

İhsan Dogramacı Bilkent University

Department Of Computer Science

CS223 Digital Design

Final Report

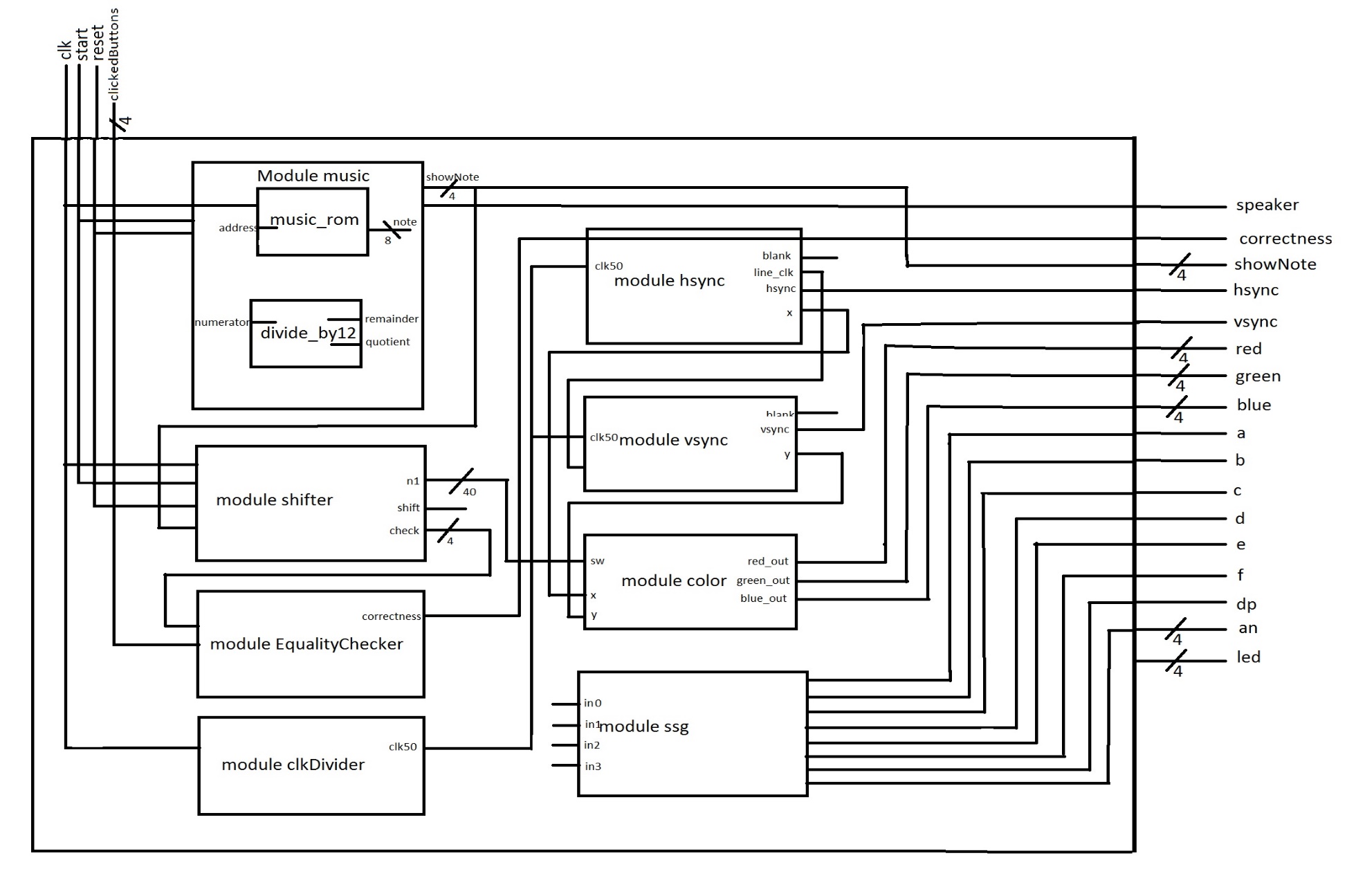
Project: FPGA Guitar HERO

Students: Ömer Faruk Ergün 21400262

Burak Sibirlioğlu 21501252

Date: December 26, 2016

# - A block diagram of the project



# Block Description

|  |  |  |
| --- | --- | --- |
| **Block Name** | **What does block do?** | **How does it do?** |
| **guitarHero** | Controls the complete game. | It binds each module together. It also counts the score of the player according to the correctness value from the equality checker module and sends it to 7segment display of Basys board. |
| **music** | It process the music operation. | It initializes and address and increases it in a time and for every address value it takes the music note and its octave value and by clock dividing process it sends the signal to the headphone jack with particular frequency. |
| **music\_ROM** | It keeps the notes in a sequence. | It takes an address as an input and returns 8bits notes value which is in sequence. And send it to divide by 12 module. It is modified from fpga4fun web page. |
| **divide\_by12** | It returns octave value off particular notes | It takes 8bits notes input and divide it by 12 and returns 4bit state of notes and its octave value. It has taken from fpga4fun web page. |
| **vsync** | Control the line number in the screen and synchronization. | Module uses counter to count the line\_clk which is created by hsync and counts the vertical position of the pixels. It creates a synchronization signal and y coordinate for color module. This code is modified from one GitHub project which is named as simple vga controller. |
| **hsync** | Controls and decide the x coordinate of the screen and decide when screen uses a new line. | Hsync module uses a counter to count the bits in the screen and when counter reaches the end of visible are of screen it creates a line\_clk output. It also sends coordinate of screen to color module. This code is modified from one GitHub project which is named as simple vga controller. |
| **color** | Color module draw and decide the color of everything on the screen. | It takes x and y coordinates of the screen and according to this coordinates it choose a color and creates 3 output which are also 4 bits (red,green,blue). |
| **clkDivider** | Vga system works with specific frequencies and divider arrange the frequency of clk. | We choose to use 800x600 for resolution of screen. According to timing diagram we need 50Mhz of clock. Divider creates this clock by using a one bit counter. |
| **ssg** | It controls the seven segment display on the Basys board. | It is taken from the instructions of lab 4. |
| **shifter** | It controls screen by using 40bits of array and change this array according to new data taken from music. | It takes 4bits input from music module and shift the array 4 times.Than it assign new array to output and also it creates another 4 bits of output which is sended to equality checker. |
| **EqualityChecker** | It compares the 4 bit signal comes from buttons and the last note combination that we have at the bottom of the screen. | Shifter module controls the screen. It sends last line of the data to checker and checker compares buttons and shifter output. |
