Trillanes, Lyle Denzell C.
2020-00175
Section - FVW

CoE 168 Machine Project Documentation

Introduction

This documentation discusses the thought process of the programmer and the inner workings of the code to make the code fit the given specifications. To keep the discussion simple, the programmer might just give a pseudocode in the discussions with how each milestone was approached through the coding process.

States

There are 4 states in total: s_init (for the initialization phase), s_wait (for the waiting for 2 seconds while displaying the name of the programmer while in the initialization phase), s_typemode1 (for the writing and printing of characters), and s_typemode2 (for the cursor movement and scrolling). A photo of the code snippet can be seen below for the state encoding done in verilog.

```
// state encoding
parameter s_init = 2'b00;
parameter s_wait = 2'b01;
parameter s_typemode1 = 2'b10;
parameter s_typemode2 = 2'b11;
```

Global Variables/Registers

In order to make the logic of the system work, the programmer made use of some global variables that he manipulated. Note, that some of these variables may be optimized—such as making use of the same variable rather than having a similar variable that would just copy the same values; due to the lack of time and resources (limited FPGA controller for testing), the programmer went the nitty gritty route of having a “kinda messy code” to suit its ability to see the code better. Discussion of the relevant registers might appear when discussing the operations of each of the states.

Similarly, a parameter “directory” was made for easy referencing the needed repetitive parameters such as the changing time delays, display commands, button definition (for calling debounced buttons), etc.

Button Debouncing Logic

The programmer has 2 implementations of button debouncing. The onboard FPGA buttons should have a mechanism that prevents the actual debounce by having an internal capacitance parallel with each button, but as mentioned above, due to the unavailability of FPGA at home for testing, a button debouncing was made taking in a button press as stable if it has been pressed for a certain period of time. The 2nd implementation of debouncing, is an “edge detection” type of debouncing because operating at 100 MHz, a single button press would've counted every clock cycle. As the name suggests, it is an edge detection for button presses.

An important feature, or portion that is made use by the programmer is a btn_pressed_reg and cmd_latch, which its logic is crucial because you couldnt pass the combinational circuit with a sequential, and the latter is important so that my commands in typemode finish executing all of its commands.

State Operation (init)

For the init state, I have a crucial portion of the code that leverages how well it will function in its operation. That portion is my global register started, which is declared globally to have a value of 0. Then upon startup, in the first else statement it will make its value into 1, thus on the succeeding nrst presses, it would start cmd_idx at cmd_idx == 2 (or the 3rd command). This is done because at start up of the LCD, we have to configure the operation as 4 bit mode (0010), then configure it as 4 bit mode, 2 lines (0010 1000). The logic for sending commands was done with the cmd_idx, and a separator of the sending of 4 nibbles at a time was done by the in_cmd_idx. Note, in_cmd_idx, was also utilized for configuring global variables before hand to ensore proper timing (such as preemping the RS value to 1 or 0 for 1.4 microsecs). A counter was also utilized to make use of the clocks operating frequency of 100 MHz, thus every clock cycle counts as 10 ns. Utilizing this a logic was done changing Some_delay to change the “timer” made for the operation. Additionally, each pulse width of the enable was configured to only off after achieving the necessary pulse width of enable. Another important factor of the implementation is the global variable of fake_en (which creatively in its name is to fake the enable) which if the command doesnt need to have an enable pulse will be set to 0 so that it would trigger the enable pulse.

State Operation (wait)

This was perhaps the simplest portion of the code. Using the earlier discussion of fake_en, counters, and pulse enable counters; with the 10ns per clock cycle, i just needed a counter to count a number of clock cycles to count for 2 seconds. After that, the commands for clear display were sent and the state was transitioned to type mode.

State Operation (typemode1)

The type mode is responsible for browsing characters and writing printable characters. This makes use of the earlier discussed of btn_pressed_reg from the button debouncing which uses combinational logic. As mentioned earlier, it needed to be passed down as a reg because we cannot pass it directly to our sequential logic. A case function was made use and a stored_vals register (which was initialized to have a value of 8'h20 during init) which is a 2D register of 8 by 80 to fit all of the stored values per memory address internally (which is important for the restoring of value—not overwriting unless pressed button0). The code makes use of the fact that the character scrolling is just 1 at a time so it was done by +1 or -1 on another register that contains the current_char (current _char is fed with the stored_vals before starting its operation to enforce the "memory" of the program). Button 2 and 3 or just mirrors of their implementation, a key thing to look for is that when writing to the LCD the cursor automatically moves to the next address, so we can just send a signal to go back previously by 1 address leveraging on cursor_pos register. Button0 is tricky, because it requires us to move the cursor, so the global register is +1 then no need to have it go back a previous address. However, an implementation in pressing button 0 requires display shift, so 3 cases was made: normal typing, out of bounds display, and wrapping back to the 1st line out of bounds display. A section will discuss how display shifting was handled. Button 1 which handles backspace, is a tricky because we need to go back first a from the previous address, then input a 8'h20 (space == blank line), this will simulate a delete function. Lastly, to enforce the "memory," just as we are switching to typemode 2, a logic was coded that returns the value of the current cursor according to the stored_vals (NOTE: the stored_vals will only change upon press of btn0 or btn1).

State Operation (typemode2)

This is responsible for cursor control, so to adequately discuss; we should just think of the execution of buttons 2 and 3 are mirrors of one another. Similar to the button 0 of the earlier discussed state, it moves the cursor + or - 1 depending on the button pressed, and like earlier, it is designed to handle display shift, in which we shall discuss in detail in this section. Display shifting was made possible by us making use of a display_window_max that is initialized to have a value of 15. Shifting of the display occurs when 2 things happen: 1. The cursor_pos is out of bounds of the display window, or 2. Cursor_pos had wrapped around the next line. For the 1st one it is simple, just shift the display once on the direction of the cursor movement using the display shift command. A key thing to note is that in button 3, it isn't comparing to display_window_max, but rather the min value which is the max value - 15, in order to save memory space than having another register for that operation (an example on how other codes could be optimized but due to earlier said constraints, was made rather hastely). Moving on to the 2nd, which is the wrapping, this occurs at the end of the lines, so instead of shifting once, it is doing the display shifting operations by 16. This

was made possible by a global variable of repeater, which is going back and forth of the commands with the in_cmd_idx. Button 0, or jumping from top to bottom (vice versa) of the current line was done by adding 64. Note, it is important that the adding is 64 rather than just 40 because the end of 1st line is 8'h27 and the 1st of 2nd line is actually 8'h40 (there is a gap in between, the space between them isnt actually just 40 from top to bottom). Lastly, a change in sw0 would require us to restore the current_char stored inside of the stored_vals so that we enforce the memory aspect of our code.

