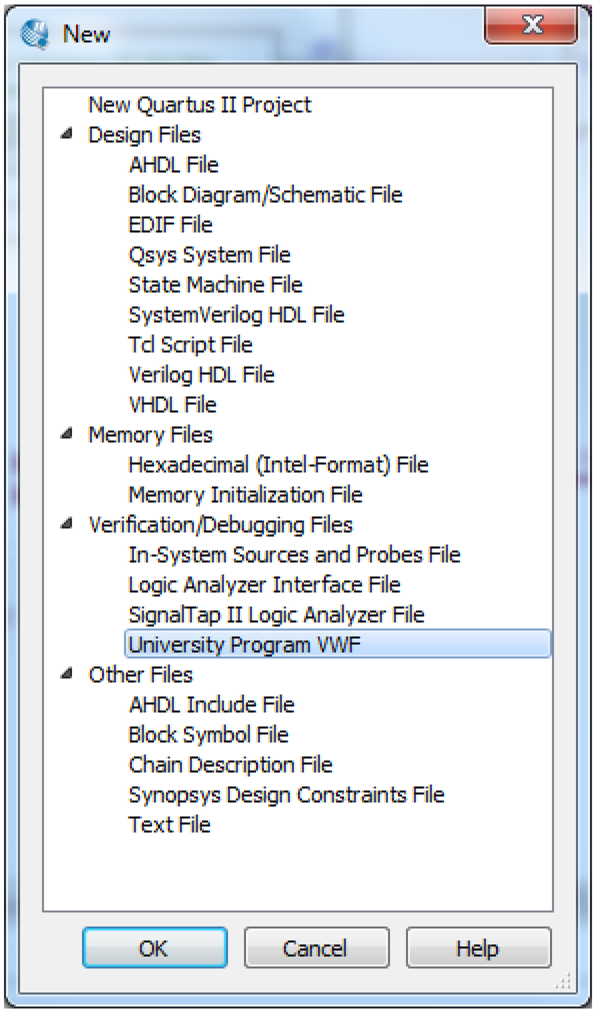
EEEE-220-Lab1

1) From mycourses download on your desktop the DE0_NanoUser_Manual.pdf and this file. In this part of the lab you will get familiar with the FPGA prototyping board that you will use throughout the semester.
2) Start by reading chapter 1. You may encounter terms that are unknown to you. Don't worry! We will explain all of these as we progress through the labs.
3) Continue reading chapters 2 and 3. As per section 2.3, we will always power the board via the USB port, i.e. no external power source or battery is necessary. As per section 3.1, we will always configure the FPGA via the USB cable/port using the on board USB-Blaster circuit, Fig.3-2. If you want, the final lab projects you can also program into the serial configuration device during labs 11 and/or 12. This will allow you to power up and down the board without the need to re-program the board.
4) In section 3.2 you find out that the two push-buttons are "debounced". Metal-to-metal contacts generate several oscillations when closed. These are interpreted by a logic input as multiple bounces between logic high and low, as shown in the upper waveform of Fig. 3-4. As you will learn later in "Electronics II", a Schmitt Trigger logic gate is able to "filter" such bouncing so that the signal changes are interpreted correctly, as shown in the lower waveform of Fig. 3-4.
5) The pushbuttons and DIP switches are called input peripherals. These provide the means for the user to enter data (information) in digital form. The eight LEDs are called output peripherals. These provide the means to output data (information) in digital form. The circuits and systems you will prototype in this lab sequence will interface with these input/output peripherals. We will not use the SDRAM, Serial EEPROM, expansion headers, A/D Converter, and accelerometer in this course, so you can just browse through sections 3.3 – 3.7.
6) As you find out in section 3.8, the board includes a 50 MHz oscillator. Using a Phased-Locked-Loop circuit, you can generate various clock signals to drive your circuits. Remember: we design clock synchronous digital circuits! You will instantiate a PLL later in this lab, but a more formal coverage of the PLL will follow in a later lecture.
7) For demonstration purposes, we will always power the board via the USB port. Section 3.9 shows a possibility to power the board from batteries. As you notice in Fig.3-16, the onboard regulators convert the input power supply voltage to a several other values. Due to a variety of integrated circuits (ICs) in use today, the need to generate many different power supply voltages has become quite commonplace. Fortunately, the DC-to-DC converters used for this purpose are highly integrated.
8) Continue to chapter 4.
    a. From the CD included in the kit, copy to the desktop the directory: DE0_Nano_ControlPanel_V1.0.3.
    b. Next, connect the board via the USB cable to your desktop computer. If the on board configuration device contains the original design, you may see the LEDs "breathing". Otherwise, it may implement and run the last configuration downloaded to the configuration device.

   c. Enter the directory DE0_Nano_ControlPanel_V1.0.3 on your desktop and launch DE0_Nano_ControlPanel.exe. It will take a few moments to download a new configuration to the FPGA. If everything goes well, it will say Connected in the lower left corner of the window.
   d. Read and perform the actions described in sections 4.1, 4.2, and 4.3. If you want you can also play with the accelerometer, as described in section 4.5.
   e. As you find out in section 4.7, the application instantiates an embedded processor called Nios II and associated peripherals into the FPGA. Then, it runs a software program to control the various peripherals as instructed by the user through the GUI (Graphical User Interface).