| **Buttons** | They are used to play  the game by pushing the correct buttons on time. | They gives a 4 bit signal and this is compared with the last note combination that we have by EqualityChecker  Module. |

# - References

Hscyn and Vsync codes: Taken from <https://github.com/mstump/verilog-vga-controller> . This code was based on 640x480 pixel resolution. Therefore it is modified to 800x600 pixel resolution. And it is modified to give x and y coordinates to draw blocks on screen and decide their colors. It is used in hsycn and vsync module of the project. Reset and pause functions also added to this codes.

Divide by 12 codes: Taken from fpga4fun web page under music box project. And iit is directly used as divide\_by12 module.

Music\_Rom and music codes: Taken from fpga4fun webpage under music box project and it is modified according to Basys3 frequency and clock division logic changed. Also the code modified to take notes at particular times and returning them some functions like start stop and reset is also added.

Seven Segment Display codes: Taken from the instructions of lab 4 and directly used in ssg module.

Headphone Jack: It is bought from Konya Sokak for 2 tl and used to take stereo sound.

Buttons: They are bought from Konya sokak each one was 0.5 tl and used to take user input during game.

# Appendices

## Module Vsync

module vsync(

input line\_clk,

output vsync\_out,blank\_out,[10:0]y\_crd);

reg [10:0] count = 11'b0000000000; //veritcal line counter

reg vsync = 0;

reg blank = 0; //blank line register

always @(posedge line\_clk) //whole screen all lines

if (count < 666)

count <= count + 1;

else

count <= 0;

always @(posedge line\_clk) //visible area

if (count < 600)

blank <= 0;

else

blank <= 1;

always @(posedge line\_clk)

begin

if (count < 637) //front porch plus visible area

vsync <= 1;

else if (count >= 637 && count < 643) //invisible area

vsync <= 0;

else if (count >= 643) //retrace

vsync <= 1;

end

assign vsync\_out = vsync;

assign blank\_out = blank;

assign y\_crd = count; //y coordinate for color module

endmodule // vsync

## Module SSG

module ssg(input clk,

input [3:0]in0,in1,in2,in3, // 4 values for 4 digits (decimal value)

output a, b, c, d, e, f, g, dp, //individual LED output for the 7-segment along with the digital point

output [3:0] an // anode: 4-bit enable signal (active low)

);

// divide system clock (100Mhz for Basys3) by 2^N using a counter, which allows us to multiplex at lower speed

localparam N = 18;

logic [N-1:0] count = {N{1'b0}}; //initial value

always@ (posedge clk)

count <= count + 1;

logic [3:0]digit\_val; // 7-bit register to hold the current data on output

logic [3:0]digit\_en; //register for enable vector

always\_comb

begin

digit\_en = 4'b1111; //default

digit\_val = in0; //default

case(count[N-1:N-2]) //using only the 2 MSB's of the counter

2'b00 : //select first 7Seg.

begin

digit\_val = in0;

digit\_en = 4'b1110;

end

2'b01: //select second 7Seg.

begin

digit\_val = in1;

digit\_en = 4'b1101;

end

2'b10: //select third 7Seg.

begin

digit\_val = in2;

digit\_en = 4'b1011;

end

2'b11: //select forth 7Seg.

begin

digit\_val = in3;

digit\_en = 4'b0111;

end

endcase

end

//Convert digit number to LED vector. LEDs are active low.

logic [6:0] sseg\_LEDs;

always\_comb

begin

sseg\_LEDs = 7'b1111111; //default

case(digit\_val)

