Digital Lab 3:

Experiment1:

Control Circuits of Memory Data Scan and Output

Date: 2023/09/21

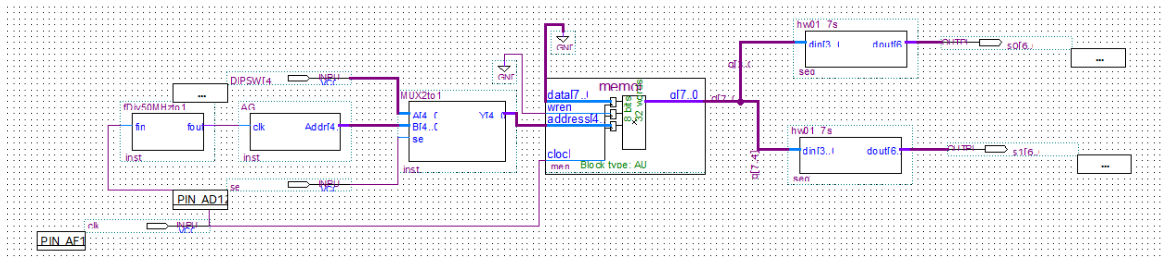
Class: 電機三全英班

Group: Group 11

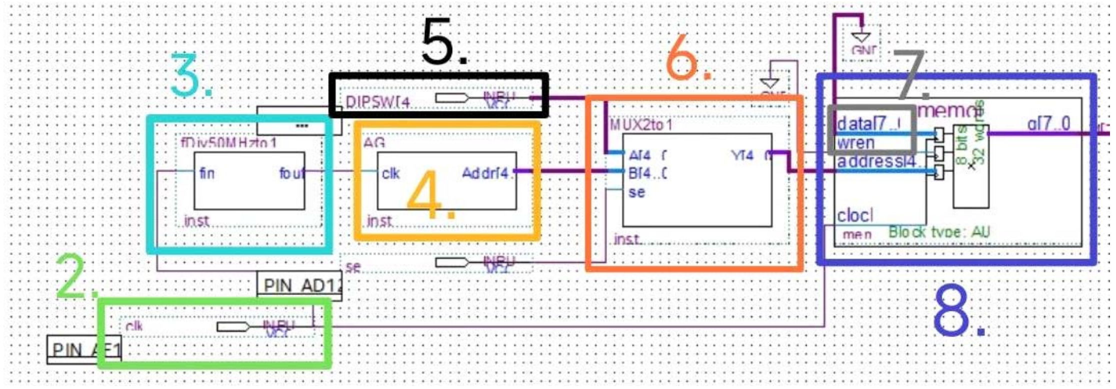
Name: B103105006 胡庭翊

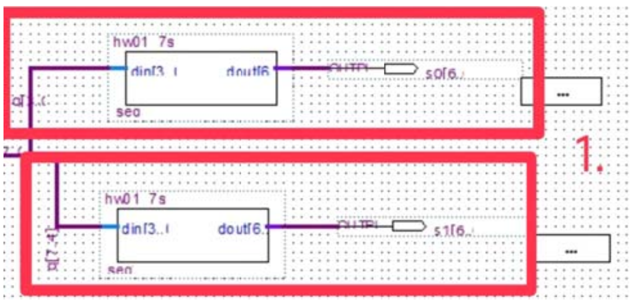
I. Block Diagram

Structure:



Function:





