ECE 362 Lab Experiment 0: Getting Started

Introduction

In ECE362, students will learn fundamentals of microcontrollers, including their operation, and usage. In addition to the lecture portion of the course, students are expected to complete a series of lab experiments using a microcontroller platform and IDE. Computers and measurement equipment are provided in the course laboratory facilities to assist students in completing labs, however, the course has also been designed to allow students to perform experiment exercises on their own computers at home or elsewhere. Instructions provided in this lab document serves as a guide to setting up the microcontroller development environment used in ECE362 to provide a consistent user experience between home and the laboratory.

Instructional Objectives

- To become familiar with the contents of the ECE 362 lab kit

- To learn how to install the SystemWorkbench IDE on your personal computer
- To practice building a project, running a program on the microcontroller, and debugging execution

Table of Contents

When you are ready for your lab evaluation, review this checklist.

Step 0: Prelab Preparation:

- Read this entire lab document.
- Read the user manual for the STM32F091 development board.
- Do the prelab exercises and submit them before attempting the lab experiment.

Step 1: Place the development board on a breadboard

For this, and every other lab experiment, you must use your own lab kit. You must not share your microcontroller or other lab kit materials with other students. Sharing equipment, when discovered, will result in a zero score for the lab.

Open your ECE 362 lab kit and find the STM32F091RC microcontroller development board. It should look like Figure 1, below. It is a 60-pin board with double-ended pins so that the board can be used in a breadboard as well as used with header connectors that plug in on top. On one end of the development board, there is a 10-pin programming header.

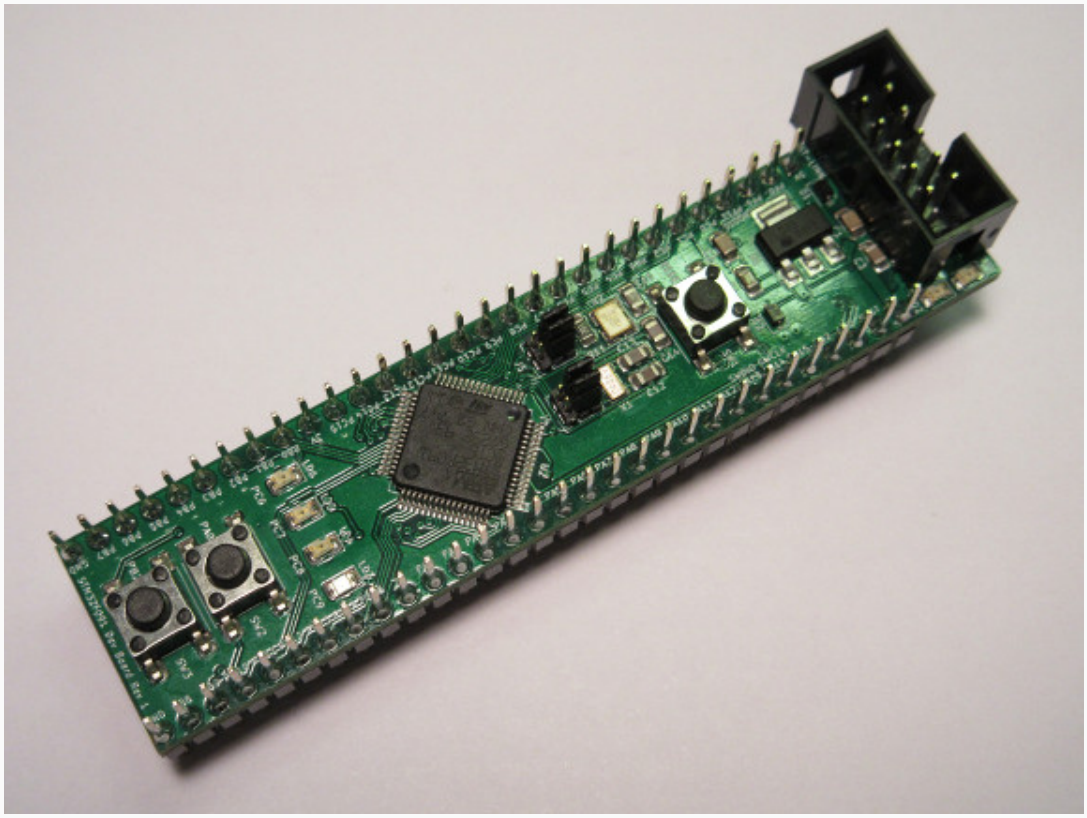


Figure 1: The ECE 362 STM32F091 development board

Your ECE Master Lab Kit has a four-panel solderless breadboard similar to this one. Read the user manual to learn how to insert the development board into the breadboard. It should be placed so that the programming header overlaps the power-distribution buses and two holes are available for use on either side of each

pin the development board as in Figure 2. Do not insert it into a panel at either end of the breadboard assembly. Leave a panel open on either side. You may wish to pre-widen the holes into which you will insert the development board. Be careful to press down only on the push buttons rather than the more delicate components on the board.

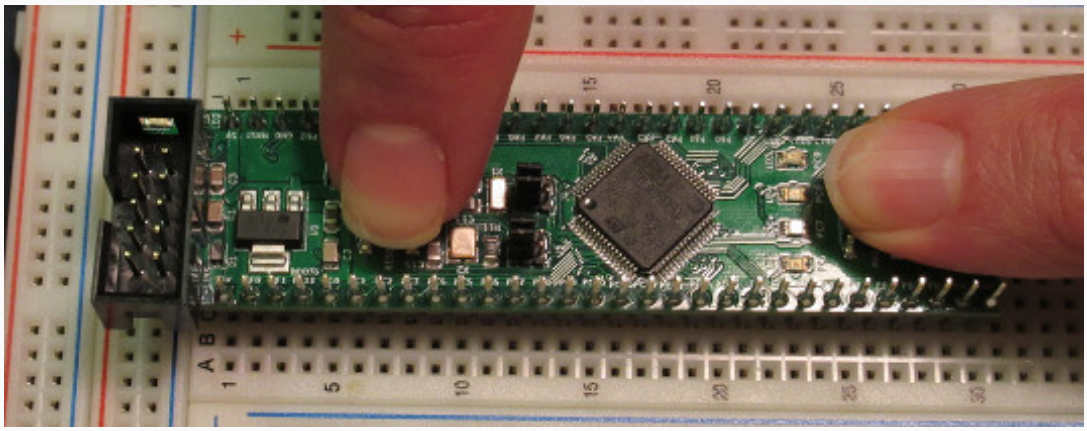


Figure 2: Inserting the development board

Consult the user manual to determine how to connect the USB programmer to the development board. If you have a programmer with the wrong pin configuration, you will need to connect four wires between the programmer and the development board in the required locations.

As long as the USB programmer pins are connected only to the development board programming header pins, there is little chance of damaging the board. **Do not connect the programmer directly to any of the long rows of 30 pins on either side of the development board. That could cause damage.** Once the programmer is connected to the programming header, plug the USB programmer into your computer. Use the 3-foot-long USB extension cable if you do not have a USB port near your development board. You should see the red and green LEDs in the corner of the development board illuminate. If you do not see these LEDs, unplug the USB programmer immediately and check your wiring.

Step 2: Install the development software

For embedded software development within ECE362, the open-source Eclipse IDE will be used, alongside the “System Workbench for STM32” Eclipse plugin (supported by STMicroelectronics). Full instructions for the installation of of this software is OpenSTM32 installation instructions, available on_the OpenSTM32 web site, however you will need to register for the site and log in to see it. While it is possible to install components individually, the lab staff finds that the environment works easiest and best if a pre-integrated system is installed. The following platform-specific instructions will allow you to do that.

2.1 Installation on Linux

The specific version of Linux supported is Ubuntu LTS, and this is the most reliable platform for SystemWorkbench. Other distributions may work, but they may involve additional steps not detailed here. Download the latest version of System Workbench from:
http://www.ac6-tools.com/downloads/SW4STM32/install_sw4stm32_linux_64bits-latest.run
Or use the locally-cached version:
https://engineering.purdue.edu/ece362/lab/install_sw4stm32_linux_64bits-latest.run
Then invoke the installation script with the following command:

```
bash install_sw4stm32_linux_64bits-latest.run
```

You will be prompted for installation locations as well as the system password to install the STlink driver configuration. **Keep**

track of the location in the file system that SystemWorkbench and Ac6 are installed. You will refer to them later. Depending on your particular computer, you may need to issue an additional command for the software to work properly:

- Ubuntu 18.04: **`sudo apt install lib32ncurses5 make`**
- Ubuntu 20.04: **`sudo apt install libpython2.7:i386 libncurses5:i386 make`**

The installation script should set up a link on your desktop by which you can invoke System Workbench. If not, invoke it with [the path to the installation directory]/eclipse.

2.2 Installation on MacOS

MacOS is the least reliable platform for SystemWorkbench, and many students who try to use it are unable to get it to install or cannot use it once it is installed. Please consult with other students on the Piazza forum to find solutions. A common solution for the installation problem is to install a virtual machine running Linux or Windows and only use SystemWorkbench in the VM. Download the latest version of System Workbench from:
http://www.ac6-tools.com/downloads/SW4STM32/install_sw4stm32_macos_64bits-latest.run
Or use the locally-cached version:
https://engineering.purdue.edu/ece362/lab/install_sw4stm32_macos_64bits-latest.run
Then invoke the installation script with the following command (in a terminal shell):

```
bash install_sw4stm32_macos_64bits-latest.run
```

You will be prompted for installation locations as well as the system password to install the STlink driver configuration. **Keep track of the location in the file system that SystemWorkbench and Ac6 are installed. You will refer to them later.** The installation script should set up a link on your desktop by which you can invoke System Workbench. If not, invoke it with [the path to the installation directory]/eclipse.

2.3 Installation on Windows

Windows is a reliable platform on which to use SystemWorkbench, but you you should avoid using project names with spaces in them. (e.g., do not create a project with the name "Project 1".) Download the latest version of System Workbench from:
http://www.ac6-tools.com/downloads/SW4STM32/install_sw4stm32_win_64bits-latest.exe
Or the locally-cached version:
https://engineering.purdue.edu/ece362/lab/install_sw4stm32_win_64bits-latest.exe
Then invoke the installer. You will be prompted for installation locations as well as the system password to install the STlink driver configuration. **Keep track of the location in the file system that SystemWorkbench and Ac6 are installed. You will refer to them later.**

2.3.1 STLink Driver Installation on Windows

The STM32F091 development board requires a programming interface, known as STLink, which can be used to program your microcontroller and other microcontrollers. In order to interact

with the STLink programmer for programming and debugging functions, an appropriate driver must be installed. Acquire the driver from STMicroelectronics, extract the resulting zip file, and install the driver.

3 Download the Standard Peripheral Library

We will use the OpenSTM32 Standard Peripheral Library for most projects in the class. You may not be able to download it from the host site in the next step. If not, download a locally-cached version:
https://engineering.purdue.edu/ece362/lab/stm32f0_stdperiph_lib_v150.zip
You must save the downloaded ZIP file into a particular subdirectory of the "Ac6" installed. The subdirectory is "***[The Ac6 installation location]***`/SW4STM32/firmwares/`". It must be saved with the file name "stm32f0_stdperiph_lib_v150.zip". Unzip the file in that subdirectory to create a new subdirectory named "STM32F0xx_StdPeriph_Lib_V1.5.0". SystemWorkbench will look for both the ZIP file as well as the unzipped version. Both must have exact names.

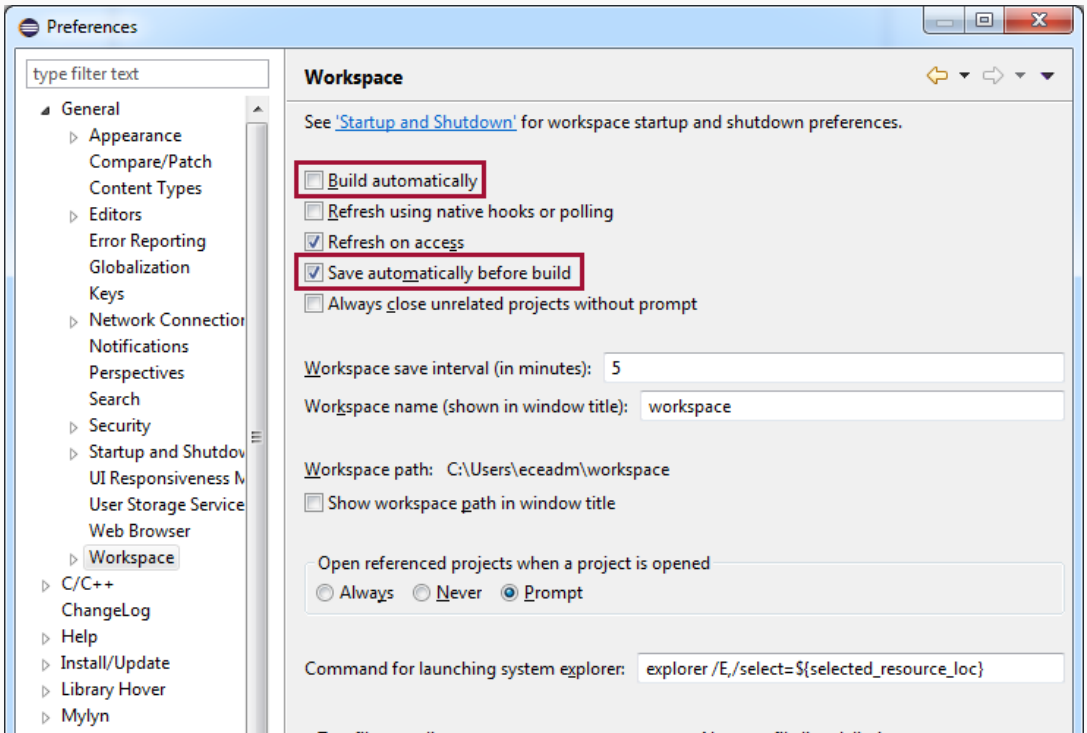
4 Configure SystemWorkbench

Now that all software tools needed for the STM32F0 have been installed, the Eclipse environment should be configured for student use. The workspace preferences configuration instructions are heavily based on a set of instructions from the GNU ARM Eclipse community. To access the Eclipse Preferences menu, select **Window >> Preferences** from the Eclipse main menu.