4'd0 : sseg\_LEDs = 7'b1000000; //to display 0

4'd1 : sseg\_LEDs = 7'b1111001; //to display 1

4'd2 : sseg\_LEDs = 7'b0100100; //to display 2

4'd3 : sseg\_LEDs = 7'b0110000; //to display 3

4'd4 : sseg\_LEDs = 7'b0011001; //to display 4

4'd5 : sseg\_LEDs = 7'b0010010; //to display 5

4'd6 : sseg\_LEDs = 7'b0000010; //to display 6

4'd7 : sseg\_LEDs = 7'b1111000; //to display 7

4'd8 : sseg\_LEDs = 7'b0000000; //to display 8

4'd9 : sseg\_LEDs = 7'b0010000; //to display 9

default : sseg\_LEDs = 7'b0111111; //dash

endcase

end

assign an = digit\_en;

assign {g, f, e, d, c, b, a} = sseg\_LEDs;

assign dp = 1'b1; //turn dp off

endmodule

## Module Shifter

module shifter( input clk,

input reset,start,

input logic [3:0]data, //input data that will be diplayed at first row

output [39:0]out, //output data which controls the color module

output reg[3:0]check, //data of last line that will be comparised with clickedButtons

output enable); //enable to control comparison event

reg [39:0]current=40'b0000000000000000000000000000000000000000; //will be output

reg [39:0]array=40'b0000000000000000000000000000000000000000; //array that supply continuity of shifter

reg [3:0]dtemp; //temp register which controls shifting event and prevent infinite loop

logic control; //infinite loop control variable

integer i; //for the for loop

logic shift=1'b0;

integer counter=0; //shifter inner clock implementation

always@( posedge clk )

begin

counter <= counter + 1;

if( counter == 35000000 )

counter <= 30'd0;

if( counter == 30'd0 )

shift <= 1'b1;

else

shift <= 1'b0;

end

always @(posedge shift) //shifting

begin

if((reset==1)) //if reset output are zero

begin

current=40'b0000000000000000000000000000000000000000;

array=40'b0000000000000000000000000000000000000000; end

else if(start==0)

current=array;

else begin

for(i=4; i<=39; i=i+1)

current[i-4]=array[i]; //assigning values except first four

current[36]=data[0]; //assigning first for digit

current[37]=data[1];

current[38]=data[2];

current[39]=data[3];

array=current;

dtemp=data;

end

end

assign out=current;

assign check=current[3:0]; //last line of diplay

assign enable=shift; //to control comparison event

endmodule

## Module Music Rom

module music\_ROM(

input clk,

input [7:0] address,

output reg [7:0] note

);

always @(posedge clk)

case(address)

0: note<= 8'd25;

1: note<= 8'd27;

2: note<= 8'd27;

3: note<= 8'd25;

4: note<= 8'd22;

5: note<= 8'd22;

6: note<= 8'd30;

7: note<= 8'd30;

8: note<= 8'd27;

9: note<= 8'd27;

10: note<= 8'd25;

11: note<= 8'd25;

12: note<= 8'd25;

13: note<= 8'd25;

14: note<= 8'd25;

15: note<= 8'd25;

16: note<= 8'd25;

17: note<= 8'd27;

18: note<= 8'd25;

19: note<= 8'd27;

20: note<= 8'd25;

21: note<= 8'd25;

22: note<= 8'd30;

23: note<= 8'd30;

24: note<= 8'd29;

25: note<= 8'd29;

26: note<= 8'd29;

27: note<= 8'd29;

28: note<= 8'd29;

29: note<= 8'd29;

30: note<= 8'd29;

31: note<= 8'd29;

32: note<= 8'd23;

33: note<= 8'd25;

34: note<= 8'd25;

35: note<= 8'd23;

36: note<= 8'd20;

37: note<= 8'd20;

38: note<= 8'd29;

39: note<= 8'd29;

40: note<= 8'd27;

41: note<= 8'd27;

42: note<= 8'd25;

43: note<= 8'd25;

44: note<= 8'd25;

45: note<= 8'd25;

46: note<= 8'd25;

47: note<= 8'd25;

48: note<= 8'd25;

49: note<= 8'd27;

50: note<= 8'd25;

51: note<= 8'd27;

52: note<= 8'd25;

53: note<= 8'd25;

54: note<= 8'd27;

55: note<= 8'd27;

56: note<= 8'd22;

57: note<= 8'd22;

58: note<= 8'd22;

59: note<= 8'd22;

60: note<= 8'd22;

61: note<= 8'd22;

62: note<= 8'd22;

63: note<= 8'd22;

64: note<= 8'd25;

65: note<= 8'd27;

66: note<= 8'd27;

67: note<= 8'd25;

68: note<= 8'd22;

69: note<= 8'd22;

70: note<= 8'd30;

71: note<= 8'd30;

72: note<= 8'd27;

73: note<= 8'd27;

74: note<= 8'd25;

75: note<= 8'd25;

76: note<= 8'd25;

77: note<= 8'd25;

78: note<= 8'd25;

79: note<= 8'd25;

80: note<= 8'd25;

81: note<= 8'd27;

82: note<= 8'd25;

83: note<= 8'd27;

84: note<= 8'd25;

85: note<= 8'd25;

86: note<= 8'd30;

87: note<= 8'd30;

88: note<= 8'd29;

89: note<= 8'd29;

90: note<= 8'd29;

91: note<= 8'd29;

92: note<= 8'd29;

93: note<= 8'd29;

94: note<= 8'd29;

95: note<= 8'd29;

96: note<= 8'd23;

97: note<= 8'd25;

98: note<= 8'd25;

99: note<= 8'd23;

100: note<= 8'd20;

101: note<= 8'd20;

