

00.L Lab Exercise: Getting started

Objective

The objective of this exercise is to get acquainted with the following.

1. The Eclipse IDE and debugger.
2. Editing, building, loading, and running a program on the target computer.
3. Setting break points and single-stepping through a running program.
4. Displaying register and memory contents.

Pre-laboratory preparation

Read [Resource 2](#) and [Resource 7](#), following the instructions on your personal computer, preferably a laptop you can bring to lab. Pay particular attention to procedures for editing, building, and debugging. The following laboratory procedure is a tutorial that will allow you to become familiar with the Eclipse IDE.

Laboratory procedure

Perform this procedure in the lab, either on your laptop or a lab computer, connected to a myRIO. Launch Eclipse, Open the `main.c` file of your myLab0 project. Edit the file to include the C code in [Figure L.1](#). Substitute your name for `<your name>` (12 characters max). Note the use of `<tab>`s and indenting.

Look carefully at this program. The `main` program loops four times, calling `sumsq` each time. What values do you expect in the `x` array after the program has executed?

Use the build command to compile and link your program. Errors and Warnings are shown in the Console pane. Correct any errors and re-build.

Run→Run Configurations the program. Select your myLab0 project. Click Run. The results will be printed on the LCD display.

Using the debugger

In the following, you may find it useful to refer to the outline of important debugger commands in the Eclipse IDE for myRIO Notes. No program may be running on the target myRIO when you start the debugger.

Select Run→Debug Configurations. Select your myLab0 project. Click Debug. The Debug perspective opens, showing the source code for the function main.

At this point, the execution has been suspended at the line highlighted in the source code.

Notice that the values of the program variables are displayed in the Variables pane anytime that execution is suspended. What are the current values of *i* and the array *x*? Notice also that the values of the processor registers are shown in the Registers pane.

```
/* Lab #0 - <your name> */

/* includes */
#include "stdio.h"
#include "MyRio.h"
#include "me477.h"

/* prototypes */
int sumsq(int x); /* sum of squares */

/* definitions */
#define N 4 /* number of loops */

int main(int argc, char **argv) {
    NiFpga_Status status;
    static int x[10]; /* total */
    static int i; /* index */


    status = MyRio_Open(); /* Open NiFpga.*/
    if (MyRio_IsNotSuccess(status)) return status;

    printf_lcd("\fHello, <your name>\n\n");
    for (i=0; i<N; i++) {
        x[i] = sumsq(i);
```

```
        printf_lcd("%d, ",x[i]);
    }
    status = MyRio_Close(); /* Close N:Fpga. */
    return status;
}
int sumsq(int x) {
    static int    y=4;

    y = y + x*x;
    return y;
}
```

Figure L.1: C code for Lab 00

Executing the Program—You are about to execute your program from the debugger. Use the  icon to resume execution of your program. The only indication that your program has executed will be that your name and the values of the x-array should be displayed on the LCD display attached to the myRIO. Are the results consistent with your understanding of the program? Explain.

Running to a Breakpoint—A “breakpoint” is an address in memory where we would like the processor to stop while we examine or modify the state of the myRIO and/or memory. The target processor runs continuously unless it is stopped at a breakpoint.

Now let’s re-execute the program until the execution arrives at a specified breakpoint. Start the debugger again from Debug

Configurations....


Suppose that we want to continue execution (from the beginning) and determine the values of x and i just before the first C statement inside the “for” loop executes for the first time. Set a breakpoint at the “x[i]=sumsq(i);” statement by double-clicking on the marker bar next to that source code line.

A new breakpoint marker appears on the marker bar, directly to the left of the line where




you added the breakpoint. Also, the new breakpoint appears in the Breakpoints pane.


Run to the breakpoint using the  icon.


The text window should now show execution suspended at that line. The Variables pane should now display the new values of `x` and `i`. The window marks in yellow values that have changed. Are they what you expected?




Finally, (assuming that execution has stopped at the first “`for`” loop iteration), cause the debugger to execute the loop one more time using the  icon. The values of `x` and `i` should be updated in the Variables pane. Are they what you expected? Try it again. ...And again. ...And again! Watch the progress of the program on the LCD display.

Single-Stepping—The debugger can also step through the execution of the program in three ways:

1. “Step Over”  Execute the current line, including any routines, and proceed to the next statement.
2. “Step Into”  Execute the current line, following execution inside a routine.
3. “Step Return”  Execute to the end of the current routine, then follow execution to the routine’s caller.

