Department of Electrical and Electronic Engineering

ELEN90066 Embedded System Design

Workshop Two (W02)

Embedded Development Tools

Welcome to Workshop 2 for Embedded System Design. In this workshop you will learn about several different ways of programming an embedded system, a National Instruments myRIO 1900. The tasks in this workshop are prescriptive in nature and introduce you to common tools for programming embedded systems. Completing these tasks will not make you an expert in any one of the tools used – each of them is in fact the entire subject matter of one or more textbooks. Focus instead on efficiently extracting useful information from technical documentation, understanding the connection between the tool and your embedded hardware, and becoming familiar with the tools used in later workshops.

Pre-workshop background

In this workshop you will be programming the myRIO in two different ways - using C in an Integrated Development Environment (IDE) called Eclipse, and graphically using VIs in LabVIEW.

Familiarise yourself with the myRIO 1900 User Guide and Specifications before attending the workshop.

Workshop Exercises

The following tasks reference files that are distributed on the LMS as a `.zip` file. You need to download and extract these files to a folder with write permissions in order to complete this workshop. All file paths given are with respect to the top-level folder of the downloaded archive file.

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite workshops (if applicable):

- Computer with Windows 10
- National Instruments LabVIEW 2018 myRIO Toolkit

- National Instruments myRIO 1900
- 12V DC power supply
- USB cable
- C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition 2017-2018

In this workshop you will do the following:

- Discover and configure the myRIO device **(TASK 1 - 5 marks)**
- Execute a binary file on the myRIO that was compiled from C code to display "Hello, World!" on the Eclipse IDE console and toggle an onboard LED on the myRIO board. **(TASK 2 - 10 marks)**
- Modify and run a LabVIEW VI to compile an application that illuminates onboard LEDs when acceleration on the corresponding axis of the onboard accelerometer is above a certain threshold, download it to the myRIO, begin execution, and display debugging information to the front panel. **(TASK 3 - 10 marks)**

There are 25 marks available for successful completion of all tasks in this workshop, worth 1% of your final mark in the subject.

Task 1: Configuring the myRIO

1. **Connect myRIO:** Connect a power supply to myRIO. When powered, the Power LED on the controller will illuminate. Connect myRIO to your computer using a standard USB cable.

2. **Run the Getting Started Wizard:** The **myRIO USB Monitor** should automatically open when you connect myRIO to your computer. From within the myRIO USB Monitor, press Launch Getting Started Wizard. If the NI myRIO USB Monitor does not automatically open, you may launch the Getting Started Wizard from the LabVIEW myRIO start screen by selecting "Set Up and Explore→Launch the Getting Started Wizard". Complete the wizard as follows:

   (a) In the myRIO Discovery pane, verify that your myRIO device appears as discovered. Click the device to continue to the next step.

   (b) In the Rename your myRIO pane, optionally rename your myRIO device. Press Next to continue.

   (c) If this is the first time the myRIO device has been configured, the wizard will install software to the device.

   (d) In the Test Onboard Devices pane, verify that the onboard accelerometer, LEDs, and buttons are active and respond as expected. Press Next to continue.

   (e) In the final pane, you may choose to create a new project or open LabVIEW. You may also press Close to complete the wizard.

3. **Discover myRIO using Measurement and Automation Explorer:** The Measurement and Automation Explorer (MAX) may be launched from the Windows Start menu by searching for "NI MAX".

   In the left navigation pane, expand Remote Systems to autodiscover National Instruments devices on the local network. Autodiscovery may take some time to complete. If your myRIO appears under Remote Systems, it has been discovered. Click on the myRIO device to see detailed information about the target, including the IP address of the Ethernet over USB adapter and some of the interfaces that have been configured.

TASK 1 Show that you have successfully connected the myRIO to the NI MAX software and can read the device's IP Address.

Note : Your demonstrator will assess this task once you have finished it.

There are 5/25 marks for successful completion of this task.

Task 2: Program the myRIO Processor from Eclipse

This workshop task serves as a tutorial on programming the myRIO processor using C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition, an Eclipse-based Integrated Development Environment (IDE). You will configure your development environment, build a "Hello World" application in C, and connect to and program the ARM processor in Eclipse.

