

Class Report 2: Block Memory

Cameron Anderson

May 13, 2019

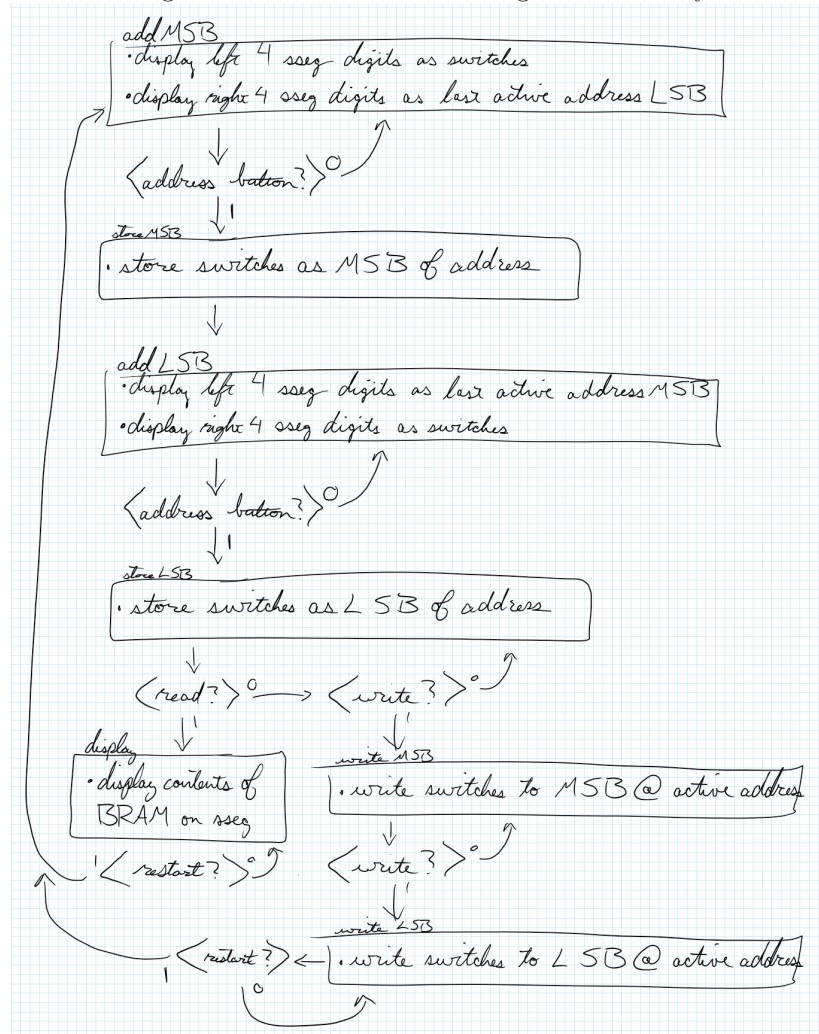
1 Introduction

The goal of this project was to use the seven-segment display and binary switches to read and write to and from the block memory within the Nexys 4 DDR prototyping board. For this project, the data and address registers are each 32 bits. The seven-segment display has 8 digits and can display the full range of hexadecimal digits at each of the 8 locations. With only 16 switches, the registers will need to be divided into MSB and LSB registers to read in the switches.

2 Experimental Plan

Figure 1 on page 2 shows the top level state machine that was used to transition between collecting the address and reading and writing to and from Block RAM. Because the registers were each 32 bits and there were only 16 switches, collecting the values through the switches had to be broken up into MSB and LSB states. This did not, however, affect the read from the BRAM because the seven segment display was able to read 8 hexadecimal digits equalling a total of 32 binary bits. The state machine transitions were completed by four buttons on the Nexys 4 DDR board. The buttons were arranged nicely in the order of the state machine. The top button was used to set first the MSB of the address and then the LSB of the address. On the next row down, the outside buttons were used to transition to the next state. The left button on the middle row was used to select the read function to display the BRAM's contents at the previously selected address. The right button on the middle row was used to select the write function. This button was used twice in succession after changing the switches to match the MSB 2 bytes of what was to be written to the previously selected address first and then similarly the LSB 2 bytes of the data. In addition to the state machine, Chu provided a module for displaying hexadecimal digits to the seven segment display but only the rightmost 4 digits. This was altered for this project to include all 8 digits of the seven segment display to read all 32 bits from an address in BRAM.