“Step Over” single steps to the next sequential C statement, but executes through functions, and out of branches and loops before pausing. For example, if the next line of code is a call to a function, pressing  will cause the entire function to be executed and debugger will pause at the line of code following the function call. To begin the stepping process the target must be suspended.

Terminate execution , and then restart the debugging. Execution will be suspended at the beginning of the program.

Now, single step from this point using “Step Over”  repeatedly. Notice that step corresponds to a single line of C code. Notice also that the current values in the Variables pane change as you step. Watch the progress of the program on the LCD display. Eventually, execution exits through the `return` statement. Restart the debugging again. This time run to your breakpoint at “`x[i]=sumsq(i);`”. Now, use “Step Into”  to follow the execution into `sumsq`. Continue stepping using  until execution exits back to `main`.

Quitting—You may terminate the debugging at anytime using terminate .

Feel free to repeat these procedures and to try other commands.

Resource R1 High–level embedded system

The Embedded Computing Laboratory (ECL) at Saint Martin's University is a space dedicated to teaching embedded computing in electromechanical systems. It is hosted by the Robotics Laboratory and developed in collaboration with Prof. Joe Garbini of the Department of Mechanical Engineering at the University of Washington (UW), to whom credit for much of the design is owed.

The following description is of the apparatuses at ECL, which are similar to those at the UW.

The primary differences are that each lab has a different set of motors and the UW uses a custom analog amplifier to drive the motor whereas the ECL uses pulse-width modulation.

The developers of the following content distribute it in the hopes that others will find it educational and perhaps useful as a template for similar laboratories. Furthermore, we hope students will be able to reference it when they want to design their own embedded systems.

ECL has four identical systems for student use. Each system consists of four subsystems:

1. an embedded computer and development environment subsystem consisting of a National Instruments myRIO microcontroller, a personal computer, and the Eclipse IDE;
2. a user interface hardware subsystem consisting of a keypad, LCD display, and associated circuit boards;
3. a motor driver subsystem consisting of a dc power supply and a pulse-width modulation motor driver circuit board and
4. a motor and mechanical apparatus subsystem consisting of a flywheel supported by bearings and coupled to the shaft of a dc servomotor (with encoder for

feedback).

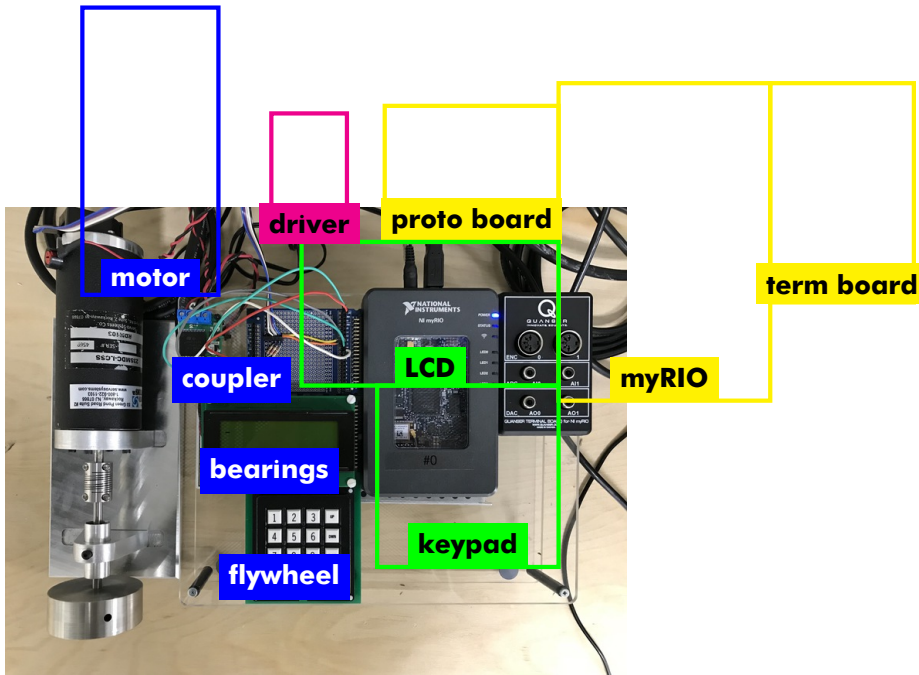


Figure 00.2: top view of most of the ECL apparatus.