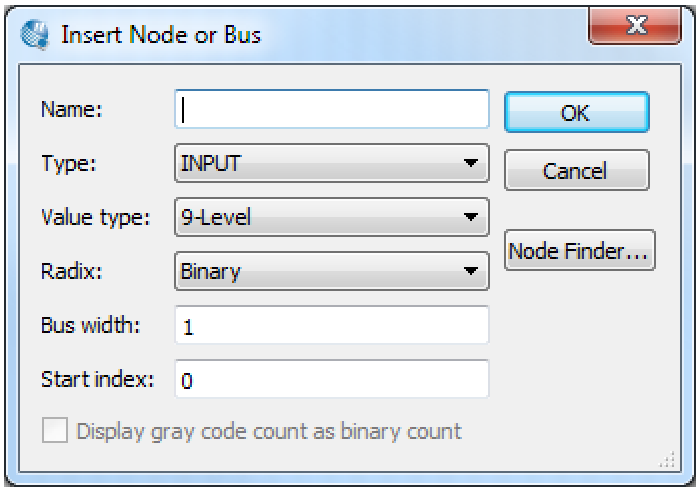
9) At this point you should be fairly familiar with the structure and features of the DE0-Nano board. Skip chapter 5 and proceed to chapter 6.
10) During the second major part of this lab you will learn how to create an FPGA project using the Altera Quartus II CAD environment, capture a design, compile it, simulate it to verify its correctness, and finally physically implement it by downloading it to the FPGA.
11) Read section 6.1. The design flow described here is similar to the one we have discussed in lecture 2 and 5. Ignore the statement: “except for simulation”. We will perform simulation too.
12) Skip section 6.2. The software environment and USB-Blaster driver have been installed.
13) Continue on to section 6.3. Here you find out what the circuit actually does. Don’t worry: you don’t have to know Verilog yet! The code will be provided.
14) The actual work starts in section 6.4. Start Altera Quartus II and follow the tutorial up to but excluding 6.9. You will return to 6.9 after we verify the functionality of the design in simulation.
15) Additional notes along the way:
   a. In step 3a create the project in a directory on the Desktop called fml_EEEE-220_F13/fml_first_fpga. Further, call your first project fml_first_fpga. Review the lab policy about the mandatory requirement to use fml for all your files and folders in this lab and class. Fml = first, middle, and last name initials.
   b. In step 4 of the Verilog HDL counter you can copy/paste the text from the manual. Make sure that the final code looks exactly as in Fig.6-15.
   c. For all new and custom components you create with the MegaWizard, use the prefix fml_.
   d. On page 70: you’ll probably have to run “Start Analysis & Elaboration” in the Assignments > Pin Planner. Then you’ll see the pins.
16) At this point, it is assumed that you have successfully compiled your project, i.e. have arrived to section 6.9. Before moving on to section 6.9 we want to simulate the design.
   a. File > New > and select “University Program VWF”.


New

New Quartus II Project
- Design Files
  - AHDL File
  - Block Diagram/Schematic File
  - EDIF File
  - Qsys System File
  - State Machine File
  - SystemVerilog HDL File
  - Tcl Script File
  - Verilog HDL File
  - VHDL File
- Memory Files
  - Hexadecimal (Intel-Format) File
  - Memory Initialization File
- Verification/Debugging Files
  - In-System Sources and Probes File
  - Logic Analyzer Interface File
  - SignalTap II Logic Analyzer File
  - University Program VWF
- Other Files
  - AHDL Include File
  - Block Symbol File
  - Chain Description File
  - Synopsys Design Constraints File
  - Text File

b. File > Save As > fml_first_fpga.vwf

c. Edit > Insert > Insert Node or Bus

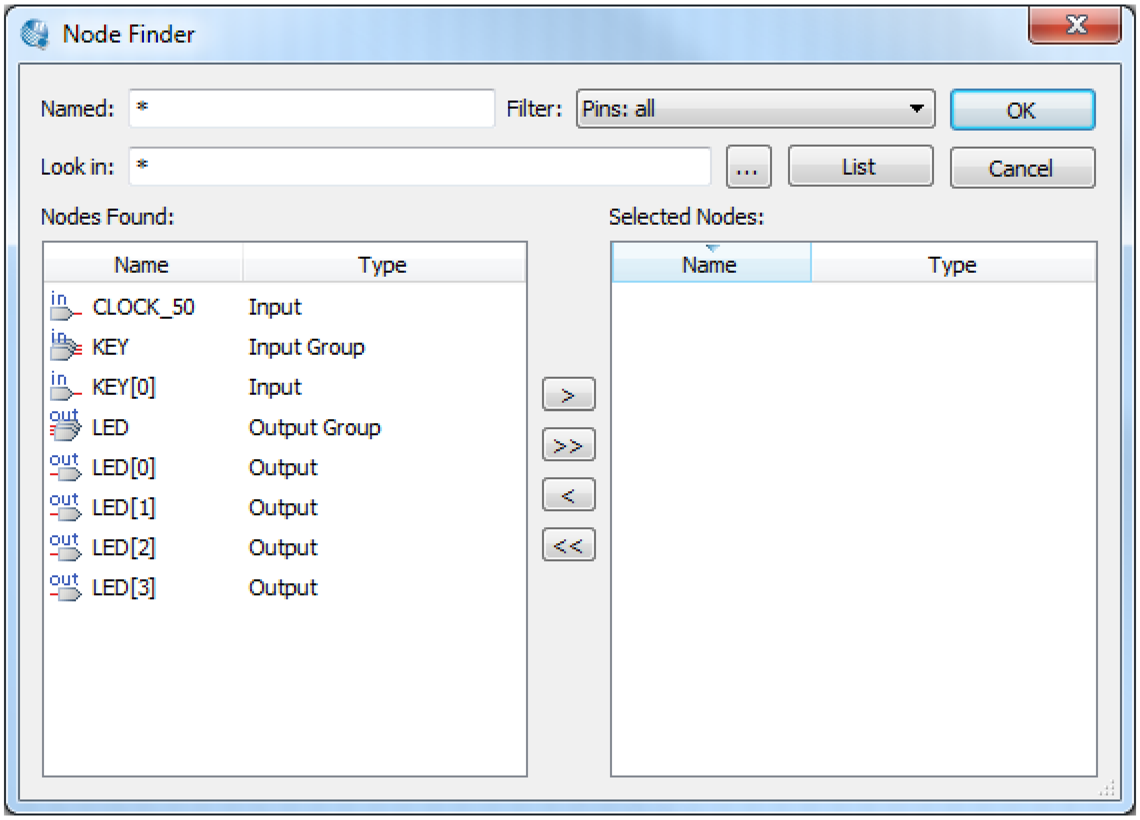

Insert Node or Bus

Name:	
Type:	INPUT
Value type:	9-Level
Radix:	Binary
Bus width:	1
Start index:	0

[ ] Display gray code count as binary count

d. In the window above click Node Finder.

e. In the window below click List. The “Nodes Found” lists all pins found in the top level design file.



