Logic Circuits Project, Dr.Movahedian A.

Mohammad Parsa Dini - 400101204

June 12, 2024

1 Introduction and Objectives

In this project the aim is to design a Heart beat Monitor which will receive a signal of heart beat pulse and estimates the number of heart beats per minutes. The output must be showed using an LED, a buzzer and 3 seven segments.

2 Some calculations

2.1 Calculating the clock frequency

In the first place we want to set the margin of error of beats less than or equal to 1 beats for$220\frac{beats}{min}$. So the corresponding error of BPM must be less than equal to 1. let f be the frequency of clock and T be the corresponding period. we will obtain that:
$\frac{60}{220} \geq n_{220} \times T$ and $\frac{60}{219} \geq n_{219} \times T$
Therefore we can deduce that: $f(\frac{60}{219} - \frac{60}{220}) \geq n_{219} - n_{220} \geq 1$

$$f \geq \frac{1}{\frac{60}{219} - \frac{60}{220}} = 803Hz \approx 1kHz \tag{1}$$

2.2 calculating the number of desired bits for $30\frac{beats}{min}$

by setting $N = 30\frac{beats}{min}$ we will get:
$\frac{60}{30} \geq n_{30} \times T = n_{30}\frac{1}{f} \approx \frac{n_{30}}{1000}$ which leads to $n_{30} = 2000$. Thus, the number of desired bits for getting $30\frac{beats}{min}$ is :

$$N_{30} = \log_2 n_{30} = 11bits \tag{2}$$

3 Designing the circuit's hardware

We are given a signal of heart beat for which the signal is active high whenever a beat happens and it is active high for $10ms$, after this pulse of length $10ms$, the signal will be active low until the next beat occurs.
Here, we will receive the input signal and using a reset signal we will measure the period (time between posedges in perpetuity) by counting the number of posedges of clock in this interval. And at the end we will reset the system again...
Therefore, we will need to design an asynchronous circuit that plays the role of the reset signal which is so thin but thick enough to make tangible differences.

So, in order to design a circuit for reset signal, we will define states as we did before and we will derive its flow table as:

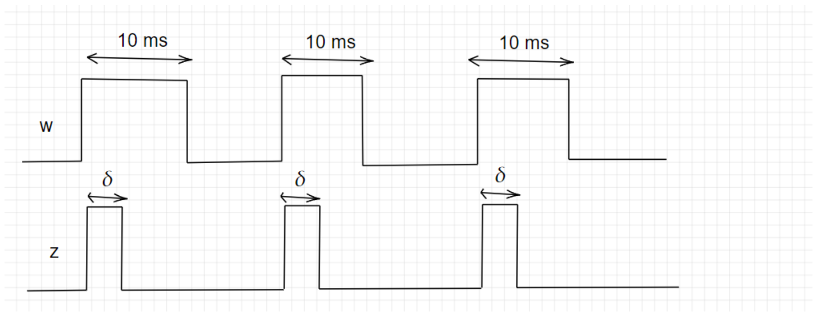

Figure 1: Caption

Present state	next state, $w = 0$	next state, $w = 1$	z (reset)
A	A	B	0
B	-	C	1
C	A	C	0

we can easily notice that the state diagram will have the "race", since each state us connected to other states in the state diagram of the circuit. Thus, in order to do the state assignment, we will need another state to mediate between state A and D, so that the hamming distance of connected nodes is 1. Thus we will obtain:

Present state	next state, $w = 0$	next state, $w = 1$	z (reset)
A	A	B	0
B	-	D	1
C	A	C	0
D	-	C	x

Now let y_1,y_2 be the feedback of Y_1,Y_2 to the circuit were w is the input(heart beat pulse) and z is the output(reset signal). Now we will draw the K-Map of these variables:

K-map of Y_1		
y_1y_2	$w = 0$	$w = 1$
00	0	0
01	x	1
11	0	1
10	x	1

Which will suggest that :

$$Y_1 = \bar{y_1}y_2 + wy_1 \tag{3}$$

K-map of Y_2		
y_1y_2	$w = 0$	$w = 1$
00	0	1
01	x	1
11	0	0
10	x	0

Which will suggest that :

$$Y_2 = \bar{y_1}w \tag{4}$$

K-map of z		
y_1	$y_2 = 0$	$y_2 = 1$
0	0	1
1	x	0

Which will suggest that :

$$z = \bar{y_1}y_2 \tag{5}$$

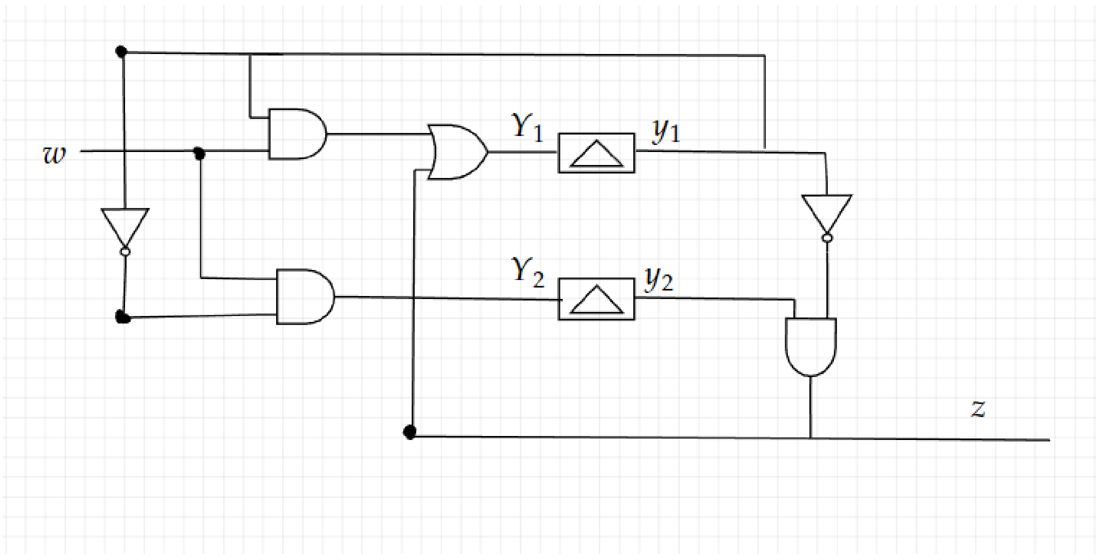

Figure 2: Caption

Now that we have Y_1, Y_2, z in terms of y_1, y_2, w, we can design the circuit as below:

Finally, by getting the reset signal, we will connect it to the reset pin out of the "11" bit counter (which counts the number of posedges in a period of heart beat: N) and to the Enable beat of the "EPROM" which will map this so called N to its corresponding heart beat in $\frac{beat}{min}$. We knew that: $N(n) = \frac{60f}{n}$, hence we can program the EPROM as such. And in this way the very "Look-up Table" will be created for the EPROM.
By getting $A = (a_0, a_1, .., a_{10})^T$ vector from the counter which represents n in binary form, we can find $B = (b_0, b_1, ..., b_7)^T$ vector which represents the estimated number of heart beats per minute N. In this stage this 8-bit binary number can be transformed into 3 ,4-bit-BCD code which after Decoder will be connected to the seven segments.

4 Implementing by Verilog code

4.1 hello

This is the code to generate all possible outputs by getting all possible valid inputs ranging from 273 to 2000. This will be used as the look up table as for n we will pass the n-th line of the *data.txt* file.

```python
def round(x):
    if(x-int(x))>=0.5:
        return int(x)+1
    return int(x)-1

def to_seven_bit(y):
    z = 8- len(y)
    t ='0'*z
    return(t+y)

file = open("data.txt", "x")
for i in range(0,272):
    file.write("00000000\n")
for i in range(273,2000):
    y = 60000/i
    z = format(int(round(y)),'b')
    h = str(z)
    zz = to_seven_bit(h)
    file.write(zz+str("\n"))
```

furthermore we can see that the more n rises, the more the BPM will get close to 30. And the reason behind this is that the function of $N(n) = \frac{60f}{n}$ is decreasing since its derivative $\frac{dN}{dn} = \frac{-60f}{n^2} < 0$.

