

University of Victoria
Electrical and Computer Engineering
ECE 355: Microprocessor-Based Systems
Laboratory Manual

By
Brent Sirna and Daler Rakhmatov
© University of Victoria, 2023

Brent Sirna and Daler Rakhmatov
© University of Victoria, 2021

B. Sirna, K. Kelany, and D. Rakhmatov
© University of Victoria, 2020

A. Jooya, K. Jones, D. Rakhmatov, and B. Sirna
© University of Victoria, 2018

INTRODUCTORY LAB

Part 1: Embedded Software Development with ECLIPSE

Objective

This part of the introductory lab will help you get started with the ECLIPSE integrated development environment (IDE). You will learn how to use ECLIPSE IDE to develop embedded software for 32-bit ARM® Cortex™-M0-based microcontroller unit (MCU) platforms – specifically, the **STM32F0 Discovery** board featuring the STM32F051R8T6 MCU. Upon completion of Part 1 of the introductory lab, you will know how to:

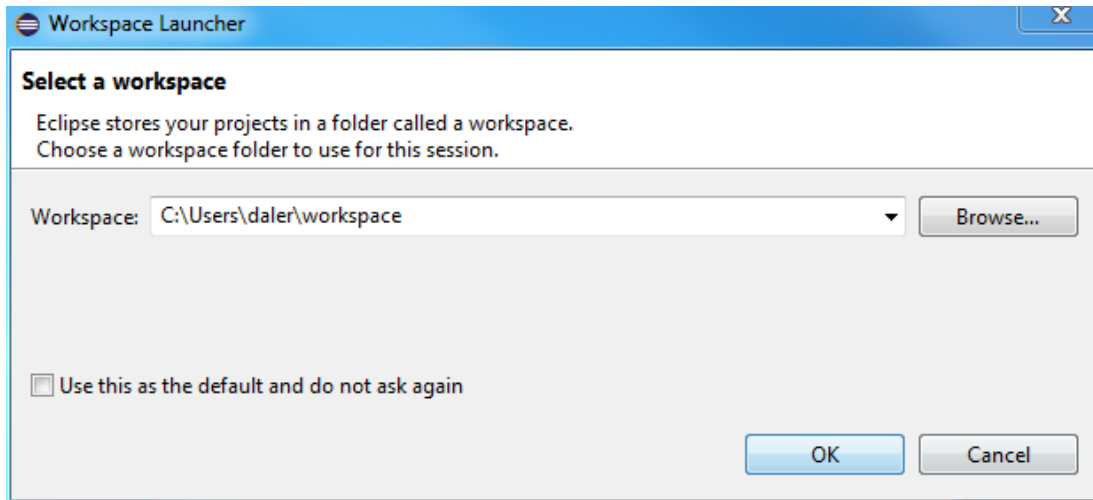
- a) Create C-based projects using ECLIPSE;
- b) Use a cross-compiler for building binary executables;
- c) Use a debugger for loading and troubleshooting executables;
- d) Use a console to observe the output of your embedded application.

General Guidelines

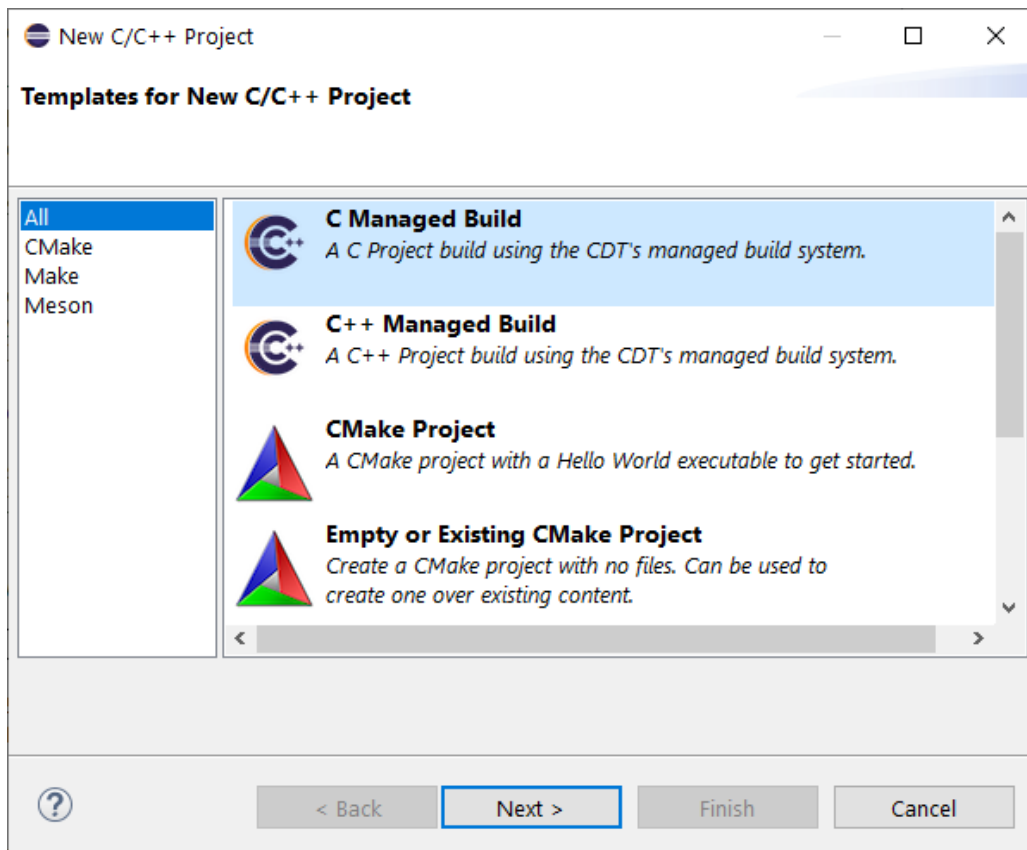
- **IMPORTANT:** *Save your work on the **M:** drive, which is your password-protected ENGR home directory, which is backed up regularly.*
- All lab-related materials are posted online at <https://www.ece.uvic.ca/~ece355/lab>.
- Please handle the hardware components with due care. To minimize potential damage due to electrostatic discharge (ESD), make contact with a grounded surface before touching the board or anything else connected to it.
- The lab equipment may be damaged if you attempt to power up an electrically faulty circuit. If in doubt, please ask your lab TA to check your circuit before powering it up.
- When working with a Function Generator, please use the **SYNC** output whose voltage is limited to the digital signal range (from 0 to +3.3 V).
- Please keep your backpacks, food/drinks, and other personal items away from the lab equipment (instruments, boards, wires, etc.), thus helping prevent accidental damage and hardware malfunctioning. If you encounter a hardware-related problem, please notify your lab TA and the technical support staff person (see the lab's white board).

