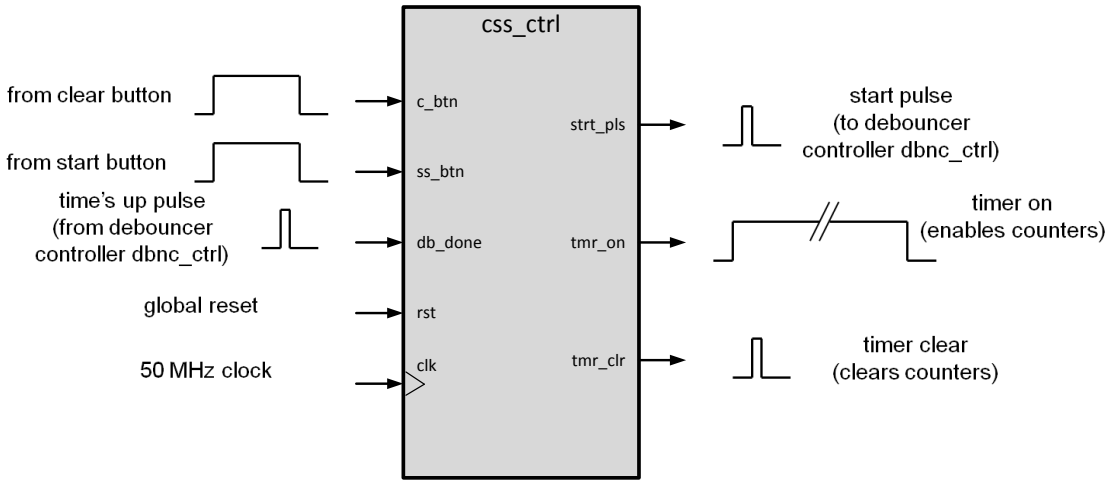
Labs 9 & 10 – Controller Design and Integration

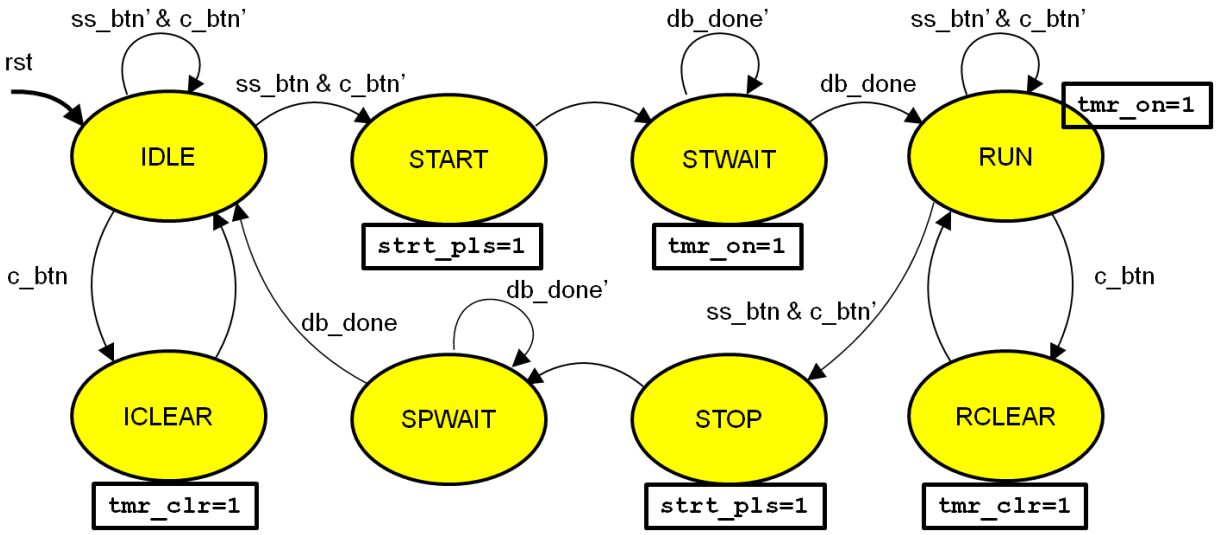
In this lab you will design two new controller blocks. These are the final blocks for the stopwatch design. You will integrate these blocks with the components you have designed up to and including Lab 8. At the end of these labs you should have a working stopwatch.

