileged to get this all along the completion of our task. All that we have done is only due to such supervision and assistance and we would not forget to thank them. We respect and thank the entire core team, for providing us an opportunity to carry out the task and giving us all support and guidance which made us complete the task duly.

We would like to express our sincere gratitude to the co-founders, Prit Varmora and Ritvik Tiwari for providing us an ingenuitive atmosphere. At last but not the least, we are highly thankful to the Abhiyanta Community for providing us such a platform to give life to our classroom learned skills.

1. Overview

Pulse Width Modulation (PWM) is a very popular modulation technique which is mainly used to control the power delivered to electrical devices such as motors. In this task the team had to generate pulse width signals using Verilog(Hardware Descriptive Language) and realize its hardware using FPGA. A field-programmable gate array (FPGA) is an integrated circuit that can be programmed or reprogrammed to the required functionality or application after manufacturing. Important characteristics of field-programmable gate arrays include lower complexity, higher speed, volume designs and programmable functions.

2. Softwares Used

1. Quartus Verilog
2. Modelsim

3. FPGA

FPGA stands for Field Programmable Gate Array and are essentially a sea of gates that can be reconfigured to build almost any digital circuit that one can imagine. This great flexibility along with the ability to reconfigure the device with different designs at-will makes FPGA a better choice compared to ASICs (Application Specific Integrated Circuit) for a lot of applications. FPGA consists of built-in hard blocks such as memory controllers, high speed communication interfaces, etc. They are fundamentally similar to CPLDs but FPGAs are preferred as CPLDs are very small in size and less capable.

4. Verilog HDL

It is Hardware Description Language(HDL) that describes the hardware of digital systems in textual form. It is used to design a digital network or system such as flip flop, latch, MUX, DEMUX, etc. It is used to describe the behaviour of hardware. One can design any hardware at any level. Simulation of designs was very difficult before fabrication. With the advent of VLSI, it is not possible to verify a complex design with millions of gates on a breadboard. HDLs came into existence to verify the functionality of these circuits. The major levels of abstraction which verilog supports are:

- Behavioral level: The topmost layer of abstraction in the verilog HDL is behavioural modelling. It refers to the behaviour and the nature of the circuitry. It contains the procedural statements controlling the simulation which generally the designer wants to know.
- Register-transfer level: It is a layer of abstraction used in hardware descriptive languages(HDLs), the RTL view is created to view the circuitry and the actual wiring associated with the system is carried out.
- Gate level: Verilog has inbuilt models developed for primitive gates, to implement the lowest level of modules in a design. Various AND, OR, NOT, NAND, NOR, XOR and XNOR which have two or more inputs are supported in verilog.

5. Verilog Modules

A Verilog module is a design unit similar to a black-box, with a specific purpose as engineered by the RTL designer. As shown in Fig.1, it has inputs, outputs and

functions as per its intended design. A simplest Verilog module could be a simple NOT gate whose sole job is to invert the incoming input signal. There is no upper bound on the complexity of the Verilog modules, they can even describe complete processor cores! Verilog deals with digital circuits. A module can be simply represented graphically as a box with a number of ports. The ports can be inputs, outputs or bidirectional. Ports can be single bit or multiple bits in width. The image below represents a module with a few inputs and outputs. The number of inputs and outputs, their width and direction depends solely on the functionality of the module.

Fig.1 Block diagram for verilog module

6. Test Bench

A test bench is an HDL program used for applying stimulus to an HDL design in order to test it and observe its response during simulation. It is a program that verifies the functional correctness of the hardware design.

