ECSE 4770 LAB 4 REPORT

Nov. 2021

Yuchen Wang, Rin: 661960546, section 2

Partner: Yilu Zhou, Rin: 661931826, section 2

Abstract

- The purpose of this lab is to build the stopwatch with the controller part of lab3. In this lab, students will build the stopwatch with the controller made in the lab 3, on the FPGA board, DE10-Lite. To implement the function, students need to design and develop each part of the stopwatch, including SA logic, D flip-flops, memory, counter and MUX.
- The target is to achieve the stopwatch on the FPGA board.

Overall design description

- The stopwatch controller is designed and verified in the lab3.
- As described in the previous lab, the stop watch controller can control the memory, switch decoder, and counter in the stopwatch.
- The time will be saved in a 20-bit form of variable, it will be transformed into the form of time showed on the 5-bit decimal number, with minute, second, and hundredth second.
- SA logic is the switch decoder, which read the address switch on the FPGA board. Then it will decode the input and generate a 4 bit address signal to be hold in a D flip-flop. It needs to read the clock signal to make an output which has a 10ms duration. The SA

logic is debounced on the split time switch. When a split time is pressed, it will provide the controller with a high signal to inform that the switch is pressed.

- The memory part will read and write the time. The output data of the MUX will go into the block. The output of the block goes to load port of the counter and the MUX. To control this block, memory enable and read/write enable of the stopwatch controller are connected to the block. Both of the wire will determine whether the memory will save or output the data. The address of the data is provided by the SA logic, through a 4-bit D flip-flop. The output will be saved in a 20-bit D flip-flop. It is controlled by the memory latch from the stopwatch controller.
- The Counter is similar to the counter made in lab2, it can be enabled, cleared by the controller. It can also load the output time of the memory and started from that time. All the input are connected to the stop watch controller. The load port is connected to the D flip-flop hold the output of the memory.
- The MUX can determine which output should be displayed on the board, the counter or the memory. It is controlled by MUXC of the controller.
- A time decoder is implemented here to transform the 20-bit time data to the number can be displayed on the board.

