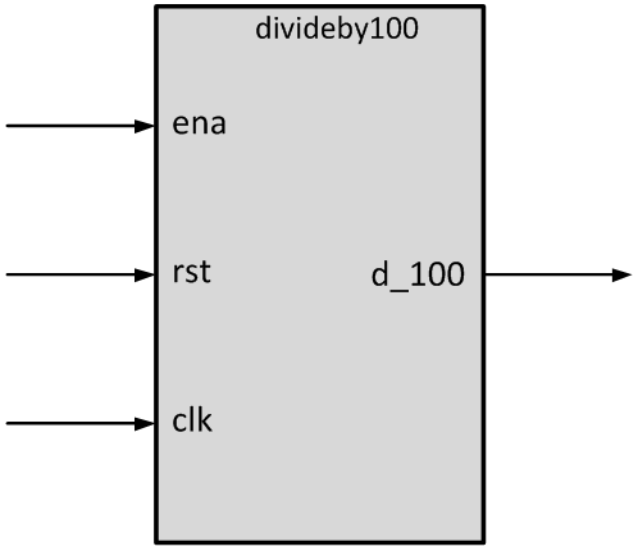
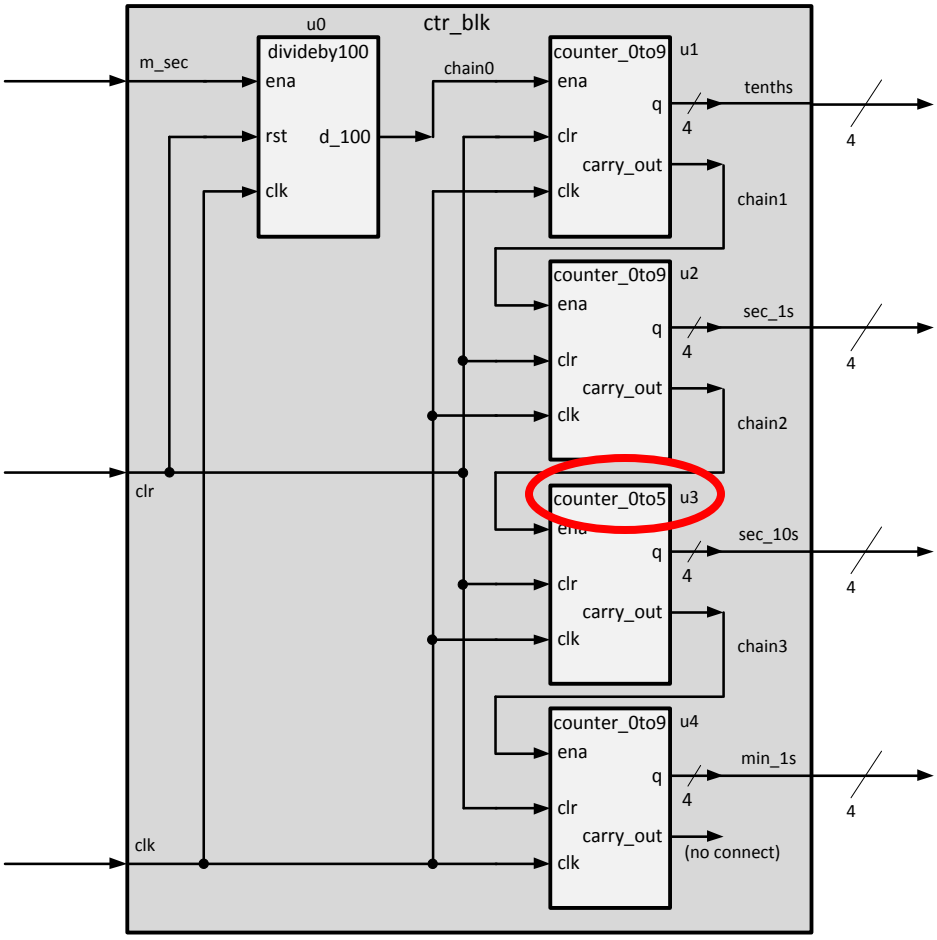
Lab 8 – Counter Integration

In this lab you will use the counters you designed in Lab 6 and integrate them into a new module called a counter block (*ctr_blk*). The counter block will then be integrated into the stopwatch core module from Lab 7 to achieve minimal functionality of a stopwatch/timer.

