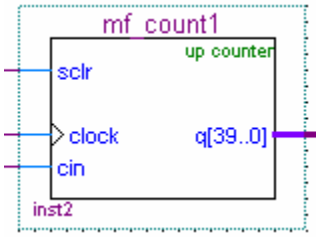
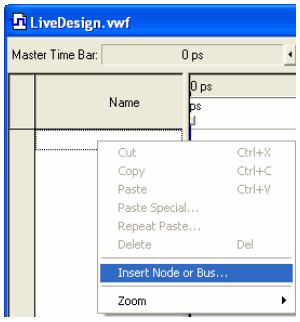
# FPGA Lab 3 - Counters

**Purpose:** In this lab we will build digital counters, which are simple and useful examples of synchronous (clocked) logic. You will also be introduced to the Quartus simulator and learn how to use this tool to check and debug your logic. Given the complexity of digital designs, simulators are essential for getting your designs to function properly. You will rely on the simulator extensively in later labs and will find that a design, once simulated properly, almost always works when tested on the actual hardware. Because the simulator is very powerful, it takes some effort to set it up and define parameters properly. It's not too hard - and certainly much easier than testing a design with only a signal generator and an oscilloscope!

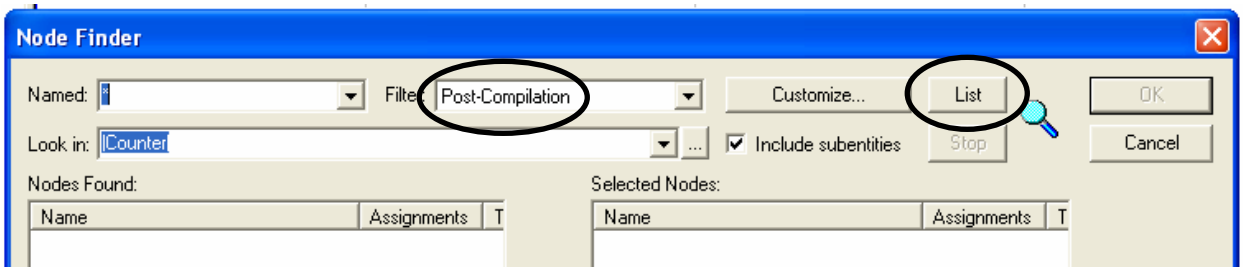
1) **<u>Lab3.1</u> : Binary counter, simulator.** This lab will have you build a simple 20-bit binary counter, and display in hexadecimal the counter output on the 7-segment LEDs. For this counter, you will understand how the synchronous-clear and carry-in input lines work. Copy the Lab1.1 folder and open the project. The 7-segment displays are already entered in the project, as well as the clock input "fclk" and the pushbutton switches "SW_USER[1..0]". These switches are located just below the 7-segment LED's on the board.



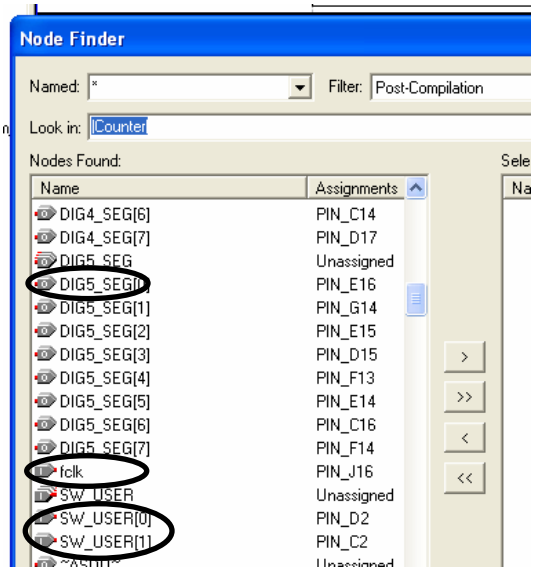

2) Create the counter function with the megawizard (as done for the multiplier last lab) with the path "/installed plug-ins/arithmetic/LPM_COUNTER". Choose a 40 bit counter, with carry-in and synchronous-clear input lines.
3) Connect the inputs of the counter function to the switches and clock, and the output to the 7-segment display. Decide the correct logic for the SW_USER inputs - they are high if not pushed. You want to design the logic to have the counter clear or stop counting when the buttons are pushed. You can use the Quartus help command to get the definition of the carry-in and synchronous clear input lines for this counter function.
4) Compile and test your design. Redefine your bus inputs to the 7-segment modules so that the rightmost display can be seen to count. ***From the counting rate of the displays, determine the clock frequency of* fclk.**
5) **How to use the Quartus simulator:** In this exercise we will want to view the counting of the flip-flops at the counter output, and demonstrate with the simulator that the clear and carry-in functions are functioning properly.




