LABORATORY MANUAL

ECE 255

Introduction to Computer Architecture

Laboratory Experiment #0

This manual was prepared by

The many dedicated, motivated, and talented graduate students and faculty members in the Department of Electrical and Computer Engineering

The laboratory experiments are developed to provide a hands-on introduction to the ARM architecture. The labs are based on the open source tools Eclipse and OpenOCD.

You are expected to read each lab experiment manual carefully and prepare **in advance** of your lab session. Pay particular attention to the parts that are **bolded and underlined**. You are required to address these parts in your lab report. In particular, all items in the **Prelab** section must be prepared in a written form **before your lab**. You are required to submit your written preparation during the lab, which will be graded by the lab instructor.

Laboratory Experiment 0: Introduction to Eclipse

1.1 Goal

Eclipse is an integrated development environment (IDE) that can be used to develop applications with various programming languages such as C and C++. With the proper plug-ins, one can develop ARM assembly applications in Eclipse and execute/debug programs on ARM development boards. This is a tutorial to introduce Eclipse and the process of developing a C program to be executed on the STM32F0 Discovery Boards in the lab.

1.2 Eclipse

Double click **eclipse**; this brings up the *workspace launcher* window.

Type **C:\Users\YourUserName\workspace** in the "Workspace" box (Figure 1.1). You need to create all your projects in this directory, otherwise your project would not work.

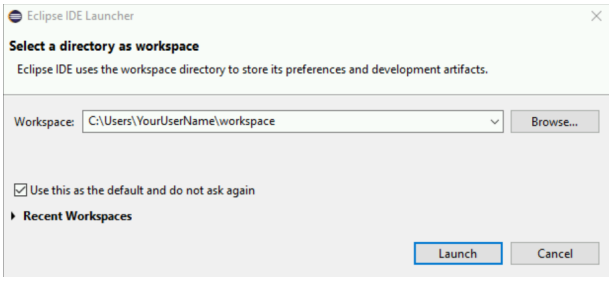

Figure 1.1: Workspace launcher window

Follow the next section to configure Eclipse for the STM32F0 board.

1.3 Part 2: Create a Blinky C Project

1.3.1 Configure Eclipse for STM32F0

You develop and store your source code as projects. To create a project for the C programming language, go to Eclipse menu, **`File ->New`**, and select **C Project**

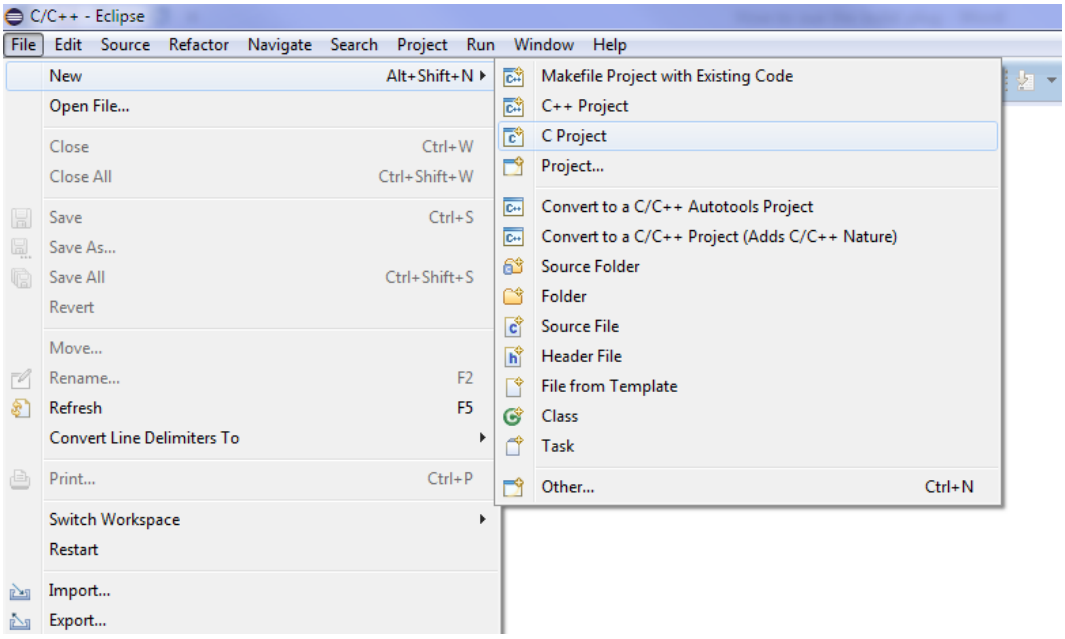

Figure 1.2: New C project

Inside the "C Project" window (Figure 1.3):

- In Project name: enter the name of a new project, for example, **Blinky**
- In Project type: expand the **Executable** type and select **STM32F0xx C/C++ Project**
- In Toolchains: select **Cross ARM GCC**
- Click the **`Next>`** button brings you to the "Target processor settings"

In the "Target processor settings" window, use all the default values (Figure 1.4):

- Chip family: default value (STM32F051) is the target ARM board
- Flash size (KB): the flash memory size of our ARM board
- RAM size (KB): the RAM memory size of our ARM board
- Clock (Hz): the default operating frequency of our ARM board

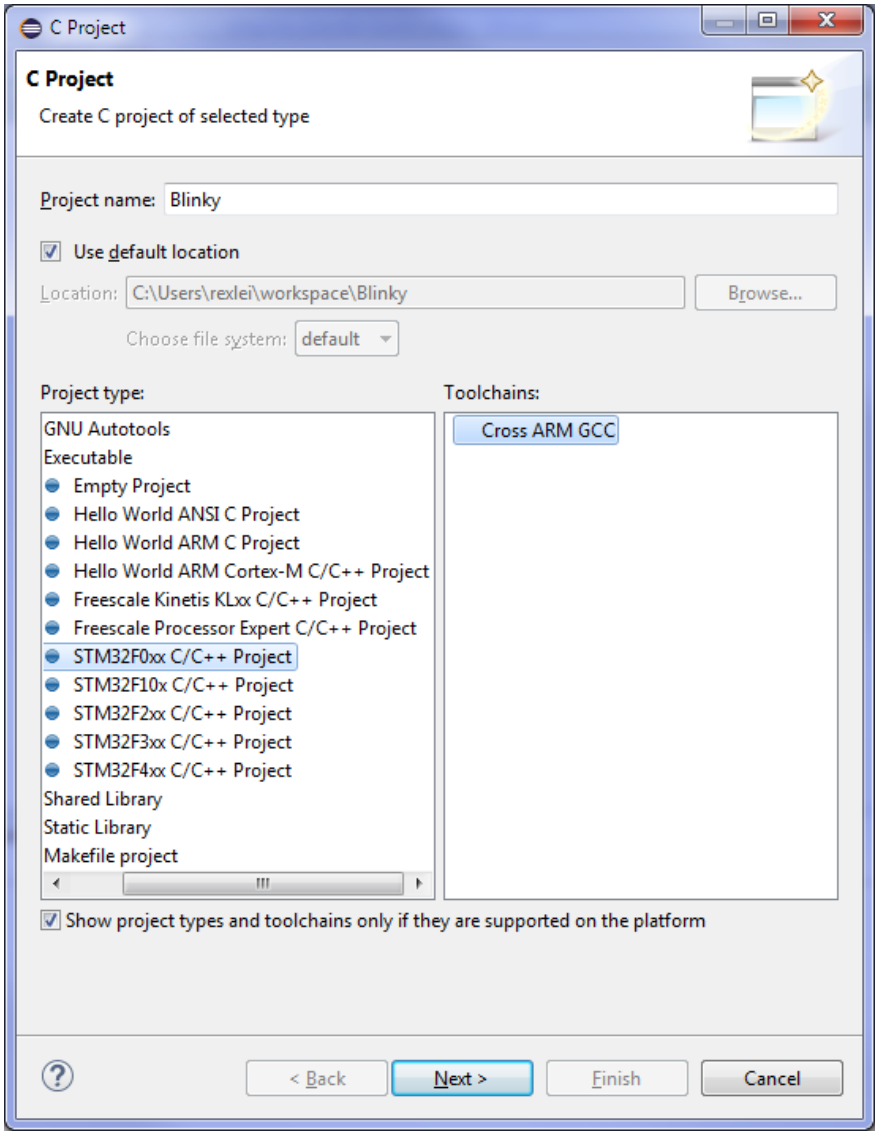

Figure 1.3: Project name and processor selection

- Content: use this default value – Blinky (blink a LED) as this tutorial is to create a Blinky project. If you want to create another new project, you can switch to Empty (add your own content)
- Click the **`Next>`** button brings you to the "Folder settings"

