Configure and Program the myRIO

Aaron Feldman, Antonio Rohit de Lima Fernandes

University of California, Berkeley

September 18, 2014

Abstract

In this lab, we got familiar with connecting to, configuring, and programming the myRIO. We did this using a number of development tools - LabVIEW, NI MAX, Xilinx SDK, and Eclipse. This lab write-up demonstrates and summarizes the authors' experience through the process.

I. Connect to and Configure the myRIO

Through the prelab, the authors (henceforth referred to as 'we') developed a familiarity with the myRIO model 1950. This is an embedded micro-controller with two independent processors on board: an ARM Cortex-A9 and an FPGA.

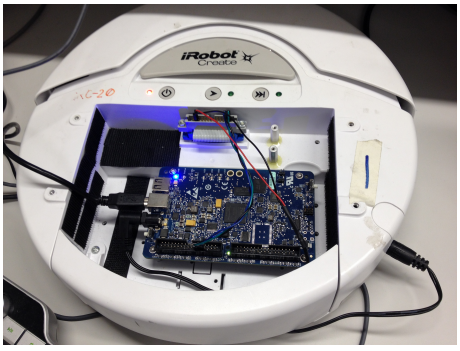

Figure 1: myRIO board

Labview:

To begin with, we placed the battery into the myRIO, powered it and connected the myRIO to the computer. To configure it, we launched the getting started wizard from LabVIEW for myRIO>Set Up and Explore.

The myRIO was discovered by the software. We renamed it and verified that the accelerometer, button and LEDs were responding as expected. In particular, we noticed that at rest, the Z-acceleration was 1 g, while X and Y were 0.

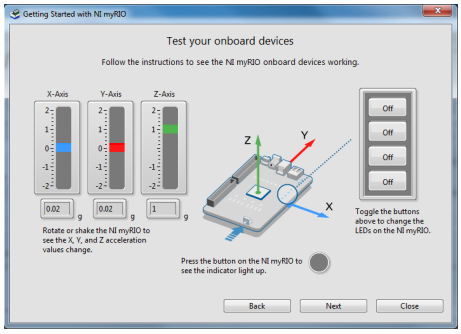

Figure 2: myRIO Onboard Devices

NI MAX:

Next, in order to see detailed information about the myRIO, we switched to the NI MAX tool - which provides many configurations options for the myRIO - Network adapters, Installed software, Date/Time, Web services management, Web server config, and Installed config tools. Most importantly, it displayed the IP address of the myRIO - which is useful to access the board. Once we were done, we restarted the myRIO from software.

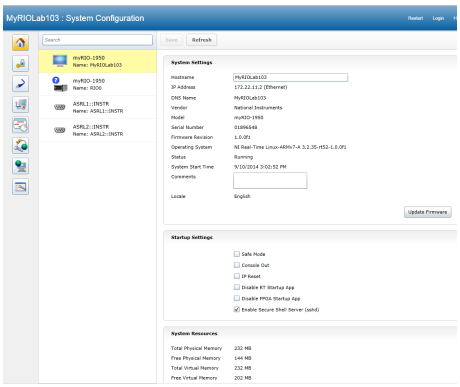

Figure 3: myRIO through NI MAX

II. Program MicroBlaze from Xilinx SDK

Now that we configured the myRIO, we were ready to program the FPGA on board. ote that though the steps seemed straightforward, this was the section we had most trouble with - we had to return on a separate day to finish this section of the lab.

The first step was to connect the Xilinx Platform cable to JTAG header J5 on board. This required attention to detail to get the pins mapped correctly. When correctly mapped (power and gnd particularly), the status LED on the platform cable turns from yellow to green.