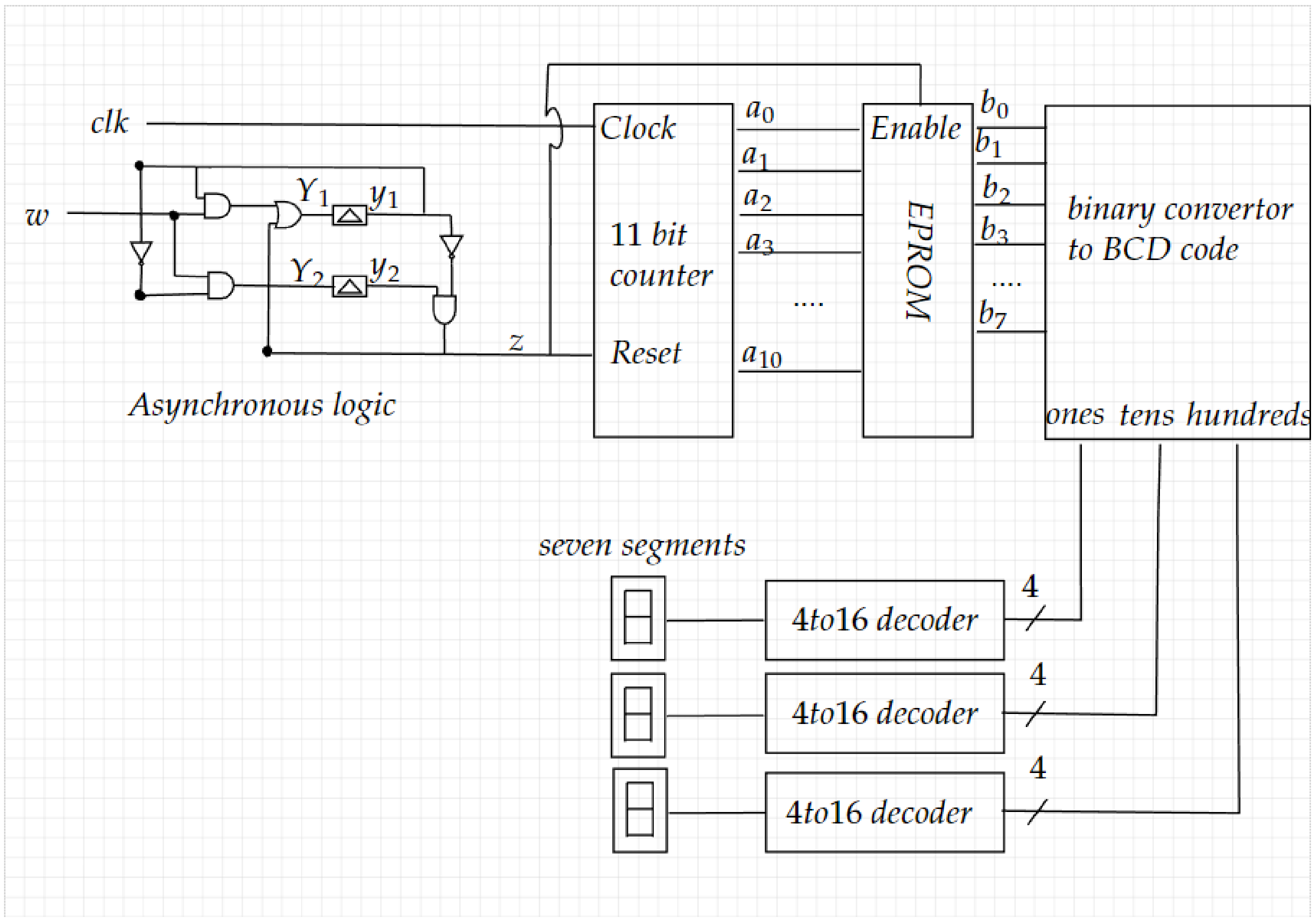

Figure 3: Caption

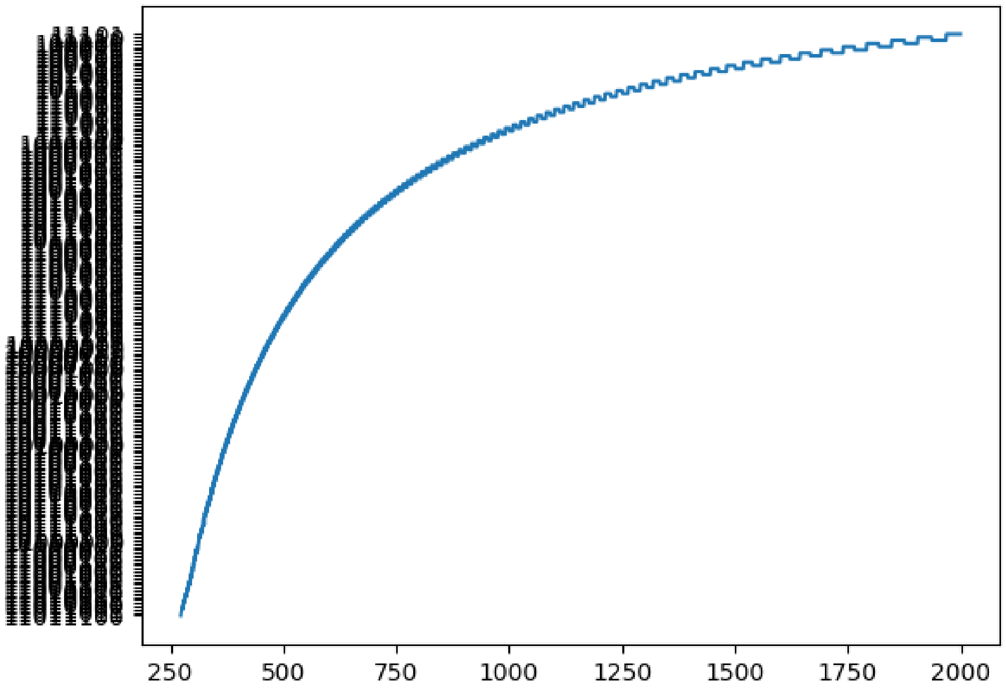

Figure 4: Caption

Now this is the code implemented in verilog (furthermore the file *main.v* is inside the project's file:

```
`timesacle 1ns/1ps
// MohammadParsa Dini 400101204
// Logic Circuits Project
//verilog implementation of the circuit
//aiming to count the rate of heart beats per minute by receiving the heart beat
    pulses

// first we will create a module named test to manage all the buisiness
module test(heart_pulse, clk, out_lut, stop_buzzer, mute, ONES,TENS,HUNDREDS, _7seg_1,
     _7seg_2, _7seg_3);
// here our inputs are the hearbeat(x), the clock signal(clk) and the mute button(mute
    )
input heart_pulse, clk, stop_buzzer, mute;
// here out outputs are the output of lookuptable(out_lut),
// besides the LED and the buzzer and the eight bit binary number of beats/min
// and the inputs to 3 seven segments (ONEs, TENs, HUNDREDS)
output reg [10:0] out_lut;
output reg LED, buzzer;
output reg [7:0] beats;
output reg [7:0] _7seg_1, _7seg_2, _7seg_3;
output reg [3:0] ONES,TENS, HUNDREDS;
// now we will define two variables one is the output of counter and it will
// be reset to zero by reset signal and the other is the exact value of the
// counter right before the reset
reg [10:0] out_counter = 11b'00000000000;
reg [10:0] temp_counter = 11b'00000000000;
// Here we will define two other variables which they aim to specify the state
// of the circuit as we defined earlier
reg HeartPulsePosedge = 0;
reg is_stateA = 0;
reg is_stateB =0
reg temp2 =0;
reg is_heartpulse_posedge = 0;

// Now we  will read and set the memory for the lookup table
reg [10:0] memory [2000:0];
initial begin
@readmemb("data.txt",memory)
 end

// this section of code is concerned with shutting down the buzzer and the LED after
// 100 ms when the pulse ocurred
@always(*) begin
  temp2 = stop_buzzer;
end

//Now we wish to count the number of posedges of clk signal to measure the period time
// and determine the instantaneous BPM of heartbeat
@always(posedge clk) begin
//checking if the current state is A, then a transition to B
//and if the current state is B, then a transition to C and then to A
//meanwhile we will count the number of clk posedges whithin the period
// and increment temp_counter
  if( !heart_pulse && is_stateA)
    is_stateB =1;
  if( heart_pulse && is_stateB) begin
    is_stateA =0;
    is_stateB =0;
  end
  if( HeartPosedge && !is_stateB) begin
    out_counter = 11b'00000000000;
    is_stateA =1;
  end
  temp_counter = 11b'00000000001 + temp_counter;