7. Universal Shift Register

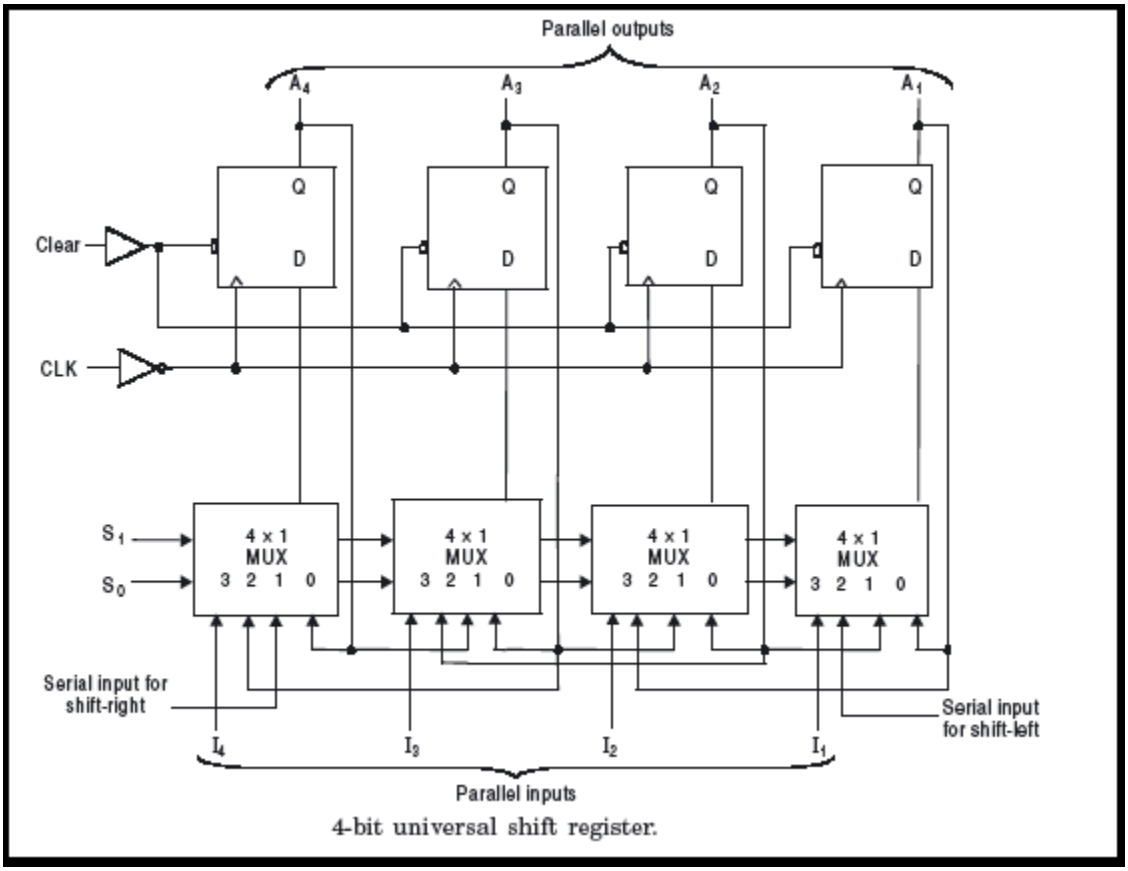

Fig.2 Universal Shift Register block diagram

Flip flops can be used to store a single bit of binary data (1or 0). However, to store multiple bits of data, we need multiple flip flops. A *Register* is a device that is used to store such information. It is a group of flip flops connected in series used to store multiple bits of data. The information stored within these registers can be transferred using *shift registers*. As shown in Fig.2, consider it to be a 4-bit bidirectional universal shift register, the basic purpose that USR serves is its multiple functionality in circuitry, it uses 4 MUX, 4 D-ff, 2 Inverters, 2 input select lines, right in, left in pins, parallel i/o pins, clock, the types it allows are:

- Hold/No change state: When both the select lines(S1 & S0) are at logic low, then it is in hold state where previous data is kept as it is without any change in that, doesn't matter whether it’s input is given serially or parallely.
- Left shift: When select line S1 is at logic low, select line S0 is at logic high and when data input is from right then on every rising clock pulse the data is shifted left and left shift takes place.
- Right shift: When select line S0 is at logic low, select line S1 is at logic high and when data input is from left then on every rising clock pulse the data is shifted right and right shift takes place.
- Parallel in/out: When both the select lines(S1 and S0) are at logic high and data is input parallely(every bit of data on respective pins) then data is received parallely on every clock pulse in the output(every bit of data is received in output respectively).

8. Mod-N Counter:

Counters are sequential logic devices that follow a predetermined sequence of counting states triggered by an external clock (CLK) signal. The number of states or counting sequences through which a particular counter advances before returning to its original first state is called the *modulus* (MOD). In other words, the modulus (or modulo) is the number of states the counter counts and is the dividing number of the counter. Counters are the sequential logic which generally counts up to its maximum value and then resets to 0. MOD-N counter will count up to n and then it’ll be back to 0. It has N number of states. For example, a 2-bit counter that counts from 00_2 to 11_2 in binary, 0 to 3 in decimal, has a modulus value of 4 ($00 \rightarrow 1 \rightarrow 10 \rightarrow 11$, and return to 00); therefore, be called a modulo-4, or mod-4, counter. Note also that it has taken four clock pulses to get from 00 to 11.

- For both inputs at logic low level the input is passed on to output.
- For j=0 and k=1 output is set to logic low
- For j=1 and k=0 output is set to logic high
- For both inputs at logic high level the negation of input is passed on to output(toggled).(other 2 pulse are passed)

9. PWM

PWM stands for Pulse Width Modulation, almost all devices that we see around use a train of pulses of square waves. Pwm signals can be generated through microcontrollers and devices such as 555 TIMER and used to control switches such as FETs. Duty cycle refers to the time for which the pulse is at logic HIGH(on time) to the total time period(on time + off time). As shown in Fig.3, the duty cycle of the signal is set accordingly so that the desired output can be achieved.

