DD 1 PROJECT (2)

Adham Ali / 900223243
Omar Saqr / 900223343
Ebram Thabet / 900214496
Digital Design I (Spring 2024)

The American University in Cairo

Contents

Introduction

Finite State Machines (FSMs) are computational models used to design both computer programs and sequential logic circuits. They are used to simulate a system with a finite number of states and transitions between those states, driven by input events. FSMs are widely utilized in various fields such as digital design, control systems, linguistics, and artificial intelligence, primarily due to their simplicity and efficacy in modeling complex behavior.

Types of Finite State Machines

There are two primary types of FSMs: Moore Machines and Mealy Machines. Both types serve similar purposes but differ in how they produce outputs.

Moore Machines

A Moore Machine is a type of FSM where the output is solely determined by the current state. In other words, the output is a function of the state the machine is in, and it does not directly depend on the input values. This characteristic makes Moore Machines relatively simple to design and understand. The output is stable and changes only when the machine transitions from one state to another.

Characteristics of Moore Machines:

The output is associated with states.

Outputs change only on state transitions.

Often simpler to design and debug due to the separation of state and output logic.

Example:

Consider a simple traffic light controller, where each state corresponds to a specific light configuration (e.g., Green, Yellow, Red). The output (light color) depends solely on the current state of the controller.

Mealy Machines

A Mealy Machine, on the other hand, is an FSM where the output is determined by both the current state and the current input. This means the output can change in response to an input event without requiring a state transition. Mealy Machines can be more responsive to inputs and may require fewer states compared to equivalent Moore Machines, but they can also be more complex to design.

Characteristics of Mealy Machines:

The output is associated with transitions.

Outputs can change immediately in response to inputs.

Potentially more efficient, with fewer states than equivalent Moore Machines.

Example:

Consider a sequence detector that outputs a signal when a specific sequence of bits is detected. The output depends on both the current state of the detector and the current bit being processed.

In this project we design an alarm system, that allows the user to adjust a clock, and an alarm and when the alarm time matches the clock time, a buzzing sound is produced.

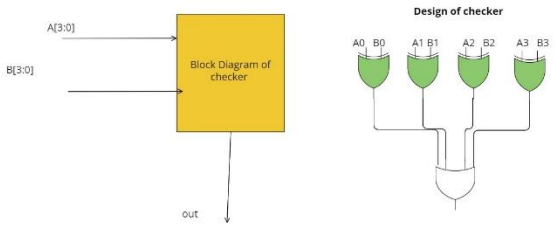
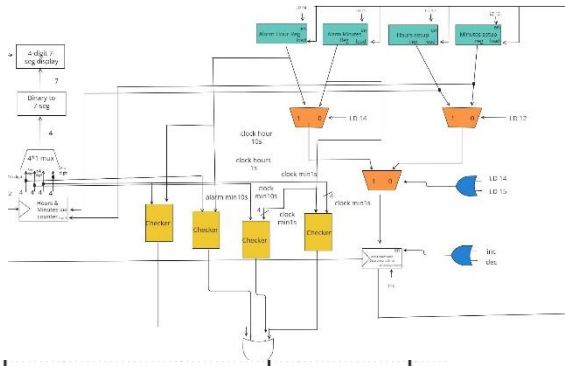


Fig.1

Design:

First, we started by designing the CU, DP block diagrams. Where we designed a checker (Fig.1) which was used to check when the alarm equals the clock. We did this by checking that the hours equal hours, and minutes equal minutes as illustrated in (Fig.2). We then produced a flag done which is equal to 1 when the alarm time matches the clock time.

Then, we designed our Logisim file where we implemented the multiplexed 7 segment display we created in the DP, CU diagram(Fig.3).



Then we started using the code from Lab 7,8 we included

the debouncer, synchronizer, rising edge detector, press button detector from Lab 7, and the 4*1 MUX, n_bit counter, 7 segment display from lab 7.

In the end we used the Push button detector, its logic is represented by (Fig.4), and the following code:

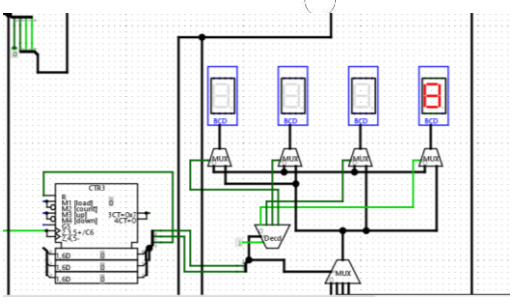


Fig.3

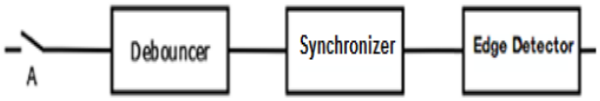


Fig.4

```
debouncer deb (.clk(clk), .rst(rst), .in(in), .out(out_1));
synchronizer syn (.clk(clk), .rst(rst), .in(out_1),
.out(out_2));
fsm detector(.clk(clk), .rst(rst), .w(out_2), .z(out));
```