102: note<= 8'd29;

103: note<= 8'd29;

104: note<= 8'd27;

105: note<= 8'd27;

106: note<= 8'd25;

107: note<= 8'd25;

108: note<= 8'd25;

109: note<= 8'd25;

110: note<= 8'd25;

111: note<= 8'd25;

112: note<= 8'd25;

113: note<= 8'd27;

114: note<= 8'd25;

115: note<= 8'd27;

116: note<= 8'd25;

117: note<= 8'd25;

118: note<= 8'd32;

119: note<= 8'd32;

120: note<= 8'd30;

121: note<= 8'd30;

122: note<= 8'd30;

123: note<= 8'd30;

124: note<= 8'd30;

125: note<= 8'd30;

126: note<= 8'd30;

127: note<= 8'd30;

128: note<= 8'd27;

129: note<= 8'd27;

130: note<= 8'd27;

131: note<= 8'd27;

132: note<= 8'd30;

133: note<= 8'd30;

134: note<= 8'd30;

135: note<= 8'd27;

136: note<= 8'd25;

137: note<= 8'd25;

138: note<= 8'd22;

139: note<= 8'd22;

140: note<= 8'd25;

141: note<= 8'd25;

142: note<= 8'd25;

143: note<= 8'd25;

144: note<= 8'd23;

145: note<= 8'd23;

146: note<= 8'd27;

147: note<= 8'd27;

148: note<= 8'd25;

149: note<= 8'd25;

150: note<= 8'd23;

151: note<= 8'd23;

152: note<= 8'd22;

153: note<= 8'd22;

154: note<= 8'd22;

155: note<= 8'd22;

156: note<= 8'd22;

157: note<= 8'd22;

158: note<= 8'd22;

159: note<= 8'd22;

160: note<= 8'd20;

161: note<= 8'd20;

162: note<= 8'd22;

163: note<= 8'd22;

164: note<= 8'd25;

165: note<= 8'd25;

166: note<= 8'd27;

167: note<= 8'd27;

168: note<= 8'd29;

169: note<= 8'd29;

170: note<= 8'd29;

171: note<= 8'd29;

172: note<= 8'd29;

173: note<= 8'd29;

174: note<= 8'd29;

175: note<= 8'd29;

176: note<= 8'd30;

177: note<= 8'd30;

178: note<= 8'd30;

179: note<= 8'd30;

180: note<= 8'd29;

181: note<= 8'd29;

182: note<= 8'd27;

183: note<= 8'd27;

184: note<= 8'd25;

185: note<= 8'd25;

186: note<= 8'd23;

187: note<= 8'd20;

188: note<= 8'd20;

189: note<= 8'd20;

190: note<= 8'd20;

191: note<= 8'd20;

192: note<= 8'd25;

193: note<= 8'd27;

194: note<= 8'd27;

195: note<= 8'd25;

196: note<= 8'd22;

197: note<= 8'd22;

198: note<= 8'd30;

199: note<= 8'd30;

200: note<= 8'd27;

201: note<= 8'd27;

202: note<= 8'd25;

203: note<= 8'd25;

204: note<= 8'd25;

205: note<= 8'd25;

206: note<= 8'd25;

207: note<= 8'd25;

208: note<= 8'd25;

209: note<= 8'd27;

210: note<= 8'd25;

211: note<= 8'd27;

212: note<= 8'd25;

213: note<= 8'd25;

214: note<= 8'd30;

215: note<= 8'd30;

216: note<= 8'd29;

217: note<= 8'd29;

218: note<= 8'd29;

219: note<= 8'd29;

220: note<= 8'd29;

221: note<= 8'd29;

222: note<= 8'd29;

223: note<= 8'd29;

224: note<= 8'd23;

225: note<= 8'd25;

226: note<= 8'd25;

227: note<= 8'd23;

228: note<= 8'd20;

229: note<= 8'd20;

230: note<= 8'd29;

231: note<= 8'd29;

232: note<= 8'd27;

233: note<= 8'd27;

234: note<= 8'd25;

235: note<= 8'd25;

236: note<= 8'd25;

237: note<= 8'd25;

238: note<= 8'd25;

239: note<= 8'd25;

240: note<= 8'd25;

241: note<= 8'd0;

242: note<= 8'd00;

default: note <= 8'd0;

endcase

endmodule

## Module Music

module music(

input clk,

input start,

input reset,

output reg speaker,

output reg [3:0] showNote,

output endSong

);

//extra try

reg [7:0] addr;

int counterForAdress = 3500000;

logic takeAddress = 1'b0;

int counterAdress = 0;

reg [30:0] tone;

//variables

int counter = 0;

int counter1 = 0;

int counterMax = 3000000;

logic takeNote = 1'b0;

logic clk\_en = 1'b0;

logic stop;

logic scoreIcreaseController = 1'b0;

reg [7:0] noteKeeperAtCurrentTime;

//segment display values

reg [3:0] firstDigit, secondDigit, thirdDigit, fourthDigit;