6) First, create the test inputs for the simulator with a vector waveform file "LiveDesign.vwf". Click on "File/New", then tab to "other files" and choose "vector waveform file". Save this file as "LiveDesign.vwf" by clicking on "File/Save As", then entering "LiveDesign" as the file name.
7) Next, specify the pins and nodes you want to monitor in the simulation. In the .vwf file, right click in the column under "Name" and choose "insert node or bus". In this dialog box, click the "Node Finder…" button. Choose as the "Filter" the "post-compilation" pull down, and then click the button "List". In the left column you have a list of possible nodes that you can choose as inputs and outputs. Notice that you have input (I) pins, output (O) pins, combinatorial/gate nodes (C), and register/flip-flops outputs (R); also note that buses are represented as multiple symbols.



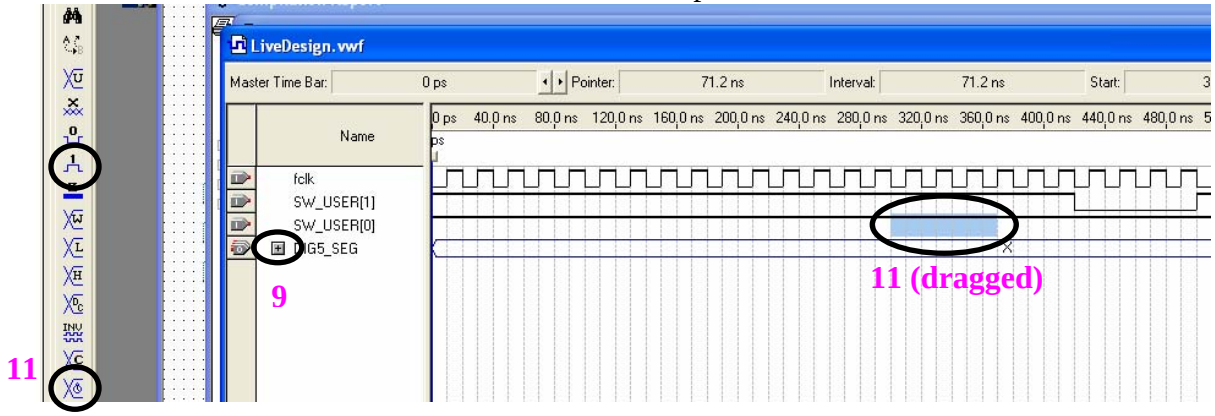

8) Double click on the nodes you want to simulate and they will move to the right hand column. Choose the inputs “fclk”, “SW_USER[1]”, and “SW_USER[0]”; and the output bus “DIG5_SEG”. We have chosen the inputs as individual lines, not buses, to have separate control of the two inputs. We have chosen the output to be a bus because we want to look the entire bused output. Click “ok” twice to close the node finder window and add the nodes to the .vwf file.




9) To ease the interpretation of bus signals, the waveform entries can have their values displayed in more convenient ways. Push the “+” box to display the individual bits of “DIG5_SEG”. To change number format, right click on “DIG5_SEG” and choose “properties”. Under the pull-down menu “Radix” one can change the display of the bus as binary, hexadecimal, octal, signed decimal (via 2’s-complement encoding), or unsigned decimal. Choose hexadecimal for this output.

10) Now define the time axis. First, set the end time by choosing “edit/endtime..”, and entering the end time value of 10 us. Click “ok”. Next, you can zoom in and out of the time axis by right clicking in the center of this window, and choosing “zoom…”. In addition, the bottom scroll bar and mouse scroll can be used to scroll left and right in time.

11) Now you will define the inputs. First, click on the “fclk” symbol. To define it as a clock input, click on the stopwatch symbol on the left hand bar, then enter 20 ns for the period of the clock (fclk runs at 50 MHz). You should now see a square wave in the waveform to indicate this clock. Change the zoom so that you see times from approximately 0 to 720 ns. Next, enter the SW_USER[1] and SW_USER[0] inputs. Click on their name, then on the “1”



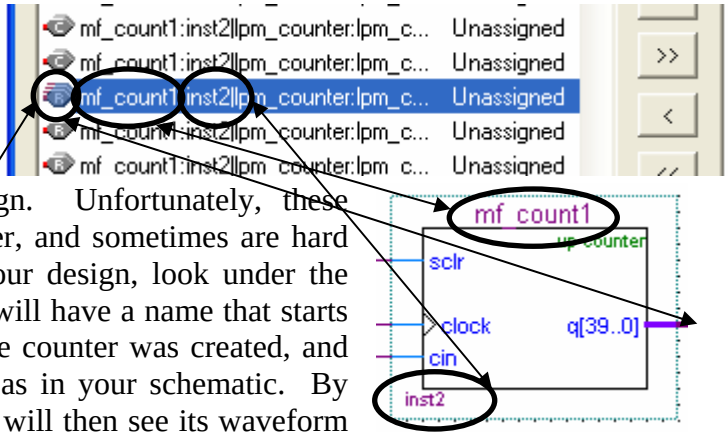
symbol to set their value to 1 for all time. Next, right click and drag over one of these waveforms to select a span of time ("dragged" above fig.). This selected time span can be set to zero by then clicking the "0" symbol. Note that the other useful input functions are invert, which inverts a selected node, and 'c',which increments a bus value at each clock cycle.