Setup Procedure

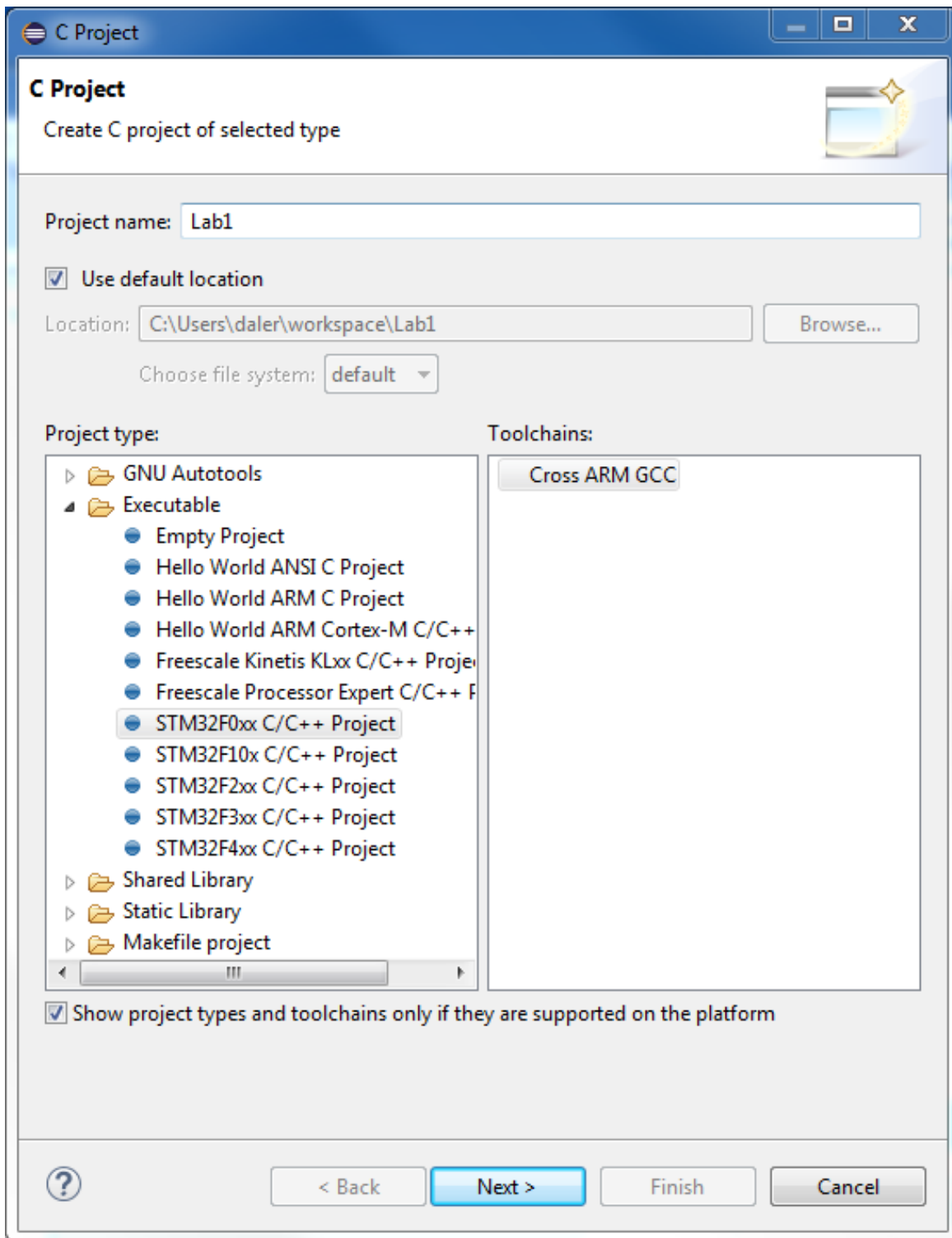
1. Login to one of the lab computers, open the **ECE355** folder on the **Desktop**, and double-click on **eclipse**, which brings up the WORKSPACE LAUNCHER window. Type ***C:\Users\YourUserName\workspace*** in the “Workspace” box (e.g., see below).



2. First, in the ECLIPSE window, select **File > New > C/C++ Project**. Then, in the NEW C/C++ PROJECT window, select ***C Managed Build***. Click **Next**.



3. In the C PROJECT window, type **Lab1** in the “Project name” box. In the “Project type” panel, select **STM32F0xx C/C++ Project**. Click **Next**.



4. In the C PROJECT window, ensure that your **Target processor settings** selections are as shown below. Notice that the “Content” box reads: *Empty (add your own content)*. **IMPORTANT:** Ensure that the “Use systems calls” box reads: *POSIX (system calls implemented by application code)*. Click **Next**.

C Project

Target processor settings

Select the target processor family and define flash and RAM sizes.

Chip family: STM32F051

Flash size (kB): 64

RAM size (kB): 8

Clock (Hz): 8000000

Content: Empty (add your own content)

Use system calls: POSIX (system calls implemented by application code)