Some useful references

Some of the references was utilized by the programmer to help aid in the creation of this program. Aside from the given LCD user guide, this youtube video was especially useful to kickstart the progress in understanding the inner workings of the LCD [1]. I also made use of the delays made use in the LiquidCrystal.h arduino library (Note: while I made use of this reference, I still crossed reference with the User manual — some of the delays are still not easily found in the said library and the programmer had to trust his gut when setting the appropriate time delays)[2]. Lastly, a website showing the characters with the corresponding ascii was also made use to aid the programmer in typing his name for the init stage [3]. All of the above was also made into a simulation through tinkercad, which allowed the programmer to understand the inner workings of the LCD even more.

References

[1] M. Davis, "Datasheets: 16x2 LCD By Hand (No microcontroller)" YouTube, Oct. 14, 2023. [Online]. Available: https://www.youtube.com/watch?v=cXpeTxC3_A4. [Accessed: Dec. 17, 2024].

[2] Arduino Libraries, "LiquidCrystal/src/LiquidCrystal.h," GitHub Repository. [Online]. Available: https://github.com/arduino-libraries/LiquidCrystal/blob/master/src/LiquidCrystal.h. [Accessed: Dec. 17, 2024].

[3] Calculla, "ASCII Table," Calculla Website. [Online]. Available: http://v1.calculla.com/ascii_table#google_vignette. [Accessed: Dec. 17, 2024].

Code Appendix

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01.12.2024 12:27:41
// Design Name:
// Module Name: lcd_controller
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module lcd_controller(
    input wire clk,          // 100 MHz clock
    input wire nrst,         // Active-low reset
    input wire [3:0] buttons, // BTN0-BTN3 inputs
    input wire sw0,          // SW0 input
    output reg [3:0] data,    // Data lines DB4-DB7
    output reg enable,       // Enable signal
    output reg rs           // Register Select
);

    // Parameters for LCD instructions
    parameter CMD_CLEAR_DISPLAY = 8'h01; // clear display
    parameter CMD_RETURN_HOME   = 8'h02; // return home
    parameter CMD_DISPLAY_ON_W_CURSOR_BLINK = 8'h0F; // Display on, cursor blinking
    parameter CMD_FUNCTION_SET  = 8'h28; // 4 bit mode, 2-line
    parameter CMD_DISPLAY_SHIFT_RIGHT = 8'h18; // display shifts right
    parameter CMD_DISPLAY_SHIFT_LEFT = 8'h1C; // display shifts left
    parameter CMD_MOVE_CURSOR_2ND_LINE = 8'hC0; // moves the cursor to the 2nd line
    parameter CMD_MOVE_CURSOR_LEFT = 8'h10; // moves the cursor left
    parameter CMD_MOVE_CURSOR_RIGHT = 8'h14; // moves the cursor left

    // state encoding
    parameter s_init = 2'b00;
    parameter s_wait = 2'b01;
    parameter s_typemode1 = 2'b10;
    parameter s_typemode2 = 2'b11;

    // global registers
    // Declare stored_vals as a memory array for 80 addresses
    reg [7:0] stored_vals [103:0]; // not yet used but i want to make use of it for typemode, 40 lines for line 1 and 40 lines for line 2, thats why its 80, also 8 bits to store the character in each tile
    // Note: my implementation of above might be wrong
    reg [6:0] cursor_pos;    // 7-bit cursor position (0 to 79)
    reg [7:0] current_char;  // ASCII value of the current character

    reg [7:0] current_addr;
    reg [31:0] counter;
    reg [4:0] cmd_idx;
    reg [3:0] in_cmd_idx;//made into 4 bits for typemode purposes
    reg [1:0] state;
    reg [31:0] SOME_DELAY;
    reg [1:0] fake_en; //used for flagging if will trigger enable or not
    reg cmd_latch = 0;
    reg started = 0;
    reg [8:0] display_window_max;
    reg [5:0] repeater;

    // Maximum execution times in clock cycles (100 MHz clock)
    // will base the delays on the executions of the arduino library
    parameter T_CLEAR_DISPLAY = 32'd200000; // 2000 ms
    parameter T_RETURN_HOME   = 32'd200000; // 2000 ms
    parameter T_ENTRY_MODE    = 32'd4000;  // 40 µs
    parameter T_DISPLAY_ON    = 32'd4000;  // 40 µs
    parameter T_FUNCTION_SET  = 32'd4000;  // 40 µs
    parameter T_START         = 32'd2500000; // 25 ms
    parameter T_NIBBLE        = 32'd100; // 1 µs
    parameter T_WAIT          = 32'd200000000; // 2 seconds
    parameter T_ENABLE        = 32'd45; // 450 ns

    // Debounce counters for each button
    reg [2:0] btn_pressed_reg;
    reg [19:0] debounce_counter [3:0]; // Separate counters for BTN3, BTN2, BTN1, BTN0
    reg [3:0] btn_stable;             // Stable signals after debouncing

    // Debounce logic
    integer i;
    always @(posedge clk or negedge nrst) begin
        if (!nrst) begin
            debounce_counter[3] <= 0;
            debounce_counter[2] <= 0;
            debounce_counter[1] <= 0;
            debounce_counter[0] <= 0;
            btn_stable <= 4'b0000;
        end else begin
            for (i = 0; i < 4; i = i + 1) begin
                if (buttons[i] == 1) begin
                    if (debounce_counter[i] < 20'd100000) begin
                        debounce_counter[i] <= debounce_counter[i] + 1; // Increment debounce counter
                    end else begin
                        btn_stable[i] <= 1; // Button press is stable
                    end
                end else begin
                    debounce_counter[i] <= 0; // Reset debounce counter
```

```
            btn_stable[i] <= 0;      // Button is not pressed
        end
      end
    end
end


// this code is working but not debounced
/*always @(posedge clk or negedge nrst) begin
    if (!nrst) begin
        btn_stable <= 4'b0000;     // Reset stable button signals
    end else begin
        if (cmd_latch == 0) begin   // Only check buttons when cmd_latch is 0
            btn_stable <= buttons;  // Directly register current button states
        end else begin
            btn_stable <= 4'b0000;  // Ignore buttons while cmd_latch is active
        end
    end
end*/