12) Save the .vwf file, and then run the simulation by selecting "processing/start simulation". Upon completion, a simulation output file will show the simulated input and output waveforms. You should see the output "DIG5_SEG" changing with time if it is connected to the lowest bits of the counter in your design.

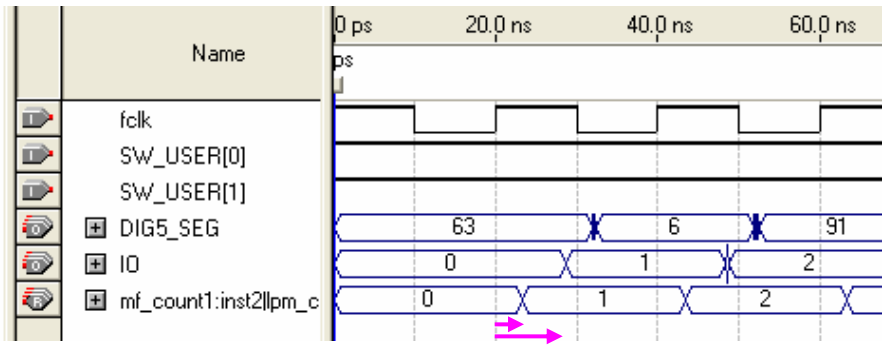
13) The output "DIG5_SEG" is hard to understand because it is coded to display the 7-segment LEDs. It would be much easier to check for proper counter operation by directly looking at the counter output bits. It is possible to look at logic within your design, but unfortunately the Quartus software is a bit clunky for this task. There are two ways to do this.

14) You may look at <u>any</u> logic state in your design by temporarily attaching that wire to an output port, then looking at the output port with the simulator, as done previously. In our LiveDesign board, there are 36 unused input/output lines named "IO[35..0]" that can be used for this function. First, an output bus is created in the schematic by choosing the "device tool" and typing "output" in the name field. After insertion, clicking on "pin_name" allows it to be changed to IO[35..0]. Now connect this output bus to the least significant bits of your counter. Recompile the design, add the output pin name to the .vwf file using the node finder, and rerun the simulation. This output bus will now increments by one at each clock cycle.

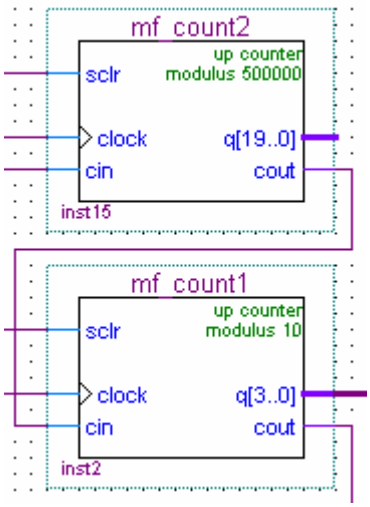
15) It is also possible to directly look at the output of registers/flip-flops within your design. Without using an output pin, registers are the <u>only</u> signals you have access to within your design. Unfortunately, these registers are named by the computer, and sometimes are hard to find and decode by name. In our design, look under the node finder for the <u>register bus</u>. It will have a name that starts with the name you gave it when the counter was created, and has an "instance" that is the same as in your schematic. By inserting this into the .vwf file, you will then see its waveform in the simulator output.




16) Carefully note the waveforms of the counter outputs. The Quartus simulator has been carefully programmed to account for all the various delays internal to the FPGA chip, and you see these in the simulations. The register output is slightly delayed with respect to the clock and accounts for the small output delay of the registers. The output pins change at a later time compared to the registers and accounts for the wire delays between the register and the output pins.



17) **Lab 3.2 : Binary-coded decimal counter.** Copy your entire folder for lab3.1 to a new folder named lab3.2. In this lab you will design a counter that displays the count in base-10 (binary-coded decimal) numbers. Make the rightmost digit of the display count in 1/100 of a second, and display a decimal point after the first two digits.




18) Counting by 10 may be accomplished by using single counting modules 4 bits wide, but with a modulus-10 count. A 4-bit binary counter gives a carry-out signal when the counter has value 15=H"F"; a modulus-10 counter gives a carry-out signal at value 9. Connecting together carry-out and carry-in lines of several modules will give a counter with multiple digits that count in binary coded decimal.

19) **Lab 3.3 : Counter from elementary gates.** Make a 4 bit counter that counts once a second. Construct the counter using only D flip-flops and elementary gates (ie. ANDs and XORs) according to the design discussed in class and in the homework. Use a megafunction counter to generate one carry-in pulse per second.