end
//Here when we reset before setting temp_counter to zero we will store it
// into out_counter
@always(posedge heart_pulse) begin
  out_counter = temp_counter;
  is_heartpulse_posedge = 1;
  temp_counter = 11b'00000000001 + temp_counter;
end
// Here we will make LED on for 100 ms after the heart beat pulse
// and the buzzer will be pon if mute is off and LED is on obviously
assign LED = temp2;
assign buzzer = LED && (!mute);

integer index_lut;
@always(out_counter) begin
  index_lut = out_counter;
end

//assign out_counter = temp_counter;
// here we will make the output of the EPROM (out_lut) in terms of its input
// which is out_counter --> index_lut
assign out_lut = memory[index_lut];
binary_to_bcd btb(BPM,ONES,TENS,HUNDREDS);
bcd7Segment(ONES,_7seg_1);
bcd7Segment(TENS,_7seg_2);
bcd7Segment(HUNDREDS,_7seg_3);
endmodule
// this module will act as an 3 bit full adder
module add3(x,y,s,cin)
input [3:0] x,y;
wire cout;
input cin;
output [3:0] s;
assign {s,cout} = x + y+ cin;
endmodule
// this module will convert binary to 3 digit BCD code(4 bit)
module binary_to_bcd(A,ones,tens,hundreds)
```

```
  input [7:0] A;
  output [7:0] ones,tens,hundreds;
  wire [3:0] c1,c2,c3,c4,c5,c6,c7;
  wire [3:0] d1,d2,d3,d4,d5,d6,d7;
  assign d1 = {1b'0,A[7:5]};
  assign d2 = {c1[2:0],A[4]};
  assign d3 = {c2[2:0],A[3]};
  assign d4 = {c3[2:0],A[2]};
  assign d5 = {c4[2:0],A[1]};
  assign d6 = {1b'0,c1[3],c2[3],c3[3]};
  assign d7 = {c6[2:0],c4[3]};
  add3 m1(d1,c1);
  add3 m2(d2,c2);
  add3 m3(d3,c3);
  add3 m4(d4,c4);
  add3 m5(d5,c5);
  add3 m6(d6,c6);
  add3 m7(d7,c7);
  assign ones = {c5[2:0],A[0]}
  assign tens = {c7[2:0],c5[3]}
  assign hundreds = {c6[3],c7[3]}
endmodule
// this module will get the bcd code and make an instant of the pinouts of a seven
    segment
module bcd7Segment(bcd,seg); // HW Code:
  input [3:0] bcd; //initializing bcd as an 4 bit input signal
  output[7:0] seg; //initializing seg as an 8 bit output signal
  reg [7:0] seg; //initializing bcd signal as registers
  always @ (bcd)begin
    case(bcd)
      0: seg = 8'b11000000; //when bcd = 0
      1: seg = 8'b11111001; //when bcd = 1
      2: seg = 8'b10100100; //when bcd = 2
      3: seg = 8'b10110000; //when bcd = 3
      4: seg = 8'b10011001; //when bcd = 4
      5: seg = 8'b10010010; //when bcd = 5
      6: seg = 8'b10000010; //when bcd = 6
      7: seg = 8'b11111000; //when bcd = 7
      8: seg = 8'b10000000; //when bcd = 8
      9: seg = 8'b10010000; //when bcd = 9
      default: seg=8'b11111111; //any other value
    endcase
  end
endmodule
```

