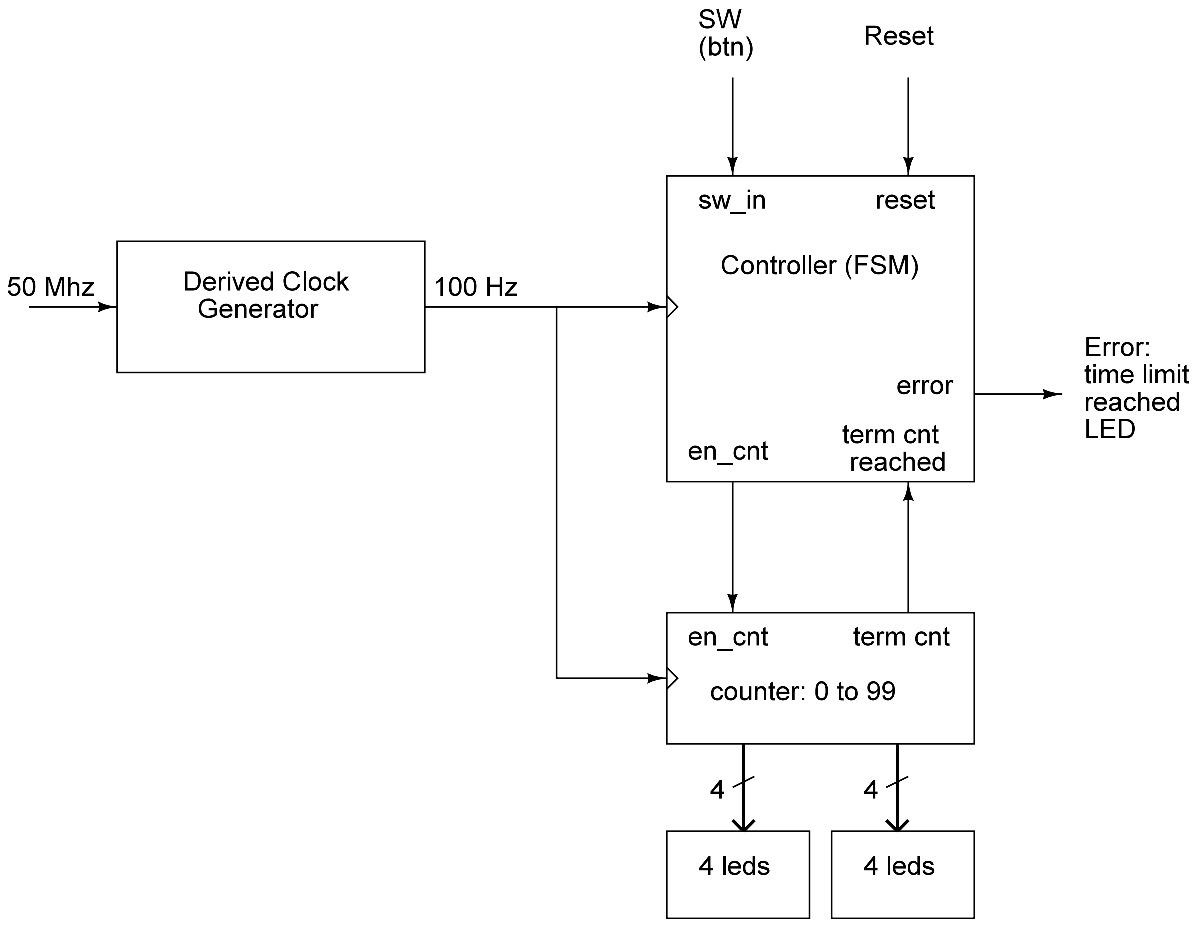
ENGR-433
Notes about lab 2

I have shown the information contained in this note to many of you already, but some may not have seen it. This is information I have on the white board but repeated here for your convenience.

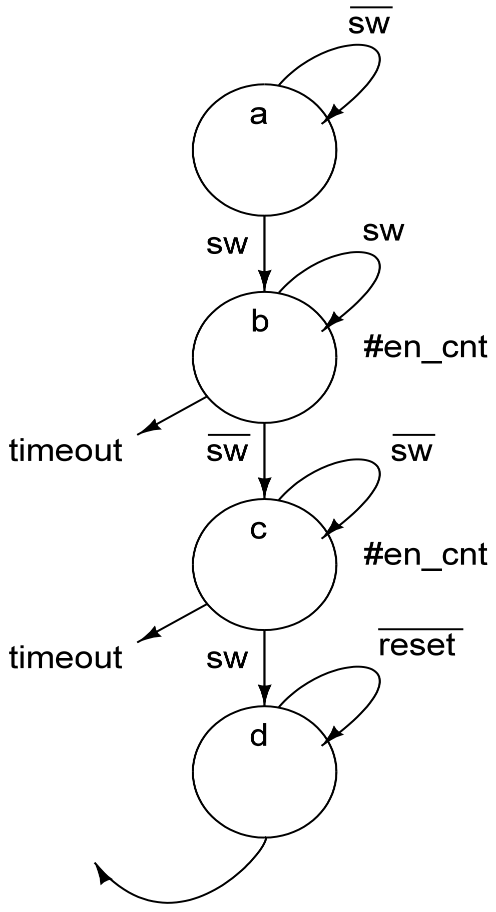
When doing design a first step is to have a good definition of the problem. For this lab, that was given. For digital systems, a next step is often to break the larger problem into smaller functional units and develop a block diagram showing these functional units and signals going between them. The figure below illustrates a block diagram as it might appear for this lab (this is a first draft, you might find details to add).