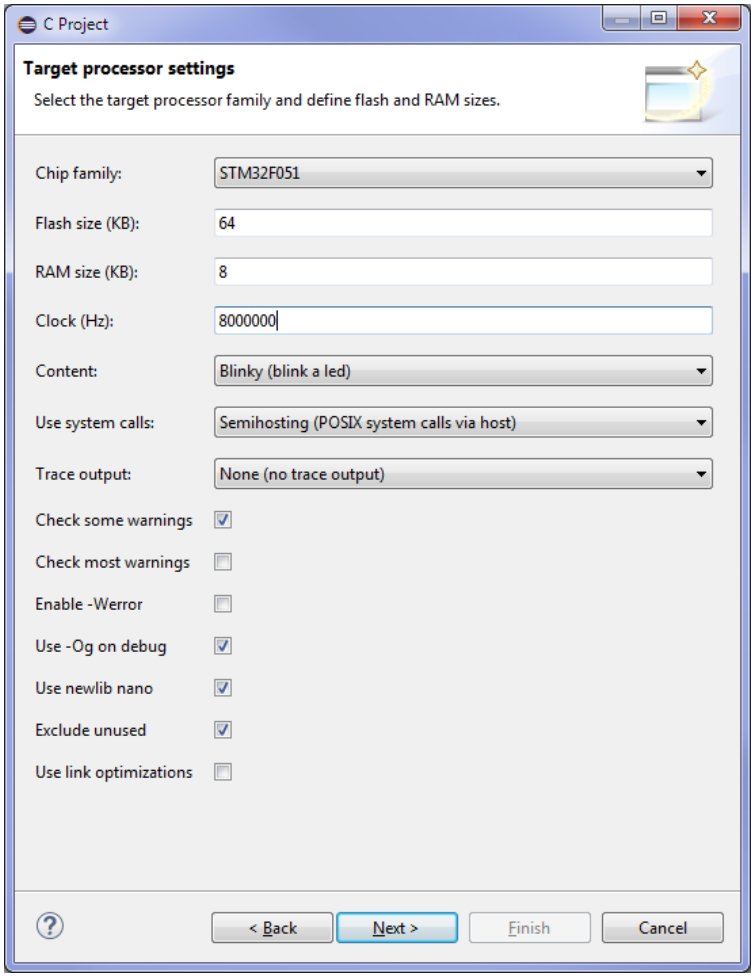

Figure 1.4: Target processor settings

In the “Folders settings” window (Figure 1.5):

- Leave the default folders unchanged and click the **`Next>`** button, to the “Select configuration” window

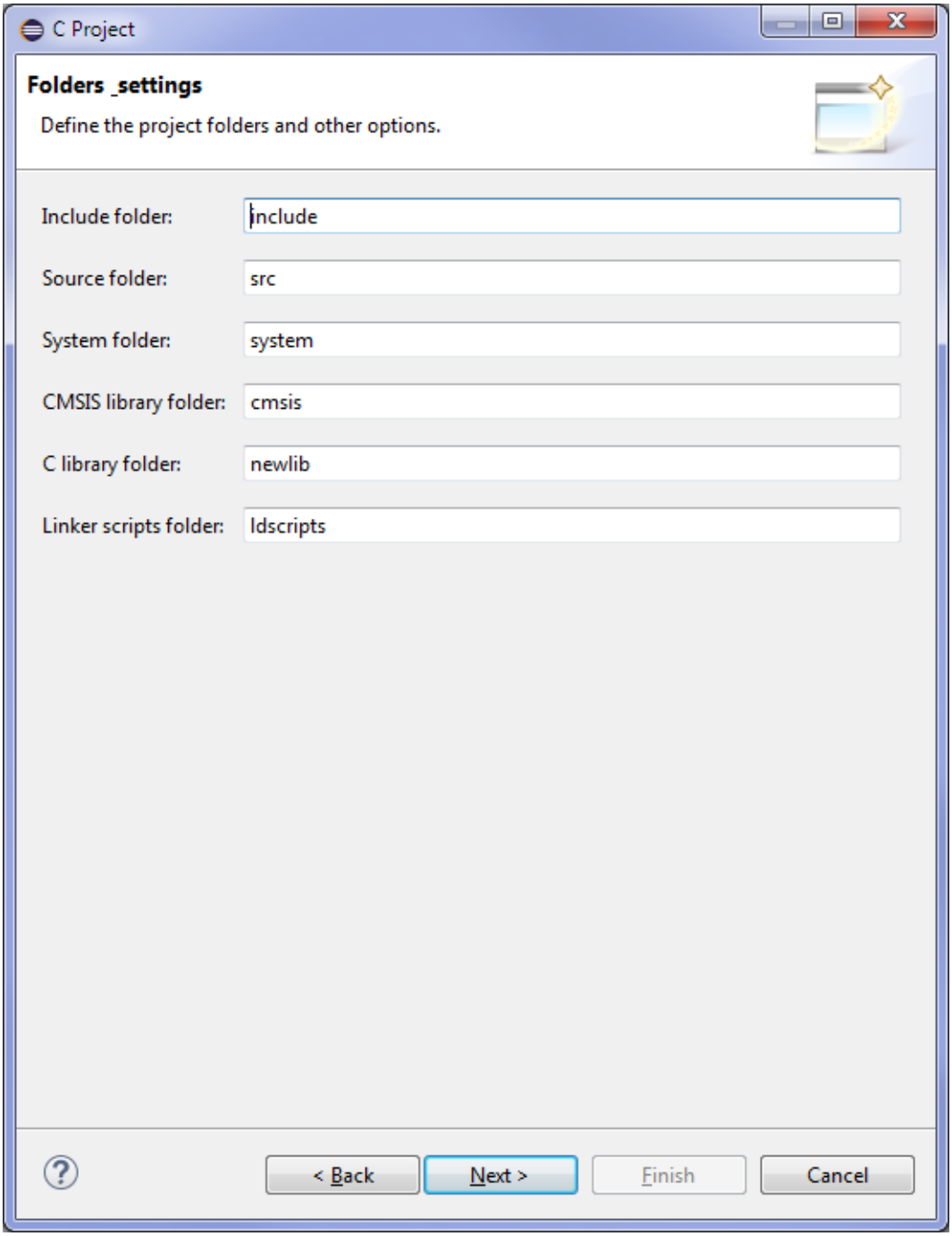

Figure 1.5: Project folder settings

In the “Select Configurations” window (Figure 1.6):

- Leave the default settings unchanged and click **`Next>`** button, to the “Cross GNU ARM Toolchain” window

