Digital Lab 3:

Experiment3:

Simple Music Box Design

Date: 2023/10/26

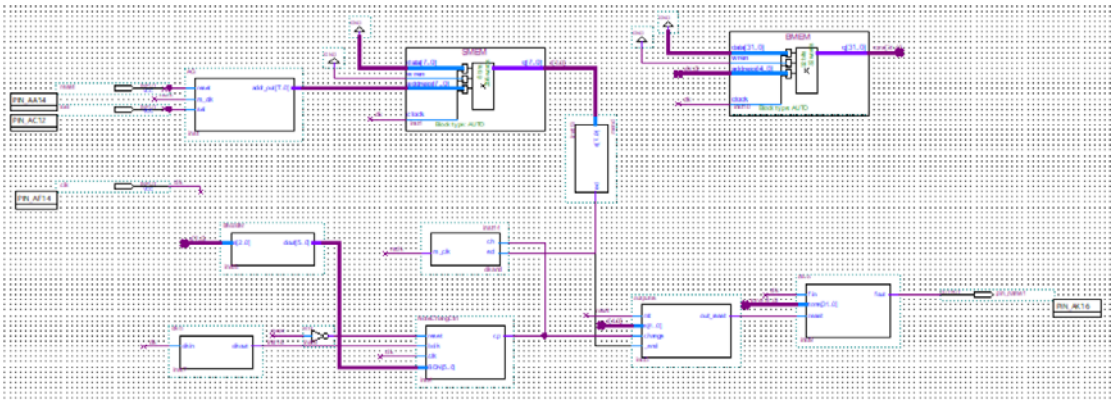
Class: 電機三全英班

Group: Group 11

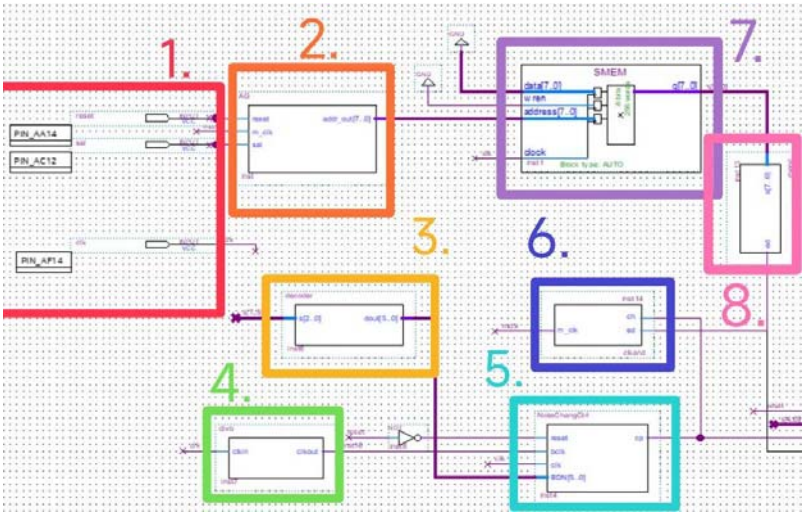
Name: B103105006 胡庭翊

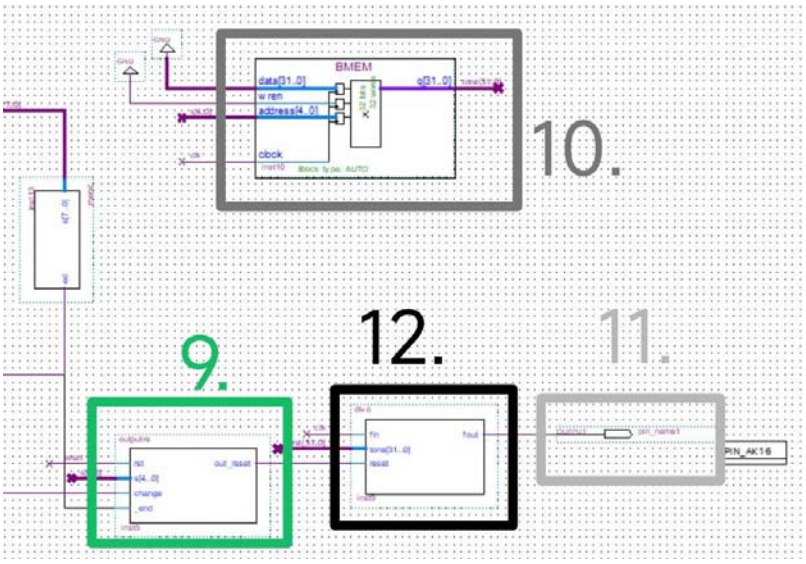
I. Block Diagram

Structure:



