Visual Fidget Toy (o_0)

User Manual

Nate Sherman & Colin Babian

Product Description

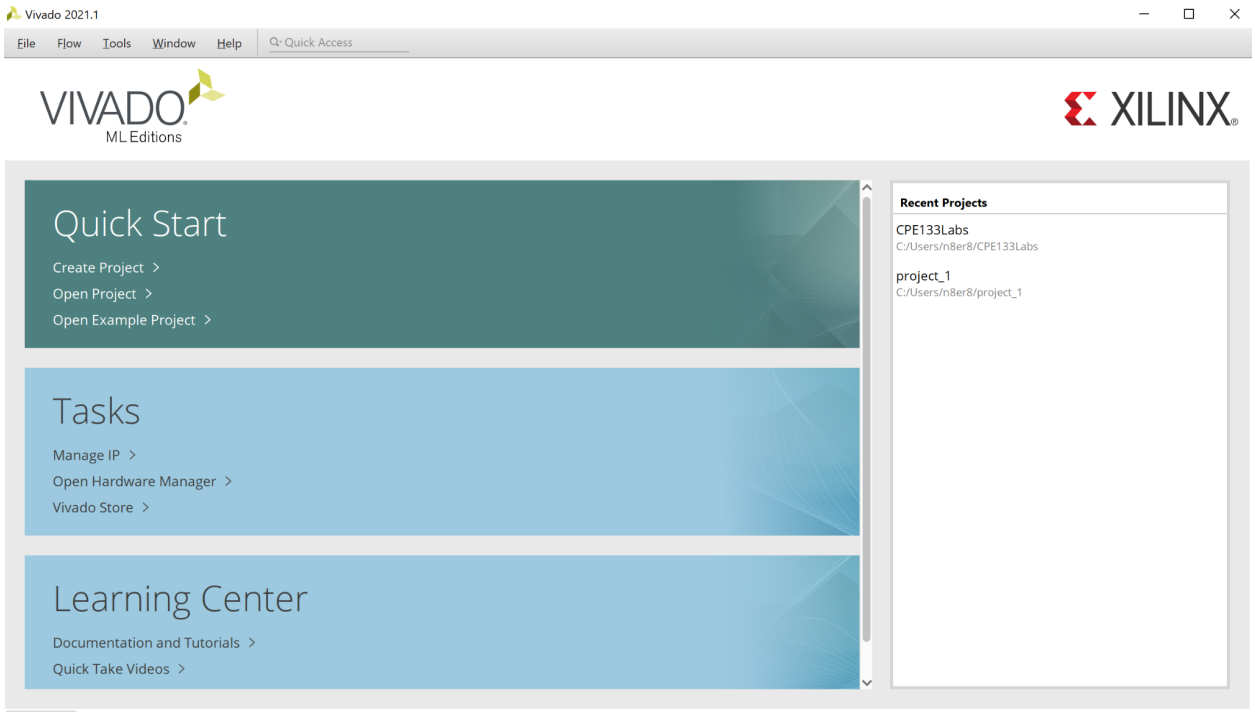
The visual fidget toy is a toy designed to relieve stress and give your mind a break. The provided assortment of patterns will leave the user mesmerized and the variation and switch/button functionality acts as a visual fidget toy as well.

The design includes the usage of a Basys board 7 segment display to display a pattern that will leave you in awe! To manipulate the pattern try pressing some buttons and switches to see what they will do! (Spoilers: the buttons will change the pattern shape and orientation and the switches will adjust the pattern oscillation speed)