Eclipse from the Eclipse Project is the most widely-used open-source IDE, having been adapted as the IDE of choice for many hardware vendors. Like most IDEs, Eclipse provides a workspace to organize project files and development perspectives, a project to organise source code and dependencies, and build specifications that define communication with hardware targets.

Like most IDEs, Eclipse provides a *workspace* to organise project files and development perspectives. A *project* organises source code, dependencies, and build specifications that define communication with hardware targets.

1. **Setup myRIO:** Ensure myRIO is powered and connected to your computer.
2. **Enable SSH:** Your computer communicates with myRIO using the SSH protocol. This allows you to see `printf()` messages transmitted from myRIO to your computer, and to interact with myRIO using standard Linux shell commands. You may use NI MAX to enable terminal access:
   (a) You should see the connected myRIO device under the Remote Systems item in NI MAX. Click on it and ensure that under the Startup Settings section, the options "Enable Secure Shell Server (sshd)" and "Enable Console" are checked.
   (b) Click the Save icon at the top to apply the new setting.
   (c) Click the Restart icon at the top to restart myRIO.
3. **Launch Eclipse:** C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition may be found in the Windows Start menu under "National Instruments→C & C++ Development Tools for NI Linux Real-Time 2017, Eclipse Edition". Upon launching, you may be asked to select a workspace. A workspace organises projects, perspectives, and application preferences. Select a location to store your workspace – Eclipse will create a workspace at this location if none exists. The location must be writeable.

   If you are not prompted to select a workspace, then Eclipse has been configured to automatically load a default workspace. If you need to change to a new location, then from the top menu bar select "File→Switch Workspace".
4. **Import the Template Project:** The myRIO project template specifies the compile toolchain used to compile programs for myRIO; the *cross-compile* toolchain executes on your desktop and compiles your program for the ARM processor on myRIO. The project further specifies a run specification to execute the successfully compiled program.

To import the myRIO project template, from the top menu bar select "File→Import..." which will launch the Import wizard:

(a) In the Import panel, select "General→Existing Projects into Workspace" and press Next > .

(b) In the Import Projects pane, press "Select archive file" and browse to the `src\Eclipse` folder to find the file

`C_Support_for_NI_myRIO_v1.0.zip`

Several projects are available for import; the example projects are a useful resource but not needed for this workshop, so deselect all example projects, leaving only **C Support for NI myRIO** and **myRIO Template**.

Press Finish to complete the import wizard.

5. **Rename the Template Project:** In the Project Explorer view, right-click on the project **myRIO Template** and select "Rename...". Rename the project **myRIO Hello World**.

6. **Set Model Number:** Set a project-level symbol to specify which model of myRIO you are using. This informs the generated C code about the pin configurations and peripherals. Right-click on the **myRIO Hello World** project and select "Properties" to open project settings. Navigate to "C/C++ General→Paths and Symbols" and open the #Symbols tab to see project-level symbols that are defined across all source files in the project.

   There are a number of symbols that are built-in for compiler, operating system, and Eclipse settings – you may wish to uncheck "Show built-in values" to show only symbols you have defined. The symbol `MyRio_1900` should be defined by default to instruct the compiler to build for the correct hardware version. If it is not defined, make sure you define it by clicking on the Add... button.

7. **Set Compiler Options:** We need to make sure that the correct compiler is selected for the given architecture.

   First, the compiler options have to be unhidden by going to "Window→Preferences→C/C++ →Property Pages Settings" and making sure the "Display "Discovery Options" page" is selected. Make sure to apply your changes on every edit by clicking Apply before clicking OK.

   (a) Right click the project and go to properties "C/C++ Build → Discovery Options". Change the Compiler invocation command

   from: `arm-none-linux-gnueabi-gcc`

   to: `arm-nilrt-linux-gnueabi-gcc`

   Note: If your screen doesn't show the Compiler invocation command option, click on "Settings" then click on "Discovery Options" again. Magic!