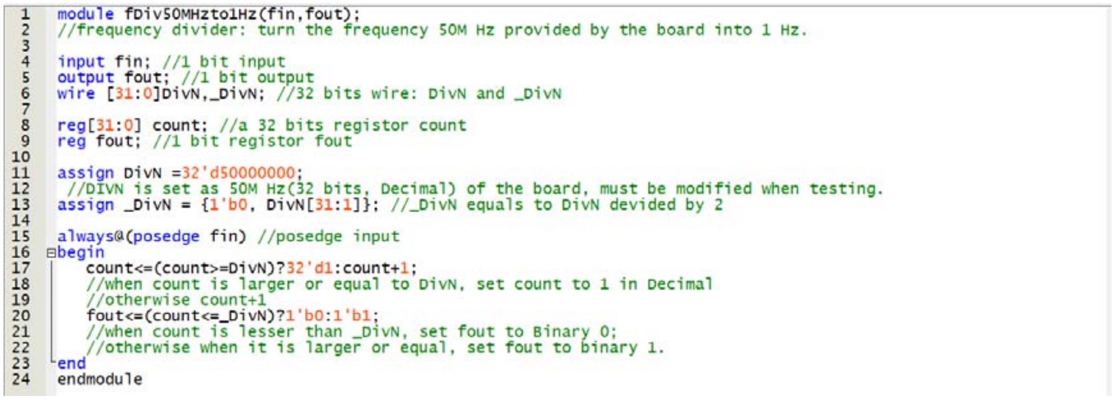
Descriptions:

1. 7 Segment Decoder: In this project, we use hexadecimal number to display and hence we need 8 bits for output. Since 4 bits is the largest number that a 7 segment display can modify, we here use two 7 segment display.
2. We use 50MHz that set by the board.
3. Frequency Divider: Convert 50MHz into 1Hz.
4. Address Generator: Using the fout of frequency divider, the clk follows a frequency of 1Hz to automatically generate the address of memory.
5. Use DIPSW to be the input of address.

6. 2-to-1 Multiplexer: Select the source of memory address: when sel is 0, the address of memory is decided by Dip Switch; when sel is 1, the address of memory is decided by address generator.
7. Since our experiment is control circuit of memory data scan and output, our wren and data should just be grounded.
8. Memory: This is the model generated by megafunction, the MUX should be its input.

II. Frequency Divider (FDiv)

A. Verilog Code and Comment



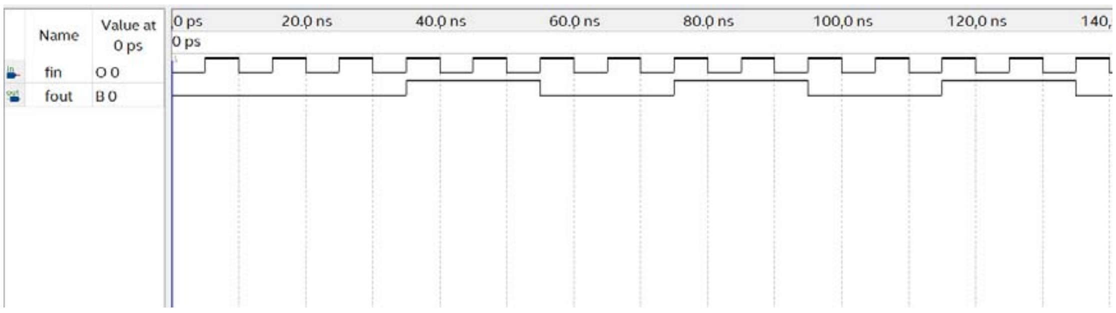
```verilog
module fDiv50MHzto1Hz(fin,fout);
//frequency divider: turn the frequency 50M Hz provided by the board into 1 Hz.

input fin; //1 bit input
output fout; //1 bit output
wire [31:0]DivN,_DivN; //32 bits wire: DivN and _DivN

reg[31:0] count; //a 32 bits registor count
reg fout; //1 bit registor fout

assign DivN =32'd50000000;
 //DIVN is set as 50M Hz(32 bits, Decimal) of the board, must be modified when testing.
assign _DivN = {1'b0, DivN[31:1]}; //_DivN equals to DivN devided by 2

always@(posedge fin) //posedge input
begin
    count<=(count>=DivN)?32'd1:count+1;
    //when count is larger or equal to DivN, set count to 1 in Decimal
    //otherwise count+1
    fout<=(count<=_DivN)?1'b0:1'b1;
    //when count is lesser than _DivN, set fout to Binary 0;
    //otherwise when it is larger or equal, set fout to binary 1.
end
endmodule
```

