ELC 4396 02
Class Report 1

David Davis

September 9, 2021

Introduction

The purpose of this class project was to get familiar with working with the Nexys4 DDR by rotating a square around the seven segment LED component. Different modules were built using Verilog to program the Nexys board. Using the on-board features of the Nexys board, the project was completed with without any exterior electrical components.

GitHub Repo: https://github.com/davisdavid/SoC.

Implementation

The first step in the process was to create the counter design source. It would be implemented in two sections of the project that allowed for different case statements to be utilized. The counter had parameters that could set the size of the counter and the speed at which it was incremented. The first counter was used to increment between each anode display. The anode counter was only 2 bits, which allowed for the four separate anode displays. The second counter was 3 bits and incremented between the 8 segment options that allowed for specific patterns, such as the square in this case. The counters would both interact to rotate the lit up squares in a continuous loop as shown in the video.

Listing 1: Counter Module

```
module counter#(parameter N=18, parameter M=2)(
   input logic clk,
   input logic rst,
   input 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 created was the seven segment driver that would turn on the correct anode display. As mentioned in the previous section, there are four different anode displays on the board. The case statement in the code belows shows the four different options. As the counter increments, it moves from case to case, giving an alternating anode to light up. The assign statement on the last line sets the other four anodes on the board to off, so the square will only rotate throught the four rightmost displays.

Listing 2: Seven Segment Driver Module

```
module ssegdrv(
    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;

    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

    assign an[7:4] = 4'b1111;

endmodule
```

The third module implemented was the rotator. The rotator module was very similar to the Seven Segment Driver module. It uses a counter to count through the different cases. The only difference here is that the rotator increments through the various position of the LED square. The first LED square would start in the upper-leftmost square and then work its way to the right. The rotator code, as shown below, sets one of the seven segment displays to top or bottom, depending on where the square should be located, and sets the other three seven segment displays to off.

Listing 3: Rotator Module

```
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] ticker;

    counter#(.N(25), .M(3)) mycounter(
    .clk(clk),
    .rst(rst),
    .count(ticker)
    );

    always_comb
    case(ticker)
    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
```

The last step in this project was putting the different modules together in a main source code. The main module just connects the rotator module with the seven segment driver module.

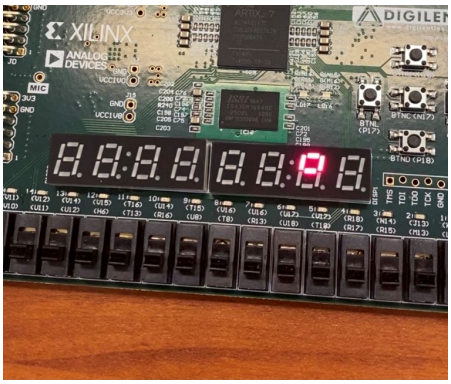
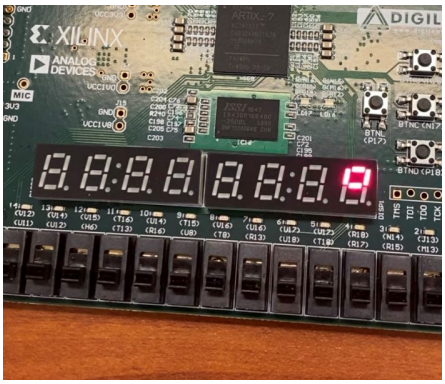
Listing 4: Main Module

```
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#(.N(3)) myrotator(
    .clk(clk),
    .rst(!reset_n),
    .ss0(ss0),
    .ss1(ss1),
    .ss2(ss2),
    .ss3(ss3)
    );
```

```
    ssegdrv myssegdrv(
    .clk(clk),
    .rst(!reset_n),
    .ss0(ss0),
    .ss1(ss1),
    .ss2(ss2),
    .ss3(ss3),
    .sseg(sseg),
    .an(an)
    );

endmodule
```

Results

The LED square lit up and rotated around the four anode displays in a clockwise pattern. The speed of the counter made the square move pretty quickly around the loop. It was very quick, but still visible to the eye. The pictures below show the transition from one square to the next.

Conclusion

The project was very successful in being introduced to the board and learning how to program it. The constraints given made it very easy to manipulate the board without having to go into too much detail. There was very minimal code to achieve the goal of this project, so it's clear that there are many possibilities and functions that can be implemented.