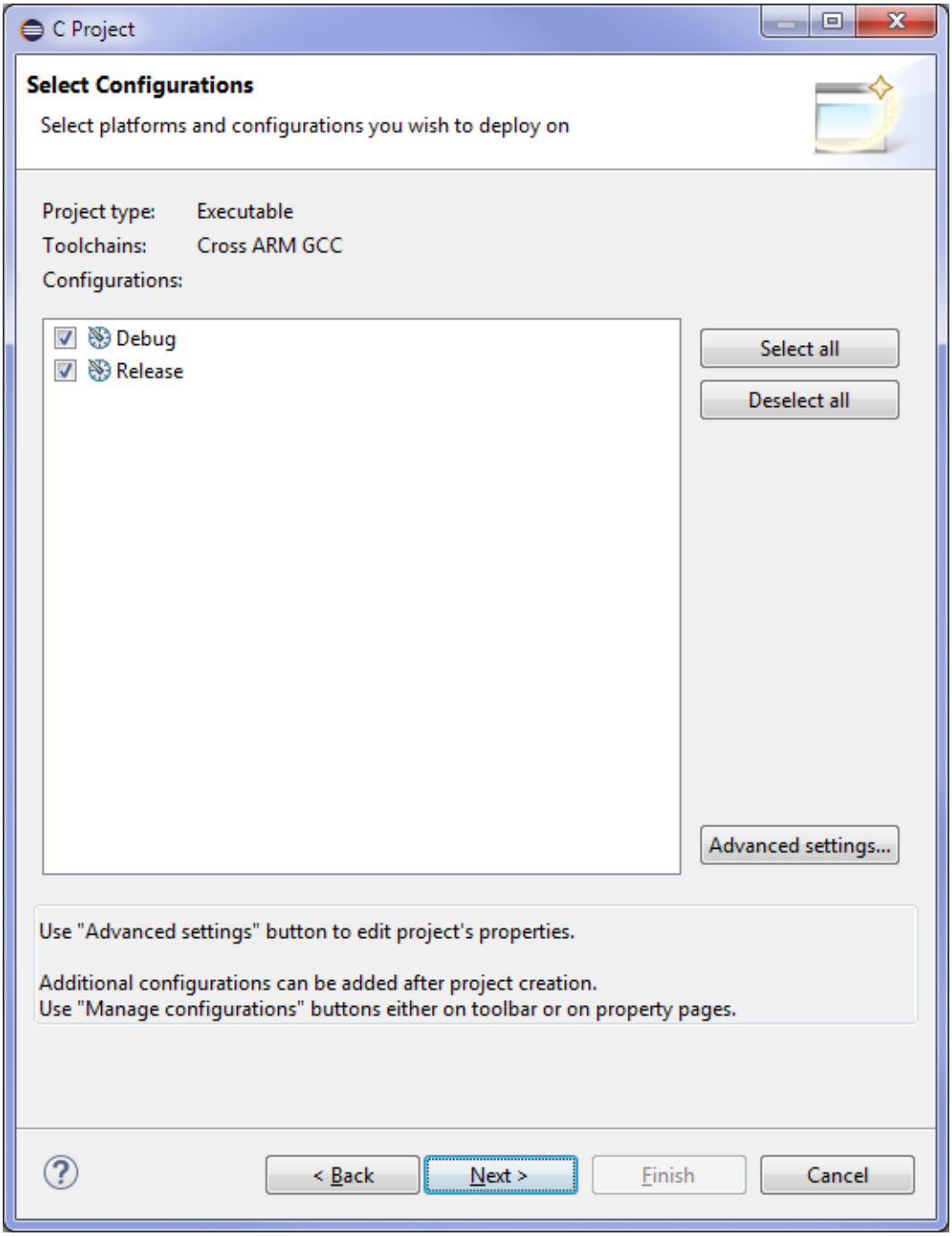

Figure 1.6: Project select configurations

In the "**Cross GNU ARM Toolchain**" window (Figure 1.7):

- Select the **Toolchain** name: **GNU Tools for ARM Embedded Processors (arm-none-eabi-gcc)**
- Click the Finish button to complete the setup process.

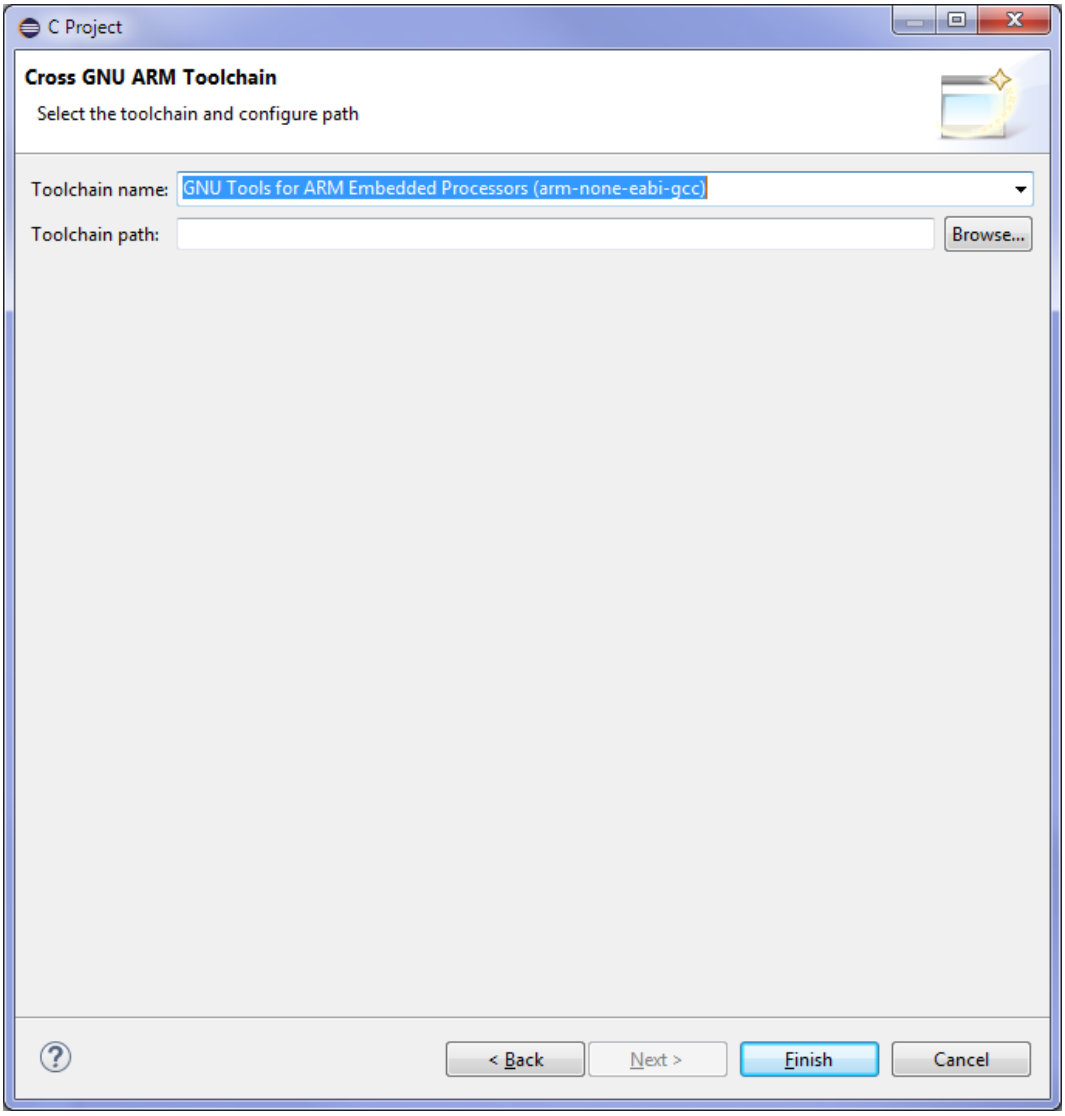

Figure 1.7: Cross GNU ARM toolchain

The result of this wizard (i.e., a software setup assistant) is a simple project, with a main() printing greetings and blinking a LED (Figure 1.8).

1.4 Build the project (convert source code to executable binaries)

- Select the Blinky in Project Explorer. If Project Explorer window is not open in the left side of the window, you can open it from *window* → *Show view* → *Project Explorer*.
- Click the hammer icon, which is the shortcut to build the selected project (Figure 1.9).