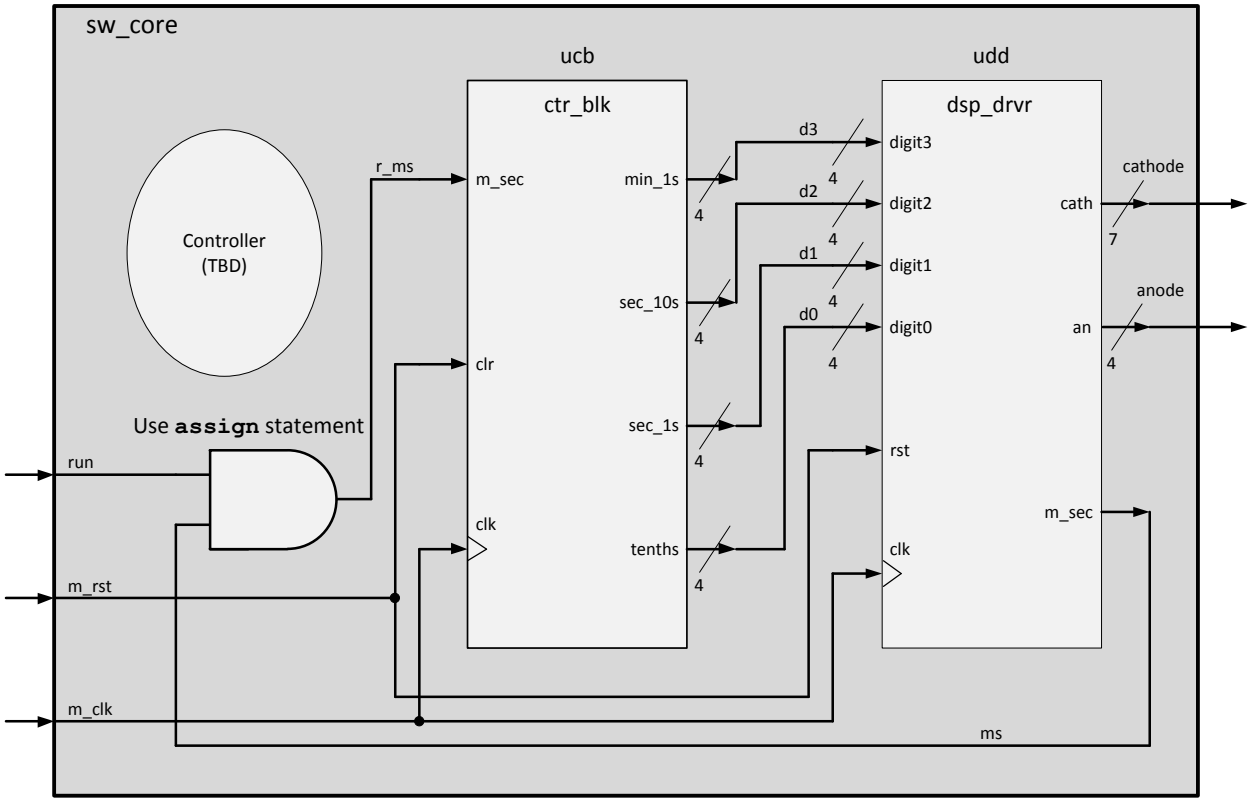
You will also design a new counter, a divide by 100 counter, called *divideby100*. This is necessary to generate a time resolution of $1/10^{th}$ of a second from our 1 millisecond pulse train from the display driver [designed in the course lectures and used in Lab 7]. It will have three input signals: a clock (clk), an enable (ena) and reset (rst), and one output signal: a pulse, one clock period in width, synchronous with the enable signal, but active only every 100^{th} enable (d_100). A good example design of this was done in the class lectures for the display driver. The block diagram for this design is shown below.