Named: *
Filter: Pins: all
Look in: *

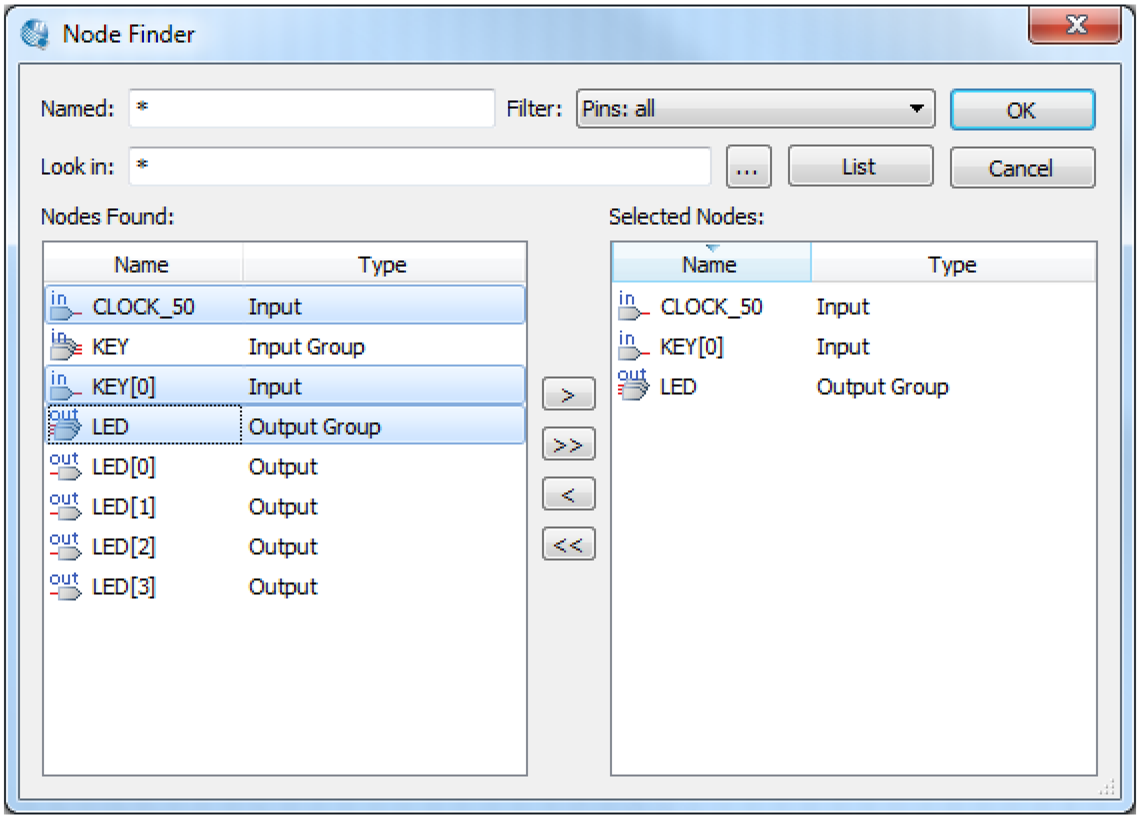
Nodes Found:

Name	Type
CLOCK_50	Input
KEY	Input Group
KEY[0]	Input
LED	Output Group
LED[0]	Output
LED[1]	Output
LED[2]	Output
LED[3]	Output

Selected Nodes:

Name	Type

f. Now, transfer to “Selected Nodes” the ones listed below:



Named: *

Filter: Pins: all

Look in: *

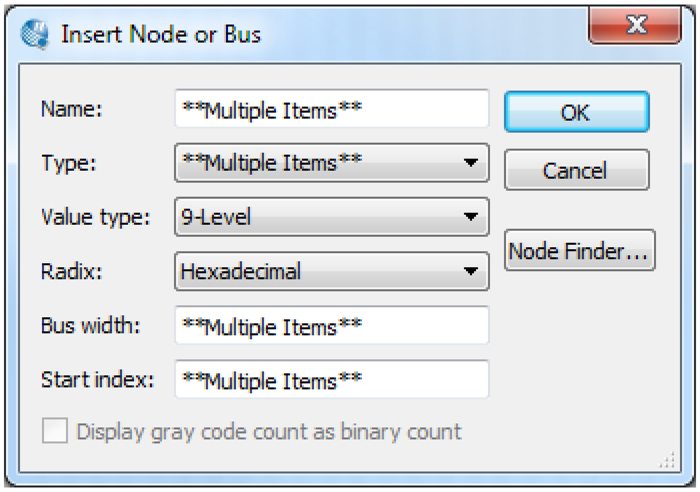
Nodes Found:

Name	Type
CLOCK_50	Input
KEY	Input Group
KEY[0]	Input
LED	Output Group
LED[0]	Output
LED[1]	Output
LED[2]	Output
LED[3]	Output

Selected Nodes:

Name	Type
CLOCK_50	Input
KEY[0]	Input
LED	Output Group

g. Click OK. Back to the window below, select the Radix Hexadecimal. This will show the hexadecimal value for busses, and the binary value for single pins or wires.



