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

By

Brent Sirna, Khaled Kelany, and Daler Rakhmatov

[Previous Version: Ali Jooya, Kevin Jones,

Daler Rakhmatov, and Brent Sirna]

INTRODUCTORY LAB

Part 1: Embedded Software Development with ECLIPSE

Objective

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

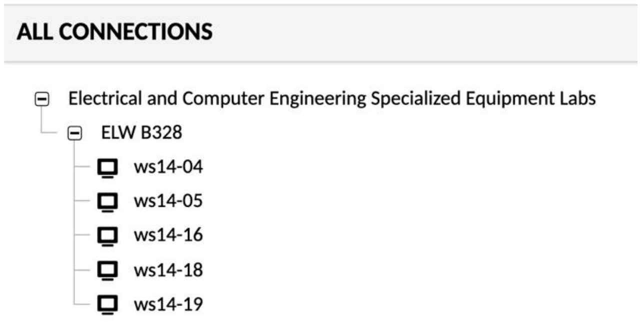
a) Create C-based projects using the ECLIPSE SDK;
b) Use a cross-compiler for building binary executables;
c) Use a debugger for loading and troubleshooting executables;
d) Use a console to observe the output of your embedded application.

General Guidelines

- **IMPORTANT:** *Save your work on the* ***M:*** *drive, which is your password-protected* ***ENGR*** *home directory, which is backed up regularly*.

Setup Procedure

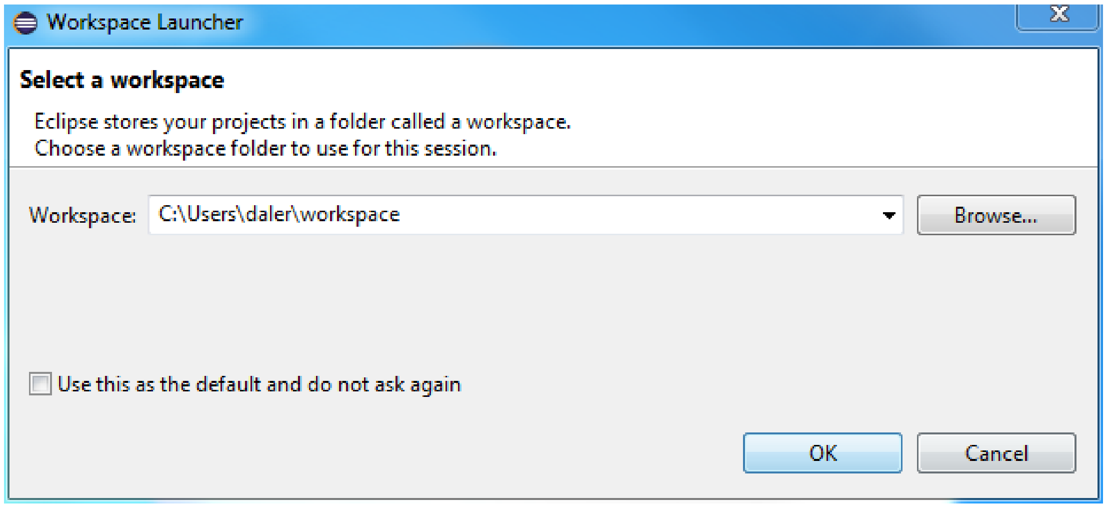
1. Go to **labs.engr.uvic.ca** with your web browser and login using your Netlink ID. Select *Electrical and Computer Engineering Specialized Equipment Labs* > *ELW B328* > *your_machine_name*.



ALL CONNECTIONS

Electrical and Computer Engineering Specialized Equipment Labs
ELW B328
ws14-04
ws14-05
ws14-16
ws14-18
ws14-19

2. After you 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\\<u>YourUserName</u>\workspace*** in the "Workspace" box (e.g., see below).



Workspace Launcher

Select a workspace

Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

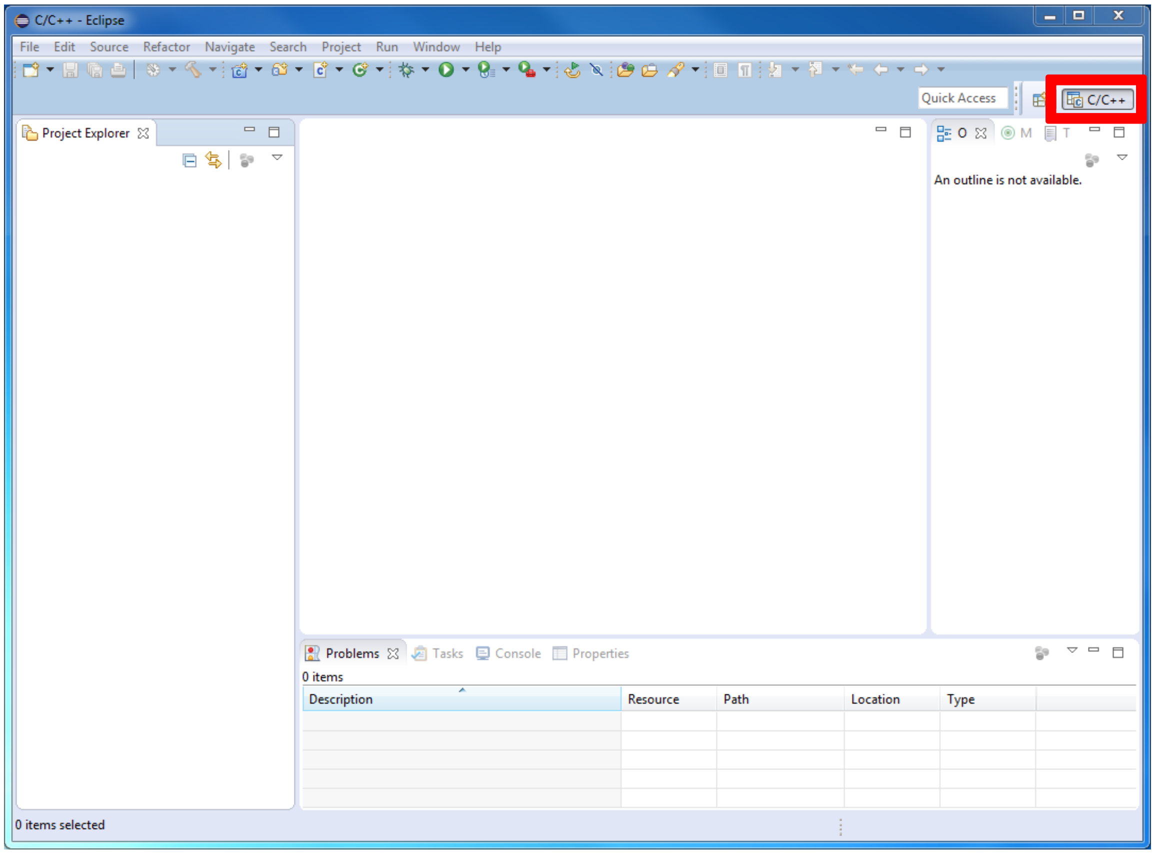
Workspace: C:\Users\daler\workspace

Browse...

Use this as the default and do not ask again

OK Cancel

3. The ECLIPSE window will appear on the screen, as shown below (close the "Welcome" tab, if needed). Notice that the **C/C++** perspective is selected.



C/C++ - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
C/C++
Project Explorer
O
M
T

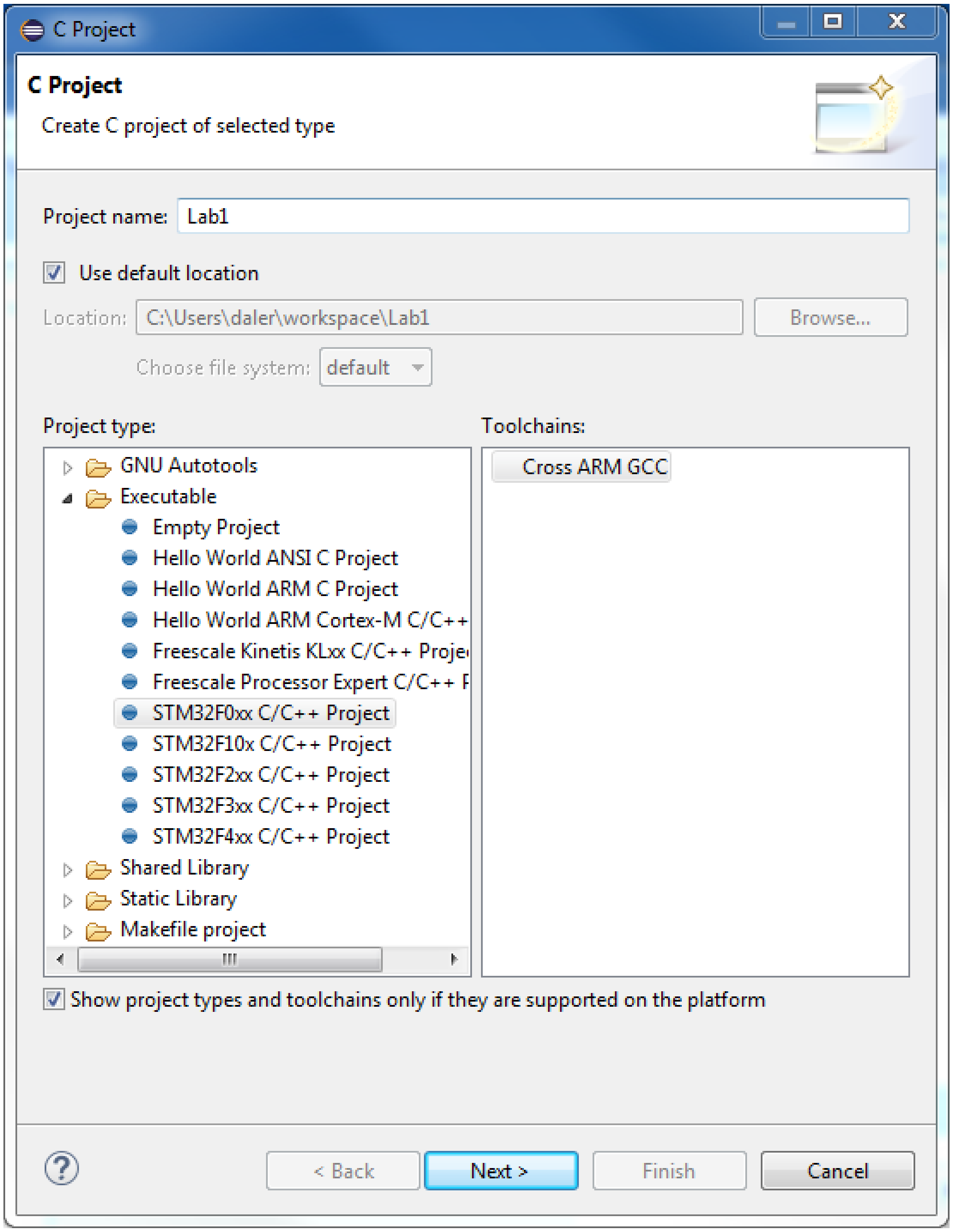
An outline is not available.