4.1 General Workspace Preferences

The first destination is Workspace Preferences, located at **General >> Workspace** in the Preferences menu. Uncheck the box which says “Build automatically” (in these experiments, the build process will be initiated via a command). Additionally, check the box titled “Save automatically before build”. **This is probably the most important setting you can change. Without it, you will be confused why the program you just modified does not act as you expect.**

Text encoding and newline delimiters can vary from operating system to operating system, and should be standardized for the purposes of this course. At the bottom of the Workspace preferences pane, there are two fields with radio buttons: “Text file encoding” and “New text file line delimiter”. Ensure that the Text file encoding field is specified as UTF-8, while the New text file line delimiter is specified to use Unix-style endings. (You should not use the "Default (Windows)" even if you are using Windows. Once all of these changes have been made, click Apply. All of these settings are summarized in Figure 3.



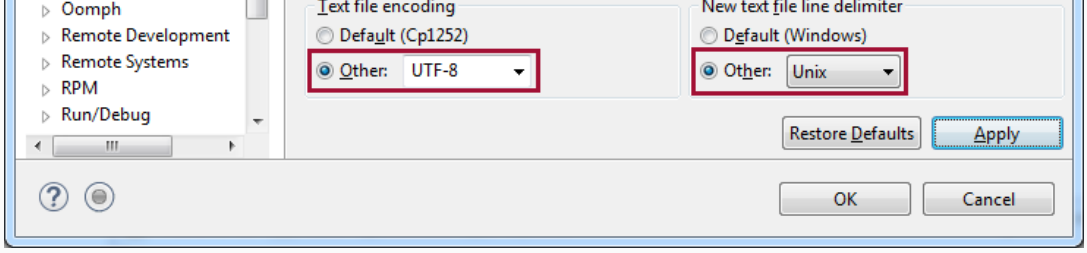
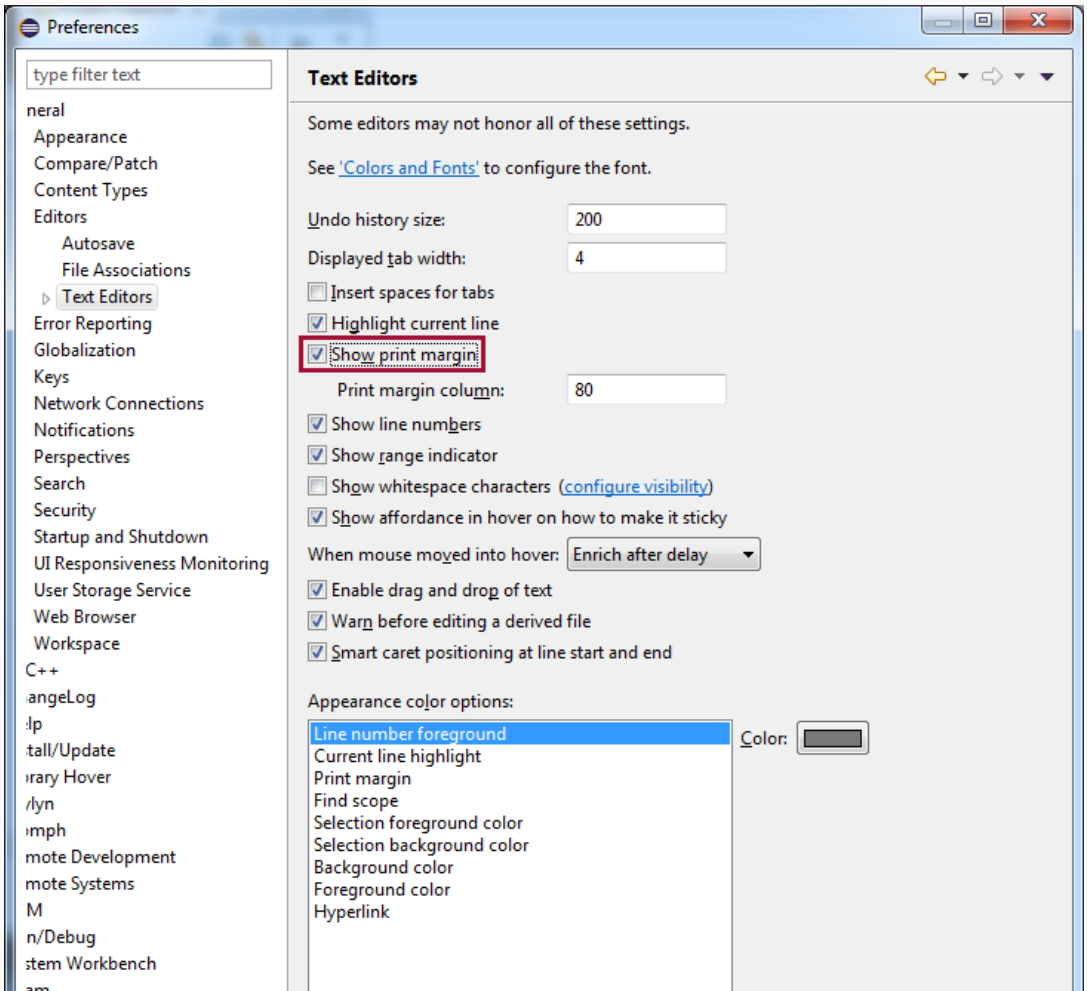


Figure 3: General Workspace Preferences Configuration

4.2 General Text Editor Preferences

Occasionally, source code will have to be printed. When this is done, it is helpful to know where the edge of a page would be, that lines of code can be written of appropriate lengths. Under the preferences hierarchy, navigate to **General >> Editors >> Text Editors**. In the Text Editors preferences pane, click the box labeled "Show print margin" and ensure that the "Print margin column" field is set to 80. Once these preferences have been made, click Apply.



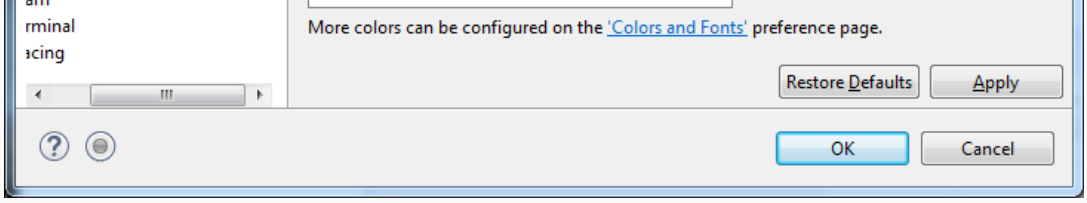
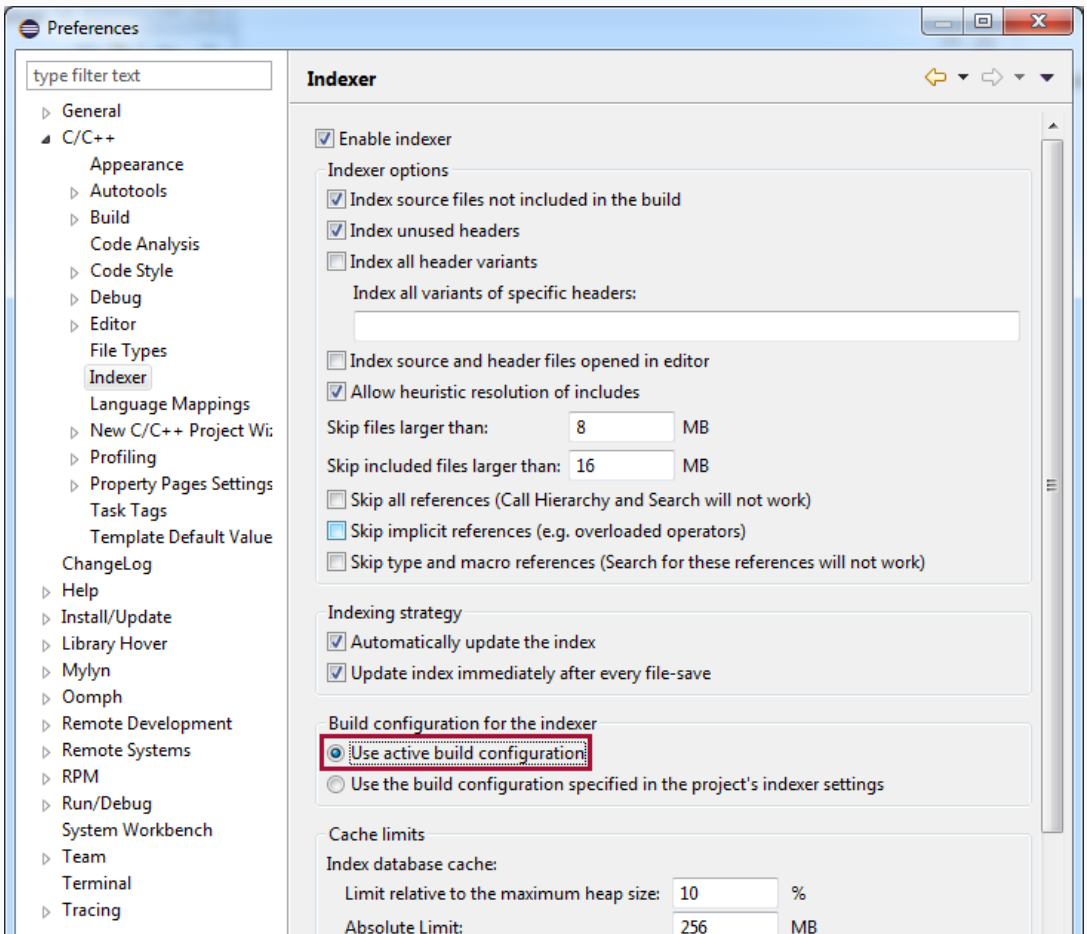


Figure 4: General Text Editor Preferences Configuration

4.3 C/C++ Indexer Preferences

Certain IDEs, including Eclipse, feature a software tool called an Indexer. An indexer parses source code on the fly, providing hints, auto-completion suggestions, and error reporting to the developer during the process of editing, without having to initiate the build process. In order for the indexer to provide accurate and relevant hints, the indexer must be in sync with the compiler.

In the Preferences menu, navigate to **C/C++ >> Indexer**. In the field titled "Build Configuration for the Indexer", select the field titled "Use active build configuration". Click Apply.



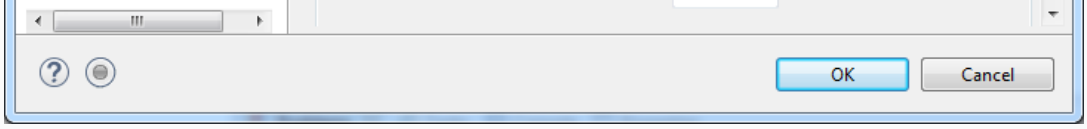


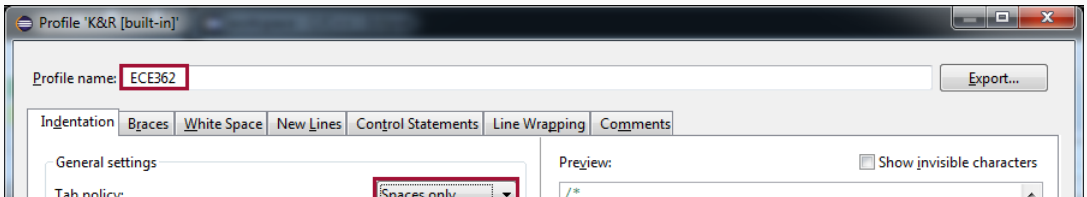
Figure 5: C/C++ Indexer Preferences Configuration

4.4 C/C++ Code Formatting Settings

Eclipse features a powerful integrated code formatter, which is capable of automatically formatting source code files to conform to a particular coding style of choice. Eclipse will automatically adhere to the settings of the code formatter when working on new source files. Alternatively, a file open in Eclipse can be automatically formatted using **Source >> Format** from the Eclipse main menu.

To modify the formatter settings, navigate to **C/C++ >> Code Style >> Formatter**. In the resulting pane, a drop down box will appear showing different code formatting styles, as well as a preview of what source code might look like in the resulting style. Select your style of choice, then click the Edit… button. A new window will appear, showcasing code formatter style configuration settings.