Trace output: Semihosting DEBUG channel

Check some warnings ☒

Check most warnings ☐

Enable -Werror ☐

Use -Og on debug ☒

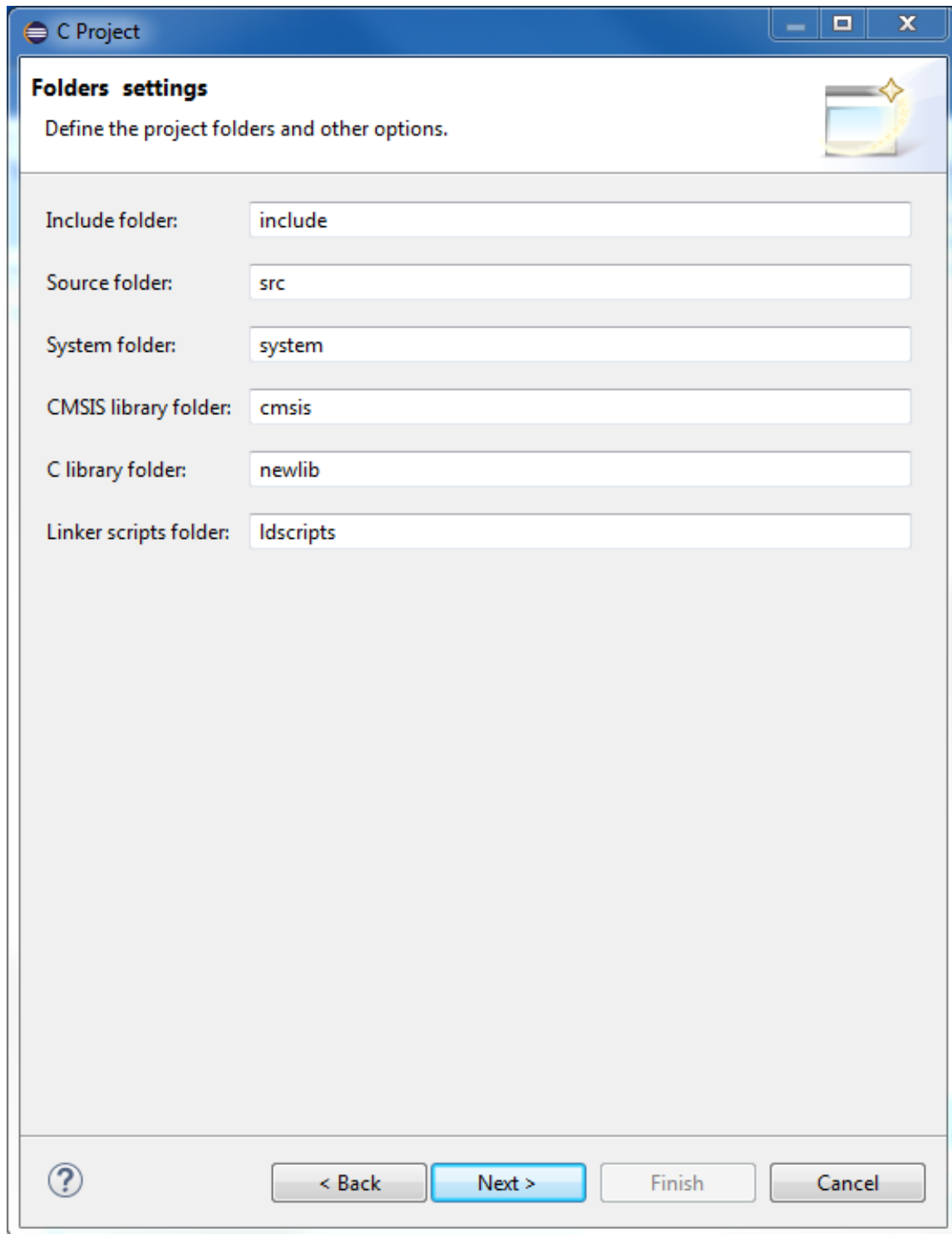
Use newlib nano ☒

Exclude unused ☒

Use link optimizations ☐

? < Back Next > Finish Cancel

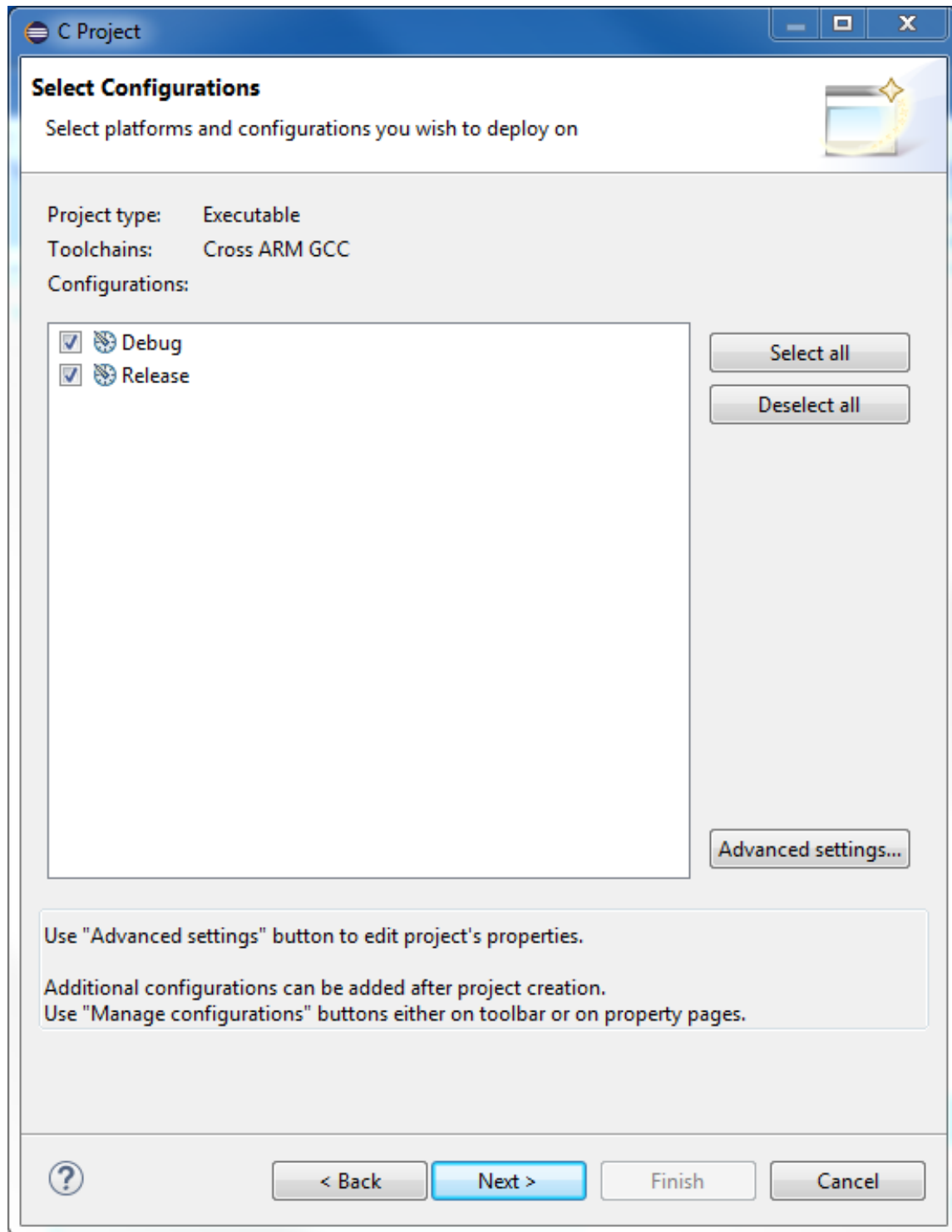
5. In the C PROJECT window, ensure that your **Folder settings** selections are as shown below. Click **Next**.



The screenshot shows a window titled 'C Project' with a blue header bar. Below the header, the title 'Folders settings' is displayed in bold, followed by the instruction 'Define the project folders and other options.' and a small icon of a folder with a star. The main area contains six text input fields, each with a label on the left and a text box on the right. The labels and their corresponding values are: 'Include folder:' with 'include', 'Source folder:' with 'src', 'System folder:' with 'system', 'CMSIS library folder:' with 'cmsis', 'C library folder:' with 'newlib', and 'Linker scripts folder:' with 'ldscripts'. At the bottom of the window, there is a row of four buttons: a help button with a question mark icon, a '< Back' button, a 'Next >' button (which is highlighted with a blue border), a 'Finish' button, and a 'Cancel' button.

Label	Value
Include folder:	include
Source folder:	src
System folder:	system
CMSIS library folder:	cmsis
C library folder:	newlib
Linker scripts folder:	ldscripts

6. In the C PROJECT window, ensure that your **Select Configurations** selections are as shown below. Click **Next**.



7. In the C PROJECT window, ensure that your **Cross GNU ARM Toolchain** selections are as shown below. Click **Finish**.

If prompted to open C/C++ perspective, click **Open Perspective**.

The screenshot shows the Eclipse IDE's 'C Project' wizard. The main window is titled 'C Project' and has a close button. The current step is 'GNU ARM Cross Toolchain', with the instruction 'Select the toolchain and configure path'. The 'Toolchain name' dropdown is set to 'GNU MCU Eclipse ARM Embedded GCC (arm-none-eabi-gcc)'. The 'Toolchain path' field is empty, with a 'Browse...' button next to it. Below this, a smaller dialog box titled 'Open Associated Perspective?' is open. It contains a question mark icon and the text: 'This kind of project is associated with the C/C++ perspective. Do you want to open this perspective now?'. There is an unchecked checkbox labeled 'Remember my decision'. At the bottom of the dialog are two buttons: 'Open Perspective' (highlighted with a blue border) and 'No'. At the very bottom of the main wizard window, there is a bar with a help icon, and four buttons: '< Back', 'Next >', 'Finish' (highlighted with a blue border), and 'Cancel'.

C Project

GNU ARM Cross Toolchain
Select the toolchain and configure path

Toolchain name: GNU MCU Eclipse ARM Embedded GCC (arm-none-eabi-gcc)

Toolchain path: **Browse...**

Open Associated Perspective?

? This kind of project is associated with the C/C++ perspective. Do you want to open this perspective now?

