Digital Lab 3:

Experiment4:

LED Light Control Circuit

Date: 2023/11/09

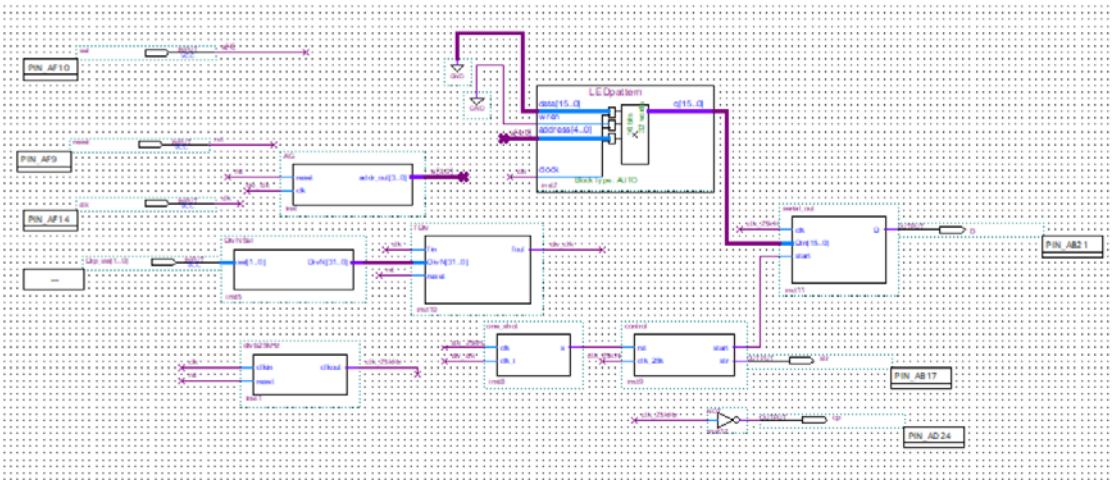
Class: 電機三全英班

Group: Group 11

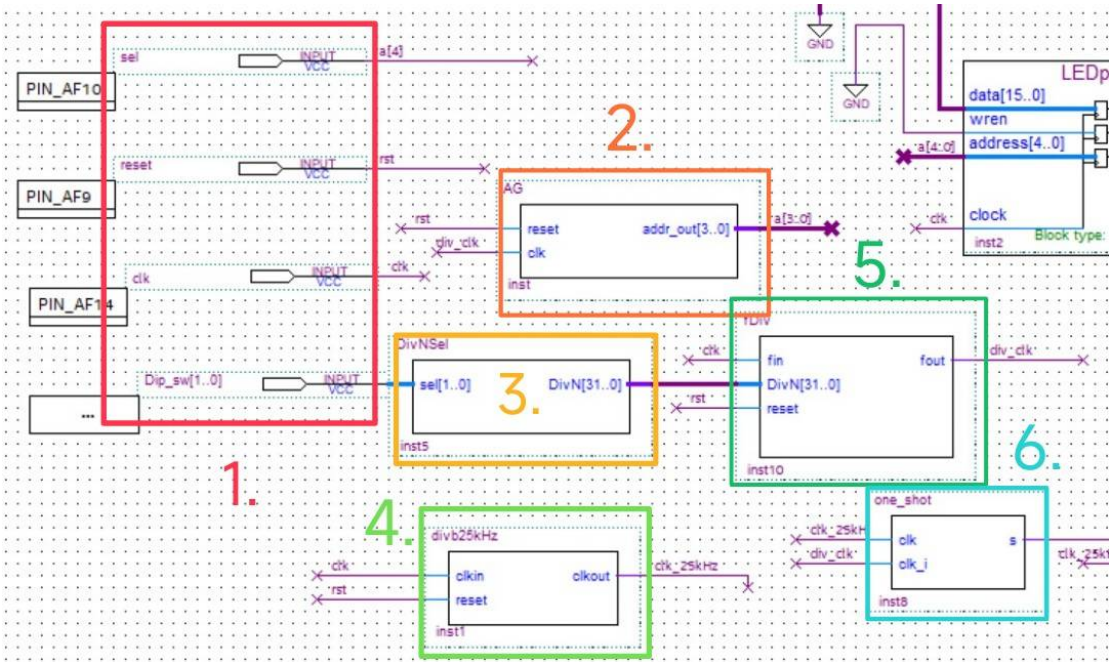
Name: B103105006 胡庭翊

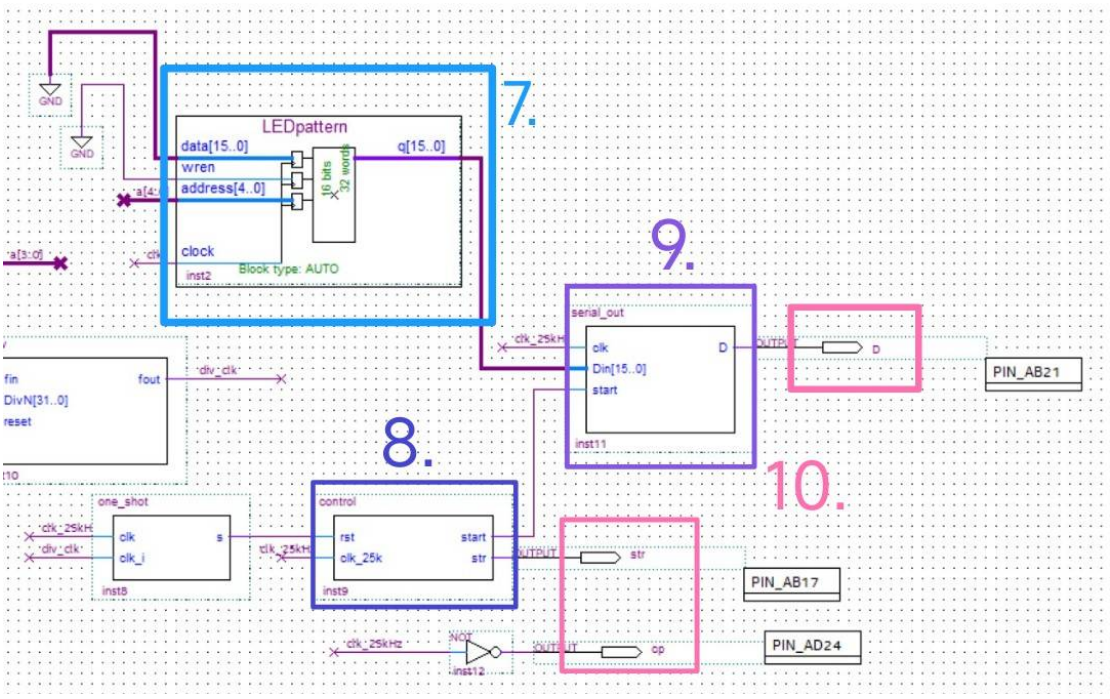
I. Block Diagram

Structure:



Function:





Descriptions:

1. Inputs

   Here, to control the circuit, we have one bit of reset, a 50M Hz clock input connected to the FPGA board, a two bit dip switch of speed control for selecting frequency , and a select bit to choose the output pattern mode of the LED which is connected to the memory.

2. Address Generator

   Generates the 4-bit address values required for memory. When the "rst" signal is raised, the address is reset to zero, allowing the memory to start outputting data from the beginning.

3. Division Selector

   Select the needed frequency corresponding to the 2-bit dip-switch, the possible frequencies that are needed are 2Hz, 4Hz, 6Hz and 8Hz.

4. 25kHz Frequency Divider

   Convert 50MHz of the clock on the FPGA board into 25kHz.

5. Frequency Divider 50M/divHz(iHz):

   Converts the 50 MHz signal on the board into 2 Hz, 4 Hz, 6 Hz, and 8 Hz signals to control the speed of switching the stored patterns in memory.

6. One Shot

Designed using a state machine, it generates a pulse signal of length 1 clock cycle (25 kHz) after a longer input signal's positive edge trigger.
Transfer the low frequency into an active high frequency circuit module.

7. Memory
   Megafunction-generated memory module (16 bits * 32 words) that stores two different LED patterns, as provided in the attached MIF file.
8. Control
   Provides the clock signal to the IC. As Serial Out uses a positive-edge trigger, an inverter is applied to the IC's CLOCK (CP) signal to make it negative-edge triggered.
   This ensures that data is written to the IC when it's ready, synchronized with the
   negative edge trigger.
9. Serial Output
   When "start" is set to 1, the input 16-bit value is stored in the register.
   After "start" becomes 0, on each positive edge of the clock, the highest bit of the
   register is sequentially sent out to D
10. Outputs
    Pins: str, CP, D
    Control the latch enable, input clock, and data for each IC separately.