The built-in code formatter styles cannot be modified, so press New and a dialog window will open with a field labeled “Profile Name”. Provide a name for the new code formatting style (for the purposes of this exercise, the profile name “ECE362” was used). Choose "K&R C" as a template. To ensure uniformity of code across development environments, set the field labeled Tab Policy to “Spaces Only”, then click OK. You will be returned to the Eclipse Preferences menu. Click Apply.



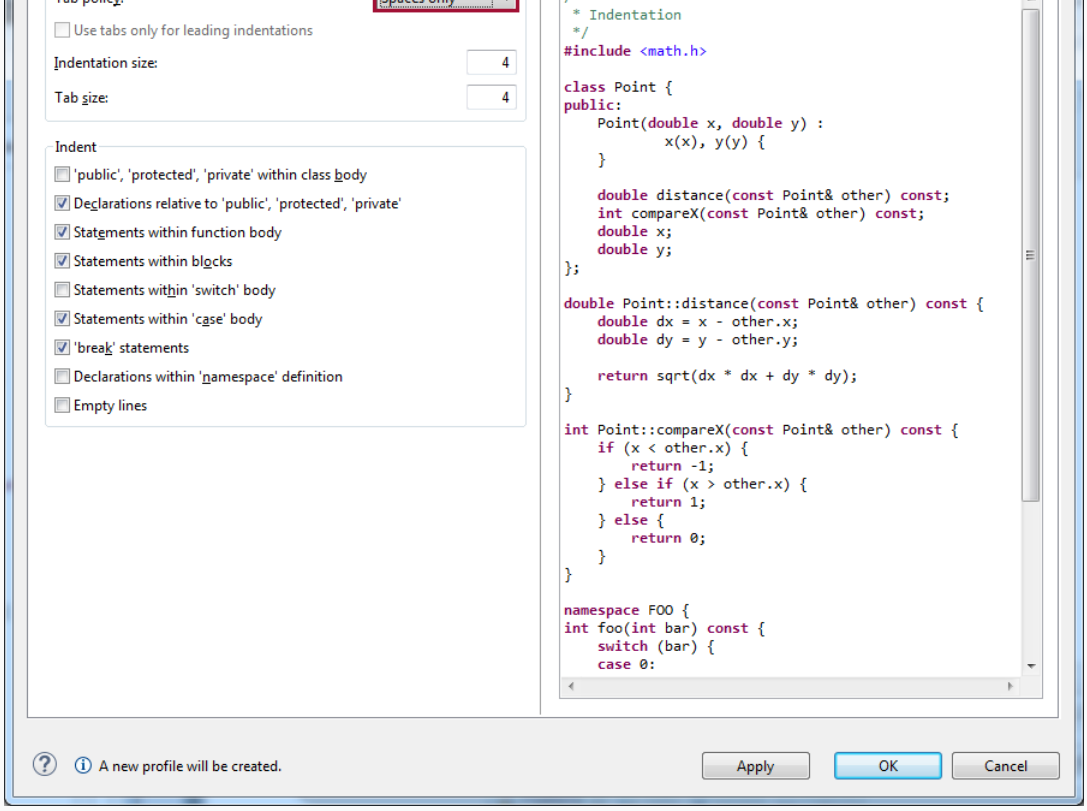
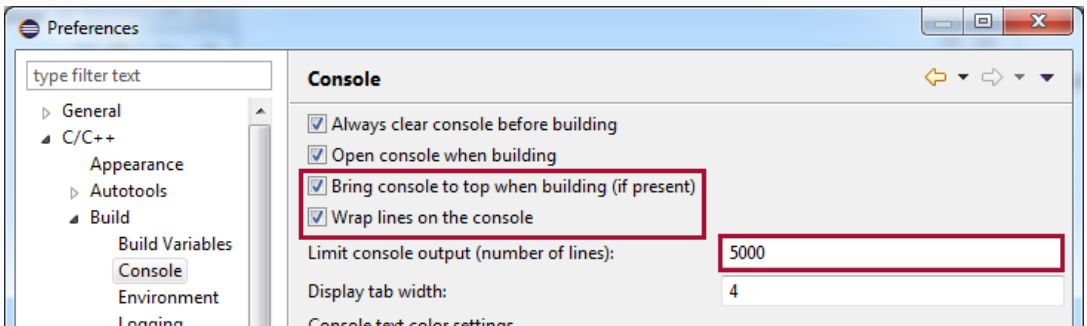


Figure 6: C/C++ Formatter Style Preferences Configuration

4.5 Build Console Settings

The build console is a portion of the Eclipse UI which is displayed during the build process. Build console settings can be found under **C/C++ >> Build >> Console** in the Eclipse Preferences menu.

To ease the build process, check the boxes in the Build Console Preferences pane stating "Bring console to top when building (if present)" and "Wrap lines on the console". Additionally, increase the field "Limit console output (number of lines):" to a higher value, such as 5000. Click Apply.



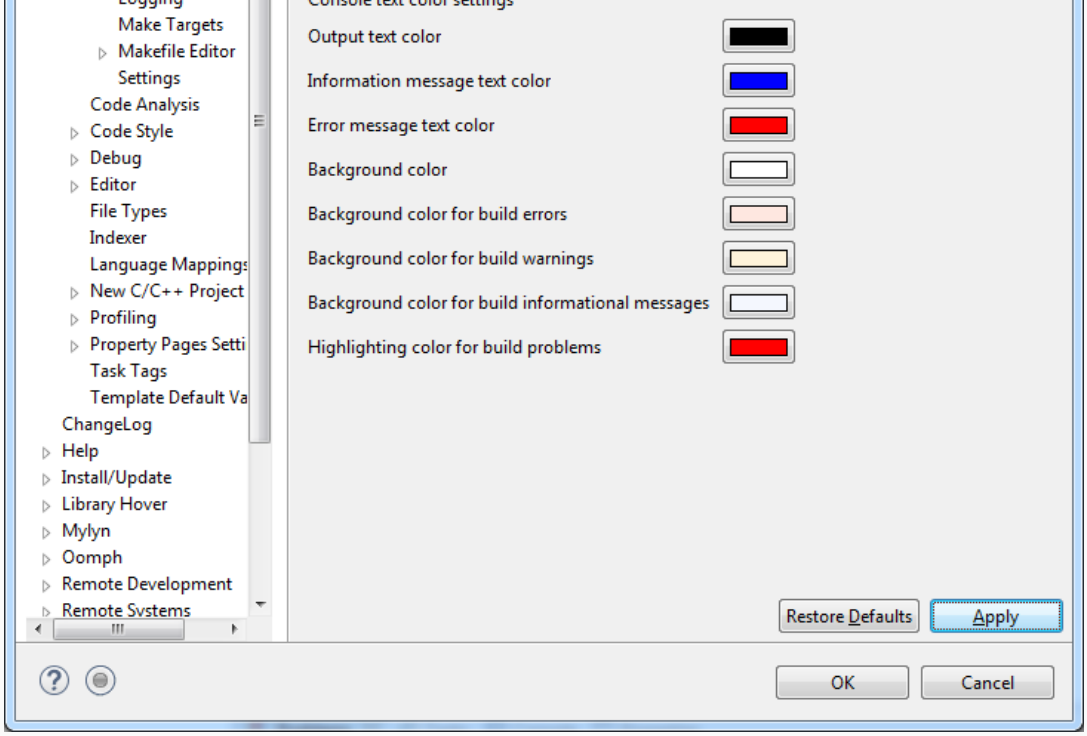


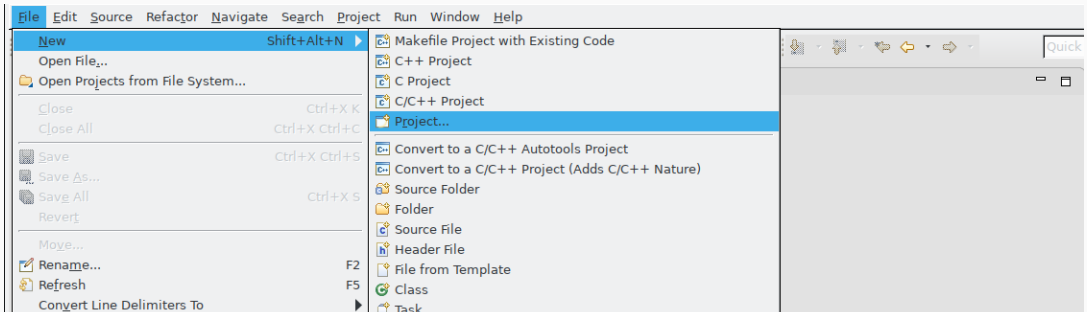
Figure 7: Build Console Preferences Configuration

5 Create a project in SystemWorkbench

Now that the software toolchain is configured, it's time to create a simple project for the STM32F0. Connect the STM32F091 development board to the USB programmer, and start SystemWorkbench. A Welcome Screen will appear, which can be closed. Creation of a project is a six-step incremental process of making selections in dialogs to configure the computer language used for a project, the type of microcontroller (target) to be used, and the support software to be used (firmware).

5.1 New project

In the top level Eclipse menu, select **File >> New >> C Project**.



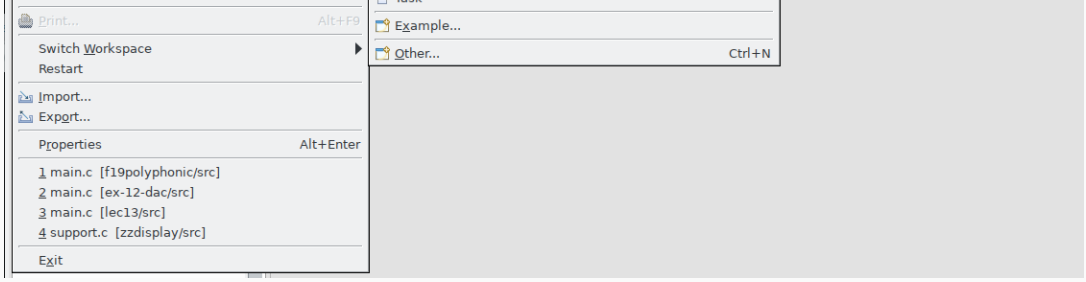


Figure 8: Create Project Menu

5.2 Select a wizard

Next, you will arrive at the "Select a wizard" dialog. For the first half of the semester, we will be writing files in assembly language, but we show you how to adapt a C environment to that. Select "C Project" and press **Next**.

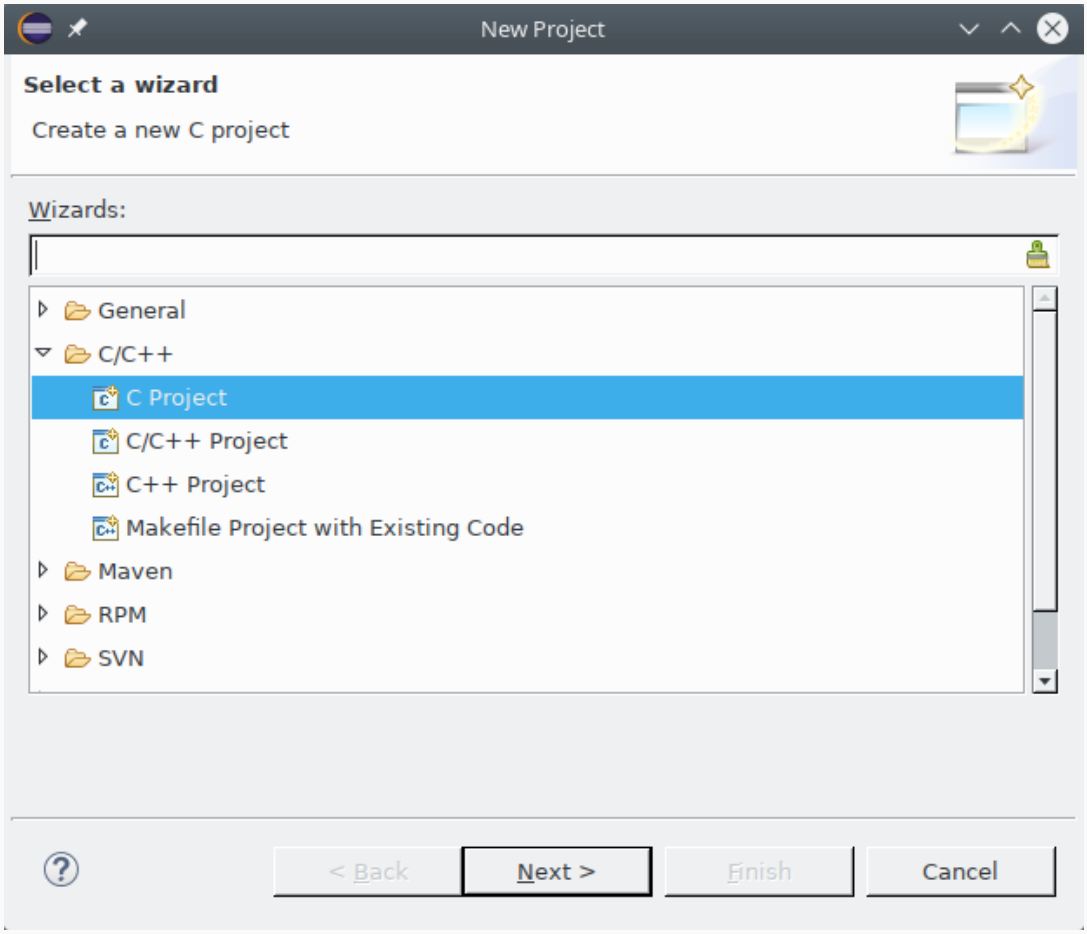


Figure 9: Select a wizard dialog

5.3 C Project dialog

The next dialog is the "C Project" dialog. Specify "Ac6 STM32 MCU Project", specify a Project name ("project1" was used, but you can pick something meaningful such as "lab0") and click **Next>**.