0 items

Description	Resource	Path	Location	Type

0 items selected

4. In the ECLIPSE window, select **File** > **New** > **Project** > **C/C++** > **C Project**. In the C PROJECT window, type ***Lab1*** in the "Project name" box. In the "Project type" panel, select ***STM32F0xx C/C++ Project***. Click **Next.**


C Project

C Project

Create C project of selected type

Project name: Lab1

[✓] Use default location

Location: C:\Users\daler\workspace\Lab1

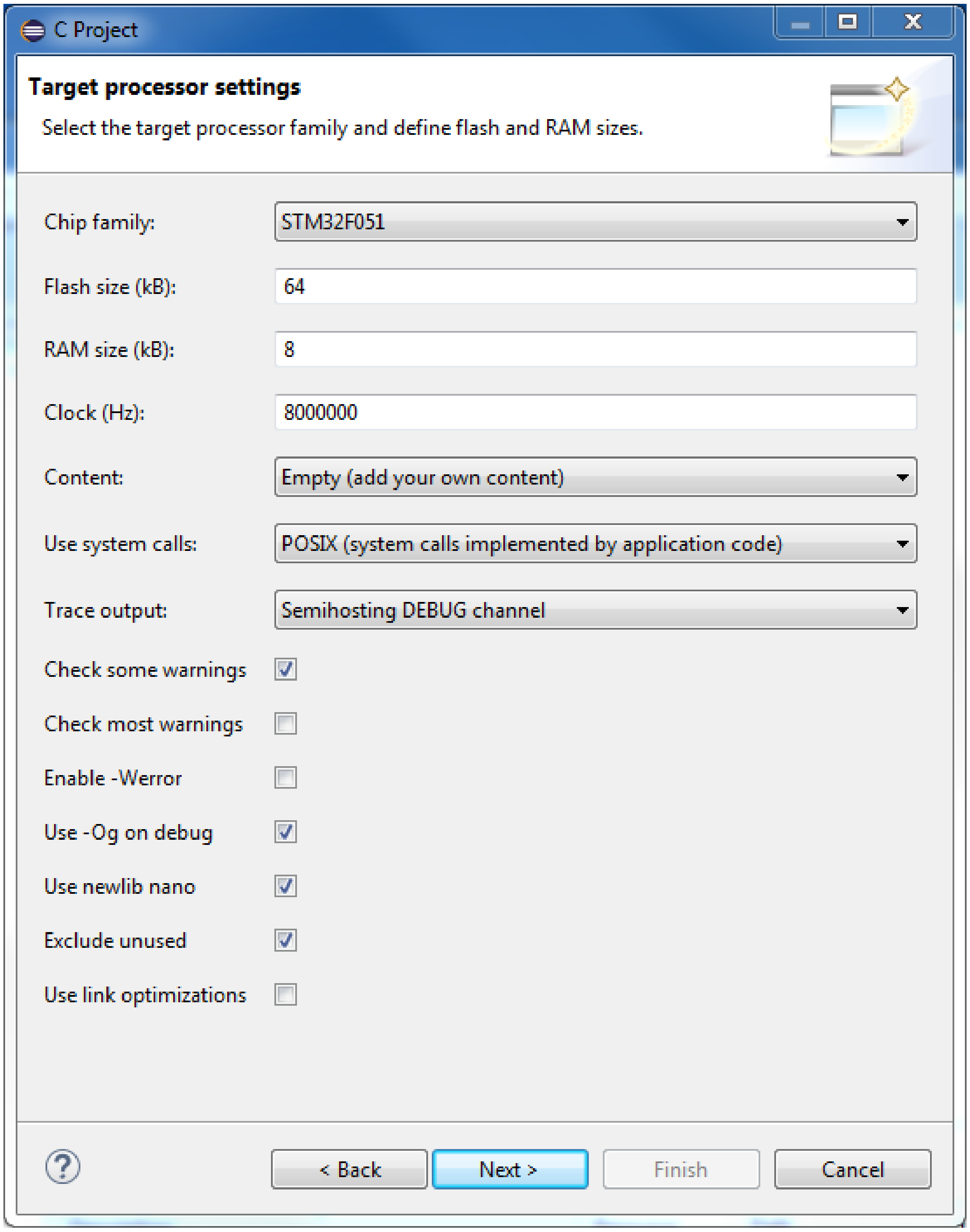
Choose file system: default

Project type:
- GNU Autotools
- Executable
  - Empty Project
  - Hello World ANSI C Project
  - Hello World ARM C Project
  - Hello World ARM Cortex-M C/C++
  - Freescale Kinetis KLxx C/C++ Proje
  - Freescale Processor Expert C/C++ F
  - STM32F0xx C/C++ Project
  - STM32F10x C/C++ Project
  - STM32F2xx C/C++ Project
  - STM32F3xx C/C++ Project
  - STM32F4xx C/C++ Project
- Shared Library
- Static Library
- Makefile project

Toolchains: Cross ARM GCC

[✓] Show project types and toolchains only if they are supported on the platform

5. 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.**

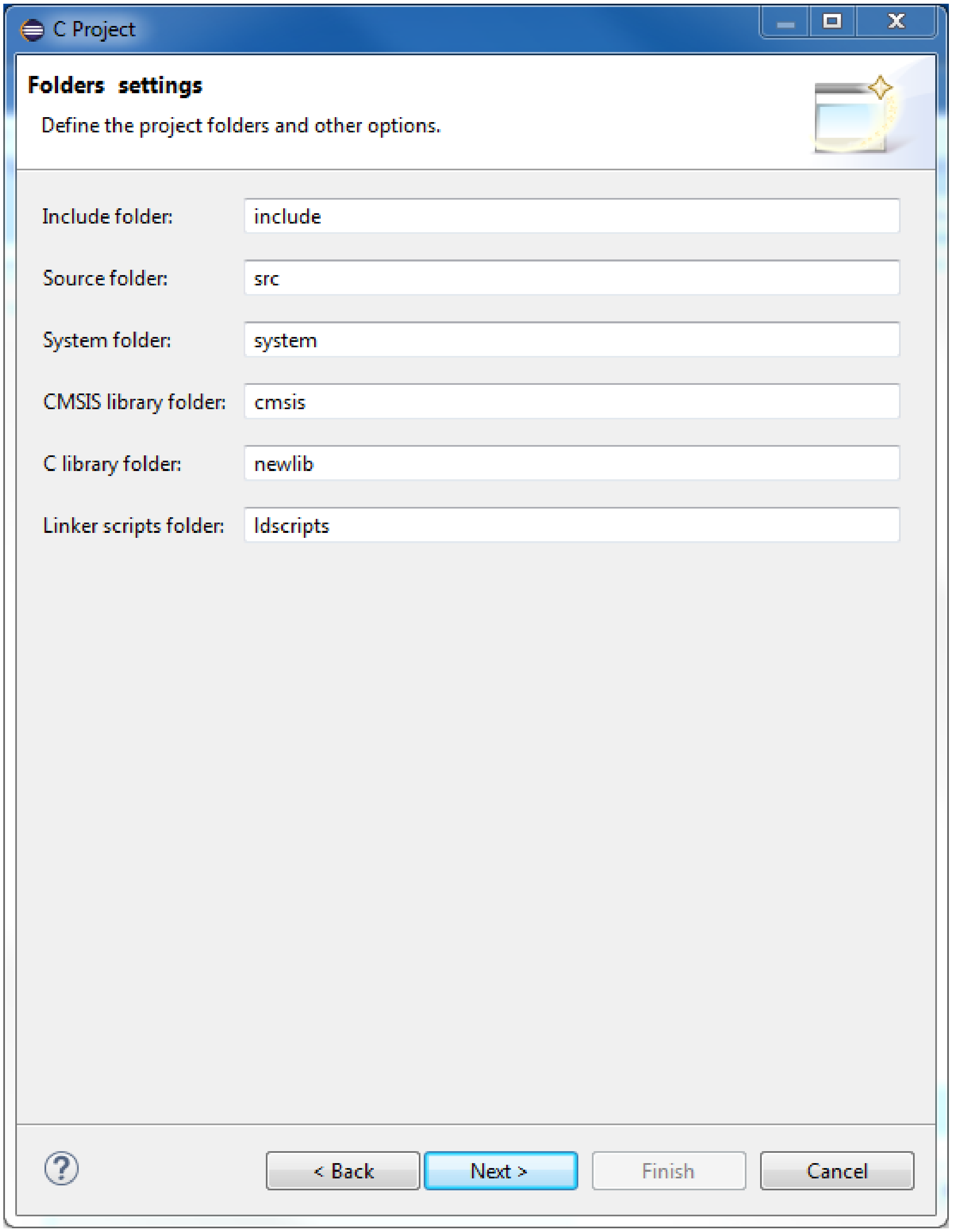


Target processor settings

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	☐

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



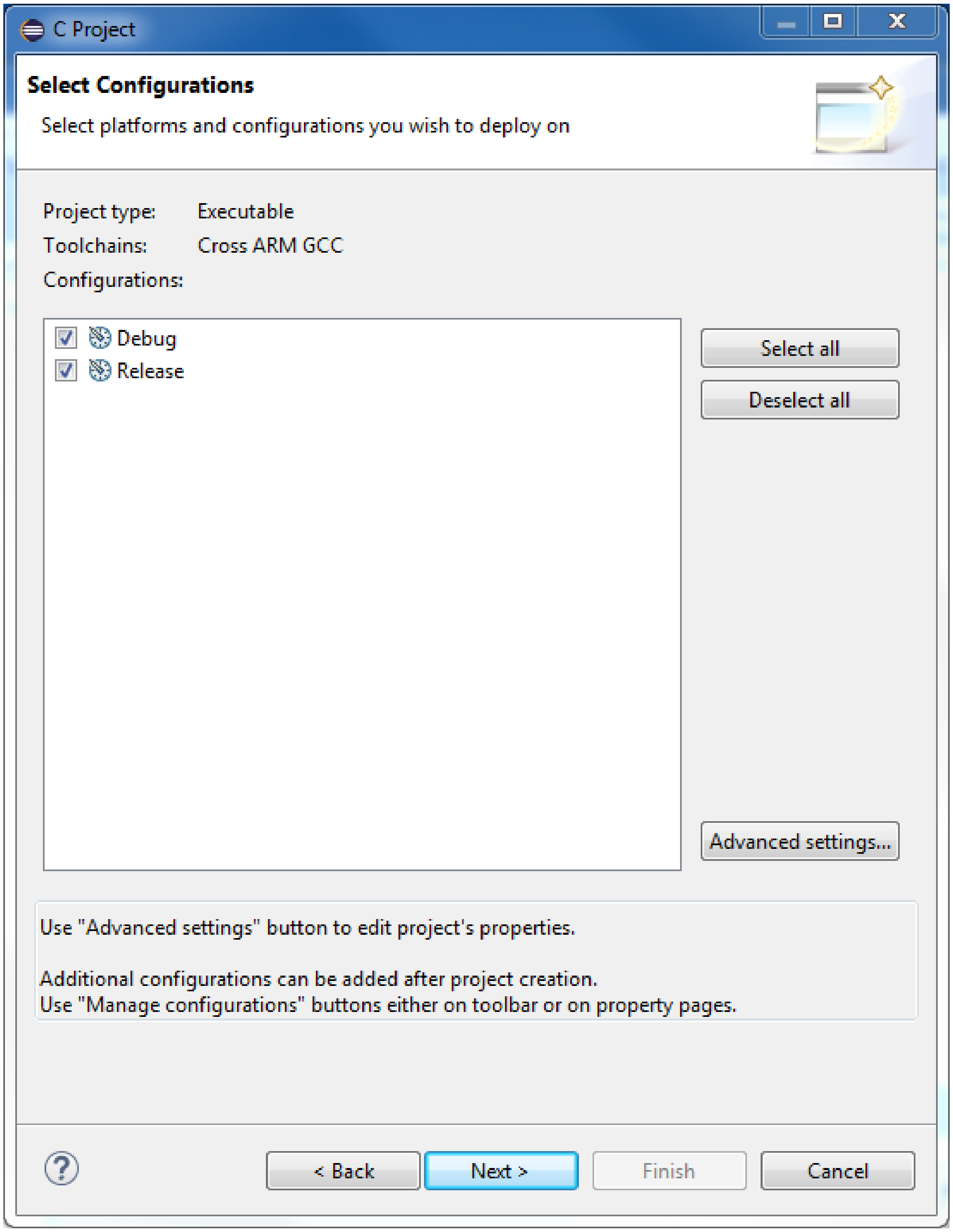
C Project

Folders settings

Define the project folders and other options.