(b) Still under Project properties, go to "C/C++ Build → Settings" and under the Tool Settings tab select Cross Settings. Change the prefix to `arm-nilrt-linux-gnueabi-` the path to

`C:\build\17.0\arm\sysroots\i686-nilrtsdk-mingw32\usr\bin\arm-nilrt-linux-gnueabi`

(c) Within the "C/C++ Build → Settings" page of the same properties view, under the Tool Settings tab, select "Cross GCC Compiler → Miscellaneous" and add the following to the list of other flags:

`--sysroot=C:\build\17.0\arm\sysroots\cortexa9-vfpv3-nilrt-linux-gnueabi`

(d) Within the "C/C++ Build → Settings" page of the same properties view, under the Tool Settings tab, select "Cross GCC Linker → Miscellaneous" and add the following to the list of linker flags:

`--sysroot=C:\build\17.0\arm\sysroots\cortexa9-vfpv3-nilrt-linux-gnueabi`

To verify the project settings are correct and that the project can be built, from the top menu bar select "Project→Build All". Watch the Console to verify all projects build successfully. If not, go over ALL steps again carefully.

8. **Create a Remote Target:** To view connections to remote systems, use the Remote System Explorer (RSE) perspective. To open this perspective, from the top menu bar select "Window→Open Perspective→Other" and choose "Remote System Explorer".

   From the top menu bar select "File→New→Other..." to launch the New Connection wizard:

   (a) In the "Select a wizard" panel, select "Remote System Explorer→Connection" and press Next >.

   (b) In the Select Remote System Type panel, select "Linux" and press Next >.

   (c) In the Remote Linux System Connection panel, enter the IP address of your myRIO into the Host name field and the name "myRIO" into the Connection name field and press Next >.

   (d) In the Files panel, choose the configuration "ssh.files" to indicate the filesystem connection protocol. Press Next >.

   (e) In the Processes panel, choose the configuration "processes.shell.linux" to indicate the Shell process subsystem for connecting to remote processes. Press Next >.

   (f) In the Shells panel, choose the configuration "ssh.shells" to indicate the shell and command protocol. Press Next >.

   (g) In the Ssh Terminals panel, choose the configuration "ssh.terminals" to indicate the terminal protocol.

(h) Press Finish to complete the New Connection wizard.

You should now see your myRIO target appear in the Remote Systems pane as shown in Figure 1.

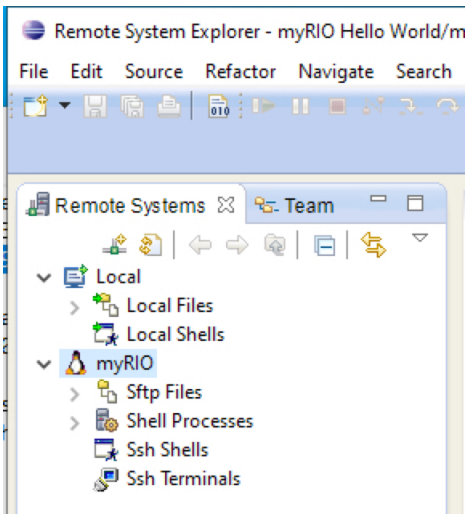

Figure 1: Remote System Explorer perspective

9. **Connect to myRIO:** From the Remote Systems pane, right-click on your myRIO target and select "Connect". Confirm the IP address, and enter "admin" and leave the password blank. Choose to save the username and password for convenient reconnecting as shown in Figure 2. Press OK to connect.

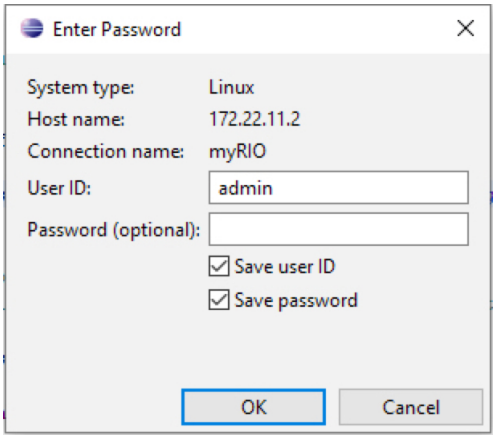

Figure 2: Remote system connection configuration

You may receive a security warning that the authenticity of the target could not be independently verified as shown in Figure 3. You may disregard this warning and click "Yes".

Figure 3: Security warning

When connected, a green arrow should appear over the target icon in the Remote Systems panel. You can verify the target is connected by right-clicking on the target icon – if "Disconnect" appears in place of "Connect" in the context-menu, then your target is connected.