The exercise can be divided into four parts: **a)** design the Clear/Start/Stop Controller; **b)** design the Debouncer Counter/Controller; **c)** integrate a and b into a Stopwatch Controller; **d)** integrating c into the Stopwatch Core block from Lab 8. The scope of these labs is two weeks of effort (plus homework defined in the lectures). **DO NOT** wait until the second week to start this work. You are responsible for submitting only one lab report for Labs 9 & 10, however this lab report is worth two lab report grades.

Part A: Design of the Clear/Start/Stop Controller

The block diagram below illustrates this module (*css_ctrl*). This contains the Finite State Machine reviewed in the lecture notes and also shown below.



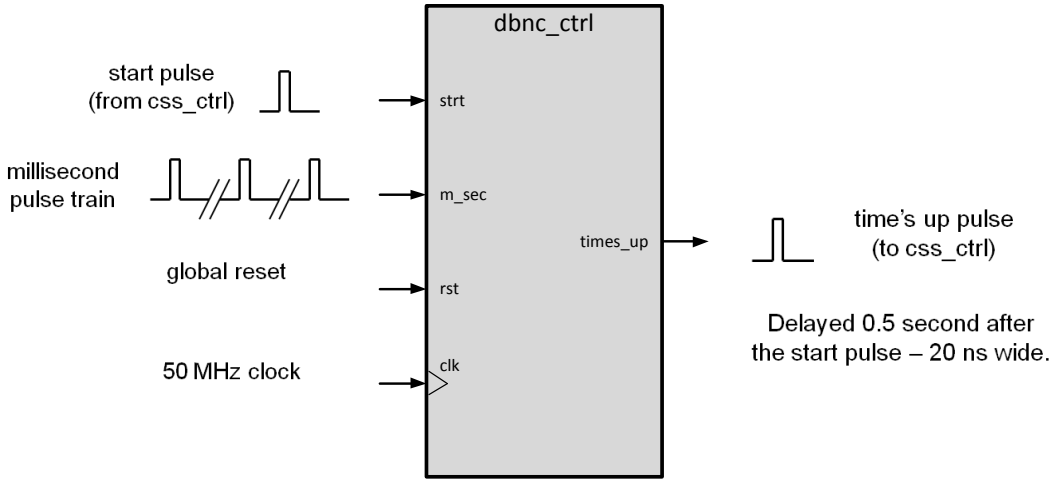


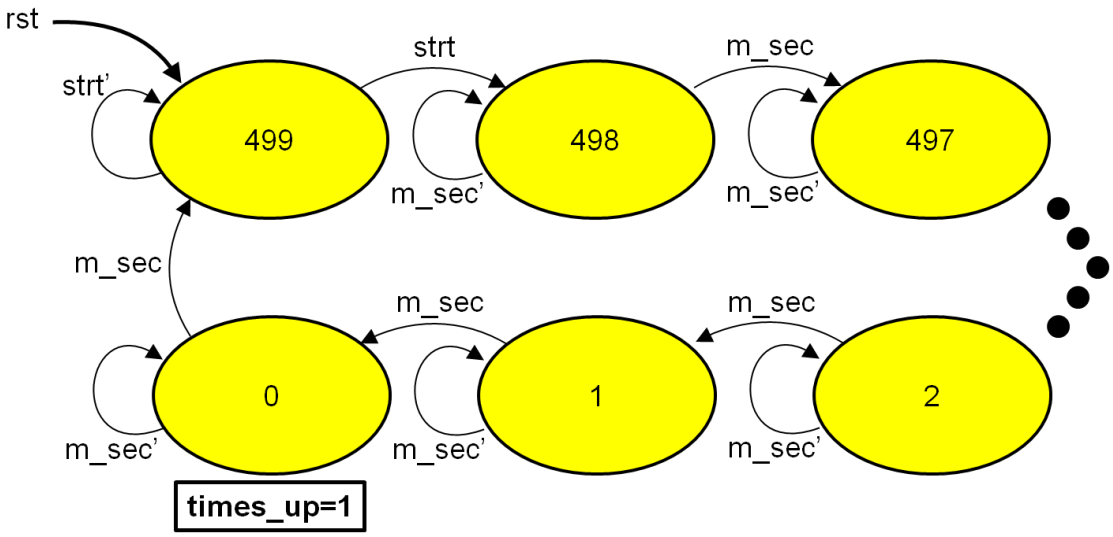
Part A steps:

1. Open the *lab9_10* project in directory lab9_10. It contains supporting files for the *css_ctrl* block, including testbench modules: *tb_css_ctrl.v,* and *tb_css_ctrl.txt*.
2. Create a new *css_ctrl* module using the New Source Wizard (*Project->New Source… Verilog Module*, etc.). Use the *css_ctrl* block diagram above to guide your design in defining the module i/o's. You must name the i/o's **exactly** as illustrated in the block diagram.
3. Follow the steps given in the lecture handout (and on the course website) for designing the controller based on the Finite State Machine above – **Controller Design Process, Step 2: Convert to circuit**.
4. Simulate your design using the testbench: *tb_css_ctrl.v*. Correct any mismatches using the techniques discussed in the lectures.
5. There is no hardware implementation of this module until Part D.

Part B: Design of the Debouncer Counter/Controller

The block diagram below illustrates this module (*dbnc_ctrl*). This contains the Finite State Machine reviewed in the lecture notes and also shown below.





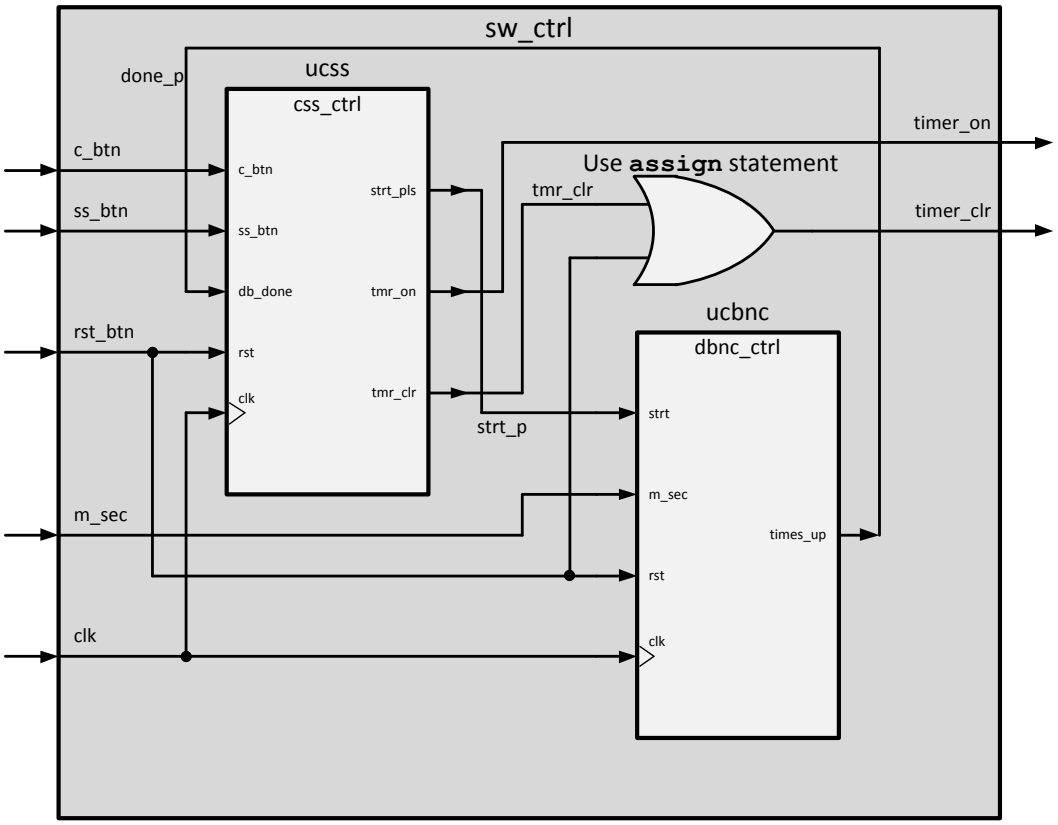
Part B steps:

1. The *lab9_10* project in directory lab9_10 contains supporting files for the *dbnc_ctrl* block, including testbench modules: *tb_dbnc_ctrl.v,* and *tb_dbnc_ctrl.txt*.
2. Create a new *dbnc_ctrl* module using the New Source Wizard (*Project->New Source… Verilog Module*, etc.). Use the *dbnc_ctrl* block diagram above to guide your design in defining the module i/o's. You must name the i/o's **exactly** as illustrated in the block diagram.
3. Follow the steps given in the handout (and on the course website) for designing the controller based on the Finite State Machine above – **Controller Design Process, Step 2: Convert to circuit**.

4. Simulate your design using the testbench: *tb_dbnc_ctrl.v*. Correct any mismatches using the techniques discussed in the lectures.
5. There is no hardware implementation of this module until Part D.

Part C: Integrate Parts A & B into a Stopwatch Controller

The block diagram below illustrates this module (*sw_ctrl*). This contains the instantiated modules from parts A & B. In addition you will have to create an **OR** gate using the **`assign`** statement whose output will clear the timer block either from the system reset signal (*rst_btn*) or the timer clear signal (*tmr_clr*) from the *css_ctrl* block.



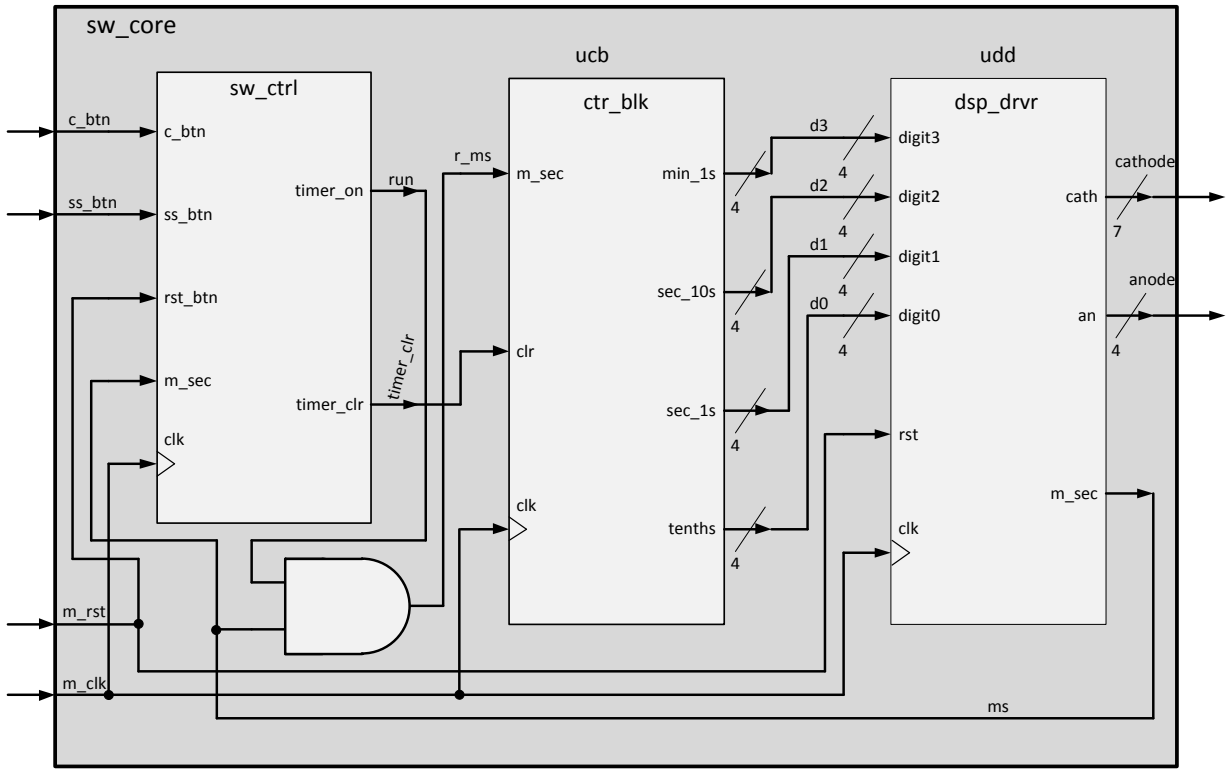
Part C steps:

1. The *lab9_10* project in directory lab9_10 contains supporting files for the *sw_ctrl* block, including testbench modules: *tb_sw_ctrl.v,* and *tb_sw_ctrl.txt*.
2. Create a new *sw_ctrl* module using the New Source Wizard (*Project->New Source… Verilog Module*, etc.). Use the *sw_ctrl* block diagram above to guide your design in defining the module i/o's. You must name the i/o's **exactly** as illustrated in the block diagram.

3. Instantiate the modules using the information in the block diagram. You will need to define some internal *wires* to connect the modules.
4. Use the **`assign`** statement to create an **OR** gate as illustrated in the block diagram.
5. Simulate your design using the testbench: *tb_sw_ctrl.v*. Correct any mismatches using the techniques discussed in the lectures.
6. There is no hardware implementation of this module until Part D.

Part D: Integrate Stopwatch Controller (Part C) into the Stopwatch Core block (Lab 8)

The block diagram below illustrates the module *sw_core*. Compare it carefully to the module (block diagram) you used in Lab 8. You will be copying the module you used from Lab 8 and editing it to add the stopwatch control block. There are also some minor changes in the input signals. The *run* signal (formerly an input signal) now originates from the stopwatch control block.



Part D steps:

1. The *lab9_10* project in directory lab9_10 contains a supporting file for the *sw_core* block, testbench module: *tb_sw_core.v*.

