Lab #5: Counters

Physics 127BL Winter 2024

Lab report due **Thursday, February 15, at 11:55 P.M.**

Please read the lab report and homework guidelines handout on the course web page.

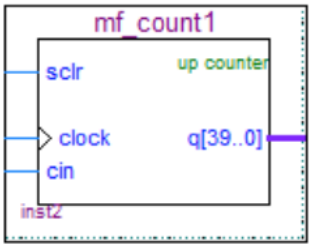
Introduction

In this lab, you will build digital counters, which are simple and useful examples of synchronous (clocked) logic. You will also use the Quartus simulator to check and debug your logic. Given the complexity of digital systems, a simulator is 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 correctly on the actual hardware. This is certainly much easier than testing a design with a signal generator and an oscilloscope!

1 Binary counter and simulator.

In this part you will build a simple 40-bit binary counter and display in hexadecimal the counter output on the 7-segment LEDs. You will learn how the synchronous-clear and carry-in input lines work. Copy the `lab5` directory and open the project. The 7-segment displays are already entered in the project, as well as the clock input `CLOCK_50` and the pushbutton switches `KEY[1..0]`. These switches are located to the right of the switches.

1. Create the counter function with the megawizard (as done for the multiplier in the previous FPGA lab) with the menu `Basic Functions→Arithmetic→LPM_COUNTER`. Choose a 40 bit counter, with carry-in and synchronous-clear input lines.



2. 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 KEY 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.

3. Compile and test your design. Depending on how you split the 40-bit bus to the displays, they may all be counting too fast to see. Redefine your bus inputs to the 7-segment modules

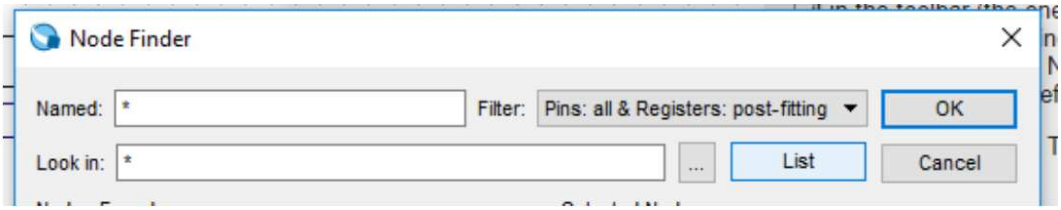
so that the HEX2 or HEX3 display can be seen to count. From the counting rate of the displays, determine the clock frequency.

Using the Simulator

Now 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 working properly.

1. First, create the test inputs for the simulator with a vector waveform file LiveDesign.vwf. Choose File→New, then under "Verification/Debugging Files" choose "University Program VWF". Save this file as LiveDesign.vwf by choosing File→Save As, then entering LiveDesign as the file name.

2. 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 "Pins: all & Registers: post-fitting", 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 pins (IN), output pins (OUT), and register/flip-flop outputs (R); also note that buses are represented as multiple symbols.



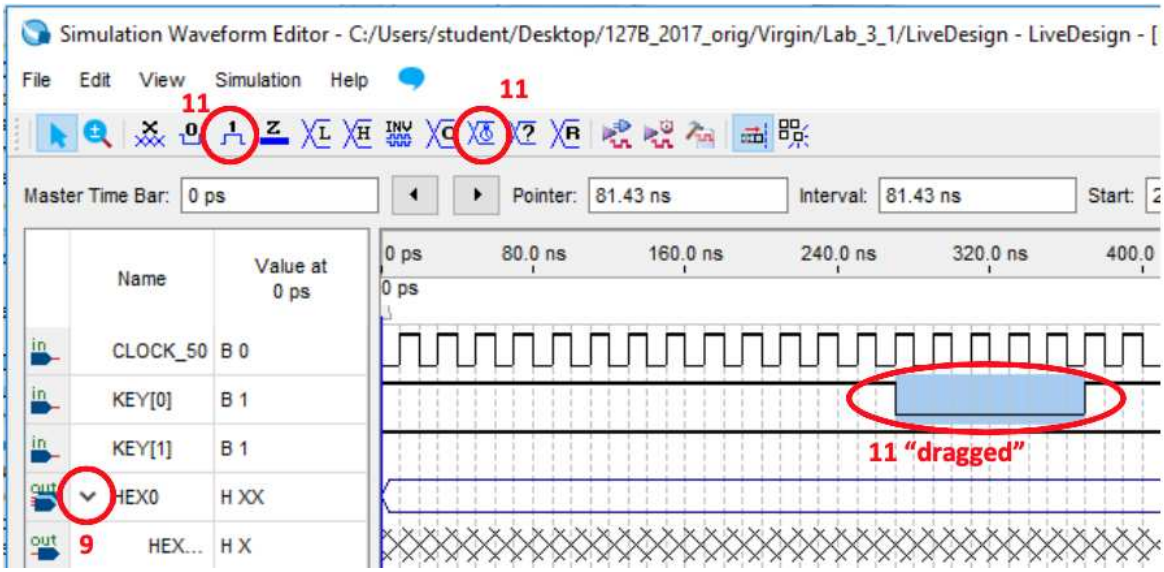
3. Double click on the nodes you want to simulate and they will move to the right hand column. Choose the inputs CLOCK_50, KEY[1], and KEY[0]; and the output bus HEX0. We have chosen the inputs as individual lines, not buses, to have separate control of the two inputs. We have chosen the output to be a bus because we want to look at the entire bused output. Click OK twice to close the node finder window and add the nodes to the .vwf file.