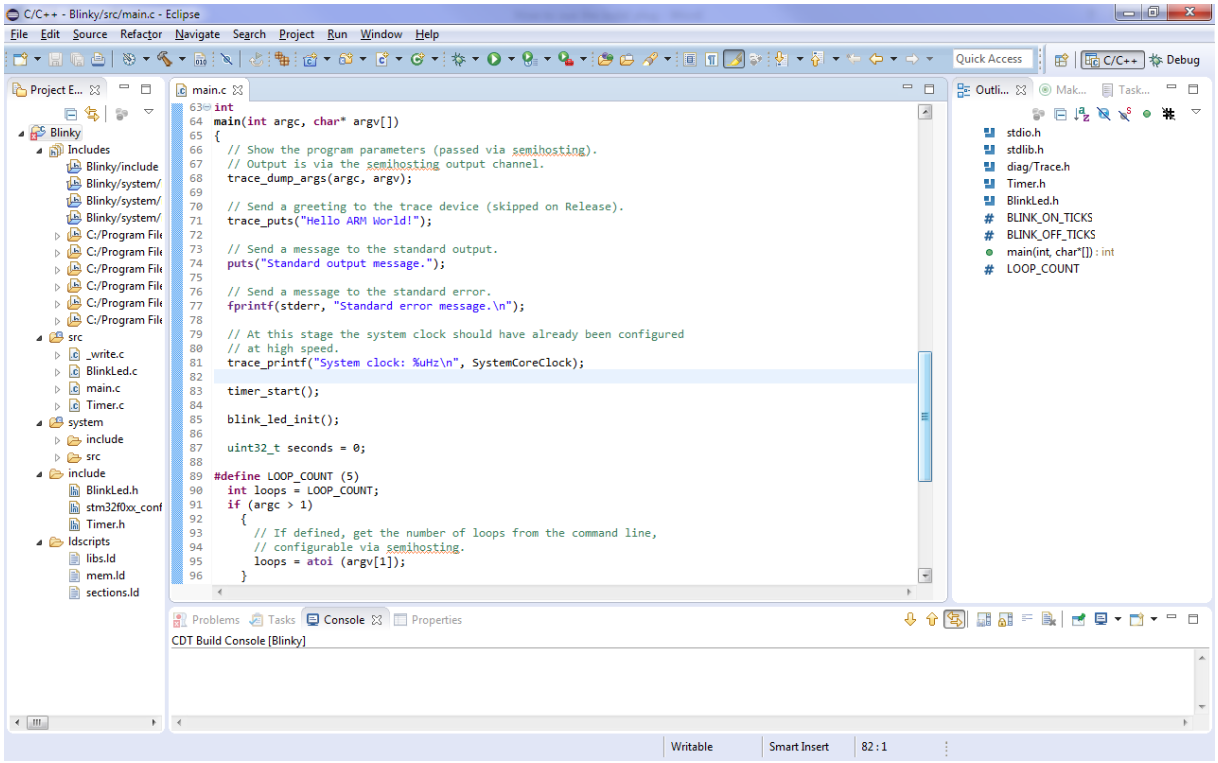

Figure 1.8: Blinking LED application

Or

- Right click Blinky in Project Explorer
- Select Build Project in the popup window (Figure 1.10)

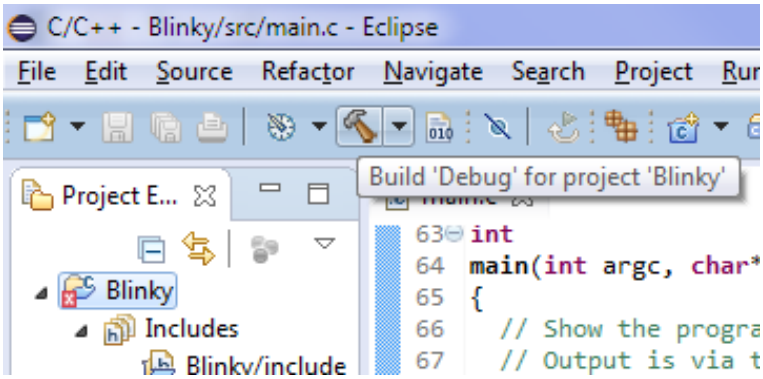

Figure 1.9: First method to build a project

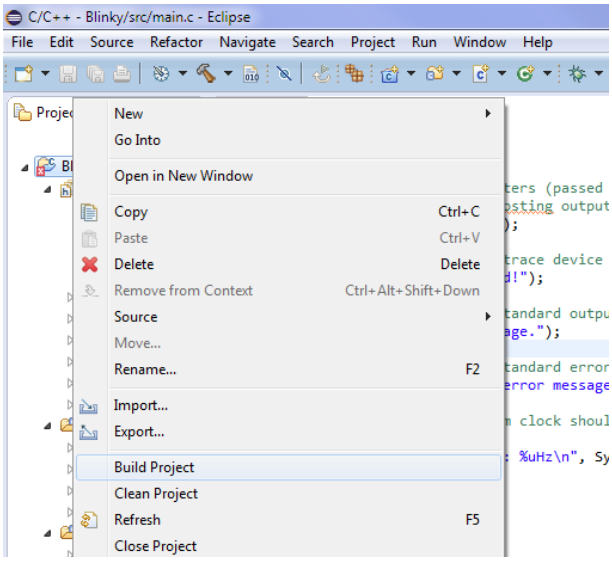

Figure 1.10: Second method to build a project

The build process produces a listing in the Console window like this (Figure 1.11) :

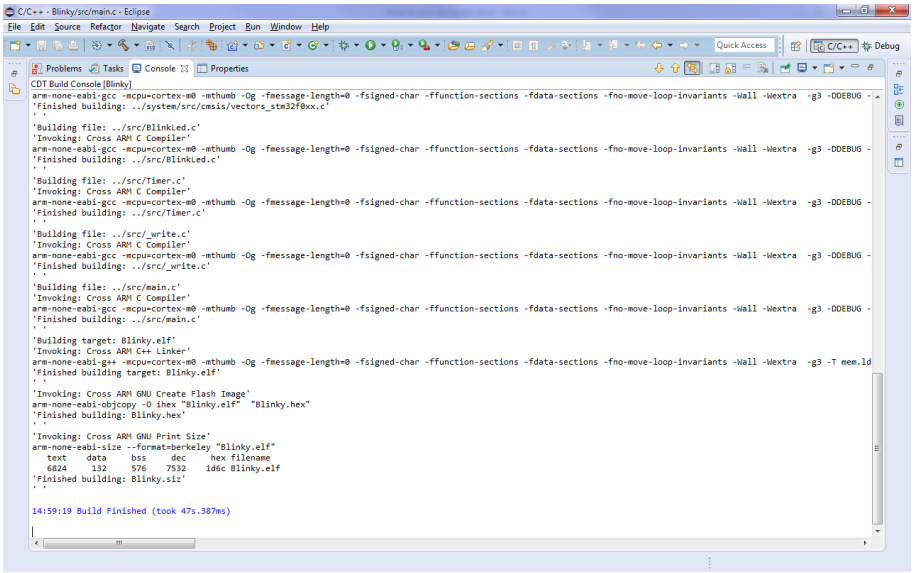

Figure 1.11: Build process listing

The files created by the build process can be found in the Debug folder (Figure 1.12):

1.5 Configure debugging

To set debugging:

- Select the forward button on the right side of the bug icon on the top menu.

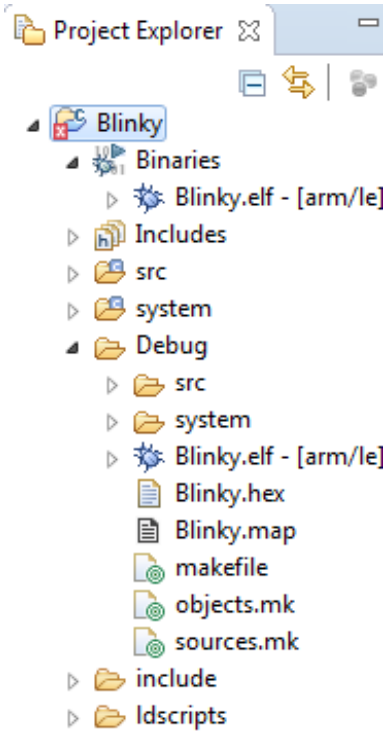

Figure 1.12: Debug folder contents

- Select **Debug Configurations...** in the popup window (Figure 1.13).