☐ Remember my decision

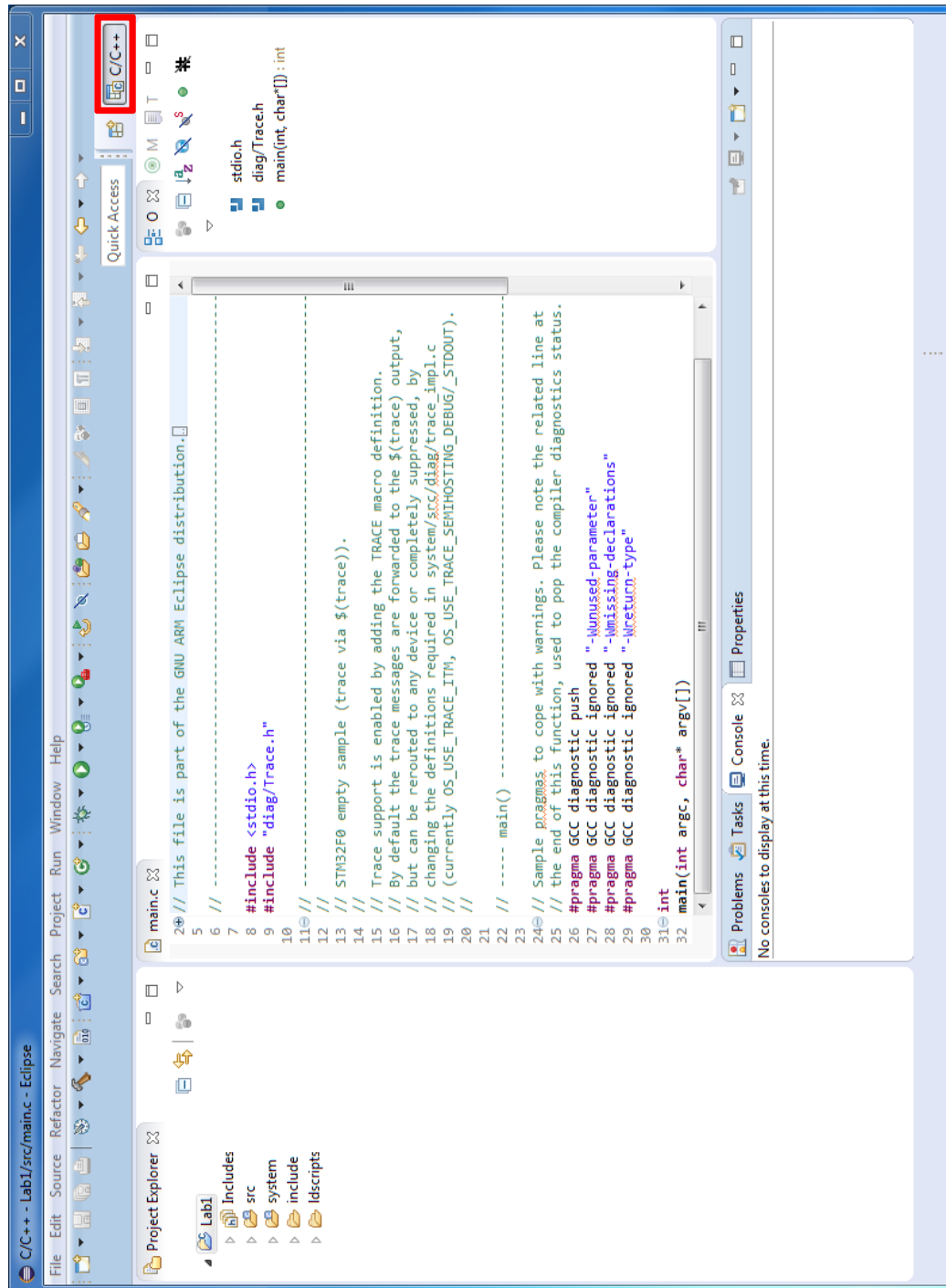
Open Perspective **No**

? < Back Next > **Finish** Cancel

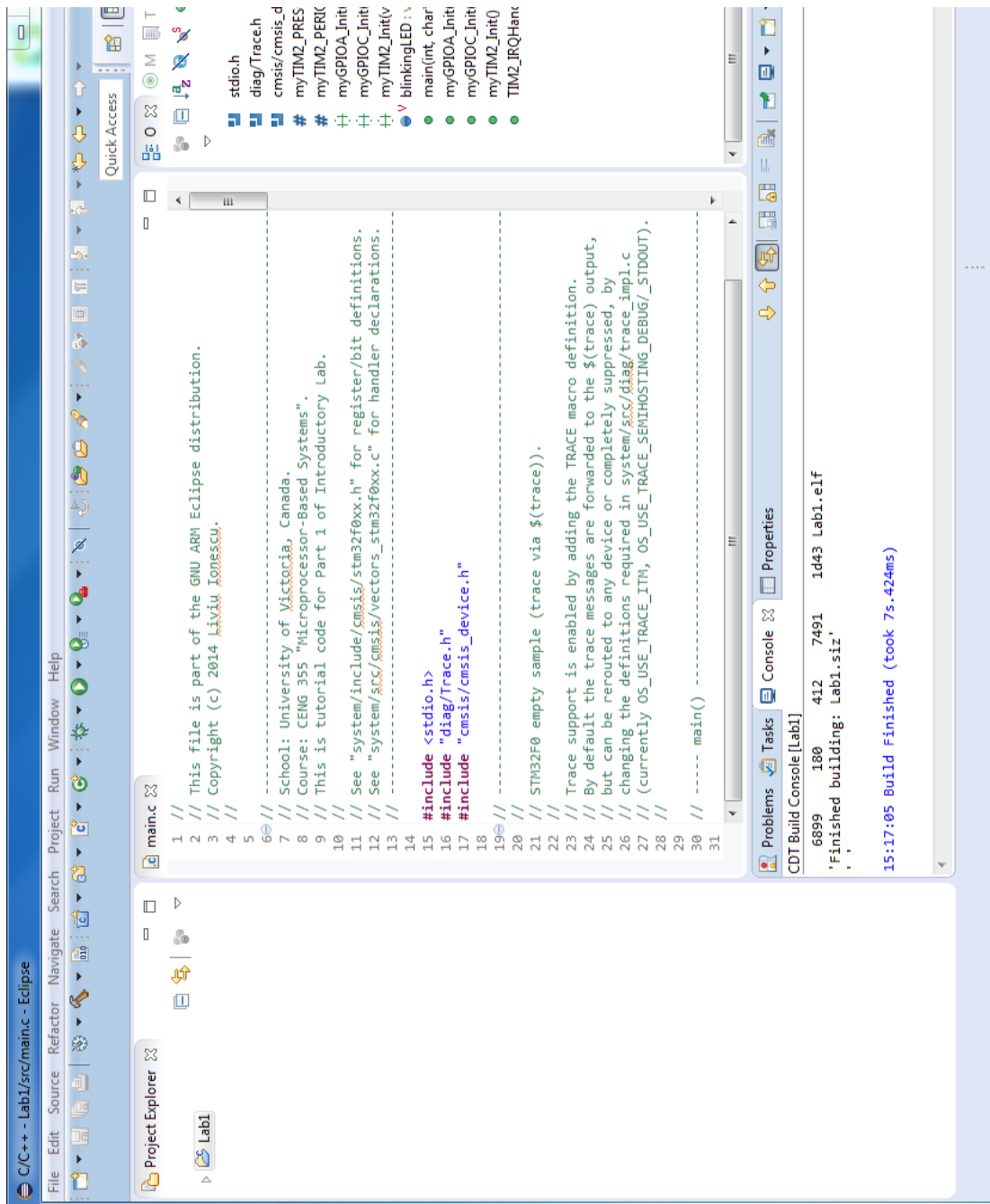
8. In the ECLIPSE window, expand the **Lab1** folder in the “Project Explorer” tab in the left panel and switch to the “Console” tab in the bottom panel. In the newer ECLIPSE versions, the buttons for C/C++ and **Debug** perspectives may appear as small icons



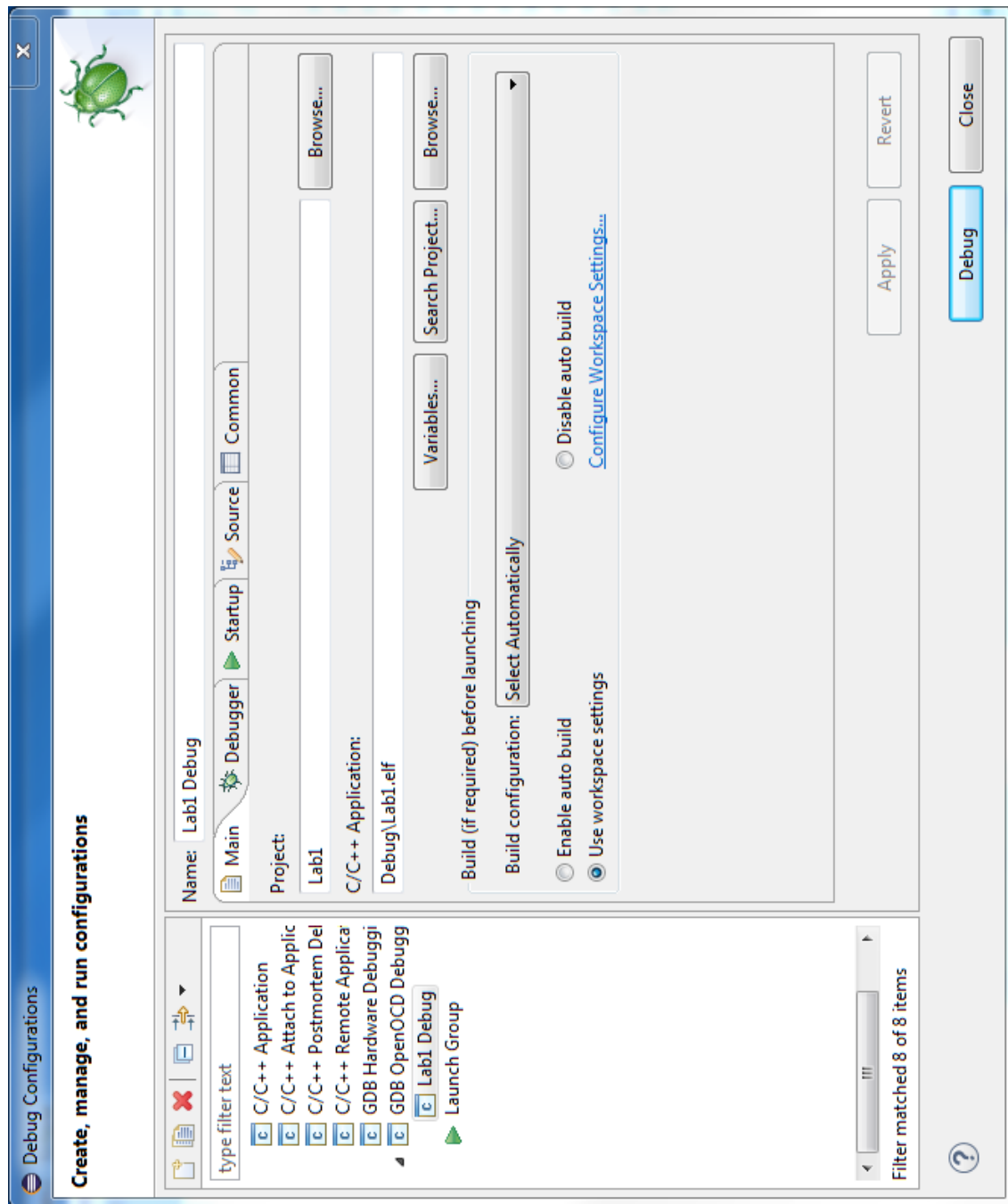
(left: C/C++, right: **Debug**).