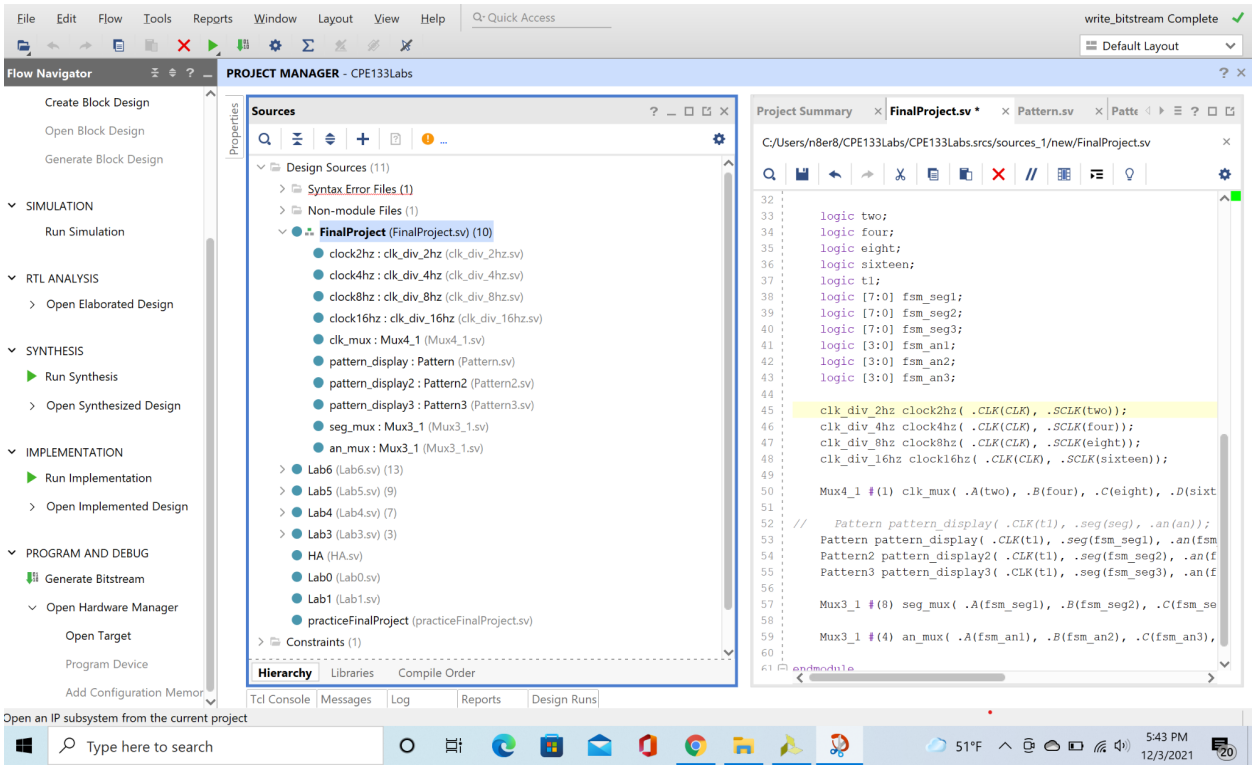
Set-Up and Directions

1. Open Vivado 2021.1. The icon should look like this:

2. After Vivado is launched, open the project titled “CPE133Labs”

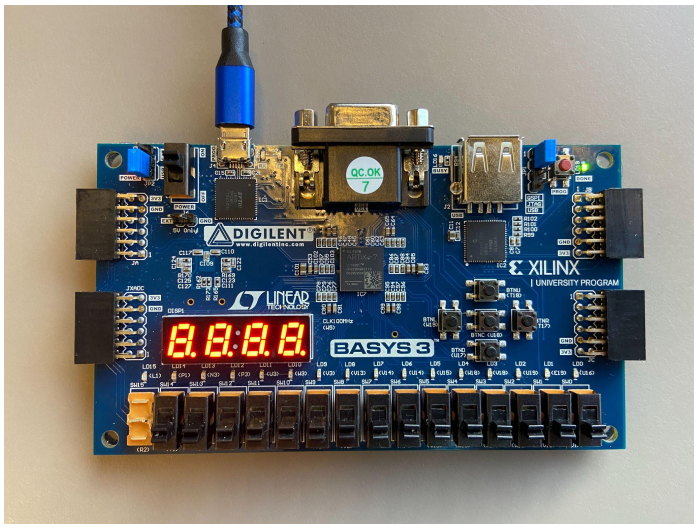


3. When the CPE133Labs is open, make sure the file titled “FinalProject” is set as top by right-clicking on the file and selecting “set as top”. The project should look as below:



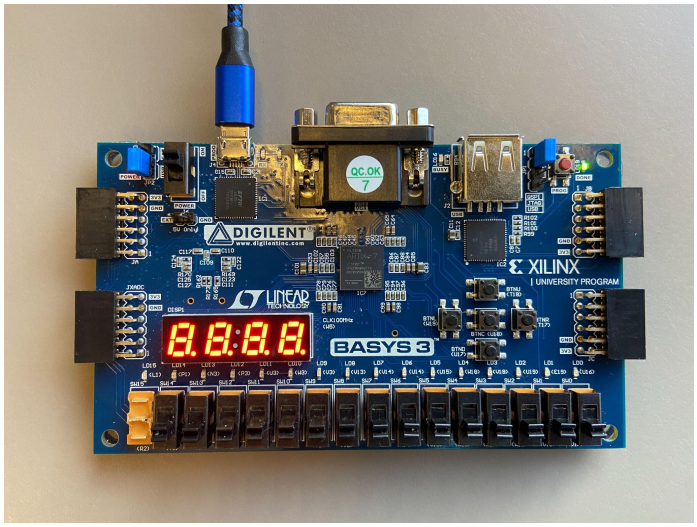
4. Next, select “Generate Bitstream” and select “Okay”. Wait for the Bitstream to finish generating. This may take a minute.
5. Plug in your Basys Board to your computer.