We used these Push button detectors in the code to detect the signals of the CU, we used the following code:

```
   SMASH_buttondetector Ub (.clk_in(clk_200_hz), .rst(rst),
.in(U), .out(out1));
  SMASH_buttondetector Db (.clk_in(clk_200_hz), .rst(rst),
.in(D), .out(out2));
  SMASH_buttondetector Rb (.clk_in(clk_200_hz), .rst(rst),
.in(R), .out(out3));
  SMASH_buttondetector Lb (.clk_in(clk_200_hz), .rst(rst),
.in(L), .out(out4));
  SMASH_buttondetector Ce (.clk_in(clk_200_hz), .rst(rst),
.in(C), .out(out5));
wire [4:0] usedoutput = {out1, out2, out3, out4, out5};
```

Then we used “usedoutput” to control the transitions in the finite state machine.

For example in,

```
 if (usedoutput == 5'b00001) begin
        nextstate = clk_hour;
        LD=5'b11000;
        chooser_clk=clk_200_hz;
        sec_en=0;
        Up_down=1'bx;
        min_en=0;
        hour_en=0;
        alarm_min_en=0;
```

```
        alarm_hour_en=0;
      end
```

We check if we are in the clock mode, and button C is pressed then we move to the next state which is to adjust the hours of the clock. Then we assign the LEDs accordingly, the LEDs are in the following order: 15,14,13,12,0. Then we change the clock frequency to be suitable for the pushbuttons. Then we stop the clock from counting, during the adjustment of the alarm. We assign Up_Down as x to avoid incrementing or decrementing until the user chooses to do so, and we assign all enables to 0 to avoid incrementing during the adjustment if not U,P buttons are pressed. We wrote the logic of the rest of the states in the following fashion then we used

assign z_flag = ((clock_hr_units == alarm_hr_units) & (clock_min_units == alarm_min_units)& (clock_min_tens == alarm_min_tens)& (clock_hr_tens == alarm_hr_tens) & (seconds_OUT==0));

Z_flag: it is assigned to 1 when we are not in the 0000,0000 state for the first time, and when the time of the clock, and alarm matches.

Then we choose which digits will be displayed on the fpga with the following code:

```
always @(state) begin
    case(state)
    alarm_hour:begin
    display4 = ai_4;
    display3 = ai_3;
    display2 = ai_2;
    display1 = ai_1;
    end
    alarm_min: begin
    display4 = ai_4;
    display3 = ai_3;
    display2 = ai_2;
    display1 = ai_1;
    end

    clk_hour:begin
    display4 = wi_4;
    display3 = wi_3;
    display2 = wi_2;
    display1 = wi_1;
    end
```

```
    clk_min:begin
    display4 = wi_4;
    display3 = wi_3;
    display2 = wi_2;
    display1 = wi_1;
    end
    display_clock:begin
    display4 = wi_4;
    display3 = wi_3;
    display2 = wi_2;
    display1 = wi_1;
    end
    alarm_mode:begin
    display4 = wi_4;
    display3 = wi_3;
    display2 = wi_2;
    display1 = wi_1;
    end

    endcase
end
```

We then called modules that accept outputs from the states.

```
Clock_Counter clock(selector_clk, reset, min_en,hour_en, sec_en,
Up_down,clock_hr_units, clock_min_units, clock_hr_tens,
clock_min_tens,seconds_OUT);

Alarm_Counter alarm(clk_200_hz, reset, alarm_min_en,
alarm_hour_en, Up_down, alarm_hr_units, alarm_min_units,
alarm_hr_tens,alarm_min_tens);

counter_x_bit #(2,4) gut(.clk(clk_200_hz),
.reset(reset),.en(1'b1),.Up_down(1'b1), .count(sel));

seven_segment
display(.inp1(display4),.inp2({display3}),.inp3(display2),.inp4(
```

```
{display1}),.enable(sel),.anode_active(anode_active),.segments(s
egments));
```

The first module is responsible for counting in the clock mode, the second module is responsible for counting the alarm mode(incrementing or decrementing). The third module is responsible for generating selection lines for the 4*1 mux, and the 7-segment display is responsible for displaying on the FPGA.

Implementation Issues:

1-) When we started the integration between lab 8, and lab 7 files as a starting point to our code, we faced the inconsistent naming problem. This problem was fixed by first removing any redundant, or duplicate files. Then we changed the naming to be consistent, and the order of inputs in accordance with the guidelines.

2-)When we tried to test the push button detector, it did not work. Later, we found that this error was due to a misspelling of the name of the clock.

