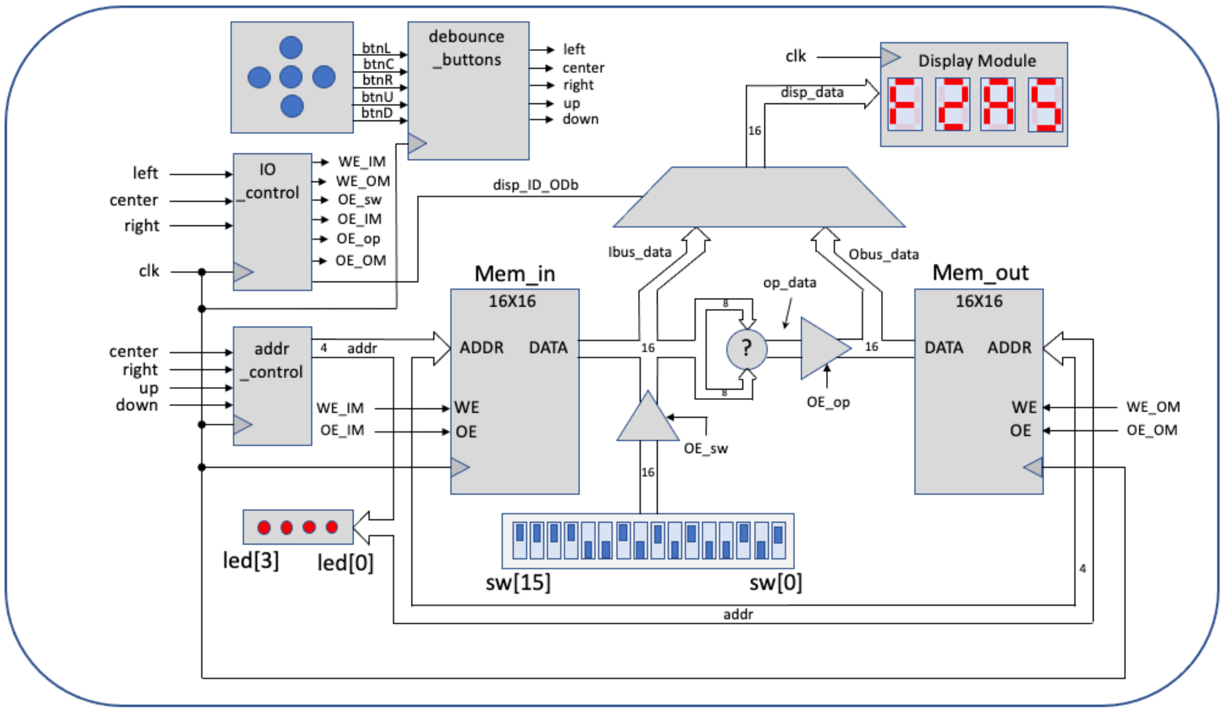
ECEN340
Verilog Code for Homework 8



Block Diagram for Homework 8

```
module memory_top(
    input clk,            //100MHz clk.
    input [15:0] sw,      //Data for input memory.
    input btnC,           //Used with other buttons to select an operation
    input btnU,           //Increment the memory address by one.
    input btnD,           //Decrement the memory address by one.
    input btnL,           //Select the input memory bank.
    input btnR,           //Select the output memory bank.
    output [3:0] an,      //The anode drivers for the 7-seg display.
    output [6:0] seg,     //The 7-segments. It is used to display the data in memory.
    output dp,            //The decimal place on the 7-seg display.
    output [15:0] led     //The 16 leds are used to display the address information and the mode
information.
    );

    wire [15:0] Ibus_data;          // Data for the input memory bank.
    wire [15:0] Obus_data;          // Data for the output memory bank.
    wire [15:0] op_data;            // The results of a math operation on the two halves of the input
memory bank.
    wire disp_ID_ODb;               // Ibus_data will be displayed when this signal is true,
Obus_data when it is false.
    reg  [15:0] disp_data;          // data to be displayed (Ibus or Obus).
    wire left, center, right, up, down;  // debounced buttons
    wire [3:0] addr;                      // address for the input and output memories

    sseg_x4_top disp (.clk(clk), .btnC(1'b0), .sw(disp_data), .seg(seg), .an(an), .dp(dp));
    memory IM (.we(WE_IM), .oe(OE_IM), .clk(clk), .data(Ibus_data), .addr(addr));
    memory OM (.we(WE_OM), .oe(OE_OM), .clk(clk), .data(Obus_data), .addr(addr));
    debounce_buttons bdb (.clk(clk), .btnL(btnL), .btnC(btnC), .btnR(btnR), .btnU(btnU),
              .btnD(btnD), .left(left), .center(center), .right(right), .up(up), .down(down));
    IO_control IOcont (.clk(clk), .left(left), .center(center), .right(right),
              .disp_ID_ODb(disp_ID_ODb), .WE_IM(WE_IM), .WE_OM(WE_OM),
              .OE_sw(OE_sw), .OE_IM(OE_IM), .OE_op(OE_op), .OE_OM(OE_OM));
    addr_control addrcont(.clk(clk), .center(center), .right(right), .up(up), .down(down),
              .addr(addr));

    assign Ibus_data = OE_sw?sw:16'hZZZZ;
    assign Obus_data = OE_op?op_data:16'hZZZZ;
    assign op_data = {8'h00, Ibus_data[15:8]} + {8'h00, Ibus_data[7:0]};

    assign led[7:0] = {4'b0000, addr}; // show the address on the four, least significant LEDs.