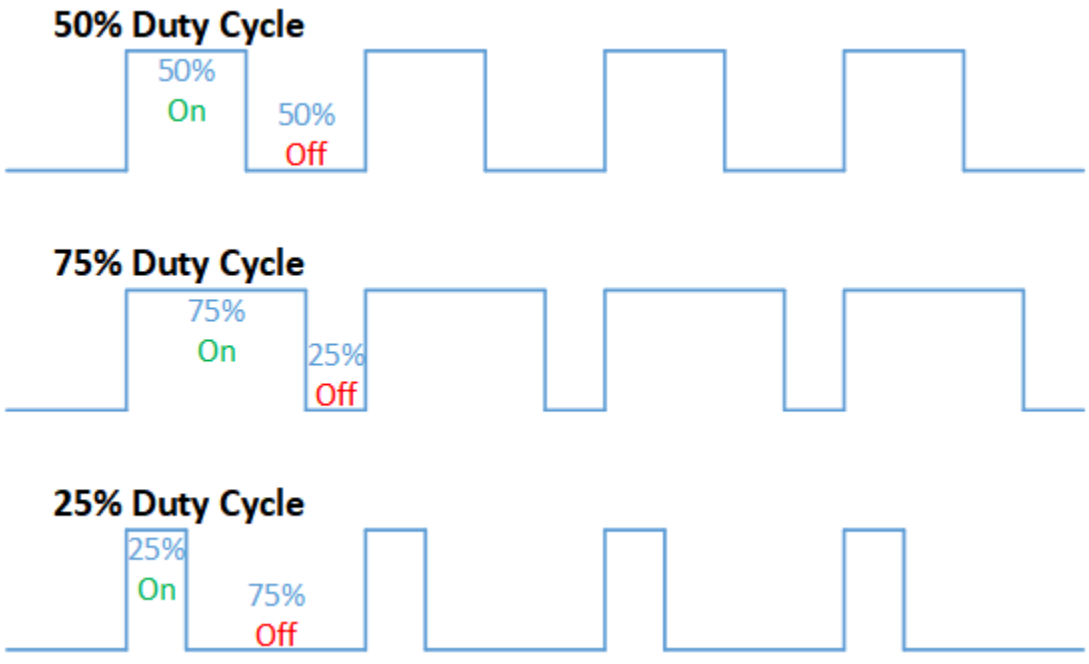

Fig.3 Waveforms with different duty cycles

10. Task 3.1 - Counters and Universal Shift Register

Aim:

1. Design Universal Shift Register using the Verilog HDL and shows its RTL Design and Waveform
2. Design MOD 14 Synchronous up counter using JK Flip Flop by using Verilog HDL and shows its RTL Design and Waveform

Implementation:

1. Implementation of Universal shift register

➢ Inputs:

1. [1:0]se : 2 bit register for selection line to select mode of operation
2. clk : To set synchronous clock
3. left_in : To take input from left side during right shift operation
4. right_in : To take input from right side during left shift operation
5. [3:0]par_in : To take input of 4 bits during parallel mode of operation
6. clr : To clear the output

➢ Output:

1. [4:0]out : 4 bits output register “out”

```
input [1:0]se;
input clk,left_in,right_in,clr;
input [3:0]par_in;
output reg [3:0]out;
```

All modes are selected from the selection input “se” 2 bits register. For different inputs of “se” we will assign output to “out” if no clear signal was there as shown in code below.

```
if(clr)
        out = 4'b0000;

else

begin
        case(se)
        2'b00: begin
                out[0] <= out[0];
                out[1] <= out[1];
                out[2] <= out[2];
                out[3] <= out[3];
        end
        2'b01: begin

                out <= out >> 1;
                out[3] <= left_in;
        end
        2'b10: begin

                out <= out << 1;
                out[0] <= right_in;
        end
        2'b11: begin
                out[0] = par_in[0];
                out[1] = par_in[1];
                out[2] = par_in[2];
                out[3] = par_in[3];
        end
        endcase
end
```

➢ RTL View:

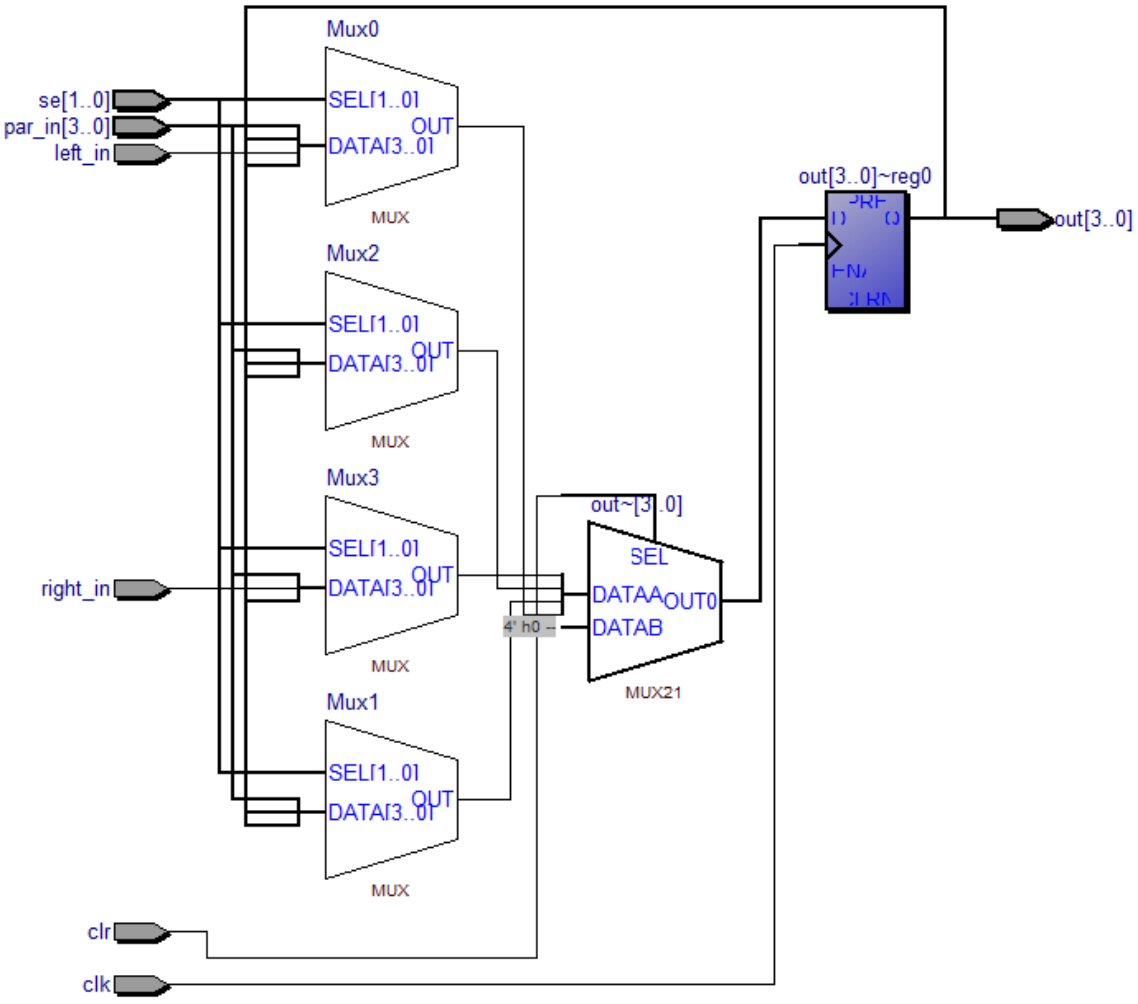

Fig.4 RTL view of Universal Shift Register

➢ Output:

- Right shift operation

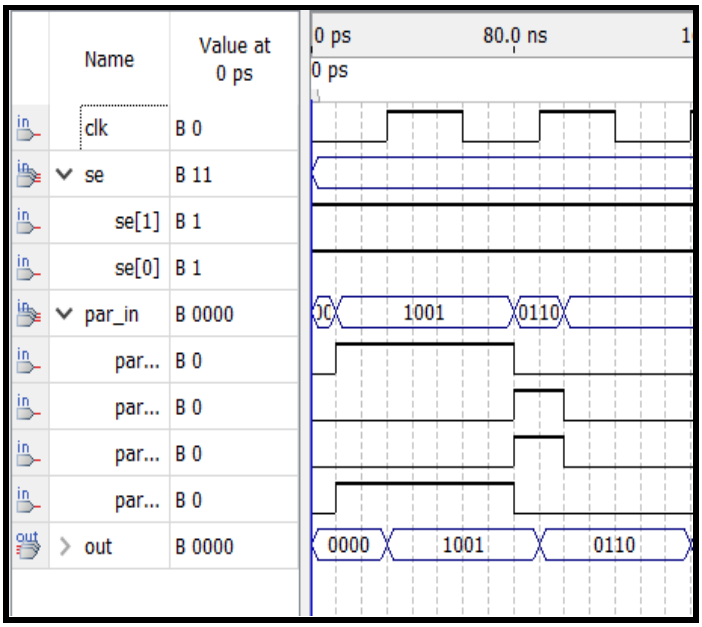

Fig.5 Output waveform for right shift operation

- Left shift operation

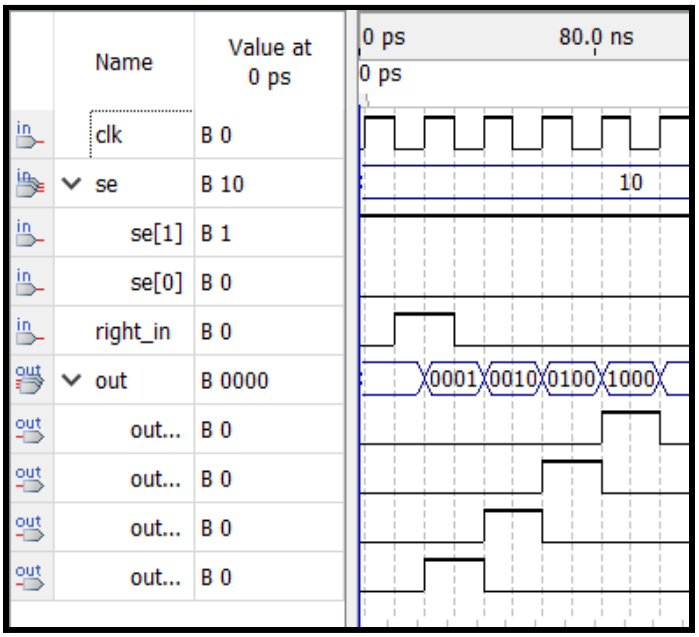

Fig.6 Output waveform for left shift operation

- Parallel load operation

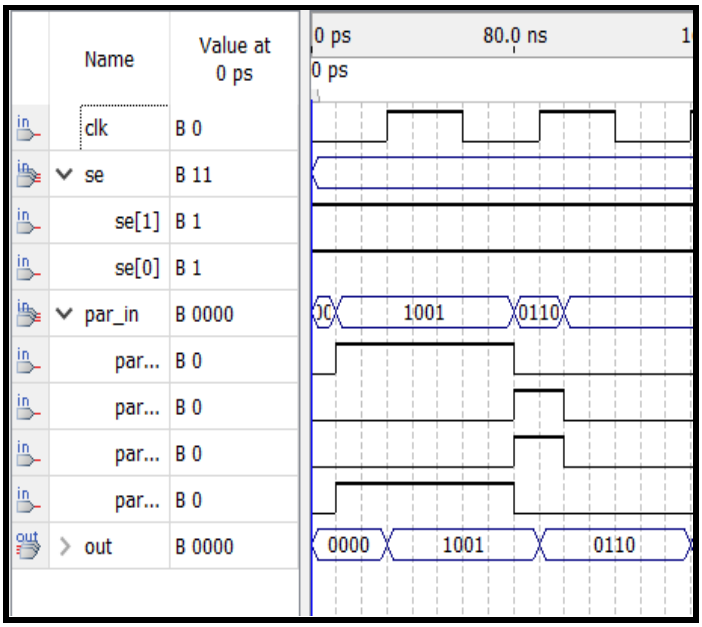

Fig.7 Output waveform for parallel shift operation

2. Implementation of Modulo 14 counter using JK Flip Flop:

➢ Input:

1. w : To start counter
2. clk : To set synchronous clock

➢ Output:

1. q0 : 1st Output of JK Flip Flop(LSB)
2. q1 : 2nd Output of JK Flip Flop
3. q2 : 3nd Output of JK Flip Flop
4. q3 : 4th Output of JK Flip Flop(MSB)

```
module mod14_JK(w,q0,q1,q2,q3,clk);

input w,clk;
output q0,q1,q2,q3;
```

- In this task we have to Implement MOD14 counter using Jk Flip Flop. So we have deine a module named JK_FF in which inputs are J,K,clear,clk,OUT and one output register ”OUT”.

```
module JK_FF(J,K,clk,clear,OUT);

input J,K,clear,clk;
output reg OUT;

always@(posedge clk)
if(~clear)
begin
        if(~J & ~K)
                OUT <= OUT;
        else if(~J & K)
                OUT <= 0;
        else if(J & ~K)
                OUT <= 1;
        else if(J & K)
                OUT <= ~OUT;
end
else
        OUT <= 0;

endmodule
```