Function:





Descriptions:

1. Input pins:
   Reset is connected to a bottom on the FPGA board, sel is a connected to the dip switch, and clk is connected to 50MHz that set by the board.
2. Address Generator:
   Send the starting address of the selected song to the Memory (sheet music), and increment the address by 1 when the m_clk signal triggers on the positive edge, causing the Memory (sheet music) to output the data for the next note. Reset setsthe address back to zero.
3. Decoder(Duration):
   Decode the first 3 bits of the music score into 6 bits of note duration for comparison by the comparator.
   The decoder decodes values following the table below:

note	Beat	Decoder Value
1/32 note	1/8 beat	1
1/16 note	1/4 beat	2
1/8 note	Half beat	4
1/4 note	1 beat	8
1/2 note	2 beat	16
3/4 note	3 beat	24
1 note	4 beat	32

4. Beat frequency divider:
   Convert the 50MHz clock provided by the board into the shortest note duration (32nd note).
5. Simple Clock Domain Crossing Model:
   It contains functions of Asynchronous Counter, Comparator, and a DFF.
   Comparator:
   Check if the counter has counted to the correct number
   Asynchronous counter:
   Increments on the rising edge of clk, resets on the rising edge of the change signal. This counter is an asynchronous counter (reset is not synchronized with clk).
6. Mclkand:
   Logic gates.

7. Sheet music memory (SMEM):
   Store the sheet music data for all songs in memory with a size of 8 bits * 256, and choose song 2f.mif.
8. Mend:
   Logic gates.
9. Output Reset
   When reset(music reset), end(music end), change(change the sound) or rest(S[4..0] is 0), the output(rst) is High.
10. Pitch music memory(BMEM):
    Store all the divisors corresponding to each note so that the divider (output sound) can generate the frequency of the desired note. The memory size is 32 bits * 32, and choose sound1.mif.
11. Output pin:
    We should connect the output pin to the FPGA's GPIO, after that, we connect a buzzer to the board, at last music would simply be generated.
12. Audio frequency generation divider (divo):
    Convert the 50MHz clock provided by the board into the desired output frequency for the notes.

II. Address Generator

A. Verilog Code and Comment

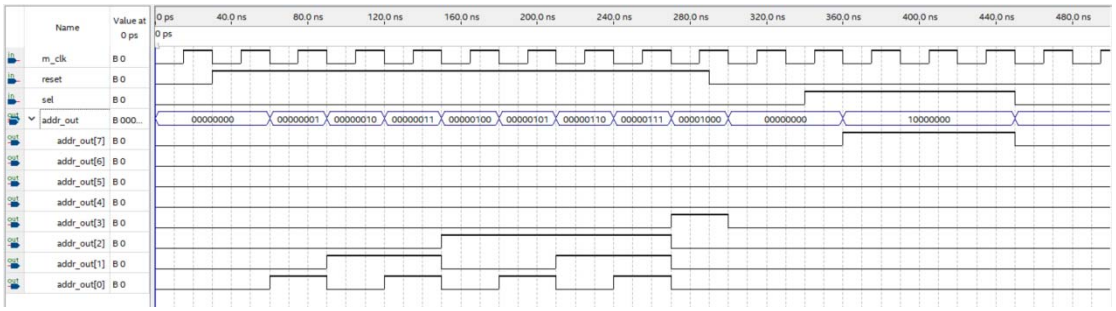
```
1   module AG(reset, m_clk, sel, addr_out); //address generator
2   //input
3   input reset, m_clk, sel;
4   //output
5   output reg [7:0] addr_out;
6   //reg&wire
7   reg [6:0]count;
8   wire [7:0]_addr_out;
9
10  always@(posedge m_clk or negedge reset) //trigger when m_clk is posedge or reset is negedge
11  begin
12      if(reset == 0)     // reset control
13          count <= 7'b0; // set count to 0000000
14      else               //up counter
15          count <= (count>= 7'd127)?7'd0:count+1; //if  count>= 7'd127, set count to 7'd0, else count +1
16  end
17
18  always@(negedge m_clk) //adding sel pin
19      addr_out <= {sel, count[6:0]}; //the address is 1 bit of sel add 7 bits of count
20
21  endmodule
```