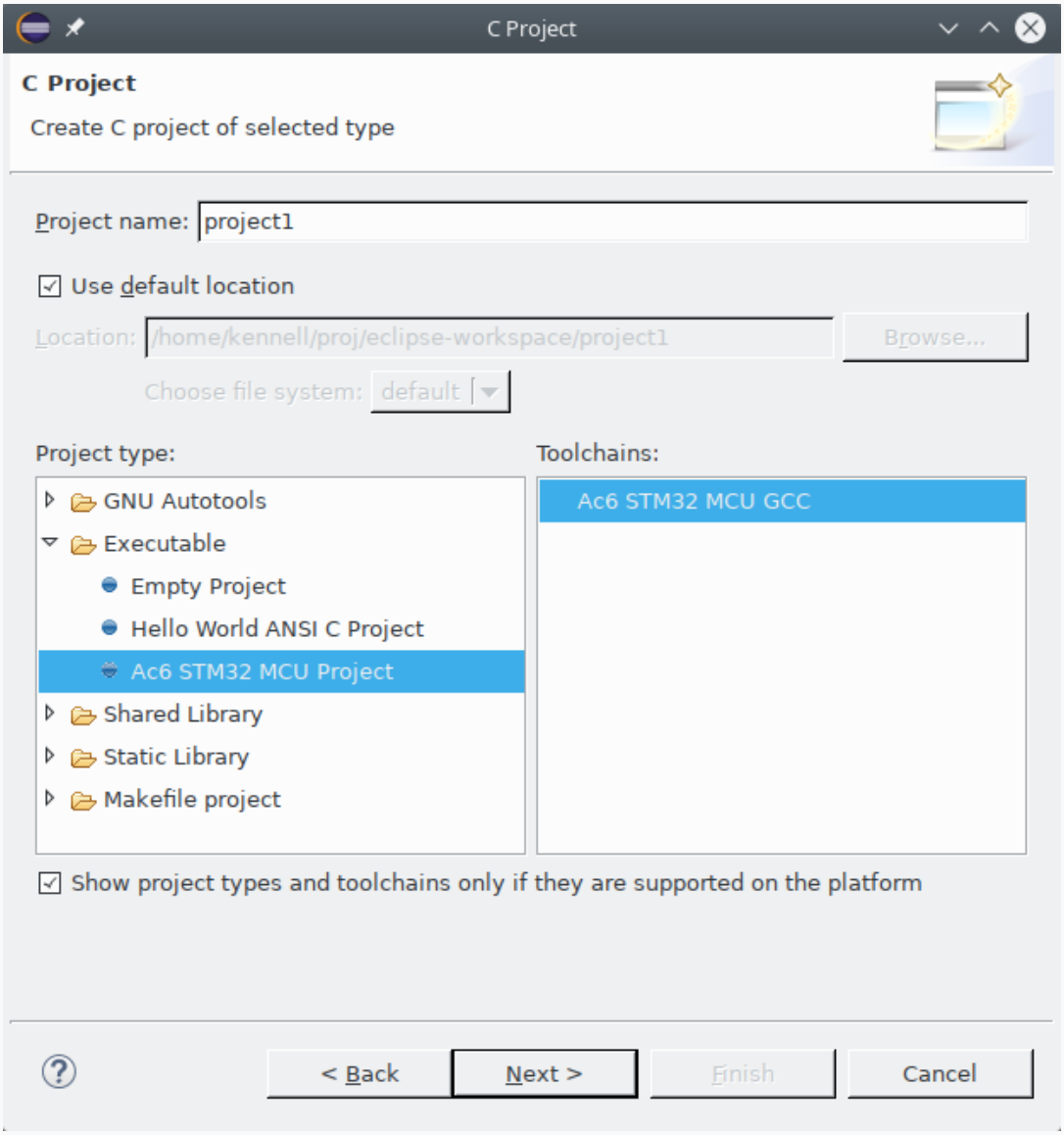


Figure 10: C Project dialog

5.4 Select Configuration

The next dialog is the "Select Configurations" dialog which allows you to select the build configurations to be used for the project. By default, it shows "Debug" and "Release". We will never use anything other than these, and we will only rarely use "Release". Just press **Next** to move on.

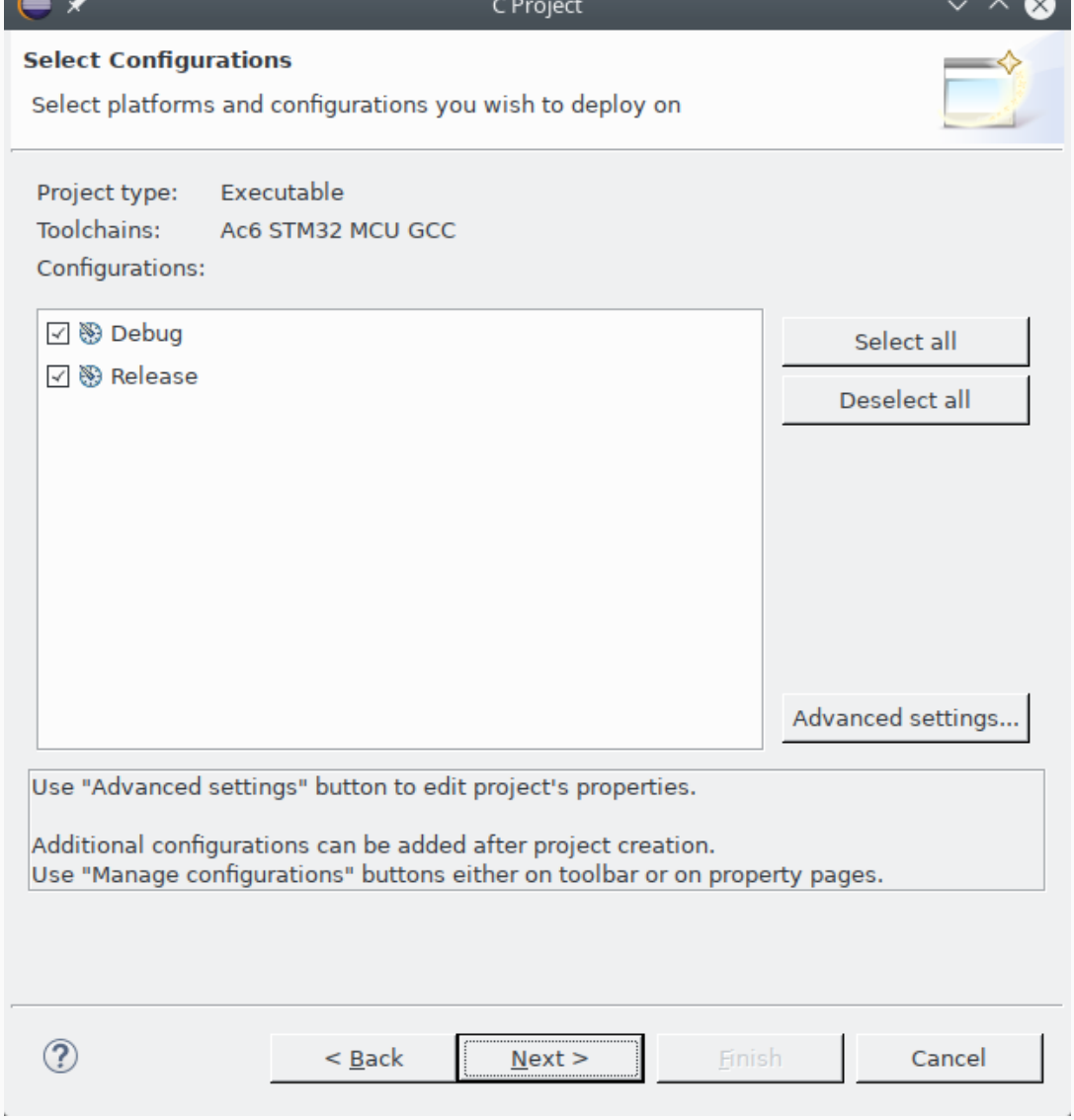


Figure 11: Select Configurations dialog

5.5 Target Configuration

The next dialog is the "Target Configuration" dialog. It selects the type of microcontroller to use. The microcontroller we will use for the class is the STM32F091RCT6. You should mentally split that up into four parts: STM32F0 91 RC T6. To find that with the dialog, **first choose the "Mcu" tab at the top of the dialog**. The "Series" is STM32F0 (the default), and the "Mcu" will be found near the bottom of the drop-down list. There is no specific entry for "STM32F091RCT6". Instead, you should use the slightly more generic "STM32F091RCTx". You might temporarily select other microcontrollers on the list and see how it affects the parameters

in the box in the center. Make sure you set it back to "STM32F091RCT6" and then press **Next**.

This is a tedious step, and we'll try to find a way to make this just a little easier to select.

