ELC 4396 02
Class Report 2

Jordan Cook

September 21, 2021

Introduction and Problem Statement

The purpose of this coding project was to create a reaction timer game. A user would press start, wait a random amount of time, and press a button as soon as possible when the reaction LED lit up. This utilized several modules that needed to be created for this specific project as well as some older ones.

Link to my GitHub Repo: https://github.com/jordanstarr/System-on-Chip
Link to the Verilog Code: https://github.com/jordanstarr/lucky_charm

Method and Approach

This project needed a total of 4 modules, often using lower-level modules created.

The first step was to make a counter module. This was going to be used in two places: the first being a 2 bit count up to 3 for the anode display placing, and the second was a 3 bit count up to 7 for the seven segment display. It was paramarterized in order to get the total number it would count to as well as the number of bits needed on the most significant end.

The concept here was sort of a timer that would be created in a counter, taking advantage of the clock rate of the board. The first parameter (N) was going to be the largest amount of bits it would count up to. The larger the number, the longer it would take for this most significant bits to change in a board that runs extremely fast. The second parameter (M) represents how many bits that are necessary for analysis.

Listing 1: Counter Module Code

```
module counter#(parameter N=18, parameter M=2)(
   input logic clk,
   input logic rst,
   output logic [M-1:0] count
   );

   logic [N-1:0] state, nstate;

   always_ff @(posedge clk, posedge rst)
      if(rst)
         state <= 0;
      else
```

```
        state <= nstate;

    assign nstate = state + 1;

    assign count = state[N-1:N-M];

endmodule
```

The next module made was the driver for the seven segment. This got input from another module's output (soon to be made) and output which specific anode was going to be lit-up with a repeating pattern. It's (N) parameter was 18, meaning that the system would count so quickly, it could not be observed by the human eye that things were changing. Since the anode values only went up to 3, the (M) parameter was 2 because that's all the bits that were needed in this count.

Listing 2: Seven-Segment Driver Code

```
module sseg_driver(
   input logic clk,
   input logic rst,
   input logic [7:0] ss0,
   input logic [7:0] ss1,
   input logic [7:0] ss2,
   input logic [7:0] ss3,
   output logic [7:0] sseg,
   output logic [7:0] an
   );

   logic [1:0]count;
   logic [2:0]ssegCount;

   counter#(.N(18), .M(2)) myCounter(
      .clk(clk),
      .rst(rst),
      .count(count)
      );

   always_comb
      case(count)
         0: begin
            sseg = ss0;
            an = 4'b1110;
            end
         1: begin
            sseg = ss1;
            an = 4'b1101;
            end
         2: begin
            sseg = ss2;
            an = 4'b1011;
            end
         default: begin
            sseg = ss3;
            an = 4'b0111;
            end
```

```
      endcase

   //this will turn all of the anodes off
   assign an[7:4] = 4'b1111;

endmodule
```

(Note: the last line of code shown below sets the left-most 4 anodes off because they are not being utilized.)

Another module that needed to be created was one that would determine the rotating pattern of the 7-segment and output that information to the driver file. This also utilized that counter module previously made. It's (N) value was around 25-27, the lower the number, the faster the rotation. (If the rotation was too fast, it could not be observed by the human eye.)

Listing 3: Rotator Module Code

```
module rotator#(parameter N = 3)(
   input logic clk,
   input logic rst,
   output logic [7:0] ss0,
   output logic [7:0] ss1,
   output logic [7:0] ss2,
   output logic [7:0] ss3
   );

   parameter top    = 8'b10011100;
   parameter bottom = 8'b10100011;
   parameter off    = 8'b11111111;

   logic [N-1:0] tally;

   counter#(.N(25), .M(3)) myCounter(
      .clk(clk),
      .rst(rst),
      .count(tally)
      );

   always_comb
      case(tally)
         0: begin
            ss0 = off;
            ss1 = off;
            ss2 = off;
            ss3 = top;
         end
         1: begin
            ss0 = off;
            ss1 = off;
            ss2 = top;
            ss3 = off;
         end
```

```
         2: begin
            ss0 = off;
            ss1 = top;
            ss2 = off;
            ss3 = off;
         end
         3: begin
            ss0 = top;
            ss1 = off;
            ss2 = off;
            ss3 = off;
         end
         4: begin
            ss0 = bottom ;
            ss1 = off;
            ss2 = off;
            ss3 = off;
         end
         5: begin
            ss0 = off;
            ss1 = bottom;
            ss2 = off;
            ss3 = off;
         end
         6: begin
            ss0 = off;
            ss1 = off;
            ss2 = bottom;
            ss3 = off;
         end
         7: begin
            ss0 = off;
            ss1 = off;
            ss2 = off;
            ss3 = bottom;
         end
      endcase
endmodule
```

Putting all these modules together, a a final top level main module was created. It was comprised of the rotator and the driver. The rotator (using the counter module) determined what pattern was going to be output for each of the four 7-segment values. It needs to be connected to the clock to facilitate the counters. It needs to be connected to reset because usually, anything that deals with memory (in the counter) needs to be connected to a reset. The driver was very similar to the rotator, the differences being that the four 7-segment values were given to the module, the counter is different, and it outputs to the board's "sseg" and "an".

Listing 4: Main Module Code

```
module rotator#(parameter N = 3)(
module main (
   input logic clk,
   input logic reset_n,
   output logic [7:0] an,
```

```
    output logic [7:0] sseg
    );

    logic [7:0] ss0;
    logic [7:0] ss1;
    logic [7:0] ss2;
    logic [7:0] ss3;

    rotator myRotator (
        .clk(clk),
        .rst(!reset_n),
        .ss0(ss0),
        .ss1(ss1),
        .ss2(ss2),
        .ss3(ss3)
        );

    sseg_driver myDriver (
        .clk(clk),
        .rst(!reset_n),
        .ss0(ss0),
        .ss1(ss1),
        .ss2(ss2),
        .ss3(ss3),
        .sseg(sseg),
        .an(an)
        );

endmodule
```

Note: The code states "!reset_n" because the board uses negative logic (denoted by the "n").

Testing

Testing on this project was fairly simple: either the code worked or it didn't. It was supposed to start on the upper left box and rotate around clockwise. It was important that they were not all lit up (this would indicate a timing issue). A lot of times there were issues with the timing or placement of 7-segments, but this simply took trial and error (as well as consulting the textbook).

Results

After careful timing, the pattern as stated in the textbook was observed on the board. For submission, a video of the board working will be included on the Github link.

Conclusion

With the knowledge gained from this experiment, the 7-segment display could be adjusted for more patterns or more uses. Learning how to use the counter module in order to essentially create a pseudo-timer was invaluable and not a approach that I would initally have considered. With more practice in projects like this, I will be able to understand how to create the necessary modules for the needed assignment.