B. Simulation

First we need to adjust the value of DivN in verilog code into a better-simulated value.

```verilog
assign DivN =32'd4;
```

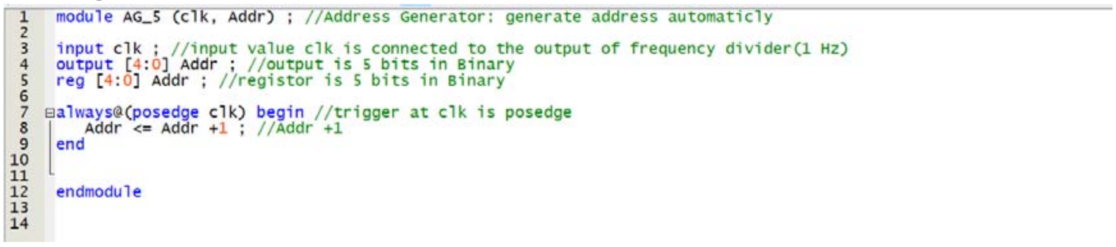
This value would make the output (fout) be 1/4 of the input (fin).



We can see that after count became 3 in the first period, since count = 3 is larger than _DivN, fout changed from 0 to 1, and hence the frequency divider started to work.
The later-on waveform of fout is 1/4 of fin's, which showed that the frequency divider is well-functioned.

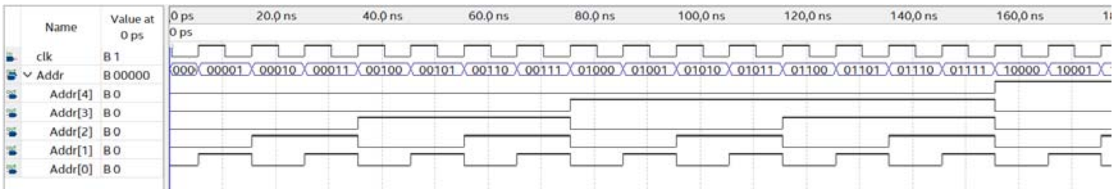
III. Address Generator (ADDG)

A. Verilog Code and Comment



```
module AG_5 (clk, Addr) ; //Address Generator: generate address automaticly

input clk ; //input value clk is connected to the output of frequency divider(1 Hz)
output [4:0] Addr ; //output is 5 bits in Binary
reg [4:0] Addr ; //registor is 5 bits in Binary

always@(posedge clk) begin //trigger at clk is posedge
    Addr <= Addr +1 ; //Addr +1
end


endmodule
```

B. Simulation



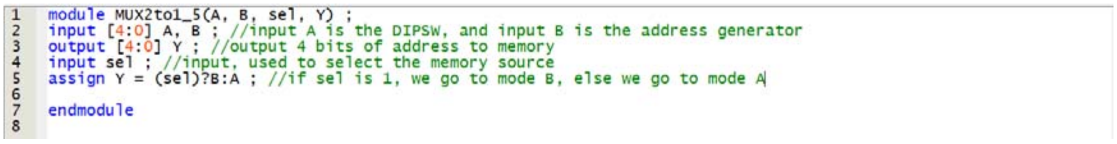
The address generator triggered when clk is in posedge, it makes its output Addr+1.

The Addr[4], Addr[3], Addr[2], Addr[1], Addr[0] represent the 5 bits of the binary address.

The output satisfied the cycle of binary numbers, so the Address Generator is functioned normally.