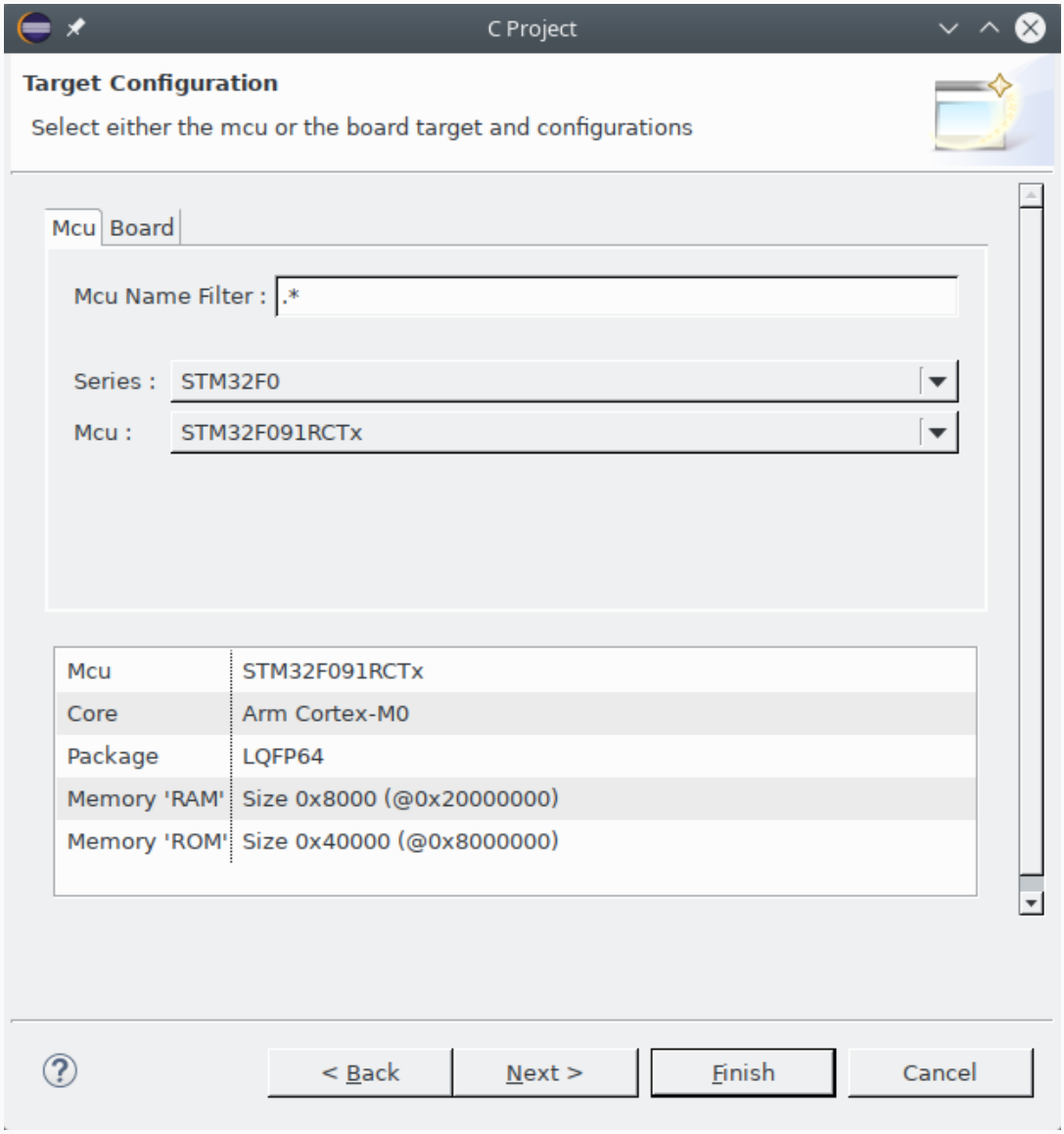


Figure 11: Target Configuration dialog

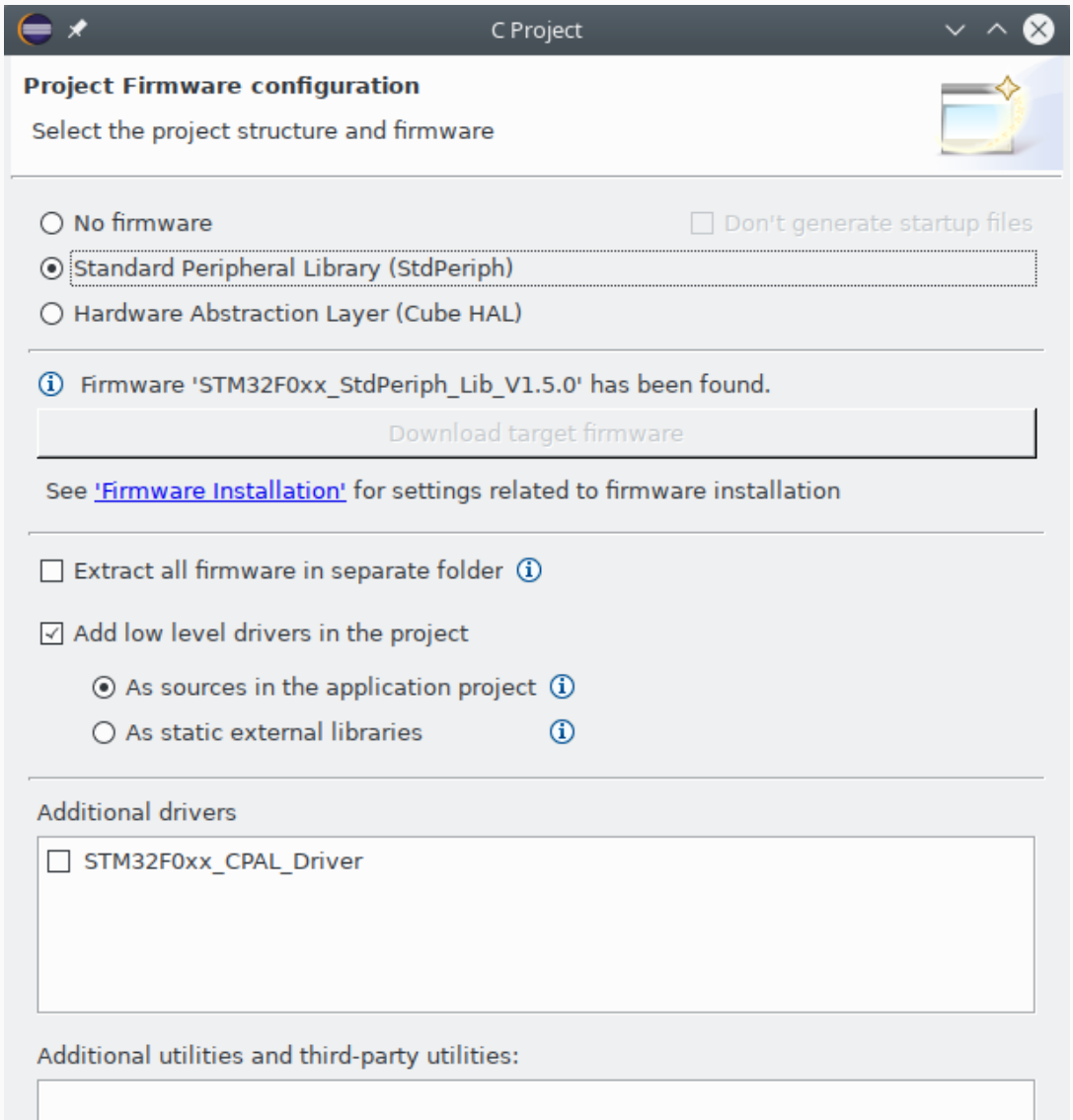
5.6 Project Firmware configuration

Next, the "Project Firmware configuration" dialog appears to ask you what type of firmware you should use for the project you are creating. For all projects in this course, we will use either "No Firmware" or the "Standard Peripheral Library (StdPeriph)" firmware. For this experiment, choose "Standard Peripheral

Library (StdPeriph)". Since this is the first time you have used this firmware type, the dialog will appear as it does on the left of the figure below. It will warn you that the firmware is not installed and prompt you to download it. Press the "Download target firmware" button, accept the license agreement, and wait a few seconds for it to complete. If System Workbench fails to download the StdPeriph firmware, download the locally-cached version as detailed in Step 4.

You only need to download the firmware the first time you use it. Once the firmware is downloaded, the dialog will always appear as it does on the right. Then press **Finish**.

Eventually, all of these steps necessary to create a project will become automatic for you. You won't even think about it.



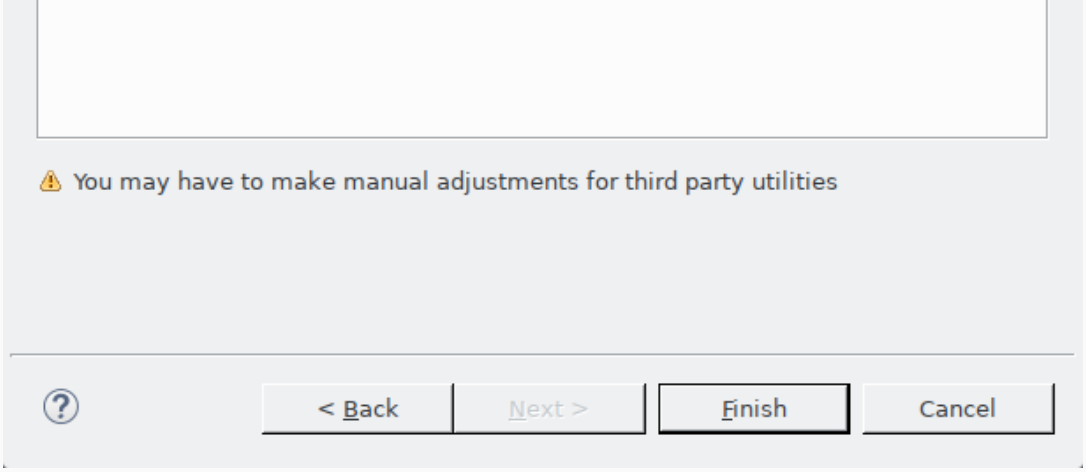


Figure 12: Project Firmware configuration dialog

6 Edit build, and debug a project

Finally we arrive at the main project workspace where we can start manipulating files related to the software project we wish to build. Eventually, you will become accustomed to going through the steps of creating a new project and it will become an automatic action for you. (You will usually create one new project for every prelab, lab, and homework that you write.)

If you click open the "lab0" element in the Project Explorer on the left, you will see the followinga number of top-level folders. Not all of these folders will exist at this point. Some of them will be created after you edit and compile your code.

- Binaries: Folder containing your compiled program.
- Includes: Global folder containing all of the system-provided C header files used in the project.
- inc: Folder containing project-specific C header files you create for the project.
- src: Folder containing C source files (including main) you create for the project.
- startup: Folder containing device-specific startup files. Startup files are generally provided as assembly files (for the

purposes of ECE362, STM32F0-series assembly files are provided).

- Debug: Folder containing the "debug version" of your compiled program.
- LinkerScript.ld is the configuration file that tells the linker where to put the segments (e.g. text and data) of the executable for your compiled program.

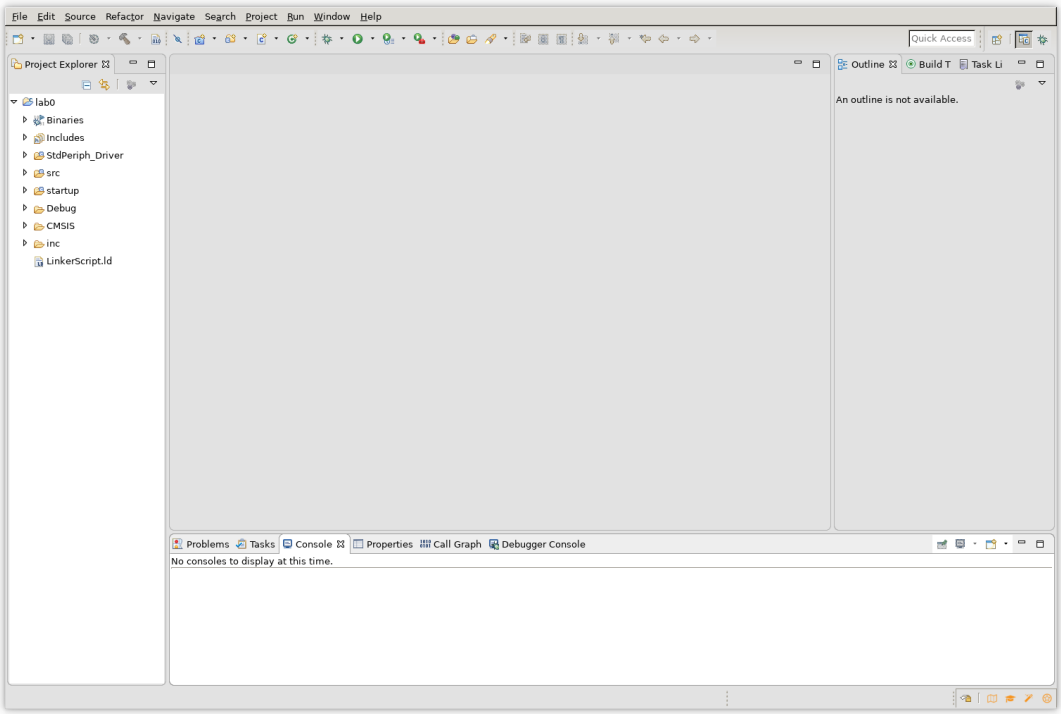
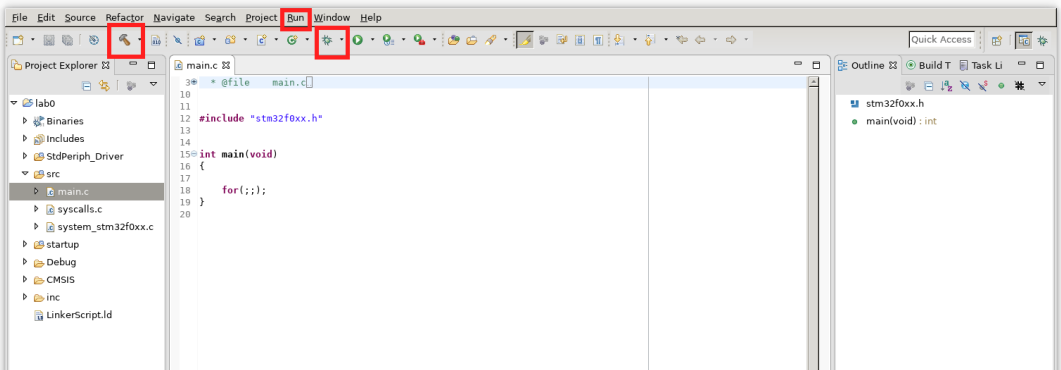


Figure 13: Project Workspace default view at startup

6.1 Editing a source file

Double-click on the **src** folder to open it, then double-click on the **main.c** file to edit it. The editor window should look similar to the following figure:



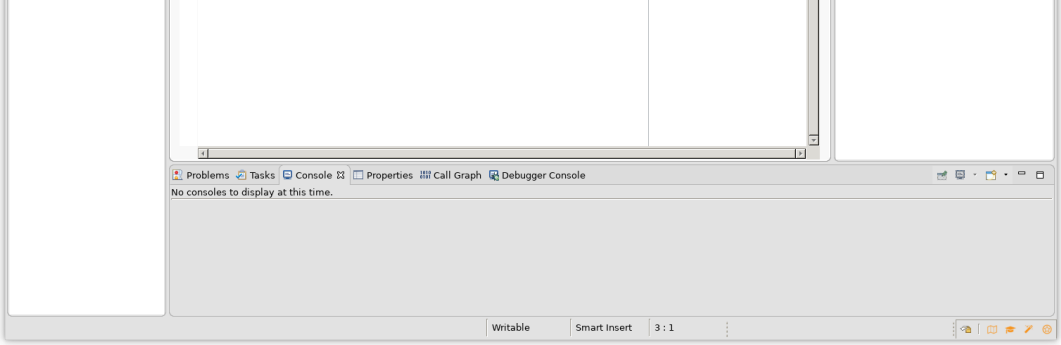


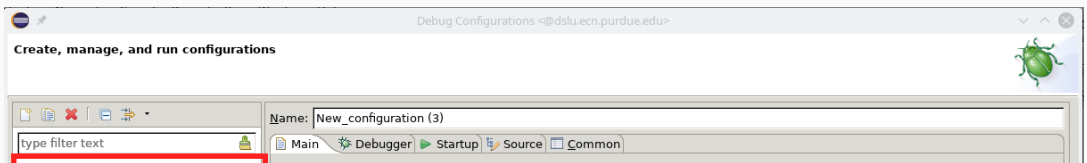
Figure 14: Editor window

System Workbench automatically created this "main.c" file for you. It includes a comment at the top of the window that looks like "* @file main.c" that is folded. If you click on the plus icon near the line number, it will unfold the comment, allowing you to view it.

If you press the hammer icon you will see many messages go by explaining that it is building firmware files and compiling the main.c file to produce an executable. You can run and debug the execution of this simple program on you development board. Before you do so, you must configure the debugger as well as the programming interface.