Each of these is described in detail in the following sections. Together, they allow a student to program the microcontroller (in the C programming language) to instantiate completely embedded control of the motor speed and position, which are set by the user through the keypad and LCD display.

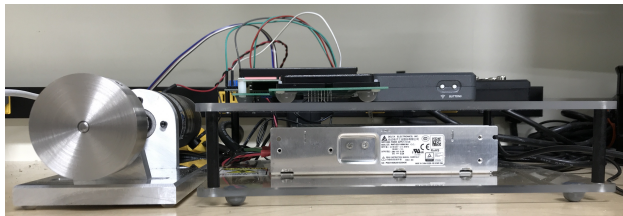


Figure 00.3: front view of most of the ECL apparatus.

Resource R2 Embedded computer and development environment subsystem

The development system is a powerful and convenient tool for embedded computing applications. As shown below, the development system consists of a personal computer, connected via a USB cable to target computer.

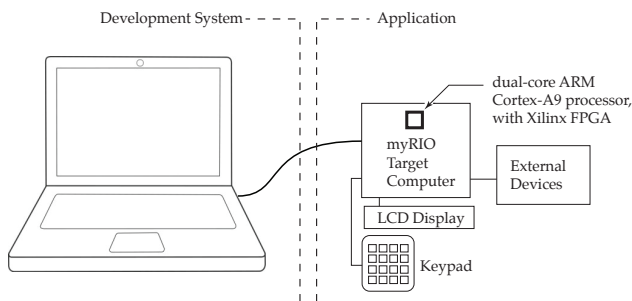


Figure 00.4:

During the development of an embedded computing application, the development system communicates with the real-time Linux operating system of the myRIO target computer. The development environment includes an integrated set of hardware and software tools that help to debug a microcomputer design by allowing you to watch your program execute, as well as to stop it and inspect system variables. As you will see, it allows you to monitor and control the target computer, without interfering with its timing.

Once hardware and software development is completed, the development system is disconnected from the target system. In the final application, the target program resides in ROM on the target computer.

Getting Started with CDT

Eclipse is an integrated development environment (IDE). We will use Eclipse through

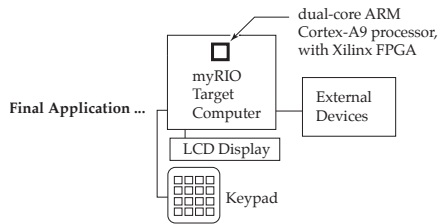


Figure 00.5:

its C Development Tool (CDT) to create, edit, build, deploy, and debug C language projects for the myRIO target computer. Within Eclipse all of your projects are organized into a single workspace on your computer. Each project, along with all of its necessary resources, are stored in a named project folder.

The outline below describes the basic functions of the IDE in preparing a C program for subsequent loading and execution on the myRIO remote system. Additional features are described in the Help menu.

Box 00.1 is CDT set up?

If the development PC has not yet been set up on the development computer, follow the procedure of [Resource 7](#) to do so, before continuing.

Begin by starting the Eclipse IDE application.

C/C++ Perspective

Enter the C/C++ Perspective by selecting that button in the upper right.

The Project The ME 477 C Support for myRIO archive that you imported into your Eclipse workspace when you set up the CDT contains a template project for each of the nine laboratory exercises this quarter. They are listed in the right pane of the C/C++ perspective. Open a project by double clicking on its folder.

Each C program consists of a collection of functions, one of which must be called `main{}`, and is executed first. For large projects, additional functions are often in separate files. However, the organization of the assignments in this class is such that all of the functions for a single assignment can be conveniently stored in `main.c` along with `main{}`.

Run and Debug Configurations Among other things, Run and Debug Configurations specify how the project will be stored on the remote target. Configurations for all ME 477 laboratory exercises were loaded into your workspace in steps 4 and 5 of Part 1 of the C Development Tool Setup documentation—see [Resource 7](#).


Building the Project Building the project consists of compiling your C source code into object modules, and linking them with other resources. Many coding errors can be found during the building process. Since building does not require that the development system be connected to the target, time spent in the lab is minimized. Before building, save any edit changes in the source code (`ctrl-s`). Either right click the project and use Build Project, or select and use Build Project from the Project pull down menu. Errors and warnings are displayed in the console menu in the bottom pane.

During each build, the CDT automatically re-compiles any file that has been edited (and saved). The build operation creates an output file in project's Debug folder.

