EE/CprE 465 -- Fall 2015

Lab 10: Elevator Controller Design

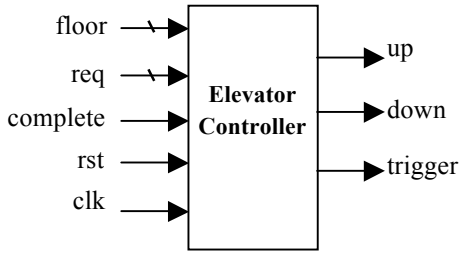
In this lab, we will learn how to design finite state machine (FSM). We will design an elevator controller for a toy parking garage. We will do Verilog coding, and then synthesize and layout the design by tools. We will have a chance to see the performance of our design.

First, please go online to watch how this toy parking garage works.

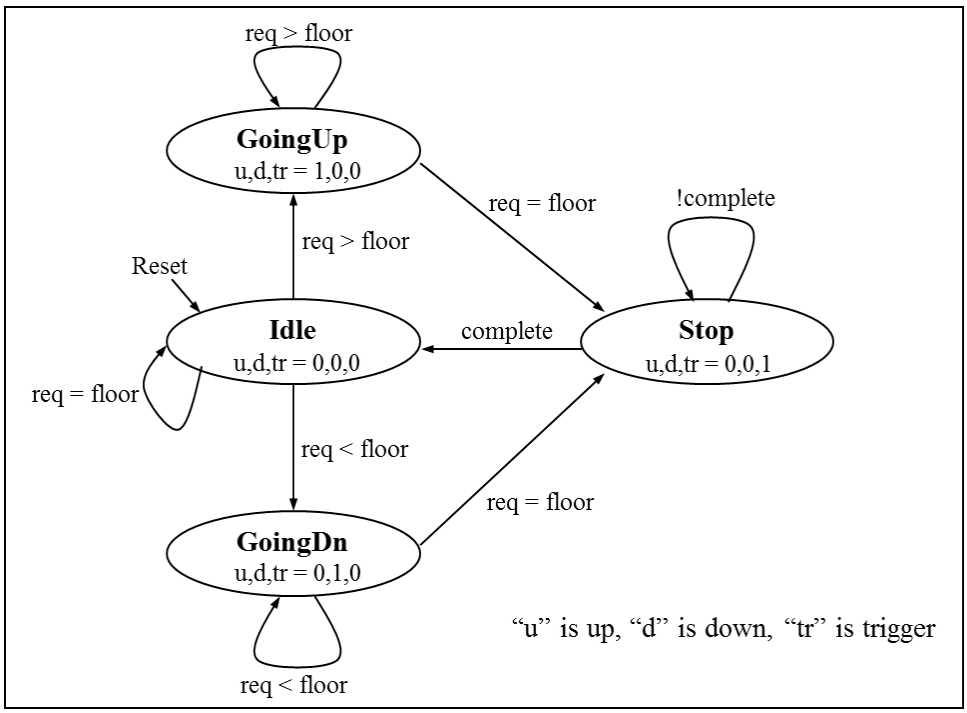
http://www.youtube.com/watch?v=7_7tBTBnOzQ&p=9925BEB8A3E15E02&playnext=1&index=72

In this lab, we assume the toy parking garage has six stories. This elevator controller moves the elevator either up or down to reach the requested floor. Once at the requested floor, it will trigger the transfer arm controller. The transfer arm controller then transfers the toy car from the elevator to the requested floor or from the requested floor to the elevator. When the transfer is completed, the transfer arm controller will inform the elevator controller so that the elevator controller may handle the next request. For this lab, we will just design the elevator controller and ignore the transfer arm controller.

The interface of the elevator controller is given in the following diagram. Inputs "floor" and "req" are multiple-bit signals specifying the current floor and the requested floor of elevator respectively. "complete" is a signal from transfer arm controller indicating that the transfer is completed. "rst" is the reset signal. "clk" is the clock signal. "up" and "down" tells the elevator to go up and down respectively. "trigger" is a control signal to inform the transfer arm controller to transfer the toy car.



The elevator controller can be described by a finite state machine (FSM) model. The FSM has four possible states: *Idle*, *GoingUp*, *GoingDn*, *Stop.* The state transition diagram of the FSM is given below:



Task 1: Verilog HDL Design.

Write the elevator controller design in Verilog HDL. Because the elevator controller uses AAA batteries, we should try our best to save power. Please keep this in mind when writing the design in Verilog HDL. Hint: There should be a clock signal but make sure the frequency is as low as possible. Why? Because system clock frequency affects dynamic power dramatically. So we just define it as 1MHz. Also, try to simplify your design as much as possible. Use ModelSim to simulate and verify your design.

Task 2: Adding Sleep Mode.

To further save more dynamic power, we will design a sleep mode feature to make the controller sleep if there are no active input signals after 3 minutes. This is necessary because children often forget to power off their toys so that batteries will be dead very soon. Work out a solution and draw a block diagram. Then implement this feature into your Verilog HDL codes. Run ModelSim simulation again to verify it. Hint: The sleep mode is some kind of clock-gating mode. Use a timer which is basically a counter, to keep track of time. When time is up, disable the clock which is fed into the elevator controller, and then the elevator controller will sleep. To wake it up, we need to keep tracking if there is an active input, which means the input changes between 2 consecutive clock cycles. As we can see from this task, we may save more dynamic power by system design, and not only depending on the optimization of the tools (e.g., inserting clock-gating cells automatically).

Things to turn in for Task 1 and Task 2:

1. A modified diagram of the above FSM after adding sleep mode.
2. Verilog HDL code.
3. ModelSim simulation results (in the form of pictures).

Task 3: Synthesis and Layout with Clock Gating.

Use the Cadence tools RTL Compiler and Encounter to do the synthesis and layout. Please note that this is the first time you will set up the whole scripts for a design by yourself and it is a warm-up for the project. Hint: Just modify the scripts we used in the previous labs. In particular, you need to: (1) Keep the library unchanged, which means we use the same library as previous labs; (2) Change the input Verilog codes file to your design file; (3) Modify the names of ports in .conf file (constraints file) because this time our design has different port names from those in previous labs; (4) Change the clock period and keep the duty cycle as 50%.

In addition, in order to save power, please make sure clock gating will be performed by RTL Compiler automatically during synthesis. Please do not try to insert clock gating manually because you have already practiced it in Lab 9. After running RTL Compiler and Encounter, please check the log files to make sure there is no error.

Things to turn in:

1. Constraints for RTL Compiler and Encounter.
2. Report on clock-gating summary as we did in Lab 9.
3. Report for area, timing and power (dynamic and leakage) after RTL Compiler synthesis.
4. Report for area, timing and power after Encounter layout.

Task 4: Power Saving by Power Gating. (Just something for you to think about. No need to turn in.)

We have already done something to save dynamic power. Is there any way to save leakage power by system design? The answer is yes. For leakage power, the technique is power gating, which cuts the power of the modules if they are asleep. But before cutting their power, we need to store current values inside any memory elements. During wake up, those values should then be restored. Of course these operations take time, just like your laptop sleeping and waking up.