B. Simulation

The address would be generated when the clock is in negedge, the output address is the combination of select number and count value.

And ever since the clock is in posedge or the reset is set, the

value of count would be verified: either 0000000 when reset is 0, or count +1 until count is larger than 1111111 (the value would be cycled back to 0000000 if count is larger than 1111111).



III. Beat Frequency Divider(divb)

A. Verilog Code and Comment

```
1  module divb(clkin, clkout); //beat frequency divider
2  //input
3  input clkin;
4  //output
5  output reg clkout;
6  //reg&wire
7  reg [31:0]count;
8  wire [31:0]divn, divnh;
9
10 assign divn = 32'd3750000; //frequency division ratio
11 assign divnh = divn>>1;    //half of divn
12
13 //when clkin is posedge, if count >= frequency division ratio, it will loop back to 1, else count+1
14 always@(posedge clkin)
15 begin
16     if(count>=divn)
17         count<=1;
18     else
19         count<=count+1;
20 end
21
22 // when clkin is negedge, if count<= half of the frequency division ratio, clkout = 1, else = 0.
23 always@(negedge clkin)
24 clkout = (count<=divnh)?1:0;
25
26 endmodule
```

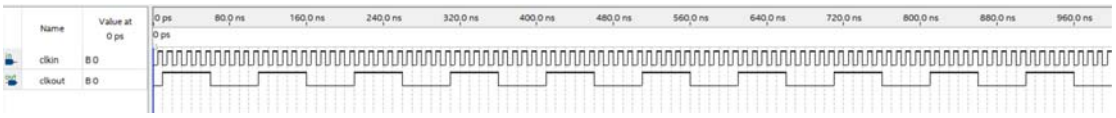
B. Simulation

We replace the frequency division ratio 32’d3750000 into 32’d10 in order to make the simulation better be seen.

```
assign divn = 32'd10;    //frequency division ratio
assign divnh = divn>>1;    //half of divn
```

The addition of count is triggered in posedge clkin, while the output clkout is triggered in negedge.

Value of count would be added until it meets the division ration, and clkout would be 1 when clkout is larger than half of the division ratio.