// Button press detection
reg [2:0] btn_pressed; // Encodes which button is pressed

always @(*) begin
    if (cmd_latch == 0) begin //might change this if this didnt work
        if (btn_stable[3]) btn_pressed = 2'b11; // BTN3
        else if (btn_stable[2]) btn_pressed = 2'b10; // BTN2
        else if (btn_stable[1]) btn_pressed = 2'b01; // BTN1
        else if (btn_stable[0]) btn_pressed = 2'b00; // BTN0
        else btn_pressed = 3'b100; // No button pressed
    end
end

//do not use this
/*always @(*) begin
    if (cmd_latch == 0) begin
        if (btn_stable[3]) btn_pressed = BTN3;
        else if (btn_stable[2]) btn_pressed = BTN2;
        else if (btn_stable[1]) btn_pressed = BTN1;
        else if (btn_stable[0]) btn_pressed = BTN0;
        else btn_pressed = NONE;
    end else begin
        btn_pressed = NONE;  // Ignore button input when cmd_latch is active
    end
end*/

// Define button parameters for clarity
parameter BTN3 = 2'b11; // Corresponds to buttons[3]
parameter BTN2 = 2'b10; // Corresponds to buttons[2]
parameter BTN1 = 2'b01; // Corresponds to buttons[1]
parameter BTN0 = 2'b00; // Corresponds to buttons[0]
parameter NONE = 3'b100; // Corresponds to nothing pressed


//Turning this off muna to account for button debouncing 2.0
/*// Register logic for storing the button press
always @(posedge clk or negedge nrst) begin
    if (!nrst) begin
        btn_pressed_reg = 3'b100; // Reset button press
    end else if (cmd_latch == 0) begin
        btn_pressed_reg = btn_pressed; // Update with current btn_pressed value
    end
end*/

reg [3:0] btn_stable_prev;   // Previous state of debounced buttons
reg btn_latched;             // Latch to hold if a button press is detected

// Button Debouncing 2.0
// Edge detection for button presses
always @(posedge clk or negedge nrst) begin
    if (!nrst) begin
        btn_stable_prev <= 4'b0000;    // Reset previous state
        btn_pressed_reg <= NONE;       // Reset button press register
        btn_latched <= 0;              // Clear latch
    end else begin
        // Update previous state of btn_stable
        btn_stable_prev <= btn_stable;

        // Rising edge detection and latch logic
        if (cmd_latch == 0 && !btn_latched) begin
            case (btn_stable & ~btn_stable_prev)  // Detect rising edge
                4'b1000: begin
                    btn_pressed_reg <= BTN3; // BTN3 pressed
                    btn_latched <= 1;        // Set latch
                end
                4'b0100: begin
                    btn_pressed_reg <= BTN2; // BTN2 pressed
                    btn_latched <= 1;        // Set latch
                end
                4'b0010: begin
                    btn_pressed_reg <= BTN1; // BTN1 pressed
                    btn_latched <= 1;        // Set latch
                end
                4'b0001: begin
                    btn_pressed_reg <= BTN0; // BTN0 pressed
                    btn_latched <= 1;        // Set latch
                end
                default: begin
                    btn_pressed_reg <= NONE; // No new press
                    btn_latched <= 0;        // Clear latch
                end
            endcase
        end

        // Reset latch when cmd_latch is active (button processing starts)
        if (cmd_latch == 1) begin
            btn_latched <= 0;  // Clear latch to allow next detection
```

```
        end
    end
end

// ------------- Main File ---------------------- //
always @(posedge clk or negedge nrst) begin
    if (!nrst) begin
        // Reset sequence
        // Note Have to make this the reset sequence of the actual LCD
        counter = 0;
        cmd_idx = 0;
        state = s_init;
        enable = 0;
        rs = 0;
        data = 0;
        SOME_DELAY = T_START;
        in_cmd_idx = 0;
        fake_en = 1;
        current_char = 8'h20;
        cursor_pos = 0;
        cmd_latch = 0;
        display_window_max = 9'd15;
        repeater = 6'd0;
        // Initialize stored_vals to 8'h20
        for (i = 0; i < 103; i = i + 1) begin
            stored_vals[i] = 8'h20;
        end

        if (started == 1)begin // new addition as latch of startup
            cmd_idx = 2;
        end
    end else begin
        case (state)
            s_init: begin
                if (counter == SOME_DELAY) begin
                    started = 1;
                    counter = 0;
                    // Send commands based on cmd_idx
                    case (cmd_idx)
                        0: begin // send 0010 (4'h2)
                            data = CMD_FUNCTION_SET[7:4];
                            SOME_DELAY = T_FUNCTION_SET;
                            cmd_idx = cmd_idx + 1;
                        end
                        1: begin // send 0010 1000 (4 bit mode, 2 line)
                            if (in_cmd_idx == 0) begin
                                data = CMD_FUNCTION_SET[7:4];
                                SOME_DELAY = T_NIBBLE;
                                in_cmd_idx = in_cmd_idx + 1;
                            end else begin
                                data = CMD_FUNCTION_SET[3:0];
                                SOME_DELAY = T_FUNCTION_SET;
                                cmd_idx = cmd_idx + 1;
                                in_cmd_idx = 0;
                            end
                        end
                        2: begin // send clear display (0000 0001)
                            if (in_cmd_idx == 0) begin
                                data = CMD_CLEAR_DISPLAY[7:4];
                                SOME_DELAY = T_NIBBLE;
                                in_cmd_idx = in_cmd_idx + 1;
                            end else begin
                                data = CMD_CLEAR_DISPLAY[3:0];
                                SOME_DELAY = T_CLEAR_DISPLAY;
                                cmd_idx = cmd_idx + 1;
                                in_cmd_idx = 0;
                            end
                        end
                        3: begin // return home (0000 0010)
                            if (in_cmd_idx == 0) begin
                                data = CMD_RETURN_HOME[7:4];
                                SOME_DELAY = T_NIBBLE;
                                in_cmd_idx = in_cmd_idx + 1;
                            end else begin
                                data = CMD_RETURN_HOME[3:0];
                                SOME_DELAY = T_RETURN_HOME;
                                cmd_idx = cmd_idx + 1;
                                in_cmd_idx = 0;
                            end
                        end
                        4: begin // display settings (0000 1111)
                            if (in_cmd_idx == 0) begin
                                data = CMD_DISPLAY_ON_W_CURSOR_BLINK[7:4];
                                SOME_DELAY = T_NIBBLE;
                                in_cmd_idx = in_cmd_idx + 1;
                            end else if (in_cmd_idx == 1) begin
                                data = CMD_DISPLAY_ON_W_CURSOR_BLINK[3:0];
                                SOME_DELAY = T_DISPLAY_ON;
                                in_cmd_idx = in_cmd_idx + 1;
                            end else begin //turn RS early
                                rs = 1;
                                fake_en = 0; // NOTE set flag to 0 do that wont turn enable
                                SOME_DELAY = 27'd140;
                                cmd_idx = cmd_idx + 1;
                                in_cmd_idx = 0;
                            end
                        end
                        5: begin //write L
                            fake_en = 1; // return the flag to 1 so that enable would function
                            rs = 1;
                            if (in_cmd_idx == 0) begin
                                data = 4'h4;
                                SOME_DELAY = T_NIBBLE;
                                in_cmd_idx = in_cmd_idx + 1;
                            end else begin
                                data = 4'hc;
                                SOME_DELAY = T_DISPLAY_ON;//same delay as with sending naman
                                cmd_idx = cmd_idx + 1;
```