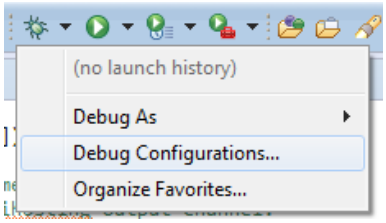

Figure 1.13: Debug menu

In the "**Debug Configurations**" window (Figure 1.14):

- Double click GDB OpenOCD Debugging; this creates a project debug with the project name Blinky Debug

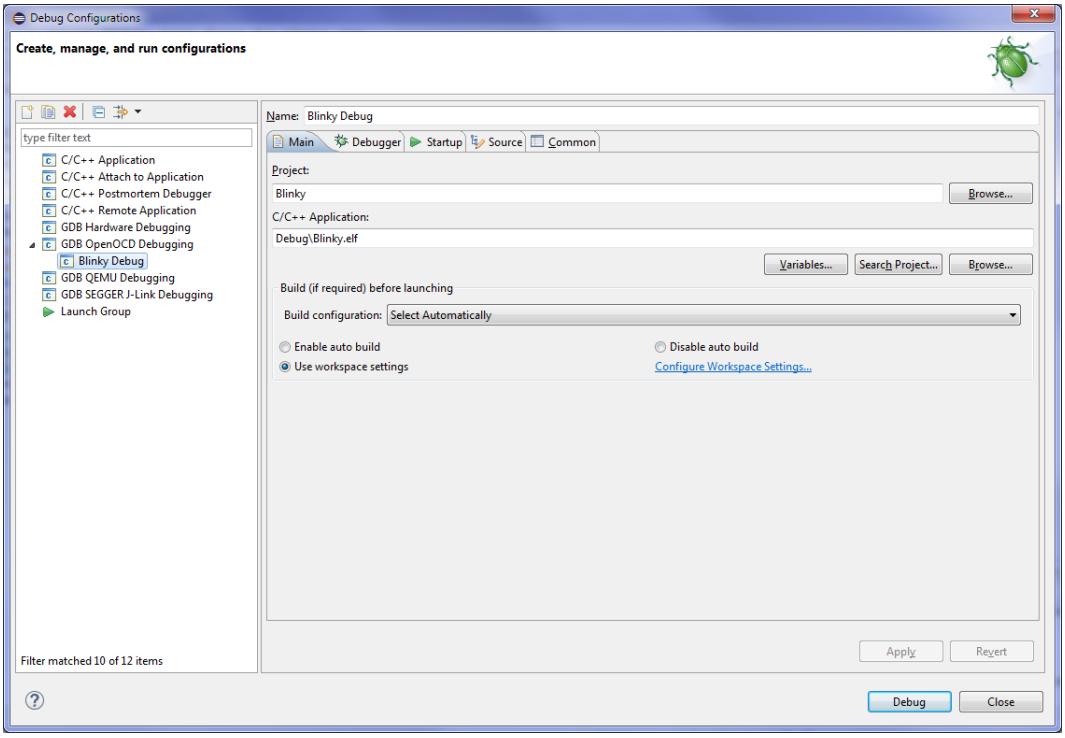

Figure 1.14: Creating an GDB OpenOCD debug configuration file.

- Go to the **Debugger** tab (Figure 1.15)
- In Executable: enter openocd.exe
- In Config options: fill in the following setting (f is a flag indicating the file to be used) :

  `-f board\stm32f0discovery.cfg`

Go to the Startup tab (Figure 1.16)

- In the rectangle space above the Enable ARM semihosting checkbox, fill in the following setting (this command tells the debugger to stop at the beginning of the main function so that the programmer can debug the project step by step).