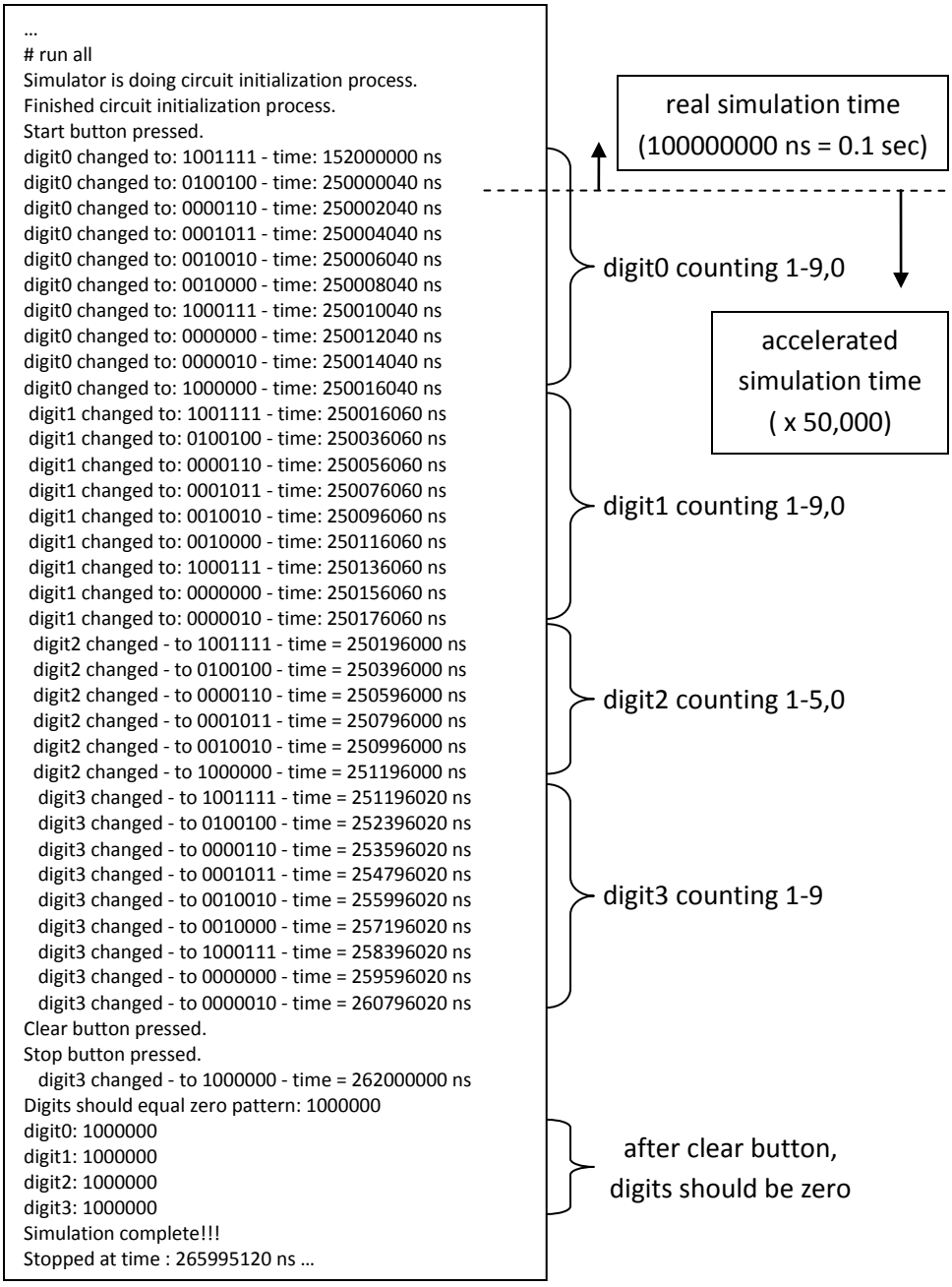
2. Copy design files from your Lab 8 project. Do **not** do a project copy as it will overwrite *tb_sw_core.v* and *top_io_wrapper.v*. Do **not** copy these two files. Use *Project->Add Copy of Source…*. You should copy the following files: ***sw_core.v, ctr_blk.v, dsp_drvr.v, svn_seg_decoder.v, counter_0to9.v, counter_0to5.v*** and ***divideby100.v***.
3. Open *sw_core.v* and add (instantiate) your stopwatch controller from part D, defining and connecting it as illustrated in the stopwatch core block diagram above. You will also need to add two inputs for the start-stop button (*ss_btn*) and clear button (*c_btn*). The master reset (*m_rst*) will be also controlled from another button rather than a switch, but this is taken care of in the *top_io_wrapper* which is supplied to you. You will also have to add some internal wires and re-route the *run* signal. You must define the i/o's **exactly** as illustrated in the block diagram.

   In the *top_io_wrapper* module, the three buttons are each buffered by a pair of D-flip-flops to synchronize the signals with *clk*. This was discussed in one of our past lectures (on metastability).

4. Simulate your design using the testbench wrapper: *tb_sw_core.v*. Similar to Lab 8, the testbench will output a line of text every time digit0 changes as it cycles from 0 to 9, then for digit1, and so on. Once it cycles through the lower digit, the testbench ignores printing subsequent changes to that digit; otherwise the lower digits would overwhelm the output.

   Like the simulation of Lab8, the simulation behavior and characteristics are the same. The first two lines of output will take about 20 seconds each to display – be patient. Note the simulation time displayed in the lower right hand corner. When complete, your simulation output should match the text output below.

   Note the time *difference* in digits changing in the *accelerated* simulation time:
   - digit0 = 2000 ns (equals 0.1 second / 50,000)
   - digit1 = 20000 ns (equals 1 second / 50,000)
   - digit2 = 200000 ns (equals 10 seconds / 50,000)
   - digit3 = 1200000 ns (equals 60 seconds, or 1 minute / 50,000)



5. After you are satisfied that your design passes the simulation testing, implement the design in hardware. Your module *sw_core* should have already automatically plugged into the top level wrapper, *top_io_wrapper*. Synthesize, then download the bit file and test the operation.

Congratulations! You have completed the design of the stopwatch. You should test all of its functionalities – for instance, see what happens when you hold down the start-stop button. It should cycle on-off with a frequency of about one half second (0.4 if truncated).