Running the Project The project must build without errors before it can be run. The first time a project is run, pull down the Run menu and select Run

Configurations.... In the Run Configurations window, select the Run Configuration of your project. Then click Run.

The first run after a connection, you may be asked to login. Use User ID: admin and Password: me477.

Recently run projects may be conveniently run from the pull down menu under the run icon .


A project will not run if a project is already running.

Barring execution problems, the project runs until `main{}` terminates.

Debug Perspective

Enter the Debug Perspective by selecting that button in the upper right. The Debug perspective lets you manage the debugging or running of a program. You can control the execution of your program by setting breakpoints, suspending launched programs, stepping through your code, and examining the contents of variables.

Debugging the Project The project must build without errors before it can be debugged. The first time a project is debugged, pull down the Run menu and select Debug Configurations.... In the Debug Configurations window, select the configuration of your project. Then click Debug.

After the first debug, the project may be conveniently selected for debugging by pulling down menu under the debug icon .

A project may not be debugged if a project is already running.

Breakpoints A breakpoint suspends the execution of a program at the location


where the breakpoint is set. To set a line breakpoint, right-click in the marker bar area on the left side of an editor beside the line where you want the program to be suspended, then choose Toggle Breakpoint. You can also double-click on the marker bar next to the source code line. A new breakpoint marker appears on the marker bar, directly to the left of the line where you added the breakpoint. Also, the new breakpoint appears in the Breakpoints view list.


Once set, a breakpoint can be enabled and disabled by right-clicking on its icon or by right-clicking on its description in the Breakpoints view.


- When a breakpoint is enabled, it causes the program to suspend whenever it is hit. Enabled breakpoints are indicated with a blue enabled breakpoint circle.
- Enabled breakpoints that are successfully installed are indicated with a checkmark overlay.
- When a breakpoint is disabled, it will not affect the execution of the program. Disabled breakpoints are indicated with a white disabled breakpoint circle.


Debug view toolbar commands


The Debug perspective also drives the C/C++ Editor. As you step through your program, the C/C++ Editor highlights the location of the execution pointer.


Resume  Select the Resume command to resume execution of the currently suspended debug target.

Suspend  Select the Suspend command to halt execution of the currently selected thread in a debug target.

Terminate  Ends the selected debug session and/or process. The impact of this action depends on the type of the item selected in the Debug view.

Step Over  Select to execute the current line, including any routines, and proceed to the next statement.

Step Into  Select to execute the current line, following execution inside a routine.

Step Return  Select to continue execution to the end of the current routine, then follow execution to the routine's caller.

Debug information

Variables You can view information about the variables in a selected stack frame in the Variables view. When execution stops, the changed values are by default highlighted in red. Like the other debug-related views, the Variables view does not refresh as you run your executable. A refresh occurs when execution stops.

Expressions An expression is a snippet of code that can be evaluated to produce a result. The context for an expression depends on the particular debug model. Some expressions may need to be evaluated at a specific location in the program so that the variables can be referenced. You can view information about expressions in the Expressions view.

Registers You can view information about the registers in a selected stack frame. Values that have changed are highlighted in the Registers view when the program stops.

Memory You can inspect and change memory.

Disassembly You can view disassembled code

mixed with source information.