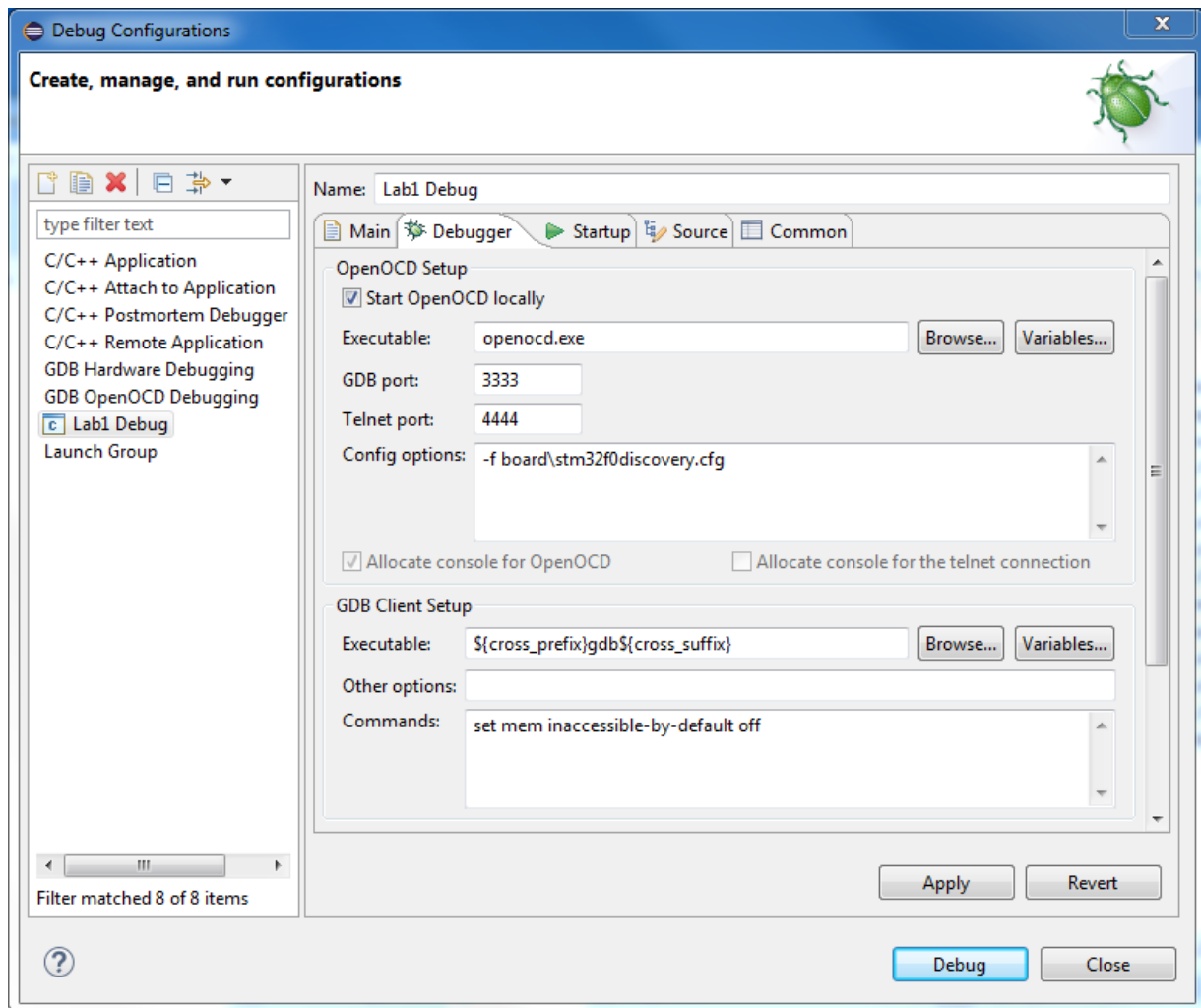
9. In the ECLIPSE window, select the “main.c” tab in the center panel and replace the existing C code with the tutorial *main.c* program provided on the lab website. Select **File > Save** and then select **Project > Build Project**, thus compiling and linking your *main.c*. The status of the project building process appears in the “Console” tab in the bottom panel (see below). If any errors are found, you can switch to the “Problems” tab to view them. There should be no errors reported.



10. Connect the USB cable to the **STM32F0 Discovery** board.
11. In the ECLIPSE window, select **Run > Debug Configurations**, which brings up the DEBUG CONFIGURATIONS window. Double-click on **GDB Open OCD Debugging** in the left panel. The DEBUG CONFIGURATIONS window should appear as shown below:



12. In the DEBUG CONFIGURATIONS window, select the “Debugger” tab and ensure that your selections are as shown below. Click **Apply**.



OpenOCD Setup – “Executable” Box:

openocd.exe

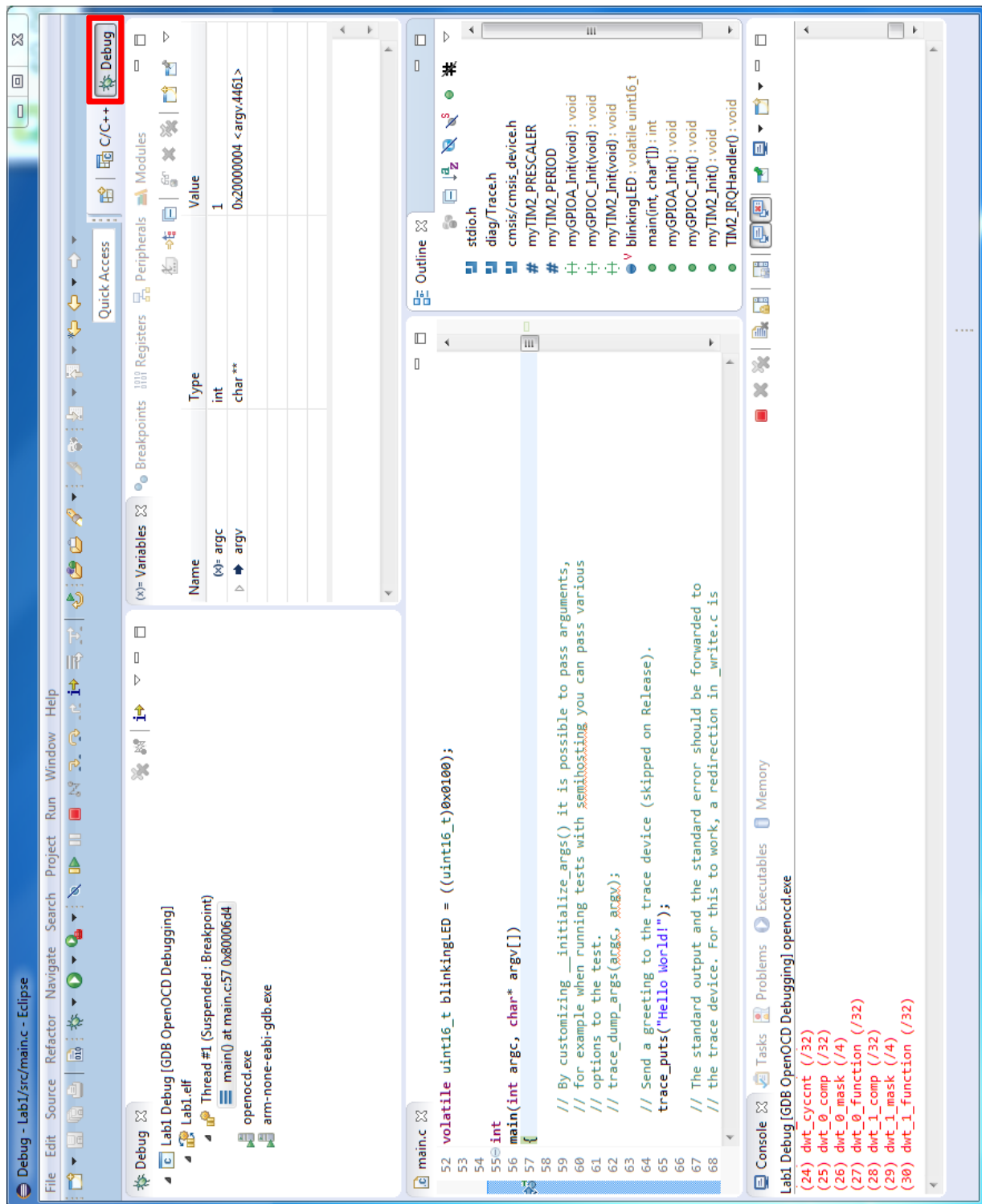
OpenOCD Setup – “Config options” Box:

-f board\stm32f0discovery.cfg

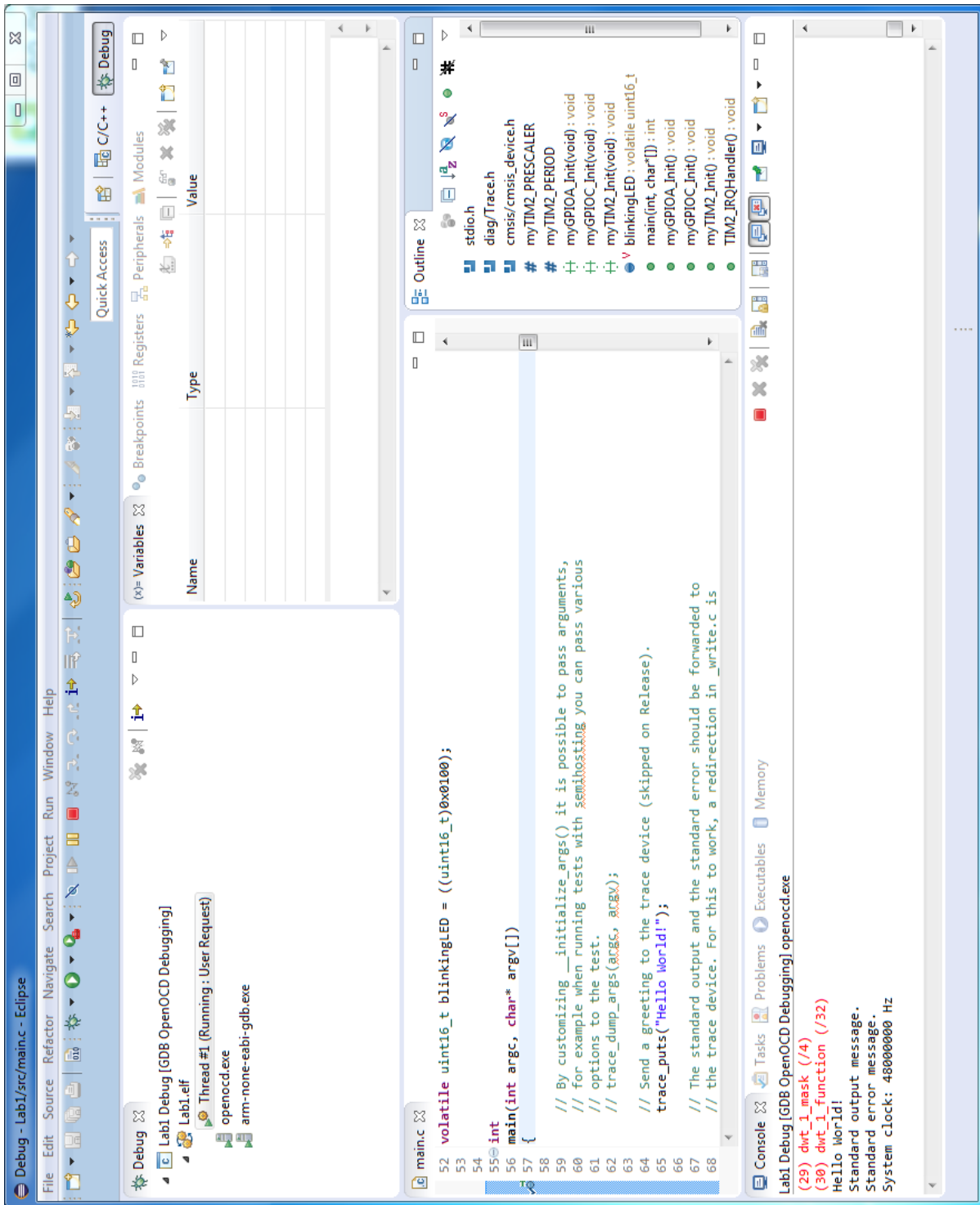
GDB Client Setup – “Executable” Box:

\${cross_prefix}gdb\${cross_suffix}

13. In the DEBUG CONFIGURATIONS window, click **Debug**. In the CONFIRM PERSPECTIVE SWITCH window, click **Switch**. The ECLIPSE window should appear as shown below:



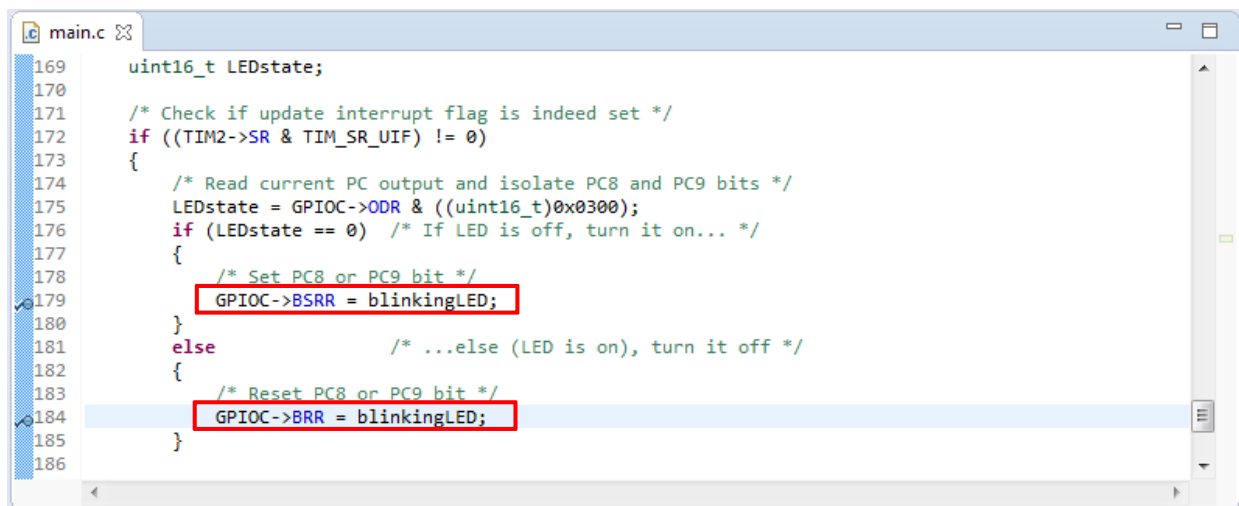
14. In the ECLIPSE window, select **Run > Resume** (or press F8 key). Four messages are printed in the “Console” tab in the bottom panel (see below), and the blue LED on the **STM32F0 Discovery** board starts blinking.



15. On the **STM32F0 Discovery** board, press the blue button (USER). The blue LED is turned off, and the green LED starts blinking. Pressing the USER button switches the blinking LED and prints “*Switching the blinking LED...*” in the “Console” tab in the bottom panel of the ECLIPSE window. This is the intended functionality of the **Lab1** executable code.

NOTE: Before proceeding to the next step, ensure that the blue LED is blinking.

16. In the ECLIPSE window, select **Run > Suspend**. In the “main.c” tab in the center panel, set breakpoints at lines 224 “**GPIOC->BSRR = blinkingLED;**” and 229 “**GPIOC->BRR = blinkingLED;**” by double-clicking on them. These two lines are marked in red below (please ignore older line numbering in this screenshot).



```
main.c
169  uint16_t LEDstate;
170
171  /* Check if update interrupt flag is indeed set */
172  if ((TIM2->SR & TIM_SR_UIF) != 0)
173  {
174      /* Read current PC output and isolate PC8 and PC9 bits */
175      LEDstate = GPIOC->ODR & ((uint16_t)0x0300);
176      if (LEDstate == 0) /* If LED is off, turn it on... */
177      {
178          /* Set PC8 or PC9 bit */
179          GPIOC->BSRR = blinkingLED;
180      }
181      else /* ...else (LED is on), turn it off */
182      {
183          /* Reset PC8 or PC9 bit */
184          GPIOC->BRR = blinkingLED;
185      }
186  }
```

17. In the ECLIPSE window, select **Run > Resume**. The program execution should stop either at line 224 (the blue LED is off) or at line 229 (the blue LED is on).

