**ECSE 4770 LAB 3 REPORT**

Yuchen Wang, Rin: 661960546, section 2

Partner: Yilu Zhou, Rin: 661931826, section 2

**Abstract**

* The purpose of this lab is helping student build a state machine that can be applied in the real world. The state machine should be able to respond to inputs and sequence outputs correctly.
* The requirements for this lab is to build the controller for a stopwatch timer. In this lab only the stopwatch controller need to be built. It will be used in Lab 4 to implement the whole timer. So the input and output pings should not be registered and the name needs to be consistent with those given in the lab instruction. File name should be stopwatch\_controller.v.

**Overall design description**

* All the inputs and outputs are wire. To apply always and condition function in the Verilog, there are temporary variable in the program that are registers. But they transformed to wire when output the data.
* The address latch and memory latch are connected to 1 kHz clock. Here Memory is always enable, only the Read/Write signal control when it should be read or write. For most case, the memory is read only, only when the counter is running and SA is pressed, the machine will enable memory write to record the time.
* The basic idea of this controller is set correct states and link them together with proper conditions. To make the controller works as expected, students designed 8 states for the state machine, 2 for START/STOP and COUNTER/SPLIT switch debouncing, the rest 6 are the running state of the machine.
* State for debouncing: SSD and CSD, They will hold the state and make the machine do nothing until the START/STOP and COUNTER/SPLIT are depressed. The controller will work until next positive edge of the input clock.
* Remaining 6 state: COUNT, WAIT, SHOW, ADDL, MEML, and SAVE. COUNT state is when the timer is running, it shows the timer on the LED. SAVE state is for saving the time to the memory. SHOW is the state to show the time saved in the memory, this will be change the MUX\_C output. WAIT is for the state that hold the state. ADDL is the state to use address latch. MEML is the state to use memory latch.