A following block diagram illustrates the counter block module (*ctr_blk*). This module has three input signals: *clk*, *m_sec* and *clr*. The internal 50 MHz clock is connected to *clk*. The *m_sec* signal (from the display driver module, Lab 7) will provide a 20 nsec pulse every millisecond. The *divideby100* module (u0) then divides this by 100 to provide a 20 nsec pulse every 0.1 second to the *tenths* counter (u1). The output of this counter enables the next counter in the string (u2), which provides a BCD output representing *seconds*. This chaining is continued, creating a 4 digit counter block, from tenths of a second to a minute. [Note: the 10 second counter in the string is the 0 to 5 counter.]



You will finally update your stopwatch core module (*sw_core*) from your Lab 7 design by adding this counter block to it. This will be the high level module that will eventually contain all of the stopwatch design. A block diagram of the *sw_core* module for this lab is shown below.



The *sw_core* module i/o's are the same as in Lab 7. The input digit0-3 that were hardwired to 1, 3, 5 and 7 in Lab 7 will now be connected to the counter block. You will also add an AND gate to control the millisecond pulses going to the counter block, thus enabling/disabling the counter using a *run* signal. This signal is connected to switch 1. The master reset signal (*m_rst*) is connected to switch 0, just as in Lab 7 (defined in *top_io_wrapper.v*).

Your hardware implementation for this lab will result in a *timer like* display that can be reset, enabled or disabled. When enabled, the timer/stopwatch display will start counting up through time, with a resolution of 0.1 seconds. To see that all conditions are met, (i.e., all of the possible display values are activated) you will need to run the hardware for 10 minutes (0 through 9 minute's digit). To duplicate this with the simulator (which is about 200 times slower than real time for this design), it would take about 200 x 10 minutes, or over 30 hours. And this is a relatively simple design! You will have a testbench that will provide a shortcut for this, explained in the simulation step below. This is another reason for thorough testing of your previous designs, so that we can assume that the lower level components in the design are error free.

Lab steps:

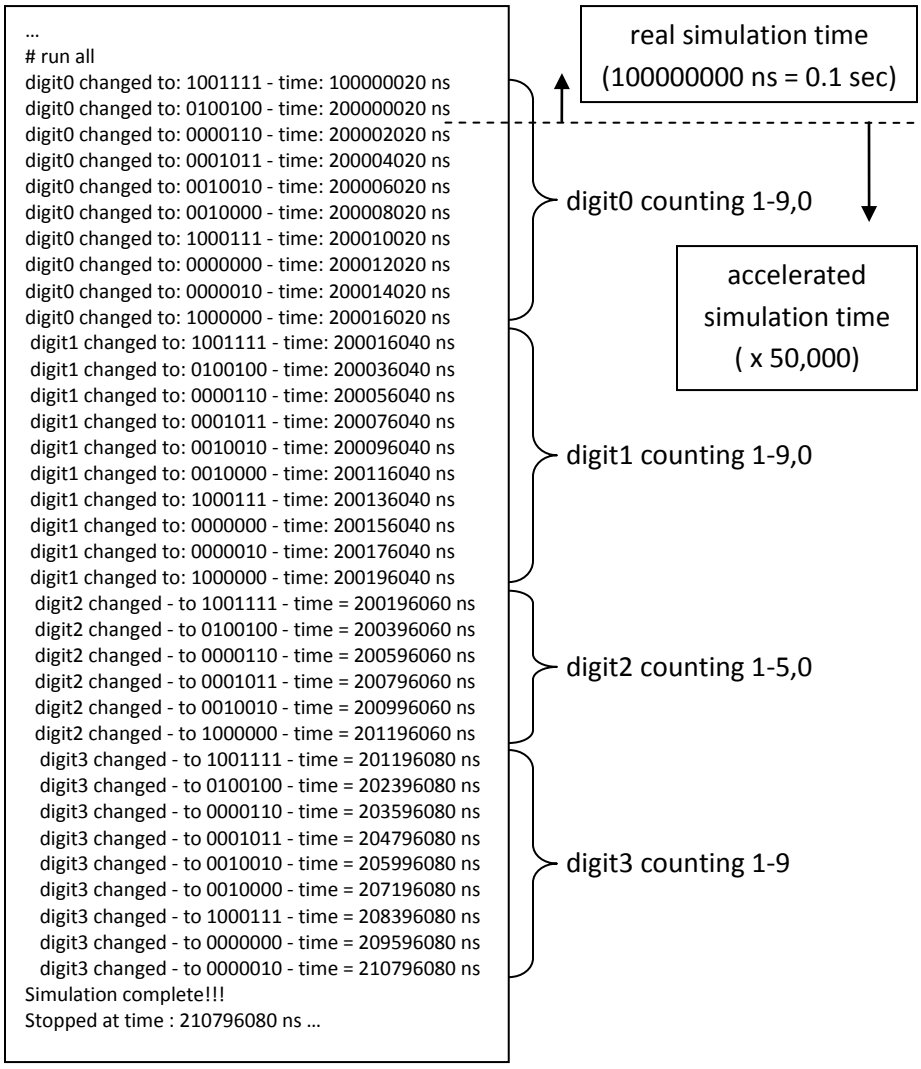
1. Open the *lab8* project in directory lab8. It will contain new testbench modules *tb_sw_core.v*, *tb_divideby100.*v, and *tb_divideby100.txt*.
2. Copy ***design files*** from your Lab 6 and 7 projects. Use *Project->Add Copy of Source…*. Do **not** do a project copy as it will overwrite *tb_sw_core.v*. You should copy the following files: Lab 7 (five files: *sw_core.v, dsp_drvr.v, svn_seg_decoder.v*, *2612_lab7.ucf*, *lab7_top_io_wrapper.v* [*Association* drop down for lab7_top_io_wrapper: **Implementation *not* All**]); Lab 6 (two files: *counter_0to9.v, counter_0to5.v*).
3. Create a new module (**divideby100**) for this project using the module wizard. Refer to its block diagram for appropriate naming of the input and output ports. Enter your design to achieve a *divide by 100* functionality. Use the same techniques and styles as used in your counter design in Lab 6.
4. Highlight the module **tb_divideby100** and simulate your design. Correct any mismatches before continuing.
5. Create a new *ctr_blk* module using the New Source Wizard (*Project->New Source… Verilog Module*, etc.). Use the *ctr_blk* block diagram above to guide your design: a) defining the module i/o's, b) instantiation of the five internal counters, c) defining the internal wires for interconnecting the counter instances and d) connecting the appropriate signals to the module i/o's. You must define the i/o's **exactly** as illustrated in the block diagram.
6. Open *sw_core.v* and add (instantiate) your counter block from step 3, defining and connecting it as illustrated in the stopwatch core block diagram on page 2. You will also need to create an AND gate with a *continuous assignment* (`assign`). This will be replaced as we add controller blocks in future labs.
   When you are finished, note in the hierarchy window and open the branches to see how they match the hierarchy in your design.
7. Simulate your design by highlighting the testbench wrapper: **tb_sw_core**. 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.

   The simulation will take some time as it is running about 200 times slower than the hardware. The testbench has been set up so that for the first 0.2 seconds it is running a true simulation. You will see that the first two lines of output are true in simulation time (digit0 changes every 100,000,000 ns). Then after 0.2 seconds, the testbench forces the internal *m_sec* signal high (a testbench *trick*), which speeds up the simulation by 50,000 times. [by doing this we are bypassing the function of the divide by 50,000 counter]

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. The bit code represents your seven segment decoded digits (inverted seg_out). [e.g., a *one* code is 1111001]

Note the time *difference* in digits changing in the accelerated simulation time:

- digit0 = 2,000 ns (equals 0.1 second / 50,000)
- digit1 = 20,000 ns (equals 1 second / 50,000)
- digit2 = 200,000 ns (equals 10 seconds / 50,000)
- digit3 = 1,200,000 ns (equals 60 seconds, or 1 minute / 50,000)



8. 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.

Can you explain the behavior of the hardware for all four switch conditions of the two switches? Is the behavior what you expected?

Extras:

- Can you see how the signal override works in the testbench file? How would you postpone it for 0.5 seconds rather than 0.2 seconds?