//clockdividing process for 100 mhz to 25 mhz 2

always@( posedge clk )

if(start == 1 && stop == 0 ) begin

counter1 <= counter1 + 1;

if( counter1 == 3'd4 )

counter1 <= 3'd0;

if( counter1 == 30'd0 )

clk\_en <= 1'b1;

else

clk\_en <= 1'b0;

end

//clockdividing process for taking note at particular times 1

always@( posedge clk )

if(start == 1 && stop == 0 && clk\_en == 1) begin

counter <= counter + 1;

if( counter == counterMax )

counter <= 30'd0;

if( counter == 30'd0 )

takeNote <= 1'b1;

else

takeNote <= 1'b0;

end

//clockdividing process for taking note at particular times 1

always@( posedge clk )

if(start == 1 && stop == 0 && clk\_en == 1) begin

counterAdress <= counterAdress + 1;

if( counterAdress == counterForAdress )

counterAdress <= 30'd0;

if( counterAdress == 30'd0 )

takeAddress <= 1'b1;

else

takeAddress <= 1'b0;

end

//////////////////////////

//if time is came, take next address

always @(posedge clk)

if(reset == 0) begin

if(clk\_en == 1'b1 && start == 1 && stop == 0 && takeAddress == 1'b1)begin addr <= addr+ 8'd1; end //

end

else

addr <= 8'd0;

////////////////////////

always\_comb

if(addr== 8'd242)

stop = 1'b1;

else

stop = 1'b0;

//teking full note at particular adress

wire [7:0] fullnote;

music\_ROM get\_fullnote(.clk(clk\_en), .address(addr), .note(fullnote));

//taking octave value and 4 bit note representation

wire [2:0] octave;

wire [3:0] note;

reg [3:0] goingNote;

divide\_by12 get\_octave\_and\_note(.numerator(fullnote[5:0]), .quotient(octave), .remainder(note));

//passing current note to takeNoteBit

always @(posedge clk)

if(takeNote == 1'b1 && start == 1 && stop == 0 && clk\_en == 1)begin

showNote <= note;

goingNote <= showNote;

end

//determining different clockdividers for different notes to give signal to speaker with different frekanss

reg [8:0] clkdivider;

always @\*

case(note)

0: clkdivider = 9'd511;//A

1: clkdivider = 9'd482;// A#/Bb

2: clkdivider = 9'd455;//B

3: clkdivider = 9'd430;//C

4: clkdivider = 9'd405;// C#/Db

5: clkdivider = 9'd383;//D

6: clkdivider = 9'd361;// D#/Eb

7: clkdivider = 9'd341;//E

8: clkdivider = 9'd322;//F

9: clkdivider = 9'd303;// F#/Gb

10: clkdivider = 9'd286;//G

11: clkdivider = 9'd270;// G#/Ab

default: clkdivider = 9'd0;

endcase

//to give to speaker a paticular note by clock dividing process

reg [8:0] counter\_note;

reg [7:0] counter\_octave;

always @(posedge clk) if(clk\_en == 1'b1 && start == 1 && stop == 0)begin counter\_note <= counter\_note==0 ? clkdivider : counter\_note-9'd1; end

always @(posedge clk) if(clk\_en == 1'b1 && start == 1 && stop == 0)begin if(counter\_note==0) counter\_octave <= counter\_octave==0 ? 8'd255 >> octave : counter\_octave-8'd1; end

always @(posedge clk) if(clk\_en == 1'b1 && start == 1 && stop == 0)begin if(counter\_note==0 && counter\_octave==0 && fullnote!=0) speaker <= ~speaker; end

//working the equalith checker method the check if button input is equal or not to the current note and increase or decrease the score

assign endSong=stop;

endmodule

## Module Hsync

module hsync(

input clk50,

output hsync\_out, blank\_out, newline\_out,[10:0]x\_crd);

reg [10:0] count = 11'b0000000000; //pixel counter horizontal

reg hsync = 0; //sync variable

reg blank = 0; //blannk variable

reg newline = 0; //newline variable to control vertical sync

always @(posedge clk50) //whole screen

begin

if (count < 1040)

count <= count + 1;

else

count <= 0;

end

always @(posedge clk50) //when one line of screen passed

begin

if (count == 0)

newline <= 1;

else

newline <= 0;

end

always @(posedge clk50) //visible area

begin

if (count >= 800)

blank <= 1;

else

blank <= 0;

end

always @(posedge clk50)

begin

if (count < 856) // pixel data plus front porch

hsync <= 1;

else if (count >= 856 && count < 976) //invisible area

hsync <= 0;

else if (count >= 976) //retrace

hsync <= 1;

end // always @ (posedge clk50)

assign hsync\_out = hsync;

assign blank\_out = blank;

assign newline\_out = newline;

assign x\_crd = count; //x coordinate for color module

endmodule // hsync

## Top Module Guitar Hero

module guitarHero(

input clk, start, reset,

input [3:0] clickedButtons,

output reg speaker,correctness,

output reg [3:0] showNote,

output logic hsync, vsync,

[3:0] red, blue, green,

output a, b, c, d, e, f, g, dp, //7 segment display controls

output [3:0] an, //7 segment display controls

output [3:0]led); //leds for notes

reg [12:0] score = 0; //player score

wire line\_clk, blank, hblank, vblank, clk50; //vga controls and vga clock

logic [10:0]x,y; //xand y coordinates of screen pixel

logic [10:0]y1=11'd0; //starting point for boxes

logic [10:0]y2=11'd50; //ending point for first box

logic shift; //shifter output timing

reg [3:0]check; //check register

wire [39:0]n1; //vga data taht will be displayed

music msc(clk,start,reset,speaker,showNote,stop); //music module

shifter shifting(clk,reset,start,showNote,n1,check,shift); //shifter module to move boxes downwards