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

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


C Project

Select Configurations

Select platforms and configurations you wish to deploy on

Project type: Executable

Toolchains: Cross ARM GCC

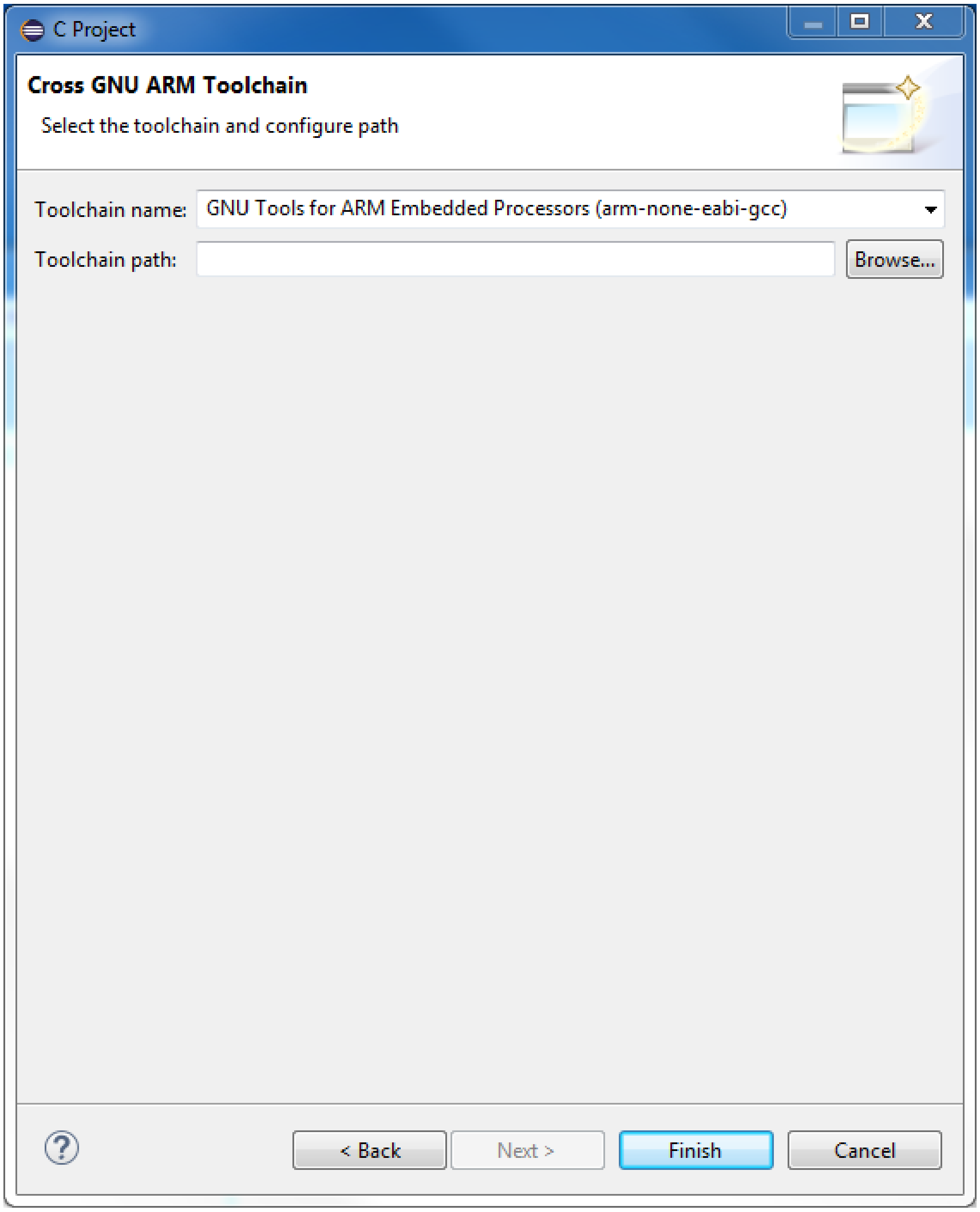
Configurations:

- [✓] Debug
- [✓] Release

Use "Advanced settings" button to edit project's properties.

Additional configurations can be added after project creation.
Use "Manage configurations" buttons either on toolbar or on property pages.

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


C Project

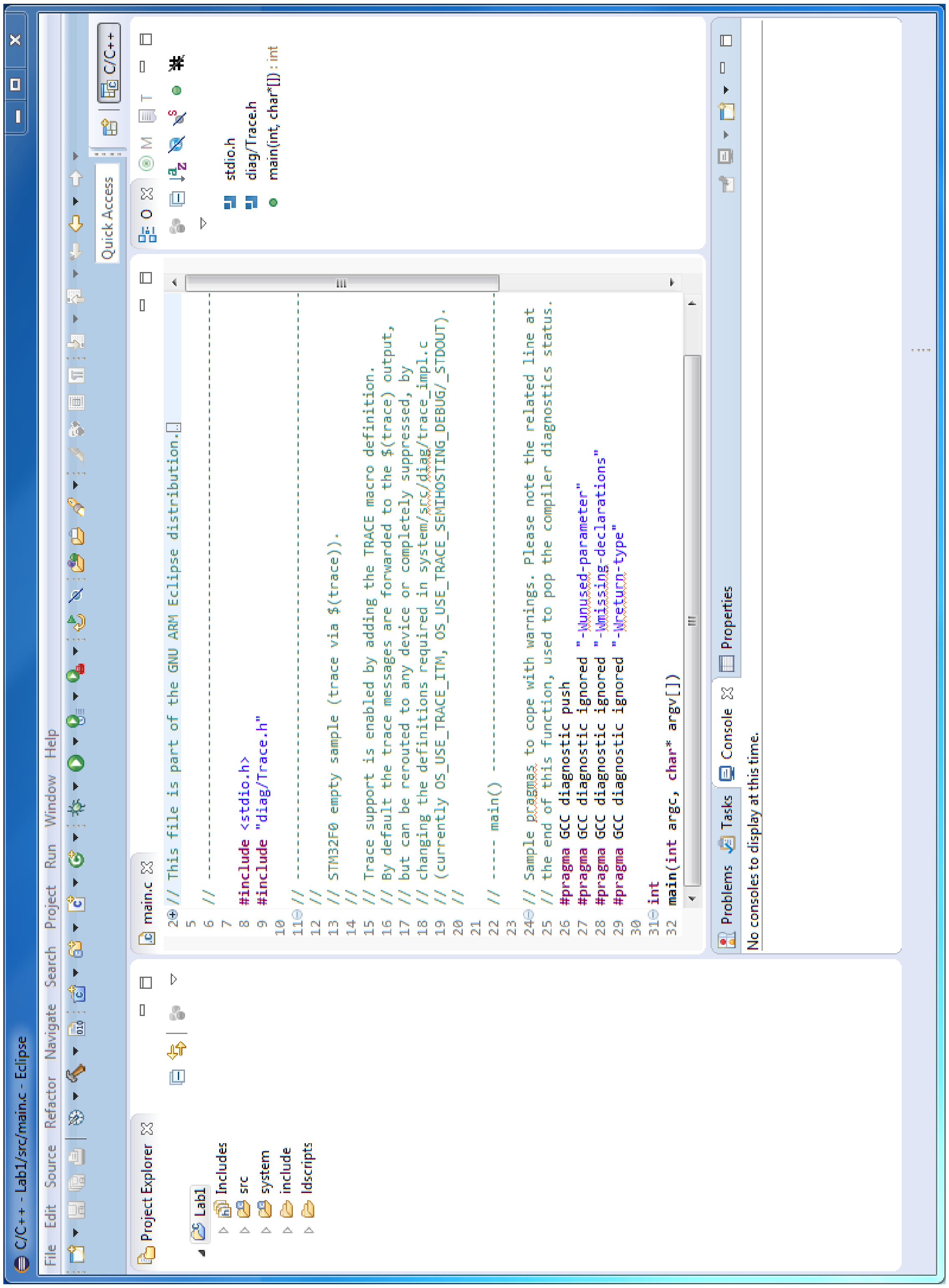
Cross GNU ARM Toolchain

Select the toolchain and configure path

Toolchain name: GNU Tools for ARM Embedded Processors (arm-none-eabi-gcc)

Toolchain path:

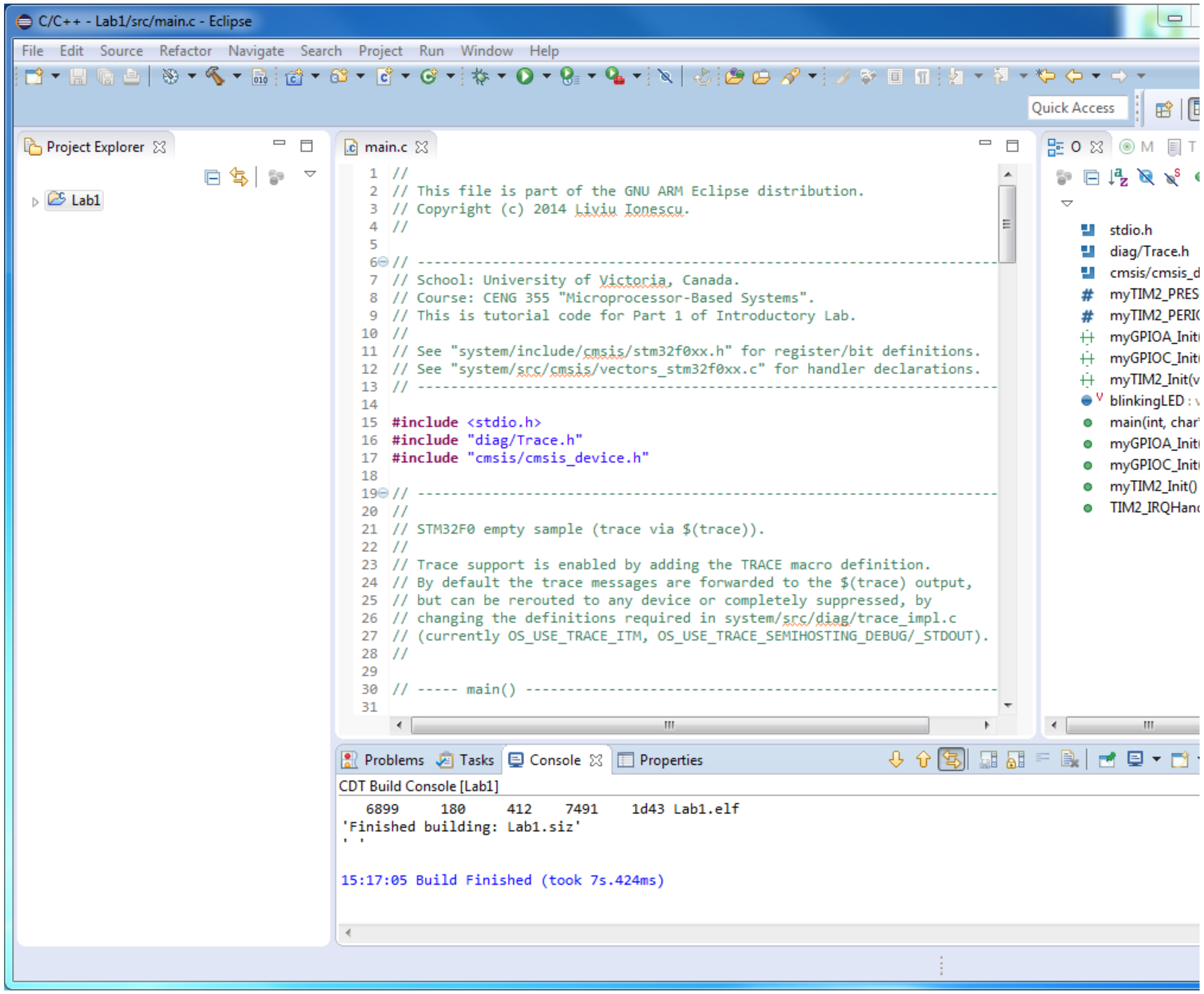
9. In the ECLIPSE window, expand the **Lab1** folder in the “Project Explorer” tab in the left panel and switch to the “Console” tab in the bottom panel (see below).



```
// This file is part of the GNU ARM Eclipse distribution.
//
// ----------------------------------------------------------------------------

#include <stdio.h>
#include "diag/Trace.h"

// ----------------------------------------------------------------------------
//
// STM32F0 empty sample (trace via $(trace)).
//
// Trace support is enabled by adding the TRACE macro definition.
// By default the trace messages are forwarded to the $(trace) output,
// but can be rerouted to any device or completely suppressed, by
// changing the definitions required in system/src/diag/trace_impl.c
// (currently OS_USE_TRACE_ITM, OS_USE_TRACE_SEMIHOSTING_DEBUG/_STDOUT).
//

// ----- main() ---------------------------------------------------------------

// Sample pragmas to cope with warnings. Please note the related line at
// the end of this function, used to pop the compiler diagnostics status.
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wmissing-declarations"
#pragma GCC diagnostic ignored "-Wreturn-type"

int
main(int argc, char* argv[])
```

```
No consoles to display at this time.
```

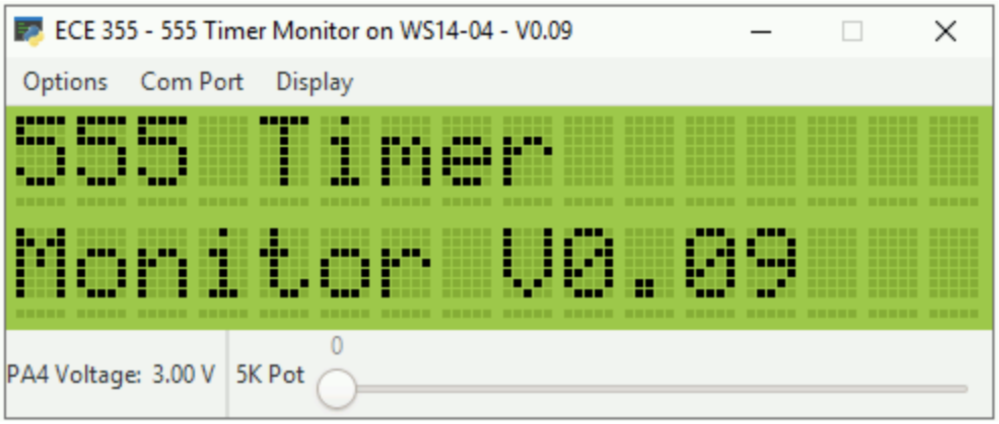
10. In the ECLIPSE window, select the “main.c” tab in the center panel and replace the existing C code with that provided in <u>APPENDIX A</u> (also available on the lab website). Select **File** > **Save** and then select **Project** > **Build Project**, thus compiling and linking your ***main.c*** code. 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.



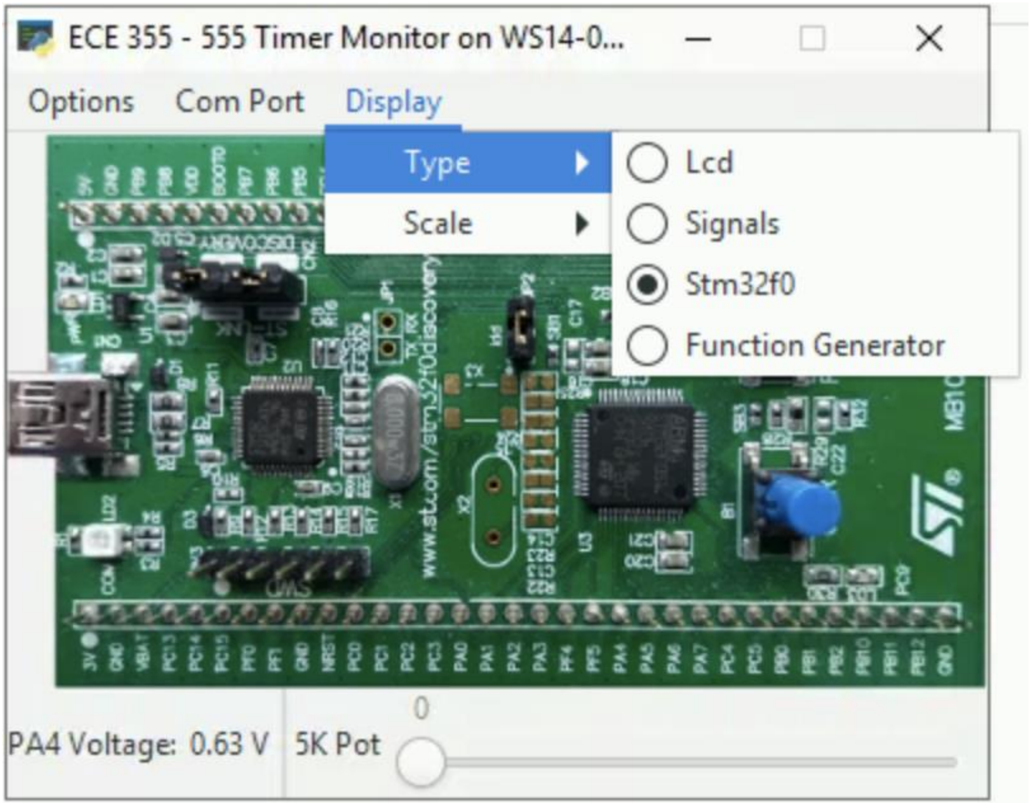
```
//
// This file is part of the GNU ARM Eclipse distribution.
// Copyright (c) 2014 Liviu Ionescu.
//

// ----------------------------------------------------------------------------
// School: University of Victoria, Canada.
// Course: CENG 355 "Microprocessor-Based Systems".
// This is tutorial code for Part 1 of Introductory Lab.
//
// See "system/include/cmsis/stm32f0xx.h" for register/bit definitions.
// See "system/src/cmsis/vectors_stm32f0xx.c" for handler declarations.
// ----------------------------------------------------------------------------

#include <stdio.h>
#include "diag/Trace.h"
#include "cmsis/cmsis_device.h"

// ----------------------------------------------------------------------------
//
// STM32F0 empty sample (trace via $(trace)).
//
// Trace support is enabled by adding the TRACE macro definition.
// By default the trace messages are forwarded to the $(trace) output,
// but can be rerouted to any device or completely suppressed, by
// changing the definitions required in system/src/diag/trace_impl.c
// (currently OS_USE_TRACE_ITM, OS_USE_TRACE_SEMIHOSTING_DEBUG/_STDOUT).
//

// ----- main() ---------------------------------------------------------------

```

```
CDT Build Console [Lab1]
   6899     180     412    7491    1d43 Lab1.elf
'Finished building: Lab1.siz'
' '

15:17:05 Build Finished (took 7s.424ms)
```

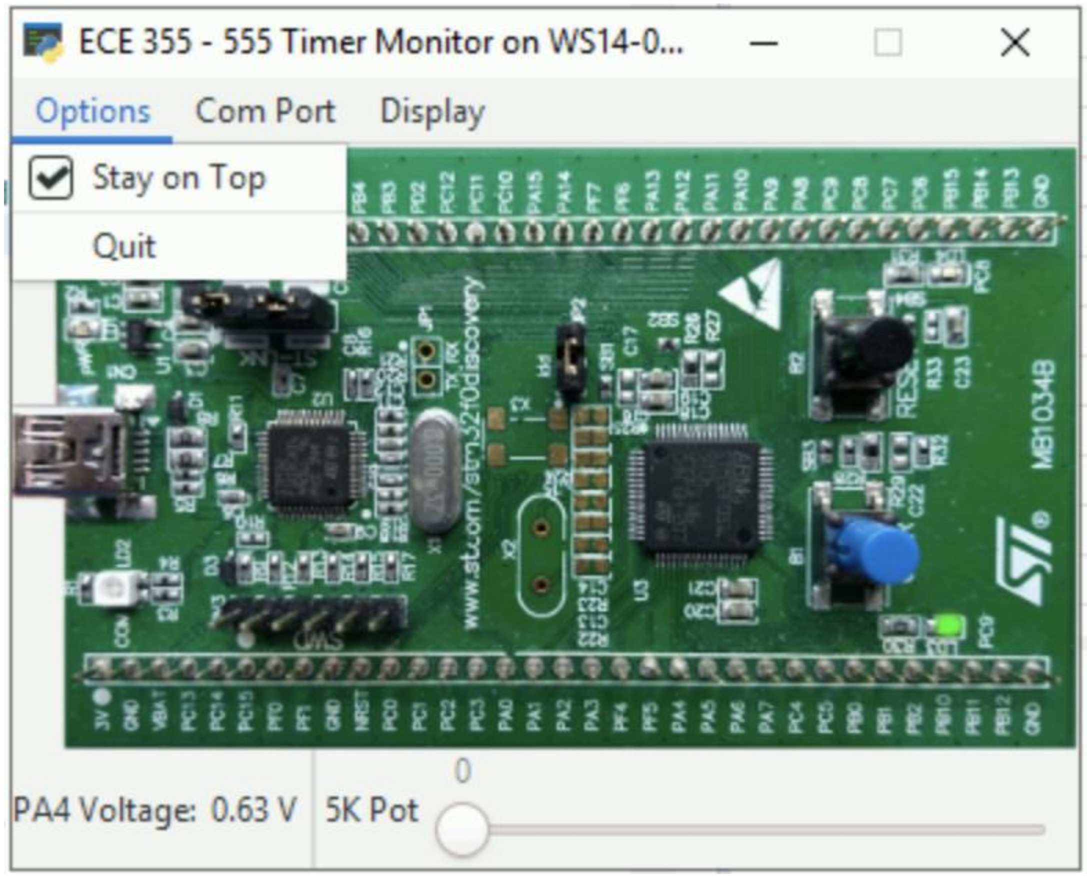
11. Start the **STM32F0 Discovery board interface** by opening the **ECE355** folder on the **Desktop** and double-clicking on **555 timer monitor**, which brings up the ECE 355 - 555 TIMER MONITOR window.