If the program execution stops at line 224, the **LEDstate** variable should be equal to **0**.

If the program execution stops at line 229, the **LEDstate** variable should be equal to **256 (0x0000100 in hex)** – see the window snapshot on the next page.

Selecting **Run > Resume** switches between the two breakpoints, thus switching the blue LED state between off and on.

The screenshot displays the Eclipse IDE interface for debugging a Lab1 project. The main window shows the source code of `main.c` with a breakpoint set at line 184. The console output indicates that the program is running and the LED state is being toggled. The variables panel shows the current state of the `LEDstate` variable, which is of type `uint16_t` and has a value of 256. The console output shows the following messages:

```

Switching the blinking LED...
Switching the blinking LED...
Info : halted: PC: 0x08000788
Info : halted: PC: 0x0800077c

```

The source code in the editor is as follows:

```

172 if ((TIM2->SR & TIM_SR_UIF) != 0)
173 {
174     /* Read current PC output and isolate PC8 and PC9 bits */
175     LEDstate = GPIOC->ODR & ((uint16_t)0x0300);
176     if (LEDstate == 0) /* If LED is off, turn it on... */
177     {
178         /* Set PC8 or PC9 bit */
179         GPIOC->BSRR = blinkingLED;
180     }
181     /* ...else (LED is on), turn it off */
182     {
183         /* Reset PC8 or PC9 bit */
184         GPIOC->BRR = blinkingLED;
185     }
186     TIM2->SR &= ~(TIM_SR_UIF); /* Clear update interrupt flag */
187     TIM2->CR1 |= TIM_CR1_CEN; /* Restart stopped timer */
188 }

```

The console output also shows the state of the LED and the current PC value:

```

Switching the blinking LED...
Switching the blinking LED...
Info : halted: PC: 0x08000788
Info : halted: PC: 0x0800077c

```


18. In the “main.c” tab in the center panel of the ECLIPSE window, remove breakpoints at lines 224 and 229 by double-clicking on them. Select **Run > Resume** (the blue LED starts blinking again).
19. Press the USER button, so that the green LED is blinking. Repeat steps **18** and **19**. Whenever the green LED is on (i.e., whenever the program execution stops at line 224), the **LEDstate** variable should be equal to **512** (**0x0000200** in hex). Repeat step **20** (the green LED starts blinking again).
20. To make any changes to your **Lab1** code, follow the sequence of steps listed below:
 - Suspend your program execution (if running): **Run > Suspend**.
 - Terminate your debugging session (if active): **Run > Terminate**.
 - Switch to the C/C++ perspective (see the top right corner of the ECLIPSE window).
 - Make any desired changes to your code in the center panel of the ECLIPSE window.
 - Save your changes: **File > Save**.
 - Rebuild your project: **Project > Build Project** (there should be no errors reported).
 - Restart debugging: **Run > Debug History > Lab1 Debug**.
21. When you are done using the **STM32F0 Discovery** board, make sure to suspend your program execution and terminate your debugging session. Then select **File > Exit**.
22. **IMPORTANT:** Copy your *workspace* folder from the **C:** drive into the **M:** drive for safekeeping.

This is the end of Part 1 of the introductory lab, where you have learned how to use the ECLIPSE IDE to build, run, and debug your embedded software code targeting the STM32F0 Discovery board.

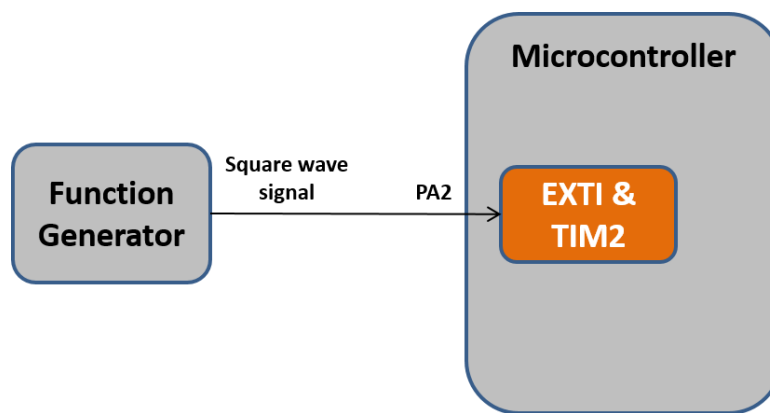
Additional up-to-date information can be found on the ECE 355 lab website:

<https://www.ece.uvic.ca/~ece355/lab>.

Part 2: Signal Frequency Measurement

Objective

Using the ECLIPSE IDE and the **STM32F0 Discovery** board, you are to develop a system that measures the frequency of a square-wave signal generated by the Function Generator (see below). The measured period and frequency are to be displayed on the console. The minimum and the maximum detectable frequencies must also be determined.



Specifications

- The input signal will be a square wave with the amplitude ranging from 0 to +3.3 V, generated by the Function Generator – use the **SYNC** output!
- You will use the **TIM2** general purpose timer to measure the frequency of the input signal. Your goal is to determine the number of timer pulses (clock cycles) elapsed between two consecutive rising (or falling) edges of the input signal. A current count value of the timer pulses is recorded in the **TIM2_CNT** counter register of **TIM2**: it must be configured to increment every cycle of the **TIM2**'s clock, whenever **TIM2** is enabled to count.
- On the **STM32F0 Discovery** board, you will use the microcontroller's **PA2** I/O pin, serving as the **EXTI2** external interrupt line, to generate interrupt requests when a rising (or falling) edge of the input signal is detected. Your **EXTI2** interrupt handler will need to access **TIM2** (e.g., start/stop the counting process, read **TIM2_CNT**, etc).

- Your C code must calculate the period and the frequency of the input signal, and then display those values on the console.
- You will need to determine the range of detectable frequencies of the input signal.

Deliverables

- **Demonstration.** You must demonstrate a working system by the end of your *second lab session*. You must also determine the limitations of your system (the maximum and the minimum detectable frequencies) and be able to explain why your system has such limitations. This deliverable is worth **10%** of your final lab grade.
- **NOTE:** No report submission is required for Part 2. You will describe your frequency measurement work in your final Project Report.

Hints and Advice

- You will need to read Chapters 7, 9, 12, 17 of the **STM32F0xx Reference Manual** providing necessary technical details on RCC (reset and clock control), GPIO (general purpose I/Os), interrupt controllers (NVIC and EXTI), and general purpose timers (TIM2 and others).
- Your goal is to determine the number of clock cycles between two consecutive rising (or falling) edges of the input signal. On the first edge, enable the timer to start counting. On the second edge, stop the counting process, calculate the signal period and frequency, and then display those values on the console.
- **IMPORTANT:** You can find a basic code template (to get you started) on the ECE 355 lab website:

<https://www.ece.uvic.ca/~ece355/lab.>

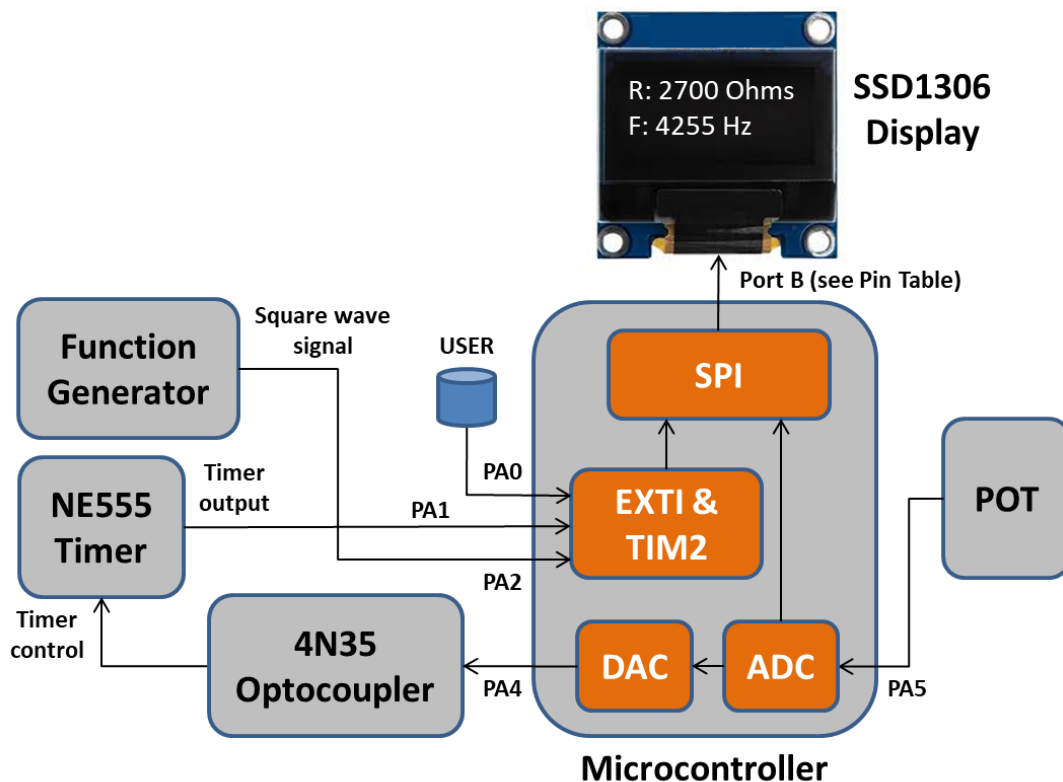
PROJECT: PWM Signal Generation and Monitoring System