Name: **Multiple Items**

Type: **Multiple Items**

Value type: 9-Level

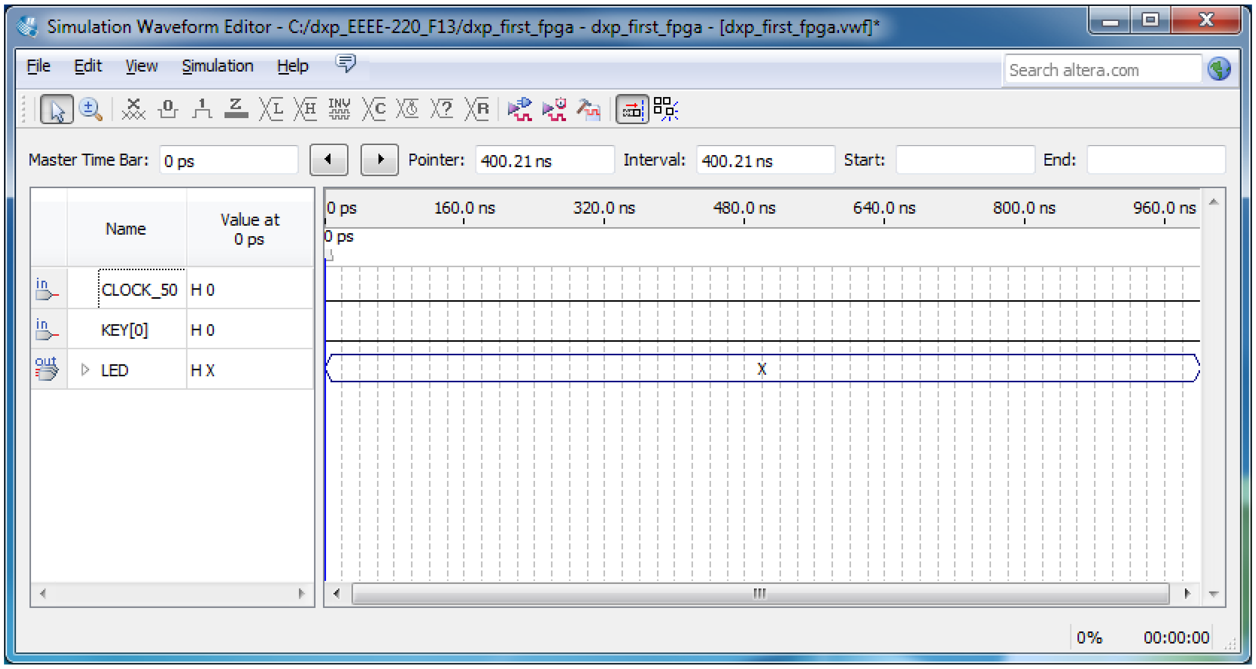
Radix: Hexadecimal

Bus width: **Multiple Items**

Start index: **Multiple Items**

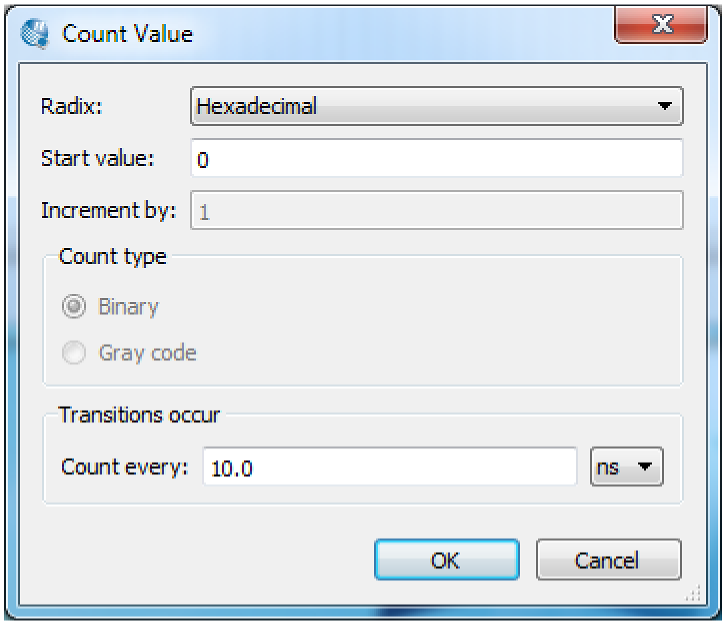
Display gray code count as binary count

h. Click OK. Your simulation waveform editor window should look like this now:



i. The waveform editor lets us edit the shape of the waveform for input pins or signals. After the simulation it displays these along with the values of all selected output pins or signals. This allows us to verify that the function implemented by the circuit is correct. A simulator is an invaluable tool in the process of debugging, which we will cover in more detail later.

j. Select the "CLOCK_50" pin name in the LHS column. Next, click on the C in the upper toolbar. The window below pops up.


Count Value

Radix: Hexadecimal

Start value: 0

Increment by: 1

Count type

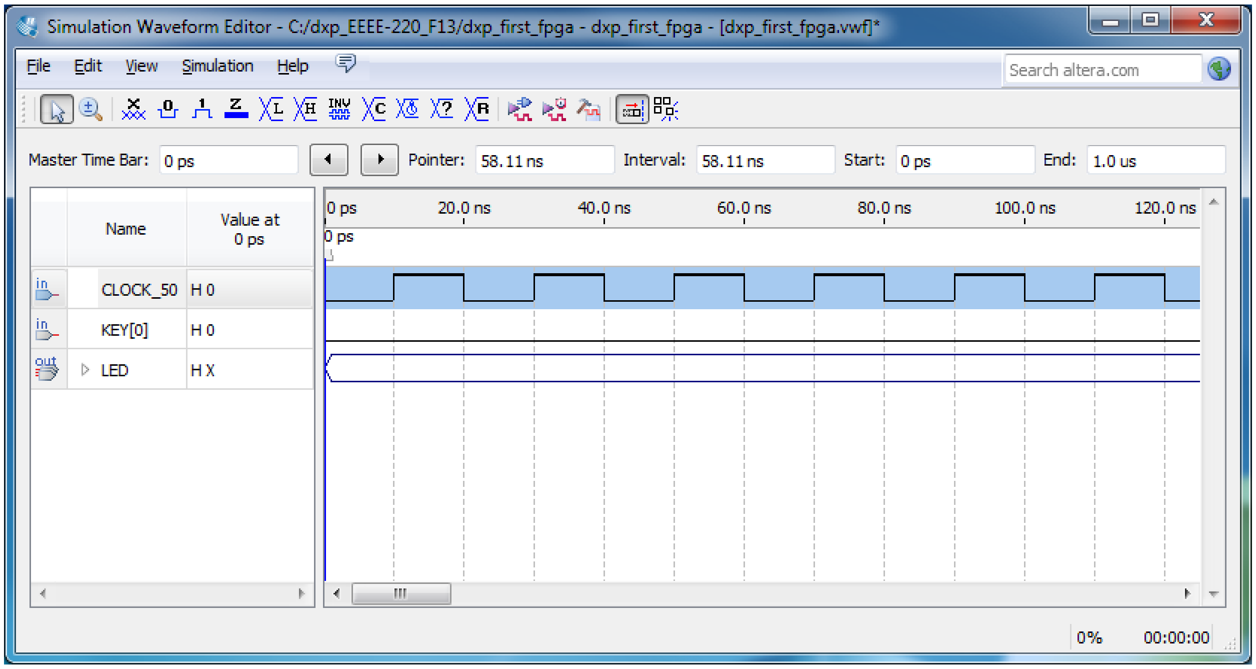
Binary [✓]

Gray code

Transitions occur

Count every: 10.0 ns

k. This allows you to select a periodic signal for this input pin. Click OK, i.e. leave the count value at 10 ns. Back in the main window you can see now a periodic signal with a period of 20 ns and a duty cycle of 50% (the signal is high for half the period). To see this better you may want to go to View > Zoom In several times.