Schematic and State Diagram

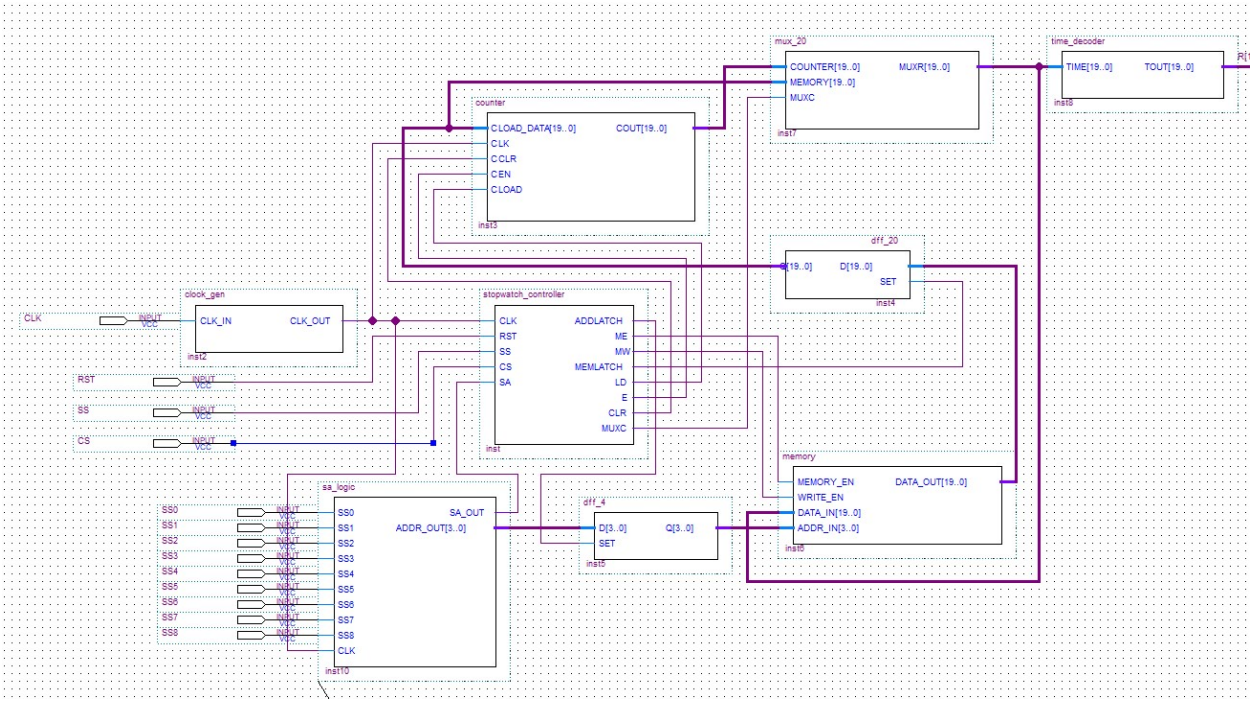


Figure: Schematic 1 main part

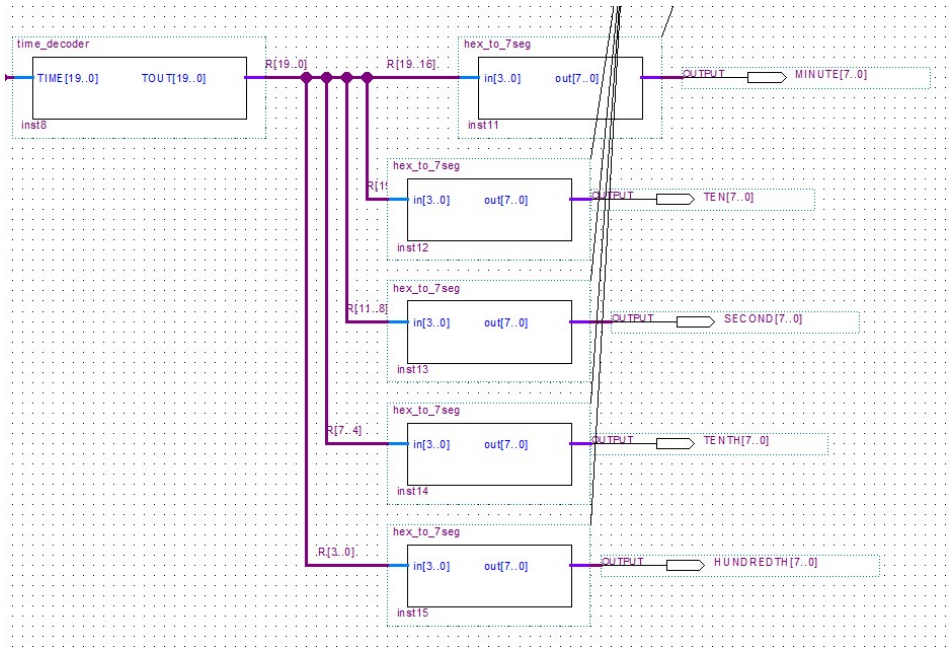


Figure: Schematic 2 output display part

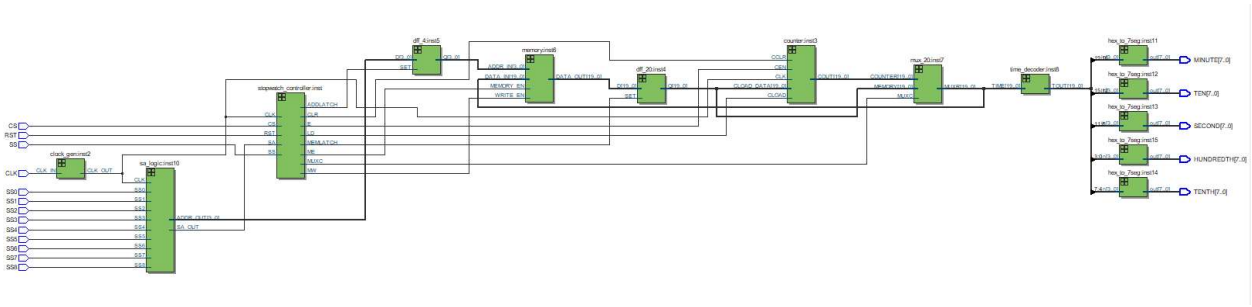


Figure: RTL view

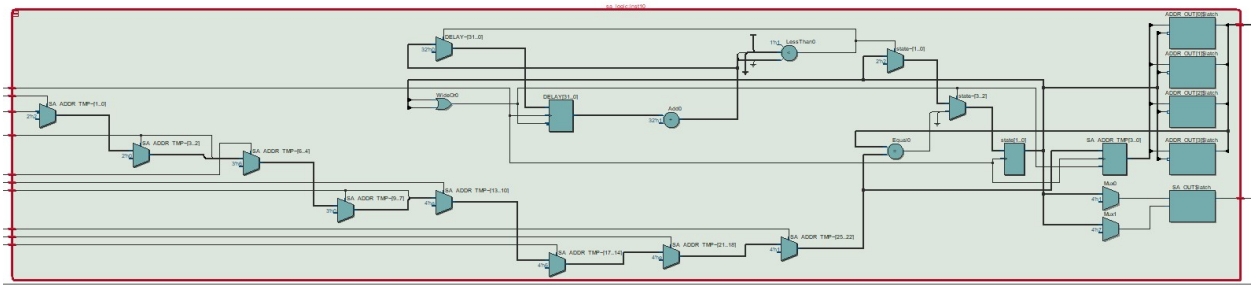


Figure: RTL view of SA logic

