EEL 3701C – Digital Logic & Computer System

LAB 4

Goal:

By the end of this lab, you should be able to:

- Design an alarm system for a custom clock.

Problem 1 – Alien Part 1 – What happens after 9?

The citizens of Planet Mux don't know what time is. Somehow, this spacefaring civilization has managed to colonize half of the Milky Way without any sort of clock or watch (do sundials even work in space??). The Government of Mux has contracted you to invent their very first clock, because Lord Mux missed his favorite TV show *To Bit or Not to Bit* after realizing he didn't know what the current time was.

There are 10 Muxian minutes in a Muxian hour and 6 hours in a Muxian day. The earliest possible time in Muxian is 0:0. The latest possible time is 5:9.

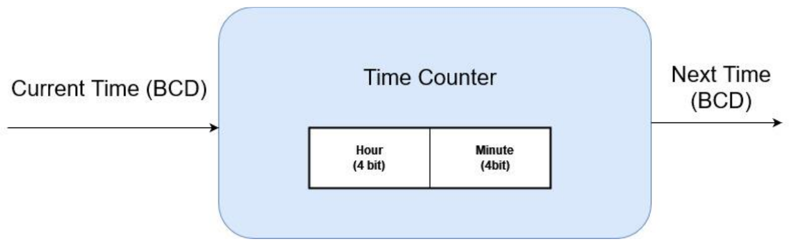


Figure 1 - Black box diagram for the TimeCounter. The inner section shows the structure of the BCD input and output.

The first step to making the clock is counting in this alien time system. We must make another clock divider circuit from the last lab to a 1 Hz clock (1-sec). Then we will use a process and a combination of if /else to generate the muxian clock. Check if the current minute is less than 9. If less than 9, set the next minute equal to the current minute plus 1, and the next hour equal the current hour. Else, set the minute back to 0. In this same **else** statement, there should be another **if** much like the previous one, but this time for the hours instead!

Increase the muxian clock minute with every one-second clock tick and display it on the seven-segment display. Take the help of previous lab 3 to implement the seven-segment LCD part. There is two seven segment display in the MXDB board. Use the left one for the muxian hour and the right one for the muxian minute.

Pre-Lab Homework (10 pts)

1. Implement the design in **VHDL CODE: Problem 1** by instantiating a 1 hz clock and building the logic for 7 segment displays. Complete the code for muxian clock. Compile and make sure that the written code is error free. **(10 pts)**

In-Lab Implementation (7 pts)

Synthesize your circuit, download, and test your design in the MXDB Board.

For your deliverable, record a video of the implementation that muxian clock is working and its repeating its order after the last possible time 5:9.

Problem 2 – Alien Alarm Clock - The Finite State Machine

Your successful invention of the very first Muxian clock has caught the eye of Dr. Latch, the most premiere engineering professor at the University of Flipflop. She sees great potential in the product and thinks that you can take it further by allowing Muxians to set a very bright reminder of the current time (Muxians are deaf, after all).

After a human translator interpreted Dr. Latch's rather convoluted instructions, the following directions appeared on your desk:

Make a state machine with four states: **INIT**, **STATE_TIME**, **ALARM**, **SNOOZE**.
The state machine has 1 input: **snooze_btn**. (And an *implied* **CLK**)

INIT:

- Set the alarm hour and min.
- Move to STATE_TIME state

STATE_TIME (default state):

- Move to **ALARM** state if muxian hour and min are equal to alarm hour and min.

ALARM:

- Display the alarm in the seven segment LCD.
- Move to SNOOZE state.

SNOOZE:

- Return to **STATE_TIME** state if snooze_btn is HIGH. Else, continue to display the alarm beep.

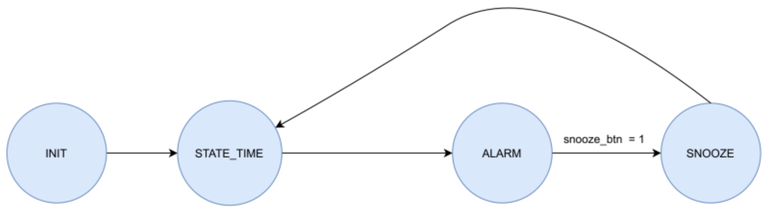
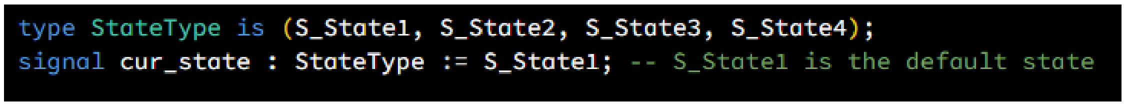


Figure 2 - FSM state machine for the alien clock

Create a new VHDL file **FSM.vhd**. Set up the necessary inputs and outputs. Then, we will define an **enum** type. This will allow us to make a **signal** of said type to store the current state of the FSM!