12. Switch the display type from **Lcd** to **Stm32f0**, as shown below.



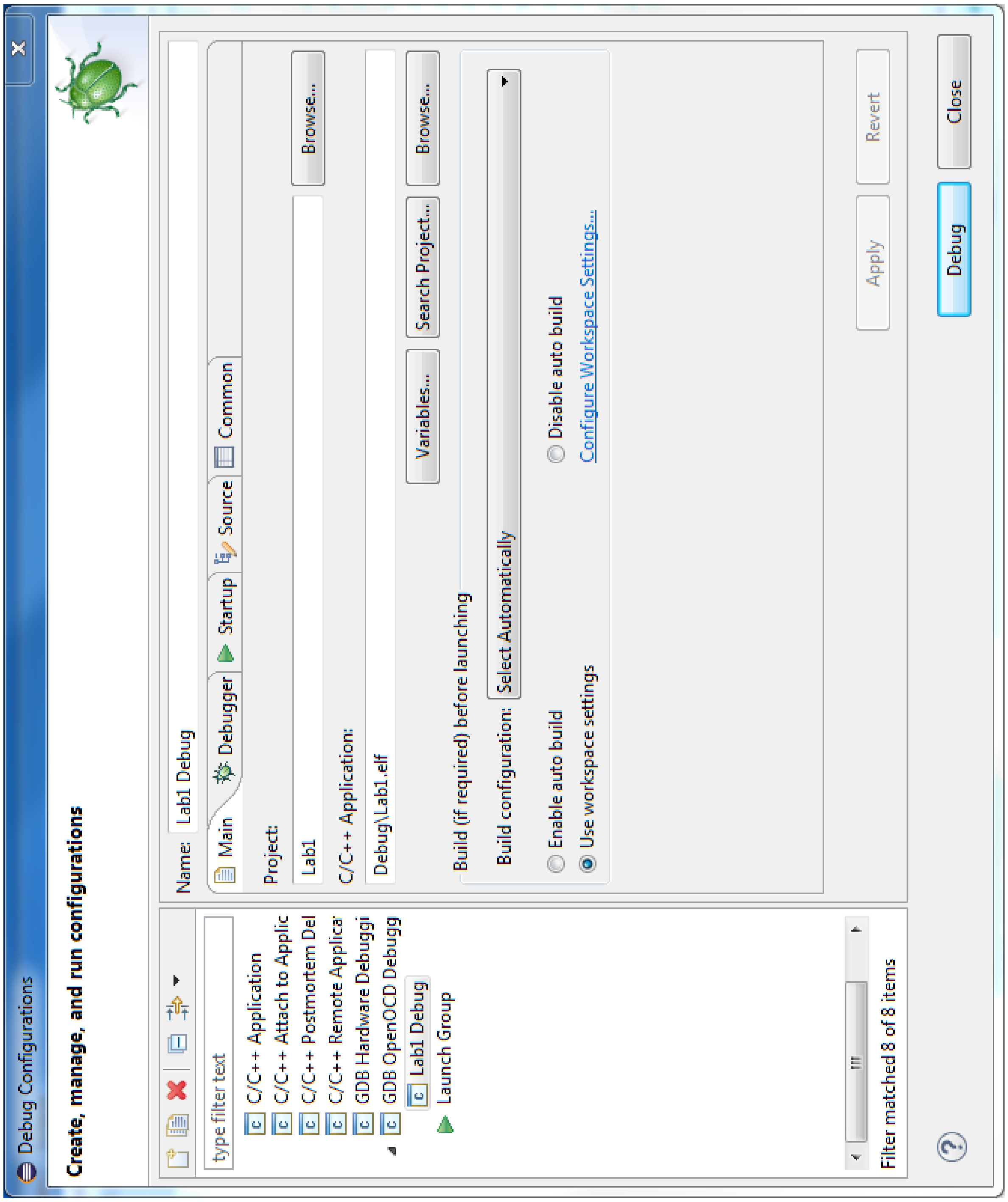
13. Optionally, enable the **Stay on Top** option.



PA4 Voltage: 0.63 V

Continued on the next page…

14. 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:



Name: Lab1 Debug

Project: Lab1

C/C++ Application: Debug\Lab1.elf

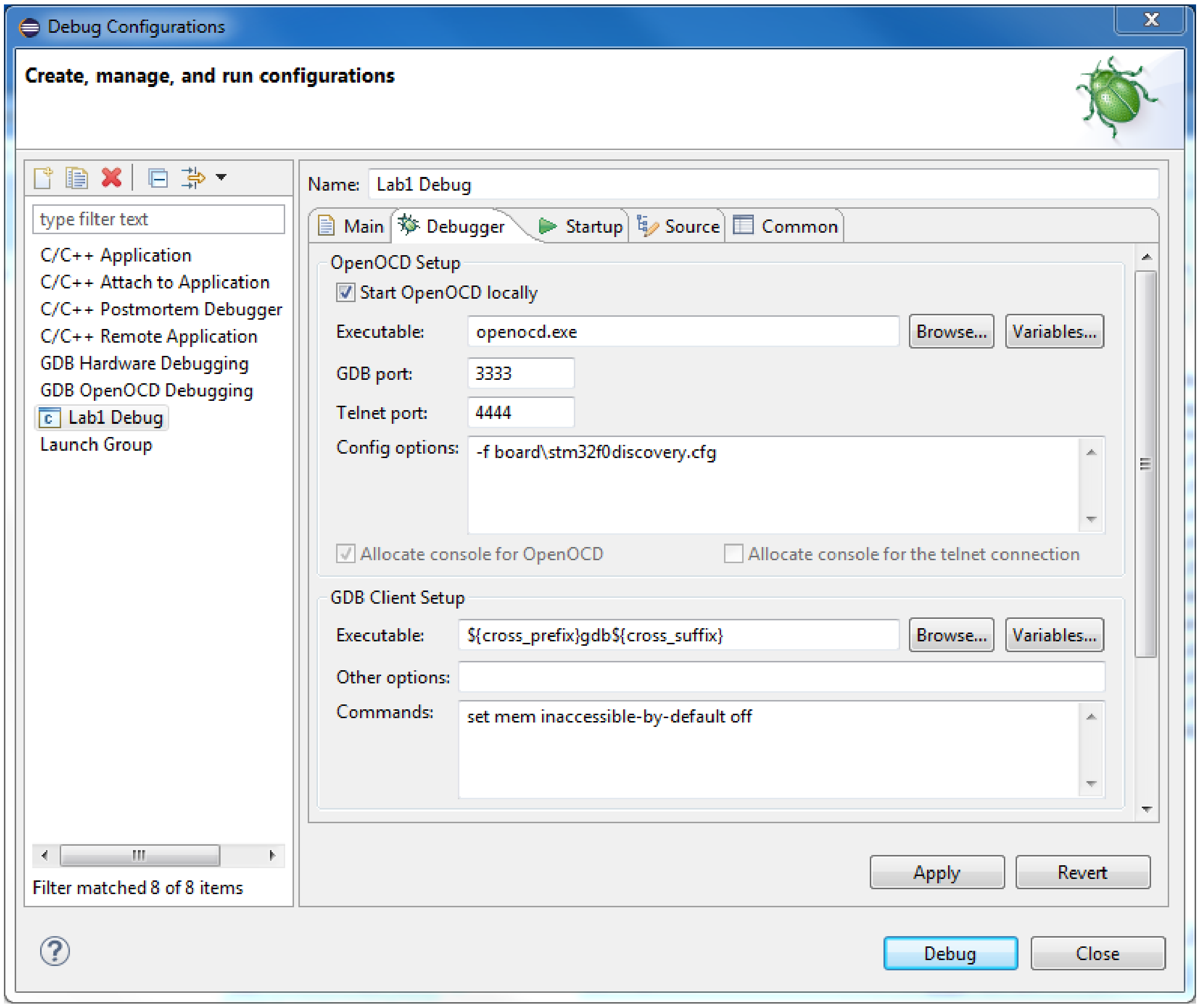
Build (if required) before launching

Build configuration: Select Automatically

( ) Enable auto build ( ) Disable auto build

(●) Use workspace settings Configure Workspace Settings...

15. In the Debug Configurations window, select the "Debugger" tab and ensure that your selections are as shown below. Click **Apply**.



Name: Lab1 Debug

OpenOCD Setup

☑ Start OpenOCD locally

Executable: openocd.exe

GDB port: 3333

Telnet port: 4444

Config options: -f board\stm32f0discovery.cfg

☑ Allocate console for OpenOCD ☐ Allocate console for the telnet connection

GDB Client Setup

Executable: ${cross_prefix}gdb${cross_suffix}

Other options:

Commands: set mem inaccessible-by-default off

OpenOCD Setup – "Executable" Box:

openocd.exe

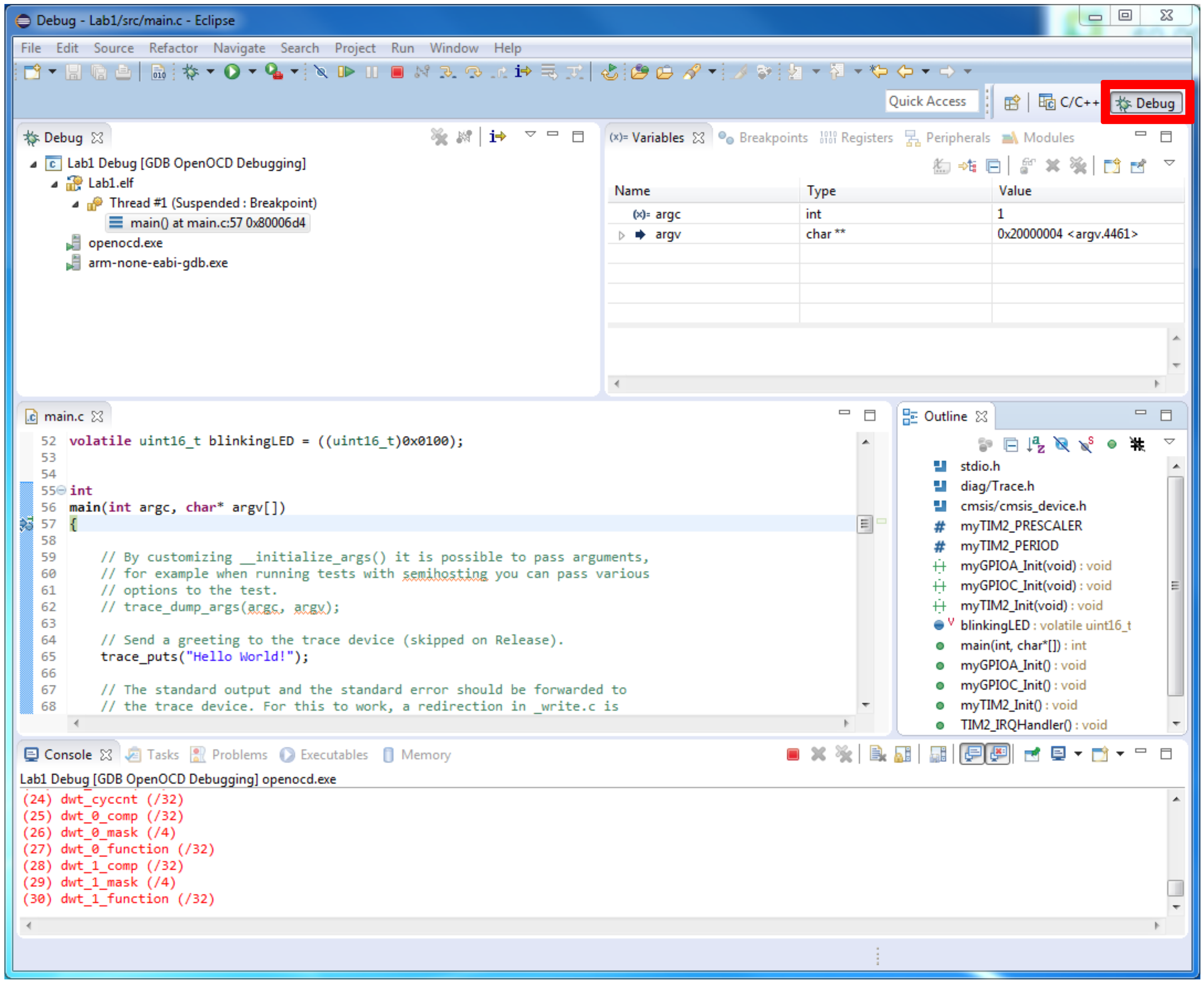
OpenOCD Setup – "Config options" Box:

-f board\stm32f0discovery.cfg

GDB Client Setup – "Executable" Box:

${cross_prefix}gdb${cross_suffix}

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



```
Lab1 Debug [GDB OpenOCD Debugging]
  Lab1.elf
    Thread #1 (Suspended : Breakpoint)
      main() at main.c:57 0x80006d4
  openocd.exe
  arm-none-eabi-gdb.exe
```

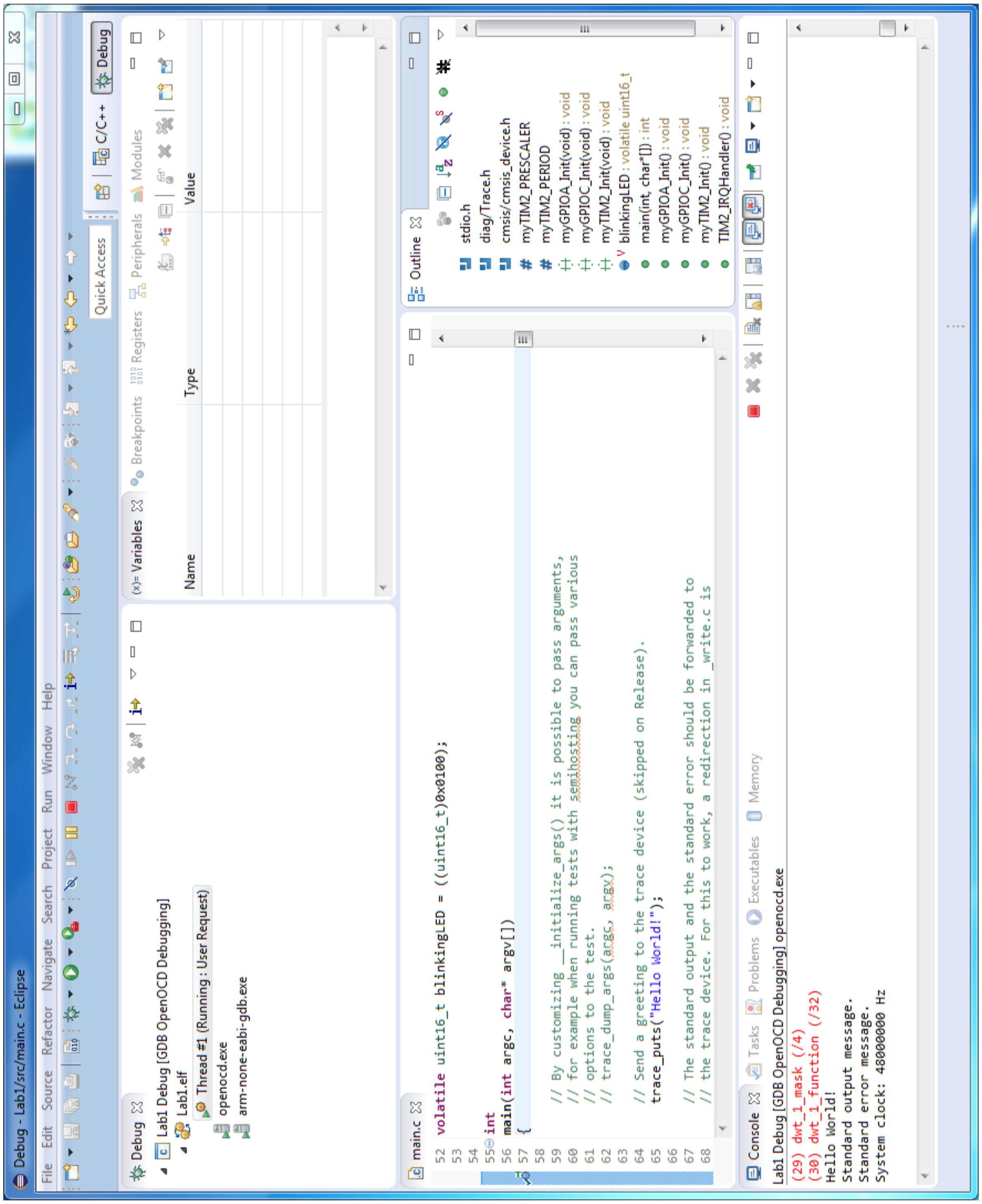
Name	Type	Value
argc	int	1
argv	char **	0x20000004 <argv.4461>

```
volatile uint16_t blinkingLED = ((uint16_t)0x0100);


int
main(int argc, char* argv[])
{

    // By customizing __initialize_args() it is possible to pass arguments,
    // for example when running tests with semihosting you can pass various
    // options to the test.
    // trace_dump_args(argc, argv);

    // Send a greeting to the trace device (skipped on Release).
    trace_puts("Hello World!");

    // The standard output and the standard error should be forwarded to
    // the trace device. For this to work, a redirection in _write.c is
```

```
stdio.h
diag/Trace.h
cmsis/cmsis_device.h
myTIM2_PRESCALER
myTIM2_PERIOD
myGPIOA_Init(void) : void
myGPIOC_Init(void) : void
myTIM2_Init(void) : void
blinkingLED : volatile uint16_t
main(int, char*[]) : int
myGPIOA_Init() : void
myGPIOC_Init() : void
myTIM2_Init() : void
TIM2_IRQHandler() : void
```

```
Lab1 Debug [GDB OpenOCD Debugging] openocd.exe
(24) dwt_cyccnt (/32)
(25) dwt_0_comp (/32)
(26) dwt_0_mask (/4)
(27) dwt_0_function (/32)
(28) dwt_1_comp (/32)
(29) dwt_1_mask (/4)
(30) dwt_1_function (/32)
```

17. 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.



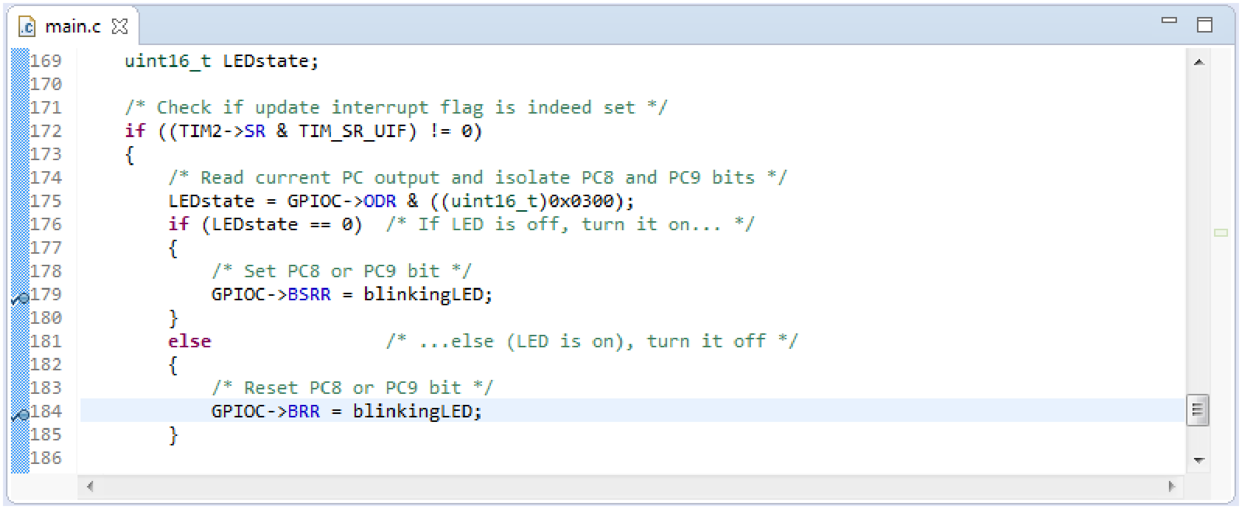
```
volatile uint16_t blinkingLED = ((uint16_t)0x0100);


int
main(int argc, char* argv[])
{
  // By customizing __initialize_args() it is possible to pass arguments,
  // for example when running tests with semihosting you can pass various
  // options to the test.
  // trace_dump_args(argc, argv);

  // Send a greeting to the trace device (skipped on Release).
  trace_puts("Hello World!");

  // The standard output and the standard error should be forwarded to
  // the trace device. For this to work, a redirection in _write.c is
```

```
Lab1 Debug [GDB OpenOCD Debugging] openocd.exe
(29) dwt_1_mask (/4)
(30) dwt_1_function (/32)
Hello World!
Standard output message.
Standard error message.
System clock: 48000000 Hz
```

18. 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.