6. After the Bitstream is generated, select “Open Hardware Manager”, “Open Target”, and “Auto Connect”.
7. After your Basys Board is connected, make sure your Basys Board is turned on and there are lights on the display.
8. Next, select “Program Device” on Vivado. After a couple seconds, your Basys Board should be programmed and ready to use! You will know you are ready to play when the SevenSeg display is fully turned on as such:

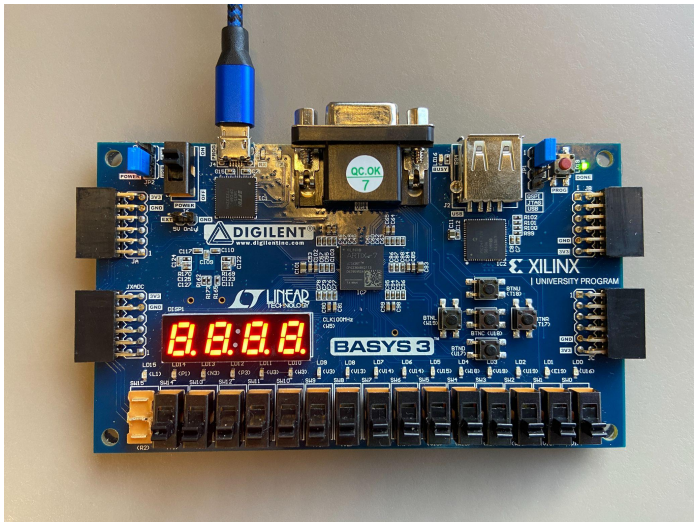


How to Use/Play

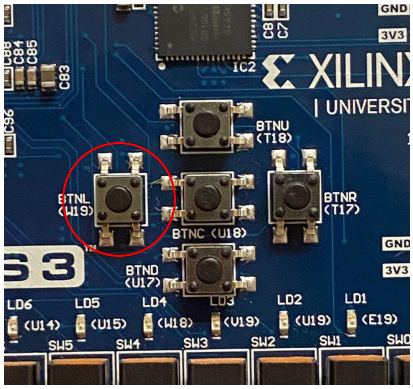
You will know you are ready to play when you have programed your basys 3 board device and the SevenSeg display is completely on as pictured below:



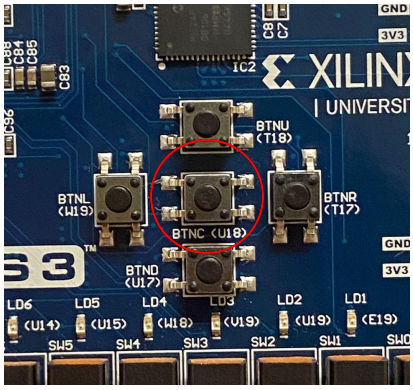
Next, you can PRESS and HOLD one of three buttons to activate a display pattern. These buttons are the W19 button, the U18 button, and the T17 button on the Basys 3 board.



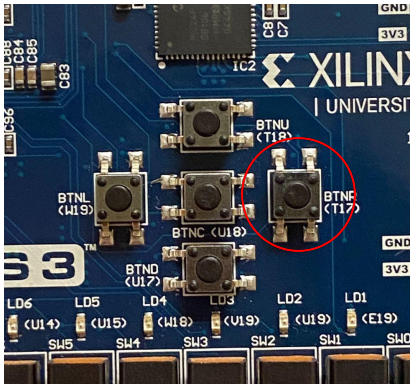
This is the W19 button:



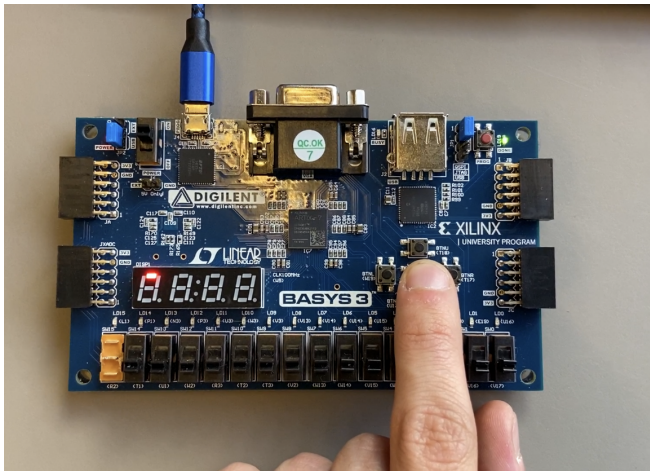
This is the U18 button:

