ECEN2350 Digital Logic Spring 2016

Project 3 - The Reaction Timer

Milan Formanek SID-103017710

May 5, 2016

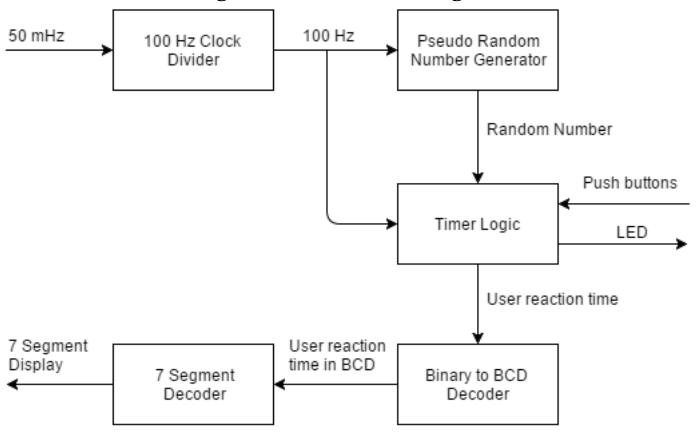
1 Introduction

The goal of this project is to create a reaction timer using the DE0 FPGA board. To make this work I will use more advanced features like the 50 mHz clock provided by the DE0 board as well as counters a binary to BCD decoder and a pseudo random number generator.

2 Overview

When the user releases the reset button, the device waits a random amount of time then flashes the LED and times the delay it takes the user to depress the "game" button. The delay between the the reset and the led bilking is to prevent the user from anticipating the exact time the reaction timer starts, thus truly testing his or her reaction time. After the users reaction time has been recorded the resultant reaction time is displayed in decimal form on the 7 segment screen.

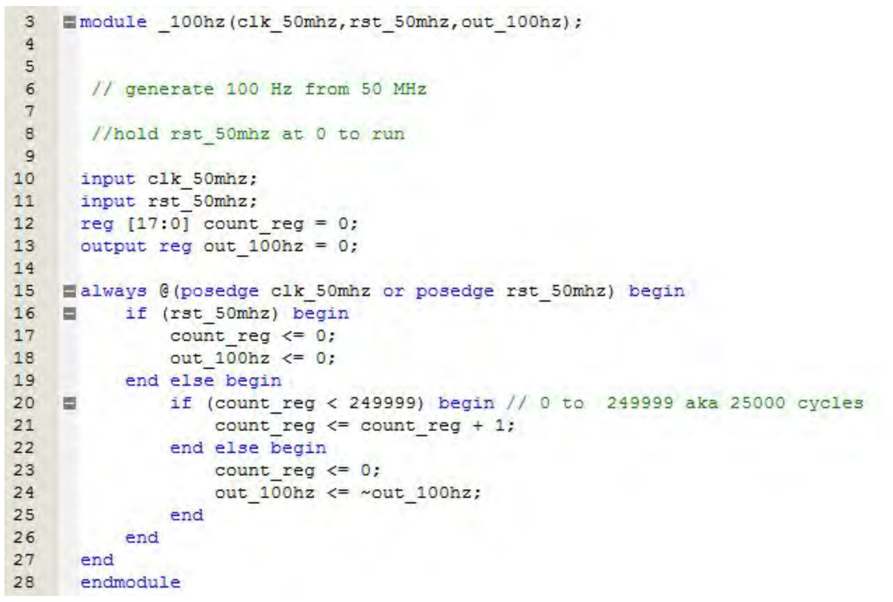
Figure 2.1: Device diagram



2.1 100 Hz Clock Divider

The 50 mHz on board clock the DE0 provides, needs to be stepped down to a 100 Hz in order to trigger the Timer logic every $\frac{1}{100}$ of a second achieving the precision desired. This is implemented using a counter that overflows every 250000 cycles of the 50 mHz clock giving a final divided frequency of a 100 Hz.

