ELEC 326 Lab 2

Alarm Clock

Objectives

Work in Xilinx Vivado and design a simple alarm clock in Verilog. There will be two alarms that can be set, either of which can be triggered based on the current clock time.

Inputs/Outputs

The individual inputs and outputs of the top module alarm_clock are already connected. You only need to code two modules, alarm_fsm and clock_fsm, described in steps 1 & 2 below.

Inputs to the Alarm Clock

- CLK – the on board 100Mhz clock
- SW[7:0] – the eight board switches
    - SW[7:2] are unused in this lab
    - SW[0] enables alarm0
        - Ialarm0_fsm.alarm_en_pi
    - SW[1] enables alarm1
        - Ialarm1_fsm.alarm_en_pi
- BTN[3:0] – the four board push buttons
    - BTN [0] is the minute increment button. When not pressing other button, this button will increment the clock minute value by 1 when pressed.
        - When an alarm is being programmed, this instead increments the alarm's minute value by 1.
    - BTN[1] is the hour increment button. When not pressing other buttons, this button will increment the clock hour value by 1 when pressed.
        - When an alarm is being programmed, this instead increments the alarm's hour value by 1.
    - BTN[2] displays the current setting for alarm 0, and allows alarm 0 to be programmed with buttons 0 & 1.
    - BTN[3] displays the current setting for alarm 1, and allows alarm 1 to

be programmed with buttons 0 & 1.

Outputs of the Alarm Clock

- LED[7:0] – the eight board LEDs
  - LED[5:0] will display the current clock seconds in binary. It should never exceed 59, 6'111011.
  - LED[6] is the output of alarm 0. It will blink when alarm 0 is triggered.
  - LED[7] is the output of alarm 1. It will blink when alarm 1 is triggered.
- SEG[6:0], DP, AN[3:0] – the seven segment display
  - This works exactly as it did in the previous lab, although the hours and minutes from either the clock or the alarm being displayed are passed through a binary to decimal converter to display a more human readable digital clock.
  - To look more like a real world clock, there is also a modification to make the clock 'blink' when it is first running, before the user has programmed the clock

Provided files

Download lab2.v and the Nexys4DDR design constraint file from the FPGA Lab 2 Files on Canvas.

In this lab, the combinational logic to display the currently set time and currently programmed alarms is already present in lab2.v. You will only need to implement the two blocks of sequential logic, alarm_fsm and clock_fsm, described below.

Step 1: 12-hour Alarms

The module alarm_fsm is a module that has a set of clock inputs for minute and hour values, and a set of minute and hour value outputs that represent the 'programmed' alarm time. When the input values match the programmed values and the alarm is enabled, the module drives an 'alarm triggered' event that the system outputs to one of the board LEDs.

In lab2.v, implement the alarm_fsm module, as follows:

- The outputs of the module that you will need to generate logic for are:
  - minutes_po – the 'minutes' value of the alarm
  - hours_po – the 'hours' value of the alarm
  - alarm_triggered_po – the alarm event itself, true when the alarm triggers

- If the alarm is enabled (input: alarm_en_pi) and the clock input time in minutes and hours matches the alarm minutes_po and hours_po, trigger the alarm (output: alarm_triggered_po)
- The event alarm_triggered_po should remain high until it is cleared, i.e. the alarm enable is driven to 0
    - The alarm enables signals are mapped to switches, so this means once an alarm goes off, you will need to flip the alarm's switch to disable the alarm
- Increment the alarm minutes_po and hours_po reg based on the increment inputs
    - Both values should increment whenever the increment inputs are high. The increment inputs will be high for exactly one clock cycle whenever the corresponding buttons are pressed
    - Both values should wrap appropriately
        - Minutes_po should transition from 59->0
        - Hours_po should transition from 12->0

You should be able to test this step before you move on to Step 2. There are two alarm_fsm modules instantiated in the alarm_clock top level module. Both should function, using the buttons and switches as described in the inputs/ouputs section above. Test them to make sure the alarms trigger correctly before moving on to the next step.

Step 2: 12-hour Clock

The clock_fsm module is a simple 12-hour digital clock that continuously counts upwards. It has seconds, minutes, and hour values stored. It also has inputs for manually programming the minutes and hour values, similar to how the alarm_fsm module minutes and hour values were programmed in the previous step.