EqualityChecker checkButtonsandNoteEquality(check,clickedButtons,correctness); //equality checker

always @(posedge clk) //score updater

if(reset == 0 && stop == 0)begin

if(shift == 1'b1 && start == 1 )begin

if(correctness == 1)

score = score + 1'b1; //increase score

else

score = score - 1'b1; //decrease score

end

end

else

score = 0; //if reset or not start score=0

logic [3:0]in1,in2,in3,in0; //seven segment diplay inputs

always @(posedge shift) //score to 4 digit for diplaying in seven segment

begin

in0=score%10;

in1=(score/10)%10;

in2=(score/100)%10;

in3=(score/1000)%10;

end

ssg seven(clk,in0,in1,in2,in3,a, b, c, d, e, f, g, dp,an); //seven segment module

clkDivider clock (clk,clk50); //clock divider for vga sync

hsync hs(clk50, hsync, hblank, line\_clk,x); //horizontal

vsync vs(line\_clk, vsync, vblank,y); //vertical

color cg(clk50,x,y,y1,y2,n1 ,red, green, blue); //darawing and painting boxes

assign blank = hblank || vblank; //if one of the sync outputs are blank it show screen blank

assign led=clickedButtons; //cliked buttons to leds to check

endmodule // guitarHero

## Module Equality Checker

module EqualityChecker(

input [3:0] comingNote,

input [3:0] clickedButtons,

output reg result

);

//Debouncer Process

/\* logic bouncer0 = 1'b0;

logic bouncer1 = 1'b0;

logic bouncer2 = 1'b0;

logic bouncer3 = 1'b0;

logic [3:0]sendButtonStates;

//nextstate process

always @(bouncer0,bouncer1,bouncer2,bouncer3,clickedButtons,sendButtonStates)

begin

//FirstButton

if(clickedButtons[0] == 1'b1 && bouncer0 == 1'b0)begin

sendButtonStates[0] <= 1'b1;

bouncer0 <= 1'b0;

end

else if(clickedButtons[0] == 1'b0 && bouncer0 == 1'b1)begin

bouncer0 <= 1'b0;

//sendButtonStates[0] <= 1'b0;

end

//SecondButton

if(clickedButtons[1] == 1'b1 && bouncer1 == 1'b0)begin

sendButtonStates[1] <= 1'b1;

bouncer1 <= 1'b0;

end

else if(clickedButtons[1] == 1'b0 && bouncer1 == 1'b1)begin

bouncer1 <= 1'b0;

//sendButtonStates[1] <= 1'b0;

end

//ThirdButton

if(clickedButtons[2] == 1'b1 && bouncer2 == 1'b0)begin

sendButtonStates[2] <= 1'b1;

bouncer2 <= 1'b0;

end

else if(clickedButtons[2] == 1'b0 && bouncer2 == 1'b1)begin

bouncer2 <= 1'b0;

//sendButtonStates[2] <= 1'b0;

end

//FourthtButton

if(clickedButtons[3] == 1'b1 && bouncer3 == 1'b0)begin

sendButtonStates[3] <= 1'b1;

bouncer3 <= 1'b0;

end

else if(clickedButtons[3] == 1'b0 && bouncer3 == 1'b1)begin

bouncer3 <= 1'b0;

//sendButtonStates[3] <= 1'b0;

end

end \*/

//clockdividing process

always@(\*)

begin

//if( comingNote == sendButtonStates)

if( comingNote == clickedButtons)

result <= 1'b1;

else

result <= 1'b0;

end

endmodule

## Module Divide by 12

module divide\_by12(

input [5:0] numerator, // value to be divided by 12

output reg [2:0] quotient,

output [3:0] remainder

);

reg [1:0] remainder3to2;

always @(numerator[5:2])

case(numerator[5:2])

0: begin quotient=0; remainder3to2=0; end

1: begin quotient=0; remainder3to2=1; end

2: begin quotient=0; remainder3to2=2; end

3: begin quotient=1; remainder3to2=0; end

4: begin quotient=1; remainder3to2=1; end

5: begin quotient=1; remainder3to2=2; end

6: begin quotient=2; remainder3to2=0; end

7: begin quotient=2; remainder3to2=1; end

8: begin quotient=2; remainder3to2=2; end

9: begin quotient=3; remainder3to2=0; end

10: begin quotient=3; remainder3to2=1; end

11: begin quotient=3; remainder3to2=2; end

12: begin quotient=4; remainder3to2=0; end

13: begin quotient=4; remainder3to2=1; end

14: begin quotient=4; remainder3to2=2; end

15: begin quotient=5; remainder3to2=0; end

endcase

assign remainder[1:0] = numerator[1:0]; // the first 2 bits are copied through

assign remainder[3:2] = remainder3to2; // and the last 2 bits come from the case statement

endmodule

## Module Color

module color(

input clk,

input [10:0]x,y,y1,y2, //starting and ending points of boxes

input [39:0]sw, //first data that enters the shifter it will be 40bits zero

output [3:0] red\_out, green\_out, blue\_out); //rgb outputs

logic [2:0]rgb=3'b100; //initialization of rgb

always

begin

if((y>=y1)&&(y<=y2)) //first line

case(sw[39:36])