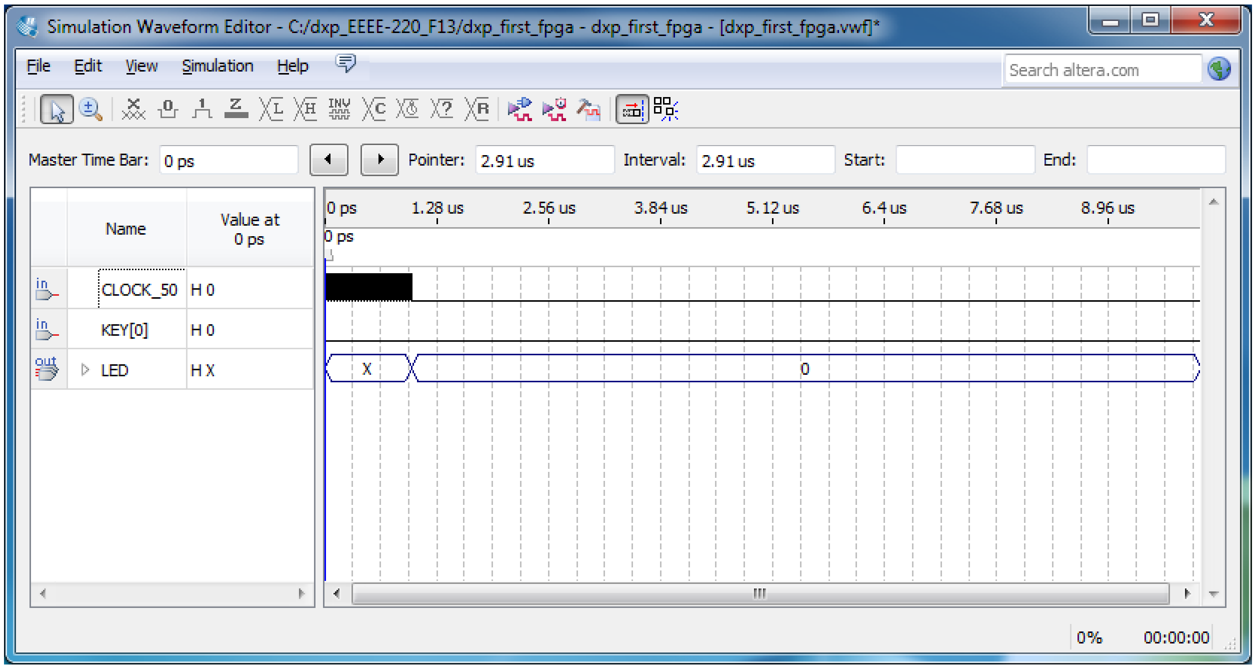

Simulation Waveform Editor - C:/dxp_EEEE-220_F13/dxp_first_fpga - dxp_first_fpga - [dxp_first_fpga.vwf]*
File
Edit
View
Simulation
Help
Search altera.com
0 ps
20.0 ns
40.0 ns
60.0 ns
80.0 ns
100.0 ns
120.0 ns
0%
00:00:00

Master Time Bar: 0 ps   Pointer: 58.11 ns   Interval: 58.11 ns   Start: 0 ps   End: 1.0 us

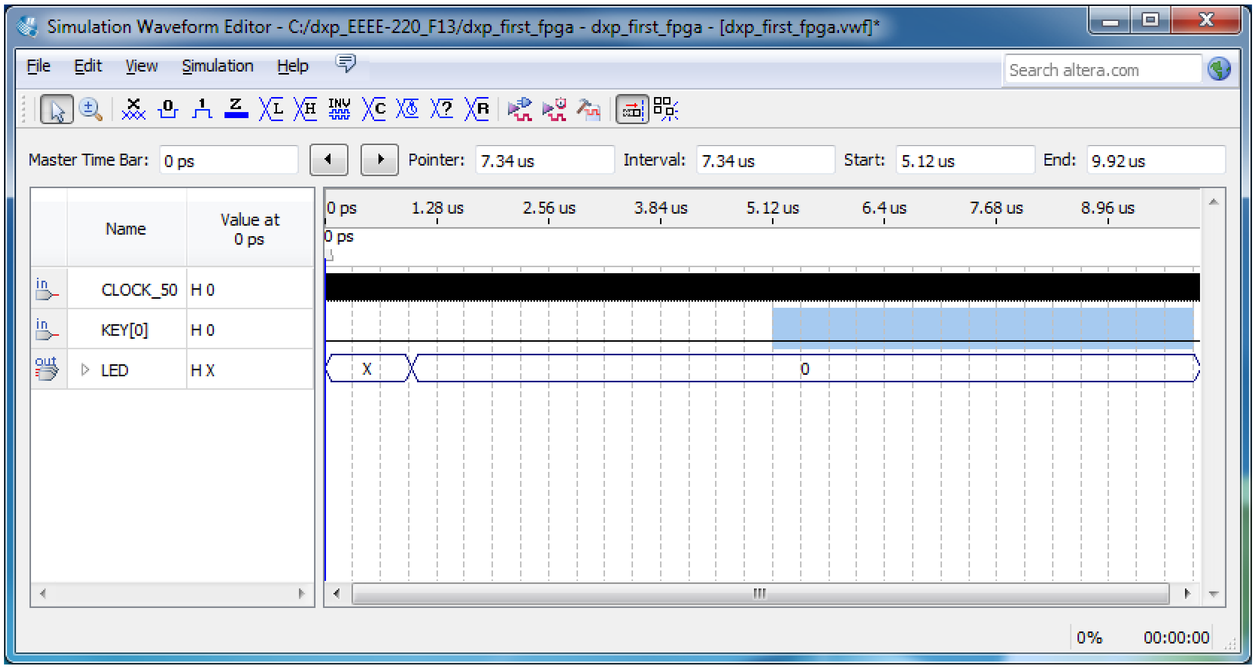
Name	Value at 0 ps
CLOCK_50	H 0
KEY[0]	H 0
LED	H X

l. View > Fit Window. Remember that the KEY[0] input pin selects different counter outputs to be displayed as LEDs ON or OFF. Let's first extend to total simulation time. Go to Edit > Set End Time and set it to 10 us. Click OK.