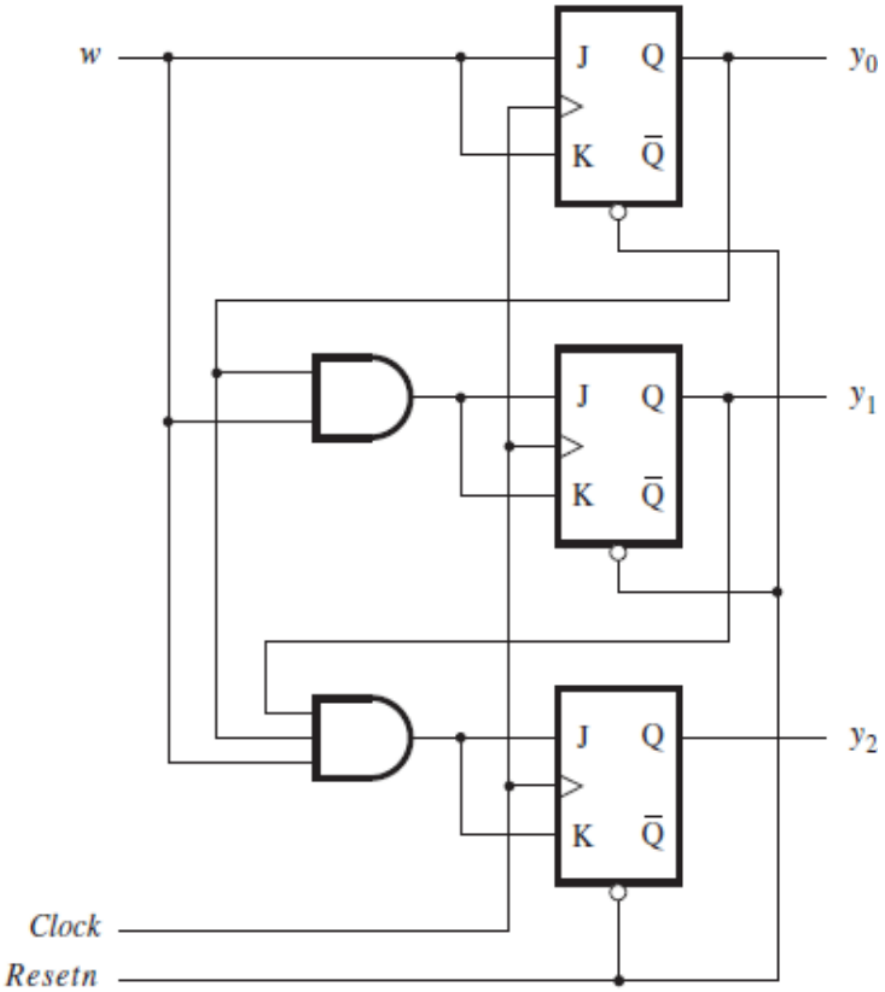

Fig.8 3 bits ip counter

- Now we will use logic as shown in Fig.8 to make a 4 bits up counter.
- Now if we want to make MOD 14 counter then we have to reset our counter after decimal value 13. So that it can count from 0 to 13(14 counts). For that we can write the code given below.

```
assign reset = q0 & q2 & q3;

assign j0 = w;
assign k0 = j0;
JK_FF(j0,k0,clk,reset,out0);
assign q0 = out0;

assign j1 = w & q0;
assign k1 = j1;
JK_FF(j1,k1,clk,reset,out1);
assign q1 = out1;

assign j2 = j1 & q1;
assign k2 = j2;
JK_FF(j2,k2,clk,reset,out2);
assign q2 = out2;

assign j3 = j2 & q2;
assign k3 = j3;
JK_FF(j3,k3,clk,reset,out3);
assign q3 = out3;
```

➢ Mod-14 counter (RTL view)

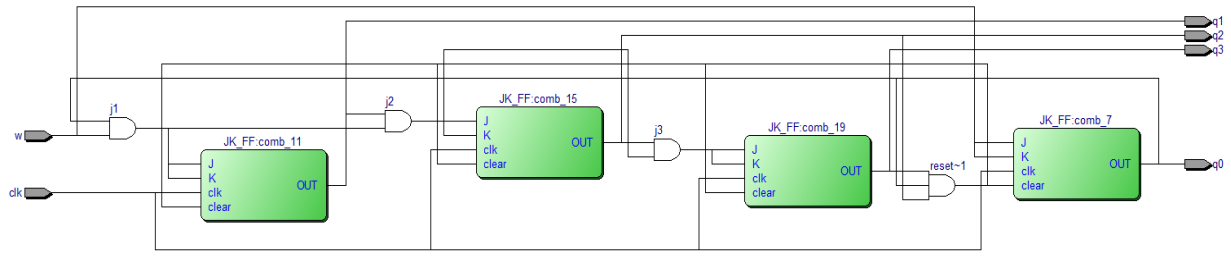

Fig.9 RTL view of MOD14 counter using JK Flip Flop

➢ Mod-14 counter (Output Waveform)

