Digital Lab 3:

Experiment5:

Analog-to-Digital Converter Control Circuit

Date: 2023/11/23

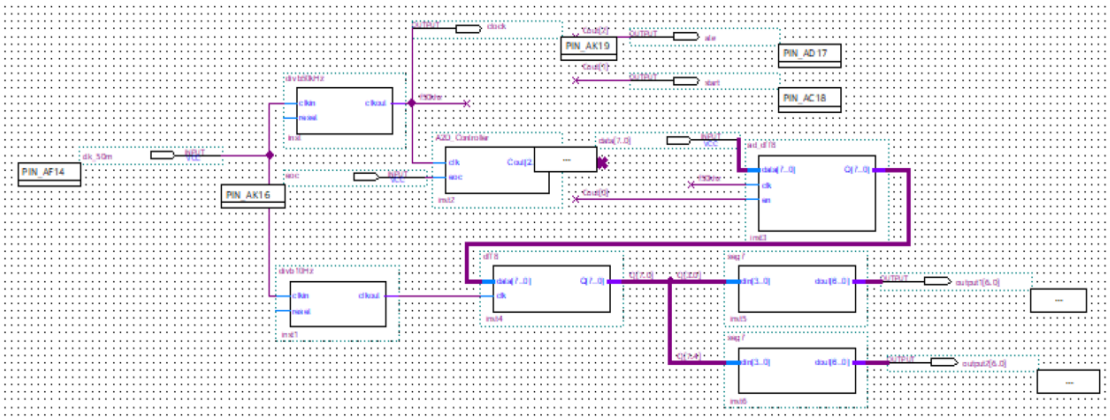
Class: 電機三全英班

Group: Group 11

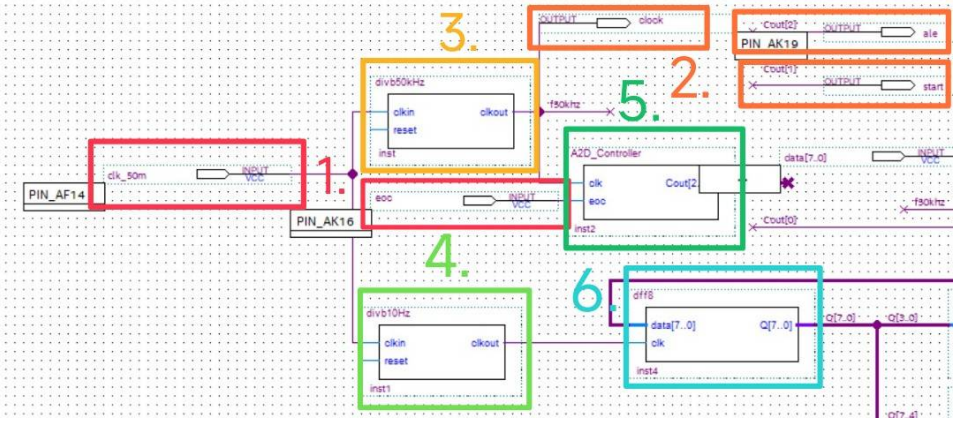
Name: B103105006 胡庭翊

I. Block Diagram

Structure:



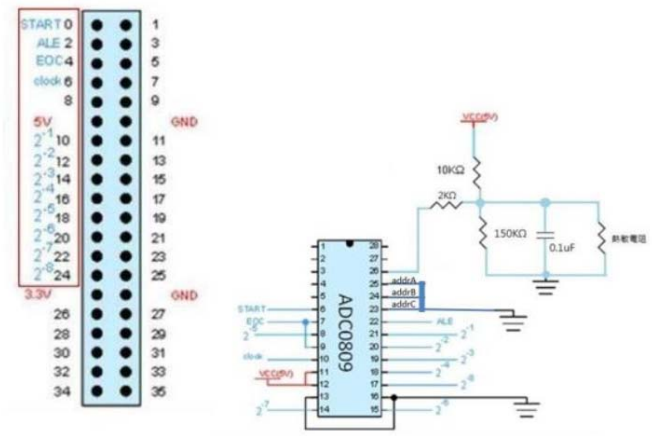
Function:



Descriptions:

1. Inputs:
   The inputs are 50MHz clock set by the board, 1 bit of eoc, and 8 bits of data, where the data and eoc are connected to the

ADC0809 IC so the result of the input value will depend on the 10k thermistor designed in our circuit.



