CSEE W4840 Embedded System Design Lab 2

Stephen A. Edwards

Due February 17, 2011

Abstract

Learn to use the Nios II IDE (programming environment) to implement an Ethernet chat client on the DE2 board.

Unlike the first lab, your job here is to develop software. We have supplied a hardware design for the DE2 that includes the Nios II processor, memory, an Ethernet controller, a VGA controller, and a controller for the PS/2 keyboard.

Your task is to develop software that uses this hardware to implement an Ethernet-based chat client. This will function as a terminal: the user should be able to type in a line of text using the attached keyboard and see it appear on the video display. When s/he presses enter, the contents of the line should be sent as a UDP broadcast packet.

Similarly, the board should display on the screen every UDP broadcast packet it receives. This assumes that any such packet comes from a similar project.

To clarify who is typing what, the first few characters of each packet should contain the user's name. Have the user type this when your system starts, save it, and send it automatically at the beginning of each packet.

We have connected all the boards in the lab to hubs to form a local-area network that is not connected to the Internet to avoid causing problems for others. We have also connected a Linux workstation operating as a "packet sniffer" (the *tcpdump* program) that will let you observe details of each packet and warn you when it is malformed.

Programming the board for this lab will take two steps. First, open the lab2 project in Quartus like you did for lab1. We already compiled it for you, so you can go directly to the programmer and download the DE2_NET.sof file to the board. This configures the hardware but leaves the software unconfigured. To bring the board to life, download and run software on it using the Nios II IDE, described below.

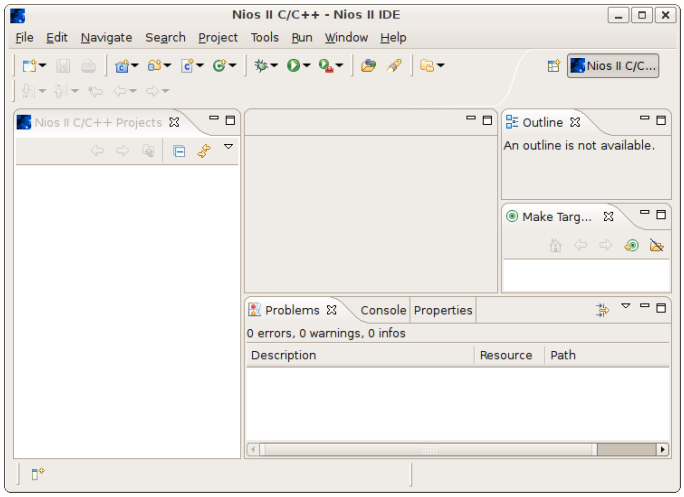

Figure 1: The Nios II IDE workbench ready to accept a new project

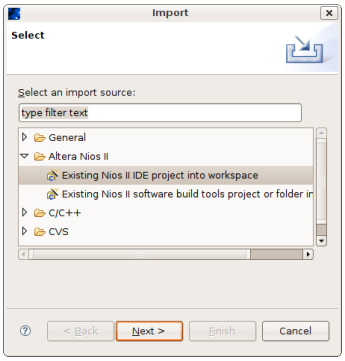

Figure 2: Importing projects into the IDE

1 The Nios II IDE

Start the Nios II IDE by typing *nios2-ide*. This will eventually bring up a window. If you get a (usually bogus) "Workspace currently in use" message, try deleting the "nios2-ide-workspace-6.1" directory in your home directory. The IDE is based on Eclipse, so you may find it familiar. The environment variables *SOPC_KIT_NIOS2*, *SOPC_BUILDER_PATH_61*, and *QUARTUS_ROOTDIR* must be set. Check these if you have problems starting the IDE.

When it starts, it may give you a few icons to choose among. Select "Workbench" to get started.

First, set your workspace. This is the directory in which the IDE will put files. Select File→Switch Workspace and select your directory for lab2.

When you select a new workspace location, it should begin empty (Figure 1). Start by informing the IDE about the provided projects.

To bring the software into the IDE, right-click on the "Nios II C/C++ Projects" window on the left side of the main window and select "Import." Under "Altera Nios II," select "Existing Altera Nios II Project into Workspace" (Figure 2). Click on Next >, then choose the lab2 directory in the project directory. Click "Finish" to import it.

