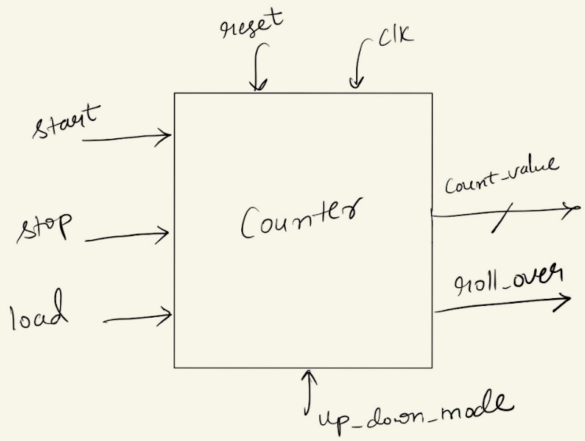
Problem Statement:

Challenge 1:

Design a synchronous decimal counter using Verilog.



Note: all inputs are asynchronous in nature.

It should have following features:

1. On application of 'reset' the count_value must be
   (a) 00)d – in up mode
   (b) 99)d – in down mode

2. On application of 'load' the count_value must be
   (a) 90)d – in up mode
   (b) 10)d – in down mode

3. On application of 'start' the count_value
   (a) Increments – in up mode
   (b) decrements – in down mode

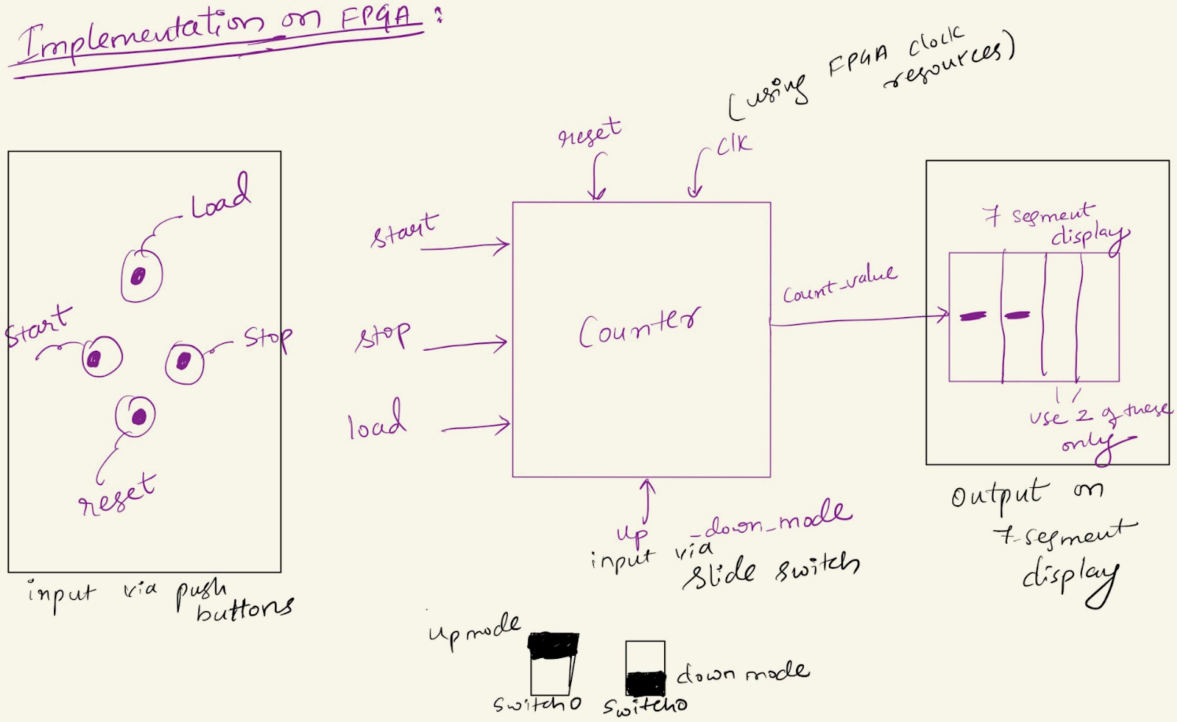
4. On application of 'stop' the count_value halts. On application of start, the counting operation resumes from where it left.