II. Address Generator

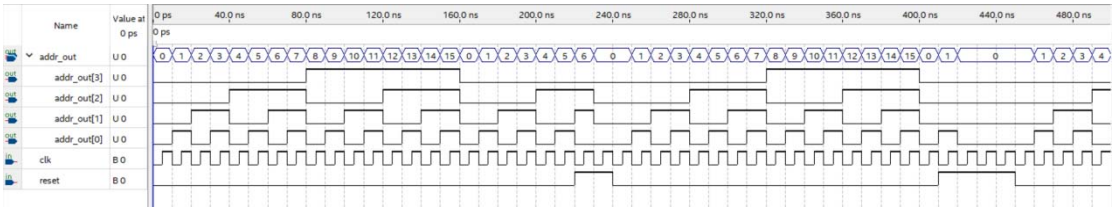
A. Verilog Code and Comment

```
1   module AG(reset, clk, addr_out);
2
3   input reset, clk;//input declaration
4   output reg [3:0] addr_out;//4-bit output address
5
6   reg [3:0] count;
7   wire [3:0] _addr_out;
8
9   always@(posedge clk or posedge reset)//when clk or reset is in posedge
10  begin
11      if(reset)
12          count<=7'b0;//when reset goes to 1, assign 7 bits of 0 into count, that is, reset its value
13      else
14          count<=(count>=7'd15)?7'd0:count+1;/*if count reaches to 7'd15, restart its value,
15                                             otherwise, keep counting*/
16  end
17
18  always@(negedge clk)//clk is negedge triggered
19      addr_out<=count;//assign count value to the output
20  endmodule
21
```

B. Simulation

The address would be generated when either the clock or reset is in posedge.

The value of count will be reset to 0 if reset is 1, otherwise, it will keep increasing until it reaches to 15, and will restart counting from 0.



III. 25k frequency divider(divb25kHz)

A. Verilog Code and Comment

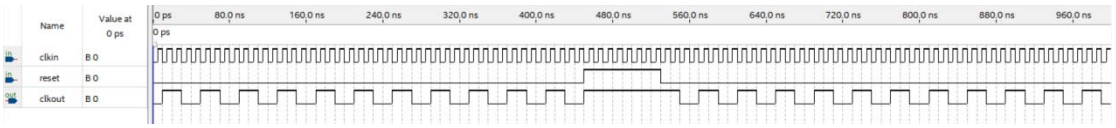
```
1   module divb25kHz(clkin, reset, clkout);
2
3   input clkin, reset;
4   output reg clkout;
5   reg [31:0] count;
6   wire [31:0] divn, divnh;
7
8   assign divn= 32'd2000; //define divn to be 50M/25k = 2000
9   assign divnh= divn>>1;//define divnh to be half of divn
10
11  always@(posedge clkin) //while clkin is in posedge
12  begin
13      if((count>=divn)||(reset)) //if counter is larger than the specified period of time
14          count<=1; //restart the counter
15      else
16          count<=count+1; //otherwise keep counting
17  end
18
19  always@(negedge clkin) //wile clkin is in negedge
20  clkout= (count<= divnh)?1:0;
21  //output of the frequncy diveder is 1 while counting from 1 to the half period, otherwise is 0
22
23  endmodule
```

B. Simulation

We replace the frequency division ratio 32’d2000 into 32’d4 in order to make the simulation better be seen.

```
assign divn= 32'd4; //here, to simulate, we set the frequency of this divider into 1/4 of the input
assign divnh= divn>>1;//define divnh to be half of divn
```

After modifying, we generate a frequency divider that can convert the input frequency into 1/4 times of the frequency.



When clkin is in posedge, the value of count will be varied, while if reset is 1 at this moment, then 1 will be assigned into count.

When clkin is in negedge, clkout will compare the value of

count and the set value divnh, and output the correspond signal.