2. Outputs:
   The outputs are clock, ale, start, and two 7-segment outputs. The output clock frequency is of 50kHz generated from the 50MHz to 50kHz frequency divider.
   And the ale and start are the last two bits of output of AD Controller.
3. 50kHz Frequency Divider: Convert the 50MHz input down into 50kHz.
4. 10Hz Frequency Divider: Convert the 50MHz input down into 10Hz.
5. AD Controller: Control the conversion process of the ADC0809 and send sampling signals to the display circuit.
6. Dff8: Latch the data from AD-Dff88.
7. AD-Dff8: Latch the data signal sent from the ADC0809.
8. 7 Segment Decoder1: Output the first bit data into hexadecimal number.
9. 7 Segment Decoder2: Output the second bit data into hexadecimal number.

II. Frequency Divider

A. Verilog Code and Comment

1. 10Hz

```
module divb10Hz(clkin, reset, clkout);

input clkin, reset;
output reg clkout;
reg [31:0] count;
wire [31:0] divn, divnh;

assign divn= 32'd5000000;//50MHz/5000000=10Hz
assign divnh= divn>>1;//define divnh to be half of divn

always@(posedge clkin) //while clkin is in posedge
begin
   if((count>=divn)||(reset)) //if counter is larger than the specified period of time
      count<=1; //restart the counter
   else
      count<=count+1; //otherwise keep counting
end

always@(negedge clkin) //wile clkin is in negedge
clkout= (count<= divnh)?1:0;
//output of the frequncy diveder is 1 while counting from 1 to the half period, otherwise is 0

endmodule
```

2. 50kHz

```
module divb50kHz(clkin, reset, clkout);

input clkin, reset;
output reg clkout;
reg [31:0] count;
wire [31:0] divn, divnh;

assign divn= 32'd1000;//50MHz/1000=50kHz
assign divnh= divn>>1;//define divnh to be half of divn

always@(posedge clkin) //while clkin is in posedge
begin
   if((count>=divn)||(reset)) //if counter is larger than the specified period of time
      count<=1; //restart the counter
   else
      count<=count+1; //otherwise keep counting
end

always@(negedge clkin) //wile clkin is in negedge
clkout= (count<= divnh)?1:0;
//output of the frequncy diveder is 1 while counting from 1 to the half period, otherwise is 0

endmodule
```

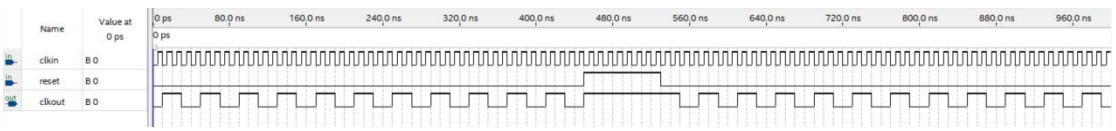
B. Simulation

The code of 10Hz and 50kHz frequency divider are approximately the same, the only difference of the two is its frequency division ratio.

Hence, to simulate, we replace both of the frequency division ratio 32’d5000000 and 32’d1000 into 32’d4 in order to make the simulation better be seen:

```
assign divn= 32'd4;
```

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.

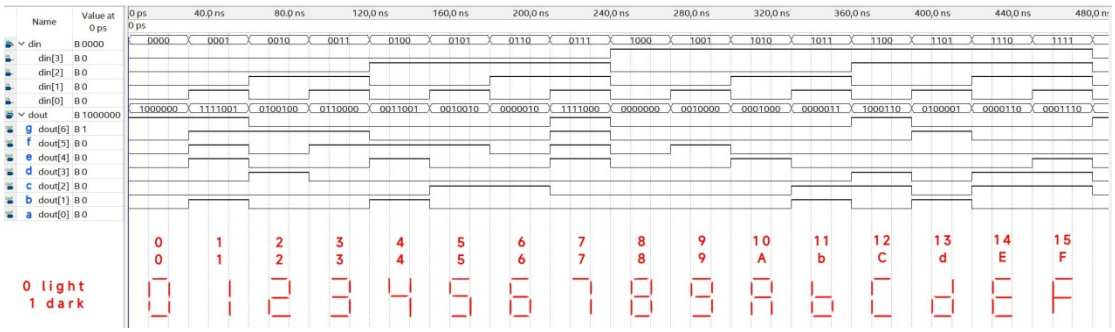
When clkin is in negedge, clkout will compare the value of count and the set value divnh, and output the correspond signal.