6.2 Debugger Configuration

Select the Run >> Debug configurations… menu item. A dialog will appear. ***Double-click*** on the "Ac6 STM32 Debugging" item on the left side of the screen to create a New_configuration entry. Notice that the selection for "C/C++ Application" is empty in your dialog window. Press the "Search Project..." button and select the "lab0.elf" entry that appears. (This is the executable that you just built in the previous step.) The result should look similar to the following picture:



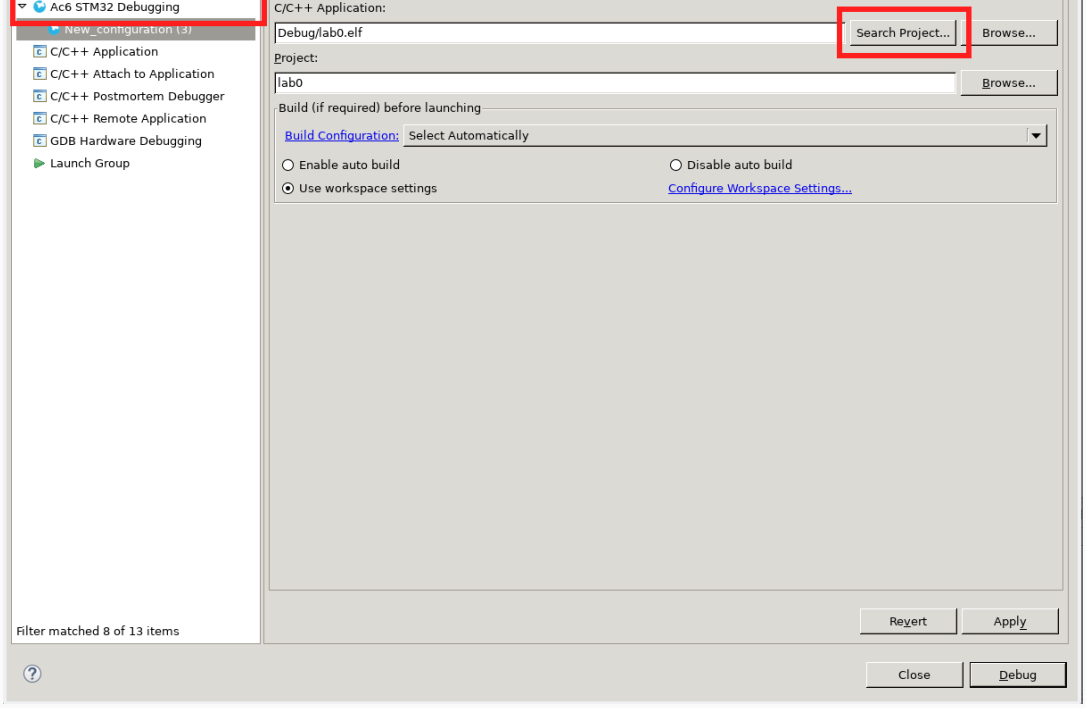


Figure 15: Debugger executable selection

Next, you must configure the programmer interface. While still in the **Run >> Debug configurations...**, select the "Debugger" tab. Change the button that normally says "Show generator options..." to "Hide generator options..." and, in the information shown below it, change the option that says "Connect under reset" to "Software system reset". Then press **Apply** and **Close**.

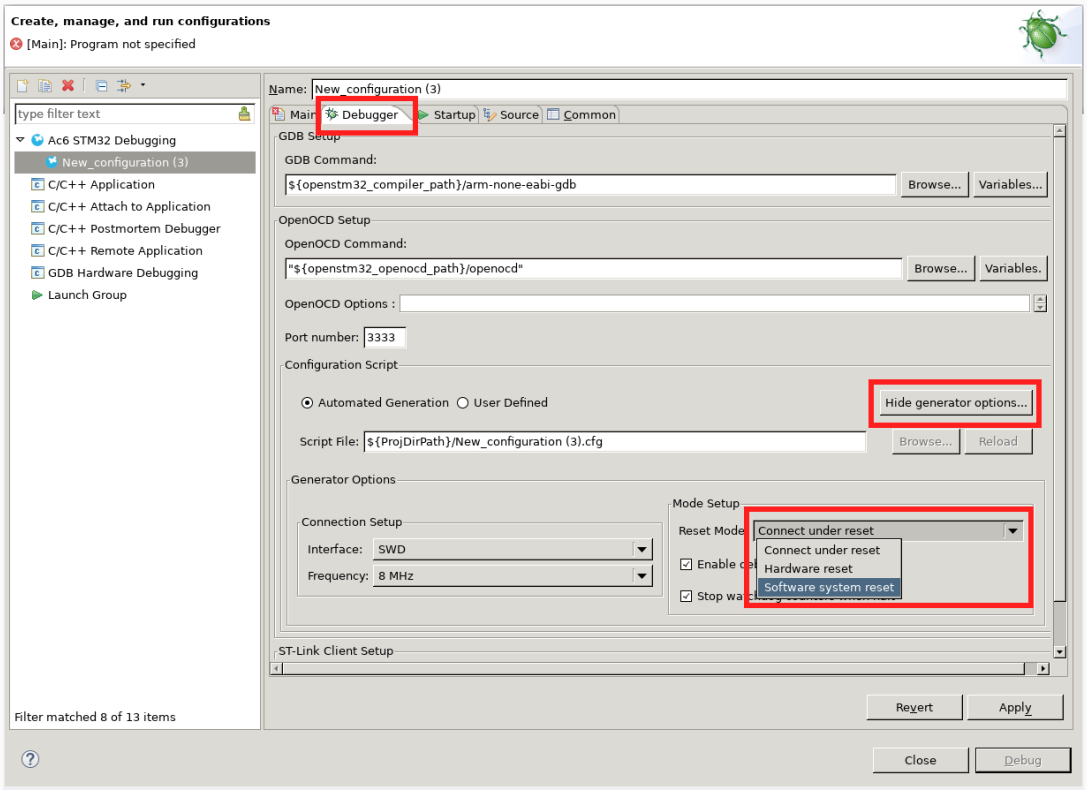


Figure 16: Debugger interface configuration

Why do we need to do this? We're using an inexpensive "clone" ST-LINK programmer that does not have as many features of more sophisticated (and expensive) programmers. This USB programmer does not use the "reset" signal. Instead, it asserts a signal on the SWDIO and SWCLK lines to reset the system it is programming and debugging. We need to change this configuration each time we create a new project. Thereafter, it is remembered for the project, and the settings will be reinstated when the project is reopened.

Note: When you try to program the microcontroller without switching to "Software system reset" SystemWorkbench will respond with one of many error dialogs. When you start the debugger, it may tell you something like "Error erasing flash with vFlashErase packet". When you press the "Run" button, it may show a message on the console at the bottom that says "** Unable to reset target **", wait a minute, and then display a dialog window that says that the microcontroller is not responding. Learn to recognize any of these problems, change the debugger configuration, and continue.

6.3 Debugging a program

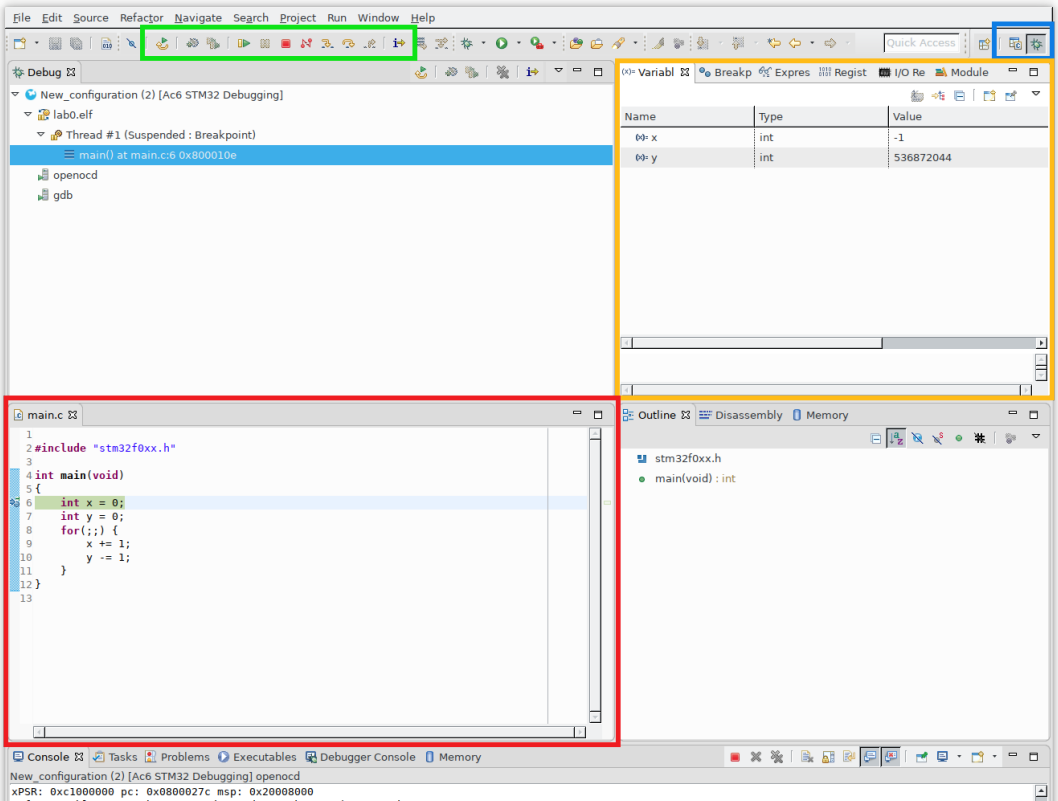
At this point, you will be able to use the programmer and debugger. Before doing so, change the `main()` function to look

like this:

```
#include "stm32f0xx.h"

int main(void) {
    int x = 0;
    int y = 0;
    for(;;) {
        x += 1;
        y -= 1;
    }
}
```

When you press the debug icon (the blue bug) below the menu bar, your project code should be rebuilt, the debugger will be invoked, and System Workbench will prompt you to change to a different *perspective*. Click **Yes** on the perspective change dialog and accept it as default action. The new perspective for the debugger should look like the following image:



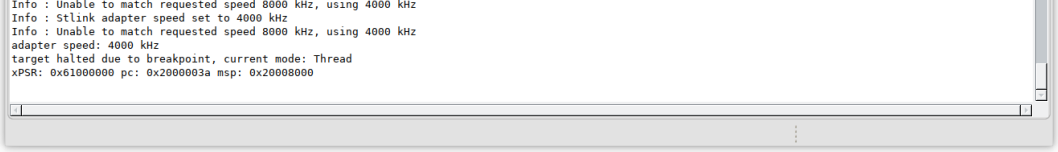


Figure 17: Debugger perspective

If you do not see the debugger perspective, it could be because the programmer was not wired properly. Check your wiring and compare it carefully to the user manual.

The editor is now reduced to a smaller portion of the screen (highlighted in red). There is now a series of buttons below the menu bar to control the debugger operation (highlighted in green). An information window (highlighted in orange) shows variable values, breakpoints, register contents, and I/O registers. Finally, a perspective selection buttons (highlighted in blue) can be used to switch back to the main editor perspective.

6.4 Using the Debugger

Hover over the debugger control buttons under the menu bar (highlighted in green in Figure 17) to learn their functions before doing the following steps. The controls are as follows:

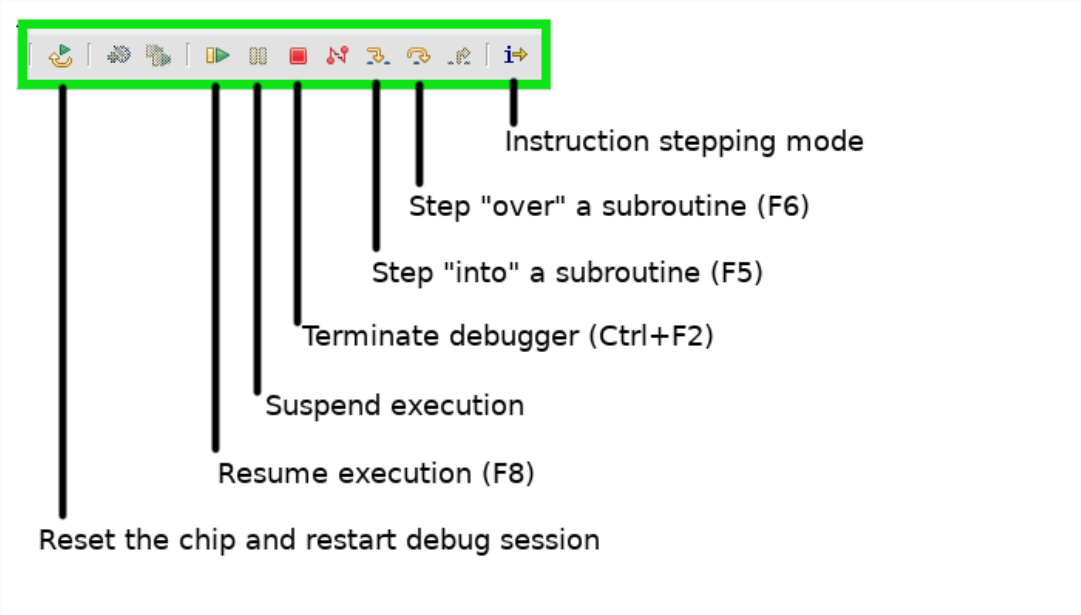


Figure 18: Debugger controls

- Press either the "Step Into (F5)" or "Step Over (F6)" buttons several times. (We have no subroutines in this program, so either one does the same thing here.) Notice how the highlighting in the editor window changes to show the statement that is ready to execute next.
- In the information window in the upper right quadrant of the screen, select the "Variables" tab. Press the "Step Into" or "Step Over" buttons again to continue executing the C program. Notice how the value of the "x" and "y" variables change.
- Press the debugger control button that will "Reset the chip and restart debug session". Notice how it does exactly what it says it does.
- Press the "Terminate (Ctrl+F2)" button and notice that all the debugger control buttons turn gray and become inoperative. To restart the debugger, press the debugger button again (small blue bug under the menu bar).
- Once the debugger is restarted, press the button that will "Resume (F8)". Notice how many of the debugger control buttons turn gray and become inoperative. At this point, your program is running unhindered on the development board.
- Press the button that will "Suspend" the debugger. Notice how the variable display shows large values now. This happened because the microcontroller was allowed to run very quickly and update the variable as fast as it could.