```
            in_cmd_idx = 0;
        end
    end
    6: begin //Y
        rs = 1;
        if (in_cmd_idx == 0) begin
            data = 4'h5;
            SOME_DELAY = T_NIBBLE;
            in_cmd_idx = in_cmd_idx + 1;
        end else begin
            data = 4'h9;
            SOME_DELAY = T_DISPLAY_ON;
            cmd_idx = cmd_idx + 1;
            in_cmd_idx = 0;
        end
    end
    7: begin //L
        rs = 1;
        if (in_cmd_idx == 0) begin
            data = 4'h4;
            SOME_DELAY = T_NIBBLE;
            in_cmd_idx = in_cmd_idx + 1;
        end else begin
            data = 4'hc;
            SOME_DELAY = T_DISPLAY_ON;
            cmd_idx = cmd_idx + 1;
            in_cmd_idx = 0;
        end
    end
    8: begin //E
        rs = 1;
        if (in_cmd_idx == 0) begin
            data = 4'h4;
            SOME_DELAY = T_NIBBLE;
            in_cmd_idx = in_cmd_idx + 1;
        end else if (in_cmd_idx == 1) begin
            data = 4'h5;
            SOME_DELAY = T_DISPLAY_ON;
            in_cmd_idx = in_cmd_idx + 1;
        end else begin //turning RS off early
            rs = 0;
            fake_en = 0; // fake enable so that no trigger again
            cmd_idx = cmd_idx + 1;
            SOME_DELAY = 27'd140;
            in_cmd_idx = 0;
        end
    end
    9: begin //MOVE CURSOR
        fake_en = 1;
        rs = 0;
        if (in_cmd_idx == 0) begin
            data = CMD_MOVE_CURSOR_2ND_LINE[7:4];
            SOME_DELAY = T_NIBBLE;
            in_cmd_idx = in_cmd_idx + 1;
        end else if (in_cmd_idx == 1) begin
            data = CMD_MOVE_CURSOR_2ND_LINE[3:0];
            SOME_DELAY = T_DISPLAY_ON; // no delay of specific but same naman
            in_cmd_idx = in_cmd_idx + 1;
        end else begin // turn rs on early again
            rs = 1;
            fake_en = 0; // fake en again so not to trigger enable
            cmd_idx = cmd_idx + 1;
            SOME_DELAY = 27'd140;
            in_cmd_idx = 0;
        end
    end
    10: begin //T
        fake_en = 1; // turn enable on again
        rs = 1;
        if (in_cmd_idx == 0) begin
            data = 4'h5;
            SOME_DELAY = T_NIBBLE;
            in_cmd_idx = in_cmd_idx + 1;
        end else begin
            data = 4'h4;
            SOME_DELAY = T_DISPLAY_ON;
            cmd_idx = cmd_idx + 1;
            in_cmd_idx = 0;
        end
    end
    11: begin //R
        rs = 1;
        if (in_cmd_idx == 0) begin
            data = 4'h5;
            SOME_DELAY = T_NIBBLE;
            in_cmd_idx = in_cmd_idx + 1;
        end else begin
            data = 4'h2;
            SOME_DELAY = T_DISPLAY_ON;
            cmd_idx = cmd_idx + 1;
            in_cmd_idx = 0;
        end
    end
    12: begin //I
        rs = 1;
        if (in_cmd_idx == 0) begin
            data = 4'h4;
            SOME_DELAY = T_NIBBLE;
            in_cmd_idx = in_cmd_idx + 1;
        end else begin
            data = 4'h9;
            SOME_DELAY = T_DISPLAY_ON;
            cmd_idx = cmd_idx + 1;
            in_cmd_idx = 0;
        end
    end
    13: begin //L
        rs = 1;
```

```
            if (in_cmd_idx == 0) begin
                data = 4'h4;
                SOME_DELAY = T_NIBBLE;
                in_cmd_idx = in_cmd_idx + 1;
            end else begin
                data = 4'hC;
                SOME_DELAY = T_DISPLAY_ON;
                cmd_idx = cmd_idx + 1;
                in_cmd_idx = 0;
            end
        end
        14: begin //L
            rs = 1;
            if (in_cmd_idx == 0) begin
                data = 4'h4;
                SOME_DELAY = T_NIBBLE;
                in_cmd_idx = in_cmd_idx + 1;
            end else begin
                data = 4'hC;
                SOME_DELAY = T_DISPLAY_ON;
                cmd_idx = cmd_idx + 1;
                in_cmd_idx = 0;
            end
        end
        15: begin //A
            rs = 1;
            if (in_cmd_idx == 0) begin
                data = 4'h4;
                SOME_DELAY = T_NIBBLE;
                in_cmd_idx = in_cmd_idx + 1;
            end else begin
                data = 4'h1;
                SOME_DELAY = T_DISPLAY_ON;
                cmd_idx = cmd_idx + 1;
                in_cmd_idx = 0;
            end
        end
        16: begin //N
            rs = 1;
            if (in_cmd_idx == 0) begin
                data = 4'h4;
                SOME_DELAY = T_NIBBLE;
                in_cmd_idx = in_cmd_idx + 1;
            end else begin
                data = 4'hE;
                SOME_DELAY = T_DISPLAY_ON;
                cmd_idx = cmd_idx + 1;
                in_cmd_idx = 0;
            end
        end
        17: begin //E
            rs = 1;
            if (in_cmd_idx == 0) begin
                data = 4'h4;
                SOME_DELAY = T_NIBBLE;
                in_cmd_idx = in_cmd_idx + 1;
            end else begin
                data = 4'h5;
                SOME_DELAY = T_DISPLAY_ON;
                cmd_idx = cmd_idx + 1;
                in_cmd_idx = 0;
            end
        end
        18: begin //S
            rs = 1;
            if (in_cmd_idx == 0) begin
                data = 4'h5;
                SOME_DELAY = T_NIBBLE;
                in_cmd_idx = in_cmd_idx + 1;
            end else if (in_cmd_idx == 1) begin
                data = 4'h3;
                SOME_DELAY = T_DISPLAY_ON;
                in_cmd_idx = in_cmd_idx + 1;
            end else begin
                rs = 0;
                fake_en = 0;
                cmd_idx = cmd_idx + 1;
                in_cmd_idx = 0;
                SOME_DELAY = 27'd140;
            end
        end
        19: begin //MOVE CURSOR UNSEEABLE
            fake_en = 1;
            rs = 0;
            if (in_cmd_idx == 0) begin
                data = 4'hE;
                SOME_DELAY = T_NIBBLE;
                in_cmd_idx = in_cmd_idx + 1;
            end else begin
                data = 4'H7;
                SOME_DELAY = T_DISPLAY_ON;
                cmd_idx = cmd_idx + 1;
                in_cmd_idx = 0;
            end
        end
        20: begin
            rs = 0;
            state = s_wait;
            SOME_DELAY = T_WAIT;
            fake_en = 0; // fake enable so that no trigger again
        end