III. 7 Segment Decoder

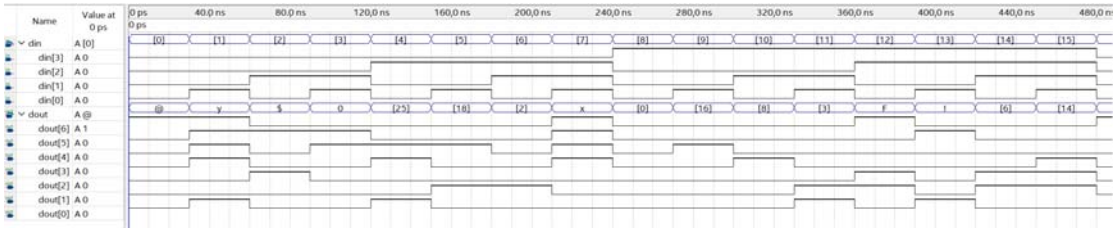
A. Verilog Code and Comment

```
module hw01_7seg(din,dout);
input    [3:0] din; //4 bits of binary input din
output   [6:0] dout; //7 bits of output (7-segments), represented in hexidecimal number)
reg      [6:0] dout; //7 bits of registor

always@(din) //trigger when din changed
begin
   case(din)         //gfedcba, 0:light; 1:dark
       4'b0000:dout<=7'b1000000; //0
       4'b0001:dout<=7'b1111001; //1
       4'b0010:dout<=7'b0100100; //2
       4'b0011:dout<=7'b0110000; //3
       4'b0100:dout<=7'b0011001; //4
       4'b0101:dout<=7'b0010010; //5
       4'b0110:dout<=7'b0000010; //6
       4'b0111:dout<=7'b1111000; //7
       4'b1000:dout<=7'b0000000; //8

       /*complete the remaining part!! */
       4'b1001:dout<=7'b0011000; //9
       4'b1010:dout<=7'b0001000; // A
       4'b1011:dout<=7'b0000011; // b
       4'b1100:dout<=7'b1000110; // C
       4'b1101:dout<=7'b0100001; // d
       4'b1110:dout<=7'b0000110; // E
       4'b1111:dout<=7'b0001110; // F

   endcase
end
endmodule
```

B. Simulation



Convert the binary value into ASK-II code to check if the decoder is correct:

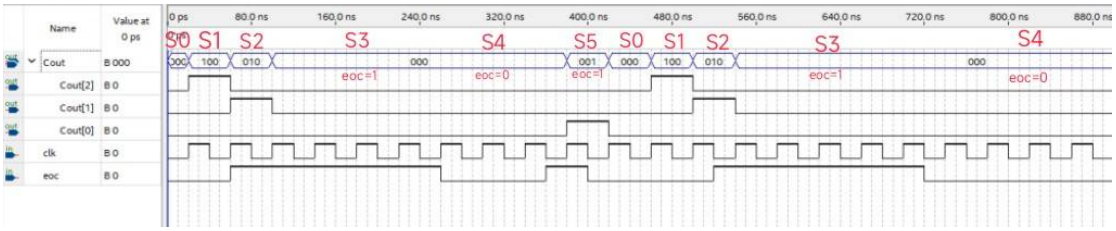


IV. AD Controller

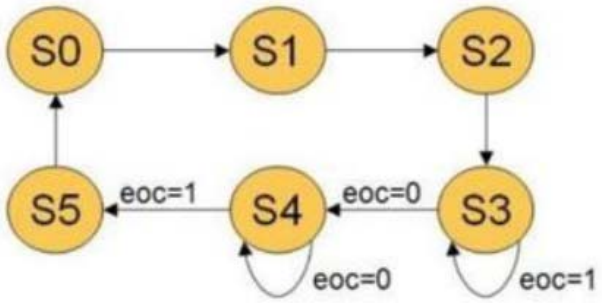
A. Verilog Code

```
module A2D_Controller(clk, eoc, Cout);
input clk, eoc;
output reg [2:0] Cout; //ALE, start, g_d;

reg [2:0] cs, ns;

always @(posedge clk)//every posedge of clk, ns will be assigned to cs
   cs<=ns;

//triggered when the value of cs or eoc differs
//the value of ns will be assigned according to the states of cs
always @(cs or eoc)
   case(cs)
   3'd0: ns <= 3'd1;
   3'd1: ns <= 3'd2;
   3'd2: ns <= 3'd3;
   3'd3: ns <= (eoc)?3'd3:3'd4;//ns will be assigned to 3'd3 if eoc=1, else it will be assigned to 3'd4
   3'd4: ns <= (eoc)?3'd5:3'd4;//ns will be assigned to 3'd5 if eoc=1, else it will be assigned to 3'd4
   3'd5: ns <= 3'd0;
   default: ns <= 3'd0;//otherwise, ns will be 3'd0
   endcase

//triggered when the value of cs differs
//the value of Cout will be assigned according to the states of cs
always @(cs)
   case(cs)
   3'd0: Cout <= 3'b000;//S0
   3'd1: Cout <= 3'b100;//S1
   3'd2: Cout <= 3'b010;//S2
   3'd3: Cout <= 3'b000;//S3
   3'd4: Cout <= 3'b000;//S4
   3'd5: Cout <= 3'b001;//S5
   default: Cout <= 3'b000;//otherwise, Cout will be 3'b000
   endcase

endmodule
```