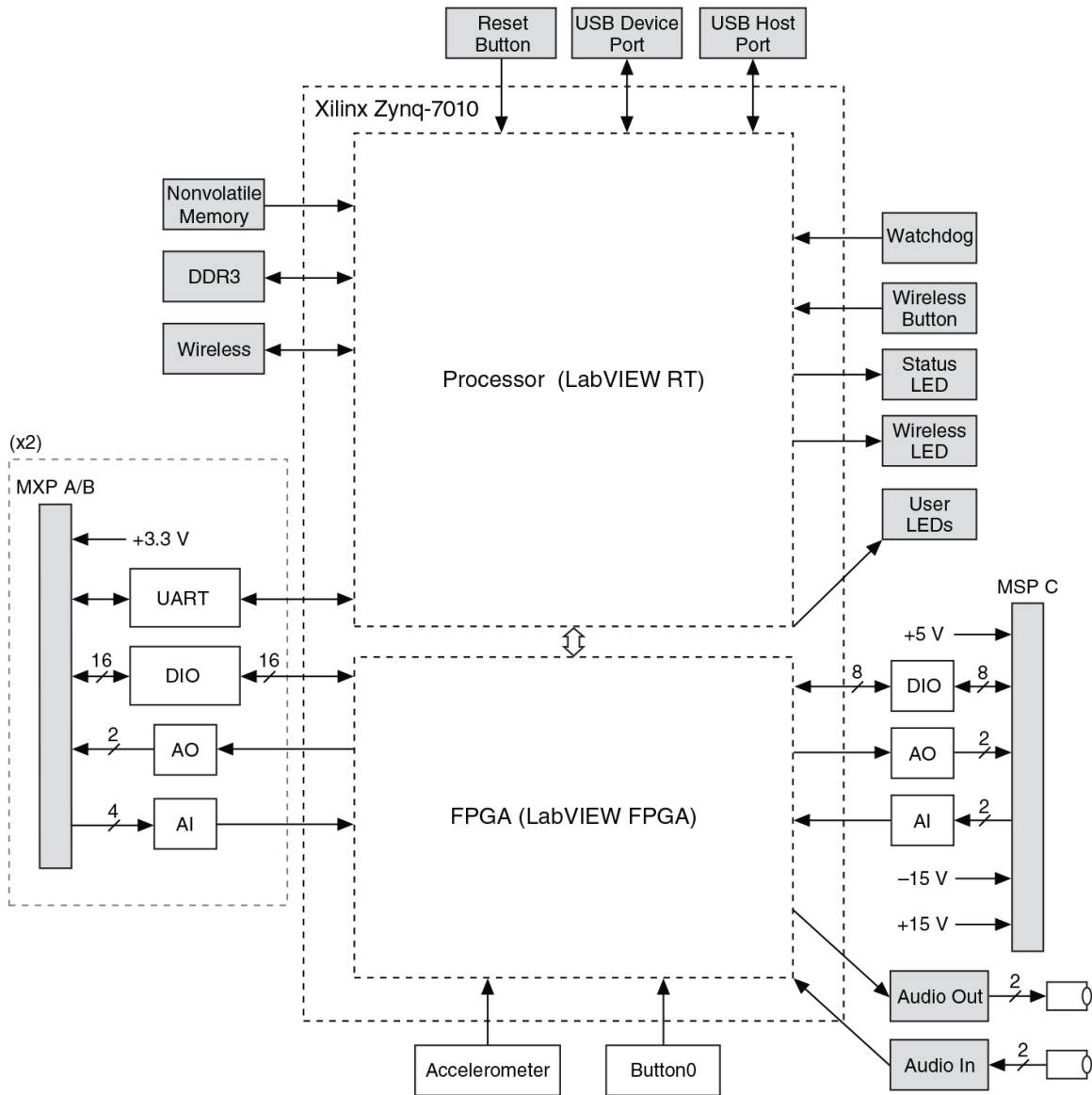


Figure 00.6: myRIO-1900 Hardware Block Diagram (source: Instruments (2013))

The system on a chip

The NI myRIO is centered around a Xilinx Z-7010 system on a chip (SoC): a dual-core Cortex A-9 CPU, memory, I/O interfaces, and

an Artix-7 fully programmable gate array (FPGA). The Z-7010 datasheet³ is available [here](#).

These are powerful SoCs. The Coretex A-9 CPUs have 667 MHz clocks, have single- and double-precision vector float point units, and include NEON extensions (Xilinx, 2017). These processors use the ARMv7-A instruction set architecture (ISA) (ARM, 2012, 2014). The Coretex-A9 Reference Manual and Programmer's Guide are available [here](#).

3. Xilinx, 2017.

Resource R3 User interface hardware subsystem

Resource R4 Motor driver subsystem

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum. Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris. Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia.

Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa. Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula. Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetur. Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum

eu, sodales cursus, magna. Donec eu purus.
Quisque vehicula, urna sed ultricies auctor,
pede lorem egestas dui, et convallis elit erat sed
nulla. Donec luctus. Curabitur et nunc.
Aliquam dolor odio, commodo pretium,
ultricies non, pharetra in, velit. Integer arcu est,
nonummy in, fermentum faucibus, egestas vel,
odio.
Sed commodo posuere pede. Mauris ut est. Ut
quis purus. Sed ac odio. Sed vehicula hendrerit
sem. Duis non odio. Morbi ut dui. Sed
accumsan risus eget odio. In hac habitasse
platea dictumst. Pellentesque non elit. Fusce
sed justo eu urna porta tincidunt. Mauris felis
odio, sollicitudin sed, volutpat a, ornare ac, erat.
Morbi quis dolor. Donec pellentesque, erat ac
sagittis semper, nunc dui lobortis purus, quis
congue purus metus ultricies tellus. Proin et
quam. Class aptent taciti sociosqu ad litora
torquent per conubia nostra, per inceptos
hymenaeos. Praesent sapien turpis, fermentum
vel, eleifend faucibus, vehicula eu, lacus.