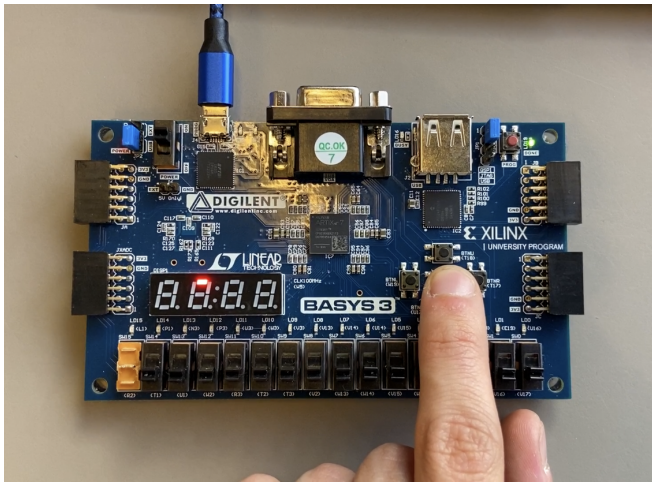


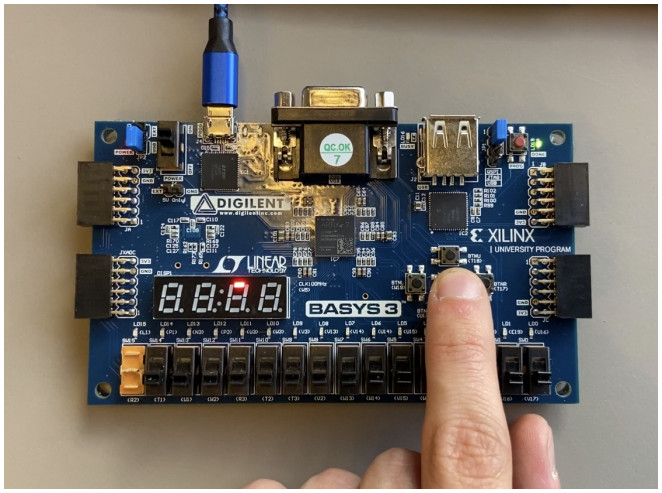
This is the T17 button:



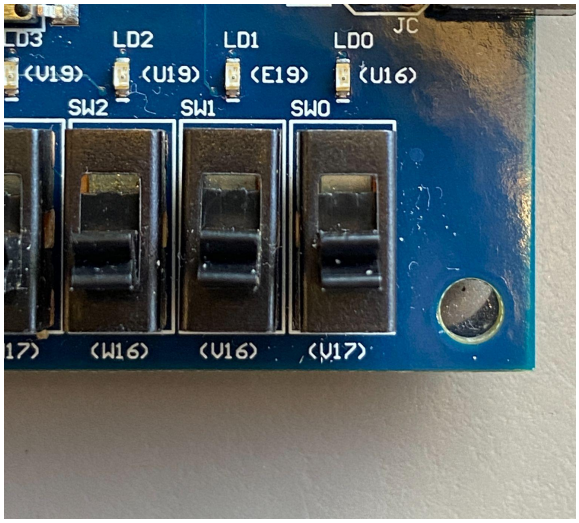
When you press and hold one of the buttons, a pattern will start to appear. This is the start of the U18 button pattern is as follows:



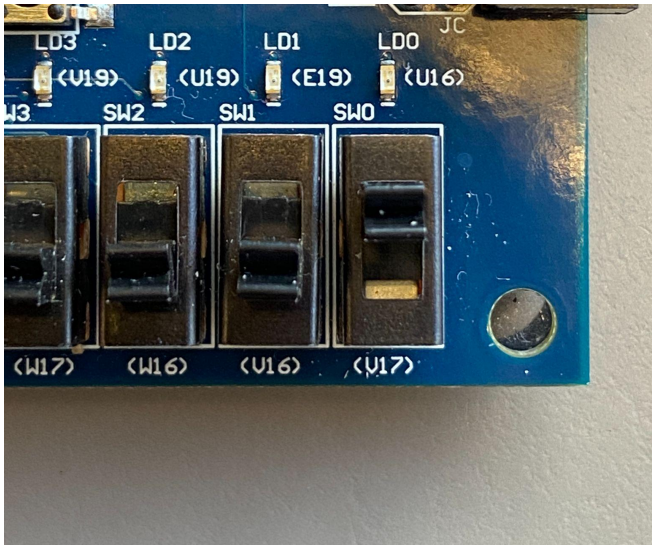




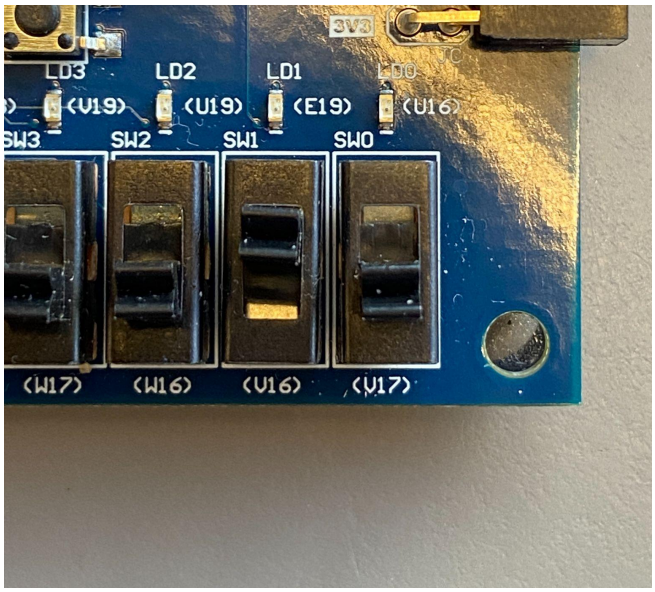
If you want to adjust the speed at which the patterns progress, you can use the two rightmost switches on the Basys Board to adjust the speed. There are four possible speed settings.
The slowest speed is when both switches are “off”:



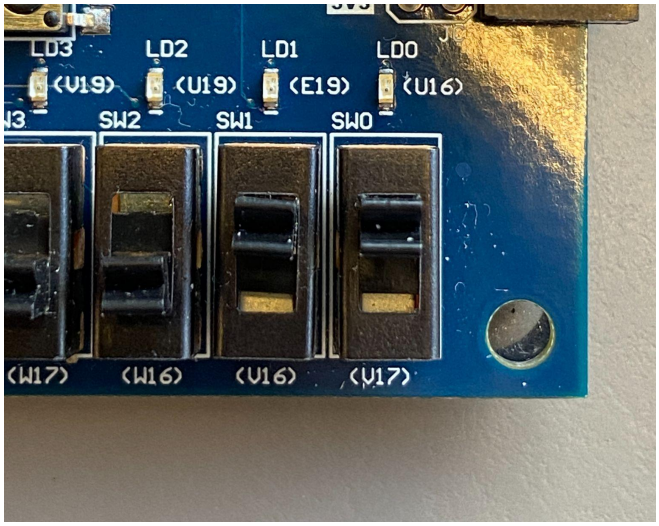
The second-slowest speed is when the rightmost switch is “on” and the second rightmost switch is “off”:



The second-fastest speed is when the rightmost switch is “off” and the second rightmost switch is “on”:



The fastest speed is when the rightmost switch is “on” and the second rightmost switch is also “on”:



That's it! Enjoy fidgeting with our toy!

Limitations

We designed our project to be as inclusive as possible and enjoyable to all ages, races, physical abilities, nationalities, religions, and genders as possible. With that being said, there are some physical ability limitations.

Blind People will not get the satisfaction of enjoying the pleasant experience that the visual fidget toy provides. Perhaps in the future a braille version of our product could be provided.

People who have no sense of touch will miss out on the feeling of flipping the switches and pressing the buttons.

Acknowledgements

- For the most part, Nate and Colin worked together on designing this project
- Nate mostly designed the four clock dividers
- Colin mostly designed the two custom muxes we had to create
- Both Nate and Colin designed the top level code, pattern FSMs, and contributed to troubleshooting our code

Top Level Code: FinalProject.sv

```
module FinalProject(
```

```
    input [1:0] sel,
    input CLK,
    input btn1,
    input btn2,
    input btn3,
    output [7:0] seg,
    output [3:0] an
    );

    logic two;
    logic four;
    logic eight;
    logic sixteen;
    logic t1;
    logic [7:0] fsm_seg1;
    logic [7:0] fsm_seg2;
    logic [7:0] fsm_seg3;
    logic [3:0] fsm_an1;
    logic [3:0] fsm_an2;
    logic [3:0] fsm_an3;

    clk_div_2hz clock2hz( .CLK(CLK), .SCLK(two));
    clk_div_4hz clock4hz( .CLK(CLK), .SCLK(four));
    clk_div_8hz clock8hz( .CLK(CLK), .SCLK(eight));
    clk_div_16hz clock16hz( .CLK(CLK), .SCLK(sixteen));

    Mux4_1 #(1) clk_mux( .A(two), .B(four), .C(eight), .D(sixteen), .SEL(sel), .mux_out(t1));

    Pattern pattern_display( .CLK(t1), .seg(fsm_seg1), .an(fsm_an1));
    Pattern2 pattern_display2( .CLK(t1), .seg(fsm_seg2), .an(fsm_an2));
    Pattern3 pattern_display3( .CLK(t1), .seg(fsm_seg3), .an(fsm_an3));

    Mux3_1 #(8) seg_mux( .A(fsm_seg1), .B(fsm_seg2), .C(fsm_seg3), .BTN1(btn1),
.BTN2(btn2), .BTN3(btn3), .mux_out(seg));

    Mux3_1 #(4) an_mux( .A(fsm_an1), .B(fsm_an2), .C(fsm_an3), .BTN1(btn1), .BTN2(btn2),
.BTN3(btn3), .mux_out(an));

endmodule
```