Figure 2.2: 50mHz to 100Hz divider verilog hardware description

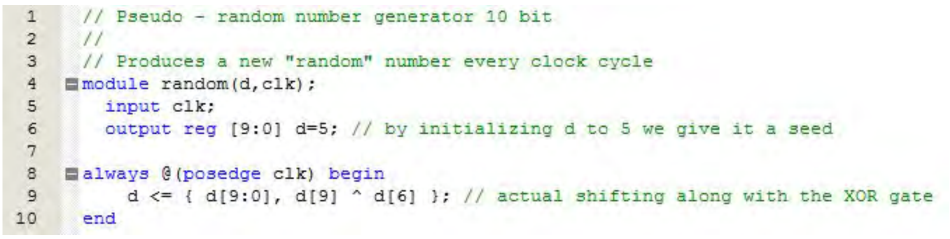


```
module _100hz(clk_50mhz,rst_50mhz,out_100hz);


  // generate 100 Hz from 50 MHz

  //hold rst_50mhz at 0 to run

 input clk_50mhz;
 input rst_50mhz;
 reg [17:0] count_reg = 0;
 output reg out_100hz = 0;

always @(posedge clk_50mhz or posedge rst_50mhz) begin
     if (rst_50mhz) begin
         count_reg <= 0;
         out_100hz <= 0;
     end else begin
         if (count_reg < 249999) begin // 0 to  249999 aka 25000 cycles
             count_reg <= count_reg + 1;
         end else begin
             count_reg <= 0;
             out_100hz <= ~out_100hz;
         end
     end
 end
 endmodule
```

2.2 Pseudo Random Number Generator

Using a linear feedback shift register (LFSR) you can generate a sequence of pseudo random numbers. This is achieved by shifting the data set in a shift register and XORing certain outputs with each other. I want to generate a 10 bit random number, that will give me a max value of 1023 translating into the max go led delay of 10.23 sec. This is suitable to give enough of a range to provide enough randomness, but worst case scenario this still doesn't let the user wait too long. Using the Xilinx application notes or xapp052.pdf table 3 I find the right outputs to XOR. A LFSR can only be classified as a pseudo random number generator as it traverses $2^n - 1$ states before repeating itself. Where n is the light of the shift register. I our case this would be $2^{10} - 1$ states. Another thing to note is that if the shift register gets filled with only 0s the machine will lock up in that state forever. This is due to the fact that 0 XOR 0 = 0.

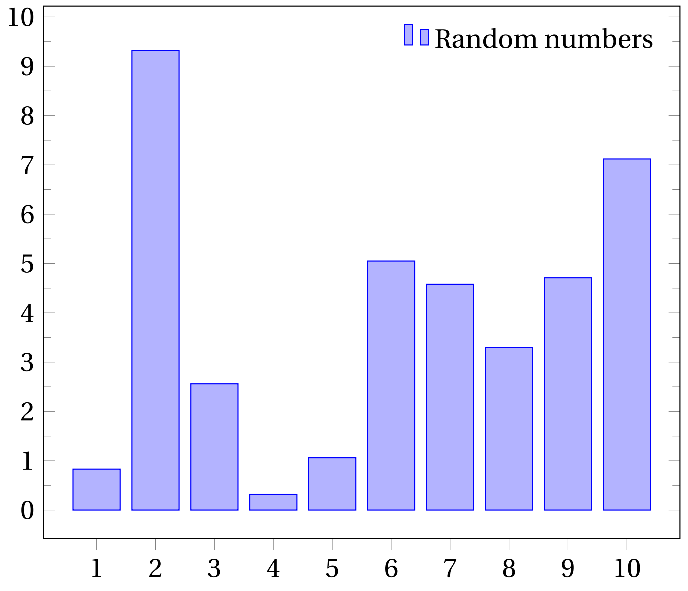
Figure 2.3: Pseudo Random Number Generator Logic verilog hardware description



```
// Pseudo - random number generator 10 bit
//
// Produces a new "random" number every clock cycle
module random(d,clk);
  input clk;
  output reg [9:0] d=5; // by initializing d to 5 we give it a seed

always @(posedge clk) begin
    d <= { d[9:0], d[9] ^ d[6] }; // actual shifting along with the XOR gate
end
```

In Fig 3.1you can see the register is prevented from initializing in the lock up 0 state by setting the output register d to 5 initially. The shift register shifts down one every clock cycle producing a "fresh" random number.
To test if the LFSR worked I had a runtime that captured a random number every time I pressed the button and displayed it on the screen. Fig. 2.4 below presents a sample of random numbers generated by the LFSR.

Figure 2.4: Histogram of 10 sample random numbers from the generator



2.3 Timer Logic

The timer logic take care of the user input, flashing the go led, counting the time using the 100 Hz clock cycles and finally outputting the user reaction time in binary to the BCD decoder.
Fig 2.5 shows of the individual states of the reaction timer. The 10bit wide wire t represent the up to date "fresh" random number from the generator that is captured into the rdm register and led[9] represents the go led.
The number of clock cycles is stored in a 32 bit register. A number of this size will probably never get used and can't even be properly displayed on a 4 digit screen without overflow, nevertheless the counter will count up that high.