Note: If you try to start a debug session when the debugger is already running, it will fail with a message that says something like "Error launching debugger". Look for the red square that indicates the debugger is already running. Use that button to

"Terminate the debugger" before trying to restart it.

6.5 Excluding a source file and adding a new one

Press the "Terminate debugger (Ctrl+F2)" button, and go back to the "C/C++" Workspace Perspective by clicking on the icon in the upper right corner with a "C" on it. It is next to the "Debug" button (another blue bug icon) in the upper right corner. The view should be restored to it was in Figure 13. On the right is the list of folders, and the **src** folder contains **main.c**. Right click (or command-click) on the **main.c** to bring up a context menu. Select "Resource Configurations" and "Exclude from Build..."

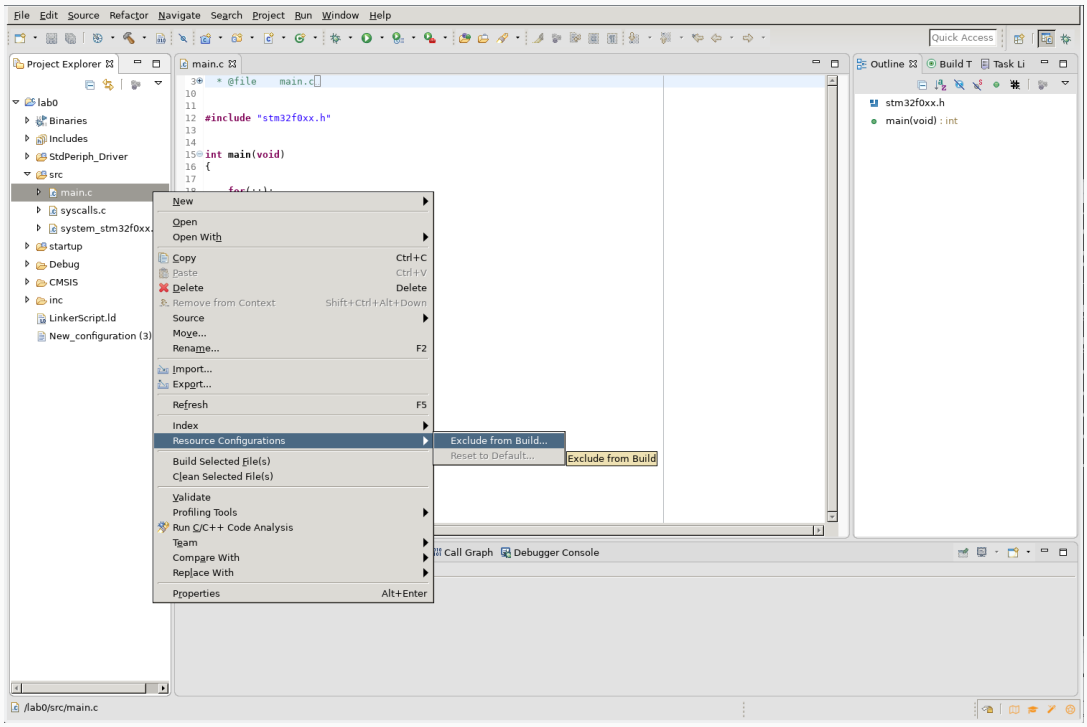


Figure 19: Exclude from build

A dialog will appear. Press "Select all" and "OK". Now main.c is *grayed out*. It is no longer compiled into your program. If you attemted to compile and run the program, an error would appear

in the message console that says "undefined reference to 'main'". (Try it.)

Add a new source file by right clicking on the **src** folder and selecting **New >> Source File**. When the dialog appears, type the name for "Source file:" as "lab0.s" and choose "<None>" as the template as shown in the figure below. This will be understood by SystemWorkbench to be an assembly language file.

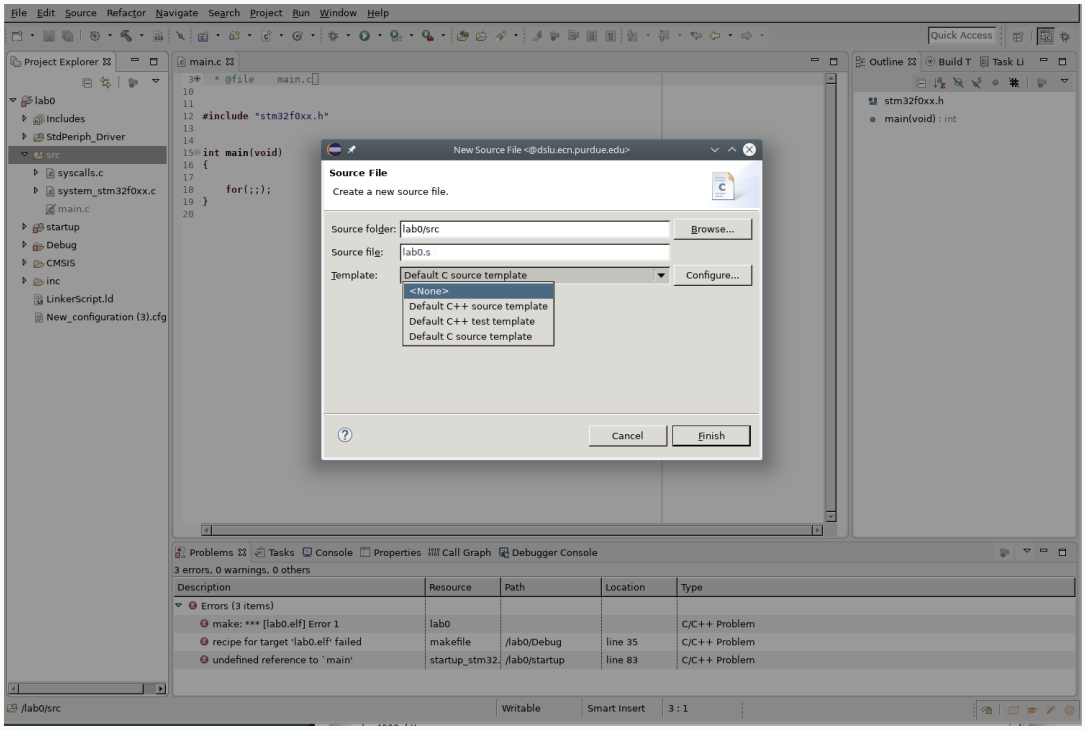


Figure 20: New source file

When the (completely empty) lab0.s appears in the editor window, enter the following text into it:

```
.cpu cortex-m0
.thumb
.syntax unified
.fpu softvfp

.global main
```

```
main:
    movs r0, #0
    movs r1, #0
loop:
    adds r0, #1
    subs r1, #1
    b    loop
```

Invoke the debugger to see how the system works with assembly language instead of C. Now the "Step Into (F5)" or "Step Over (F6)" advance one line at a time. Each line that does not start with a dot is a single machine instruction. Although you have not learned how assembly language works yet, you may recognize that "adds r0, #1" will add one to the R0 register. Click on the "Registers" tab in the information window (the orange highlighted region in Figure 17). As you step through one instruction at a time, notice how the values for general registers R0 and R1 change.

6.6 Adding a precompiled module to your program

Many times throughout the course for lab experiments and homework assignments, you will be provided with a *compiled module* that can be used to automatically test the code that you write. Sometimes, it will provide some kind of functionality that we expect you to learn how to do yourself. We don't want to just give you the source code for it.

Download the *object file* autotest.o and save it into the **src** folder of your **lab0** project. To do this, you will need to learn where the subdirectories exist in your Eclipse/SystemWorkbench

workspace. Depending on the computer platform you use, you may be able to copy and paste it.

Next, follow the instructions for integrating the compiled module into your project. For lab 0, the compiled module will act as a *device driver* that will make the OLED LCD display work, collect information about your work, and provide an indication that you completed it.

Change your "lab0.s" module to the following contents:

```
.cpu cortex-m0
.thumb
.syntax unified
.fpu softvfp

.global main
main:
    bl autotest
    bkpt

.global login
login:
    .asciz "xyz"
.align 2
```

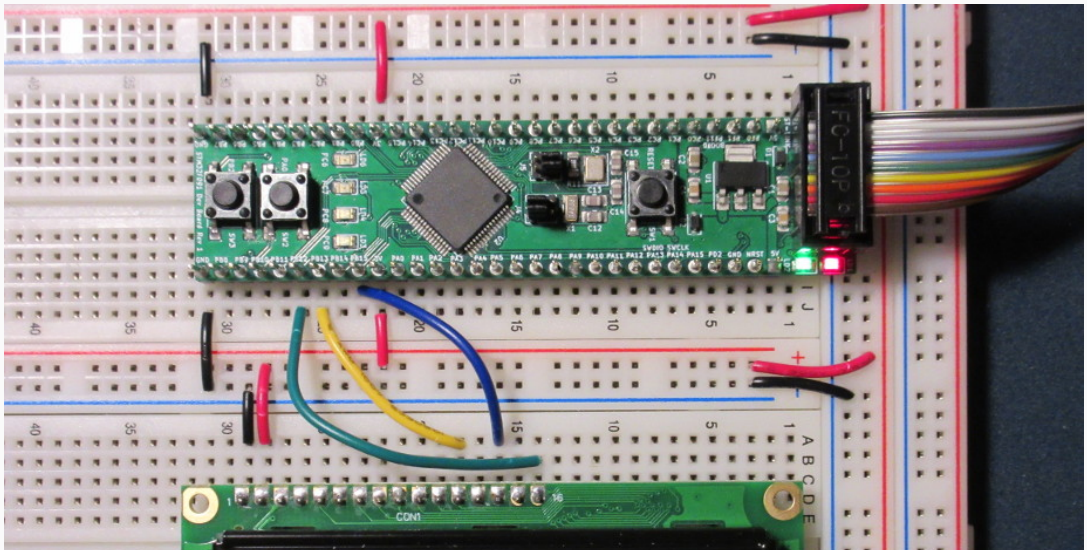
Change the letters **xyz** to your ECN login (the username you use to access the ECE 362 web page). The **bl** instruction invokes a subroutine call to a function in the compiled module named **autotest()**. This function never returns, and you will not be able to examine its source code. The function looks at the **login** symbol to find out what your login is, and it will print it on the display, along with other information.

7 Wire a device and try it

Find the OLED LCD display in your lab kit. It is a printed circuit board with a black-framed display and 16 pins. Insert it into your breadboard as shown in Figure 20, below. Connect the following pins of the OLED LCD display to the corresponding pins of the STM32 development board:

OLED pin	STM32 pin
1	GND
2	3V
12	PB13
14	PB15
16	PB12

We recommend that you get in the habit of connecting the STM32 development board's 3V and GND pins to the red and blue buses of the breadboard. This will make it easy to supply power and ground to devices that you add in experiments. **The central area of the back side of the OLED LCD has an 11 V charge-pump. (Look for the inductor.) Ensure that it does not touch any connections on the breadboard. That will certainly destroy a microcontroller.**



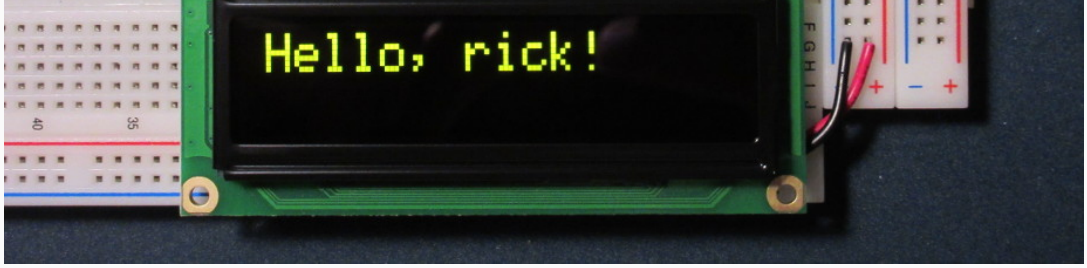


Figure 21: OLED LCD display connections

We also recommend that you place the components on the breadboard roughly as we show in this lab. You'll be adding more components in future labs, and we'll assume that you have things in approximately the positions we're describing here. You're free to put things where you want to, but it will probably be easier if you stick with the recommendations.