B. Simulation



The three bits in Cout represent ALE, start, g_d respectively.
In S1, the ALE signal is set to 1. Following by S2 where the START signal is set to 1. Transitioning to S3, The EOC condition is checked. If it's 0, it signifies that the conversion has started and the controller proceeds to S4. The conversion process completes when EOC becomes 1.
We connect EOC and Output Enable, meaning that when EOC=1, requests for value retrieval can be issued during this period. At S5, a request for value retrieval (signal line is gnd) is send, then the system returns to S0.



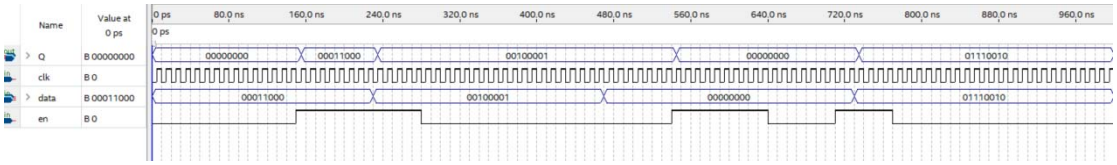
State	Output
S0	ale=0,start=0,g_d=0
S1	ale=1,start=0,g_d=0
S2	ale=0,start=1,g_d=0
S3	ale=0,start=0,g_d=0
S4	ale=0,start=0,g_d=0
S5	ale=0,start=0,g_d=1

V. AD-Dff

A. Verilog Code

```
module ad_dff8(data, clk, en, Q);
    input [7:0] data;
    input clk, en;
    output reg [7:0] Q;

    always@(posedge clk)
    begin
        if(en)
            Q <= data;//when clk is in posedge, the value of data will be assigned to Q
        else
            Q <= Q;//else, the value of Q will not change
    end

endmodule
```

B. Simulation



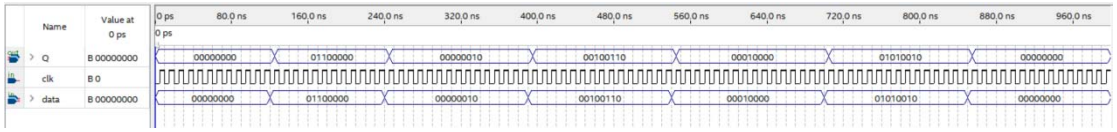
When the enable is 1, the value of data will be assigned to Q, otherwise it will remain the same.

VI. Dff

A. Verilog Code

```
module dff8(data, clk, Q);
    input [7:0] data;
    input clk;
    output reg [7:0] Q;

    always@(posedge clk) //when clk is in posedge, the value of data will be assigned to Q
    begin
        Q <= data;
    end

endmodule
```

B. Simulation



The value of data will be assigned to Q in posedge clock.

VII. Reflection

The electrical engineering experiment involved using Verilog, an FPGA board, and soldering a breadboard to create a controller that converts analog signals into digital.

Soldering proved to be a strenuous and time-consuming task. The lingering smell of solder wasn't very pleasant, and during actual operations, we invested a considerable amount of time in troubleshooting.

We encountered some minor errors in the block diagram, and the pin planner had to be edited multiple times. Luckily, our breadboard passed the tests without any issues, sparing us the frustration of having to solder everything again.

Special thanks to the teaching assistant for helping us troubleshoot each time; it seems like our group always encounters peculiar issues.