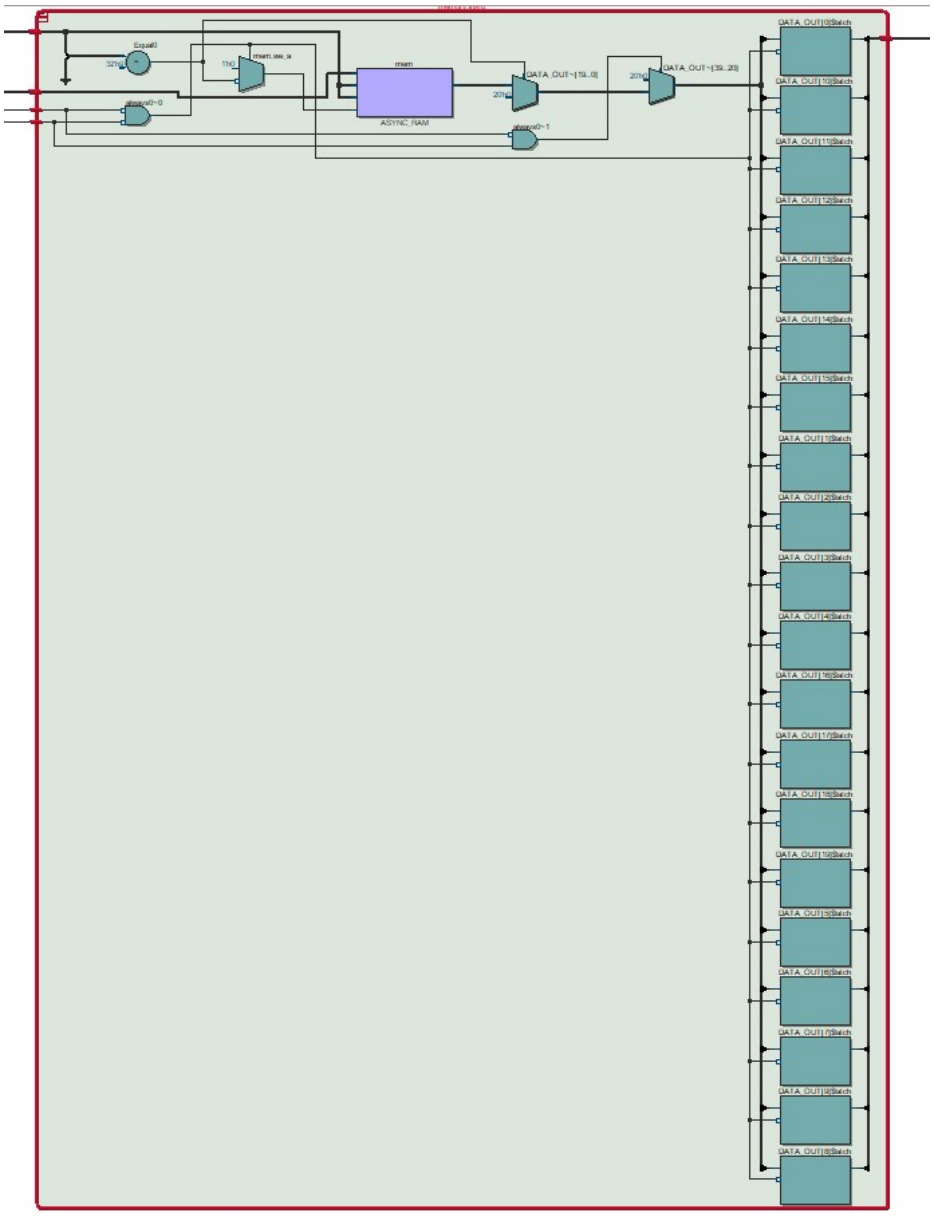


Figure: RTL view of memory

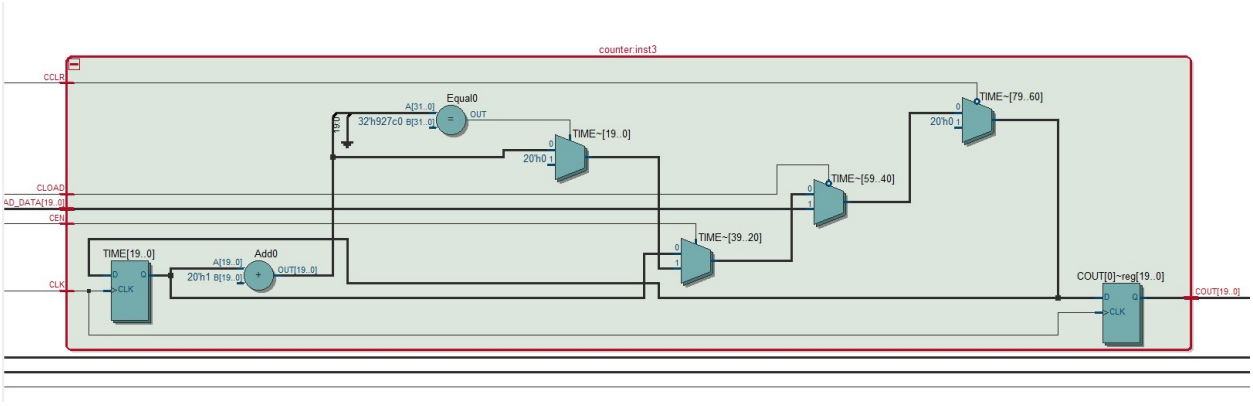


Figure: RTL view of counter

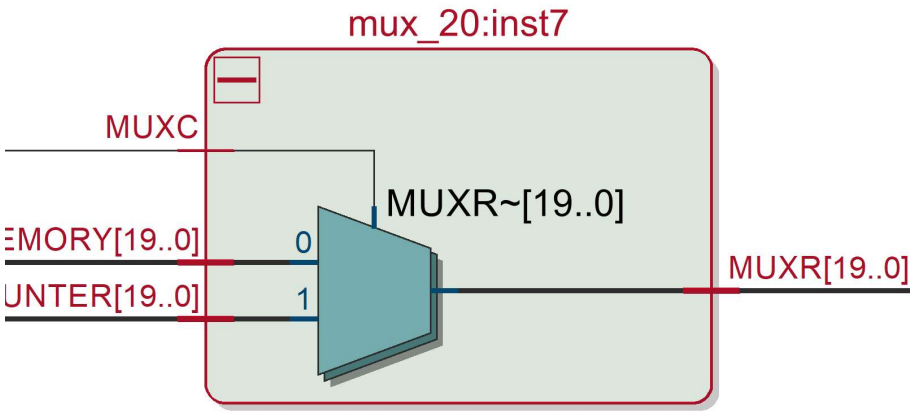


Figure: RTL view of multiplexer

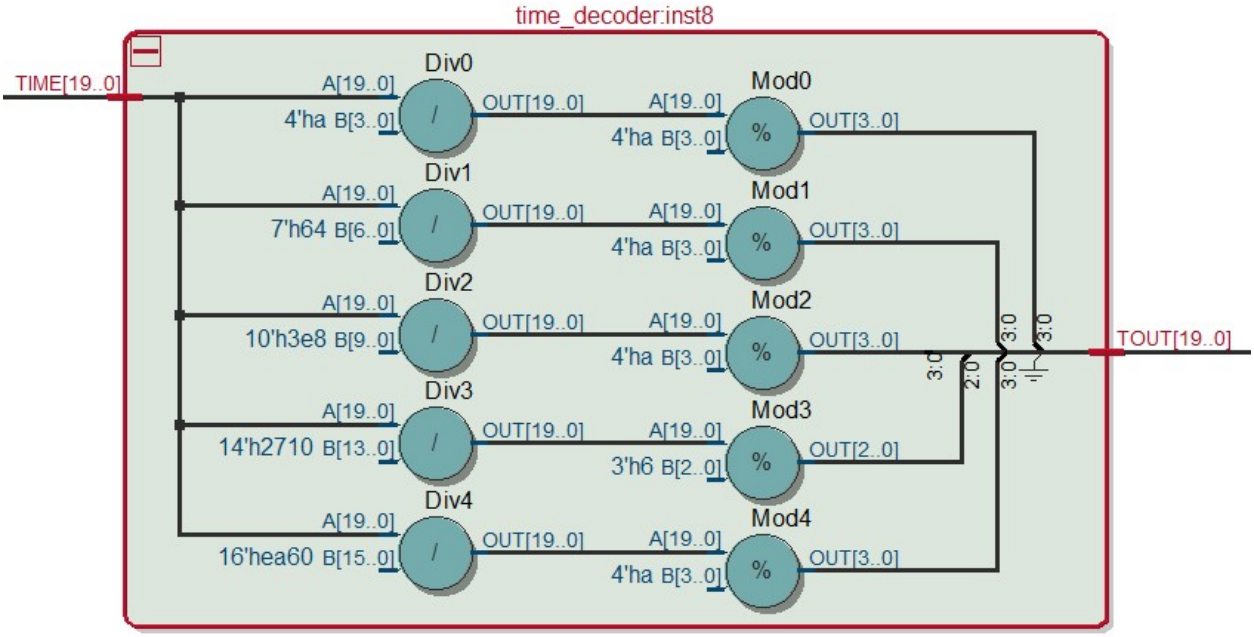


Figure: RTL view of time decoder

Code

Stopwatch Controller Description:

Inputs:

start timer, SS, and time split, CS, are the push button on the FPGA board. They are connected to the controller directly. CLK is connected to a clock frequency modifier, which can turn the clock provided by the board (10MHz) to 1000Hz. The output frequency can be used by the counter and other parts of the board. RST input is to reset the stopwatch. It is one of the push button on the board. SA, a signal provided by the SA logic part, it will be active when the remaining push button on the board is pressed. The SA logic will inform the controller that it needs to activate the Address Latch. The CLK of the counter is connected to the clock modifier.

Outputs:

The Memory Latch and Address Latch can control the 2 D flip-flop used in the stopwatch. They can hold the switch address and memory output number. Load(LD), Counter Enable(E), and Clear(CLR) are connected to the counter. They can control the counter to load, run or reset. Memory Enable and Read/Write Enable are connected to the memory block. They determine whether the counter is reading or writing and which output the memory will pass to the D flip-flop.

Display and drivers in base-10 format:

```
module time_decoder(input [19:0] TIME, output [19:0] TOUT);
    wire [3:0] HUNDREDTH_SECOND;
    wire [3:0] TENTH_SECOND;
    wire [3:0] SECOND;
    wire [3:0] TEN_SECOND;
    wire [3:0] MINUTE;

    assign HUNDREDTH_SECOND = TIME / 10 % 10;
    assign TENTH_SECOND     = TIME / 100 % 10;
    assign SECOND           = TIME / 1000 % 10;
    assign TEN_SECOND       = TIME / 10000 % 6;
    assign MINUTE           = TIME / 60000 % 10;
    assign TOUT             = {MINUTE, TEN_SECOND, SECOND, TENTH_SECOND,
HUNDREDTH_SECOND};
    // Combine the five wires together for hex displays.
endmodule
```

This is the block which can transform the 20-bit time to the ouput 20 bits with hundredth second, tenth second, second, ten second and minute.

Multiplexer:

```
module mux_20(input [19:0] COUNTER,
              input [19:0] MEMORY,
              input MUXC,
              output [19:0] MUXR);
    assign MUXR = MUXC ? COUNTER : MEMORY;
endmodule
```

The MUXC determine which output should be displayed on the board. Counter is the time from counter, memory is the output time from memory.

Split Memory:

```
module memory(input MEMORY_EN,
              input WRITE_EN,
              input [19:0] DATA_IN,
              input [3:0] ADDR_IN,
              output reg [19:0] DATA_OUT);

    reg [19:0] mem [8:0];
    initial begin
        DATA_OUT <= 20'b0000_0000_0000_0000_0000;
    end

    always @(*) begin
        if (!MEMORY_EN && !WRITE_EN) begin
            if (ADDR_IN != 9) begin
                mem[ADDR_IN] <= DATA_IN;
            end
        end
        else if (!MEMORY_EN && WRITE_EN) begin
            if (ADDR_IN == 9) begin
                DATA_OUT <= 20'b0000_0000_0000_0000_0000;
            end
            else begin
                DATA_OUT <= mem[ADDR_IN];
            end
        end
        else begin
            DATA_OUT <= 20'b0000_0000_0000_0000_0000;
        end
    end
endmodule
```

WRITE_EN is connected to Read/write EN to determine the mode of this block. If in reading mode (low), it will output the data saved in memory for the corresponding address from the SA logic. If the block is writing mode (high), the data of the given address will be the output of the block.

Here ADD_IN == 9 and the else case are preventing the block works improperly. Memory enable is set to low to make the block working.

Counters:

```
module counter(input [19:0] CLOAD_DATA,
               input CLK,
               input CCLR,
               input CEN,
               input CLOAD,
               output reg [19:0] COUT);

    reg [19:0] TIME;

    initial begin
        TIME = 20'b0000_0000_0000_0000_0000;
    end

    always @(posedge CLK) begin
        if (!CCLR) begin
            TIME = 20'b0000_0000_0000_0000_0000;
        end
        else if (!CLOAD) begin
            TIME = CLOAD_DATA;
        end
        else if (CEN) begin
            TIME = TIME + 1;
            if(TIME == 600000) begin
                TIME = 0;
            end
        end
        COUT = TIME;
    end
endmodule
```

Counter is set to 0 for initialization. The clock is positive edge triggered. CCLR is connected to the controller CLR, it will reset the counter when low. CLOAD is the load function. It is connected to the LDN of the controller. It is active low, which can load the data from the memory. CEN is the E of controller. It is active high, it can make the timer to run. The overflow is also cleared in this part.