3-) When we tried to change the frequency of the clock to count every one minute, we faced a problem that he counted 1 after only 28 seconds, then it counted every 1 minute. So, here we changed the implementation of the counter.

4-) When we were testing, the new counter the seven-segment display did not count at first, later we discovered that this was because we used the wrong assignment for the wires that store the minutes, and hours value, it was a constant 0 assignment.

5-) After fixing the above issue, we faced another error the least significant digit of the clock (the minutes) was always 0, the last of digits were updated correctly, however.

6-) The above error was due to an error in the inputs of the 4*1MUX that was responsible for the multiplexed display.

7-) The alarm counter, had a problem it was incrementing continuously after pressing U, and decrementing continuously after pressing D. We found that this was because we passed the wrong parameters to the counter.

8-) When we tested the state transitioning for the first two states we had a problem that no transitioning occurred. This is due to an error in the clock divider, which was used to produce a 200 Hz clock for the pushbutton detector.

Contributions:

Omar Saqr:

1-) Created the clock counter, and made sure it is working correctly(resets after 23:59 and is counting with the correct frequency which is 1Hz), and I did so by modifying lab7.
2-)Created a counter for alarm mode that increments or decrements the register that stores alarm values based on the signal from BTNU, and BTND. Also, tested the code for various edge cases of overflow, underflow, and modulus correctly.
3-) Created 5 buttons to represent the buttons BTNU, BTNC, BTND, BTNL, BTNR. Then used a push button detector, and create a signal to control the states. For example, it would be used to control transitions from clock/alarm mode to adjust mode and transitions within clock mode.
4-)Fixed the bug within the transitioning between states. As currently, there is no transitioning on the FPGA, only the display_clock mode worked.
5-) Complete the functionality of the rest of the states(the last 3 states), which represent 3 modes of the adjust part clk_min, alarm_hour, and alarm_min .
6-)Created the DP, CU blocking diagram, and Logisim simulation file.
7-)Wrote the report.

Adham Ahmed:

1-) Made sure that the integration of essential files from the lab is done correctly. Debugged errors, and fixed naming inconsistencies among the files from lab 7,8.
2-) Created a counter for alarm mode that increments or decrements the register that stores alarm values based on the signal from BTNU, and BTND. Also, tested the code for various edge cases of overflow, underflow, and modulus correctly.
3-) Integrated the Alarm counter and clock counter within the driver (main) file. Then, made sure each of the two counters is working correctly, and fixed bugs.
4-) Created 5 buttons to represent buttons BTNU, BTNC, BTND, BTNL, BTNR. Then used a push button detector, and create a signal to control the states. For example, it would be used to control transitions from clock/alarm mode to adjust mode and transitions within clock mode.
5-) Integrated the Alarm Counter, and clock counter in the main file, while taking into consideration the different clocks that should be assigned. And wrote the transitions of these two states, Checked that the integration was successful by checking the display_clock state, and clock_hour state on the FPGA.
6-) Complete the functionality of the rest of the states(the last 3 states), which represent 3 modes of the adjust part clk_min, alarm_hour, and alarm_min .
7-) Fixed the bug within the transitioning between states. As currently, there is no transitioning on the FPGA, only the display_clock mode worked.
8-) Created the buzzer bonus.

Ebram Rafaat:

1-) Made sure that the integration of essential files from the lab is done correctly. Debugged errors, and fixed naming inconsistencies among the files from lab 7,8.
2-) Created the clock counter, and made sure it is working correctly(resets after 23:59 and is counting with the correct frequency which is 1Hz), and I did so by modifying lab7.
3-) Integrated the Alarm counter and clock counter within the driver (main) file. Then, made sure each of the two counters is working correctly, and fixed bugs.
4-)Fixed the bug within the transitioning between states. As currently, there is no transitioning on the FPGA, only the display_clock mode worked.
5-) Integrated the Alarm Counter, and clock counter in the main file, while taking into consideration the different clocks that should be assigned. And wrote the transitions of these two states, Checked that the integration was successful by checking the display_clock state, and clock_hour state on the FPGA.
6-) Created the ASM diagram.
7-)Create a proper readme file for github.

Validation and testing:

In order to validate the functionality of:

The states: we used the LEDs to verify this. In other words, we used the specifications of the project operating specific LEDS in order to ensure correct transition between the states.

The clock: we simulated the behavior of the clock using simulation files and testbenches.

The clock / alarm behavior: we used a faster version of the clock by dividing it by a lower number. This gave us the ability to track it without waiting for the normal behavior of the hours.

The buzzer: we connected it to a led in order to check if there are hardware problems.

Logic errors: we utilized the schematic section after running synthesis to show a diagram of the circuit in order to visualize the bevahior.