Scrolling Display

Jordan Reeser

November 2, 2018

1 Executive Summary

In this lab, students were responsible for creating a scrolling display utilizing a BRAM module on the Nexys DDR 4 board. The program starts by scrolliing the default BRAM data, all zeros, across the 8-digit seven-segment display. When the user presses the program button, the LEDs above the 16 switches light up to signify the user to use the switches to input 4 digits of hex data into the BRAM. The data is loaded into the BRAM when the user presses the debounced enter button. Then the LEDs turn off and the data will scroll across the 8-digit display when the BRAM output counter reaches the input address. The program and enter process can be repeated as many times as the user desires until the reset button is pressed. The link to the github repository for this design is: `https://github.com/Joreeser/Scrolling_Display`

2 Board Setup

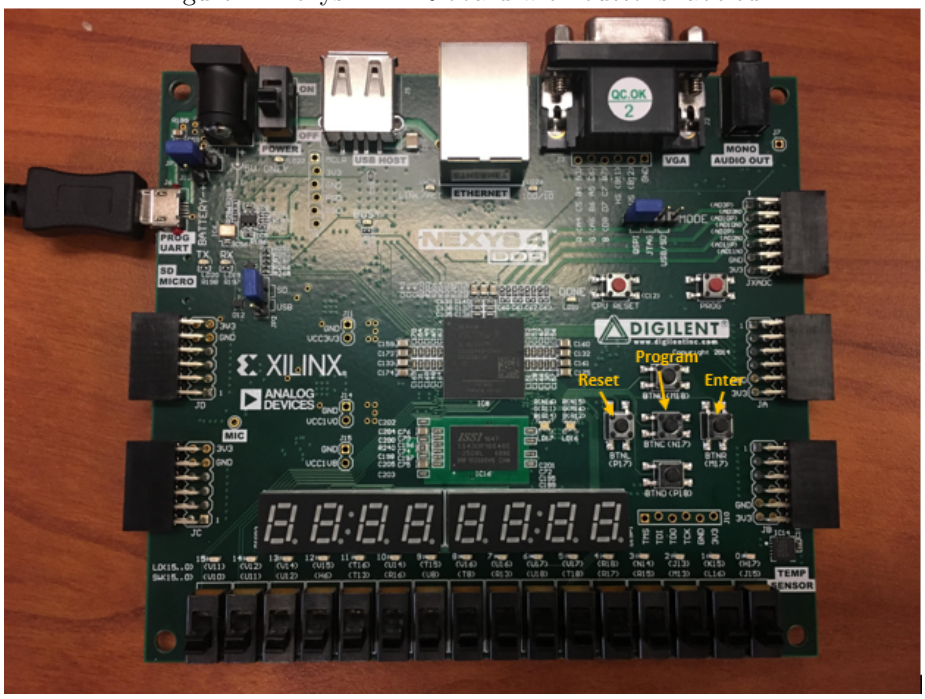
For this design, 3 buttons on the Nexys 4 DDR board are utilized as the reset, program, and enter buttons. The 16 switches are used to input data into the BRAM. The 16 LEDs above the switches and the 8-digit seven segment display are used for outputs. For ease of use, the designer decided to use the 3 horizontal buttons on the board (btnL, btnC, and btnR) for reset, program, and enter.

2.1 Reset, Program, and Enter

The designer decided to use the three horizontal buttons as reset, program (prog), and enter (enter_btn) since this setup appeared to be the most intuitive design. The left button was utilized for reset instead of the board reset button since the board reset button is logic level high when unpressed, which caused issues with the program.

1. Reset: btnL
2. Program (prog): btnC
3. Enter (enter_btn): btnR

Figure 1: Nexys 4 DDR board with buttons labeled.



3 Functionality

To control the functionality of the project, a state machine was used with the following states: DEF_DISP, PROG, and DISPLAY.

3.1 DEF_DISP

This state is the default display for the program. It scrolls the initial BRAM data, all zeros, across the display. It also initializes both of the enable signals for the BRAM high and the LEDs off. Write enable for the BRAM is also set low for this module since no data should be written to memory. The next state logic for this state sets the next state to PROG if the program (prog) button is pressed.

3.2 PROG

This is the programming state of the design. In this state, the LEDs above the 16 switches are turned on and the BRAM input is set to the switches. The next state logic sets the next state to DISPLAY and sets the write enable high when the debounced enter button is pressed. This ensures that only one set of data

is entered into the BRAM.

3.3 DISPLAY

In this state, the loaded BRAM data is scrolled across the 8-digit display. The write enable signal is set to zero so that it is only high for one clock cycle, ensuring that data is only entered once. The LEDs are also turned off to signify that the program is out of the programming state. The next state logic sets the next state to DEF_DISP if reset is pressed or to PROG if the program (prog) button is pressed.

4 Modules

To implement the reaction timer design, the following modules were used:

1. scrolling_display
2. bram
3. debounce
4. mod_m_counter
5. univ_shift_reg
6. counter
7. hex_to_sseg
8. disp_mux

4.1 scrolling_display

The scrolling_display module is the top module for the project. It connects the inputs and outputs to the board and includes all of the state logic and other module instantiation.

4.2 bram

The bram module was created with the Xilinx IP wizard and wrapped with an HDL wrapper to correctly utilize the board BRAM. For this design, a two port bram was created to enter and output data.

4.3 debounce

This module debounces a desired button. For this design, it is used to debounce the enter button so that data was only input into the BRAM once. This code was written by Pong Chu and levereged from his book *FPGA Prototyping by SystemVerilog Examples*.

4.4 mod_m_counter

This module is a simple mod m counter that is used to generate a tick after a specified amount of clock cycles. It is used to create a tick every second for the shift register and BRAM output address counter. This code was written by Pong Chu and levereged from his book *FPGA Prototyping by SystemVerilog Examples*.