Address switch logic:

```
module sa_logic(input SS0,
                input SS1,
                input SS2,
                input SS3,
                input SS4,
                input SS5,
                input SS6,
                input SS7,
                input SS8,
                input CLK,
                output reg SA_OUT,
                output reg [3:0] ADDR_OUT);

    parameter WAIT    = 0;
    parameter TRIGGER = 1;
    parameter HOLD    = 2;
    reg [1:0] state;
    reg [3:0] SA_ADDR_TMP;
    integer DELAY;

    initial begin
        SA_OUT      = 1'b0;
        ADDR_OUT    = 4'bxxxx; // An invalid address
        state       = TRIGGER;
        SA_ADDR_TMP = 4'b0000;
        DELAY       = 0;
    end

    always @(state) begin
        case (state)
            WAIT: begin     // Wait for 10ms SA high.
                SA_OUT   = 1;
                ADDR_OUT = SA_ADDR_TMP;
            end
            TRIGGER: begin  // Wait for SS change.
                SA_OUT   = 0;
                ADDR_OUT = 4'bxxxx;
            end
            HOLD: begin     // State with no SS change.
                SA_OUT   = 0;
                ADDR_OUT = 4'bxxxx;
            end
```

```
        endcase
    end

    always @(posedge CLK) begin
        if (state) begin
            if (SS8) begin
                SA_ADDR_TMP = 4'b1000;
            end
            else if (SS7) begin
                SA_ADDR_TMP = 4'b0111;
            end
            else if (SS6) begin
                SA_ADDR_TMP = 4'b0110;
            end
            else if (SS5) begin
                SA_ADDR_TMP = 4'b0101;
            end
            else if (SS4) begin
                SA_ADDR_TMP = 4'b0100;
            end
            else if (SS3) begin
                SA_ADDR_TMP = 4'b0011;
            end
            else if (SS2) begin
                SA_ADDR_TMP = 4'b0010;
            end
            else if (SS1) begin
                SA_ADDR_TMP = 4'b0001;
            end
            else if (SS0) begin
                SA_ADDR_TMP = 4'b0000;
            end
            else begin
                SA_ADDR_TMP = 4'b1001;
            end
            if (SA_ADDR_TMP == ADDR_OUT) begin
                state = HOLD;
            end
            else begin
                state = WAIT;
            end
        end
        else begin
            DELAY = DELAY + 1;
            if (DELAY >= 100) begin
```

```
                DELAY = 0;
                state = TRIGGER;
            end
        end
    end
endmodule
```

Here the decoder will read which switch is pressed and the corresponding number will generated, and a state machine is used here to make the output to hold for 10ms. The frequency is 1000Hz, to maintain 10ms, it needs 100 clock cycle. Once the block gets a input from the switch, the block can work as a decoder, it will generate a 4-bit address for output. Then it will make an output and hold the result for 100 clock cycle.

1kHz Clock:

```
module clock_gen(input CLK_IN, output reg CLK_OUT);
    integer timer;
    initial begin
        timer   = 0;
        CLK_OUT = 0;
    end
    always @(posedge CLK_IN) begin
        timer = timer + 1;
        // For a 10kHz clock frequency, we need 10000 clock cycles from the
        // FPGA board.
        if (timer > 5000) begin
            CLK_OUT = ~ CLK_OUT; // For 5000 clock cycles, alter the output.
            timer   = 0;
        end
    end
endmodule
```

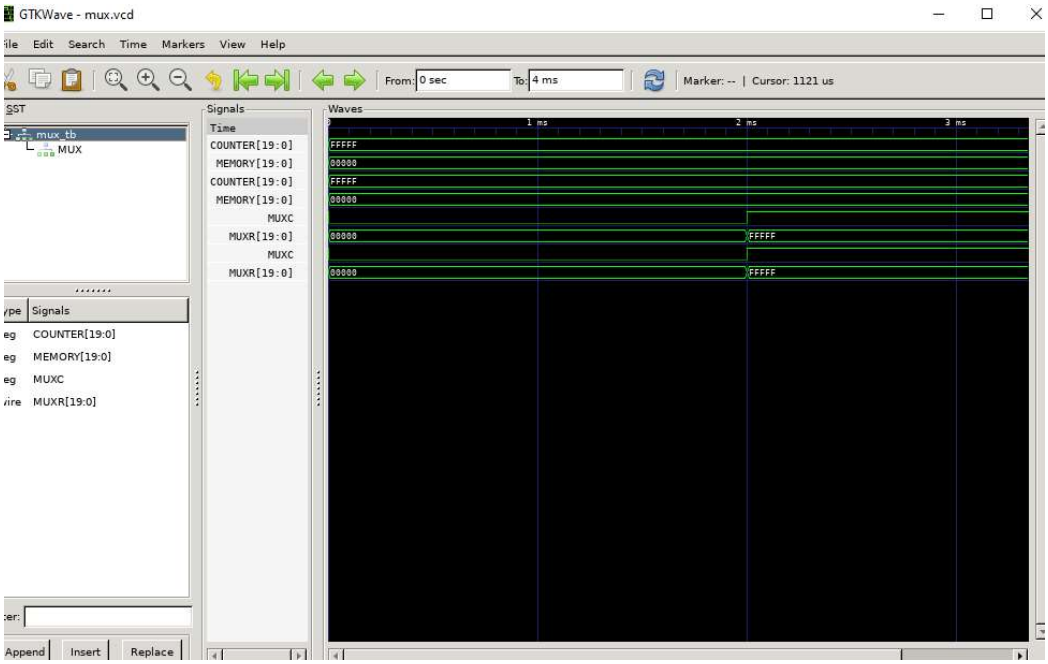
The FPGA provides a 10MHz clock, so it needs 10000 clock cycles for a 10k Hz. The clock will change once 5000 clock cycles.

Simulation

Display and drivers in base-10 format:

The simulation is verfy the decode funciton of the block. It can transform the input 20-bit time to a 20 bit displayed number of minutes and seconds. By calculation the result is correct.

Multiplexer:



The multiplexer is work as expected in the simulation result, the output changed when the MUXC changed.