Two Hz Clock Divider: clk_div_2hz.sv

```
module clk_div_2hz #(parameter COUNT = 23999999)(
   input CLK,
   output logic SCLK = 0
   );

   logic [31:0] count = 0;
   always_ff @ (posedge CLK)
   begin
      if (count == COUNT -1 )
      begin
         SCLK <= ~SCLK;
         count <= 0;
      end
      else
         count <= count + 1;
   end

endmodule
```

Four Hz Clock Divider: clk_div_4hz.sv

```
module clk_div_4hz #(parameter COUNT = 11999999)(
   input CLK,
   output logic SCLK = 0
   );

   logic [31:0] count = 0;
   always_ff @ (posedge CLK)
   begin
      if (count == COUNT -1 )
      begin
         SCLK <= ~SCLK;
         count <= 0;
      end
      else
         count <= count + 1;
   end
```

```
endmodule
```

Eight Hz Clock Divider: clk_div_8hz.sv

```
module clk_div_8hz #(parameter COUNT = 5999999)(
   input CLK,
   output logic SCLK = 0
   );

   logic [31:0] count = 0;
   always_ff @ (posedge CLK)
   begin
      if (count == COUNT -1 )
      begin
         SCLK <= ~SCLK;
         count <= 0;
      end
      else
         count <= count + 1;
   end

Endmodule
```

Sixteen Hz Clock Divider:16.sv

```
module clk_div_16hz #(parameter COUNT = 2999999)(
   input CLK,
   output logic SCLK = 0
   );

   logic [31:0] count = 0;
   always_ff @ (posedge CLK)
   begin
      if (count == COUNT -1 )
      begin
         SCLK <= ~SCLK;
         count <= 0;
      end
      else
         count <= count + 1;
```

```
    end

endmodule
```

Four to One custom two bit SEL mux: Mux4_1.sv

```
module Mux4_1 #(parameter WIDTH = 1)(
    input [WIDTH - 1:0] A, B, C, D,
    input [1:0] SEL,
    output logic [WIDTH - 1:0] mux_out
    );

    always_comb
    begin
    case(SEL)
        4'd0: mux_out = A;
        4'd1: mux_out = B;
        4'd2: mux_out = C;
        4'd3: mux_out = D;

        default: mux_out = 0;
    endcase
    end
endmodule
```

Three to One 3-button custom mux: Mux3_1.sv

```
module Mux3_1 #(parameter WIDTH = 1)(
    input [WIDTH - 1:0] A, B, C,
    input BTN1,
    input BTN2,
    input BTN3,
    output logic [WIDTH - 1:0] mux_out
    );

    always_comb
    begin
    if (BTN1)
        mux_out = A;
```

```
    else if (BTN2)
        mux_out = B;
    else if (BTN3)
        mux_out = C;
    else
        mux_out = 0;

    end
endmodule
```