⑤ The counting must happen for every 1s.
[Hint: the clock is 50MHz available from FPGA.
Frequency = $\frac{1}{\text{Time}}$]

⑥ On every rollover the buzzer must get active to indicate the roll_over
[Hint: Keep the rollover signal active for 1s to hear it's voice]
- 99)d to 00)d in up mode
- 00)d to 99)d in down mode

⑦ Use testbench to verify basic functionality
[Hint: Don't include feature ⑤ & ⑥ while using testbench to verify because 1s in simulation will take hours. So use them during the implementation]

Implementation on FPGA:



Explanation:

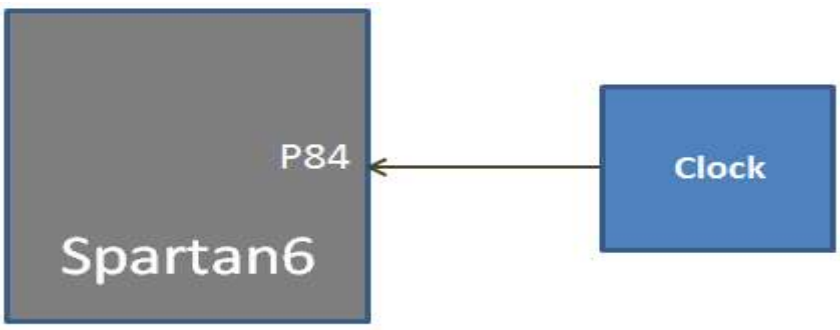
- The goal is to design a synchronous digital counter that counts from 0 to 99 in up mode and 99 to 0 in down mode.

Note the following features:

- The RESET button brings the value to 0 in up mode and 99 in down mode.
- The LOAD button brings the value to 90 in up mode and 10 in down mode.
- The START button begins the counting.
- The STOP button holds the value.
- Functions up and down are assigned to a single switch while reset, load, start and stop are assigned to push buttons.
- When the value crosses 99 in up mode and 0 in down mode, the buzzer rings for a second.
- Counting must also happen for every 1 second.
- Output is displayed on the first 2 segments of the 7 segment display.
- Testbench is used to verify functionality of RESET, LOAD, START, STOP and UP/DOWN for counting.

Clock Divider :

- A clock divider is an electronic circuit or component that takes an input clock signal and produces an output signal with a lower frequency.
- The output signal is derived by dividing the input clock signal frequency by a certain factor.
- For example, a clock divider circuit that divides the input clock signal frequency by two will produce an output signal with a frequency that is half of the input clock signal frequency.
- The Spartan 6 FPGA Development Board XC6SLX9-TQG144 contains a 50MHz Oscillator to provide clock input to the FPGA.



Code:

```
module clock(input clk_in,output led);
reg [24:0] counter=0; //clock
reg slow_clk=1'd0;
always@(posedge clk_in) //input 50MHz
 begin
counter<=counter+1;
if (counter==0)
slow_clk=~slow_clk;  //Output 1Hz
end
assign led=slow_clock;
endmodule
```

UCF:

```
NET "clk_in" LOC = "p84";
NET "led" LOC = p33; #LSB
```

UP DOWN Counter

After designing the clock divider circuit, first we wrote a code for up counter and then for down counter and then we merged both the codes to work as up down counter with reset and is verified using the below testbench.

Working:

- On application of reset the count value will be

  00(in decimal) - up mode

  99(decimal) - down mode

Design Block:

```
module udcounter(clk, reset, updown,count);
input clk,reset,updown;
output reg [7:0] count=0;
always @(posedge clk)
begin
 if(reset==1)
count <=7'd0;
else if(updown==0)
 if(count==7'd99)
count<=7'd0;
else
count<=count+7'd1;
else if(count==7'd0)
count<=7'd99;
else
count<=count-7'd1;
 end
endmodule
```

Test bench:

```
module counter_testbench();
reg clk, reset,updown;
wire [7:0] count;
udcounter dut(clk, reset,updown, count);
```

```
 initial
begin
clk=0;
forever #5 clk=~clk;
end
initial
begin
reset=1;
#10;
 reset=0;
updown=0;
#990;
reset=0;
 updown=1;
#990;
reset=0;
updown=0;
#495;
end
endmodule
```

Design block of the counter with reset,start,load and stop

```
module udcounter(clk, reset, load, updown, count, start, stop);
input clk,reset,load,updown,start,stop;
output reg [7:0] count=0; //output
reg slow_clk=1'd0;
reg [24:0] counter=0; //clock reg clk_out;
always@(posedge clk) //50MHz
 begin
counter=counter+1;
 if (counter==0)
slow_clk=~slow_clk; //1Hz
end
always @(posedge slow_clk)
 begin
if(reset==1) // Reset button is pressed
count <=7'd0; else if(stop==1) // When stop button is pressed
count <= count;
else
if(start == 1) //start counting when stop is off
if(updown==0 && load==0) // Up mode
 if(count==7'd99) count<=7'd0;
else count<=count+7'd1;
else if(updown==0 && load==1) // Up mode and load pressed
count<=7'd90;
else if(updown==1 && load==0) // In down mode
if(count==7'd0)
count<=7'd99;
else count<=count-7'd1;
else // In down mode and load pressed
count<=7'd10;
end
endmodule
```