  `monitor reset halt`

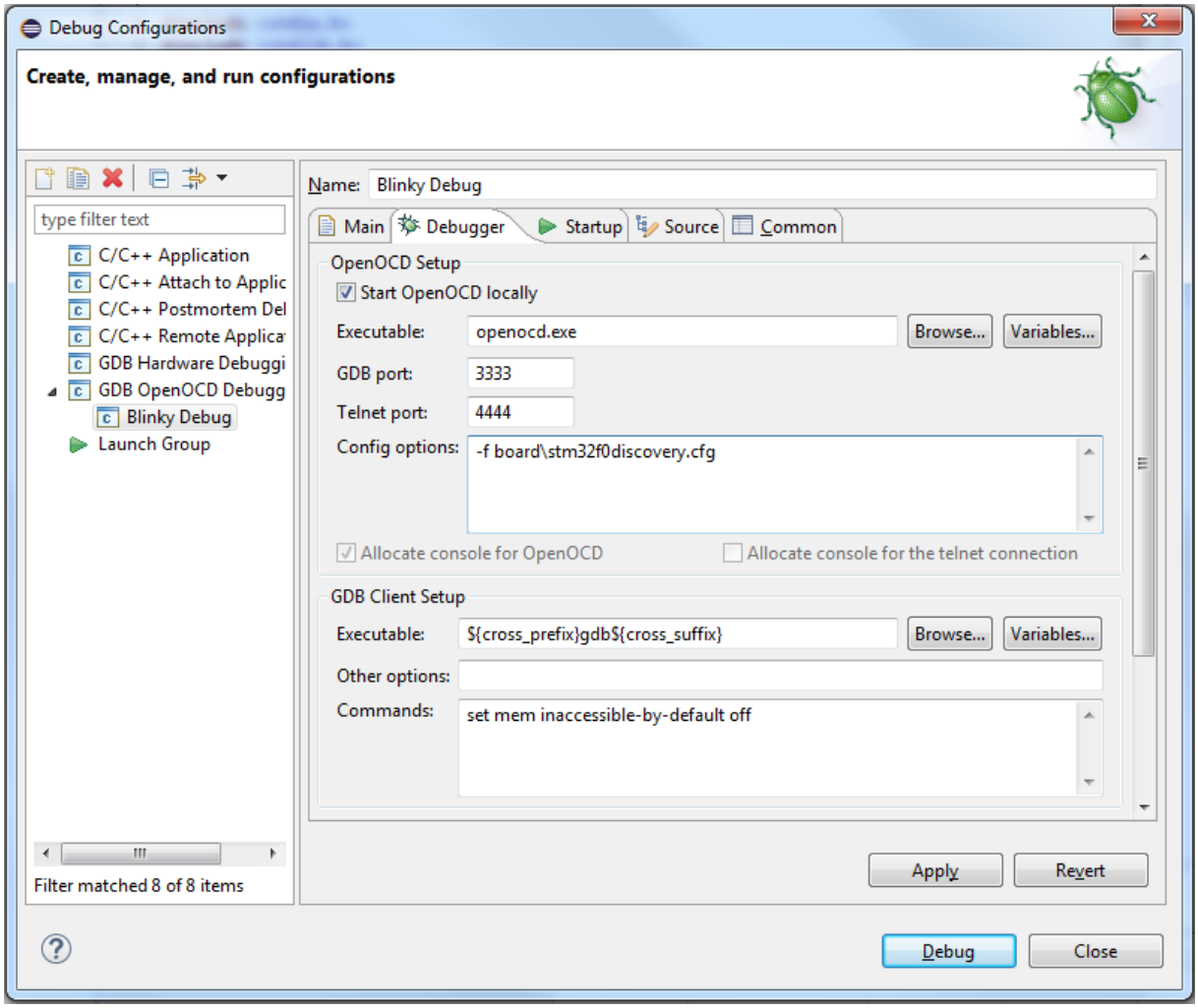

Figure 1.15: OpenOCD settings

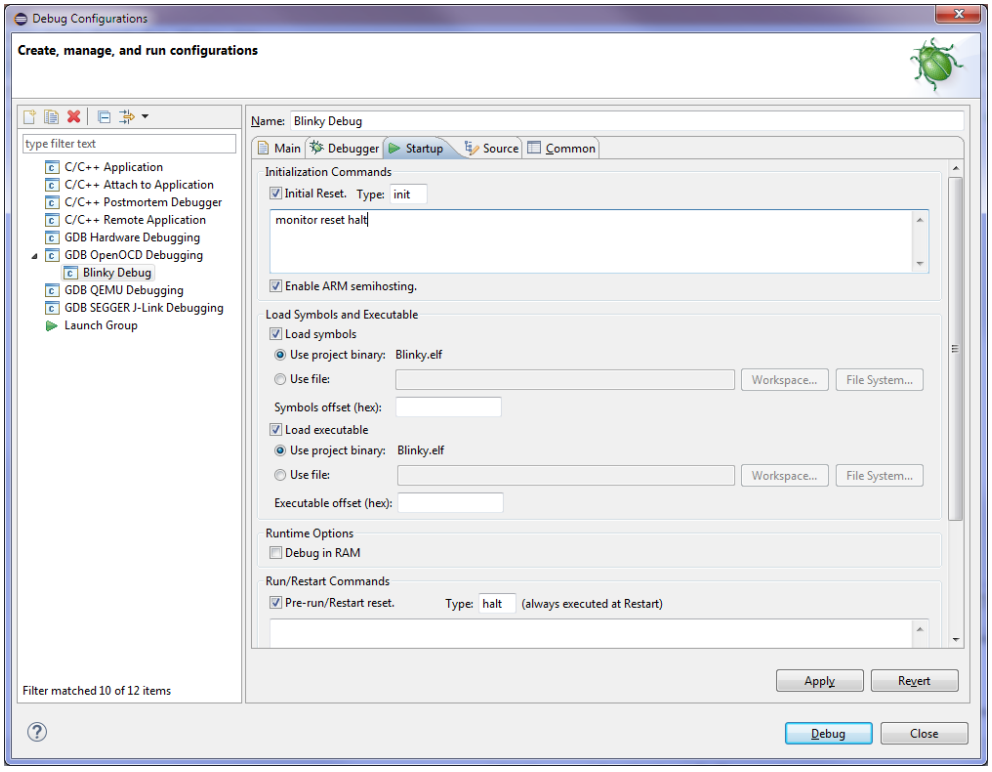

Figure 1.16: Startup tab

Go to the Common tab (Figure 1.17)

- Tick the Debug checkbox in the rectangle below Display in the favorites menu
- Click the Apply button
- Click the Debug button to start debugging

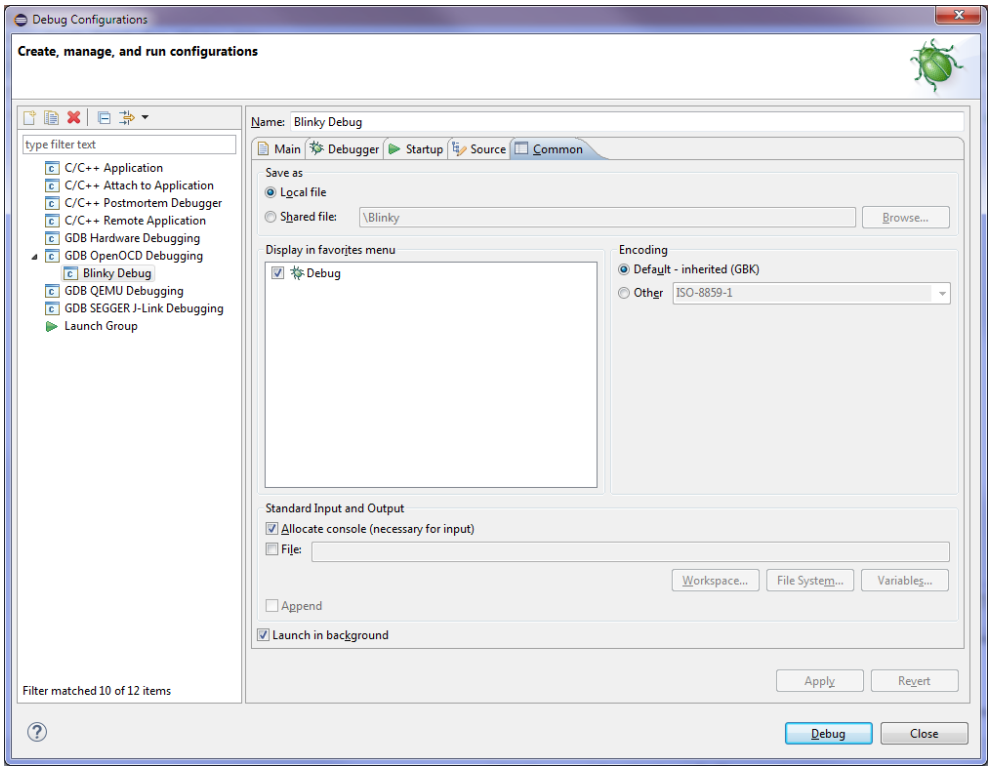

Figure 1.17: Common tab

The debugging configuration wizard creates a debug setting for the Blinky project (Figure 1.18). The result of debugging installs the binary of the project into the ARM board (that is, the executable binary file is downloaded to the RAM on board). At the beginning of the debug process, the program stops at the main function waiting for the programmer to debug the program.

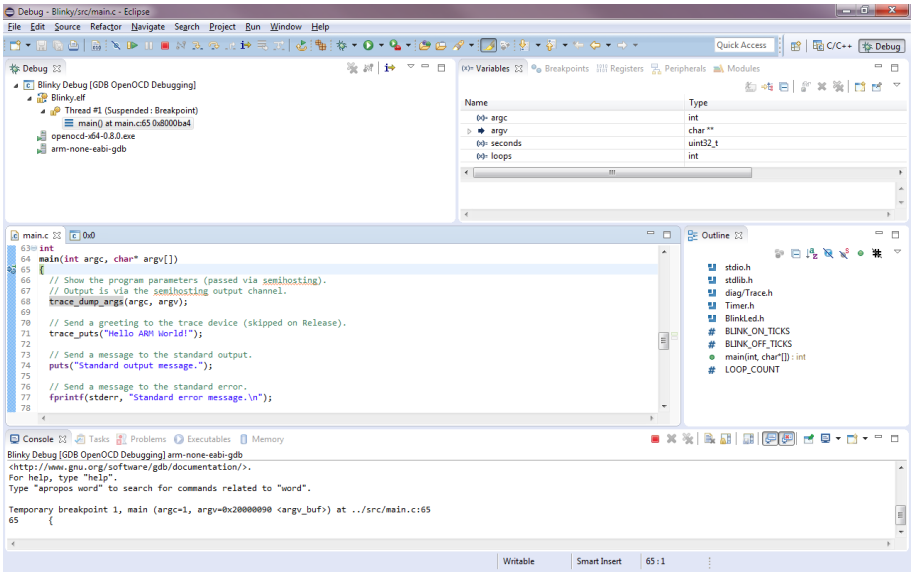

Figure 1.18: Debugging code

To debug the program, you need to run it first; go to the Eclipse menu, select **`Run ->Resume`** as shown in Figure 1.19.

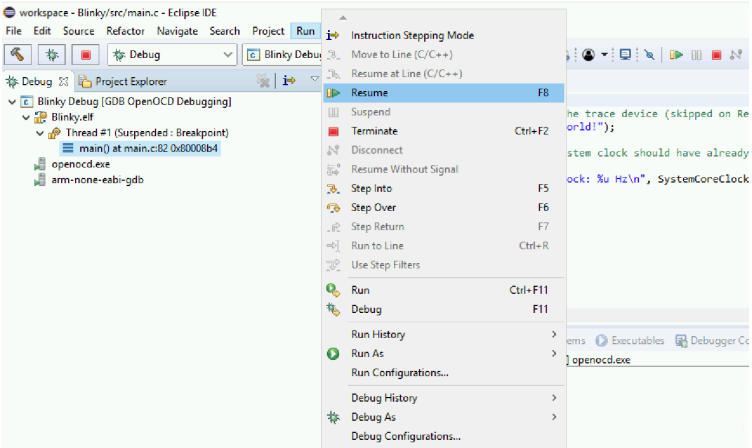

Figure 1.19: Run configuration code

You will see the LED blinking on the **stm32f0 board**. A report of seconds will be printed on screen, as shown in Figure 1.20.