7.1 Wire a serial-to-USB adapter

This is an optional step for this lab, but you will use this wiring on later labs. Sometimes, students are unable to get their OLED LCD display to work to get an autotest completion code as shown below. The autotest module is set up to print its output on the *serial port* as well as the OLED LCD.

Consult the documentation on the FTDI USB-to-serial adapter page. Be sure to set the adapter for 3V operation. Then wire the serial port as follows:

- Connect the Rx pin (2nd pin) of the FTDI adapter to pin PC12 of the STM32 board.
- Connect the Tx pin (3rd pin) of the FTDI adapter to pin PD2 of the STM32 board.
- Connect the GND pin (6th pin) of the FTDI adapter to ground of the breadboard.
- Connect the CTS pin (5th pin) of the FTDI adapter also to ground.

If you're clever with wiring you can get the adapter to fit onto the breadboard next to the OLED LCD as shown in Figures 22 and 23, below. You don't need to be quite this clever if you don't want to be. The important thing is to get the wiring correct.

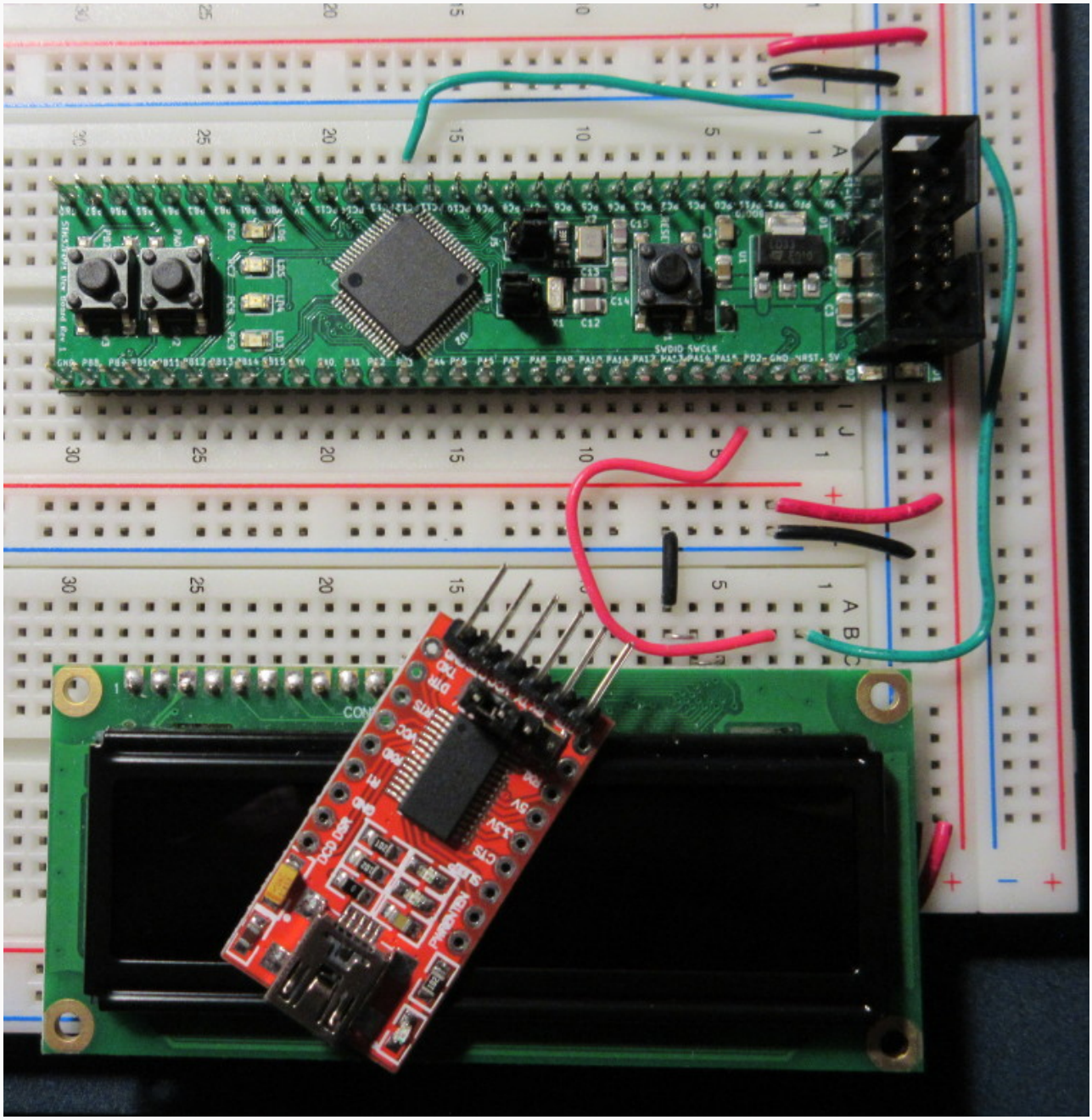


Figure 22: FTDI USB-to-serial adapter wiring

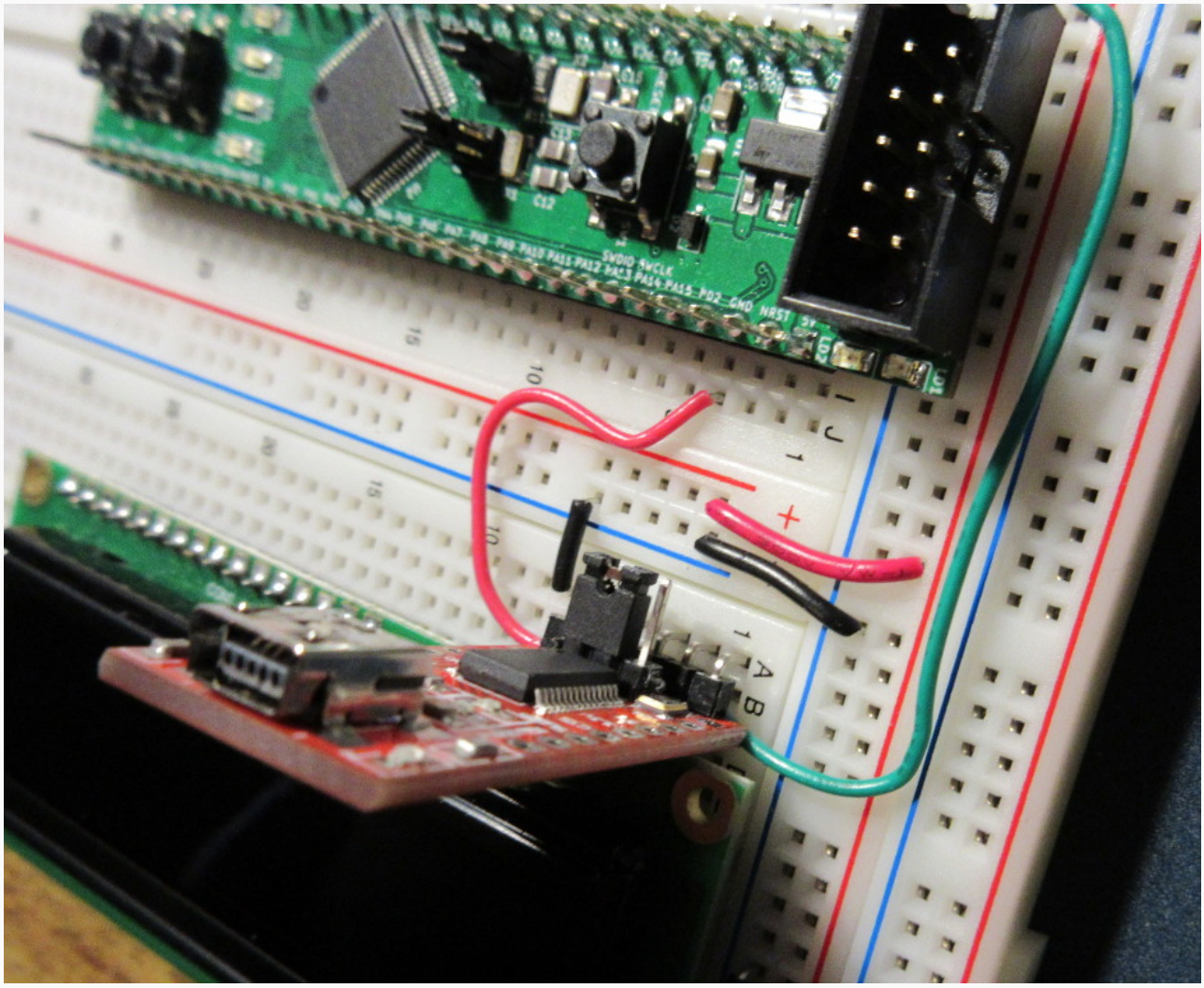


Figure 23: FTDI USB-to-serial adapter inserted

Note that Figures 22 and 23 do not show the power and ground connections to the STM32 as described in the previous step. You will still need to put those in for the system to work.

Next, follow the instructions in the adapter documentation page for setting up a terminal program and watching the output. Set the terminal for a *baud rate* of 115200 and an 8-bit word with no parity. Even if your OLED LCD does not work, you will see the same messages on the serial terminal. One more upside is that you can copy-and-paste the completion code rather than having type look-and-type.

8 Submit your postlab results

Once your program compiles, and you can program and run the code on the STM32, and you see your ECN login displayed on the OLED LCD, press and release the button on the development board labeled "PA0" and "SW2". The compiled module will display a code on the OLED LCD. It will look something like Figure 24, although the code shown is invalid, so don't try that one. This code is a cryptographic confirmation that you have completed the lab experiment. It is normal for the code to change every time you run it. Choose one of the codes and carefully enter all 32 characters into the postlab submission for this lab experiment.

Note that, if you're sharing a microcontroller with another student, you will both receive a zero for this assignment. Get your own. Use your own.

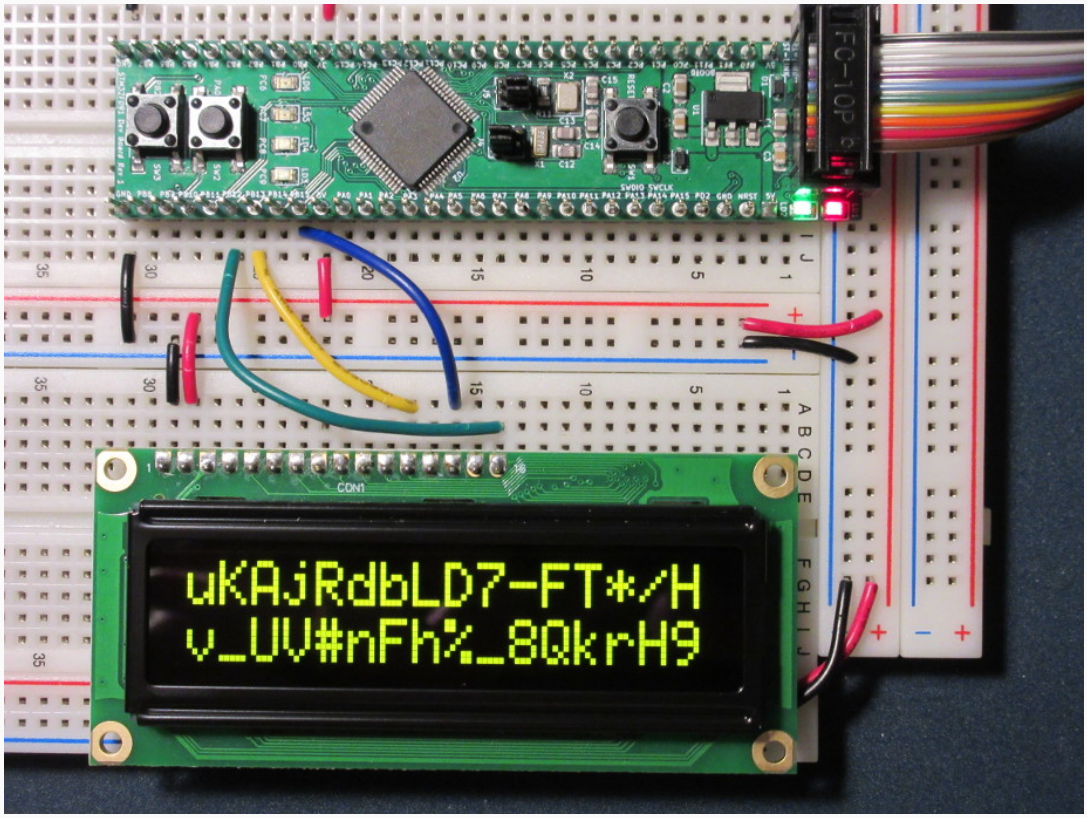


Figure 24: Example of completion code (Note: It's invalid. You'll need your own.)

For most labs, there is a requirement that you submit the program that you wrote so that it can be checked by the course staff. You don't really know how to write any programs yet, and we gave you everything you need. The real work for this lab experiment was reading, learning, stepping through the instructions, and running the result on your own microcontroller.

Lab Evaluation Checklist

Normally, we'll have a checklist of things that you should review before going into your evaluation. There will be no evaluation for this lab experiment.

- [ ] You did not share your microcontroller with anyone else, right? We will know if you did.