2.4 Binary To Bcd Decoder

The decoder gets a 14 bit number end converts it into 4 separate BCD digits. For this it uses the shift and add 3 method. The algorithm shifts the input and if the resultant shifted number is bigger than 4 it add 3 and then shift again. This process repeats itself until it has shifted the same amount of times as the input has binary digits.

Figure 2.5: State Diagram

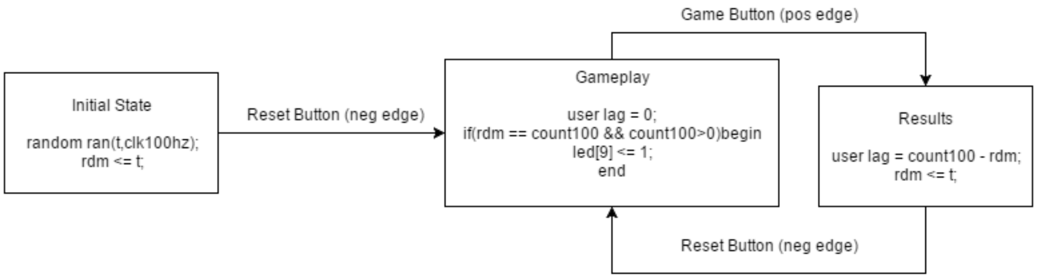
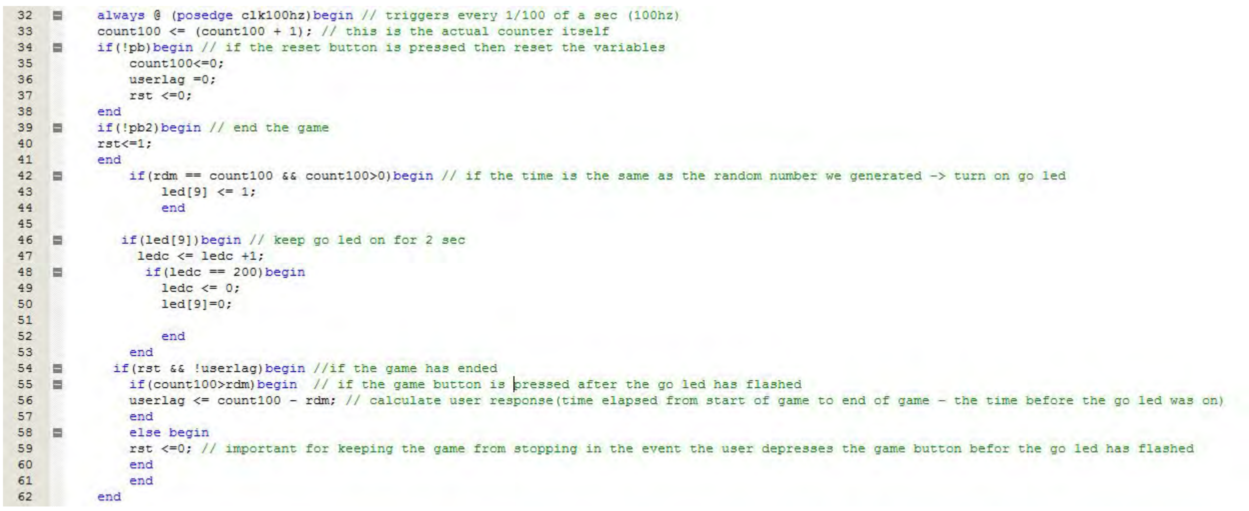
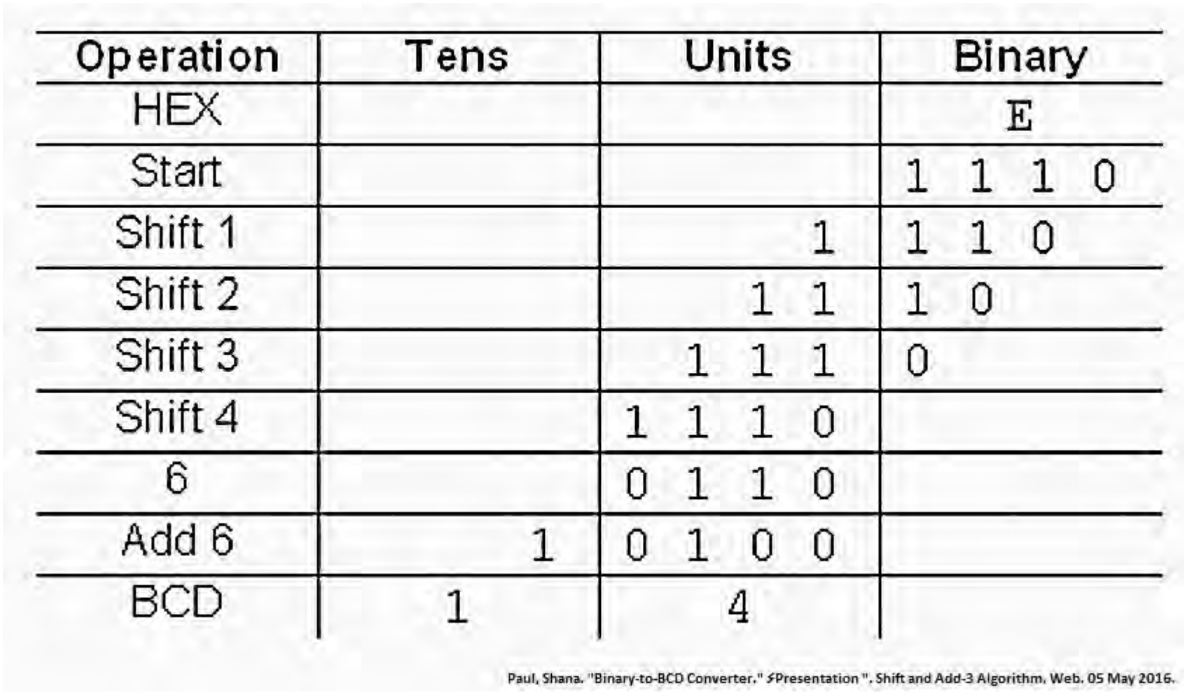