Figure 1: State Machine for using Block Memory



3 Analysis

4 Conclusion

5 Appendix

Listing 1: System Verilog code for the BRAM state machine top module

```
'timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

module rd_wr_state_machine
(
    input logic clk ,
    input logic reset ,
    input logic [15:0] sw ,
    input logic [3:0] btn ,
    output logic [7:0] an ,
    output logic [7:0] sseg
);

typedef enum {addMSB, storeMSB, addLSB, storeLSB, display, writeMSB, writeLSB} s

// declarations
state_type state_reg , state_next;

logic address; // top button [0] is the address button
logic read; // left button [3] is the read button
logic write; // right button [1] is the write button
logic restart; // bottom button [2] is the restart button
logic [31:0] display_seven; // 8 hex digits for seven seg display
logic [31:0] address_bits;
logic BRAM_en;
logic [3:0] BRAM_wr_en;
logic [31:0] BRAM_data_in;
logic [31:0] BRAM_data_out;

BRAM_wrapper BRAM_wrapper
(
    .addr(address_bits),
    .clk(clk),
    .reset(reset),
```

```

        .din(BRAM_data_in),
        .dout(BRAM_data_out),
        .en(BRAM_en),
        .write_en(BRAM_wr_en)
    );

disp_hex_mux disp_hex_mux
(
    .clk(clk),
    .reset(reset),
    .hex7(display_seven[31:28]),
    .hex6(display_seven[27:24]),
    .hex5(display_seven[23:20]),
    .hex4(display_seven[19:16]),
    .hex3(display_seven[15:12]),
    .hex2(display_seven[11:8]),
    .hex1(display_seven[7:4]),
    .hex0(display_seven[3:0]),
    .dp_in(8'b11111111),
    .an(an),
    .sseg(sseg)
);

always_ff @ (posedge clk, posedge reset)
    if (reset)
        begin
            state_reg <= addMSB; // start
        end
    else
        begin
            state_reg <= state_next;
        end
    end

always_comb
begin
    state_next = state_reg;
    case (state_reg)
        addMSB:
            begin
                display_seven[31:16] = sw[15:0]; // MSB 4 digits of seven segment display
                display_seven[15:0] = address_bits[15:0]; // LSB 4 digits of seven segment display
                if (address) // debounce
                    begin
                        address_bits[31:16] = sw[15:0]; // store switches as address MSB
                        state_next = storeMSB;
                    end
            end
    endcase
end

```

```

    end
end
storeMSB:
begin
    // (optional) blink the seven segment display to show change
    state_next = addLSB;
end
addLSB:
begin
    display_seven[31:16] = address_bits[31:16]; // MSB 4 digits of seven
    display_seven[15:0] = sw[15:0]; // LSB 4 digits of seven segment dis
    if (address) // debounce
    begin
        address_bits[15:0] = sw[15:0]; // store switches as address LSB
        state_next = storeLSB;
    end
end
storeLSB:
begin
    // (optional) blink the seven segment display to show change
    if (read) // debounce
    begin
        state_next = display;
    end
    else if (write) // debounce
    begin
        state_next = writeMSB;
    end
end
display:
begin
    BRAM_en = 1'b0; // enable (active low)
    display_seven = BRAM_data_out; // display contents of address on 7 s
    if (restart) // debounce
    begin
        BRAM_en = 1'b1; // disable (active low)
        display_seven = address_bits;
        state_next = addMSB;
    end
end
writeMSB:
begin
    BRAM_en = 1'b0; // enable (active low)
    BRAM_data_in[31:16] = sw[15:0]; // write switches to MSB of select
    // (optional) blink the seven segment display to show change

```

```

        if (write) // debounce
        begin
            state_next = writeLSB;
        end
    end
    writeLSB:
    begin
        BRAM_data_in [31:16] = sw [15:0]; // write switches to LSB of select
        // (optional) blink the seven segment display to show change
        if (restart) // debounce
        begin
            BRAM_en = 1'b1; // disable (active low)
            state_next = addMSB;
        end
    end
endcase
end
endmodule

```

Listing 2: Verilog code for the BRAM wrapper

```

// Copyright 1986–2018 Xilinx, Inc. All Rights Reserved.
//
// Tool Version: Vivado v.2018.2.2 (win64) Build 2348494 Mon Oct
1 18:25:44 MDT 2018
// Date          : Mon May 13 16:10:40 2019
// Host          : Lenny running 64-bit major release (build 9200)
// Command       : generate_target BRAM_wrapper.bd
// Design        : BRAM_wrapper
// Purpose       : IP block netlist
//
`timescale 1 ps / 1 ps

module BRAM_wrapper
    (BRAM_PORTA_0_addr,
     BRAM_PORTA_0_clk,
     BRAM_PORTA_0_din,
     BRAM_PORTA_0_dout,
     BRAM_PORTA_0_en,
     BRAM_PORTA_0_rst,
     BRAM_PORTA_0_we);
    input [31:0] BRAM_PORTA_0_addr;
    input BRAM_PORTA_0_clk;
    input [31:0] BRAM_PORTA_0_din;
    output [31:0] BRAM_PORTA_0_dout;