Repeat the process by importing the lab2_syslib project, which holds information about the hardware configuration. You must specify the target processor by selecting the nios_0.ptf file in the lab2 directory. The lab2 project will not build without the

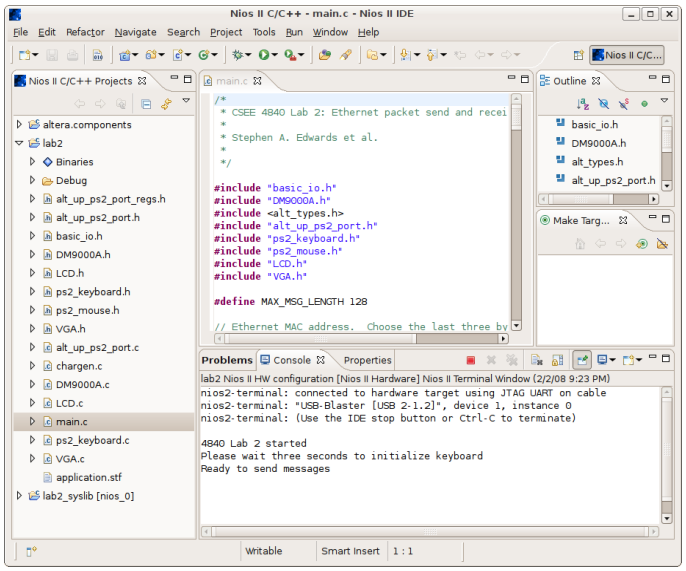

Figure 3: Running a program and observing output

lab2_syslib project.

After importing both lab2 and lab2_syslib, build them by selecting "Projects" and "Build All." Once built, you can run it on the DE2 by selecting "Run," "Run As...," and "Nios II hardware." This choice does not appear unless the project has been built, and it will not work unless you downloaded the DE2_NET.sof file to the board using Quartus.

Conveniently, after the Nios II IDE runs your program, it connects a terminal to the DE2 board that allows you to print and type to your running C program through standard *printf* and *getchar* calls. Figure 3 shows a running project with the terminal connection.

2 The Chat Application

The lab2.tar.gz file has a partially-working skeleton for the application. The code we supplied is awful, but does illustrate a few things; do not be afraid to modify or discard it. Here is a list of things you need to do:

- Make the VGA display work properly. This is a one-bit-per-pixel, 640×480 framebuffer with the ability to set the foreground ("on") and background ("off") colors across the whole screen. There is also an odd "cursor" mode that draws a big cross on the screen—don't bother with it.

  We have supplied a rudimentary text-mode character generator in *chargen.c* that displays 8×16 characters.

  Do the following with the display.

  - Clear the screen when the program starts.
  - Separate the screen into two parts with a horizontal line between. Use the bottom two rows as the user's text input area, and the rest of the screen to record what s/he and other users type.
  - When a packet arrives, print its contents in the "receive" region. Don't forget to wrap long messages across multiple lines.
  - When printing reaches the bottom of the area, you may either start again at the top, or scroll the entry region of the screen.
  - Implement a reasonable text-editing system for the bottom of the screen. Have input from the keyboard display characters there and allow users to erase unwanted characters and send the message with return. Clear the bottom area when a message is sent.
  - Display a cursor where the user is typing. This could be a vertical line, an underline, or a white box.

- Make the keyboard input work properly. Specifically,

  - Make both shift keys work (i.e., do upper and lowercase characters)
  - Make the space bar work properly (display a space)
  - Turn off the debugging information for the unrecognized keycodes
  - Make the left and right arrow keys work
  - Make the backspace key work
  - Ignore the other keys (e.g., tab, escape, print screen, the keypad, etc.).

- When the system starts, have the user enter his/her name before going into "chat" mode. Start the string sent by each packet with this name.

- Ensure the UDP packets are well-formed:

  - Make the header checksum correct
  - Make sure the packets are always at least minimum length (64 bytes)
  - Make sure the string sent in the UDP packet is always zero-terminated
  - Make sure the UDP packet length field is correct
  - Make sure the IP packet ID numbers increase
  - Choose a different Ethernet MAC address and make sure you're putting your address in the packet appropriately.

3 What to turn in

Find an overworked TA or instructor, show him/er your working chat application, demonstrate that it is sending well-formed packets, and email your source code to `sedwards@cs.columbia.edu`.