Lab Exercise 00 Getting started

Lab 00.1 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.

Lab 00.2 Pre-laboratory preparation

Read Resource 2 and Resource 7, following the instructions on your personal computer, preferably a laptop your 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.

Lab 00.3 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 00.6. 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.

Lab 00.3.1 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.

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:

```
/* 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 NiFpga. */
    return status;
}
int  sumsq(int x) {
    static int    y=4;

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

Figure 00.6: C code for Lab 00

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.