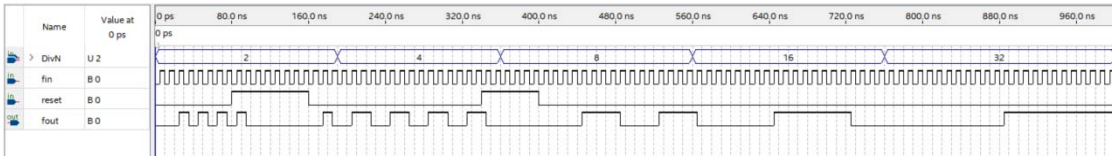
IV. fDiv (i frequency divider)

A. Verilog Code and Comment

```
module fDiv(fin,DivN, reset,fout);
input fin, reset;
output fout;
input[31:0]DivN;//the output value of DivNSel will be sent to the 32-bit input DIVN
wire [31:0]_DivN;//define 32-bit wire _DIVN
reg[31:0] count; //define 32-bit register count
reg fout;//define a register fout


assign _DivN = {1'b0,DivN[31:1]}; //_DIVN = DIVN/2

always@(posedge fin) //triggered at the positive edge of fin
begin
    if(reset)
        count<=32'd1; //when reset is 1, 32'd1 will be assigned into count
    else
        count<=(count>=DivN)?32'd1:count+1; //if count>=DIVN, count=1 in decimal. if not, count=count+1
        fout<=(count<=_DivN)?1'b0:1'b1; //if count<=DIVN, fout=0 in binary. if not fout=1 in binary
end
endmodule
```

B. Simulation

The value of DivN is the output of the DivNSel, we can modified our frequency by simply changing the input frequency.



The operation of output signal and reset both happens when fin is in posedge, this is the reson why when the input signals change, there is a slice of delay in output.

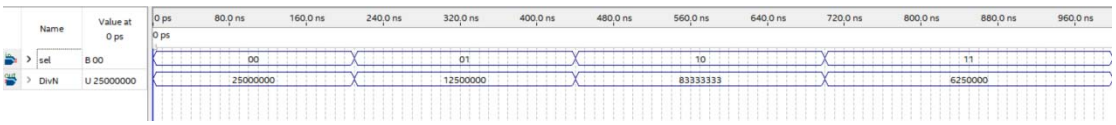
V. Division Selector (DivNSel)

A. Verilog Code and Comment

```
module DivNSel(sel, DivN);
input [1:0] sel;//define a 2-bit input sel
output reg[31:0] DivN;//define a 32-bit output DivN

always@(sel)
    case(sel)
        2'b00: DivN <= 32'd25000000;//when sel=00 in binary, we will make a 2Hz frequency divider
        2'b01: DivN <= 32'd12500000;//when sel=01 in binary, we will make a 4Hz frequency divider
        2'b10: DivN <= 32'd83333333;//when sel=10 in binary, we will make a 6Hz frequency divider
        default: DivN <= 32'd6250000;//when sel is other value, we will make a 8Hz frequency divider
    endcase
endmodule
```

B. Simulation

The output value of DivN is depended on the input value of sel.



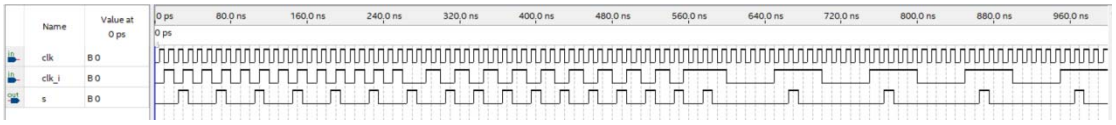
VI. One Shot

A. Verilog Code and Comment

```
module one_shot(clk,clk_i,s);
input clk,clk_i;
output reg s;
reg [1:0] cs,ns;
wire _s;

always @(posedge clk)   //triggered when clk is in posedge
   cs<=ns;  //assign ns to cs

always @(cs) //when the value of cs differs, followed by the value of cs, we have following conditions
   case(cs)
   2'b00: ns <= (clk_i)?2'b01:2'b00; //if clk_i=1, assign 2'b01 to ns, else assign 2'b00 to ns
   2'b01: ns <= 2'b10; //assign 2'b10 to ns
   2'b10: ns <= (clk_i)?2'b10:2'b00; //if clk_i=1, assign 2'b10 to ns, else assign 2'b00 to ns
   default: ns <=2'b00; //otherwise, assign 2'b00 to ns
   endcase

always @(posedge clk)
   s <= _s;
assign _s = (cs==2'b01)?1'b1:1'b0; //if cs equals to 2'b01, s will be 1'b1, else it will be 1'b0

endmodule
```

B. Simulation

The input of clk is the output of 25kHz frequency divider, while the input of clk_i is the output of changeable frequency divider. When clk is in posedge, one_shot will judge the value of clk_i, and assign the value of cs corresponding to the states.
After that, the value of s would be assigned corresponding to the status of cs.