Next, launch a terminal window which will allow you to interact with the Linux shell on myRIO. Expand the myRIO target, right-click on Ssh Terminals, and select "Launch Terminal". A terminal pane should appear in the bottom part of the perspective. Standard output messages are displayed on this terminal, and you may interact with the shell as with any Linux machine. Enter the command

```
uname --all
```

to display information about the Linux operating system on myRIO as shown in Figure 4

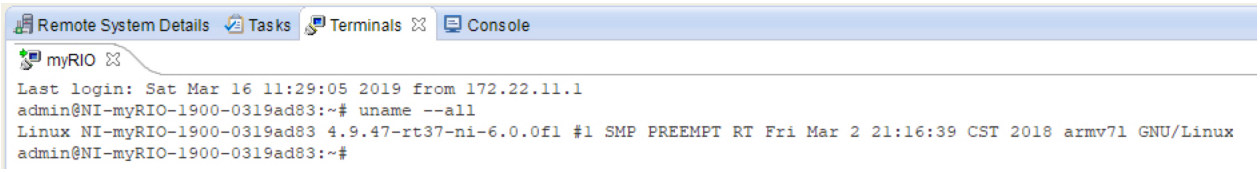

Figure 4: Terminal command running on myRIO that displays Linux kernel information

10. **Transfer the FPGA Fixed-Personality to myRIO:** The FPGA on myRIO may be configured with an *FPGA fixed-personality* stored in a LabVIEW bitfile (.lvbitx) file. In the Remote Systems pane, expand "myRIO→Sftp Files" and then "Root" to see the filesystem on myRIO. Navigate to the folder

    ```
    /var/local/natinst/
    ```

    If there is a sub-folder called "bitfiles" under this folder, click on it and check to see if the file `NiFpga_myRio1900Fpga10.lvbitx` exists. If it does, skip to the next instruction.

    Otherwise:

(a) Right-click on the folder "natinst" and select "New→Folder". Name the folder "bitfiles".

(b) Right-click on the newly created "bitfiles" folder and select "Export from Project...".

(c) Navigate to the source folder

```
C Support for NI myRIO\source
```

and check to include `NiFpga_myRio1900Fpga10.lvbitx`. No other files or folders should be checked.

(d) Verify that the destination folder indicates the "bitfiles" folder. Click Finish to transfer the file, and verify the file now appears in the Remote Systems pane under the "bitfiles" folder.

When your application executes on the target, the myRIO initialisation code will open the correct bitfile and program it to the FPGA.

11. **Write a Hello World Message:** Most programming tasks are performed in the C/C++ perspective. To open this perspective, from the top menu bar select "Window→Open Perspective→Other" and choose "C/C++".

    Expand the **myRIO Hello World** project and open the file **main.c** Add the statement

    ```
    printf("Hello, World\n");
    ```

    after the comment `/* Your application code goes here.*/`.

    To compile the source code, from the top menu bar select "Project→Build Project". This will produce a binary file that can run on the myRio processor.

    The Problems tab at the bottom of your workspace will show the results of the compilation – if there are errors (hopefully just typos), you must resolve them before continuing to the next step.

12. **Create a project folder on the myRIO:** In the Remote Systems pane, expand "myRIO→Sftp Files" and then "My Home" to see the contents of the home folder on the myRIO. If a "W02" subfolder exists then right click and delete it. Then create a new "W02" folder by right clicking and selecting "New→Folder".

13. **Create a Debug Configuration:** Debug configurations are used to build your application, download the executable to the remote target, and attach Eclipse to a remote debugging session. Edit the template configuration by navigating to the top menu bar "Run→Debug Configurations...". In the Debug Configurations dialog, expand "C/C++ Remote Application" and open the "myRIO Template" configuration. Configure the debug configuration as follows:

(a) Set the Name to "myRIO Hello World".

(b) In the Main pane:

i. Select the application to run by pressing Search Project... under C/C++ Application. If your program compiled successfully, the project **myRIO Hello World** will appear in the Program Selection dialog. Select it and press OK.

ii. For Connection, ensure "myRIO" is selected.

iii. Set "Remote Absolute File Path for C/C++ Application" to (note: no spaces in filename)

`/home/admin/W02/myRIOHelloWorld`

(c) In the Debugger tab, ensure the GDB Debugger is set to

`arm-nilrt-linux-gnueabi-gdb.exe`

by browsing to the file under the path

`C:\build\17.0\arm\sysroots\i686-nilrtsdk-mingw32\usr\bin\arm-nilrt-linux-gnueabi`

(d) Press Apply followed by Close to save settings and close the configuration window.