Resource R5 Motor and mechanical apparatus subsystem

Motor

Mechanical apparatus

The motor hanger, shaft, shaft hanger, ball bearings, and retainer rings are best purchased from a single mechanical supplier (for tolerance matching); we have chosen the following [WM Berg](#) parts (catalog pages linked):

1. motor hanger (blank): [BC7-1C](#) (needs holes drilled),
2. shaft (1/4 in diameter, 4 in length): [s4-40](#),
3. shaft hanger (supports shaft through bearings): [BC17-12C](#),
4. ball bearings (2): [B1-9](#),
5. retainer rings (2): [Q4-62](#), and
6. split cylinder shaft coupler: [CO41S-2](#).

The split cylinder shaft coupler is a flexible coupler that helps with inevitable shaft misalignment.

The flywheel is the only custom-machined mechanical part. It is 304 stainless steel, one inch thick and 2.5 in diameter with a 0.25 in hole in the center for the shaft.

The Association of Electrical Equipment and Medical Imaging Manufacturers [NEMA](#) defines standards that many motor manufacturers use for mounting geometry.

Resource R6 Sourcing and costs

Resource R7 Setting up the C Development Tool for myRIO

Box 00.2 setting up a lab PC or one's own laptop?

For configuring your own laptop, complete all the steps, below.

For configuring a lab PC, complete steps 4 and 5 of Part A, then complete all remaining parts (Part B – Part F).

Box 00.3 myRIO connected?

Parts A, B, and C can be performed without connecting your laptop to one of the lab myRIOs. For Parts D, E, and F, a myRIO connection is required.

Do Parts A, B, and C just once, in the order shown.

Part A: Setting up the software environment

Follow these instructions to set up the C Development Tool for myRIO. Clicking on hyperlinks opens the appropriate websites in your browser.

1. Download and install LabVIEW 2015 myRIO Toolkit. (2700 Mb)




For Native Windows 8 or 10:

Download [myRIOToolkit2015](#).
Mount disk image. Then run `setup.exe`.





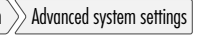

For Native Windows 7: Download from [myRIOToolkit2015](#). You may need a means of mounting the `.iso` disk image. For example, use [Virtual CloneDrive](#) to mount the `.iso` disk image files as a virtual CD-ROM drive. Then run `setup.exe`.






For Virtual Windows 7, 8, or 10 under Parallels:

Download

[myRIOToolkit2015](#) under OS X. From Parallels,    and mount the disk image.






Then run `setup.exe`.

2. Install Java. Visit the Java website [GetJava](#) to download Java. (17 Mb) Use Internet Explorer, not Microsoft Edge.
3. Install the C/C++ Development Tools for NI Linux Real-Time 2014, Eclipse Edition. Visit this link [Eclipse2014](#) to download and install Eclipse. (260 Mb)
4. Project templates have been prepared for each of the ME 477 laboratory exercises. Visit the ME 477 website [Resources Page](#) to download the ME477 myRIO support 2018 archive. Remember where you put this archive, but do not unzip. (2 Mb)
5. Eclipse uses Launch Configurations to specify how the project will be deployed and run on the myRIO. Visit the ME 477 website [Resources Page](#) to download the ME 477 Launch Configurations archive. Unzip into folder `LaunchConfig477` (40 kb). Remember where you put the folder.
6. Add the compiler path to the system environment variables.
 - a. Visit the ME 477 website [Resources Page](#). 64-bit compiler path file, select and copy with  +  the contents.
 - b. In the Windows Control Panel, select    to display the System Properties dialog box.
 - c. Click  to display the Environment Variables dialog box.

- d. Select PATH in the User variables group box and click . If PATH does not exist, click  to create it.
 - e. Click  and paste with  the compiler path to the end of Variable value (separated by the ; character). Be certain that there are no extra spaces in the path.
7. Click  to close the dialog box and save changes.

Part B: Define a connection to the myRIO

Complete the following steps to define a connection in Eclipse from your laptop to the myRIO target.