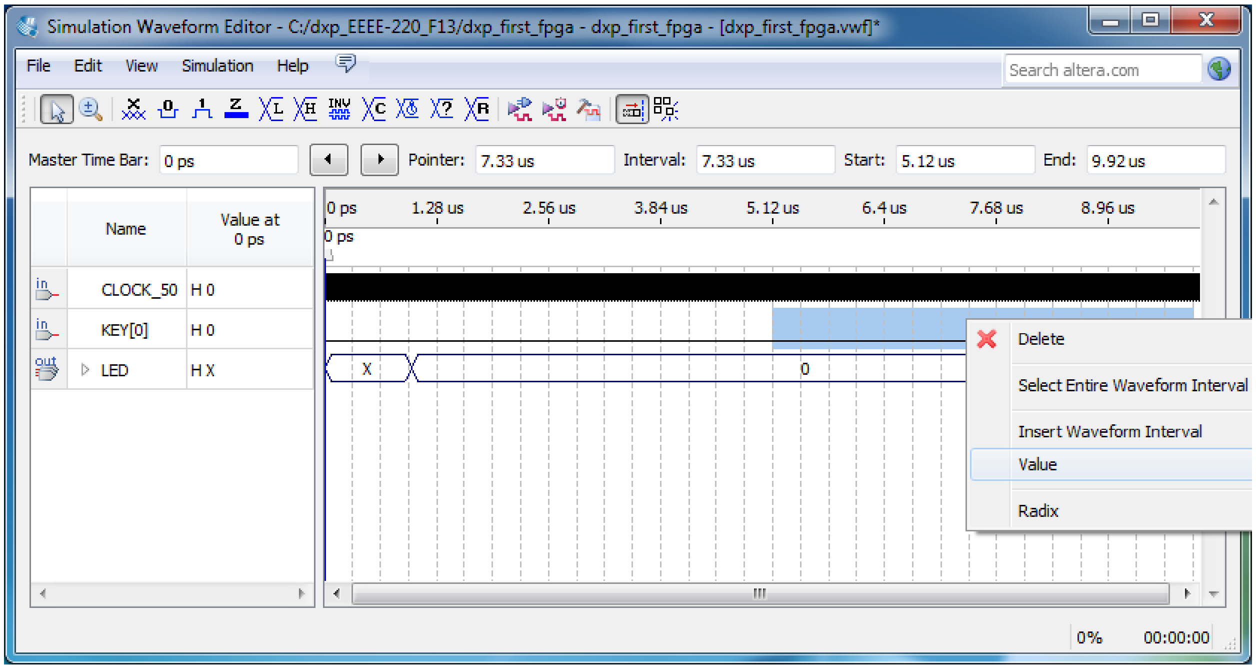
m. Notice that the CLOCK_50 signal has not automatically "stretched" to the end of the simulation time. Fix it as described in step k above. Don't worry about the LEDs. These are outputs and will change as a result of the simulation anyway.



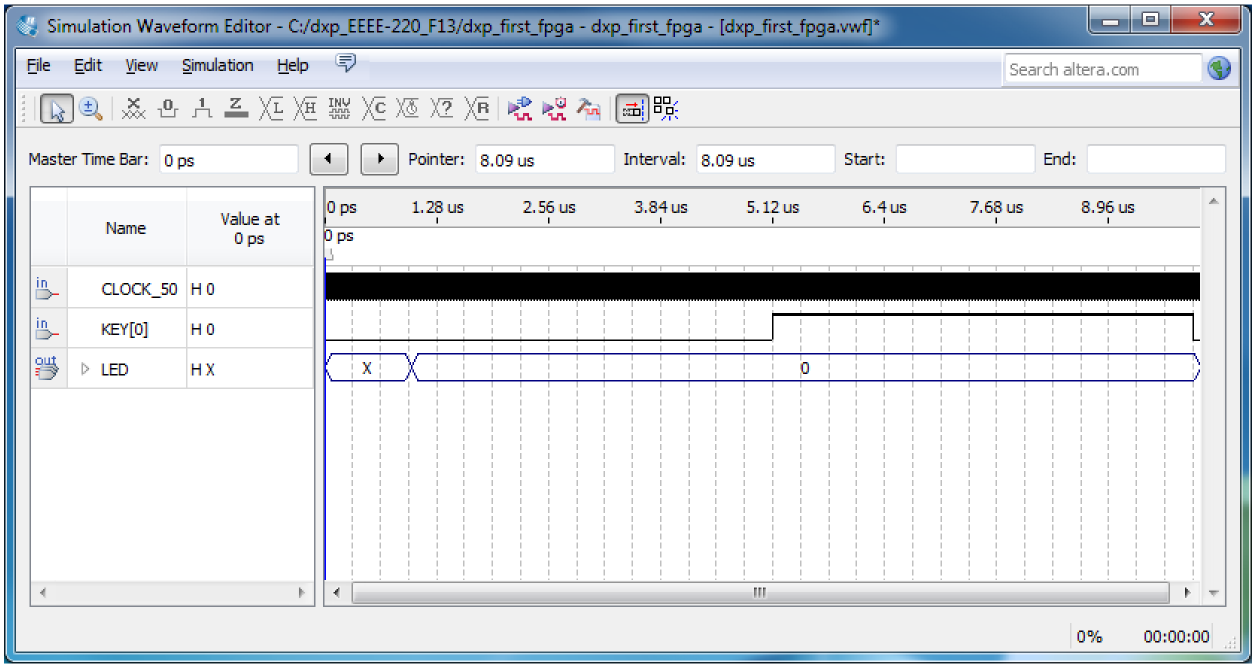
n. In the main simulation window select half of the KEY[0] signal as shown below. Click – drag – release with the mouse.