14. **Run your code on the myRIO:** From the top menu bar, select "Run→Run Configurations..." to open the Run Configurations window. In the left navigation pane, select the debug configuration you created in the previous step, and press Run to upload and execute your code on the myRIO.

    If everything is working you should see "Hello, World" appear in the Console in the Eclipse IDE. Congratulations, you have just executed your first piece of code on an embedded system!

15. **Launch a Debug Session:** Sometimes things don't behave as expected and need to be debugged. With a debug session, you can step through your C code line by line as it is executing on the myRIO and track things such as variables, registers and memory addresses.

    From the top menu bar, select "Run→Debug Configurations..." to open the Debug Configurations window. In the left navigation pane, select the debug configuration you created in a previous step, and press Debug to launch a debug session. Eclipse may prompt to open the Debug perspective, which contains several useful debugging views. Click Yes to open the Debug perspective.

    The Console contains several views. To open the Remote Shell view, press the

    "Display Selected Console" button until the console displayed reads "myRIO Hello World [C/C++ Remote Application] Remote Shell".

    The debugger will stop at the first line of the `main()` function. **Note:** You may receive a list of warnings / errors, in particular mentioning `Cannot access memory at address`. These can be ignored.

To execute the first line, from the top menu bar select "Run→Step Over". Keep stepping over the instructions until the console displays the message, "Hello, World". To complete execution of your application, from the top menu bar select "Run→Resume".

If the debugger exits successfully and you see the printed message on the console, then your code has been correctly cross-compiled, downloaded to the target, and remotely debugged. Next, you will configure your project to connect to the hardware peripherals on myRIO.

16. **Write an I/O Application:** Modify **main.c** to access the hardware peripherals on myRIO. At the top of the source file, include the myRIO header, and revise the `main()` function to open turn on an LED. Add the line

    ```
    extern NiFpga_Session myrio_session;
    ```

    immediately following the `#include` statements, and add the following lines after your print statement:

    ```
    /* Set the LED register to 1, turning on LED 1 */

    status = NiFpga_WriteU8(myrio_session , DOLED30 , 0x01);

    /* Print error messages (if any) from the DIO write */
    MyRio_PrintStatus(status);
    ```

    The complete program is shown below:

    ```
    # include <stdio.h>
    # include "MyRio.h"

    extern NiFpga_Session myrio_session ;

    int main(int argc, char **argv)
    {
      NiFpga_Status status;
      status = MyRio_Open();
      if (MyRio_IsNotSuccess(status))
      {
        return status;
      }

      /*
      * Your application code goes here.
      */
    ```

```
  printf("Hello , World\n");

  /* Set the LED register to 1, turning on LED 1 */
  status = NiFpga_WriteU8(myrio_session , DOLED30 , 0x01);

  /* Print error messages (if any ) from the DIO write */
  MyRio_PrintStatus(status);

  status = MyRio_Close();

  return status ;
}
```

Compile, download, and run your application on the target. If LED 1 illuminates, and you receive no error messages on the console, the program executed correctly. Congratulations on using Eclipse to build your first C application for myRIO!

Troubleshooting: *The Eclipse Problems pane shows an unresolved symbol, but the Console output indicates a successful build:*
If the Eclipse Problems pane indicates an error that a symbol (such as `DOLED30`) cannot be resolved, this is actually a problem with the built-in Code Analyzer. Open the Console pane to see the output of the cross-compiler – if it indicates the program was built successfully, then you may ignore the unresolved symbol warning in the Problems pane.
To prevent the Code Analyzer from falsely reporting this problem, you may disable the feature by right-clicking on your project, selecting Properties, navigating to "C/C++ General→Code Analysis", selecting "Use project settings" and unchecking "Syntax and Semantic Errors→Symbol is not resolved".

TASK 2 Show that you have successfully executed the code to print the Hello World message on the console in Eclipse and flashed the LED on the myRIO.
Note : Your demonstrator will assess this task once you have finished it.
There are 10/25 marks for successful completion of this task.