     // debug using the following LEDs
    assign led[15:8] = {1'b0, disp_ID_ODb, WE_IM, WE_OM, OE_IM, OE_OM, OE_op,
            OE_sw};


    always @ (*)                //Implements a multiplexer to select the data to be displayed
    begin
      if (disp_ID_ODb)          //disp_ID_ODb state is stored in a register
         disp_data = Ibus_data;
      else
         disp_data = Obus_data;
      end

endmodule
module debounce_buttons (
     input clk,
     input btnL, btnC, btnR, btnU, btnD,
     output left, center, right, up, down
     );
     wire clk_deb;

    debounce_div div1  (.clk(clk), .clk_deb(clk_deb));
    btn_debounce u_btn (.clk(clk_deb), .btn_in(btnU), .btn_status(up));
    btn_debounce d_btn (.clk(clk_deb), .btn_in(btnD), .btn_status(down));
    btn_debounce l_btn (.clk(clk_deb), .btn_in(btnL), .btn_status(left));
    btn_debounce r_btn (.clk(clk_deb), .btn_in(btnR), .btn_status(right));
    btn_debounce c_btn (.clk(clk_deb), .btn_in(btnC), .btn_status(center));
```

```
endmodule

module IO_control (
    input clk, left, center, right,
    output reg disp_ID_ODb, WE_IM, WE_OM, OE_sw, OE_IM, OE_op, OE_OM);

    always @ (*)  //Input and output memory mode configuration
    case ({left, center, right})
    3'b000: {WE_IM, WE_OM, OE_sw, OE_IM, OE_op, OE_OM} = 6'b00_01_01;
    3'b001: {WE_IM, WE_OM, OE_sw, OE_IM, OE_op, OE_OM} = 6'b00_01_10;
    3'b010: {WE_IM, WE_OM, OE_sw, OE_IM, OE_op, OE_OM} = 6'b00_01_01;
    3'b011: {WE_IM, WE_OM, OE_sw, OE_IM, OE_op, OE_OM} = 6'b01_01_10;
    3'b100: {WE_IM, WE_OM, OE_sw, OE_IM, OE_op, OE_OM} = 6'b00_10_01;
    3'b101: {WE_IM, WE_OM, OE_sw, OE_IM, OE_op, OE_OM} = 6'b01_10_10;
    3'b110: {WE_IM, WE_OM, OE_sw, OE_IM, OE_op, OE_OM} = 6'b10_10_01;
    3'b111: {WE_IM, WE_OM, OE_sw, OE_IM, OE_op, OE_OM} = 6'b00_10_10;
    endcase

    always @ (posedge clk, posedge right, posedge left) //Saving the input/output memory bank
button presses
      if (right)
         disp_ID_ODb <= 1'b0;
      else if (left)
         disp_ID_ODb <= 1'b1;
      else
         disp_ID_ODb <= disp_ID_ODb;
endmodule

module addr_control (
    input clk, center, right, up, down,
    output reg [3:0] addr);

    wire auto_inc_addr, inc_addr_pulse, dec_addr_pulse, rst_addr;

    create_pulse_from_step inc (.clk(clk), .step(up), .pulse(inc_addr_pulse));
    create_pulse_from_step dec (.clk(clk), .step(down), .pulse(dec_addr_pulse));
    assign auto_inc_addr = center && right;
    assign rst_addr = up && down;

    initial addr = 4'h0;   //Address generator implementation
    always @ (posedge clk)
    begin
       if (rst_addr) addr <= 4'h0;
       else if (dec_addr_pulse && (addr > 4'h0)) addr <= addr - 1;
       else if ((inc_addr_pulse || auto_inc_addr) && (addr < 4'hF)) addr <= addr + 1;
       else addr <= addr;
    end
endmodule

    // the following module transforms a single step into a pulse lasting exactly one clock cycle
module create_pulse_from_step (
     input clk,
     input step,
     output pulse //pulse lasting exactly one clock cycle for each input step
     );
     reg step_tm1, step_tm2;

     initial step_tm1 = 1'b0;
     initial step_tm2 = 1'b0;
     always @ (posedge clk)
     begin
       step_tm1 <= step;
       step_tm2 <= step_tm1;
     end

     assign pulse = step_tm1 && !step_tm2;
endmodule

module btn_debounce(
    input clk,
    input btn_in,
    output wire btn_status
    );
    reg [19:0] btn_shift;

    always @ (posedge clk)
        btn_shift <= {btn_shift[18:0], btn_in};

    assign btn_status = &btn_shift;
endmodule

module debounce_div(
    input clk,
    output clk_deb
    );
    reg [15:0] cnt;

    assign clk_deb = cnt[15];

    initial cnt = 0;
    always @(posedge clk)
        cnt <= cnt + 1;
```

```
endmodule

module memory(
        input wire oe, we, clk,
        input wire [3:0] addr,
        inout wire [15:0] data
        );

        reg [15:0] mem[15:0];
        assign data = (oe && !we) ? mem[addr]:16'hZZZZ;

        always @(posedge clk)
        begin
           if (we) mem[addr] <= data;
        end
endmodule
```