These icons in the toolbar of Eclipse are the ones used most frequently in debugging and help the programmer to interact with the debugger. Experiment with these icons and see how they work.

- Icon A: to skip all the breakpoints
- Icon B: to resume the program from debugging
- Icon C: to suspend the program and set it back to debug

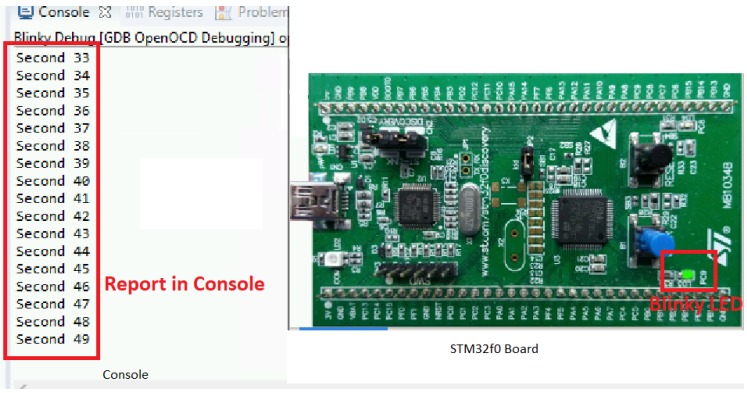

Figure 1.20: Blinking LED

- Icon D: to terminate the program
- Icon E: to step into a function
- Icon F: to step over a function
- Icon G: to step return to the function
- Icon H: to restart a process or debug target without terminating and re-launch

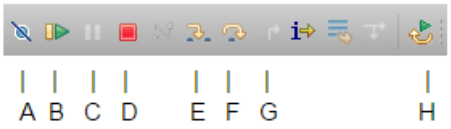

Figure 1.21: Debug icons

1.6 Part 3: Create an ARM Assembly Project

Steps in creating and running an assembly program:

- Follow the wizards in Part 2, section 1.3 of this tutorial that create an ARM C project.
- Change the **.c** extension of main.c to **.asm** or **.S** and remove all the contents in that file.
- Enter assembly codes in main.asm/main.S to implement your application.
- The **build** and **debug** steps are the same as described in Part 2.

Creating an assembly project: (using Blinky project as an example)

- Remove all the files except main.c in the **src** folder in **Blinky** project and replace the extension of **main.c** by **.asm**.

- Clear the contents in **main.asm** and write some simple assembly codes for experimentation (Figure 1.22).

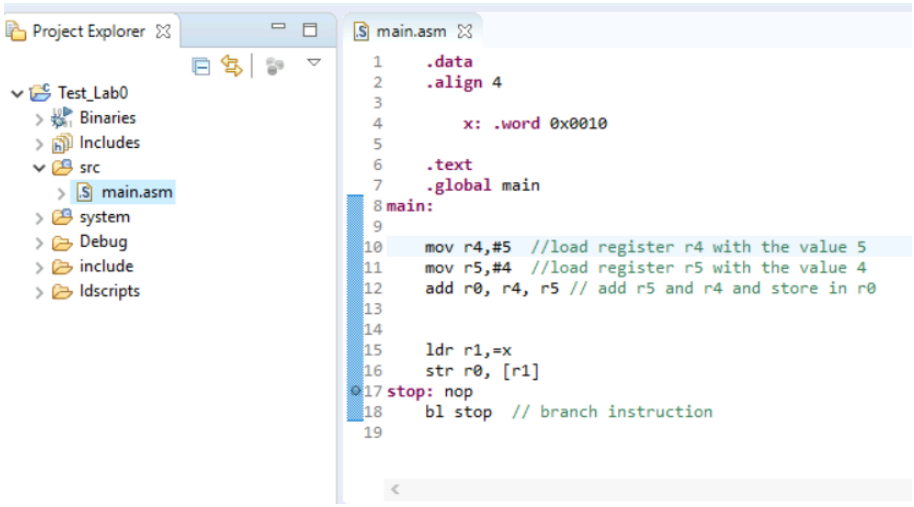

Figure 1.22: Assembly code

1.7 Part 4: Tips for using eclipse

Registers

During the debugging process you will be examining registers in the processor as shown in Figure 1.23. If the register tab is not displayed you can set it by selecting the Window option on the main menu then selecting Show `View->Registers.`

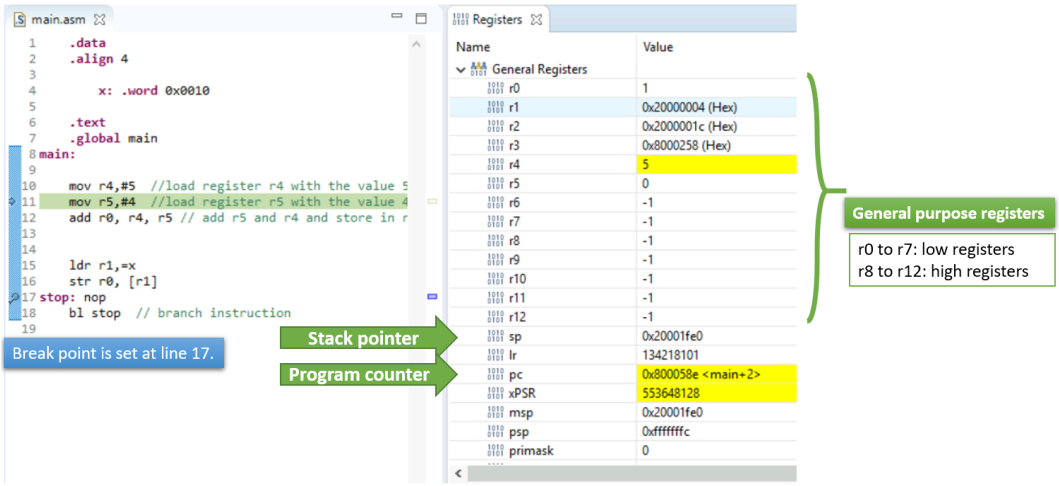

Figure 1.23: Registers

Changing displayed format in register tab:

You can change the displayed format (e.g., decimal, binary, etc.) of a register by right clicking on the specific registers and selecting the Number Format option (Figure 1.24). You can also change the format of a group of registers.

Note: Number format is also called Radix.

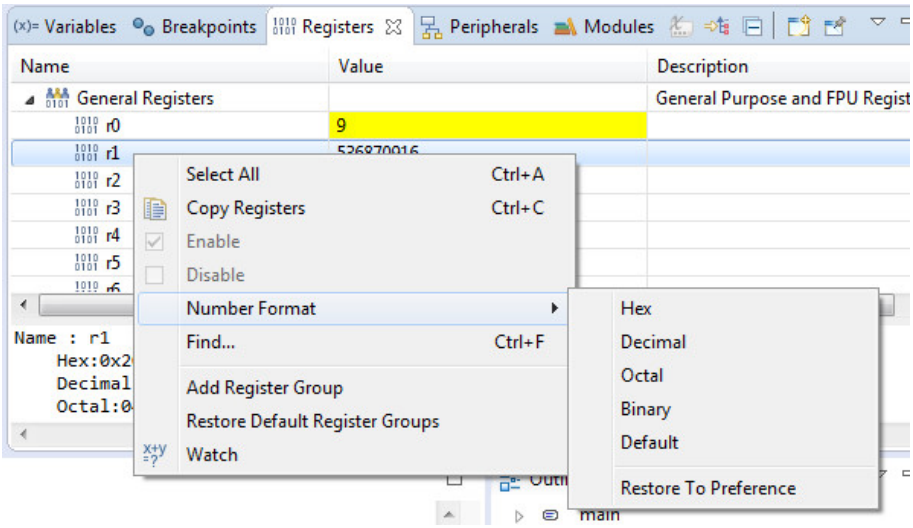

Figure 1.24: Changing number format

Memory

The memory browser allows you to examine sections of memory in you program (Figure 1.25). Check the content of x in the memory browser before and after running the code. You can display the content by selecting `Windows->Show View->Memory Browser`.