        default: begin
            state = s_wait; // Proceed to wait state
            SOME_DELAY = T_WAIT;
        end
    endcase
    if (fake_en == 1)begin
```

```
            enable = 1; // Generate enable pulse
        end
    end if (enable == 1 && counter == T_ENABLE) begin
        enable = 0; // makes sure that enable is on for 450 ns
    end else begin
        counter = counter + 1;
    end
end

// ------------ wait for 2 seconds then clear ---- //
s_wait: begin

    if (counter == SOME_DELAY) begin
        rs = 0;
        counter = 0;
        if (in_cmd_idx == 0) begin
            fake_en = 1;
            data = CMD_CLEAR_DISPLAY[7:4];
            SOME_DELAY = T_NIBBLE;
            in_cmd_idx = in_cmd_idx + 1;
        end else if (in_cmd_idx == 1) begin
            data = CMD_CLEAR_DISPLAY[3:0];
            SOME_DELAY = T_CLEAR_DISPLAY;
            in_cmd_idx = in_cmd_idx + 1;
        end else begin
            state = s_typemode1;
            in_cmd_idx = 0;
            rs = 1;
            fake_en = 0;
        end
        if (fake_en == 1)begin
            enable = 1; // Generate enable pulse
        end
    end if (enable == 1 && counter == T_ENABLE) begin
        enable = 0; // makes sure that enable is on for 450 ns
    end else begin
        counter = counter + 1; // Increment counter
    end
end

// To follow for the code of this ----------------------------------
s_typemode1: begin // swith is down (browsing of printable characters)
    if (sw0) begin

        if (enable == 1 && counter == T_ENABLE) begin
            enable = 0; // Ensure enable is deasserted after pulse

        end if (counter == SOME_DELAY) begin
        //insert redo here
            counter = 0;
            if (in_cmd_idx == 0) begin
                rs = 1; // Data mode
                fake_en = 1;
                data = stored_vals[cursor_pos][7:4]; // Upper nibble
                SOME_DELAY = T_NIBBLE;
                in_cmd_idx = in_cmd_idx + 1;
            end else if (in_cmd_idx == 1) begin
                data = stored_vals[cursor_pos][3:0]; // Lower nibble
                SOME_DELAY = T_DISPLAY_ON;
                in_cmd_idx = in_cmd_idx + 1;
                //btn_pressed_reg = 3'b100;
            end else if (in_cmd_idx == 2) begin
                // Return to cursor position
                rs = 0; // Command mode
                SOME_DELAY = 27'd140;
                in_cmd_idx = in_cmd_idx + 1;
                fake_en = 0;
            end else if (in_cmd_idx == 3) begin
                fake_en = 1;
                data = {1'b1, cursor_pos[6:4]}; // Move to top line position
                SOME_DELAY = T_NIBBLE;
                in_cmd_idx = in_cmd_idx + 1;
            end else if (in_cmd_idx == 4) begin
                data = cursor_pos[3:0]; // Move to top line position
                SOME_DELAY = T_DISPLAY_ON;
                in_cmd_idx = in_cmd_idx + 1;
            end else begin
                in_cmd_idx = 0;
                fake_en = 0;
                state = s_typemode2; // Switch to typemode2 if SW0 is UP
            end

        if (fake_en == 1)begin
            enable = 1; // Generate enable pulse
        end
        end else begin
            counter = counter + 1; // Increment counter
        end
    end else begin
        // counter for enable pulse
        if (enable == 1 && counter == T_ENABLE) begin
            enable = 0; // Ensure enable is deasserted after pulse

        end if (counter == SOME_DELAY) begin
        // write logic code here //
            counter = 0; // Reset counter
            case (btn_pressed_reg)
                BTN2: begin // Browse characters forward

                    if (cmd_latch == 0) begin
                        cmd_latch = 1;
                        current_char = (current_char == 8'h7E) ? 8'h20 : current_char + 1;
                        //btn_pressed_reg = BTN2;
                    end

                    if (in_cmd_idx == 0) begin
                        rs = 1; // Data mode
                        fake_en = 1;
```

```
            data = current_char[7:4]; // Upper nibble
            SOME_DELAY = T_NIBBLE;
            in_cmd_idx = in_cmd_idx + 1;
        end else if (in_cmd_idx == 1) begin
            data = current_char[3:0]; // Lower nibble
            SOME_DELAY = T_DISPLAY_ON;
            in_cmd_idx = in_cmd_idx + 1;
        end else if (in_cmd_idx == 2) begin
            // Return to cursor position
            rs = 0; // Command mode
            SOME_DELAY = 27'd140;
            in_cmd_idx = in_cmd_idx + 1;
            fake_en = 0;
        end else if (in_cmd_idx == 3) begin
            fake_en = 1;
            data = {1'b1, cursor_pos[6:4]}; // Move to top line position
            SOME_DELAY = T_NIBBLE;
            in_cmd_idx = in_cmd_idx + 1;
        end else if (in_cmd_idx == 4) begin
            data = cursor_pos[3:0]; // Move to top line position
            SOME_DELAY = T_DISPLAY_ON;
            in_cmd_idx = in_cmd_idx + 1;
        end else if (in_cmd_idx == 5) begin
            rs = 1;
            SOME_DELAY = 27'd140;
            in_cmd_idx = 0;
            cmd_latch = 0;
            fake_en = 0;
            //btn_pressed_reg = 3'b100;
        end
    end