Split Memory:

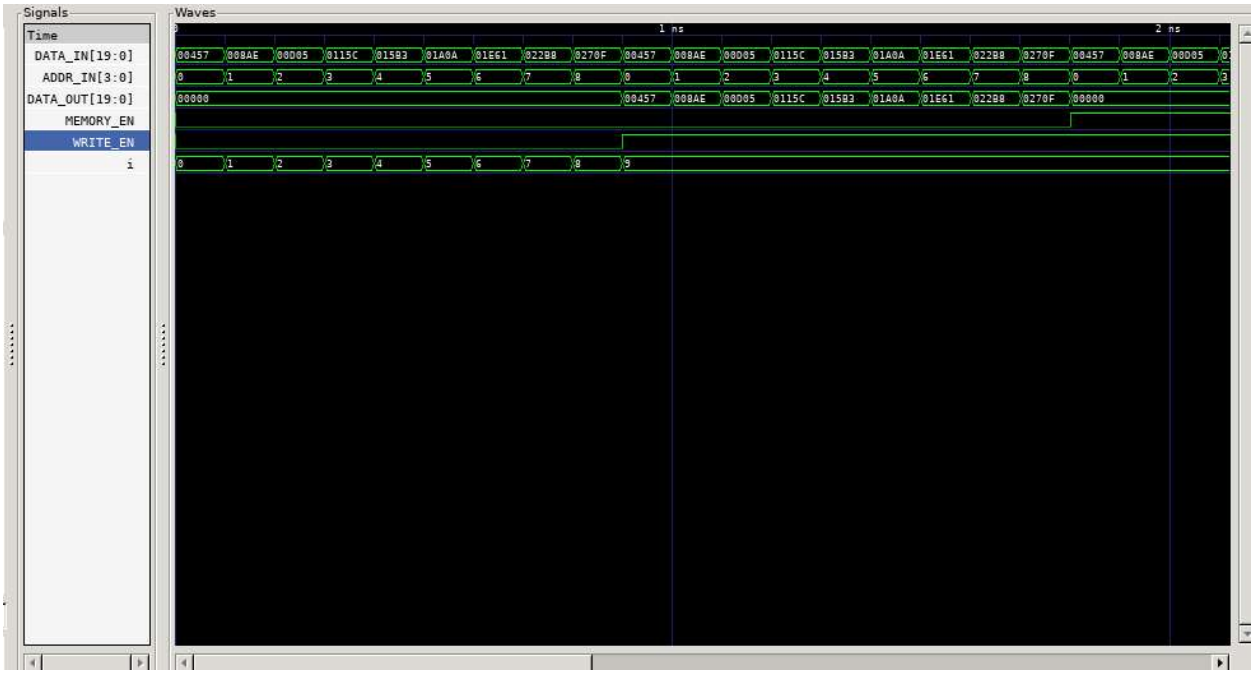


Figure: memory writing simulation

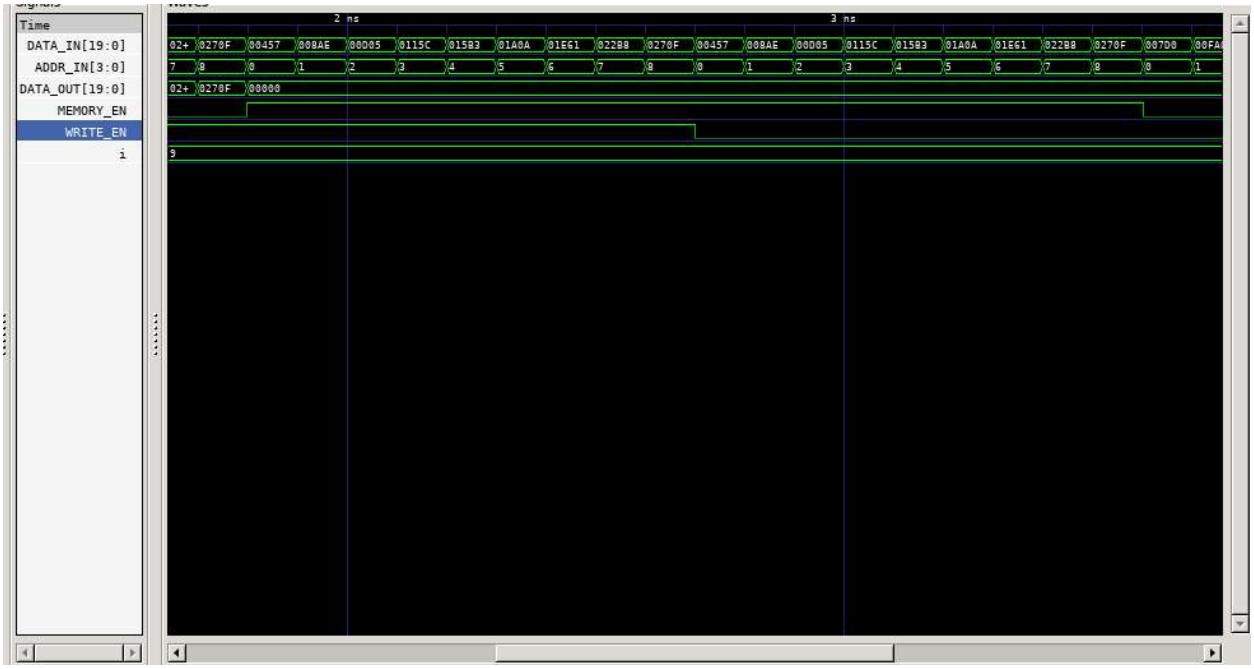


Figure: memory read simulation

The simulaiton in the first figure show the memory is initializing, and enable the written mode. Then the data out is the data of each memory. The second figure is showed in a state which both the enable are high, the output does no work. When the memory is enable, the written is not enable, the output has nothing.

The memory block is working correct, the in reading mode, the block will not output anything. In writing mode, the block has display. When both input are high, the block has no output.