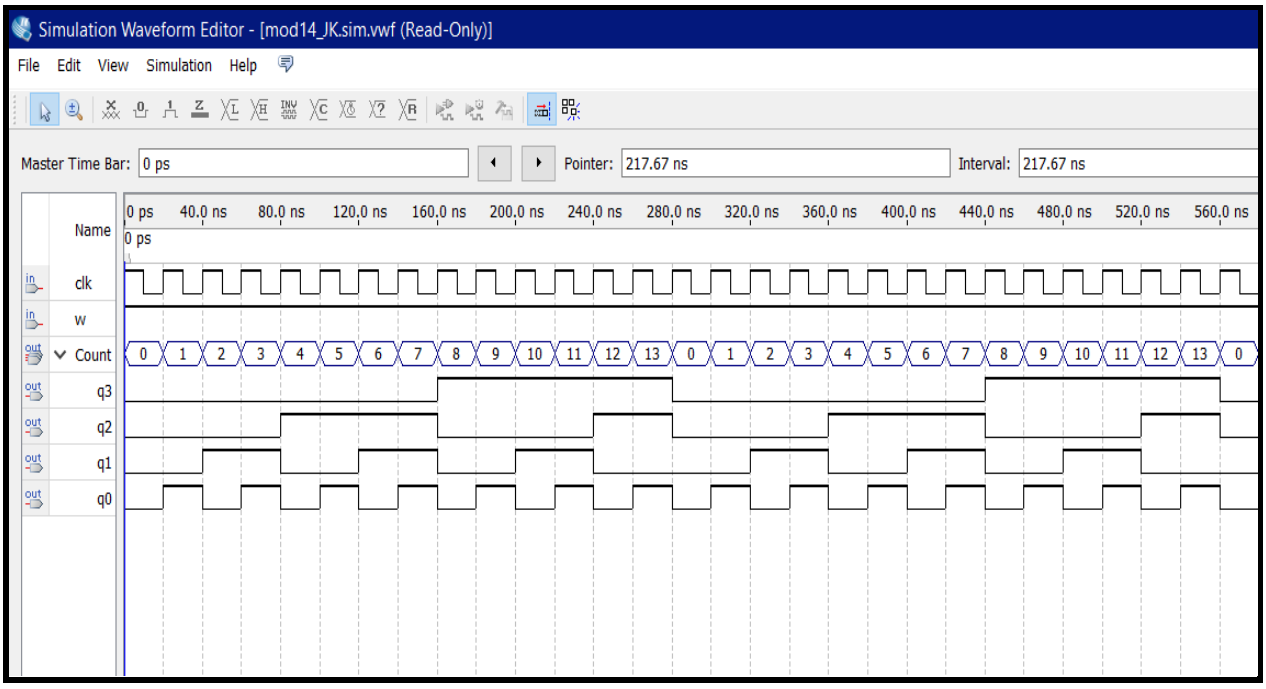

Fig.10

11. Task 3.2 - Designing main verilog code

Aim:

To implement the pwm generator block using verilog such that the frequency of the pulse should be 2 MHz. Duty cycle variation should be in the range of 5%.

Implementation:

- For implementing PWM generator with 5% variation of duty cycle, following inputs needs to be applied:

➢ Inputs:

1. clk : As modulo 100 counter as reference
2. increase : To increase duty cycle by 5% with respect to current duty cycle
3. decrease : To decrease duty cycle by 5% with respect to current duty cycle

➢ Registers used:

1. [7:0]counter : 8 bits register to count from 0 to 100

2. [7:0]duty : 8 bits register for decide value of duty cycle

➢ Output:

1. out : To get PWM with desired duty cycle

```
input clk,increase,decrease;
output out;
reg [7:0]duty = 50;
reg [7:0]counter;
```

- The value of the “counter” is incremented at every positive edge of the “clk”. Variable “duty” is kept as a desirable duty cycle value.
- If the input signal “increase” is in high state then “duty” will be incremented with 5.
- If the input signal “decrease” is in high state then duty will be decremented with 5.
- The value of “out” is assigned as high when the “counter” is less than “duty” otherwise it will be zero.

```
always@(posedge clk)
begin
        if(counter <= 99)
                counter = counter + 1;
        else
                counter = 0;
        if(increase)
                duty = duty + 5;
        if(decrease)
                duty = duty - 5;
end
assign out = (counter<duty)?1:0;
```

- Thus we can generate PWM with facility of increment and decrement of 5% duty cycle.

12. Task 3.3 - Testbench File

Aim:

To create a working testbench file using verilog and run the testbench in Modelsim software and to vary the duty cycles from 0-100% in steps of 5.

Sr No.	Duty Cycle
1.	0%
2.	25%
3.	50%
4.	65%
5.	100%

Table 1. Test cases for testbench

Implementation:

- In this task the aim was to generate a PWM signal with different duty cycles as given in the table. For that 5 wires are declared for getting PWM with the desired duty cycle. Also 5 increase and decrease registers are declared for each wire and one “clk” register for the clock.
- These values are then given to our main or top level module.

```
wire [4:0]out1;
reg clk1 = 0;
reg [4:0]increase1;
reg [4:0]decrease1;

pwm_5_variation m0(clk1,out1[0], increase1[0], decrease1[0]);
pwm_5_variation m1(clk1,out1[1], increase1[1], decrease1[1]);
pwm_5_variation m2(clk1,out1[2], increase1[2], decrease1[2]);
pwm_5_variation m3(clk1,out1[3], increase1[3], decrease1[3]);
pwm_5_variation m4(clk1,out1[4], increase1[4], decrease1[4]);
```

