Additional Hints for Lab 4

Signal Names and Pinouts

The following screen capture shows the I/O signal names I used. You are welcome to use the same ones.

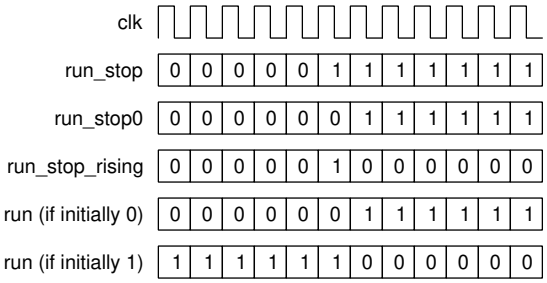
The output-only signal is **`seconds_out`** and there is a corresponding internal (to the architecture) signal named **`seconds`** which can be 'read' (e.g. decremented or tested for zero).

I named the input signal **`run_stop_in`** so that the debounced signal used in the architecture could be named **`run_stop`**.

To	Assignment Name	Value	Enab
alarm	Location	PIN_77	Yes
reset_n	Location	PIN_29	Yes
run_stop_in	Location	PIN_2	Yes
seconds_out[3]	Location	PIN_44	Yes
seconds_out[2]	Location	PIN_48	Yes
seconds_out[1]	Location	PIN_50	Yes
seconds_out[0]	Location	PIN_52	Yes
run_stop_in	Weak Pull-Up Resistor	On	Yes
reset_n	Weak Pull-Up Resistor	On	Yes
clk	Location	PIN_12	Yes

Timing Diagram for Rising Edge Detector

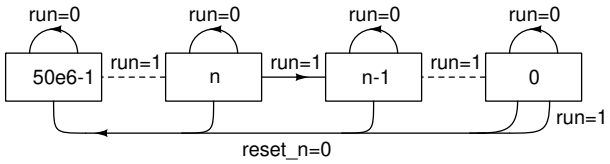
The following diagram shows the sequence of signal values for the **`run_stop`** rising-edge detector.



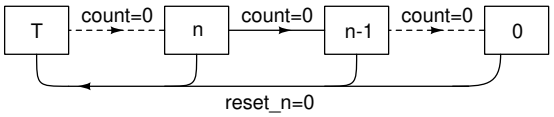
Note that all of the values change at the same time, at the rising edge of the clock. The **`run_stop_in`** signal has been synchronized by the synchronous debouncer.

State Transition Diagrams

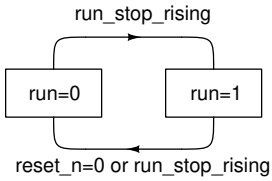
We normally don't consider counters as state machines, but if we did we might draw the state transition diagram for the counter as follows:



where the state label (e.g. *n*) indicates the current **`count`** value. The state transition diagram for the time remaining counter, **`seconds`** would be as follows:



and the state transition diagram for the run/stop state machine would be:



where **`run_stop_rising`** indicates a rising edge on the **`run_stop`** input.

Troubleshooting Hints

It's generally a good idea to design top-down (start with the most general description of your design) but implement bottom-up (implement and test the smallest parts of your design). This is because it's easier to troubleshoot when there are fewer things that could be wrong.

For example, in this lab you could start by implementing the run/stop state machine. To test it you could output the current value of the **`run`** state and verify that it's responding correctly to the different button presses. The you could implement the counter that divides down the clock. You could display the most significant bits of the counter on the LEDs and verify that the counter starts, stops and is reset correctly. Finally, you can implement the time-remaining counter.