```

```

input BRAM_PORTA_0_en;
input BRAM_PORTA_0_rst;
input [3:0]BRAM_PORTA_0_we;

wire [31:0]BRAM_PORTA_0_addr;
wire BRAM_PORTA_0_clk;
wire [31:0]BRAM_PORTA_0_din;
wire [31:0]BRAM_PORTA_0_dout;
wire BRAM_PORTA_0_en;
wire BRAM_PORTA_0_rst;
wire [3:0]BRAM_PORTA_0_we;

BRAM BRAM_i
(
    .BRAM_PORTA_0_addr(BRAM_PORTA_0_addr),
    .BRAM_PORTA_0_clk(BRAM_PORTA_0_clk),
    .BRAM_PORTA_0_din(BRAM_PORTA_0_din),
    .BRAM_PORTA_0_dout(BRAM_PORTA_0_dout),
    .BRAM_PORTA_0_en(BRAM_PORTA_0_en),
    .BRAM_PORTA_0_rst(BRAM_PORTA_0_rst),
    .BRAM_PORTA_0_we(BRAM_PORTA_0_we));
endmodule

```

Listing 3: System Verilog code for the hexadecimal display mux

```

// Listing 4.17
module disp_hex_mux
(
    input logic clk, reset,
    input logic [3:0] hex7, hex6, hex5, hex4, hex3, hex2, hex1, hex0,
    // hex digits
    input logic [7:0] dp_in, // 4 decimal points
    output logic [7:0] an, // enable
    output logic [7:0] sseg // led segments
);

    // constant declaration
    // refreshing rate around 800 Hz (50 MHz/2^16)
    localparam N = 17;
    // internal signal declaration
    logic [N-1:0] q_reg;
    logic [N-1:0] q_next;
    logic [3:0] hex_in;
    logic dp;

    // N-bit counter
    // register

```

```

always_ff @(posedge clk, posedge reset)
    if (reset)
        q-reg <= 0;
    else
        q-reg <= q-next;

// next-state logic
assign q-next = q-reg + 1;

// 2 MSBs of counter to control 4-to-1 multiplexing
// and to generate active-low enable signal
always_comb
    case (q-reg[N-1:N-2])
        3'b000:
            begin
                an = 8'b11111110;
                hex_in = hex0;
                dp = dp_in[0];
            end
        3'b001:
            begin
                an = 8'b11111101;
                hex_in = hex1;
                dp = dp_in[1];
            end
        3'b010:
            begin
                an = 8'b11111011;
                hex_in = hex2;
                dp = dp_in[2];
            end
        3'b011:
            begin
                an = 8'b11110111;
                hex_in = hex3;
                dp = dp_in[3];
            end
        3'b100:
            begin
                an = 8'b11101111;
                hex_in = hex4;
                dp = dp_in[4];
            end
        3'b101:
            begin
                an = 8'b11011111;

```



```

        hex_in = hex5;
        dp = dp_in[5];
    end
    3'b110:
    begin
        an = 8'b10111111;
        hex_in = hex6;
        dp = dp_in[6];
    end
    default:
    begin
        an = 8'b01111111;
        hex_in = hex7;
        dp = dp_in[7];
    end
endcase

// hex to seven-segment led display
always_comb
begin
    case(hex_in)
        4'h0: sseg[6:0] = 7'b1000000;
        4'h1: sseg[6:0] = 7'b1111001;
        4'h2: sseg[6:0] = 7'b0100100;
        4'h3: sseg[6:0] = 7'b0110000;
        4'h4: sseg[6:0] = 7'b0011001;
        4'h5: sseg[6:0] = 7'b0010010;
        4'h6: sseg[6:0] = 7'b0000010;
        4'h7: sseg[6:0] = 7'b1111000;
        4'h8: sseg[6:0] = 7'b0000000;
        4'h9: sseg[6:0] = 7'b0010000;
        4'ha: sseg[6:0] = 7'b0001000;
        4'hb: sseg[6:0] = 7'b0000011;
        4'hc: sseg[6:0] = 7'b1000110;
        4'hd: sseg[6:0] = 7'b0100001;
        4'he: sseg[6:0] = 7'b0000110;
        default: sseg[6:0] = 7'b0001110; //4'hf
    endcase
    sseg[7] = dp;
end
endmodule

```