Counters:

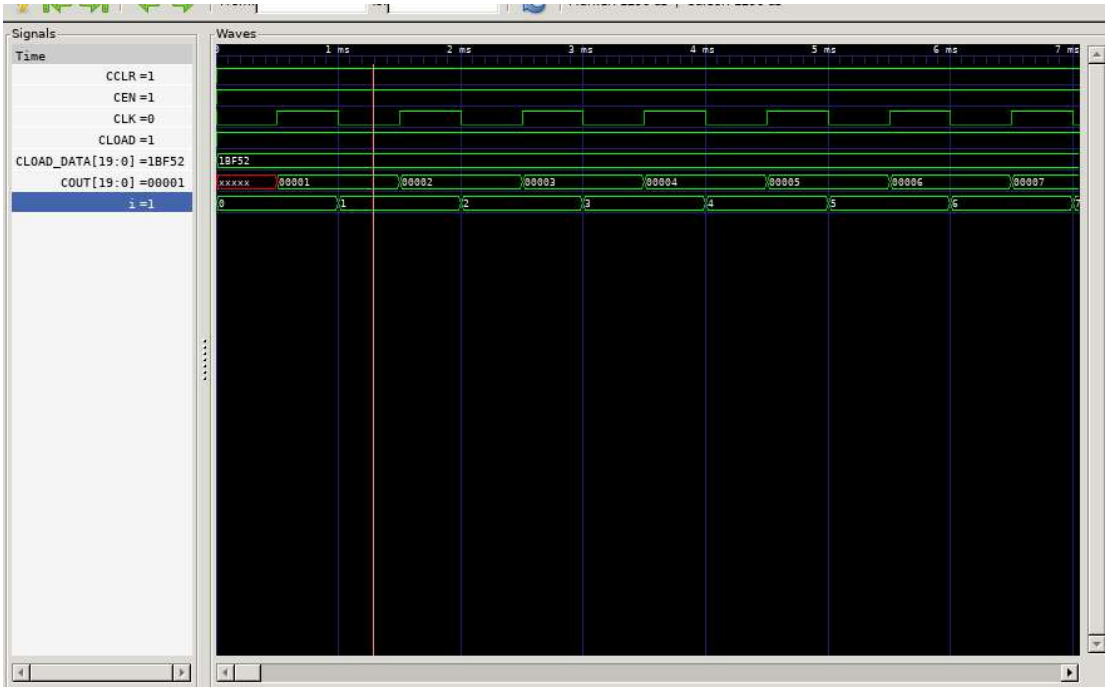


Figure: Counter begin

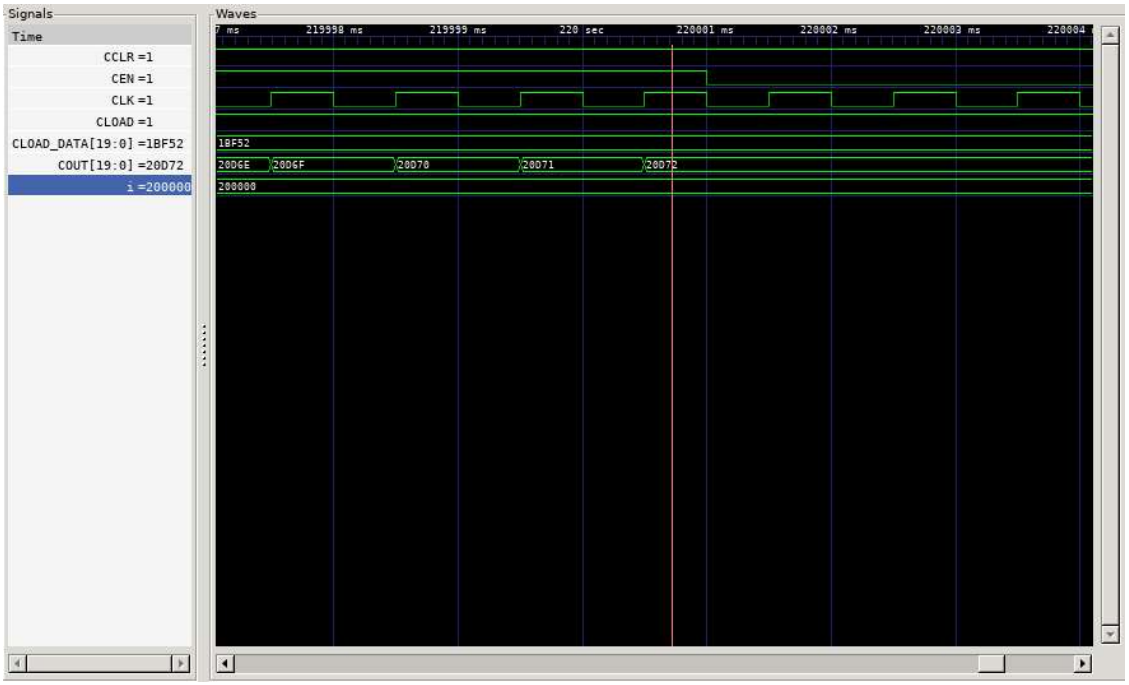


Figure: Counter stop

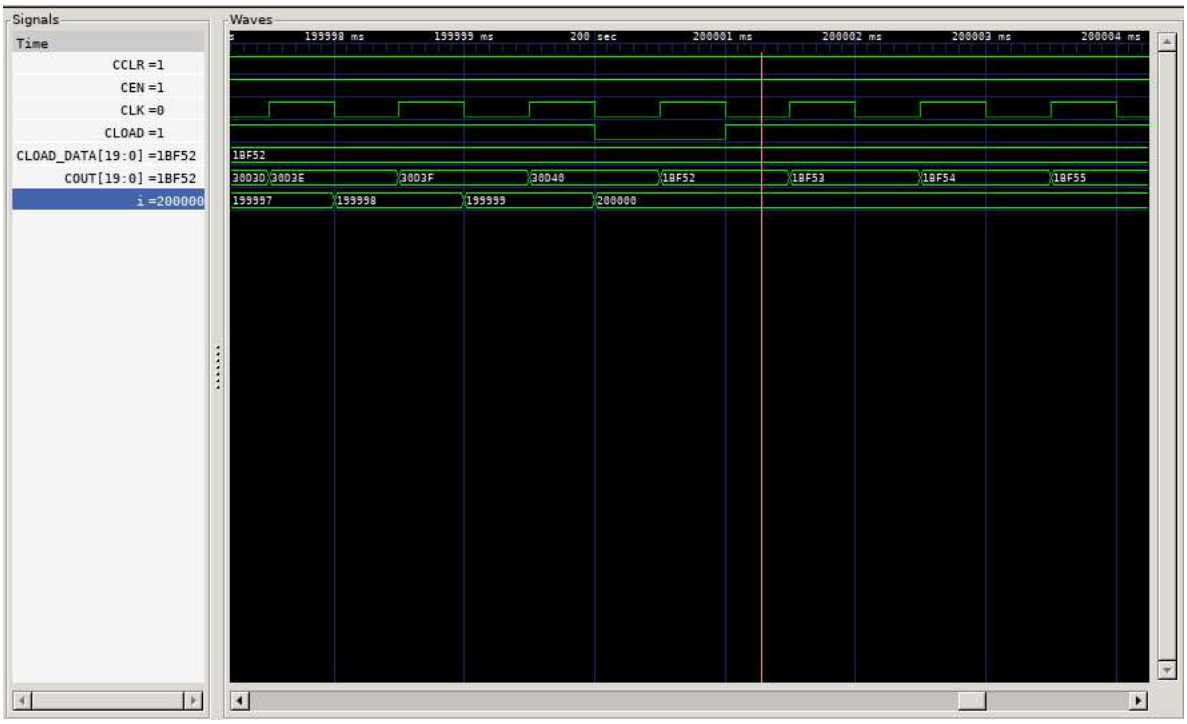
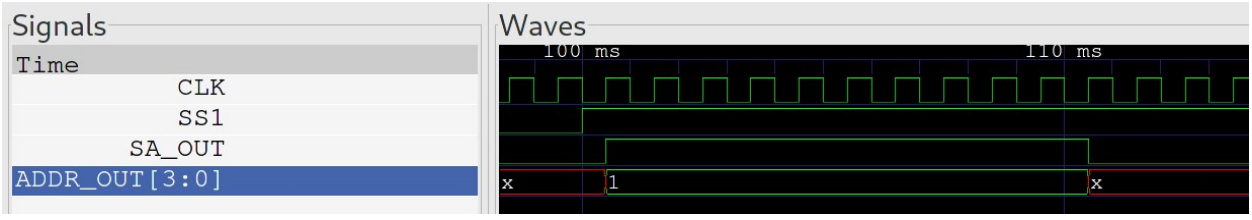


Figure: Counter load

The 3 figure shows that the counter functions. First figure is the beginning of the counter. Second figure is stopping the timer. Third figure is loading a number to the counter.

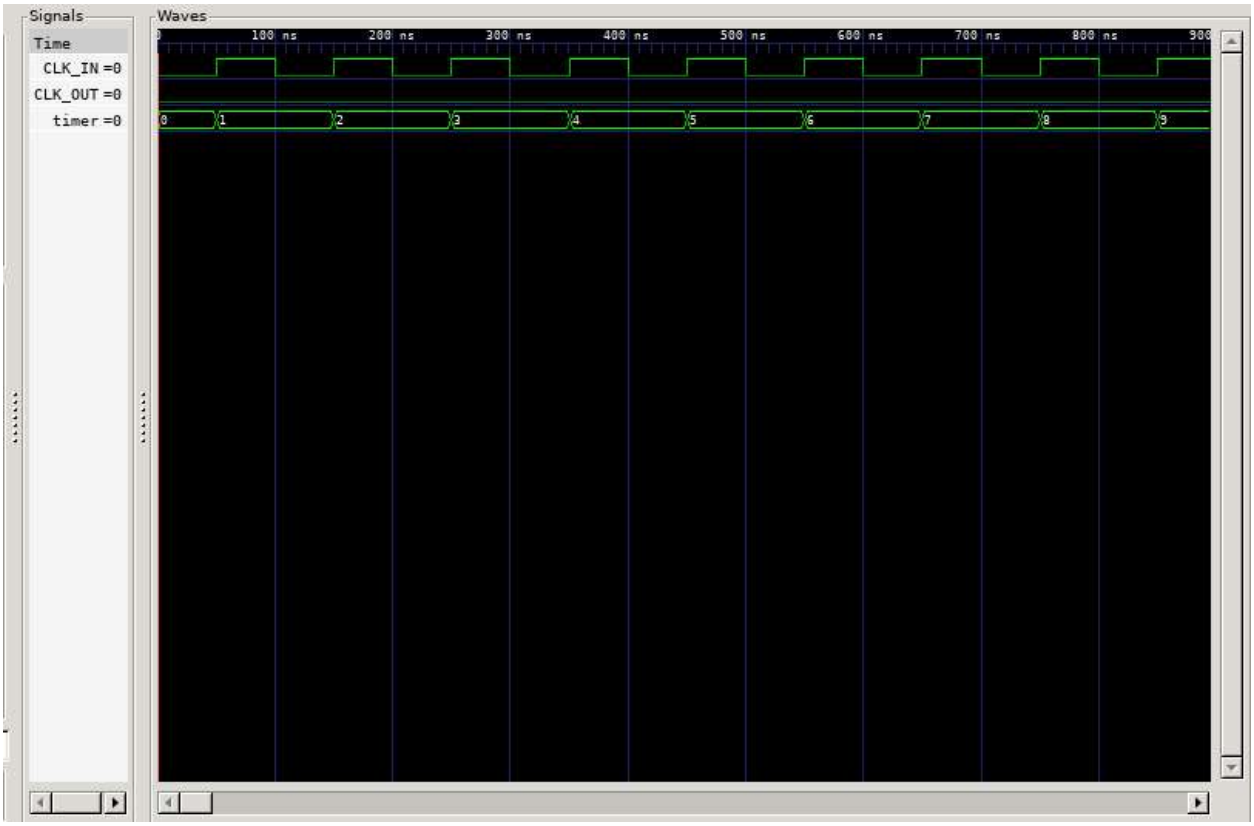
All the functions are worked as expected, can be viewed in the time diagrams.

Address switch logic:



The SA block can work as the diagram in the documentation, it can provide an address for 10ms. So it is correct.

1kHz Clock:



Clock output. The output clock frequency is 10kHz.

Verification

All the Verilog can be downloaded to the FPGA board, the board can worked as expected. So the design is complete.

Conclusion

In this lab the whole stopwatch is implemented. Students build the machine on a FPGA board successfully. The Verilog and the FPGA board is a very strong tools. Students implement a memory, a counter, a decoder, flip-flops and multiplexer in this lab. The stopwatch use the combination of these blocks.