For example, in the picture below we make a type called **StateType** and give it 4 possibilities.
You should give these meaningful names as stated above in the state list (INIT, STATE_TIME, ALARM, SNOOZE) , but beware, **"time" is a reserved keyword in VHDL!** Worse yet, using "time" will NOT THROW ERRORS, but your circuit will NOT WORK! **Do not use "time" as a variable name or state name, ever!**



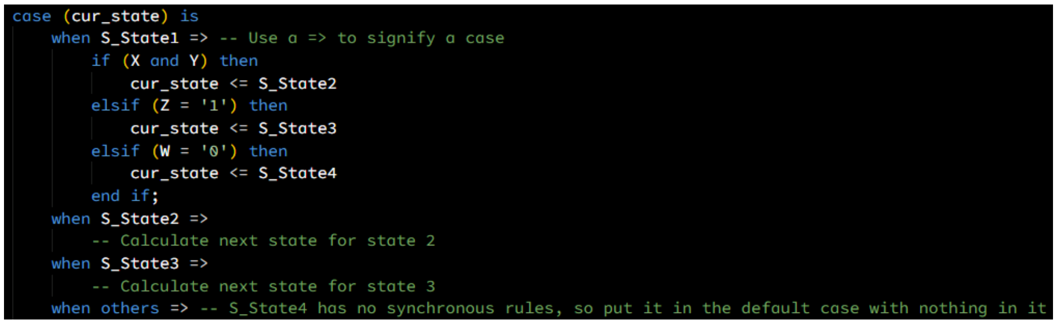
```
type StateType is (S_State1, S_State2, S_State3, S_State4);
signal cur_state : StateType := S_State1; -- S_State1 is the default state
```

Figure 3 An example of defining states

Next, you will want to make a **process** with **CLK, RESET** in its sensitivity list.

As always, first check to see if the reset input is true. If it is, set the current state back to the default. If the reset is not true, check to see if **CLK** is on a rising edge. This is the standard process when dealing with clocks and resets and will not be mentioned again going forward.

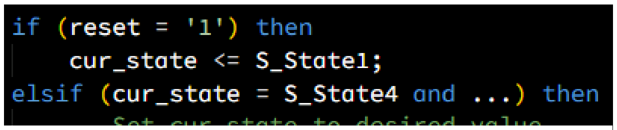
Now, we will use a **case-when** statement to determine the next state (note: these can only be used inside processes!). Using the example from above, that would look like:



```
case (cur_state) is
    when S_State1 => -- Use a => to signify a case
        if (X and Y) then
            cur_state <= S_State2
        elsif (Z = '1') then
            cur_state <= S_State3
        elsif (W = '0') then
            cur_state <= S_State4
        end if;
    when S_State2 =>
        -- Calculate next state for state 2
    when S_State3 =>
        -- Calculate next state for state 3
    when others => -- S_State4 has no synchronous rules, so put it in the default case with nothing in it
```

Figure 4 An example of using the state machine

Notice how S_State4 is never defined. Thus, it will be activated in the **others** (aka default) case. Instead, it has an **asynchronous** rule to get to a different state. Remember, since this case-when statement is inside the CLK if-statement, then anything inside it is **synchronous**! To make it



```
if (reset = '1') then
    cur_state <= S_State1;
elsif (cur_state = S_State4 and ...) then
    -- Set cur_state to desired value
```

Figure 5 Clock Enabled synchronous states.

asynchronous, we must check the asynchronous input in between the reset check and the clock check. Add another **elsif** to do this, like so.

Make sure to tie the snooze_btn to the ground while implementing the alarm states, as leaving this PIN ungrounded will force you to drive it in the floating state.

To implement the alarm beep, you can choose any text you want. It is better to use the dot letter (.) in the seven-segment display as the other numbers (0 - 9) will be used for the muxian clock.

Pre-Lab Homework (12 pts)

1. Implement the design in **VHDL CODE: Problem 2** by instantiating the FSM state for the clock. Complete the code for the alarm clock and make sure the alarm is ringed on time. Compile and make sure that the written code is error-free. **(12 pts)**

In-Lab Implementation (10 pts)

Synthesize your circuit, download, and test your design in the MXDB Board. Include the part that shows that alarm is set off when **snooze_btn** is HIGH and muxian time is shown again in the display.

For your deliverable, record a video of the implementation that alarm is working perfectly.