FSM Pattern Display One: Pattern.sv

```
module Pattern(
    input CLK,
    output logic [7:0] seg,
    output logic [3:0] an
    );

 typedef enum {S_top, S_topR, S_botR, S_bot, S_botL, S_topL, S_mid, S_top2, S_mid2,
              S2_top, S2_topR, S2_botR, S2_bot, S2_botL, S2_topL, S2_mid, S2_top2, S2_mid2,
              S3_top, S3_topR, S3_botR, S3_bot, S3_botL, S3_topL, S3_mid, S3_top2, S3_mid2,
              S4_top, S4_topR, S4_botR, S4_bot, S4_botL, S4_topL, S4_mid, S4_top2, S4_mid2}
STATES;
 STATES NS;
 STATES PS = S_top;

always_ff @ (posedge CLK)
begin
       PS <= NS;
end

always_comb
begin
    //initialize all outputs to zero
    case (PS)

        //FIRST DISPLAY
        S_top: begin
                NS = S_topR;
                seg = 8'b01111111;
                an = 4'b1110;
```

```
end
S_topR: begin
        NS = S_mid;
        seg = 8'b10111111;
        an = 4'b1110;
end
S_mid:begin
    NS = S_botL;
    seg = 8'b11111101;
    an = 4'b1110;
end
S_botL:begin
    NS = S_bot;
    seg = 8'b11110111;
    an = 4'b1110;
end
S_bot:begin
    NS = S_botR;
    seg = 8'b11101111;
    an = 4'b1110;
end
S_botR: begin
        NS = S_mid2;
        seg = 8'b11011111;
        an = 4'b1110;
end
S_mid2:begin
    NS = S_topL;
    seg = 8'b11111101;
    an = 4'b1110;
end
S_topL:begin
    NS = S_top2;
    seg = 8'b11111011;
    an = 4'b1110;
end
S_top2: begin
        NS = S2_top;
        seg = 8'b01111111;
        an = 4'b1110;
```

```
end


//SECOND DISPLAY
S2_top: begin
        NS = S2_topR;
        seg = 8'b01111111;
        an = 4'b1101;
end
S2_topR: begin
        NS = S2_mid;
        seg = 8'b10111111;
        an = 4'b1101;
end
S2_mid:begin
    NS = S2_botL;
    seg = 8'b11111101;
    an = 4'b1101;
end
S2_botL:begin
    NS = S2_bot;
    seg = 8'b11110111;
    an = 4'b1101;
end
S2_bot:begin
    NS = S2_botR;
    seg = 8'b11101111;
    an = 4'b1101;
end
S2_botR: begin
        NS = S2_mid2;
        seg = 8'b11011111;
        an = 4'b1101;
end
S2_mid2:begin
    NS = S2_topL;
    seg = 8'b11111101;
    an = 4'b1101;
end
S2_topL:begin
```

```
   NS = S2_top2;
   seg = 8'b11111011;
   an = 4'b1101;
end
S2_top2: begin
       NS = S3_top;
       seg = 8'b01111111;
       an = 4'b1101;
end

//THIRD DISPLAY
S3_top: begin
       NS = S3_topR;
       seg = 8'b01111111;
       an = 4'b1011;
end
S3_topR: begin
       NS = S3_mid;
       seg = 8'b10111111;
       an = 4'b1011;
end
S3_mid:begin
   NS = S3_botL;
   seg = 8'b11111101;
   an = 4'b1011;
end
S3_botL:begin
   NS = S3_bot;
   seg = 8'b11110111;
   an = 4'b1011;
end
S3_bot:begin
   NS = S3_botR;
   seg = 8'b11101111;
   an = 4'b1011;
end
S3_botR: begin
       NS = S3_mid2;
       seg = 8'b11011111;
       an = 4'b1011;
```

```
end
S3_mid2:begin
   NS = S3_topL;
   seg = 8'b11111101;
   an = 4'b1011;
end
S3_topL:begin
   NS = S3_top2;
   seg = 8'b11111011;
   an = 4'b1011;
end
S3_top2: begin
       NS = S4_top;
       seg = 8'b01111111;
       an = 4'b1011;
end

//FOURTH DISPLAY
S4_top: begin
       NS = S4_topR;
       seg = 8'b01111111;
       an = 4'b0111;
end
S4_topR: begin
       NS = S4_mid;
       seg = 8'b10111111;
       an = 4'b0111;
end
S4_mid:begin
   NS = S4_botL;
   seg = 8'b11111101;
   an = 4'b0111;
end
S4_botL:begin
   NS = S4_bot;
   seg = 8'b11110111;
   an = 4'b0111;
end
S4_bot:begin
   NS = S4_botR;
```

```
            seg = 8'b11101111;
            an = 4'b0111;
        end
        S4_botR: begin
                NS = S4_mid2;
                seg = 8'b11011111;
                an = 4'b0111;
        end
        S4_mid2:begin
            NS = S4_topL;
            seg = 8'b11111101;
            an = 4'b0111;
        end
        S4_topL:begin
            NS = S4_top2;
            seg = 8'b11111011;
            an = 4'b0111;
        end
        S4_top2: begin
                NS = S_top;
                seg = 8'b01111111;
                an = 4'b0111;
        end

        //DEFAULT
        default: begin
            NS = S_top;
            seg = 8'b01111111;
            an = 4'b1110;
            end
    endcase
end

endmodule
```