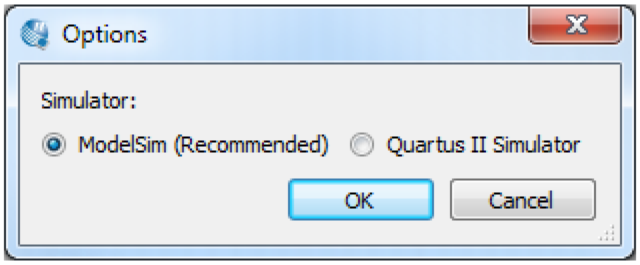
o. Now right-click, select value and force high.



p. Your edited waveform should look now as shown below:


Simulation Waveform Editor - C:/dxp_EEEE-220_F13/dxp_first_fpga - dxp_first_fpga - [dxp_first_fpga.vwf]*
File Edit View Simulation Help
Master Time Bar: 0 ps
Pointer: 8.09 us
Interval: 8.09 us
Start:
End:
Name
Value at 0 ps
CLOCK_50 H 0
KEY[0] H 0
LED H X

q. Before we simulate, go to Simulate > Options. You'll see a choice between two simulation engines. We will use throughout the semester the ModelSim simulator, i.e. leave the selected choice as is and click OK.


Options

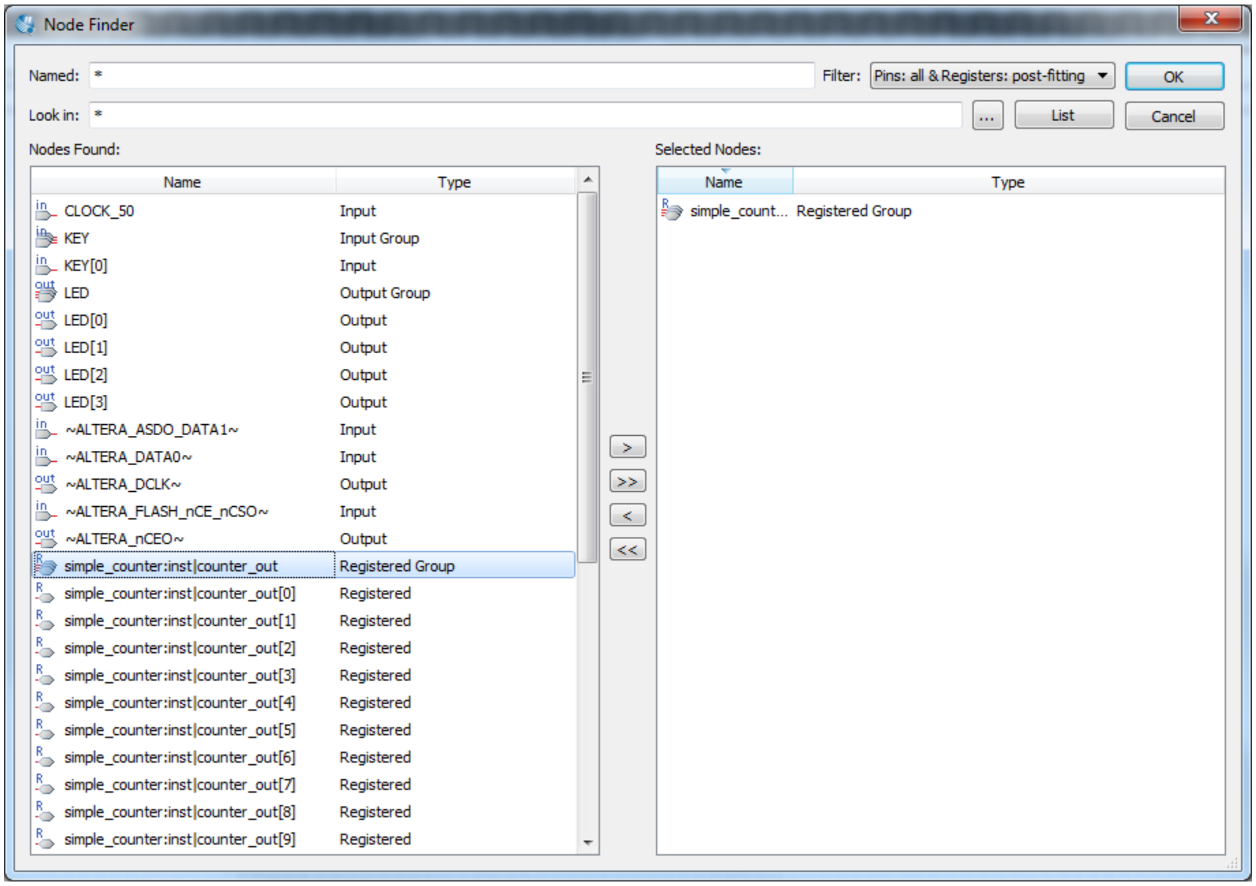
Simulator:

[✓ ModelSim (Recommended)] Quartus II Simulator

r. Go to Simulation > Run Timing Simulation. If it asks you to save click Yes.

s. A window appears in the left upper corner listing what the simulator is doing.

t. A new, read-only, waveform window appears. Notice that the LED pin values remain 0 throughout the simulation. This is because the count has not arrived at the point at which some of these would change to 1. To see that the counter works, let's go back to Node Finder, select to list "Pins: all & Registers: post-fitting" and choose the "simple_counter" registered group. These are not pins, but rather internal bus lines.