1. Launch Eclipse, specify a workspace, and click  to display the C/C++ perspective (default).
Two other perspective views, Remote Systems Explorer and Debug, will also be useful. To make these available, select  to display the Open Perspective dialog box.
Then select Remote Systems Explorer and click  to display the Remote Systems Explorer perspective. Repeat this process to display the Debug perspective. Buttons for all three perspectives should appear and can be used at any time to switch perspectives.
2. Open the Remote Systems Explorer perspective to display the Remote Systems pane at left.
3. Click the Define a connection to remote system icon  to display the New Connection dialog box.
4. Select .

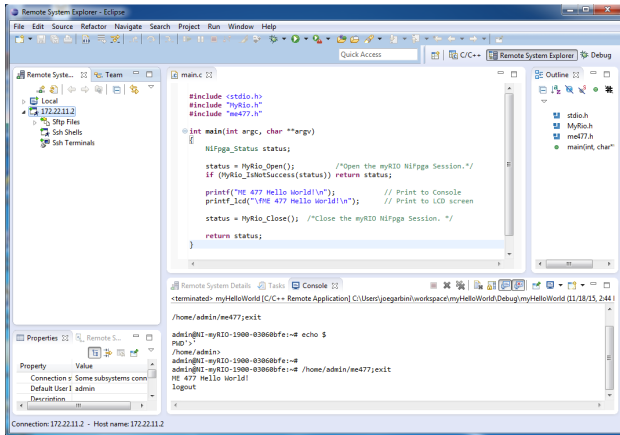


Figure 00.7: Remote Systems Explorer with myRIO connection successfully defined.

5. Enter the IP address 172.22.11.2 in the Host name textbox and click **Finish**. Your target displays in the Remote Systems tab in the Remote System Explorer pane, as shown in [Figure 00.7](#).

Part C: Importing C Support and Launch Configurations

Complete the following steps to import C Support and Launch Configurations to Eclipse.

1. From the C/C++ perspective, select **File** **Import** to display the Import dialog box.
2. Select **General** **Existing Projects into Workspace** and click **Next** to display the Import Projects page.
3. Select **Select archive file**, click **Browse** and select the ME 477 C Support for myRIO zip file downloaded in step 4 of Part A.
4. Ensure that all items are checked and click **Finish** to import ME 477 C Support for myRIO. See [Figure 00.8](#).
5. Build (compile) all projects with menu selection **Project** **Build All**.

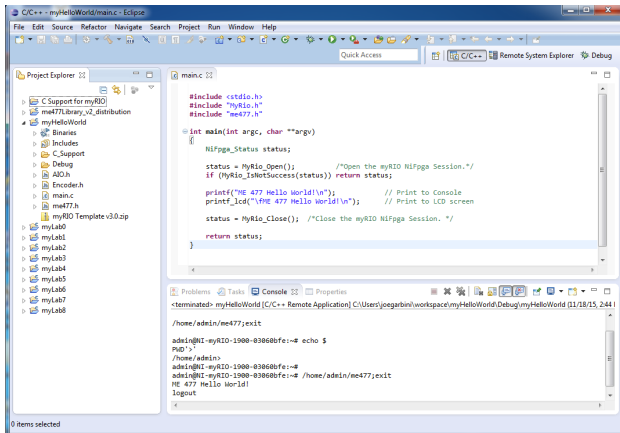


Figure 00.8: ME 477 Project Templates.

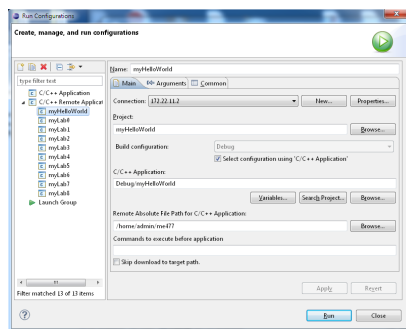


Figure 00.9: Launch Configurations.

6. Again, from the C/C++ perspective, select **File** > **Import** to display the Import dialog box.
7. Select menu item **Run/Debug** > **Launch Configurations** and click **Next** to display the Import Launch Configurations page.
8. Click **Browse** and select the LaunchConfig477 folder that you downloaded in step 5 of Part A.
9. Ensure that all items are checked and click **Finish**. To check that the import of the Launch Configurations was successful, select menu item **Run** > **Run Configurations...** and compare the dialog with **Figure 00.9**.