in this place we will make the test bench for it:

```
`timesacle 1ns/1ps
// MohammadParsa Dini 400101204
// Logic Circuits Project
//verilog implementation of the circuit
//aiming to count the rate of heart beats per minute by receiving the heart beat
    pulses

// the output to the circuit
reg heart_pulse, clk, mute=0, stop_buzzer=0;
wire[10:0] out_counter;
wire[7:0] out_lut;
wire LED, buzzer;

integer f =1000;//frequency of the clock
integer turns = 10;
integer muter_index = 8124;
integer 1microsec = 100000;
integer clk_length;
//Instant from the main.v file in order to managing the signals
test uut(.heart_pulse(heart_pulse), .clk(clk), .out_counter(out_counter),.out_lut(
    out_lut),
    .mute(mute), .LED(LED), .stop_buzzer(stop_buzzer));
// now making the clk and heart beat signals manually:
//additionally we will mute the buzzer at the 8124th loop
```

```
for(integer i =0;i<f*turns;i= i +1) begin
    if(i==muter_index)
        mute = 1;
    clk_length = 1microsec*5;
    // now setting the pause time to produce a clock signal
    #clk_length clk = 1;
    #clk_length clk =0;
end

initial begin
    for(i = 0 ; i <f * turns; i++) begin
        #(400*1microsec) heart_pulse = 1;
        #(200*1microsec) heart_pulse = 0;
        #(100*1microsec) stop_buzzer =0;
    end
end
```

5 Showing the results

References