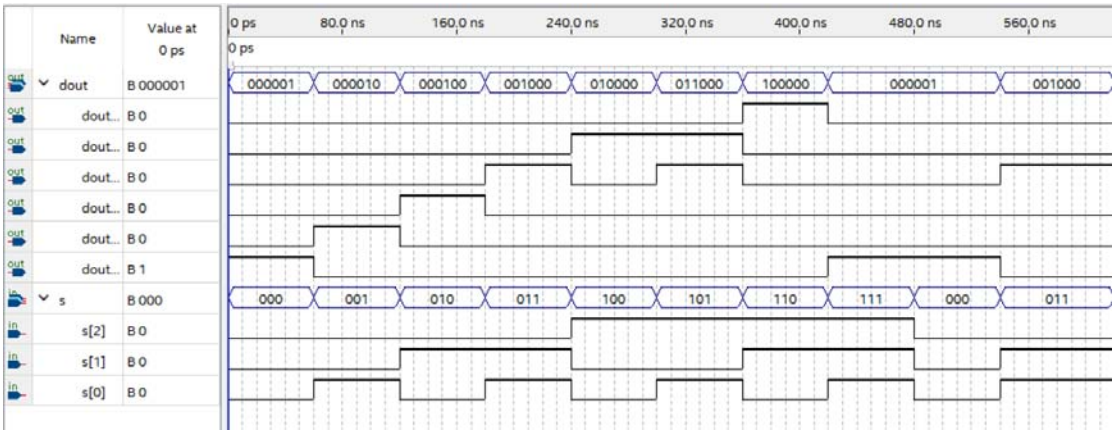
IV. Decoder

A. Verilog Code and Comment

```
module decoder(s, dout);
//input
input [2:0]s;
//output
output reg [5:0]dout;

always@(s)
   case(s)
       3'b000: dout<= 6'd1; // 1/32 note
       3'b001: dout<= 6'd2; // 1/16 note
       3'b010: dout<= 6'd4; // 1/8 note
       3'b011: dout<= 6'd8; // 1/4 note
       3'b100: dout<= 6'd16;   // 1/2 note
       3'b101: dout<= 6'd24;   // 3/4 note
       3'b110: dout<= 6'd32;   // 1 note
       default: dout<= 6'd1;
   endcase
endmodule
```

B. Simulation

Decoder will decode the value according to each cases.



V. Audio frequency generation divider (divo)

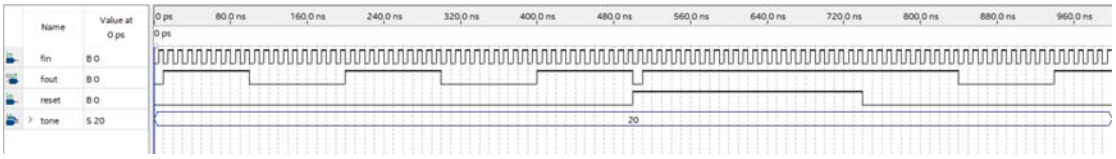
A. Verilog Code and Comment

```
module divo(fin, tone, reset, fout); //audio frequency divider
//input
input fin, reset;
input [31:0]tone;
//output
output reg fout;
//reg&wire
reg [31:0]count;
wire [31:0]divnh;

assign divnh = tone>>1; //The input tone is the division ratio retrieved from memory.

//if reset=1 or count>=1, count=1, else count+=1
always@(posedge fin or posedge reset)
begin
    if (reset == 1)
        count<=1;
    else if (count>=tone)
        count<=1;
    else
        count<=count+1;
end

//fin negedge triggered, if count<=divnh, it will be assigned to 1
always@(negedge fin)
fout = (count<=divnh)?1:0;

endmodule
```

B. Simulation

Set tone as 20 for the purpose of easier simulation, the frequency of fout is 20/2 times smaller than fin, and when reset

is set to high, fout would also be high.



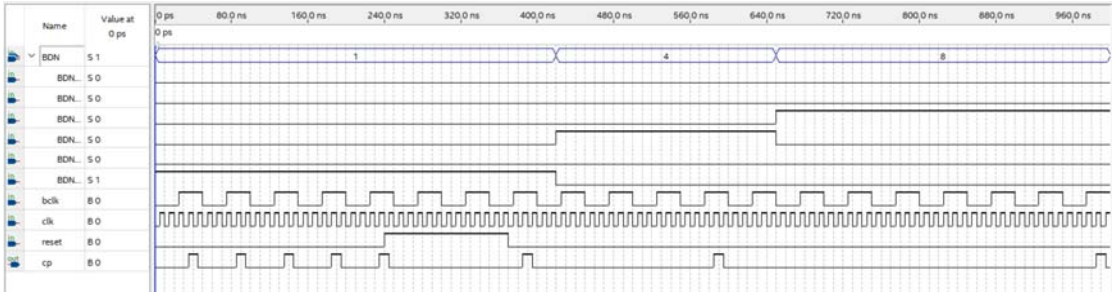
VI. Simple Clock Domain Crossing Model

A. Verilog Code and Comment

```
1   module NoteChangCtrl (reset, bclk, clk, cp, BDN);
2   input    reset, bclk, clk; //bclk is the output of beat frequency divider
3   output   reg cp;
4   input    [5:0] BDN; //the output of the decoder
5
6   reg      [5:0] count;
7   wire     _cp;
8
9   //trigger when either bclk, cp, or reset is posedge
10  //if reset or cp is 1, count=0, else count+1
11  always @(posedge bclk or posedge cp or posedge reset)begin
12      if (reset|cp)
13          count <= 0;
14      else begin
15          count <= count +1;
16      end
17  end
18
19  //when count=the output value of decoder, _cp=1, else _cp=0
20  assign _cp = (count == BDN)?1'b1:1'b0;
21
22  //clk posedge triggered, assign _cp to cp
23  always @(posedge clk)begin
24      cp <= _cp;
25  end
26
27  endmodule
28
```