I recommend that you define details, i.e. function and specifications, for each block. That will lead to further decomposition into either basic components or another block diagram of for lower level function. In this lab, decomposition of the blocks shown above will mean choosing components from the Xilinx library to implement the function of the block.

The controller should be a state machine. By creating a fully documented state diagram you will document the functionality that the state machine needs to have and from the state diagram you can directly create the next state logic. I recommend using one-hot-plus-one state encoding because I think it will be easier to develop the next state logic. However, you can use binary state encoding if you wish. You could draw the block diagram of a state machine to show the decomposition of the controller block into the next-state logic, memory, and output logic if that is helpful for your understanding of the problem.

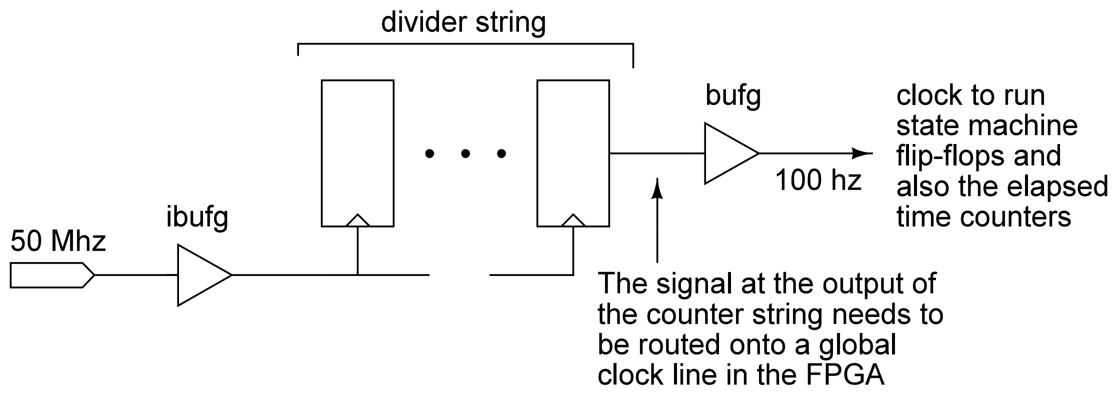
Here is the start of a state diagram that I put on the board:



Please note that this is not the complete state diagram but illustrates how I would detect and react to pressing the button. Keep in mind our objective: to measure the time between the first press of the button and the second press of the button measured in 1/100 s of a second. Thus assume that the state machine memory is clocked at 100hz and every 10 milliseconds a transition to a new state could occur depending of course on the inputs to the state machine.

In state **a** we wait for the button to be pressed. When the button is presses we transition to state **b**. When a person presses a button it most likely will be pressed for more than 10 ms and thus we need to wait and look for it to be un-presses. That is what state **b** is for. Then in state **c** we wait until the button is again pressed. Note that during state **b** and **c** the counter is enabled and is counting time. In both state **b** and **c** it is possible that the count gets up to maximum, i.e. 99, and then we need to transition to what might be termed an error state that will turn on the error LED and wait for reset to be pressed. When reset is pressed we should go back to state **a** and start the sequence over again. Caution: the state diagram is not complete as shown and branching conditions may not be complete. Make sure your branching conditions satisfy the expected sum-to-one and uniqueness requirements.

Another detail that I have pointed out is how to deal with the clock. A signal external to the FPGA that we wish to use as a clock needs to route onto a global clock line which can be used to drive the clock inputs on flip-flops. Use an ibufg rather than an ibuf to make that happen. After being divided down, the signal is not automatically on a global clock line but needs to routed to a global clock line so it can then be connected to the clock inputs of flip-flops and counters. Use a bufg component to do that as shown in the diagram below:


divider string
bufg
clock to run state machine flip-flops and also the elapsed time counters
100 hz
ibufg
50 Mhz
The signal at the output of the counter string needs to be routed onto a global clock line in the FPGA