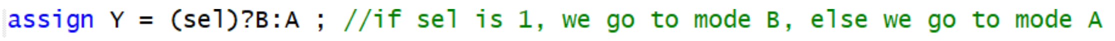
IV. 2-to-1 Multiplexer (2to1 MUX)

A. Verilog Code and Comment



```
module MUX2to1_5(A, B, sel, Y) ;
input [4:0] A, B ; //input A is the DIPSW, and input B is the address generator
output [4:0] Y ; //output 4 bits of address to memory
input sel ; //input, used to select the memory source
assign Y = (sel)?B:A ; //if sel is 1, we go to mode B, else we go to mode A

endmodule
```

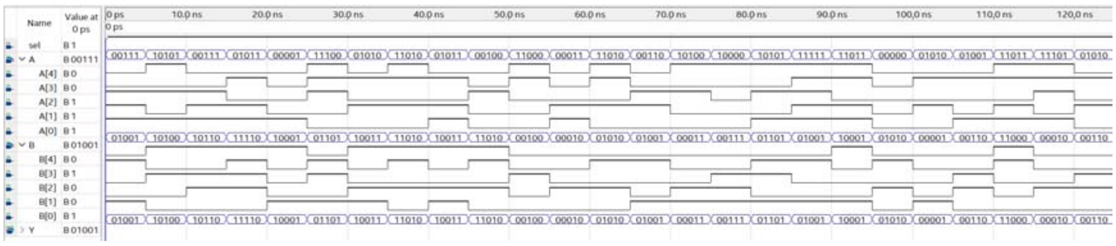
B. Simulation



```
assign Y = (sel)?B:A ; //if sel is 1, we go to mode B, else we go to mode A
```

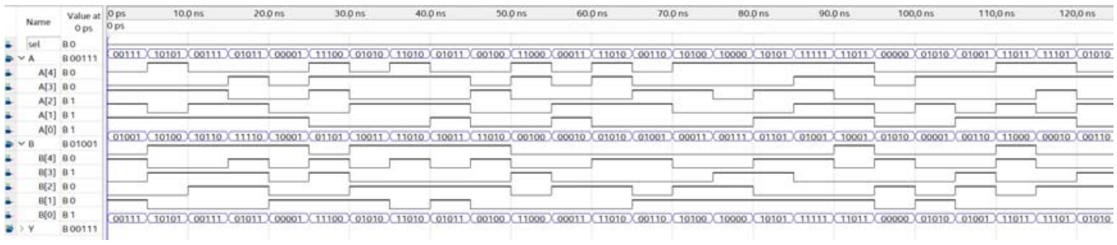
Set the address of A and B randomly.

When sel = 1:



The output Y follows directly as B.

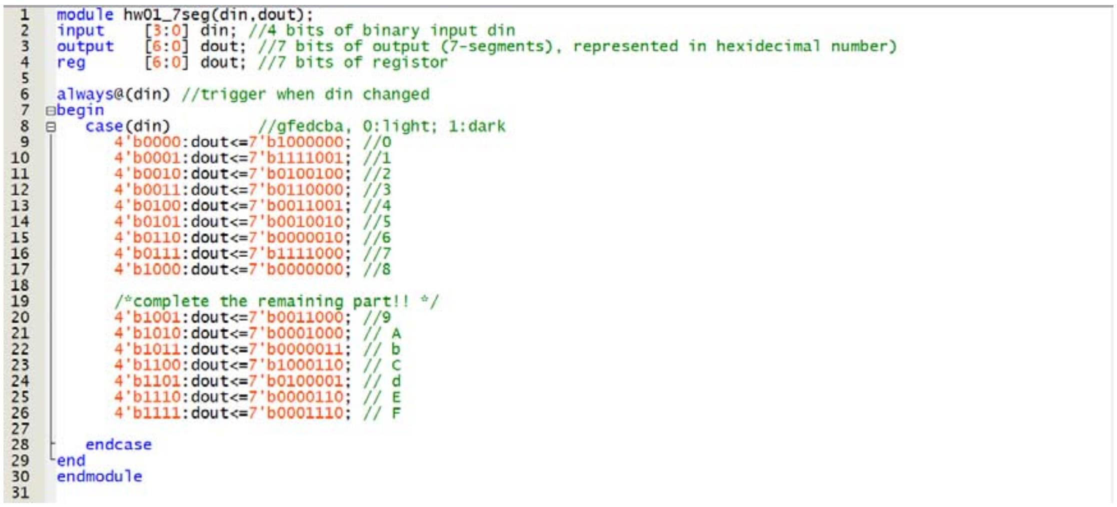
When sel = 0:



The output Y follows directly as A.