Test Bench:

```
module udcounter_tb();
reg clk, reset,load,updown, start, stop;
wire [7:0] count;
udcounter dut(clk, reset, load, updown, count, start,
stop);
initial
begin
clk=0;
forever #5 clk=~clk;
end
initial
begin
reset=1;
#10;
updown=0;
reset=0;
load=0;
start=1; //Start is always on(and hence level based)
 stop=0;
#100;
stop=1;  //Stop is an interrupt(level based) #100; stop=0;
#100;
reset=1;
#10;
reset=0;
 #990;
load=1;       //load is edge triggered
#10;
load=0;
#110;
updown=1;     //updown is level triggered(switch)
#990;
```

```
load=1;
#10;
load=0;
 end
 endmodule
```

This is the final code

```
module udcounter(clk, reset, load, updown, count, start,
stop, digit,seg1, buzzer);
input clk,reset,load,updown,start,stop;
output reg [7:0] count=0;  //output
output reg buzzer=0;
output [3:0]digit;
output reg [7:0]seg1;
reg slow_clk=1'd0;
reg [24:0] counter=0;  //clock
reg clk_out;

always@(posedge clk) //50MHz
begin
counter=counter+1;
if (counter==0)
slow_clk=~slow_clk; //1Hz:The clock is working perfectly at
1 second
end
always @(posedge slow_clk)
begin
  if(reset==1) // Reset button is pressed
    count <=7'd0;
  else if(stop==1) // When stop button is pressed
    count <= count;
  else if(start == 1) // Start counting when stop is off
      if(updown==0 && load==0) // Up mode
```

```
        if(count==7'd99)
          count<=7'd0; // Rollover from 99 to 0
        else
          count<=count+7'd1; // Incrementing
      else if(updown==0 && load==1) // Up mode and load
pressed
        count<=7'd90;

      else if(updown==1 && load==0) // In down mode
        if(count==7'd0)
          count<=7'd99; // Rollover from 0 to 99
        else
          count<=count-7'd1; // Decrementing
      else // In down mode and load pressed
        count<=7'd10;
end
//both the digits display the same number in seven segment
always @ (count)
begin
case(count)
4'b0000: seg1 = 8'b00000011;
4'b0001: seg1= 8'b10011111;
4'b0010: seg1= 8'b00100101;
4'b0011: seg1= 8'b00001100;
4'b0100: seg1= 8'b10011000;
4'b0101: seg1= 8'b01001001;
4'b0110: seg1= 8'b01000001;
4'b0111: seg1= 8'b00011111;
4'b1000: seg1= 8'b00000001;
4'b1001: seg1= 8'b00001001;
4'b1010: seg1=8'b11111101;
4'b1011: seg1=8'b11111101;
4'b1100: seg1=8'b11111101;
```

```
4'b1101: seg1=8'b11111101;
4'b1110: seg1=8'b11111101;
4'b1111: seg1=8'b11111101;
endcase
end
//The logic module is working properly and we have verified
it using testbench
always@(count)
begin
     if(count<=7'd99|count<=7'd0)
          buzzer<=1;// when the output is 99 and 0 the
buzzer will be buzzed until the upcounting and downcounting
takes place
     else
          buzzer<=0;
end
endmodule
```

UCF

```
NET "clk" LOC=p84;
NET "updown" LOC=p22;
NET "reset" LOC=P40;
NET "load" LOC=P44;
NET "start" LOC=P45;
NET "stop" LOC=P41;
NET "count<0>" LOC = p33; #LSB   //LED glows for up&down
counting
NET "count<1>" LOC = p32;
NET "count<2>" LOC = p30;
NET "count<3>" LOC = p29;
NET "count<4>" LOC = p27;
NET "count<5>" LOC = p26;
NET "count<6>" LOC = p24;
```

```
NET "digit[0]" LOC = P127;
NET "seg1[0]" LOC = P134;
NET "seg1[1]" LOC = P137;
NET "seg1[2]" LOC = P138;
NET "seg1[3]" LOC = P139;
NET "seg1[4]" LOC = P140;
NET "seg1[5]" LOC = P141;
NET "seg1[6]" LOC = P142;
NET "seg1[7]" LOC = P143;
```