Figure 2.6: The Timer Logic verilog hardware description



```
always @ (posedge clk100hz)begin // triggers every 1/100 of a sec (100hz)
count100 <= (count100 + 1); // this is the actual counter itself
if(!pb)begin // if the reset button is pressed then reset the variables
    count100<=0;
    userlag =0;
    rst <=0;
end
if(!pb2)begin // end the game
rst<=1;
end
    if(rdm == count100 && count100>0)begin // if the time is the same as the random number we generated -> turn on go led
        led[9] <= 1;
        end

   if(led[9])begin // keep go led on for 2 sec
     ledc <= ledc +1;
      if(ledc == 200)begin
        ledc <= 0;
        led[9]=0;

        end
    end
  if(rst && !userlag)begin //if the game has ended
    if(count100>rdm)begin  // if the game button is pressed after the go led has flashed
    userlag <= count100 - rdm; // calculate user response(time elapsed from start of game to end of game - the time before the go led was on)
    end
    else begin
    rst <=0; // important for keeping the game from stopping in the event the user depresses the game button befor the go led has flashed
    end
    end
end
```

Figure 2.7: shift and add 3 algorithm



Operation	Tens	Units	Binary
HEX			E
Start			1 1 1 0
Shift 1		1	1 1 0
Shift 2		1 1	1 0
Shift 3		1 1 1	0
Shift 4		1 1 1 0	
6		0 1 1 0	
Add 6	1	0 1 0 0	
BCD	1	4	

Paul, Shana. "Binary-to-BCD Converter." ⚡Presentation ". Shift and Add-3 Algorithm. Web. 05 May 2016.

Fig. 2.7 above gives a visual representation of how the algorithm works converting 1110_2 into BCD digits 1 and 4