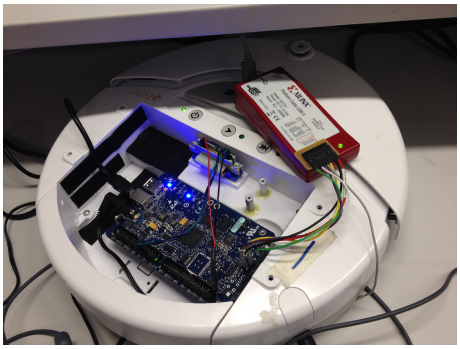

Figure 4: myRIO through NI MAX

Next, we programmed the Xilinx FPGA on board with the MicroBlaze soft-core processor. This was done by downloading the provided bitfile onto the flash memory of the FPGA using the RIO Device Setup tool. We verified that LED1 was glowing as expected.

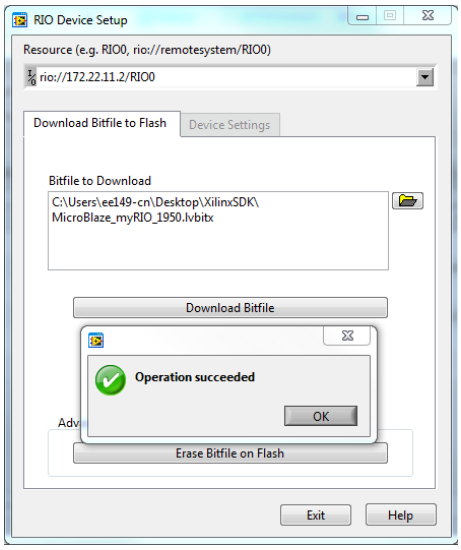

Figure 5: Successful Bitfile Download!

The FPGA was now configured as a MicroBlaze processor, and was ready to be programmed. To do so, we opened the Xilinx SDK in the C/C++ perspective, and imported the MicroBlaze hardware support package provided.

Following the instructions, we cleaned the workspace, created a new project named hellowWorldMicroBlaze, and then built and ran it. We modified the code to print "Hello Wurld" to STDIO.

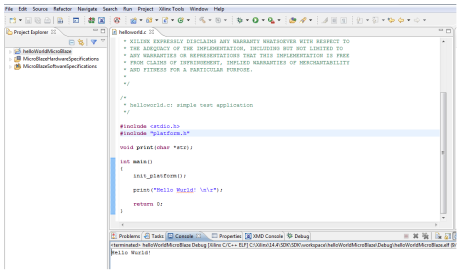

Figure 6: Xilinx SDK Hello World

Next, in order to issue commands to the FPGA using the XMD console, we entered into Debug mode, and sent commands like *rst* and *stop*.

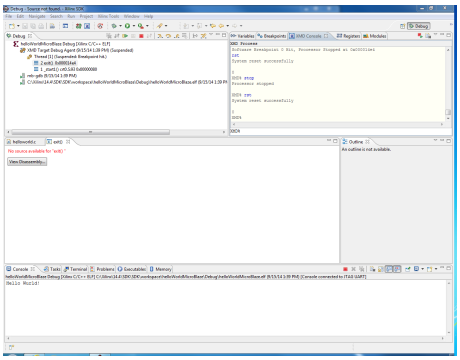

Figure 7: Using XMD Console to Issue Commands

III. Program myRIO Processor from Eclipse

In the previous exercise, we programmed the FPGA. This exercise focused on programming the ARM Cortex-A9 on the board, running Linux.

To do so, we began by entering the IP address of the myRIO in a browser to configure the A9 to allow SSH. The myRIO was restarted in order for these settings to take effect.