Task 2: Program the myRIO Processor from LabVIEW

This workshop task serves as a tutorial on programming the ARM processor on myRIO using National Instruments LabVIEW. You will build a acceleration indicator application that turns on LEDs when acceleration is exceeded on each axis of the onboard accelerometer.

Model-based design emphasises mathematical modelling to design, analyse, verify, and validate dynamic systems. Mathematical models are used to design, simulate, synthesise, and test cyber-physical systems, and are based on system specifications and analysis of the physical context in which the system resides.

Graphical design environments provide developers with a higher level of abstraction than traditional imperative programming models. National Instruments **LabVIEW** is a graphical design environment for scientists and engineers that emphasizes model-based design of embedded and cyber-physical systems. LabVIEW applications run on desktop computers or embedded controllers, and are most commonly designed to interface with sensors, actuators, instruments, data acquisition devices, and other computers and embedded devices.

LabVIEW supports heterogeneous compositions of several models of computation: continuous systems are expressed as ordinary differential equations or differential algebraic equations, and discrete systems are expressed as difference equations, in the Signal Flow model of computation; concurrent state machines are expressed in the Statecharts model of computation; imperative expressions are expressed as Formula Nodes (a subset of ANSI C) or MathScript Nodes (compatible with scripts created by developers using The Mathworks, Inc. MATLAB software and others); data acquisition and program flow are expressed in **Structured Dataflow**, the predominant language in LabVIEW. Structured Dataflow is often referred to as the **graphical language (G)**.

A LabVIEW application is called a **virtual instrument (VI)**. A VI consists of a **front panel** and a **block diagram**. The front panel provides an interface for setting parameters, executing a VI, and viewing results. The block diagram defines the behaviour of the application.

LabVIEW applications targeting the processor on myRIO are compiled into machine code and executed within an RTOS.

1. **Setup myRIO:** Ensure myRIO is powered and connected to your computer.
2. **Launch LabVIEW:** LabVIEW may be found in the Windows Start menu under "National Instruments→NI LabVIEW 2018 SP1 (32-bit)". Upon launching, you will see the Getting Started dialog which links to recent files, projects, and help documents.
3. **Create a Project:** From the top menu of the Getting Started dialog, select "File→Create Project..." to launch the Create Project Wizard. Navigate to "Templates→myRIO→myRIO Project" and select Next.

In the "Configure the new project" dialog, name your project **Acceleration Threshold** and choose a location to save your project files. Your myRIO should automatically appear under the Select Target pane – if not, first verify that you can see your myRIO in NI MAX, and re-run the wizard.

Select Finish to complete the Create Project Wizard. The Project Explorer will automatically open with your new project. The Project Explorer is used to connect to myRIO and organise VIs that will run on the target processor.

4. **Run the Template VI:** Open the top-level VI that will execute on the processor by navigating in the Project Explorer and double clicking on "Project Acceleration Threshold.lvproj→ [name of myRIO device]→Main.vi". A template VI has been created that reads data from the onboard accelerometer and displays it on a graph. On clicking the Run button, the VI will be deployed to myRIO and run (Save any files that are prompted to be saved). Verify that the accelerometer data is shown on your screen.

5. **Modify the Template VI:** In the block diagram of **Main.vi**, replicate the code shown in Figures 5 and 6:

   (a) Add a **LED Express VI** ("myRIO→Onboard→LED" in the palette). The Express VI will open a configuration dialog; uncheck LED3 and press OK to save the configuration.

   (b) Three `Greater?` nodes that will be used to compare accelerometer values to a threshold.

   (c) Wire the 'x' (top) input of the first `Greater?` node to the x output of the **Accelerometer VI**, and similarly connect the remaining two `Greater?` nodes to the y and z axes.

   (d) On the front panel, create a new Numeric Control and label it "Acceleration Threshold (g)". Set its initial value to 0.5. Right click on the control and from the context menu select "Data Operations→Make Current Value Default". On the block diagram, wire this control to the 'y' (bottom) input of each Greater? node.

   (e) Wire the output of the x axis comparator to the LED0 input of **LED VI**, and similarly connect the output of the y and z comparators to LED1 and LED2.