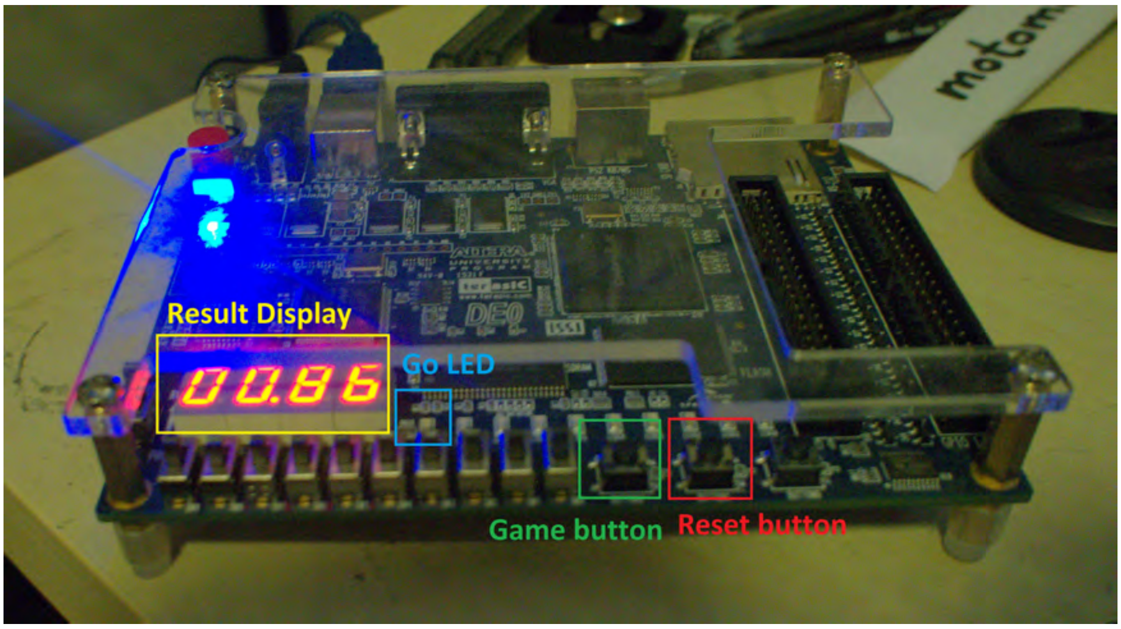
2.5 7 Segment decoder

The decoder decodes input in the BCD (binary coded decimal) format to the 7 segment display. This is done using a bitmask on the led arrays that make up the screens themselves. The same logic was used in my project 1 and 2.

3 User Manual

After the device is turned on it waits for the user to press the reset button. Once the reset button is released (neg edge) the go led flashes after a random time interval. The device times how long it takes the user to depress (pos edge) the game button after the go led has turned on. The result is displayed on the 7 segment screen in seconds. Fig 3.1 shows the physical layout of the device.

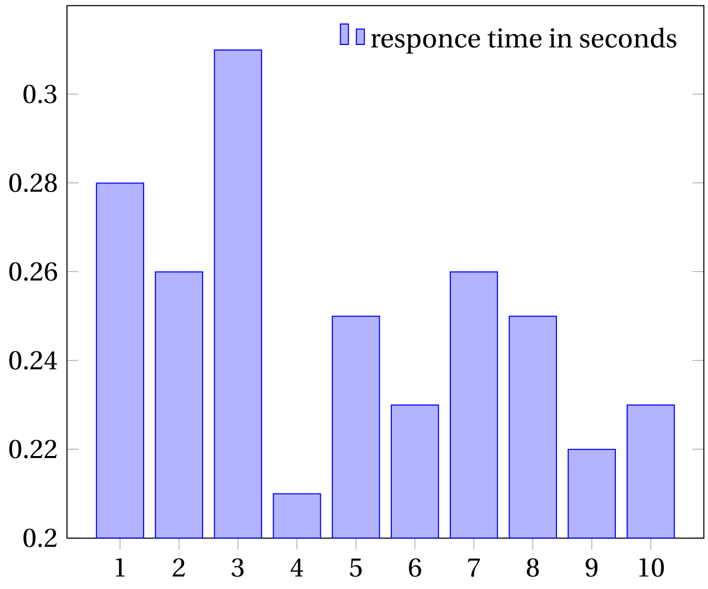
Figure 3.1: Overview



4 Testing

I have tested my reaction time at least 50 times. 10 of which are listed in fig. 4.1 below. I usually land somewhere between 0.23 sec and 0.28 sec. My best result is 0.21 sec and I finally had to give up on trying to break the 0.2 sec mark. I also tested with my flat mates and various friend with results varying from 0.20 sec to 0.53 sec. All the test subjects were in there 20s, it would be interesting to test with someone older to see if response time really decays with age.

Figure 4.1: 10 of my response times



5 Final remarks and conclusion

After project 2 I had all but given up on my DE0 board. The switches did not work and the 7 segment screen flickered intermittently. I tested it with different code and determined it was a hardware issue and I just got a lemon. Before throwing in the towel and buying a new board I though I would try one last thing. After stripping the board down I put it in the oven for 8 minutes at 380 F to try re-flow the solder. I would say this is crazy but it worked. After letting the board cool down and reassembling it worked like new. As far as the project itself goes I found this one easier than project 2. Everything was straight forward, event the random number generator which I thought would be a challenge didn't take to long. Last semester I took the digital clock project class where I used a physical 7400 series counter chip to step down a 32 kHz clock from a RTC crystal and all I have to say is the process of making an equivalent circuit in verilog "software" is so much simpler and less hassle. I'm glad that I can finally present you with a project that is up to my standards.