FSM Pattern Display Two: Pattern2.sv

```
module Pattern2(
    input CLK,
    output logic [7:0] seg,
```

```
   output logic [3:0] an
   );

 typedef enum {one_top, two_top, three_top, four_top, four_TR,
            four_mid, three_mid, two_mid, one_mid, one_BL,
            one_bot, two_bot, three_bot, four_bot, four_BR,
            four_mid2, three_mid2, two_mid2, one_mid2, one_TL} STATES;
 STATES NS;
 STATES PS = one_top;

always_ff @ (posedge CLK)
begin
      PS <= NS;
end

always_comb
begin
   //initialize all outputs to zero
   case (PS)

      //FIRST DISPLAY
      one_top: begin
            NS = two_top;
            seg = 8'b01111111;
            an = 4'b1110;
      end
      two_top: begin
            NS = three_top;
            seg = 8'b01111111;
            an = 4'b1101;
      end
      three_top: begin
            NS = four_top;
            seg = 8'b01111111;
            an = 4'b1011;
      end
      four_top: begin
            NS = four_TR;
            seg = 8'b01111111;
            an = 4'b0111;
```

```
end
four_TR: begin
        NS = four_mid;
        seg = 8'b10111111;
        an = 4'b0111;
end
four_mid: begin
        NS = three_mid;
        seg = 8'b11111101;
        an = 4'b0111;
end
three_mid: begin
        NS = two_mid;
        seg = 8'b11111101;
        an = 4'b1011;
end
two_mid: begin
        NS = one_mid;
        seg = 8'b11111101;
        an = 4'b1101;
end
one_mid: begin
        NS = one_BL;
        seg = 8'b11111101;
        an = 4'b1110;
end
one_BL: begin
        NS = one_bot;
        seg = 8'b11110111;
        an = 4'b1110;
end
one_bot: begin
        NS = two_bot;
        seg = 8'b11101111;
        an = 4'b1110;
end
two_bot: begin
        NS = three_bot;
        seg = 8'b11101111;
        an = 4'b1101;
```

```
end
three_bot: begin
        NS = four_bot;
        seg = 8'b11101111;
        an = 4'b1011;
end
four_bot: begin
        NS = four_BR;
        seg = 8'b11101111;
        an = 4'b0111;
end
four_BR: begin
        NS = four_mid2;
        seg = 8'b11011111;
        an = 4'b0111;
end
four_mid2: begin
        NS = three_mid2;
        seg = 8'b11111101;
        an = 4'b0111;
end
three_mid2: begin
        NS = two_mid2;
        seg = 8'b11111101;
        an = 4'b1011;
end
two_mid2: begin
        NS = one_mid2;
        seg = 8'b11111101;
        an = 4'b1101;
end
one_mid2: begin
        NS = one_TL;
        seg = 8'b11111101;
        an = 4'b1110;
end
one_TL: begin
        NS = one_top;
        seg = 8'b11111011;
        an = 4'b1110;
```

```
        end



        //DEFAULT
        default: begin
            NS = one_top;
            seg = 8'b01111111;
            an = 4'b1110;
            end
    endcase
end


endmodule
```

FSM Pattern Display Three: Pattern3.sv

```
module Pattern3(
    input CLK,
    output logic [7:0] seg,
    output logic [3:0] an
    );

 typedef enum {S_top, S_topR, S_botR, S_bot, S_botL, S_topL, S_top2,
              S2_top, S2_topR, S2_botR, S2_bot, S2_botL, S2_topL, S2_top2} STATES;
 STATES NS;
 STATES PS = S_top;

always_ff @ (posedge CLK)
begin
       PS <= NS;
end

always_comb
begin
    //initialize all outputs to zero
    case (PS)

        //FIRST AND THIRD DISPLAY
```

```
S_top: begin
        NS = S_topR;
        seg = 8'b01111111;
        an = 4'b1010;
end
S_topR: begin
        NS = S_botR;
        seg = 8'b10111111;
        an = 4'b1010;
end
S_botR: begin
        NS = S_bot;
        seg = 8'b11011111;
        an = 4'b1010;
end
S_bot:begin
    NS = S_botL;
    seg = 8'b11101111;
    an = 4'b1010;
end
S_botL:begin
    NS = S_topL;
    seg = 8'b11110111;
    an = 4'b1010;
end
S_topL:begin
    NS = S_top2;
    seg = 8'b11111011;
    an = 4'b1010;
end
S_top2: begin
        NS = S2_top;
        seg = 8'b01111111;
        an = 4'b1010;
end

//SECOND AND FOURTH DISPLAY
S2_top: begin
        NS = S2_topR;
        seg = 8'b01111111;
```

```
        an = 4'b0101;
end
S2_topR: begin
        NS = S2_botR;
        seg = 8'b10111111;
        an = 4'b0101;
end
S2_botR: begin
        NS = S2_bot;
        seg = 8'b11011111;
        an = 4'b0101;
end
S2_bot:begin
    NS = S2_botL;
    seg = 8'b11101111;
    an = 4'b0101;
end
S2_botL:begin
    NS = S2_topL;
    seg = 8'b11110111;
    an = 4'b0101;
end
S2_topL:begin
    NS = S2_top2;
    seg = 8'b11111011;
    an = 4'b0101;
end
S2_top2: begin
        NS = S_top;
        seg = 8'b01111111;
        an = 4'b0101;
end



//DEFAULT
default: begin
    NS = S_top;
    seg = 8'b01111111;
    an = 4'b1010;
```

```
            end
    endcase
end


endmodule
```