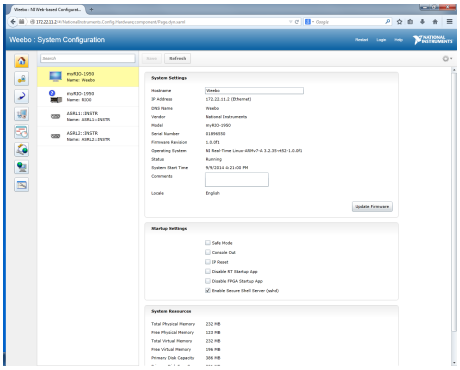

Figure 8: Web Config Interface

It was time to begin using Eclipse. We opened it, imported the provided template project, and renamed it to myRIO Hello World. The model number that the project had by default (in the Properties > C/C++ General > Paths and Symbols setting) was *MyRio*_1900, so we changed this to *MyRio*_1950 to correspond to the hardware. We verified that the project built successfully.

Since we needed to dig into the file system of the A9, we launched the Remote System Explorer (RSE) perspective in Eclipse. We created a new connection to the myRIO connected the the desktop by using its IP address, and this made it show up in the Remote Systems pane.

Then we connected to it by using the username: "admin" and a blank password. We launched a terminal window and entered the uname -all command to display the information about the Linux OS on myRIO.

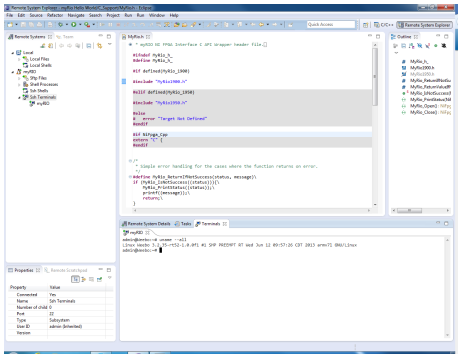

Figure 9: Remote Systems Pane and SSH

Very interestingly, the FPGA can be configured through this Linux interface, in a different manner than what we did in the previous section.

Just by exporting a bitfile to /var/local/natinst/bitfiles in the filesystem the myRIO programs the FPGA when the application code runs!

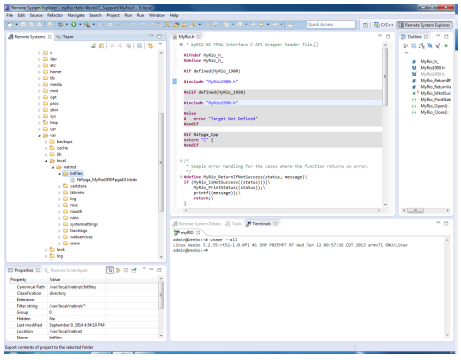

Figure 10: Location to Store bitfile

Now that we were done with the Remote Systems perspective, we switched back to the

C/C++ perspective to program a Hello World project on the A9.

We added a printf statement to the application code area. Then, we created a debug configuration, following the instructions provided, and then launched a debug session.

By stepping over code, we were able to have "Hello World" print to STDIO.

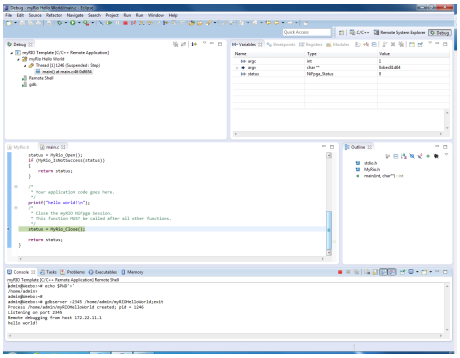

Figure 11: Hello World Debug Session

We can even control hardware through this same program! We modified the code as suggested in the instructions, using the line $NiFPGA\_WriteUS(myrio\_session, D0LED30, 0x0x)$ to control the LEDs. Here x determines which combination of LEDs are on. The figure below has x = 2, i.e. LED2 is on.