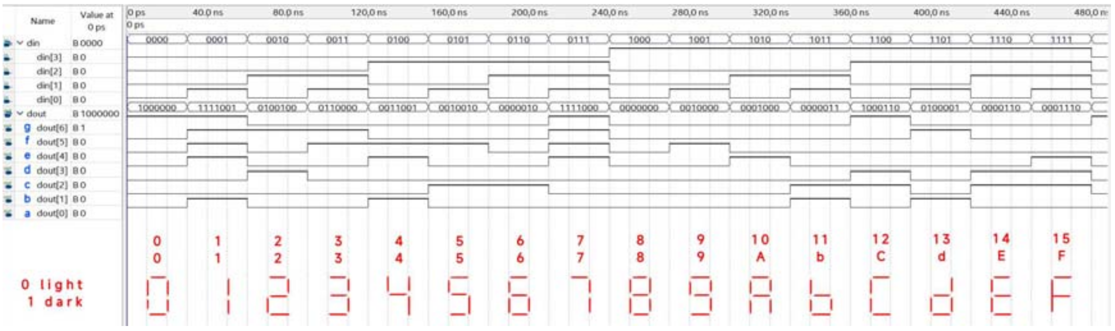
V. 7 Segment Decoder

A. Verilog Code and Comment

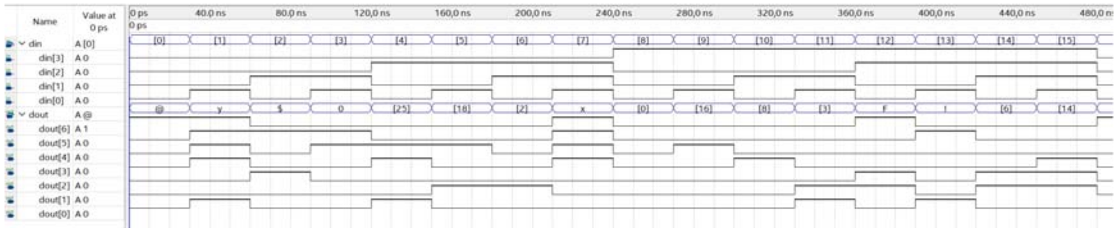


```
module hw01_7seg(din,dout);
input     [3:0] din; //4 bits of binary input din
output    [6:0] dout; //7 bits of output (7-segments), represented in hexidecimal number)
reg       [6:0] dout; //7 bits of registor

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.



VI. Reflection

In this electrical engineering experiment, I encountered Quartus, Verilog, and FPGA boards for the first time. The experiment aimed to use the Quartus software to input Verilog code from the manual and create a 50MHz to 1Hz frequency divider, 2-to-1 multiplexer, address generator, and 7-segment decoder. Additionally, we simulated waveforms and integrated pre-written memory, creating a block diagram and configuring the pin planner. The ultimate goal was to design a memory data scanning output control circuit and successfully program it into the FPGA board.

Due to my limited familiarity with the software, we couldn't complete the experiment during our class time. As a result, I devoted my own time to understand Quartus and a little verilog and also attended extra lab sessions. Quartus revealed its powerful features, although it appeared somewhat complex at first. Despite the intricate steps and the vast amount of knowledge to acquire, I believe this will be an exciting experience (if there are no conflicting exams that week).

Verilog and FPGA were also a new territory for me. While I had a brief introduction to Verilog during my freshman year, this was my first hands-on experience. This semester, I am also taking a course on practical digital systems. As a novice in hardware description languages, I realize that self-studying Verilog is crucial for my understanding and behavior in this semester.