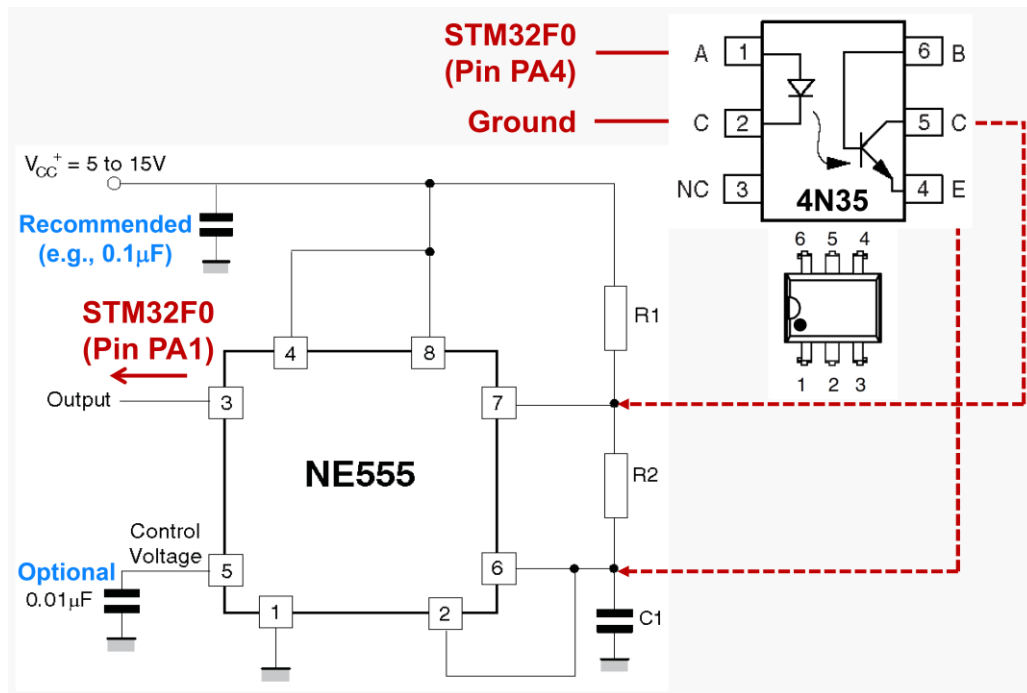
Objective

Your objective is to develop an embedded system for monitoring and controlling a pulse-width-modulated (PWM) signal generated by a 555 timer (**NE555** IC). An optocoupler (**4N35** IC), driven by the microcontroller on the **STM32 Discovery** board, will be used to control the PWM signal frequency and duty cycle.

The microcontroller will be used to measure the voltage across a potentiometer (**POT**) on the **PBMCUSLK** board and relay it to the external optocoupler for controlling the PWM signal frequency. The microcontroller will also be used to measure: 1) the frequency of the 555 timer signal, or 2) the frequency of the Function Generator signal. Pressing the USER button will toggle which frequency is to be measured by the microcontroller.

The measured frequency and the corresponding **POT** resistance are to be displayed on the LED Display (**SSD1306** IC) on the **PBMCUSLK** board.





Specifications

- In Part 2 of the introductory lab, you have used the Function Generator to generate a square-wave signal. For this project, you will also use the 555 timer to generate an additional square-wave PWM signal. An interrupt request must be raised whenever the USER button is pressed, and the corresponding interrupt handler must “force” **TIM2** to switch between measuring the Function Generator signal frequency and measuring the 555 timer signal frequency.

IMPORTANT: You must use **PA0** with **EXTI0** for the USER button interrupts, **PA1** with **EXTI1** for measuring the 555 timer signal frequency, and **PA2** with **EXTI2** for measuring the Function Generator signal frequency (as in Part 2 of the introductory lab).

- The STM32F051R8T6 MCU mounted on the **STM32F0 Discovery** board features internal analog-to-digital converter (**ADC**) and digital-to-analog converter (**DAC**). The **DAC** will be used to drive the optocoupler to adjust the signal frequency and duty cycle of the 555 timer output, based on the potentiometer voltage read by the **ADC**.

- The analog voltage signal coming from the potentiometer on the **PBMCUSLK** board will be measured continuously by the **ADC** – this task is to be accomplished using a polling approach. Using those voltage measurements, you will need to calculate the corresponding potentiometer resistance value. You must also determine the lower and the upper limits of the measurable voltage.
- You will use the digital value obtained from the **ADC** to control the frequency of the PWM signal generated by the 555 timer. For that purpose, you will use the **DAC** to convert that digital value to an analog voltage signal driving the optocoupler.
- To display the measured signal frequency and the potentiometer resistance, you will use the **SPI** (to be appropriately configured) to drive the LED Display mounted on the **PBMCUSLK** board. The interface includes 1 serial data signal (**SDIN = SPI MOSI**), 1 clock signal (**SCLK = SPI SCK**), and 3 control signals (**RES#, CS#, D/C#**). These signals and their associated pin information are shown in the table below.

STM32F0	SIGNAL	DIRECTION
PA0	USER PUSH BUTTON	INPUT
PC8	BLUE LED	OUTPUT
PC9	GREEN LED	OUTPUT
PA1	555 TIMER	INPUT
PA2	FUNCTION GENERATOR	INPUT
PA4	DAC	OUTPUT (Analog)
PA5	ADC	INPUT (Analog)
PB3	SCLK (Display D0: “Serial Clock” = SPI SCK)	OUTPUT (AF0)
PB4	RES# (Display “Reset”)	OUTPUT
PB5	SDIN (Display D1: “Serial Data” = SPI MOSI)	OUTPUT (AF0)
PB6	CS# (Display “Chip Select”)	OUTPUT
PB7	D/C# (Display “Data/Command”)	OUTPUT

Deliverables

- **Demonstration.** You must demonstrate a working project by the end of your *last lab session*. You must also determine the limitations of your system and be able to explain why your system has such limitations. Your lab TA will inspect your code and ask each group member (individually) a series of technical questions. Your (individual) mark for the project demonstration will be based on your ability to answer your lab TA's questions. This deliverable is worth **30%** of your final lab grade.
- **Project Report.** A substantial project report is required from each student group – it should be written as a standard engineering technical report. It must describe system specifications and outline your design approach (including circuit diagrams and source code), as well as present and analyze experimental results, including a discussion on your system limitations. The report should contain enough information so that another engineer could easily reproduce your system. Each group member will earn the same mark for that group's report. This deliverable is worth **60%** of your final lab grade.

Hints and Advice

- Successful completion of the project requires a thorough knowledge of the available technical documentation. You will need to read Chapters 7, 9, 12, 13, 14, 17 of the **STM32F0xx Reference Manual**, as well as the **NE555** timer datasheet and the **4N35** optocoupler datasheet. Further details on the LED Display interface (3-wire SPI) and commands are provided in the **SSD1306** data sheet.
- **IMPORTANT:** Do NOT use **PA13** and **PA14**. They are reserved for communicating with the ST-LINK chip on the **STM32F0 Discovery** board.

- Additional up-to-date information, including “Time Table” and “Project Tips”, can be found on the ECE 355 lab website:

<https://www.ece.uvic.ca/~ece355/lab>.