Bilkent University

Computer Engineering

CS223 – Digital Design

Note Master

Muhammad Hamza Khan, 21402885, 01

Muhammad Usman, 21402827, 01

22nd December, 2015

5 LEDs on BASYS2

4 Push buttons on BASYS2

Audio Module

7-segment-Display

4bit Anode 7bit Cathode 1bit speaker

mainModule

clk Nclk(25MHz)

clockDivider

1bit Nclk(25MHz) 1bit speaker

music

Nclk 8bit address 8bit note 6bit numerator 3bit quotient 4bit remainder

divide\_by12

music\_ROM

**Functions of blocks:**

* **7-segment-Display:** Display a number at a random anode.
* **Audio Module:** Produces a sound.
* **4 Push buttons on BASYS2:** Press a button in accordance with the position of the lit anode of the display.
* **mainModule:** displays the required pattern on the 7 segment display and plays the sound if correct input from the push buttons is detected.
* **clockDivider:**  reduces the frequency of the clock from 50MHz to 25MHz.
* **music:** plays a tune.
* **divide\_by12:** The "divide by 12" module takes a 6 bits value (numerator) and divides it by 12 (denominator). That gives us a 3 bits quotient and a 4 bits remainder.
* **music\_ROM:** ROM to hold the list of notes to be played by the music module.

**Descriptions of blocks:**

* **mainModule:** This module has 4 button inputs, b1, b2 , b3, b4. If any of the button is pressed, one of the four LED ouputs, l1, l2, l3 ,l4 are lighted up, l1 for b1, l2 for b2, l3 for b3 and l4 for b4. This helps to determine that if the button is being pressed or not. The ouputs are CA,CB,CC,CD,CE,CF,CG,AN3,AN2,AN1,AN0.

The CA,CB,CC,CD,CE,CF,CG are used to light up certain positions on the 7 -segment display and by doing so, four numbers, 1 ,2 ,3 or 4 are displayed on one of the positions of the display. Selection of the part on the 7- segment is done through the use of AN3,AN2,AN1,AN0. AN0 for displaying in the first position, AN1 for the second, AN2 for the third and AN3 for the fourth.

"press". This output becomes high when the correct button is pressed. We check this by comparing two variables, "check" and "bt\_check". "check" variable is used in the 7-segment FSM. It has the value of 1 when the digit is being displayed in the first position, 2 in the second position, 3 in the 3rd position and 4 in the fourth position. "bt\_check" is used in the button FSM. It has the value of 1 when button 1 is pressed, 2 when button 2 is pressed, 3 when button 3 is pressed and 4 when button 4 is pressed. As our game wants the user to press the button which is corresponding to the LED being lighted up e.g button 1 when 1st positon of LED is lighted up, we compare the values of check and bt\_check and if they are equal, press is set to high. "aud\_state" and "sound". If "press" is detected to be high, the audio FSM sets the aud\_state to aud1 where "sound" is set to high and thus the sound is outputted from the audio module.

* **clockDivider:**  Uses a counter to count delay the clock and hence reducing the frequency of the clock from 50MHz to 25MHz.
* **music:** We need a way to play notes, like on a keyboard. If we use 6 bits to encode a note, we can get 64 notes. There are 12 notes per octaves, so 64 notes gives us over 5 octaves, more than enough for a little tune. To play a range of increasing notes, we instantiate a 28 bits counter, from which we extract the 6 most significant bits, to give us the 6 bits of the note we want to play. With a 25Mhz clock, each note lasts 167ms and it takes 10.6s to play all 64 notes. We divide the "fullnote" by 12. That gives us the octave (5 octaves, so 3 bits are enough, since it goes from 0 to 4) and the note (from 0 to 11, so 4 bits). You can see that we instantiate a sub-module called "divide\_by12" which takes care of the division. To go from one octave to the next, frequency is multiplied by "2". To go from one note to the next, frequency is multiplied by "1.0594". And in order to accomplish this, we use a look-up table with pre-calculated values. We divide the main clock by 512 for note A, by 483 for note A#, by 456 for note B. Every time "counter\_note" equals 0, that gives us a tick for the next stage: the octave divider. For the lowest octave, we divide "counter\_note" by 256. For octave 1, we divide by 128, and so on.
* **divide\_by12:** To divide by 12, we first divide by 4, then by 3.  
  Dividing by 4 is trivial: we remove 2 bits out of the numerator, and copy it to the remainder. So we are left with 6-2=4 bits to divide by the value "3". That's easily done with a lookup table