**Schematic and State Diagram**

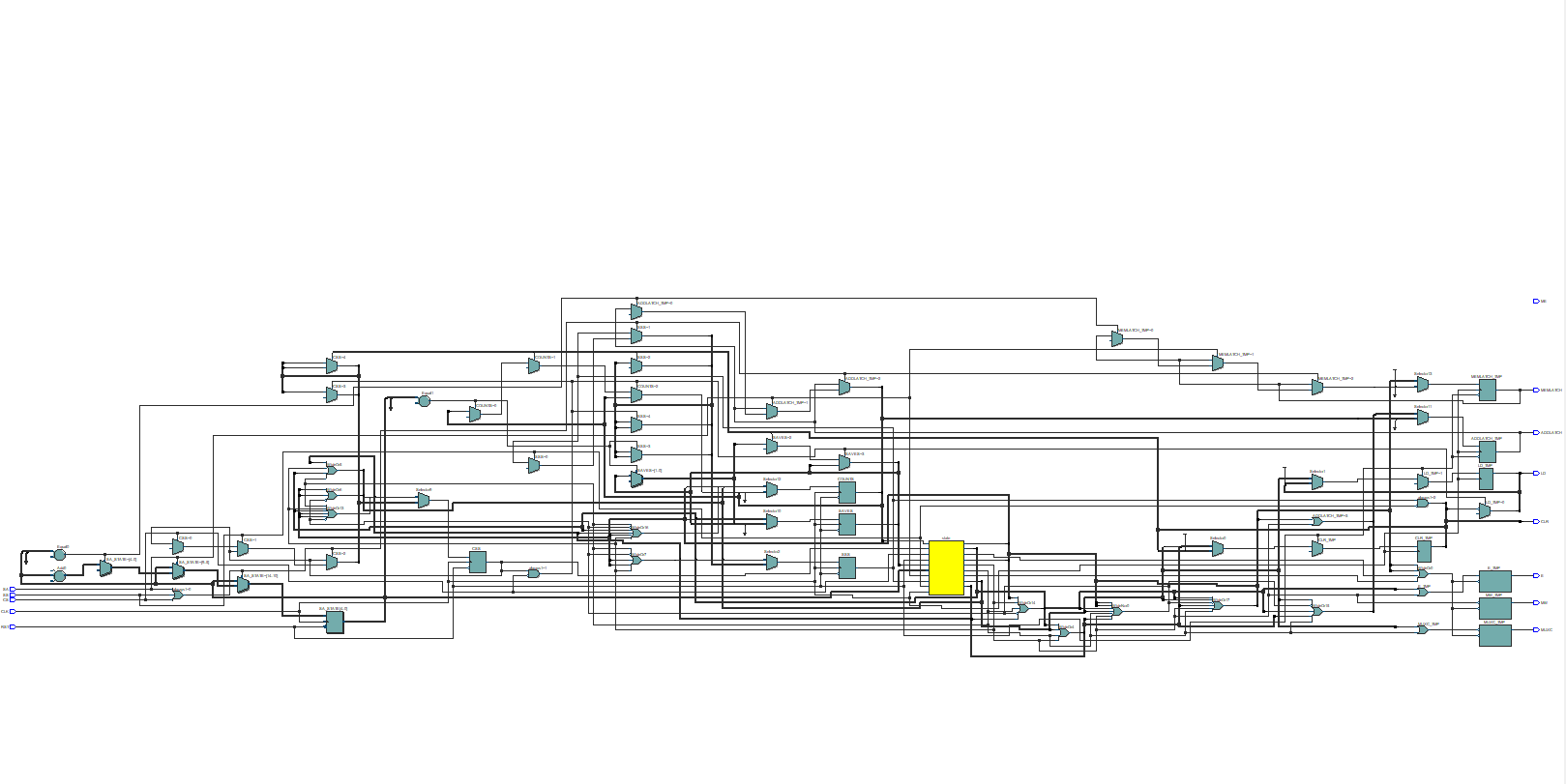


Figure: RTL view

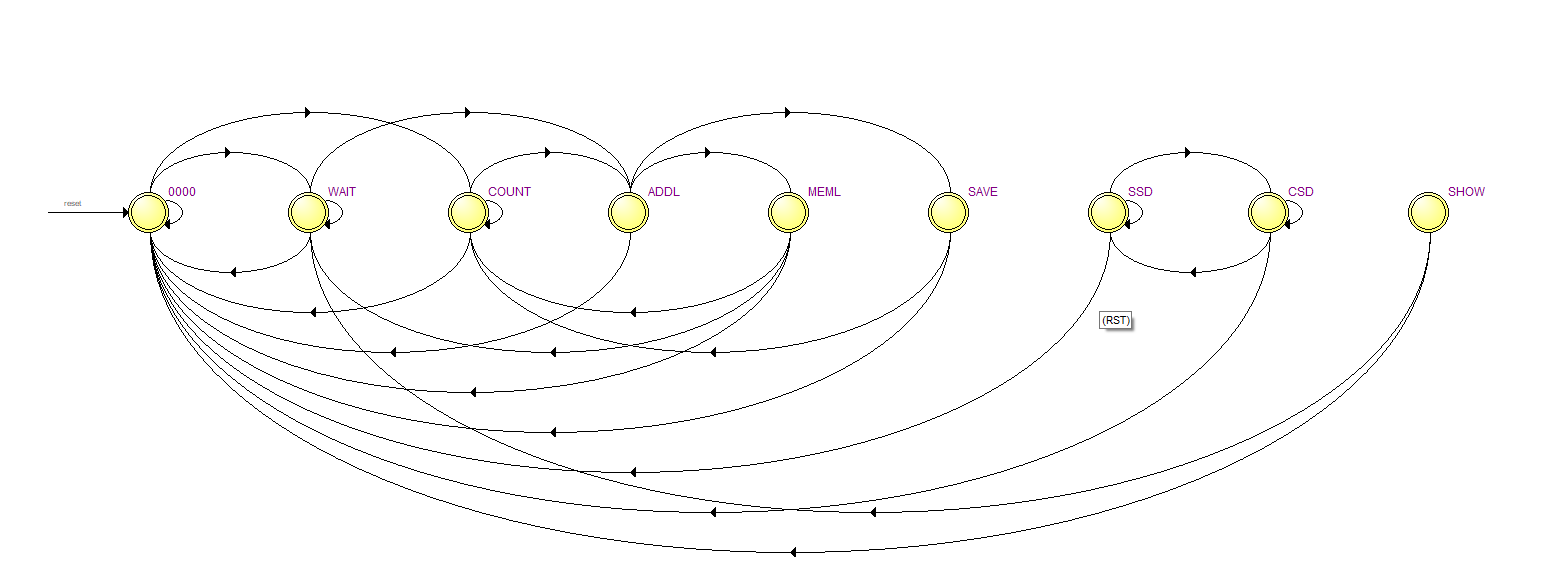


Figure: State Diagram

According to the requirements, 0000 is the state before reset pressed. Wait state wait until the beginning of running or request latches. When stop by the SS or need for reading by CS, the machine is also in the WAIT state.

Count state is the state for the timer running, it can be recovered from a memory reading(MEML), and wait state. In count state, the save state can be triggered by SA switch. When saving the data, ADDL, address latch need to be triggered to locate the place of memory.

SHOW state change the MUX\_C, show the data in the memory.

SSD and CSD are for debounce, they control the machine instead of SS and CS. SSD and CSD turns to low before the next rising clock edge when the SS and CS are released.

**Code**

// Lab3 Yilu Zhou, Yuchen Wang

module stopwatch\_controller(input SS,

                            input CS,

                            input RST,

                            input SA,

                            input CLK,

                            output ADDLATCH,

                            output ME,

                            output MW,

                            output MEMLATCH,

                            output LD,

                            output E,

                            output CLR,

                            output MUXC);

    // Define the states numbers.

    parameter STOP  = 0;

    parameter COUNT = 1;

    parameter WAIT  = 2;

    parameter ADDL  = 3;

    parameter MEML  = 4;