4.5 univ_shift_reg

This module takes in 32-bit data and shifts 4 bits at a time to shift by one hex digit. The inputs come from the output of the BRAM and the output is the hex values to be displayed on the 8-digit display. This module was originally created by Pong Chu in his book *FPGA Prototyping by SystemVerilog Examples*, and was modified to shift 4 bits instead of 1 for use in this lab.

4.6 counter

This module is a basic counter module. It increments count every time a clock tick is present, and reset sets the count value back to zero. Two instances of this module were needed for the lab. One instance is 6-bit and is used to increment the output address of the BRAM every time there is a tick from the mod m counter. The second instance is 4-bit and is used to increment the input address of the BRAM every time the debounced enter button is pressed.

4.7 hex_to_sseg

The hex_to_sseg module converts hex values fed into the module to binary values that will turn on the proper segments of the seven-segment display to show the hex value. Eight instances of this module are used for each of the displays. This module was originally created by Pong Chu in his book *FPGA Prototyping by SystemVerilog Examples*.

4.8 disp_mux

This module is used in conjunction with the hex_to_sseg module to display the values in the proper display units on the board. It multiplexes the display annodes to turn on one display at a time with the appropriate display value, and cycles through these displays at a rate that is too fast for a human viewer to discern. This module was created by Pong Chu and levereged from his book *FPGA Prototyping by SystemVerilog Examples*; it was modified to include all 8 of the display digits on the board.

5 Code Appendix

Listing 1: Verilog code for implementing the scrolling display.

```
`timescale 1ns / 1ps

module scrolling_display(
    input logic [15:0] sw,
    input logic clk, reset, enter_btn, prog,
    output logic [7:0] an, sseg,
    output logic [15:0] led);

    logic [3:0] addra;
    logic [5:0] addrb;
    logic ena, enb, write_en, tick, enter;
    logic [15:0] data_in;
    logic [3:0] data_out;
    logic [31:0] disp_val;
    logic [63:0] hex;

    // Initialize state definitions
    typedef enum {DEF_DISP, DISPLAY, PROG} scrolling_states;
    scrolling_states state, next_state;

    always_ff @(posedge clk, posedge reset)
        if (reset)
            state <= DEF_DISP;
        else
            state <= next_state;

    // State logic
    always_comb
    begin
        case (state)
        DEF_DISP:
        begin
            ena = 1;
            enb = 1;
            write_en = 0;
            led = 16'b0;
            if(prog)
                next_state = PROG;
            else
                next_state = DEF_DISP;
        end

        PROG:
        begin
```

```
            data_in = sw;
            led = 16'hFFFF;

            if(enter)
            begin
                write_en = 1'b1;
                next_state = DISPLAY;
            end
            else
                next_state = PROG;
        end

        DISPLAY:
        begin
            write_en = 0;
            led = 16'b0;

            if (reset)
                next_state = DEF_DISP;
            else if(prog)
                next_state = PROG;
            else
                next_state = DISPLAY;
        end
        endcase
    end

    // BRAM instantiation
    bram_wrapper bram1(
        addra,
        addrb,
        clk,
        clk,
        data_in,
        data_out,
        ena,
        enb,
        write_en);

    // Button debouncer for enter button
    debounce debounce_sd(
        .clk(clk),
        .reset(reset),
        .sw(enter_btn),
        .db_level(enter),
        .db_tick());
```

```
// Create tick every second for shift reg
mod_m_counter#(.M(100000000)) mem_out_delay(
    .clk(clk),
    .reset(reset),
    .max_tick(tick),
    .q());

// Shift BRAM output for scrolling display
univ_shift_reg#(.N(32)) shift(
    .clk(tick),
    .reset(reset),
    .ctrl(2'b10),
    .d(data_out),
    .q(disp_val));

// Counter to increment address out of BRAM
counter#(.SIZE(6)) mem_out_count(
    .clk(tick),
    .reset(reset),
    .count(addrb));

// Counter to increment input address after every input
counter#(.SIZE(4)) addr_in(
    .clk(enter),
    .reset(reset),
    .count(addra));

// Hex to sseg conversion for 8 displays
hex_to_sseg display_val0(
    .hex(disp_val[3:0]),
    .dp(1),
    .sseg(hex[7:0]));

hex_to_sseg display_val1(
    .hex(disp_val[7:4]),
    .dp(1),
    .sseg(hex[15:8]));

hex_to_sseg display_val2(
    .hex(disp_val[11:8]),
    .dp(1),
    .sseg(hex[23:16]));

hex_to_sseg display_val3(
    .hex(disp_val[15:12]),
```

```
        .dp(1),
        .sseg(hex[31:24]));

    hex_to_sseg display_val4(
        .hex(disp_val[19:16]),
        .dp(1),
        .sseg(hex[39:32]));

    hex_to_sseg display_val5(
        .hex(disp_val[23:20]),
        .dp(1),
        .sseg(hex[47:40]));

    hex_to_sseg display_val6(
        .hex(disp_val[27:24]),
        .dp(1),
        .sseg(hex[55:48]));

    hex_to_sseg display_val7(
        .hex(disp_val[31:28]),
        .dp(1),
        .sseg(hex[63:56]));

    // Display mux
    disp_mux display(
        .clk(clk),
        .reset(reset),
        .in0(hex[7:0]),
        .in1(hex[15:8]),
        .in2(hex[23:16]),
        .in3(hex[31:24]),
        .in4(hex[39:32]),
        .in5(hex[47:40]),
        .in6(hex[55:48]),
        .in7(hex[63:56]),
        .an(an),
        .sseg(sseg));

endmodule
```