   (f) On the front panel, create three LED indicators. Label the first "x axis threshold", and similarly for the y and z axes. On the block diagram, wire these to the corresponding output of the comparators.

6. **Run the VI:** to compile your application, download to myRIO, begin execution, and display debugging information to the front panel. You should see the onboard LEDs illuminate when acceleration on the corresponding axis of the onboard acclerometer exceeds the Acceleration Threshold (g).

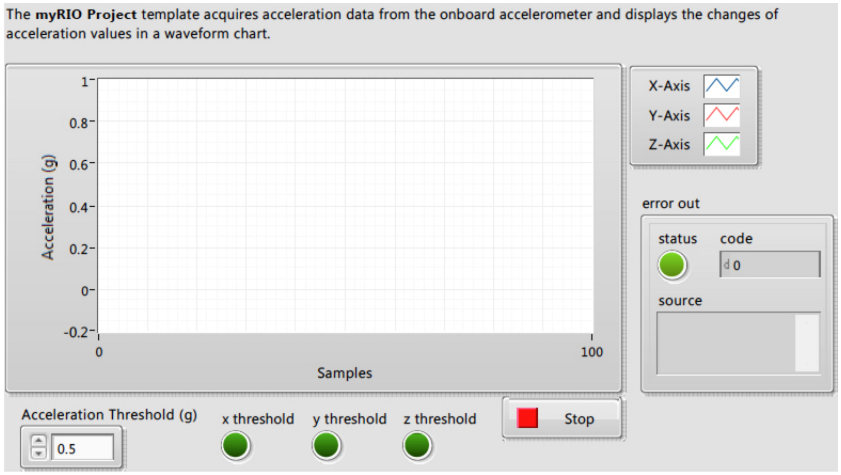

Figure 5: Threshold Indicator application front panel.

Congratulations on completing your first LabVIEW application for myRIO! To run your application again, simply press the Run button in the top toolbar.

TASK 3 Show the VI executing correctly on the myRIO by demonstrating that the onboard LEDs illuminate when the acceleration on the corresponding axis of the onboard acclerometer exceeds the Acceleration Threshold (g).

Note : Your demonstrator will assess this task once you have finished it.

There are 10/25 marks for successful completion of this task.

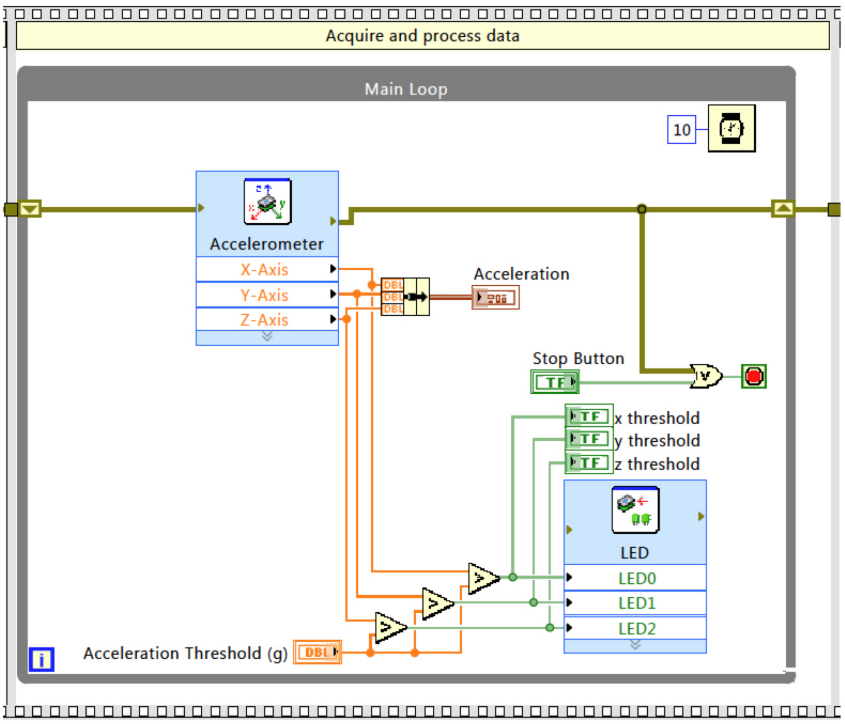

Figure 6: Threshold Indicator application block diagram (Main Loop only).