    parameter SHOW  = 5;

    parameter SAVE  = 6;

    parameter SSD   = 7;

    parameter CSD   = 8;

    // Define registers to record the stopwatch controller status.

    reg ADDLATCH\_TMP;

    reg ME\_TMP;

    reg MW\_TMP;

    reg MEMLATCH\_TMP;

    reg LD\_TMP;

    reg E\_TMP;

    reg CLR\_TMP;

    reg MUXC\_TMP;

    reg [3:0] state;

    reg [4:0] SA\_STATE;

    reg COUNTS, SAVES;

    reg SSS, CSS;

    // Define the power-on reset initialization.

    initial begin

        ADDLATCH\_TMP = 0; // Keep latch at logic low.

        MEMLATCH\_TMP = 0;

        ME\_TMP       = 1; // Set to enable RAM to reset data.

        MW\_TMP       = 1; // Set to write mode to save 0.

        LD\_TMP       = 1; // Active low, so set high to inactive.

        E\_TMP        = 0; // Stop the counter

        CLR\_TMP      = 0; // Clear the counter.

        MUXC\_TMP     = 1; // Select the cleared counter output.

        SSS          = 0; // Reg to save SS press.

        CSS          = 0; // Reg to save CS press.

        SA\_STATE     = 0; // Reg to save number of ms of SA.

        COUNTS       = 0;

        state        = 0; // Reg to save the current state.

    end

    // Define different states.

    always @(state) begin

        case (state)

            SSD: begin

                // SSD is the state to wait until SS is released.

                $display("DEB: SSD");

            end

            CSD: begin

                // CSD is the state to wait until CS is released.

                $display("DEB: CSD");

            end

            STOP: begin

                // STOP is the state to stop counting.