19. In the ECLIPSE window, select **Run** > **Suspend**. In the "main.c" tab in the center panel, set breakpoints at lines 179 and 184 by double-clicking on them (see below).



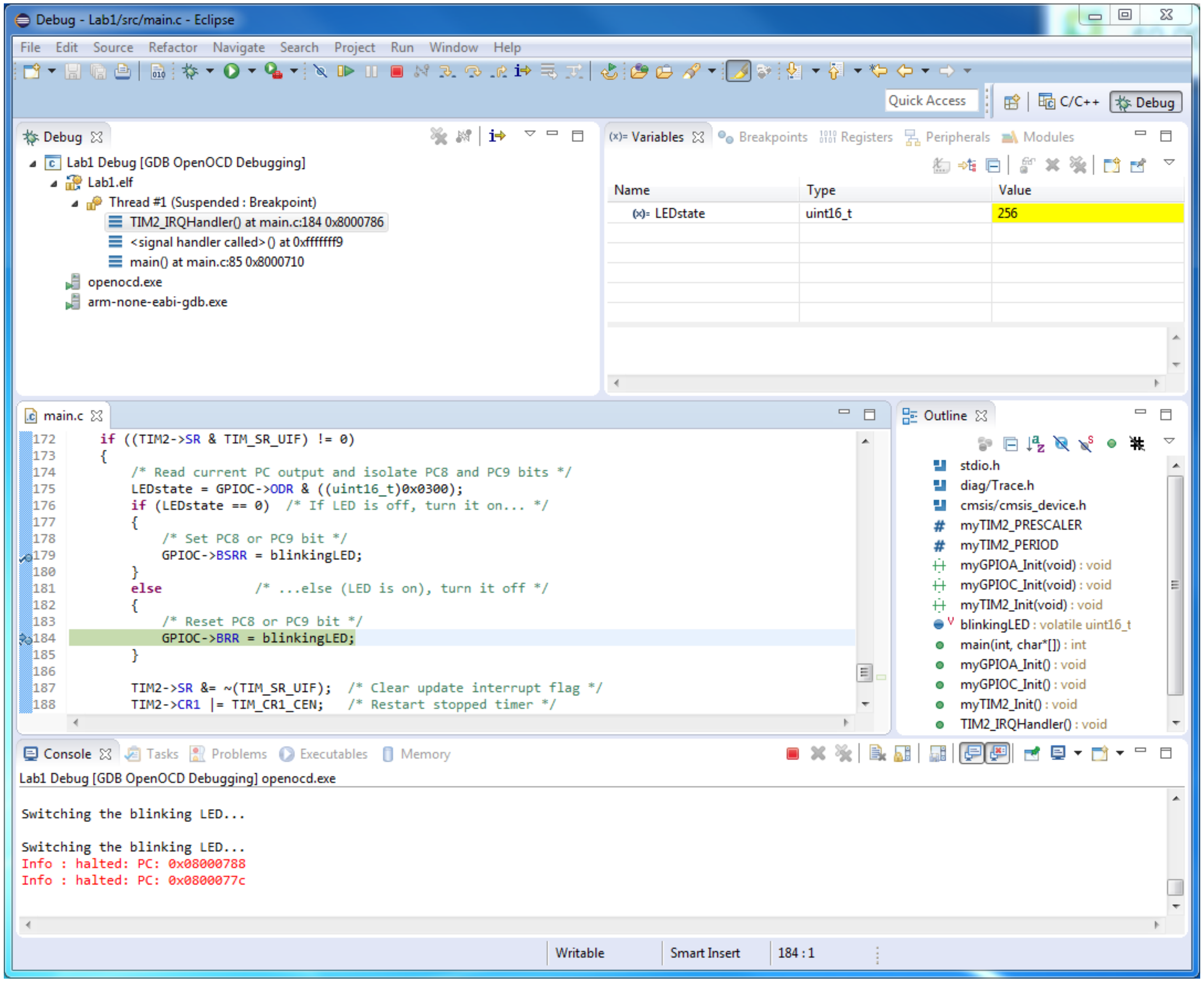
```
169     uint16_t LEDstate;
170
171     /* Check if update interrupt flag is indeed set */
172     if ((TIM2->SR & TIM_SR_UIF) != 0)
173     {
174         /* Read current PC output and isolate PC8 and PC9 bits */
175         LEDstate = GPIOC->ODR & ((uint16_t)0x0300);
176         if (LEDstate == 0)  /* 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
```

20. In the ECLIPSE window, select **Run** > **Resume**. The program execution should stop either at line 179 (the blue LED is off) or at line 184 (the blue LED is on).

If the program execution stops at line 179, the ***LEDstate*** variable should be equal to ***0***.

If the program execution stops at line 184, 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.



Debug - Lab1/src/main.c - Eclipse

Lab1 Debug [GDB OpenOCD Debugging]
Lab1.elf
Thread #1 (Suspended : Breakpoint)
TIM2_IRQHandler() at main.c:184 0x8000786
<signal handler called>() at 0xfffffff9
main() at main.c:85 0x8000710
openocd.exe
arm-none-eabi-gdb.exe

Name	Type	Value
LEDstate	uint16_t	256

```
    if ((TIM2->SR & TIM_SR_UIF) != 0)
    {
        /* Read current PC output and isolate PC8 and PC9 bits */
        LEDstate = GPIOC->ODR & ((uint16_t)0x0300);
        if (LEDstate == 0)  /* If LED is off, turn it on... */
        {
            /* Set PC8 or PC9 bit */
            GPIOC->BSRR = blinkingLED;
        }
        else            /* ...else (LED is on), turn it off */
        {
            /* Reset PC8 or PC9 bit */
            GPIOC->BRR = blinkingLED;
        }

        TIM2->SR &= ~(TIM_SR_UIF);  /* Clear update interrupt flag */
        TIM2->CR1 |= TIM_CR1_CEN;   /* Restart stopped timer */
```

stdio.h
diag/Trace.h
cmsis/cmsis_device.h
myTIM2_PRESCALER
myTIM2_PERIOD
myGPIOA_Init(void) : void
myGPIOC_Init(void) : void
myTIM2_Init(void) : void
blinkingLED : volatile uint16_t
main(int, char*[]) : int
myGPIOA_Init() : void
myGPIOC_Init() : void
myTIM2_Init() : void
TIM2_IRQHandler() : void

Lab1 Debug [GDB OpenOCD Debugging] openocd.exe

```
Switching the blinking LED...

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

Writable Smart Insert 184 : 1

21. In the "main.c" tab in the center panel of the ECLIPSE window, remove breakpoints at lines 179 and 184 by double-clicking on them. Select **Run > Resume** (the blue LED starts blinking again).

22. Press the USER button, so that the green LED is blinking. Repeat steps **19** and **20**. Whenever the green LED is on (i.e., whenever the program execution stops at line 184), the ***LEDstate*** variable should be equal to ***512*** (***0x0000200*** in hex). Repeat step **21** (the green LED starts blinking again).

23. 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**.

24. 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**.

25. **IMPORTANT:** Copy your ***workspace*** folder from the **C:** drive into the **M:** drive for safekeeping.

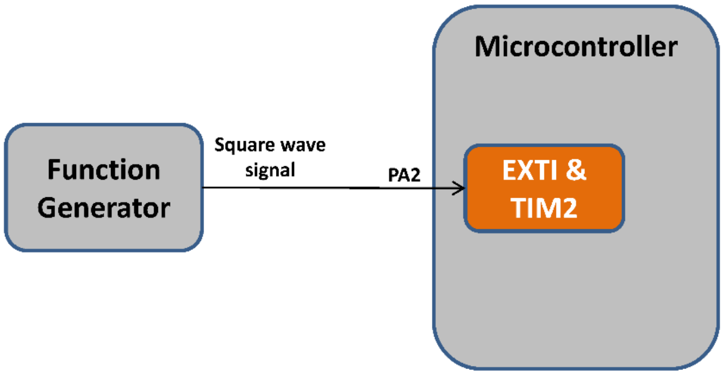
This is the end of Part 1 of the introductory lab, where you have learned how to use the ECLIPSE SDK to build, run, and debug your embedded software code targeting the STM32F0 Discovery board.

Additional up-to-date information can be found on the ECE 355 lab website: http://www.ece.uvic.ca/~ece355/lab.

Part 2: Signal Frequency Measurement

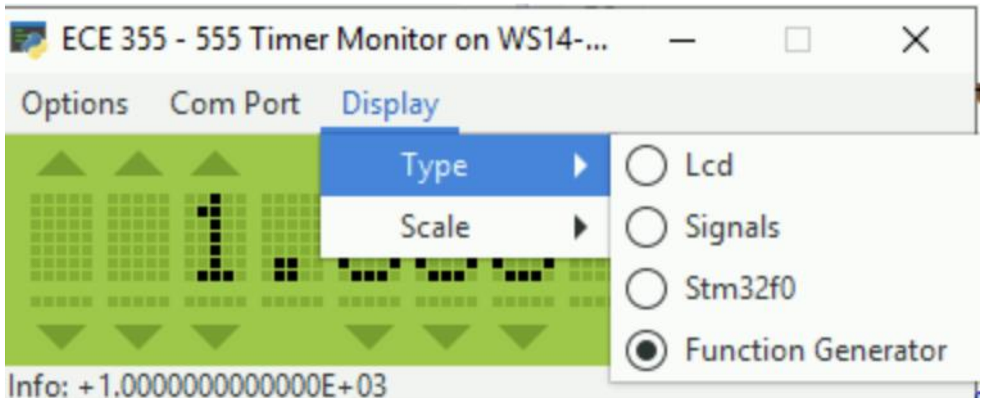
Objective

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



Specifications

- To start the Function Generator, open the ECE 355 - 555 TIMER MONITOR window and switch the display to **Function Generator**. The Function Generator will produce a square wave input signal with the amplitude ranging from 0 to +3.3 V.



Display > Type: Lcd [ ] Signals [ ] Stm32f0 [ ] [✓ Function Generator]

Info: +1.000000000000E+03

- Your 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 at during 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 study 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.

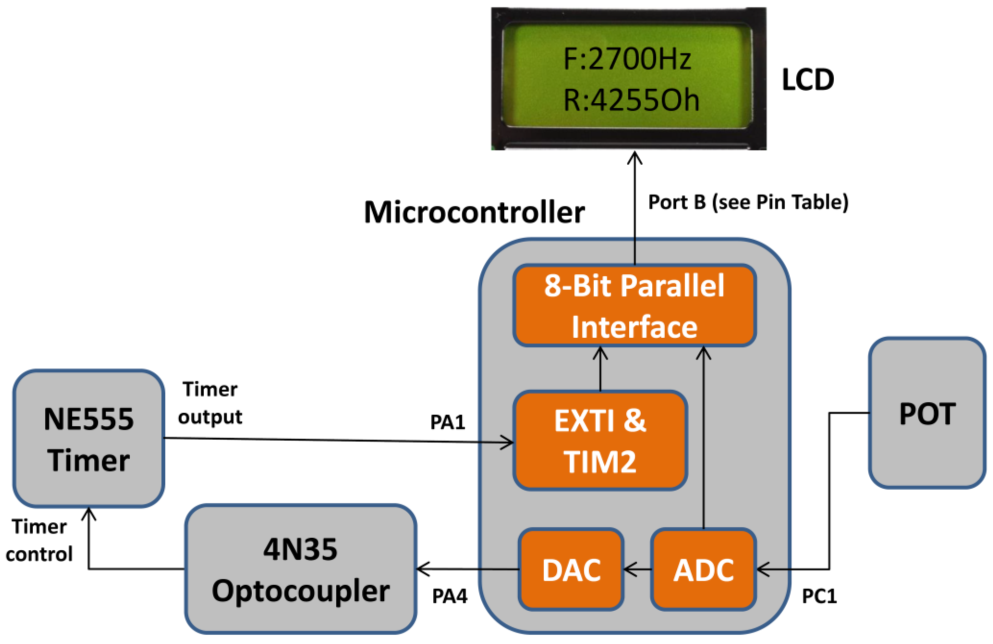
- **IMPORTANT:** You can find a basic code template (to get you started) on the ECE 355 lab website:

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

PROJECT: PWM Signal Generation and Monitoring System

Objective

You are to develop an embedded system for monitoring and controlling a pulse-width-modulated (PWM) signal generated by an external 555 timer (**NE555** IC). An external optocoupler (**4N35** IC), driven by the microcontroller on the **STMF0 Discovery** board, will be used to control the frequency of the PWM signal. The microcontroller will be used to measure the voltage across a potentiometer (POT) on the **ECE 355 Emulation** board and relay it to the external optocoupler for controlling the PWM signal frequency. The microcontroller will also be used to measure the frequency of the generated PWM signal. The measured timer frequency and the corresponding POT resistance are to be displayed on the LCD on the **ECE 355 Emulation** board. The overall system diagram is shown below.



Specifications

- In Part 2 of the introductory lab, you have used the Function Generator to generate a square-wave signal and to display its period and frequency on the console. For this project, you will use the 555 timer (instead of the Function Generator) to generate the square-wave signal, and you will use the LCD (instead of the console) to display the signal frequency and the POT resistance. You should be able to measure the signal frequency reusing most of the code you have developed in Part 2.
  IMPORTANT: You must use **PA1** with **EXTI1** for the 555 timer signal, as opposed to **PA2** with **EXTI2** used for the Function Generator signal in Part 2 of **Lab 1**.

- The STM32F051R8T6 MCU mounted on the **STM32F0 Discovery** board features built-in 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 of the 555 timer, based on the potentiometer voltage read by the ADC.

- The analog voltage signal coming from the potentiometer on the **ECE 355 Emulation** board will be measured continuously by the ADC, which 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 adjust 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 555 timer signal frequency and the potentiometer resistance, you will use an 8-bit parallel interface to communicate with the LCD emulator. The interface includes 4 control signals (**ENB**, **RS**, **R/W**, **DONE**) and 8 data signals (**D0-D7**). These LCD 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 (for Part 2 only)**	**INPUT**
PA4	**DAC**	**OUTPUT (Analog)**
PC1	**ADC**	**INPUT (Analog)**
PB4	**ENB (LCD Handshaking: “Enable”)**	**OUTPUT**
PB5	**RS (0 = COMMAND, 1 = DATA)**	**OUTPUT**
PB6	**R/W (0 = WRITE, 1 = READ)**	**OUTPUT**
PB7	**DONE (LCD Handshaking: “Done”)**	**INPUT**
PB8	**D0**	**OUTPUT**
PB9	**D1**	**OUTPUT**
PB10	**D2**	**OUTPUT**
PB11	**D3**	**OUTPUT**
PB12	**D4**	**OUTPUT**
PB13	**D5**	**OUTPUT**
PB14	**D6**	**OUTPUT**
PB15	**D7**	**OUTPUT**

Deliverables

- **Demonstration.** You must demonstrate a working project during 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 you a series of technical questions. Your 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.

- **Report.** A substantial project report is required from each student – it should be structured as a standard engineering technical report. The report 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. 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 study Chapters 7, 9, 12, 13, 14, 17 of the **STM32F0xx Reference Manual**, as well as the **NE555** timer data sheet and the **4N35** optocoupler data sheet. The emulated LCD interface borrows a major part of its functionality from the Hitachi **HD44780** LCD reference manual.

- **IMPORTANT:** Do NOT use **PA13** and **PA14**. They are reserved for communicating with the ST-LINK chip on the **STMF0 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.

APPENDIX A: File *main.c* for Part 1 of the Introductory Lab