In lab2.v, implement the clock_fsm module.

You wrote the alarm_fsm module first because most of the logic of the clock_fsm module is identical to the alarm_fsm module, with one addition:

- Like the alarm_fsm module, the clock_fsm minutes_po and hours_po outputs should increase by 1 whenever the corresponding increment input is true.
- However, there is an additional rule for the clock_fsm: the clock should actively count up once per second. There is a clk_en_pi signal on the clock_fsm module. It will be active one cycle of clk_pi per second.

    - The seconds_po signal should increment every second – so on the posedge of clk_pi, whenever clk_en_pi is true.
    - Like minutes_po, seconds_po should wrap from 59 -> 0.
- Whenever the seconds value wraps from 59->0, the minutes_po should increment by one. Similarly, when the minutes_po value wraps from 59->0, the hours_po value should increment
- Note: Like most real world digital clocks, you do not need to increment hours_po when programming minutes_po with the buttons.
    - As an example, if the clock reads "1:59" and minute passes, it will read "2:00". However, if it reads "1:59" and button 0 is pressed, the clock can transition to "1:00". Many real world clocks that have both hour and minute programming buttons will do this for simplicity. You can do either ("1:00" or "2:00") when button 0 is pressed, but the logic is far simpler if you choose to only increment hours_po when minutes_po naturally rolls over from 59 to 00 with the passage of a full minute.

Step 3: Simulation

After you successfully complete your implementation, download sim1.v from the FPGA Lab 2 Files on Canvas.

Add this file to your completed design as a simulation file.

Simulate the design in Vivado as any simulation type (behavior, post-synthesis/implementation). Set the simulation time to run 2,000,000 ns under Simulation Settings.

Maximize your simulation waveform window.

Zoom out until you see the CLK waveform (clock should be cycling), at about 10ns minor tick marks or 50ns labels.

Visit an interesting time-slice, take a screen shot, and describe for your report. This should be a point where one of the other output signals changed, and note why it changed, see simex.png.

The point of this step is to just become familiar with the simulator in Vivado.

Report:

Make sure to format your report and answer all of the sections as last time.

1. Zoom in on the alarm_fsm modules in your RTL Schematic by double clicking on the +. Describe and relate to your Verilog code.

2. Include 4 screen shots, 1 each for RTL Schematic, Zoomed RTL Schematic (alarm_fsm), Design, and Simulation. Briefly (2 sentences max) explain each.
3. Relate the Design screenshot to the Design screenshot from Lab1, does this lab utilize more or less of the board when compared to Lab 1?
4. What is the significance of the number in the line of code below (from lab2.v: module seconds_clkdiv)? What is the purpose of this line of code? What will happen if this value is changed?

   If (counter == 32'h5F5E110)
5. Briefly (2-3 sentences max) describe how the modules work together. What is the purpose of button_debouncer?
6. In Lab 1 we saw that each of the functions of the calculator were being performed in parallel and only one of the functions was displayed for output, as selected by the button. In this lab are there any significant calculations being performed in parallel?

Bonus!

+5 Points – Implement Snooze

Add logic so that when, and only when, an alarm is going off, pressing BTN[4] turns off the alarm and adds 5 minutes (or your favorite snooze time) to the next alarm time to go off. Make sure that it doesn't increment the actual, saved alarm time, but rather sets (or increments) a snoozing flag. You will have to change the structural design of the code for the alarm_fsm and add additional logic and wires. Make sure to add BTN[4] to your alarm_clock module.

+5 Points – Dual Displays

The alarm clock has two alarms, but the user doesn't know which one is set to go off next. Utilize the left 4 seven-segment displays to show the next time that an alarm will go off. Make sure that the alarms are enabled via the switches, and after an alarm goes off or the alarm time is modified, the time displayed on the left display changes. You will need to modify the sevenSegDisplay module, which sets the left displays via the segmentFormatter for anodes 4-7 to your next alarm value. The sevenSegDisplay module will also need to be changed to receive the next alarm value. If no alarm is set, make sure to display blanks. One hint is to create a new set of wires for the next alarm and a new module that takes both alarm wires, switches, and current time, and sets the next alarm wires.