                $display("STOP");

                ME\_TMP   <= 0; // Enable memory

                MW\_TMP   <= 1; // Diable memory write

                E\_TMP    <= 0; // Stop the counter

                MUXC\_TMP <= 1; // Show the counter value

            end

            COUNT: begin

                // COUNT is the state to counting.

                $display("COUNT");

                ME\_TMP         <= 0; // Enable memory

                MW\_TMP         <= 1; // Memory read-only

                E\_TMP          <= 1; // Start the counter

                MUXC\_TMP       <= 1; // show the counter value

                //MEMLATCH\_TMP <= 0;

            end

            WAIT: begin

                $display("WAIT");

                // WAIT is the state when counters are stopped and CS has been

                // pressed and released.

                ME\_TMP         <= 0; // Enable memory

                MW\_TMP         <= 1; // Memory read-only

                E\_TMP          <= 0; // Stop the counter

                MUXC\_TMP       <= 0; // Show the ram output

                //MEMLATCH\_TMP <= 0;

            end

            ADDL: begin

                $display("ADDL");

            end

            MEML: begin

                $display("MEML");

            end

            SHOW: begin

                // SHOW is the state when the timer is in WAIT state and SA is

                // valid.

                $display("SHOW");

                ME\_TMP   <= 0; // Enable memory

                MW\_TMP   <= 1; // Memory read-only

                E\_TMP    <= 0; // Stop the counter

                MUXC\_TMP <= 0; // Show the ram output

            end

            SAVE: begin

                // SAVE is the state when the timer is in COUNT state and SA is

                // valid.

                $display("SAVE");

                ME\_TMP   <= 0; // Enable memory

                MW\_TMP   <= 0; // Memory write

                E\_TMP    <= 1; // Counter enabled

                MUXC\_TMP <= 1; // Show the counter output

            end

        endcase

    end

    // To avoid the use of output reg, use temp reg and wire as output.

    assign ADDLATCH = ADDLATCH\_TMP;

    assign ME       = ME\_TMP;

    assign MW       = MW\_TMP;

    assign MEMLATCH = MEMLATCH\_TMP;

    assign LD       = LD\_TMP;

    assign E        = E\_TMP;

    assign CLR      = CLR\_TMP;

    assign MUXC     = MUXC\_TMP;

    always @(posedge CLK) begin

        if (RST) begin // RST is high, keep resetting

            CLR\_TMP <= 0; // Clear the counter

            LD\_TMP  <= 1; // Do not load counter

            state   <= STOP;

        end

        else begin if (SS || CS || SA) begin

        // One button is pressed or SA valid

        if (SA) begin

            SA\_STATE <= SA\_STATE + 1; // Record the number of ms of SA.

            if (SA\_STATE == 9) begin

                SA\_STATE     <= 0; // If reach 10 ms, reset register.

                ADDLATCH\_TMP <= 0;

                MEMLATCH\_TMP <= 0;

            end

        end

            if (SS) begin // SS is pressed.

                SSS   <= 1;

                state <= SSD;

            end

            else if (CS) begin // CS is pressed.

                CSS   <= 1;

                state <= CSD;

            end

                end

                // See state transition table for details

                case(state)

                    STOP: begin

                        if (SSS && !SS) begin

                            // SS has been pressed and released.

                            CLR\_TMP <= 0; // Clear the counter

                            LD\_TMP  <= 1; // Do not load counter

                            SSS     <= 0; // Change SSS back to 0

                            state   <= COUNT;

                        end

                        else if (CSS && !CS) begin

                            // CS has been pressed and released.

                            CLR\_TMP <= 1; // Do not clear the counter

                            LD\_TMP  <= 1; // Do not load counter

                            CSS     <= 0; // Change CSS back to 0

                            state   <= WAIT;

                        end

                        else begin

                            // Loop back to STOP.

                            CLR\_TMP <= 1; // Hold the counter value

                            LD\_TMP  <= 1; // Do not load counter

                            state   <= STOP;

                        end

                    end

                    COUNT: begin

                        if (SA\_STATE == 1) begin

                            // SA is valid, save the time.