4. 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 HEX0. To change number format, right click on HEX0. 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.

5. Now, define the time axis. First, set the end time by choosing Edit→Set Endtime and entering the end time value of 10 μs. Click OK. Next, you can zoom in and out of the time axis by holding <CTRL> and scrolling. In addition, the bottom scroll bar can be used to scroll left and right in time.

6. You next need to define the inputs. First, click on the CLOCK_50 symbol. To define it as a clock input, click on the stopwatch symbol on the uppermost toolbar, then enter 20 ns for the period of the clock (our clock runs at 50 MHz, which you should have measured

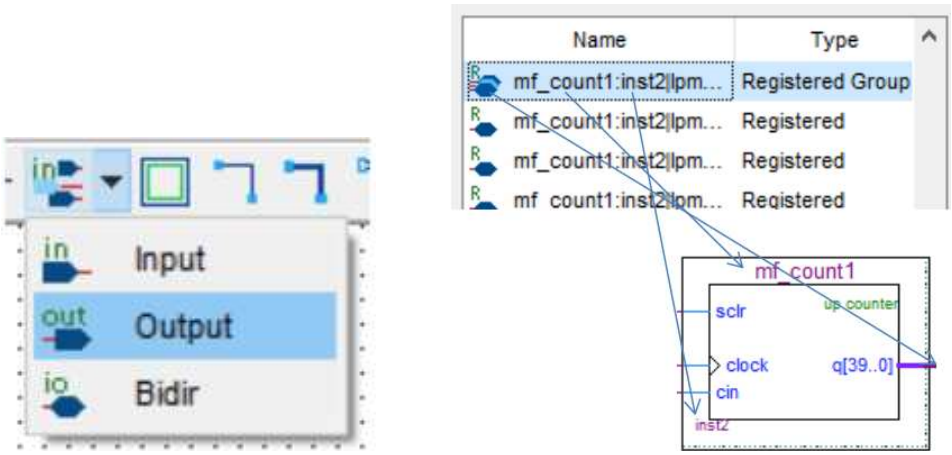
earlier). You should now see a square wave in the waveform to indicate this clock. Change the zoom so that you see times from approximately 0 to 720 ns. Next, enter the `KEY[1]` and `KEY[0]` inputs. Click on their name, then on the `1` symbol to set their value to 1 for all time. Next, right click and drag over one of these waveforms to select a span of time (denoted as "dragged" in the figure below). 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.



7. Save the `.vwf` file, then run the simulation by selecting `Simulation→Run Functional Simulation` in the waveform window. Upon completion (this may take a minute), an identical read-only simulation output window will show the simulated input and output waveforms. You should see the output `HEX0` changing with time if it is connected to the lowest bits of the counter in your design.

8. The output `HEX0` 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 it:

   - You may look at any 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. First, an output bus is created in the schematic by choosing the "pin tool" in the top toolbar and choosing "output" (see figure below, left). After insertion, change the `pin_name` to something like `OUT[39..0]` to define the size, then reference it to your `Q` bus like normal. Recompile the design, add the output pin name to the `.vwf` file using the node finder, change the "Radix" to hexadecimal, and rerun the simulation. This output bus will now increment by one at each clock cycle.

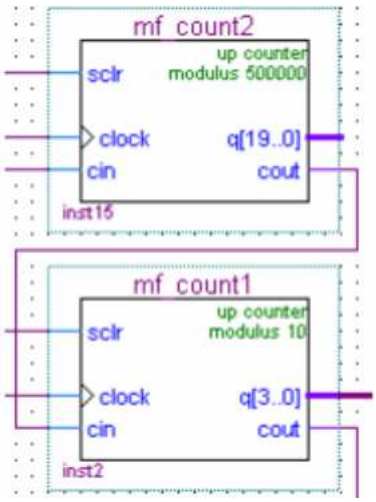


- It is also possible to look directly at the output of registers and/or flip-flops within your design. Without using an output pin, registers are the only signals you have access to within your design. Unfortunately, these registers are named by the computer, and sometimes are hard to find and decode by name. In our design, look under the node finder for the register bus. 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 (see figure above, right). The bus is of type "Registered Group". By inserting this into the `.vwf` file, you will then see its waveform in the simulator output.

9. Now, try running a "Timing Simulation", instead of a "Functional Simulation". Carefully note the waveforms of the counter outputs. The Quartus simulator has been 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, which accounts for the wire delays between the register and the output pins.

2 Binary-coded decimal counter

Copy your entire `lab5` directory to a new directory named `lab5_part2`. In this part, you will design a counter that displays the count in base-10 (binary-coded decimal) numbers. Make `HEX2` count every 1/100 of a second, and use the `HEX3-4` gap to signify the decimal. 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 = `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 (see figure below).



3 Counter from elementary gates

Make a 4-bit counter that counts once per second. Construct the counter using only `D` flip-flops and elementary gates (`AND`s and `OR`s). Use a megafunction counter to generate one carry-in pulse per second.