```
//
// This file is part of the GNU ARM Eclipse distribution.
// Copyright (c) 2014 Liviu Ionescu.
//

// ----------------------------------------------------------------------------
// School: University of Victoria, Canada.
// Course: ECE 355 "Microprocessor-Based Systems".
// This is tutorial code for Part 1 of Introductory Lab.
//
// See "system/include/cmsis/stm32f0xx.h" for register/bit definitions.
// See "system/src/cmsis/vectors_stm32f0xx.c" for handler declarations.
// ----------------------------------------------------------------------------

#include <stdio.h>
#include "diag/Trace.h"
#include "cmsis/cmsis_device.h"

// ----------------------------------------------------------------------------
//
// STM32F0 empty sample (trace via $(trace)).
//
// Trace support is enabled by adding the TRACE macro definition.
// By default the trace messages are forwarded to the $(trace) output,
// but can be rerouted to any device or completely suppressed, by
// changing the definitions required in system/src/diag/trace_impl.c
// (currently OS_USE_TRACE_ITM, OS_USE_TRACE_SEMIHOSTING_DEBUG/_STDOUT).
//

// ----- main() ---------------------------------------------------------------

// Sample pragmas to cope with warnings. Please note the related line at
// the end of this function, used to pop the compiler diagnostics status.
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wmissing-declarations"
#pragma GCC diagnostic ignored "-Wreturn-type"


/* Clock prescaler for TIM2 timer: no prescaling */
#define myTIM2_PRESCALER ((uint16_t)0x0000)
/* Delay count for TIM2 timer: 1/4 sec at 48 MHz */
#define myTIM2_PERIOD ((uint32_t)12000000)


void myGPIOA_Init(void);
void myGPIOC_Init(void);
void myTIM2_Init(void);


/* Global variable indicating which LED is blinking */
volatile uint16_t blinkingLED = ((uint16_t)0x0100);
```

```
int
main(int argc, char* argv[])
{

	// By customizing __initialize_args() it is possible to pass arguments,
	// for example when running tests with semihosting you can pass various
	// options to the test.
	// trace_dump_args(argc, argv);

	// Send a greeting to the trace device (skipped on Release).
	trace_puts("Hello World!");

	// The standard output and the standard error should be forwarded to
	// the trace device. For this to work, a redirection in _write.c is
	// required.
	puts("Standard output message.");
	fprintf(stderr, "Standard error message.\n");

	// At this stage the system clock should have already been configured
	// at high speed.
	trace_printf("System clock: %u Hz\n", SystemCoreClock);


	myGPIOA_Init();			/* Initialize I/O port PA */
	myGPIOC_Init();			/* Initialize I/O port PC */
	myTIM2_Init();			/* Initialize timer TIM2 */

	while (1)
	{
		/* If button is pressed, switch between blue and green LEDs */
		if((GPIOA->IDR & GPIO_IDR_0) != 0)
		{
			/* Wait for button to be released (PA0 = 0) */
			while((GPIOA->IDR & GPIO_IDR_0) != 0){}

			/* Turn off currently blinking LED */
			GPIOC->BRR = blinkingLED;

			/* Switch blinking LED */
			blinkingLED ^= ((uint16_t)0x0300);

			/* Turn on switched LED */
			GPIOC->BSRR = blinkingLED;

			trace_printf("\nSwitching the blinking LED...\n");
		}
	}

	return 0;

}
```

```
void myGPIOA_Init()
{
	/* Enable clock for GPIOA peripheral */
	RCC->AHBENR |= RCC_AHBENR_GPIOAEN;

	/* Configure PA0 as input */
	GPIOA->MODER &= ~(GPIO_MODER_MODER0);
	/* Ensure no pull-up/pull-down for PA0 */
	GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR0);
}


void myGPIOC_Init()
{
	/* Enable clock for GPIOC peripheral */
	RCC->AHBENR |= RCC_AHBENR_GPIOCEN;

	/* Configure PC8 and PC9 as outputs */
	GPIOC->MODER |= (GPIO_MODER_MODER8_0 | GPIO_MODER_MODER9_0);
	/* Ensure no pull-up/pull-down for PC8 and PC9 */
	GPIOC->PUPDR &= ~(GPIO_PUPDR_PUPDR8 | GPIO_PUPDR_PUPDR9);
	/* Ensure push-pull mode selected for PC8 and PC9 */
	GPIOC->OTYPER &= ~(GPIO_OTYPER_OT_8 | GPIO_OTYPER_OT_9);
	/* Ensure high-speed mode for PC8 and PC9 */
	GPIOC->OSPEEDR |= (GPIO_OSPEEDER_OSPEEDR8 | GPIO_OSPEEDER_OSPEEDR9);
}


void myTIM2_Init()
{
	/* Enable clock for TIM2 peripheral */
	RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;

	/* Configure TIM2: buffer auto-reload, count up, stop on overflow,
	 * enable update events, interrupt on overflow only */
	TIM2->CR1 = ((uint16_t)0x008C);

	/* Set clock prescaler value */
	TIM2->PSC = myTIM2_PRESCALER;
	/* Set auto-reloaded delay */
	TIM2->ARR = myTIM2_PERIOD;

	/* Update timer registers */
	TIM2->EGR = ((uint16_t)0x0001);

	/* Assign TIM2 interrupt priority = 0 in NVIC */
	NVIC_SetPriority(TIM2_IRQn, 0);
	// Same as: NVIC->IP[3] = ((uint32_t)0x00FFFFFF);

	/* Enable TIM2 interrupts in NVIC */
	NVIC_EnableIRQ(TIM2_IRQn);
	// Same as: NVIC->ISER[0] = ((uint32_t)0x00008000);

	/* Enable update interrupt generation */
	TIM2->DIER |= TIM_DIER_UIE;
	/* Start counting timer pulses */
	TIM2->CR1 |= TIM_CR1_CEN;
}
```

```
/* This handler is declared in system/src/cmsis/vectors_stm32f0xx.c */
void TIM2_IRQHandler()
{
	uint16_t LEDstate;

	/* Check if update interrupt flag is indeed set */
	if ((TIM2->SR & TIM_SR_UIF) != 0)
	{
		/* Read current PC output and isolate PC8 and PC9 bits */
		LEDstate = GPIOC->ODR & ((uint16_t)0x0300);
		if (LEDstate == 0)  /* If LED is off, turn it on... */
		{
			/* Set PC8 or PC9 bit */
			GPIOC->BSRR = blinkingLED;
		}
		else                /* ...else (LED is on), turn it off */
		{
			/* Reset PC8 or PC9 bit */
			GPIOC->BRR = blinkingLED;
		}

		TIM2->SR &= ~(TIM_SR_UIF);       /* Clear update interrupt flag */
		TIM2->CR1 |= TIM_CR1_CEN;        /* Restart stopped timer */
	}
}


#pragma GCC diagnostic pop

// ----------------------------------------------------------------------------
```