                            CLR\_TMP <= 1; // Do not clear counter

                            LD\_TMP  <= 1; // Do not load counter

                            SAVES   <= 1;

                            state   <= ADDL;

                        end

                        else if (SSS && !SS) begin

                            // SS is presses and released, stop counter.

                            CLR\_TMP <= 1; // Do not clear counter

                            LD\_TMP  <= 1; // Do not load counter

                            SSS     <= 0; // Change SSS back to 0

                            state   <= STOP;

                        end

                        else begin

                            // Continue counting.

                            CLR\_TMP <= 1; // Do not clear counter

                            LD\_TMP  <= 1; // Do not load counter

                            state   <= COUNT;

                        end

                    end

                    SAVE: begin

                        // Save the time to the memory module.

                        CLR\_TMP      <= 1; // Do not clear counter

                        LD\_TMP       <= 1; // Do not load counter

                        ADDLATCH\_TMP <= 0;

                        state        <= COUNT;

                    end

                    WAIT: begin

                        if (CSS && !CS) begin

                            // CS has been presses and releases.

                            CLR\_TMP <= 1; // Do not clear counter

                            LD\_TMP  <= 1; // Do not load counter

                            CSS     <= 0; // Change CSS back to 0

                            state   <= STOP;

                        end

                        else if (SSS && !SS) begin

                            // CS has been pressed and released.

                            CLR\_TMP <= 1; // Do not clear counter

                            LD\_TMP  <= 0; // Load counter

                            SSS     <= 0; // Change SSS back to 0

                            COUNTS  <= 1;

                            SAVES   <= 0;

                            state   <= ADDL;

                        end

                        else if (SA\_STATE == 1) begin

                            // SA is valid, show the time.

                            CLR\_TMP <= 1; // Do not clear counter

                            LD\_TMP  <= 1; // Do not load counter

                            COUNTS  <= 0; // Do not contine counting

                            SAVES   <= 0;

                            state   <= ADDL;

                        end

                        else begin

                            // Continue waiting.

                            CLR\_TMP <= 1; // Do not clear counter

                            LD\_TMP  <= 1; // Do not load counter

                            state   <= WAIT;

                        end

                    end

                    ADDL: begin

                        //ADDLATCH\_TMP <= 1;

                        MEMLATCH\_TMP   <= 0;

                        if(COUNTS) begin

                            ADDLATCH\_TMP <= 0;

                            state <= MEML;

                        end

                        else if (SAVES) begin

                            ADDLATCH\_TMP <= 1;

                            state <= SAVE;

                        end

                        else begin

                            ADDLATCH\_TMP <= 1;

                            state <=MEML;

                        end

                        SAVES <= 0;

                    end

                    MEML: begin

                        ADDLATCH\_TMP <= 0;

                        MEMLATCH\_TMP <= 1;

                        if (COUNTS == 0) begin

                            state <= WAIT;

                        end

                        else begin

                            state <= COUNT;

                        end

                        COUNTS <= 0;

                    end

                    SHOW: begin

                        // Immediaely goes back to WAIT after showing the

                        // value.

                        CLR\_TMP <= 1; // Do not clear counter

                        LD\_TMP  <= 1; // Do not load counter

                        state   <= WAIT;