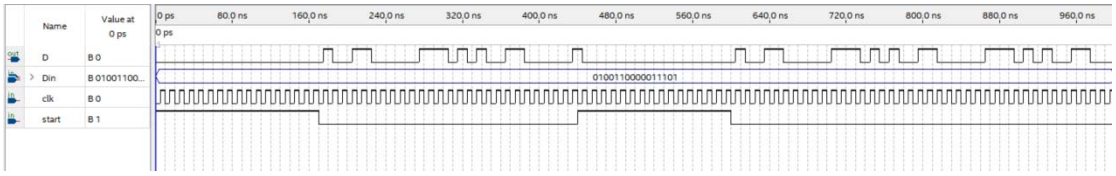
VII. Serial Out

A. Verilog Code and Comment

```
module serial_out(clk,Din,start,D);
input clk,start;
input [15:0] Din;
output D;
reg [15:0] Buf;
wire [15:0] nBuf;

assign nBuf= {Buf[14:0],Buf[15]};//rotate left, the 16th bit will be the 1st bit
assign D=Buf[15]; //the output D is the last bit of Buf

always @(posedge clk)
   if(start)
      Buf <= Din; //if start=1, the value of Buf will be the input Din
   else
      Buf <= nBuf; // else it will rotate left

endmodule
```

B. Simulation

When clk is in posedge with start =1, output the largest bit of current registor Buf, else when clk is in posedge but start=0, rotate left the current registor and output its largest bit.



VIII. Control

A. Verilog Code and Comment

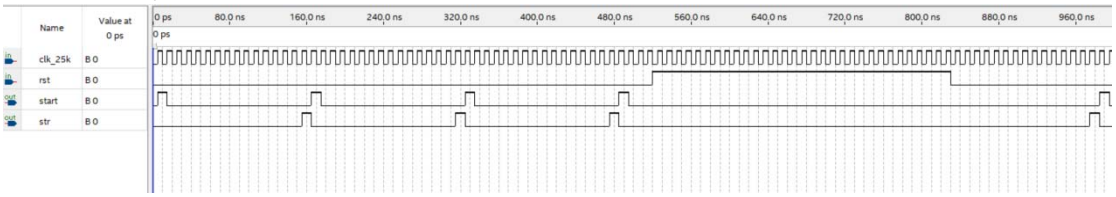
```
module control (rst, clk_25k, start, str);
input rst, clk_25k;
output reg start, str;
reg [3:0] count;
wire _start, _str;

assign _start= ~(|count); //if count is 4'b0000, _start = 1, else _start=0
assign _str= &count; //if count is 4'b1111, _str=1, else _str=0

always@ (posedge clk_25k) //triggered when clk_25k is posedge
   begin
      count <= (rst==1'b1)?4'd1:count+1; //if rst i 1'b1, assign 4'd1 to count, else count+1
      str <= _str; //assign _str to output str
      start <= _start; //assign _start to output start
   end
endmodule
```

B. Simulation

In every posedge clk_25k, count will keep counting until rst is 1’b1, and the value of str and start will be assign at this moment too.

When count is 4’b0000, the value of start will be 1, and when count is 4’b1111, the value of str will be 1, in other case, they are 0.

We know that after 1111, count will be 0000 again, that is the reason why after str=1, start will be 1 too.



IX. Reflection

The practical part of the experiment required us to solder circuits, a task I found a bit tricky and felt like it took up time and resources without a clear need. Even though the first time soldering circuit boards was kind of fun, thinking about future experiments involving soldering made me a bit tired, especially remembering the five hours we spent on it last time.

Moreover, the experiment guide didn't come with any ready-to-use code. This meant we had to tweak our existing modules, which was a bit tough. However, it turned out to be a good chance to improve our skills.