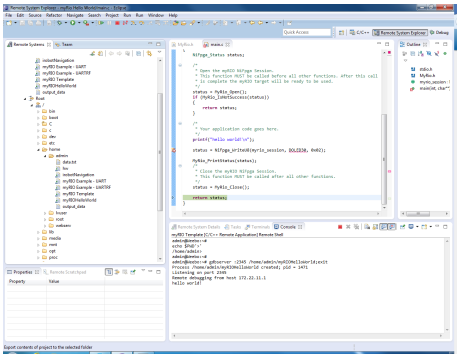

Figure 12: Modified Code to Control LEDs

IV. Program myRIO Processor from LabVIEW

In the previous two sections, we programmed the myRIO using Xilinx SDK (for the FPGA directly), then Eclipse (for the A9 and indirectly the FPGA), and now use LabVIEW for the A9.

For this, we created a new myRIO project called Acceleration Threshold. We ran it and verified that the acceleration worked as expected.

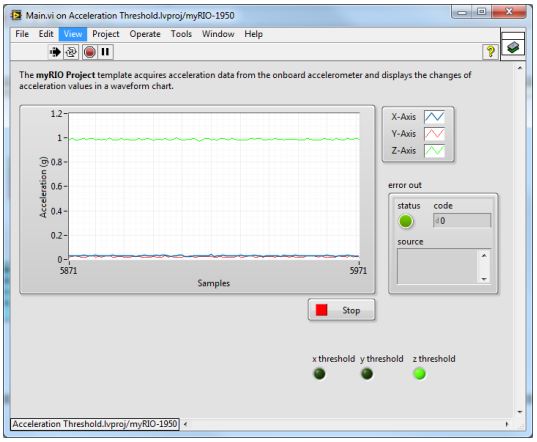

Figure 13: Acceleration Graphs using LabVIEW

Next we opened up the block diagram of the project. Our aim was to control 3 LEDs - LED0, LED1, and LED2, using the 3-axis accelerometer on board. If the acceleration in a particular axis exceeded a configurable threshold, the LED was supposed to go on.

The block diagram below shows how we implemented it:

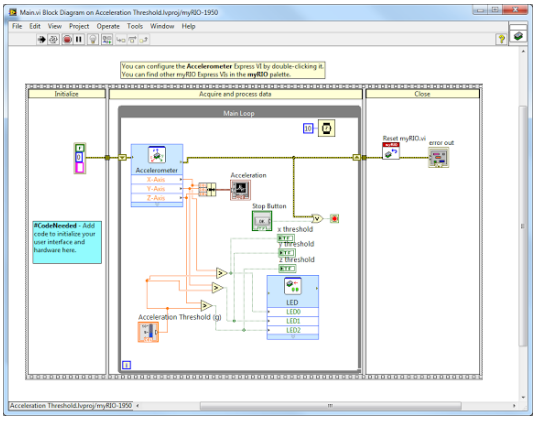

Figure 14: LEDs Controlled By Accelerometer

This is how the GUI looked:

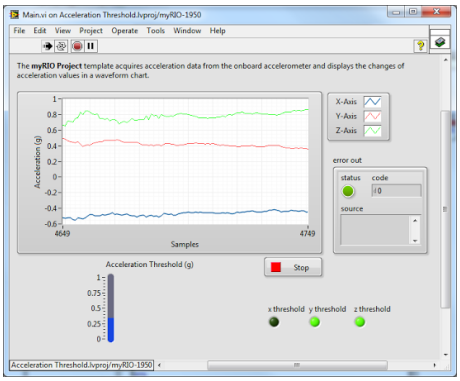

Figure 15: LED control with Configurable Thresholds

How this was done: A linear slider was fed into 3 comparators - one each for X, Y, and Z acceleration. The output of each comparator was fed into the an LED block mapped to physcial LEDs on board, and also to virtual indicator LEDs on the screen.

V. Lab Feedback

This was a fun lab, though the experience was marred by the fact that we couldn't get one section of the lab to work at all. It seems like the lab instructions are not yet mature and of course, the troubleshooting section not exhaustive. However, through the lab and the writeup (veritable revision), I believe we have gained a valuable introduction on how to use the myRIO board, and hence the objective of the lab has been achieved.