**References:**

music, divide\_by12, and music\_ROM modules from <http://www.fpga4fun.com/MusicBox4.html>

**Appendices:**

**mainModule:**

//Module which displays the required pattern on the 7 segment display and plays the sound if correct input is detected

module display(clk,rst,b1,b2,b3,b4,CA,CB,CC,CD,CE,CF,CG,AN3,AN2,AN1,AN0,l1,l2,l3,l4,press,sound,aud\_state);

//declaring the inputs and outputs

input clk,rst,b1,b2,b3,b4;

output aud\_state;

output reg CA,CB,CC,CD,CE,CF,CG,AN3,AN2,AN1,AN0,l1,l2,l3,l4,press,sound;

//the audio finite state machine which plays the sound if the correct input is detected

reg[1:0] aud\_state, aud\_nextState; // 2 states for the audio FSM

parameter aud0 = 0, aud1 = 01, aud2= 10, aud\_select = 1;

clockDivider sc1( clk, aud\_select, newClk); // newClk is 25 MHz

music music\_one( newClk, speaker);

always@(posedge clk)

aud\_state <= aud\_nextState;

reg [26:0] aud\_counter;

//always procedure block for the next state logic

always@(press, aud\_state, aud\_counter)

case(aud\_state)

aud0: if(press)

aud\_nextState = aud1;

else

aud\_nextState = aud0;

aud1: if(aud\_counter < 75000000)

aud\_nextState = aud1;

else

aud\_nextState = aud0;

default: aud\_nextState = aud0;

endcase

//always procedure block for the combinational logic

always@(posedge clk)

case(aud\_state)

aud0: begin

sound = 0;

aud\_counter = 0;

end

aud1: begin

sound = speaker; aud\_counter = aud\_counter + 1;

end

endcase

// The button finite state machine which takes the input from the user

// register declaration

reg X;

reg [25:0] counter;

reg [3:0] select;

reg [3:0] bt,nextbt;

reg [2:0] bt\_check;

//4 states for the button FSM

parameter B0 = 0, B1 = 1, B2 = 2, B3 = 3, B4 = 4;

//The combinational logic for the button FSM. It makes the corresponding LED of the button //high if an input is detected and assigns a value to the output bt\_check

//which is used to check if the button being pressed is the correct button or not.

always@(bt)

begin

case(bt)

B0:

begin

l1 <= 0;

l2 <= 0;

l3 <= 0;

l4 <= 0;

bt\_check <= 0;

end

B1:

begin

l1 <= 1;

l2 <= 0;

l3 <= 0;

l4 <= 0;

bt\_check <= 1;

end

B2:

begin

l1 <= 0;

l2 <= 1;

l3 <= 0;

l4 <= 0;

bt\_check <= 2;

end

B3:

begin

l1 <= 0;

l2 <= 0;

l3 <= 1;

l4 <= 0;

bt\_check <= 3;

end

B4:

begin

l1 <= 0;

l2 <= 0;

l3 <= 0;

l4 <= 1;

bt\_check <= 4;

end

endcase

end

// Next state logic for the button FSM

always@(b1,b2,b3,b4)

begin

if(b1 == 1)

nextbt <= B1;

if(b2 == 1)

nextbt <= B2;

if(b3 == 1)

nextbt <= B3;

if(b4 == 1)

nextbt <= B4;

if(b1 == 0 && b2 == 0 && b3 == 0 && b4 ==0)

nextbt <= B0;

end

always@(posedge clk)

begin

if(rst == 1)

bt <= B0;

else

bt <= nextbt;

end

// Slow Clock generator foer dipslaying the patterns on the 7-segment LED Display. It has a period of 1 sec and frequecny of 1MHZ.

always @(posedge clk)

begin

if(counter < 50000000)

begin

counter <= counter + 1;

X <= 0;

end

else

begin

counter <= 26'b0;

X <= 1;

select = select + 1;

end

end

// FSM for displaying the patterns on the 7-segment LED Display.