4'b1000:if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000;

4'b0100:if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000;

4'b0010:if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000;

4'b0001:if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else rgb=3'b000;

4'b1100:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000; end

4'b1110:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1111:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100;else if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else rgb=3'b000; end

4'b1010:begin if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b0110:begin if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1001:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b0101:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000; end

4'b0011:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1101:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b1011:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b0111:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000;end

4'b0000:rgb=3'b000;

endcase

else if((y>=(y1+60))&&(y<=(y2+60))) //second line

case(sw[35:32])

4'b1000:if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000;

4'b0100:if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000;

4'b0010:if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000;

4'b0001:if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else rgb=3'b000;

4'b1100:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000; end

4'b1110:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1111:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100;else if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else rgb=3'b000; end

4'b1010:begin if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b0110:begin if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1001:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b0101:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000; end

4'b0011:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1101:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b1011:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b0111:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000;end

4'b0000:rgb=3'b000;

endcase

else if((y>=(y1+120))&&(y<=(y2+120))) // third line

case(sw[31:28])

4'b1000:if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000;

4'b0100:if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000;

4'b0010:if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000;

4'b0001:if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else rgb=3'b000;

4'b1100:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000; end

4'b1110:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1111:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100;else if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else rgb=3'b000; end

4'b1010:begin if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b0110:begin if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1001:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b0101:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000; end

4'b0011:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1101:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b1011:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b0111:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000;end

4'b0000:rgb=3'b000;

endcase

else if((y>=(y1+180))&&(y<=(y2+180))) //fourth line

case(sw[27:24])

4'b1000:if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000;

4'b0100:if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000;

4'b0010:if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000;

4'b0001:if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else rgb=3'b000;

4'b1100:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000; end

4'b1110:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1111:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100;else if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else rgb=3'b000; end

4'b1010:begin if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b0110:begin if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1001:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b0101:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000; end

4'b0011:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1101:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b1011:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b0111:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000;end

4'b0000:rgb=3'b000;

endcase

else if((y>=(y1+240))&&(y<=(y2+240))) //fifth line

case(sw[23:20])

4'b1000:if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000;

4'b0100:if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000;

4'b0010:if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000;

4'b0001:if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else rgb=3'b000;

4'b1100:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000; end

4'b1110:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1111:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100;else if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else rgb=3'b000; end

4'b1010:begin if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b0110:begin if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1001:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b0101:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000; end

4'b0011:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1101:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b1011:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b0111:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000;end

4'b0000:rgb=3'b000;

endcase

else if((y>=(y1+300))&&(y<=(y2+300))) //sixth line

case(sw[19:16])

4'b1000:if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000;

4'b0100:if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000;

4'b0010:if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000;

4'b0001:if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else rgb=3'b000;

4'b1100:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000; end

4'b1110:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1111:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100;else if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else rgb=3'b000; end

4'b1010:begin if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b0110:begin if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1001:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b0101:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000; end

4'b0011:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1101:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b1011:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b0111:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000;end

4'b0000:rgb=3'b000;

endcase

else if((y>=(y1+360))&&(y<=(y2+360))) //seven line

case(sw[15:12])

4'b1000:if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000;

4'b0100:if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000;

4'b0010:if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000;

4'b0001:if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else rgb=3'b000;

4'b1100:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000; end

4'b1110:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1111:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100;else if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else rgb=3'b000; end

4'b1010:begin if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b0110:begin if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1001:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b0101:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000; end

4'b0011:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1101:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b1011:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b0111:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000;end

4'b0000:rgb=3'b000;

endcase

else if((y>=(y1+420))&&(y<=(y2+420))) //eightth line

case(sw[11:8])

4'b1000:if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000;

4'b0100:if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000;

4'b0010:if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000;

4'b0001:if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else rgb=3'b000;

4'b1100:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000; end

4'b1110:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1111:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100;else if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else rgb=3'b000; end

4'b1010:begin if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b0110:begin if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1001:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b0101:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000; end

4'b0011:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1101:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b1011:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b0111:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000;end

4'b0000:rgb=3'b000;

endcase

else if((y>=(y1+480))&&(y<=(y2+480))) //nineth line

case(sw[7:4])

4'b1000:if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000;

4'b0100:if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000;

4'b0010:if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000;

4'b0001:if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else rgb=3'b000;

4'b1100:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000; end

4'b1110:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1111:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100;else if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else rgb=3'b000; end

4'b1010:begin if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b0110:begin if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1001:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b0101:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000; end

4'b0011:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b1101:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b000; end

4'b1011:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b000; end

4'b0111:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b000;end

4'b0000:rgb=3'b000;

endcase

else if((y>=(y1+484))&&(y<=(y2+486))) //wihite line before the last line of screen which is the playing zone

rgb=3'b111;

else if((y>=(y1+540))&&(y<=(y2+540))) //tenth line

case(sw[3:0])

4'b1000:if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b111;

4'b0100:if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b111;

4'b0010:if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b111;

4'b0001:if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else rgb=3'b111;

4'b1100:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b111; end

4'b1110:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b111; end

4'b1111:begin if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100;else if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else rgb=3'b111; end