    BTN3: begin // Browse characters backward

        if (cmd_latch == 0) begin
            cmd_latch = 1;
            current_char = (current_char == 8'h20) ? 8'h7E : current_char - 1;
            //btn_pressed_reg = BTN3;
        end

        if (in_cmd_idx == 0) begin
            rs = 1; // Data mode
            fake_en = 1;
            data = current_char[7:4]; // Upper nibble
            SOME_DELAY = T_NIBBLE;
            in_cmd_idx = in_cmd_idx + 1;
        end else if (in_cmd_idx == 1) begin
            data = current_char[3:0]; // Lower nibble
            SOME_DELAY = T_DISPLAY_ON;
            in_cmd_idx = in_cmd_idx + 1;
        end else if (in_cmd_idx == 2) begin
            // Return to cursor position
            rs = 0; // Command mode
            SOME_DELAY = 27'd140;
            fake_en = 0;
            in_cmd_idx = in_cmd_idx + 1;
        end else if (in_cmd_idx == 3) begin
            fake_en = 1;
            data = {1'b1, cursor_pos[6:4]}; // Move to top line position
            SOME_DELAY = T_NIBBLE;
            in_cmd_idx = in_cmd_idx + 1;
        end else if (in_cmd_idx == 4) begin
            data = cursor_pos[3:0]; // Move to top line position
            SOME_DELAY = T_DISPLAY_ON;
            in_cmd_idx = in_cmd_idx + 1;
        end else if (in_cmd_idx == 5) begin
            rs = 1;
            fake_en = 0;
            SOME_DELAY = 27'd140;
            in_cmd_idx = 0;
            cmd_latch = 0;
            //btn_pressed_reg = 3'b100;
        end
    end

    BTN0: begin // Write character to memory and move cursor

        if (cmd_latch == 0) begin
            cmd_latch = 1;
            //btn_pressed_reg = BTN0;
            stored_vals[cursor_pos] = current_char; // Save to internal memory
            if (cursor_pos == 39) cursor_pos = cursor_pos + 25; // Move to bottom line
            else cursor_pos = cursor_pos + 1; // advance position by 1
            //write if (cursor_pos == 103) cursor_pos = 0;
        end

        //write logic of shifting here
        // insert code for display window here
        if (cursor_pos == 64) begin
            //do action to move display
            if (in_cmd_idx == 0) begin
                rs = 0; // Command mode
                fake_en = 0;
                SOME_DELAY = 27'd140;
                rs = 0;
                in_cmd_idx = 1;
            end else if (in_cmd_idx == 1) begin
                fake_en = 1;
                data = 4'h1; // Upper nibble
                SOME_DELAY = T_NIBBLE;
                in_cmd_idx = in_cmd_idx + 1;
            end else if (in_cmd_idx == 2) begin
                data = 4'h8; // Lower nibble
                SOME_DELAY = T_DISPLAY_ON;
                repeater = repeater + 1;
                if (repeater < 16) begin
                    in_cmd_idx = in_cmd_idx - 1;
                end else begin
```

```
                in_cmd_idx = in_cmd_idx + 1;
                repeater = 0;
            end
        end else if (in_cmd_idx == 3) begin
            rs = 1; // Type mode
            fake_en = 0;
            SOME_DELAY = 27'd140;
            rs = 0;
            in_cmd_idx = 1 + in_cmd_idx;
        end else if (in_cmd_idx == 4) begin
            rs = 1; // Data mode
            fake_en = 1;
            data = current_char[7:4]; // Upper nibble
            SOME_DELAY = T_NIBBLE;
            in_cmd_idx = in_cmd_idx + 1;
        end else if (in_cmd_idx == 5) begin
            data = current_char[3:0]; // Lower nibble
            SOME_DELAY = T_DISPLAY_ON;
            in_cmd_idx = 0;
            cmd_latch = 0;
            current_char = stored_vals[cursor_pos];
            display_window_max = 64 + 15;
            //btn_pressed_reg = 3'b100;
        end
    end else if (cursor_pos > display_window_max) begin
        //do action to move display
        if (in_cmd_idx == 0) begin
            rs = 0; // Command mode
            fake_en = 0;
            SOME_DELAY = 27'd140;
            rs = 0;
            in_cmd_idx = 1;
        end else if (in_cmd_idx == 1) begin
            fake_en = 1;
            data = 4'h1; // Upper nibble
            SOME_DELAY = T_NIBBLE;
            in_cmd_idx = in_cmd_idx + 1;
        end else if (in_cmd_idx == 2) begin
            data = 4'h8; // Lower nibble
            SOME_DELAY = T_DISPLAY_ON;
            in_cmd_idx = in_cmd_idx + 1;
            //btn_pressed_reg = 3'b100;
        end else if (in_cmd_idx == 3) begin
            rs = 1; // Type mode
            fake_en = 0;
            SOME_DELAY = 27'd140;
            rs = 0;
            in_cmd_idx = 1 + in_cmd_idx;
        end else if (in_cmd_idx == 4) begin
            rs = 1; // Data mode
            fake_en = 1;
            data = current_char[7:4]; // Upper nibble
            SOME_DELAY = T_NIBBLE;
            in_cmd_idx = in_cmd_idx + 1;
        end else if (in_cmd_idx == 5) begin
            data = current_char[3:0]; // Lower nibble
            SOME_DELAY = T_DISPLAY_ON;
            in_cmd_idx = 0;
            cmd_latch = 0;
            current_char = stored_vals[cursor_pos];
            display_window_max = display_window_max + 1;
            //btn_pressed_reg = 3'b100;
        end
    end else begin
        // original code to be found inside else block
        if (in_cmd_idx == 0) begin
            rs = 1; // Data mode
            fake_en = 1;
            data = current_char[7:4]; // Upper nibble
            SOME_DELAY = T_NIBBLE;
            in_cmd_idx = in_cmd_idx + 1;
        end else if (in_cmd_idx == 1) begin
            data = current_char[3:0]; // Lower nibble
            SOME_DELAY = T_DISPLAY_ON;
            in_cmd_idx = 0;
            cmd_latch = 0;
            current_char = stored_vals[cursor_pos];
            //btn_pressed_reg = 3'b100;
        end
    end