B. Simulation

When reset, bclk, or cp is in posedge, then:
When bclk is 1 and neither cp nor reset is 1, count will be assigned to 1, and if count reaches the value of BDN, cp will receive the same value in the next posedge clk.
After cp = 1, count shall be set back to 0 right away, and the new value of cp will be assigned in the next posedge clk.



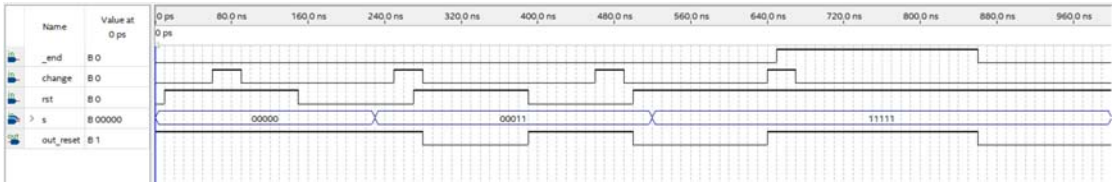
VII. Output Reset (outputre)

A. Verilog Code and Comment

```verilog
module outputre (rst, s, change, _end, out_reset);
//input
input rst,change,_end;
input [4:0] s;
//output
output out_reset;
//wire&reg
wire out_reset;

//combination of a not gate, or gate, and a nor gate
assign out_reset = (((~rst)|_end|change)|(~|s));
endmodule
```

B. Simulation

If s=0, reset=0, change=1, or _end=1, then out_reset=1.



VIII. And Data (mend, mclkand)

A. Verilog Code and Comment

mend:

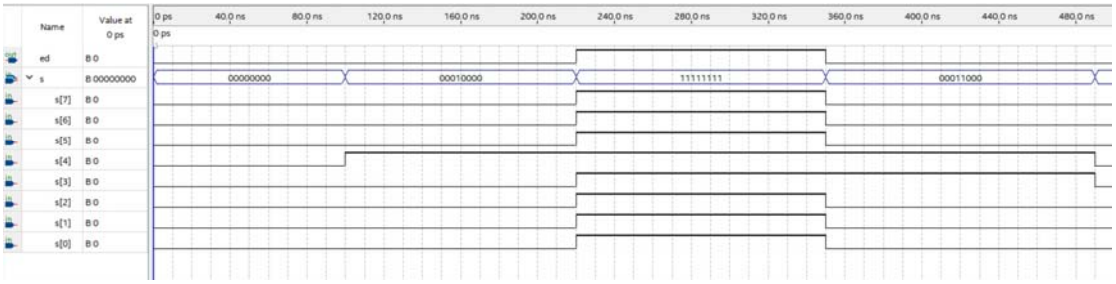
```verilog
module mend(s, ed);
//input
input [7:0]s;
//output
output wire ed;

// and all the component bits inside s
assign ed = &s;

endmodule
```

mclkand:

```verilog
module clkand(ed, ch, m_clk);
//input
input ed, ch;
//output
output wire m_clk;

// a not gate and an and gate
assign m_clk = (~ed)&ch;

endmodule
```

B. Simulation

mend:



mclkand:

IX. Reflection

This experiment was much more complex than the previous two digital electrical experiments. There were many components that needed to be written and connected. One group finished early and played music using a buzzer, for groups that cannot complete the project in time, felt that the music was a bit annoying frankly speaking.

When we were making up the experiment, we couldn’t find the reason why the music box didn’t work no matter how we looked for it. With the help of the teaching assistant, we finally found out that we forgot to connect a line to clk in our block diagram. I am grateful for the teaching assistant’s help. Being a teaching assistant in experiment class is really stressful because they have to take care of many groups at once, I respect them.