//4 states for showing 4 different patterns on the 7-segment display

parameter S1 = 1, S2 = 2, S3 = 3, S4 = 4;

//Declaring the variables

reg [3:0]state, nextstate;

reg [2:0]check;

//Setting the initial state when the system is turned on.

initial

state <= S1;

//Combinational logic for the Display FSM which sets different cathodes and anodes high for the different states of the Display FSM.

always @(state)

begin

case(state)

S1:

begin

CA <= 1;

CB <= 0;

CC <= 0;

CD <= 1;

CE <= 1;

CF <= 1;

CG <= 1;

AN3 <= 1;

AN2 <= 1;

AN1 <= 1;

AN0 <= 0;

check <= 1;

end

S2:

begin

CA <= 0;

CB <= 0;

CC <= 0;

CD <= 1;

CE <= 0;

CF <= 0;

CG <= 1;

AN3 <= 1;

AN2 <= 1;

AN1 <= 0;

AN0 <= 1;

check <= 2;

end

S3:

begin

CA <= 0;

CB <= 0;

CC <= 0;

CD <= 0;

CE <= 1;

CF <= 1;

CG <= 0;

AN3 <= 1;

AN2 <= 0;

AN1 <= 1;

AN0 <= 1;

check <= 3;

end

S4:

begin

CA <= 1;

CB <= 0;

CC <= 0;

CD <= 1;

CE <= 1;

CF <= 0;

CG <= 0;

AN3 <= 0;

AN2 <= 1;

AN1 <= 1;

AN0 <= 1;

check <= 4;

end

endcase

end

//Next state logic for the display FSM which selects which state to go in next

always@(select)

begin

case(select)

4'b0000 : nextstate <= S1;

4'b0001 : nextstate <= S3;

4'b0010 : nextstate <= S4;

4'b0011 : nextstate <= S2;

4'b0100 : nextstate <= S4;

4'b0101 : nextstate <= S1;

4'b0110 : nextstate <= S2;

4'b0111 : nextstate <= S4;

4'b1000 : nextstate <= S3;

4'b1001 : nextstate <= S2;

4'b1011 : nextstate <= S4;

4'b1100 : nextstate <= S1;

4'b1101 : nextstate <= S4;

4'b1110 : nextstate <= S2;

4'b1111 : nextstate <= S1;

endcase

end

always@ (posedge X)

begin

if(rst == 1)

state <= S1;

else

state <= nextstate;

end

// FSM for checking whether the correct button is pressed or not and deciding the corresponding output

//2 states for the FSM. P0 for no ouput and P1 for playing the sound.

parameter P0 = 0, P1 = 1;

//Declaring the variables

reg p,p\_next;

//Setting the initial state of the FSM.

initial

p <= P0;

//Combinational logic for the FSM

always@(p)

begin

case(p)

P0: press <= 0;

P1: press <= 1;

endcase

end

//Next state logic for the FSM

always@(state,bt)

begin

if(check == bt\_check)

p\_next <= P1;

else

p\_next <= P0;

end

always@(posedge clk)

begin

if(rst == 1)

p <= P0;

else

p <= p\_next;

end

endmodule

//End of module

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**clockDivider:**

module clockDivider(clk, Nclk);

input clk;

output Nclk;

parameter clkdivider = 50000000/25000000/2; // clk-frequency/Nclk-frequency/2: sets Nclk to 25MHz

reg [25:0] counter;

always @(posedge clk)

if(counter==0)

counter <= clkdivider-1;

else

counter <= counter-1;

reg Nclk;

always @(posedge clk)

if(counter==0)

Nclk <= ~Nclk;

endmodule

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**music:**

module music(

input clk,

output reg speaker

);

reg [30:0] tone;

always @(posedge clk) tone <= tone+31'd1;

wire [7:0] fullnote;

music\_ROM get\_fullnote(.clk(clk), .address(tone[29:22]), .note(fullnote));

wire [2:0] octave;

wire [3:0] note;

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

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

reg [8:0] counter\_note;

reg [7:0] counter\_octave;

always @(posedge clk) counter\_note <= counter\_note==0 ? clkdivider : counter\_note-9'd1;

always @(posedge clk) if(counter\_note==0) counter\_octave <= counter\_octave==0 ? 8'd255 >> octave : counter\_octave-8'd1;

always @(posedge clk) if(counter\_note==0 && counter\_octave==0 && fullnote!=0 && tone[21:18]!=0) speaker <= ~speaker;

endmodule

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**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

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**divide\_by12:**

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

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***