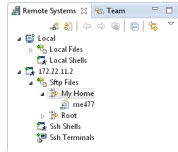


Figure 00.10: the Remote Systems tab should appear like this once a connection is established successfully.

Box 00.4 myRIO connected?

Your laptop must be connected through a USB cable to one of the myRIOs to perform Parts 4, 5, and 6. Each time you connect, a myRIO USB Monitor dialog box will appear indicating myRIO IP Address 172.22.11.2. Always select .

Part D: Connect to the myRIO target

Complete the following steps to establish a connection between Eclipse and the myRIO target.

1. In the Remote Systems pane, right-click the target and select from the shortcut menu to display the Enter Password dialog box.
2. Enter the user ID: admin and password: <UW: me477 | SMU: leave blank> and click .
3. Click in the Info dialog box.
4. If the Keyboard Interactive authentication dialog box appears, leave the password blank, and click . As shown in [Figure 00.10](#), green arrow appears on the target icon when the myRIO is connected.





Part E: Running the myHelloWorld project

In Parts 5 and 6 you will run and debug a project. Here, the myHelloWorld project is used

as example.

Eclipse uses a “Run Configuration” to specify how the project will be deployed and run on the myRIO. Run Configurations for ME 477 projects were downloaded in step 5 of Part A.

Complete the following steps to run the myHelloWorld example project.





1. In Eclipse, switch to the C/C++ perspective.
2. You can view and edit the C source code by double clicking on the myHelloWorld project in the left pane, and then double clicking on main.c.
3. In the Project Explorer pane, right-click the myHelloWorld project, and select  from the shortcut menu to build the project. Any build errors will be noted in the Problems pane.
4. Right-click the myHelloWorld project and select   to display the Run Configurations dialog box.
5. Select the myHelloWorld project in the left pane.
6. Click . The project runs on the myRIO target. You can find the result in the Console pane, and on the LCD screen.

Part F: Debugging the myHelloWorld project

Similarly, Eclipse uses a “Debug Configuration” to specify how the program will be debugged on the myRIO. Once the Debug Configuration for a project is set up, debugging the program requires just a single click.

Complete the following steps to set up the Debug Configuration for the myHelloWorld project. These include building, deploying, and debugging the project.

1. In Eclipse, switch to the C/C++ perspective.

2. In the Project Explorer pane, right-click the myHelloWorld project and select   to display the Debug Configurations dialog box.
3. Select the myHelloWorld project in the left pane.
4. Click . The project runs on the myRIO target within the debugger. Some warnings may appear in the Console pane. Under normal circumstances, these warnings are not a problem. You can find the debug tools on the toolbar of Eclipse. There will be more about this in the first laboratory exercise.
5. For now, try setting a breakpoint at the `printf()` statement by double-clicking in the margin at left of that statement. A blue dot with a small checkmark  should appear in the margin. The blue dot indicates that the breakpoint is enabled, and the checkmark indicates that the breakpoint is installed.
If you resume (green arrow) from the beginning of the program, execution should pause at the breakpoint, as shown in [Figure 00.11](#).

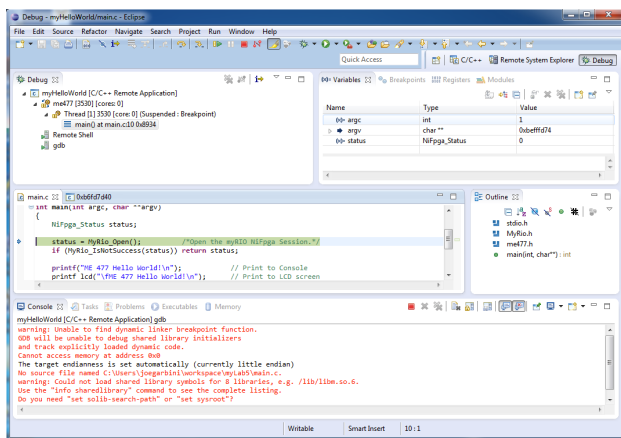


Figure 00.11: debugging and stopped at a breakpoint.

Resource R8 Suggested reading

The classic C programming language text Kernighan and Ritchie (1988), co-authored by Dennis Ritchie, who developed the language at AT&T Bell Laboratories between 1969 and 1973.

For computer hardware and software concepts, Patterson and Hennessy (2016) is a good introduction.

Part II

The User Interface

**Computing principles, myRIO C programming,
and high-level io drivers**