4'b1010:begin if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b111; end

4'b0110:begin if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b111; end

4'b1001:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b111; end

4'b0101:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b111; end

4'b0011:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b111; end

4'b1101:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else rgb=3'b111; end

4'b1011:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd100)&&(x<=11'd250)) rgb=3'b001; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else rgb=3'b111; end

4'b0111:begin if((x>=11'd550)&&(x<=11'd700)) rgb=3'b011; else if((x>=11'd400)&&(x<=11'd550)) rgb=3'b100; else if((x>=11'd250)&&(x<=11'd400)) rgb=3'b010; else rgb=3'b111;end

4'b0000:rgb=3'b111; //sides of playing zoen is white for recognition

endcase

else

rgb=3'b000;

end

assign red\_out[0] = rgb[0]; //assigning outputs for each four port of basys

assign red\_out[1] = rgb[0];

assign red\_out[2] = rgb[0];

assign red\_out[3] = rgb[0];

assign green\_out[0] = rgb[2];

assign green\_out[1] = rgb[2];

assign green\_out[2] = rgb[2];

assign green\_out[3] = rgb[2];

assign blue\_out[0] = rgb[1];

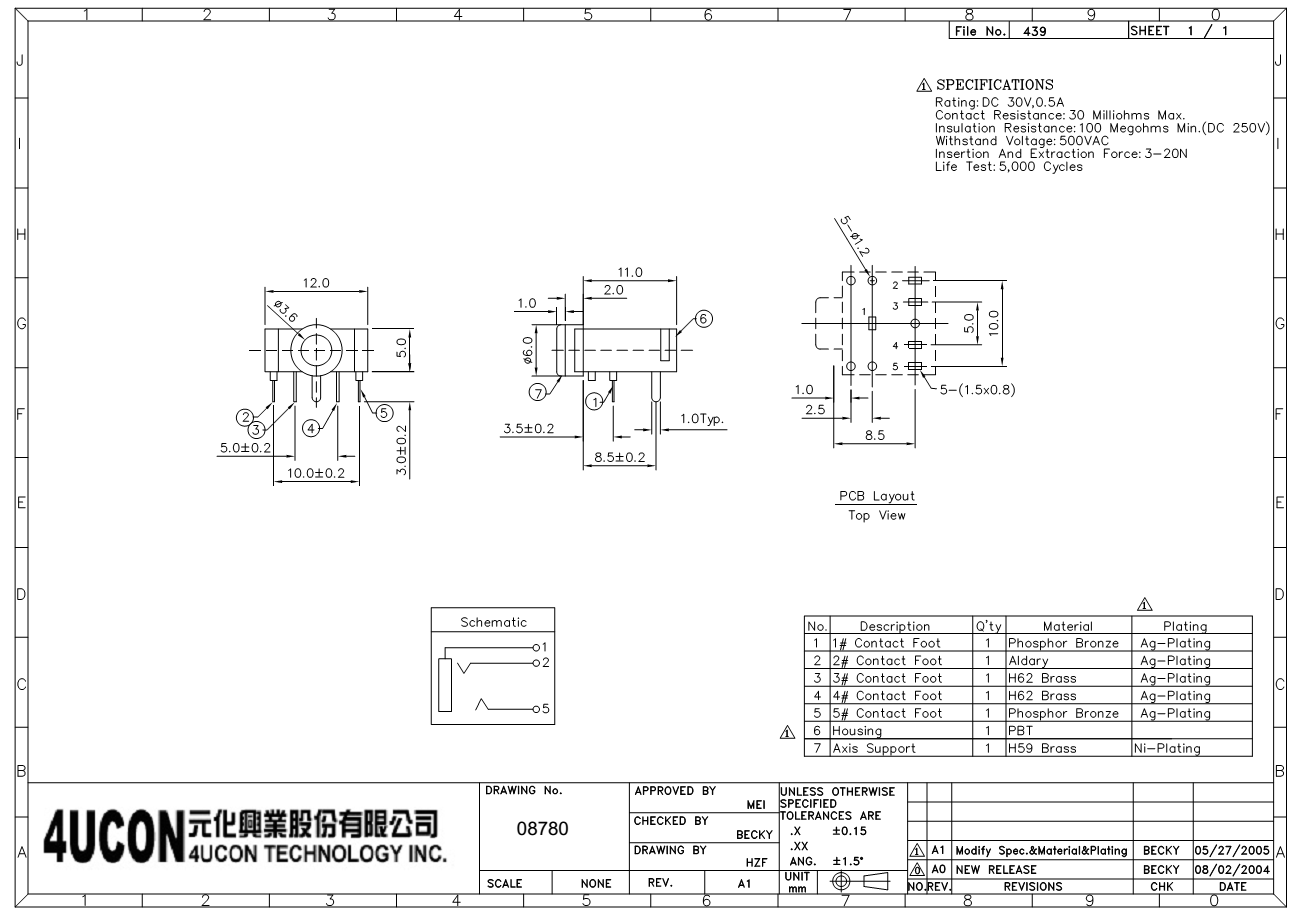
assign blue\_out[1] = rgb[1];

assign blue\_out[2] = rgb[1];

assign blue\_out[3] = rgb[1];

endmodule // color

## Data Sheet of Headphone Jack



## Data Sheet of Button