Node Finder

Named: *

Filter: Pins: all & Registers: post-fitting

Look in: *

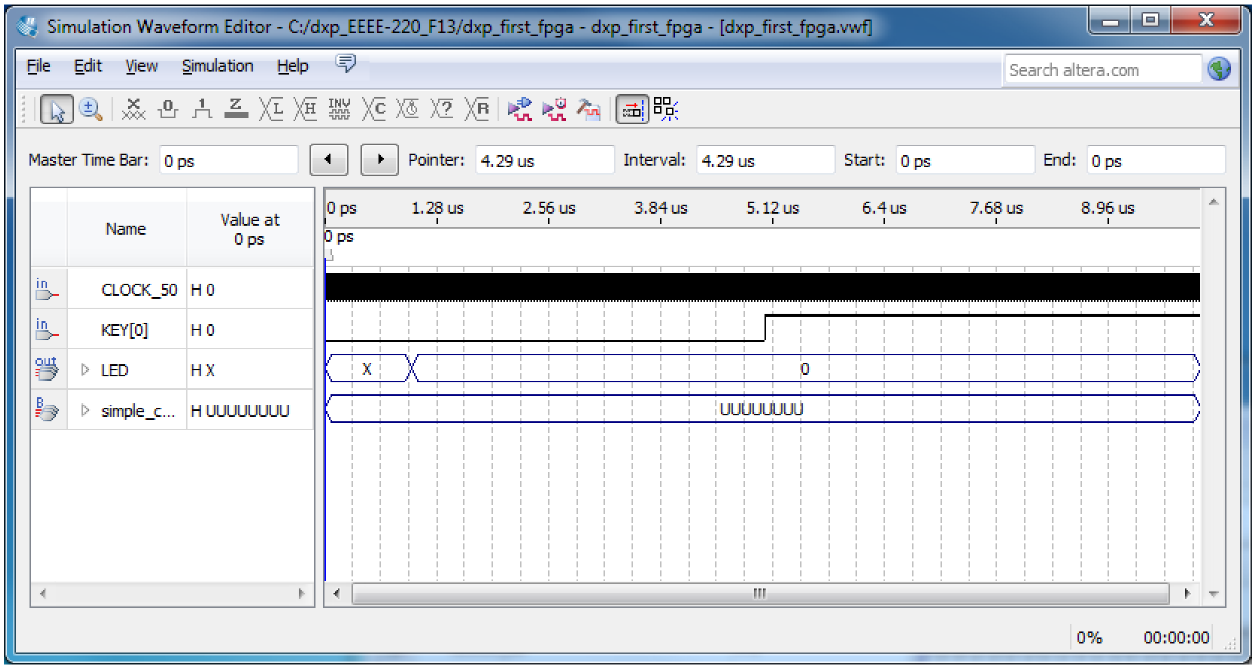
Nodes Found:

Name	Type
CLOCK_50	Input
KEY	Input Group
KEY[0]	Input
LED	Output Group
LED[0]	Output
LED[1]	Output
LED[2]	Output
LED[3]	Output
~ALTERA_ASDO_DATA1~	Input
~ALTERA_DATA0~	Input
~ALTERA_DCLK~	Output
~ALTERA_FLASH_nCE_nCSO~	Input
~ALTERA_nCEO~	Output
simple_counter:inst|counter_out	Registered Group
simple_counter:inst|counter_out[0]	Registered
simple_counter:inst|counter_out[1]	Registered
simple_counter:inst|counter_out[2]	Registered
simple_counter:inst|counter_out[3]	Registered
simple_counter:inst|counter_out[4]	Registered
simple_counter:inst|counter_out[5]	Registered
simple_counter:inst|counter_out[6]	Registered
simple_counter:inst|counter_out[7]	Registered
simple_counter:inst|counter_out[8]	Registered
simple_counter:inst|counter_out[9]	Registered

Selected Nodes:

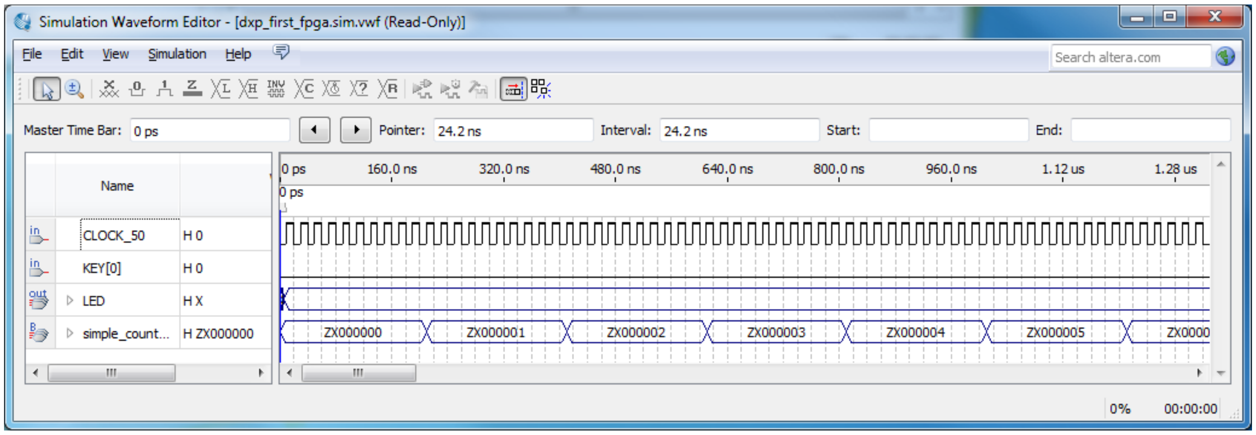
Name	Type
simple_count...	Registered Group

u. Click OK. Select Hexadecimal and OK again. Your waveform editor window will now look as below:



v. Simulate again by going to Simulation > Run Timing Simulation.

w. Now in the read-only output simulation file you can see that the counter is working. The change in the count takes 10 CLOCK-50 periods because we have chosen to divide the input clock by 10!



x. We will cover simulation in the next labs more extensively. For now close both waveform/simulation windows and return to the main Quartus II window.

17) Continue from here on with section 6.9 and 6.10 in the DE0-Nano manual. Step 3 in 6.10 is optional.

18) Archive the project by going to Project > Archive. You'll find the archived project in your working directory as fml_first_fpga.qar.

19) File > Close Project and then close Quartus.

20) Show your working design to the TA. Write your report and upload it along with your project archive in the dropbox on mycourses, as described in the lab policy.