end

BTN1: begin // Erase character

    if (cursor_pos > 0 && cmd_latch == 0) begin
        cmd_latch = 1;
        cursor_pos = cursor_pos - 1;
        stored_vals[cursor_pos] = current_char;
        //btn_pressed_reg = BTN1;
    end
    if (in_cmd_idx == 0) begin

        fake_en = 0;
        rs = 0;
        SOME_DELAY = 27'd140;
        in_cmd_idx = in_cmd_idx + 1;
    end else if (in_cmd_idx == 1) begin
        fake_en = 1;
        data = {1'b1, cursor_pos[6:4]}; // Upper nibble (space)
        SOME_DELAY = T_NIBBLE;
        in_cmd_idx = in_cmd_idx + 1;
    end else if (in_cmd_idx == 2) begin
        data = cursor_pos[3:0]; // Upper nibble (space)
        SOME_DELAY = T_DISPLAY_ON;
        in_cmd_idx = in_cmd_idx + 1;
    end else if (in_cmd_idx == 3) begin
        rs = 1;
```

```
                    fake_en = 0;
                    SOME_DELAY = 27'd140;
                    in_cmd_idx = in_cmd_idx + 1;
                end else if (in_cmd_idx == 4) begin
                    fake_en = 1;
                    data = 4'h2; // Upper nibble (space)
                    SOME_DELAY = T_NIBBLE;
                    in_cmd_idx = in_cmd_idx + 1;
                end else if (in_cmd_idx == 5) begin
                    data = 4'h0; // Upper nibble (space)
                    SOME_DELAY = T_DISPLAY_ON;
                    in_cmd_idx = 0;
                    cmd_latch = 0;
                    //btn_pressed_reg = 3'b100;
                end
            end
            default: begin
                //btn_pressed_reg = 3'b100;
                fake_en = 0;
                cmd_latch = 0;
            end
        endcase
        if (fake_en == 1)begin
            enable = 1; // Generate enable pulse
        end
    // note I want my buttons to have some form of debounce
    end else begin
        counter = counter + 1; // Increment counter
    end
  end
end


s_typemode2: begin // switch is UP (cursor control)
  if (!sw0) begin
    state = s_typemode1; // Switch to typemode1 if SW0 is DOWN
    current_char = stored_vals[cursor_pos];
  end else begin
    // counter for enable pulse
    if (enable == 1 && counter == T_ENABLE) begin
        enable = 0; // Ensure enable is deasserted after pulse

    end if (counter == SOME_DELAY) begin
    // write logic code here //
        counter = 0; // Reset counter
        case (btn_pressed_reg)
            BTN3: begin // Move cursor left

                if (cmd_latch == 0) begin
                    cmd_latch = 1;
                    //btn_pressed_reg = BTN3;
                    if (cursor_pos > 63)begin
                        cursor_pos = (cursor_pos == 64) ? 39 : cursor_pos - 1; // Wrap cursor
                    end else begin
                        cursor_pos = (cursor_pos == 0) ? 103 : cursor_pos - 1; // Wrap cursor
                    end
                end
                // insert code for display window here
                if (cursor_pos == 39 || cursor_pos == 103) begin
                    //do action to move display
                    if (in_cmd_idx == 0) begin
                        rs = 0; // Command mode
                        fake_en = 0;
                        SOME_DELAY = 27'd140;
                        rs = 0;
                        in_cmd_idx = 1;

                    end else if (in_cmd_idx == 1) begin
                        fake_en = 1;
                        data = {1'b1,cursor_pos[6:4]}; // Upper nibble
                        SOME_DELAY = T_NIBBLE;
                        in_cmd_idx = in_cmd_idx + 1;
                    end else if (in_cmd_idx == 2) begin
                        data = cursor_pos[3:0]; // Lower nibble
                        SOME_DELAY = T_DISPLAY_ON;
                        in_cmd_idx = in_cmd_idx +1;
                        //btn_pressed_reg = 3'b100;
                    end else if (in_cmd_idx == 3) begin
                        fake_en = 1;
                        data = 4'h1; // Upper nibble
                        SOME_DELAY = T_NIBBLE;
                        in_cmd_idx = in_cmd_idx + 1;
                    end else if (in_cmd_idx == 4) begin
                        data = 4'hc; // Lower nibble
                        SOME_DELAY = T_DISPLAY_ON;
                        repeater = repeater + 1;
                        if (repeater < 16) begin
                            in_cmd_idx = in_cmd_idx - 1;
                        end else begin
                            in_cmd_idx = 0;
                            cmd_latch = 0;
                            repeater = 0;
                            if (cursor_pos == 39) display_window_max = 39;
                            if (cursor_pos == 103) display_window_max = 103;
                        end
                        //btn_pressed_reg = 3'b100;
                    end
                end else if (cursor_pos < (display_window_max - 15)) begin
                    //do action to move display
                    if (in_cmd_idx == 0) begin
                        rs = 0; // Command mode
                        fake_en = 0;
                        SOME_DELAY = 27'd140;
                        rs = 0;
                        in_cmd_idx = 1;

                    end else if (in_cmd_idx == 1) begin
                        fake_en = 1;
```

```
                data = {1'b1,cursor_pos[6:4]}; // Upper nibble
                SOME_DELAY = T_NIBBLE;
                in_cmd_idx = in_cmd_idx + 1;
            end else if (in_cmd_idx == 2) begin
                data = cursor_pos[3:0]; // Lower nibble
                SOME_DELAY = T_DISPLAY_ON;
                in_cmd_idx = in_cmd_idx +1;
                //btn_pressed_reg = 3'b100;
            end else if (in_cmd_idx == 3) begin
                fake_en = 1;
                data = 4'h1; // Upper nibble
                SOME_DELAY = T_NIBBLE;
                in_cmd_idx = in_cmd_idx + 1;
            end else if (in_cmd_idx == 4) begin
                data = 4'hc; // Lower nibble
                SOME_DELAY = T_DISPLAY_ON;
                in_cmd_idx = 0;
                display_window_max = display_window_max - 1;
                cmd_latch = 0;
                //btn_pressed_reg = 3'b100;
            end
        end else begin
            if (in_cmd_idx == 0) begin
                rs = 0; // Command mode
                fake_en = 0;
                SOME_DELAY = 27'd140;
                rs = 0;
                in_cmd_idx = 1;
            end else if (in_cmd_idx == 1) begin
                fake_en = 1;
                data = {1'b1,cursor_pos[6:4]}; // Upper nibble
                SOME_DELAY = T_NIBBLE;
                in_cmd_idx = in_cmd_idx + 1;
            end else if (in_cmd_idx == 2) begin
                data = cursor_pos[3:0]; // Lower nibble
                SOME_DELAY = T_DISPLAY_ON;
                in_cmd_idx = 0;
                cmd_latch = 0;
                //btn_pressed_reg = 3'b100;
            end
        end
    end
    BTN2: begin // Move cursor right