                    end

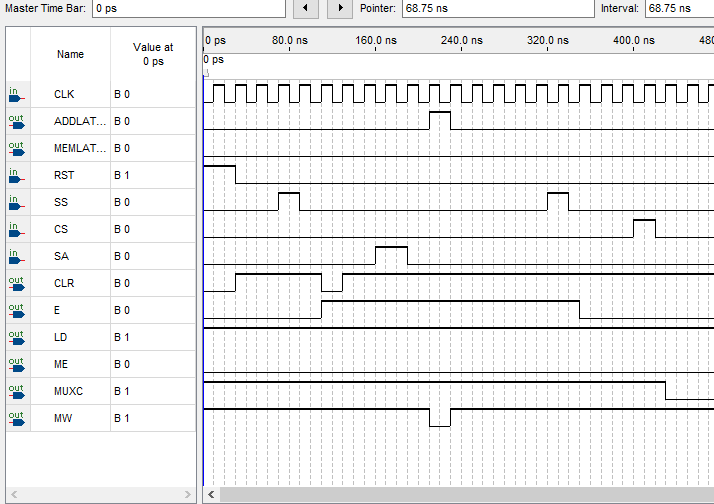
                endcase

                end

                end

                endmodule

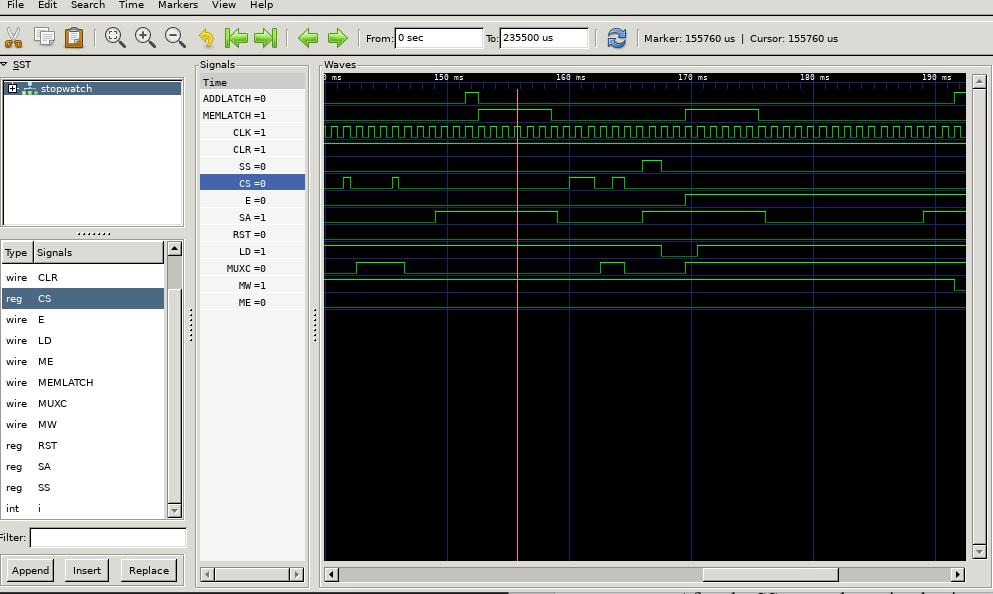
**Simulation**



Since the simulation time restriction in the Quartus, The frequency is changed here to make a simulation.

In this time diagram, After press RST turning the machine on, SS is pressed. Then the machine clear the timer to start a new count. Near the 160ns, SA is pressed to save a time to the memory. The Memory latch positive edge is triggered, and the memory change to write mode.

After the SS pressed agagin, the timer stopped, E turns to 0. Then the CS is pressed, the machine change to a show mode.



The second simulation case is from Verilog simulation. This is a simuation for loading the data from memory and operation. Near 150ms, the SA is pressed when the timer stopped and CS is pressed. The address latch first triggered, then the memory latch is triggered to read the data. After this action, CS is pressed twice to check if the loading function can work properly. Then SS pressed, the LD is triggered to load the data to the timer. And MUXC show the timer number.



The Time between 100ms and 130ms shows that when timer is running, the CS switch will not effect the running state. After the SS is pressed to stop the timer, CS can trigger the change between showing the memory data or the timer data on the LED by MUXC.

**Verification**

The verification of the data is by the Verilog simulation and Quartus simulation. The test case is written according to the description and requirements in the lab instruction.\

**Conclusion**

In this lab a state machine is built. To implement a state machine with serval states is difficult. It is significant to make every link between all the states. Additionally, the lab is hard to debug. It is hard for students to check if their design is correct.