- 500 unit time period is taken for the clock.
- By Default the duty cycle is 50% as given in previous section code.
- So to generate 25% and 0% duty cycles, input “decrease” needs to be set for 5 and 10 clock cycles respectively.
- For generating 65% and 100% duty cycles, input “increase” needs to be set for 3 and 10 clock cycles respectively.
- For that the initial block can be written as shown in code.

```
always #250 clk1 = ~clk1;

initial
  begin

    decrease1[0] = 0;increase1[0] = 0;
    decrease1[1] = 0;increase1[1] = 0;
    decrease1[2] = 0;increase1[2] = 0;
    decrease1[3] = 0;increase1[3] = 0;
    decrease1[4] = 0;increase1[4] = 0;

    #0                                  //initially
    decrease1[0] = 1;
    decrease1[1] = 1;
    increase1[3] = 1;
    increase1[4] = 1;

    #1500                               //After 3 clock pulses(3 pulse are passed)
    increase1[3] = 0;
    #1000                               //After 5 clock pulses(other 2 pulse are passed)
    decrease1[1] = 0;
    #2500                               //After 10 clock pulses(other 5 pulse are passed)
    decrease1[0] = 0;
    increase1[4] = 0;
  End
```

➢ Output:

- Duty cycles of output waveforms are 0%, 25%, 50%, 65%, 100% for waveforms 0,1,2,3,4 respectively as shown in Fig.11.

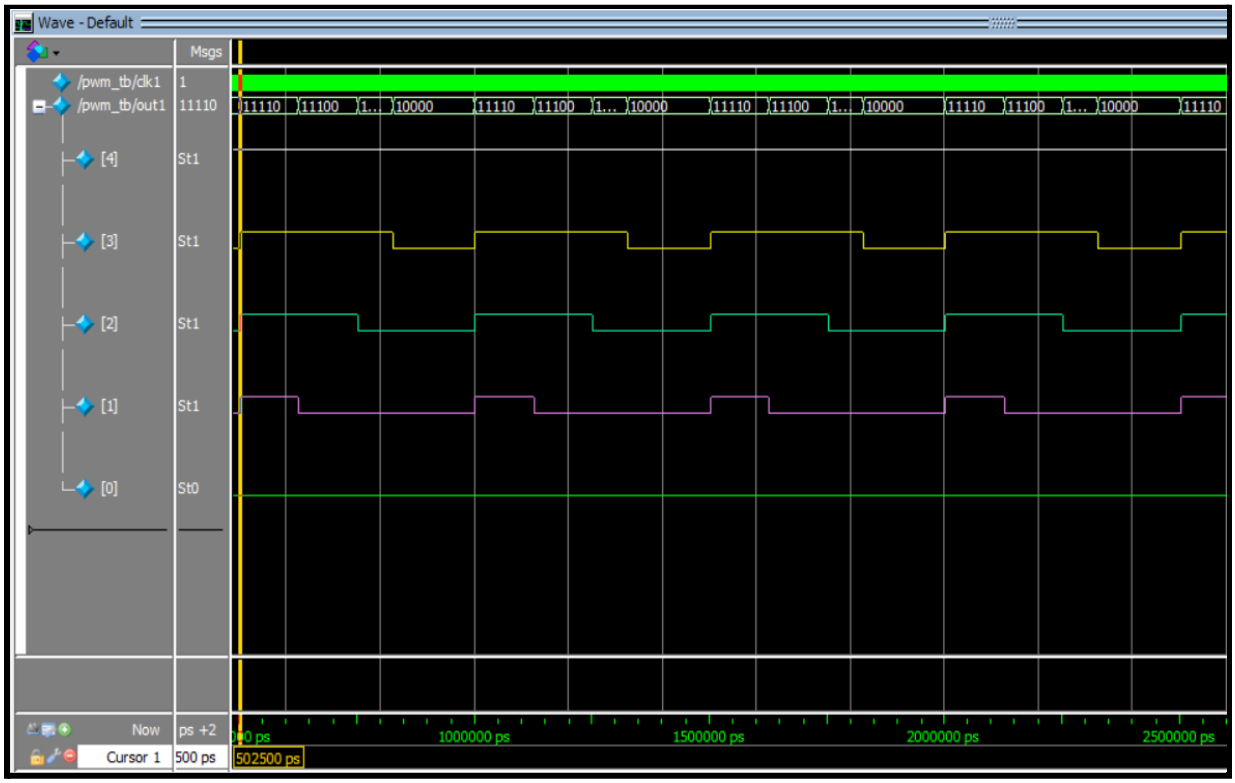

Fig.11 Output Waveform of Testbench

13. Conclusion

In this task we learnt about the designing and testing of electronic circuits using verilog hardware description language(HDL) and testbench. We also learnt about the abstraction levels that verilog uses for combinational and sequential circuitaries. From counter designing we learnt the main difference between synchronous and asynchronous counters. In PWM we observed that as the duty cycle increases, on time delay increases. This observation is very useful in many applications.

14. References

1. Verilog Tutorial for Beginners
2. Learning FPGA And Verilog A Beginner's Guide Part 1 – Introduction
3. Verilog Mod-N Counter
4. Pulse-width modulation
5. verilog hdl basics | youtube
6. Writing Basic Testbench Code in Verilog HDL | ModelSim Tutorial | Verilog Tutorial