        if (cmd_latch == 0) begin
            cmd_latch = 1;
            //btn_pressed_reg = BTN2;
            if (cursor_pos < 40) begin
                cursor_pos = (cursor_pos == 39) ? 64 : cursor_pos + 1; // Wrap cursor
            end else begin
                cursor_pos = (cursor_pos == 103) ? 0 : cursor_pos + 1; // Wrap cursor
            end
        end
        // insert code for display window here
        if (cursor_pos == 64 || cursor_pos == 0) begin
            //do action to move display
            if (in_cmd_idx == 0) begin
                rs = 0; // Command mode
                fake_en = 0;
                SOME_DELAY = 27'd140;
                rs = 0;
                in_cmd_idx = 1;

            end else if (in_cmd_idx == 1) begin
                fake_en = 1;
                data = {1'b1,cursor_pos[6:4]}; // Upper nibble
                SOME_DELAY = T_NIBBLE;
                in_cmd_idx = in_cmd_idx + 1;
            end else if (in_cmd_idx == 2) begin
                data = cursor_pos[3:0]; // Lower nibble
                SOME_DELAY = T_DISPLAY_ON;
                in_cmd_idx = in_cmd_idx +1;
                //btn_pressed_reg = 3'b100;
            end else if (in_cmd_idx == 3) begin
                fake_en = 1;
                data = 4'h1; // Upper nibble
                SOME_DELAY = T_NIBBLE;
                in_cmd_idx = in_cmd_idx + 1;
            end else if (in_cmd_idx == 4) begin
                data = 4'h8; // Lower nibble
                SOME_DELAY = T_DISPLAY_ON;
                repeater = repeater + 1;
                if (repeater < 16) begin
                    in_cmd_idx = in_cmd_idx - 1;
                end else begin
                    in_cmd_idx = 0;
                    cmd_latch = 0;
                    repeater = 0;
                    if (cursor_pos == 64) display_window_max = 64 + 15;
                    if (cursor_pos == 0) display_window_max = 0 + 15;
                end
                //btn_pressed_reg = 3'b100;
            end
        end else if (cursor_pos > display_window_max) begin
            //do action to move display
            if (in_cmd_idx == 0) begin
                rs = 0; // Command mode
                fake_en = 0;
                SOME_DELAY = 27'd140;
                rs = 0;
                in_cmd_idx = 1;

            end else if (in_cmd_idx == 1) begin
                fake_en = 1;
                data = {1'b1,cursor_pos[6:4]}; // Upper nibble
                SOME_DELAY = T_NIBBLE;
                in_cmd_idx = in_cmd_idx + 1;
            end else if (in_cmd_idx == 2) begin
```

```
                        data = cursor_pos[3:0]; // Lower nibble
                        SOME_DELAY = T_DISPLAY_ON;
                        in_cmd_idx = in_cmd_idx +1;
                        //btn_pressed_reg = 3'b100;
                    end else if (in_cmd_idx == 3) begin
                        fake_en = 1;
                        data = 4'h1; // Upper nibble
                        SOME_DELAY = T_NIBBLE;
                        in_cmd_idx = in_cmd_idx + 1;
                    end else if (in_cmd_idx == 4) begin
                        data = 4'h8; // Lower nibble
                        SOME_DELAY = T_DISPLAY_ON;
                        display_window_max = display_window_max + 1;
                        in_cmd_idx = 0;
                        cmd_latch = 0;
                        //btn_pressed_reg = 3'b100;
                    end
                end else begin
                    if (in_cmd_idx == 0) begin
                        rs = 0; // Command mode
                        fake_en = 0;
                        SOME_DELAY = 27'd140;
                        rs = 0;
                        in_cmd_idx = 1;
                    end else if (in_cmd_idx == 1) begin
                        fake_en = 1;
                        data = {1'b1,cursor_pos[6:4]}; // Upper nibble
                        SOME_DELAY = T_NIBBLE;
                        in_cmd_idx = in_cmd_idx + 1;
                    end else if (in_cmd_idx == 2) begin
                        data = cursor_pos[3:0]; // Lower nibble
                        SOME_DELAY = T_DISPLAY_ON;
                        in_cmd_idx = 0;
                        cmd_latch = 0;
                        //btn_pressed_reg = 3'b100;
                    end
                end
            end
            BTN0: begin // Switch between lines

                if (cmd_latch == 0) begin
                    cmd_latch = 1;
                    //btn_pressed_reg = BTN0;
                    if (cursor_pos < 40) begin
                        cursor_pos = cursor_pos + 64; // Move to bottom line
                        display_window_max = display_window_max + 64;
                    end else begin
                        cursor_pos = cursor_pos - 64; // Move to top line
                        display_window_max = display_window_max - 64;
                    end
                end

                if (in_cmd_idx == 0) begin
                    rs = 0; // Command mode
                    fake_en = 0;
                    SOME_DELAY = 27'd140;
                    rs = 0;
                    in_cmd_idx = 1;
                end else if (in_cmd_idx == 1) begin
                    fake_en = 1;
                    data = {1'b1,cursor_pos[6:4]}; // Upper nibble
                    SOME_DELAY = T_NIBBLE;
                    in_cmd_idx = in_cmd_idx + 1;
                end else begin
                    data = cursor_pos[3:0]; // Lower nibble
                    SOME_DELAY = T_DISPLAY_ON;
                    in_cmd_idx = 0;
                    //btn_pressed_reg = 3'b100;
                    cmd_latch = 0;
                end
            end
            default: begin
                //btn_pressed_reg = 3'b100;
                fake_en = 0;
                cmd_latch = 0;
            end
        endcase
        if (fake_en == 1)begin
            enable = 1; // Generate enable pulse
        end
    // note I want my buttons to have some form of debouncer
    end else begin
        counter = counter + 1; // Increment counter
    end
  end
 end
 default: state <= s_wait; // Fallback
 endcase
 end
 end
endmodule
```