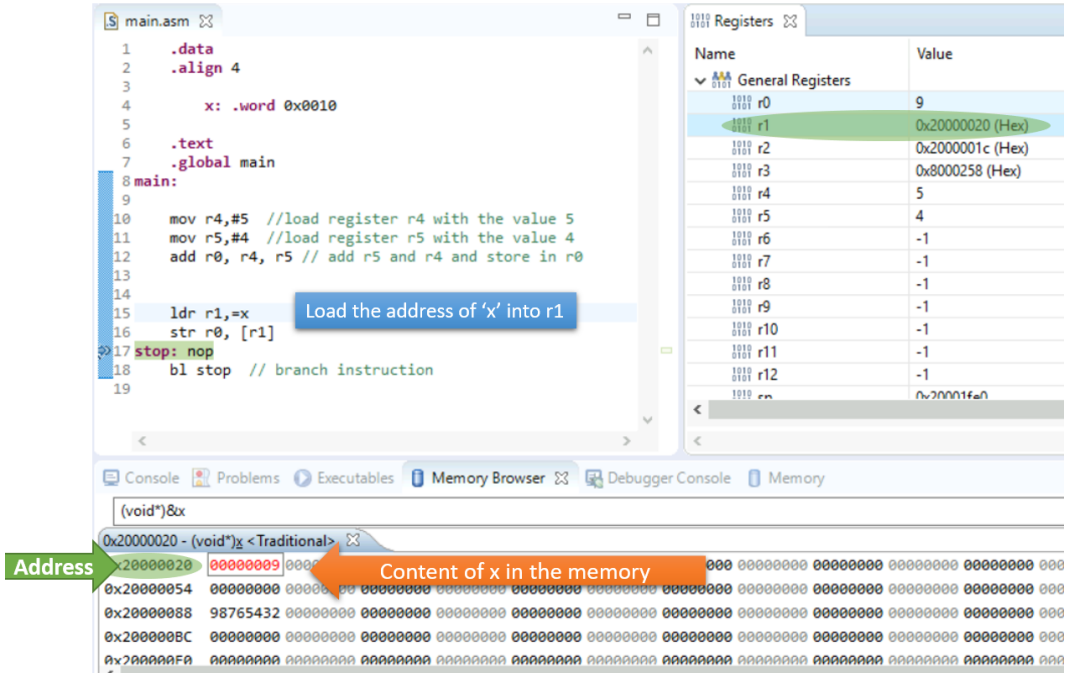

Figure 1.25: Content of memory

Changing cell size and format in the memory browser:

The cell size is the number of bytes that Eclipse will use in each cell in the Memory Browser tab. To change the cell size right click on the displayed data in the Memory Browser tab and select the Cell Size option, then select the cell size to be used. This is shown in Figure 1.26. In this figure, *table* is an example. You need to replace it by the variable name in your code.

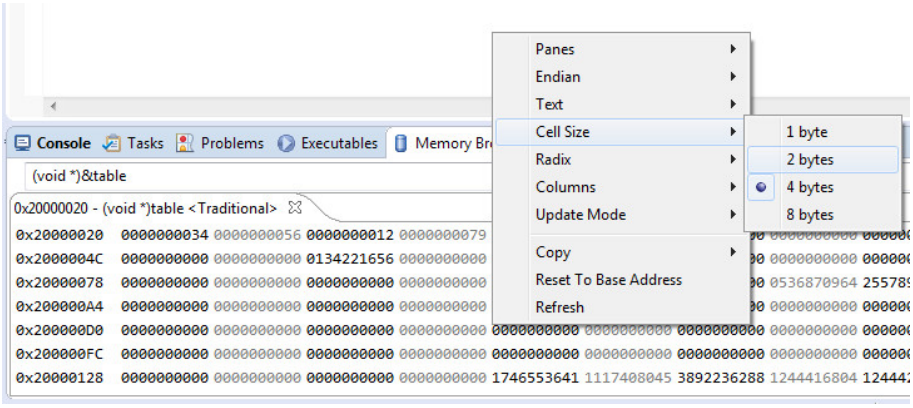

Figure 1.26: Changing memory browser cell size

To change the format of display (e.g., hex, binary, etc.) also known as Radix, right click on the displayed data and select Radix option, then select displayed format. This is shown in Figure 1.27.

Disabling console switching:

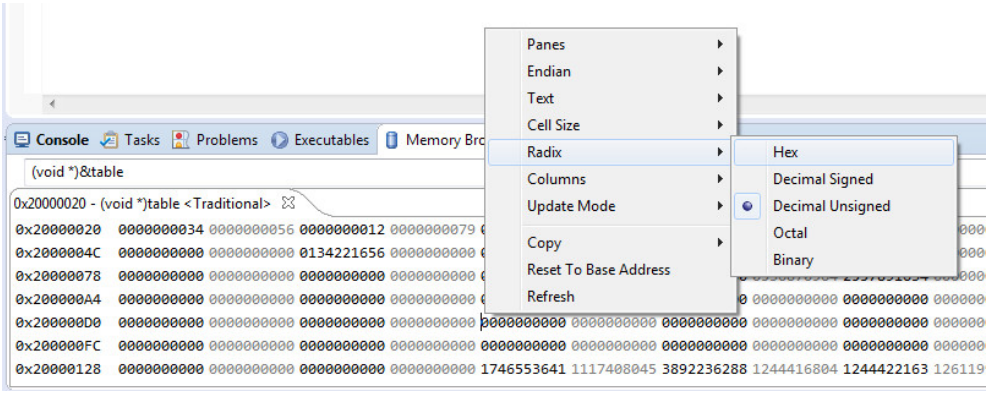

Figure 1.27: Changing memory browser displayed radix

During the debugging process when the debugger steps or encounters a break point it will print information on the console tab. When this occurs Eclipse will automatically switch to the console window to display this information. In certain situations, it would be more convenient to not have Eclipse switched to the console and just to stay in the Memory Browser tab. There are two ways to achieve this.

The first is to disable console switching. This option can be found in the *windows* → *preferences* dialog box. In the preference (Figure 1.28) dialog, expand the Run/Debug section and select console. Unselect the options "Show when program writes to standard out" and "Show when program writes to standard error" as shown in Figure 1.28. This is a global setting and will apply to all projects in the Eclipse workspace.

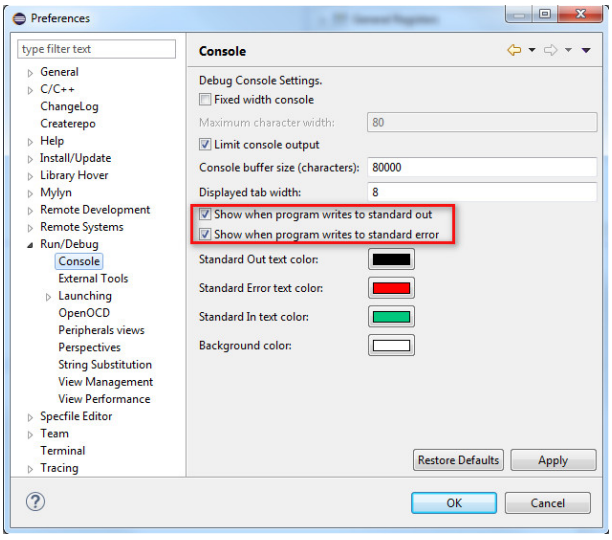

Figure 1.28: Disabling console switching

The second method is by dragging the Memory browser to a vacant area on the computer screen not being used by any program. Eclipse will automatically create a new window specifically for the memory browser. This feature can be used with any view in Eclipse.

Disassembly window

You can see the disassembled code (from human text to machine code) and the address of instructions in the disassembly window (Figure 1.29).

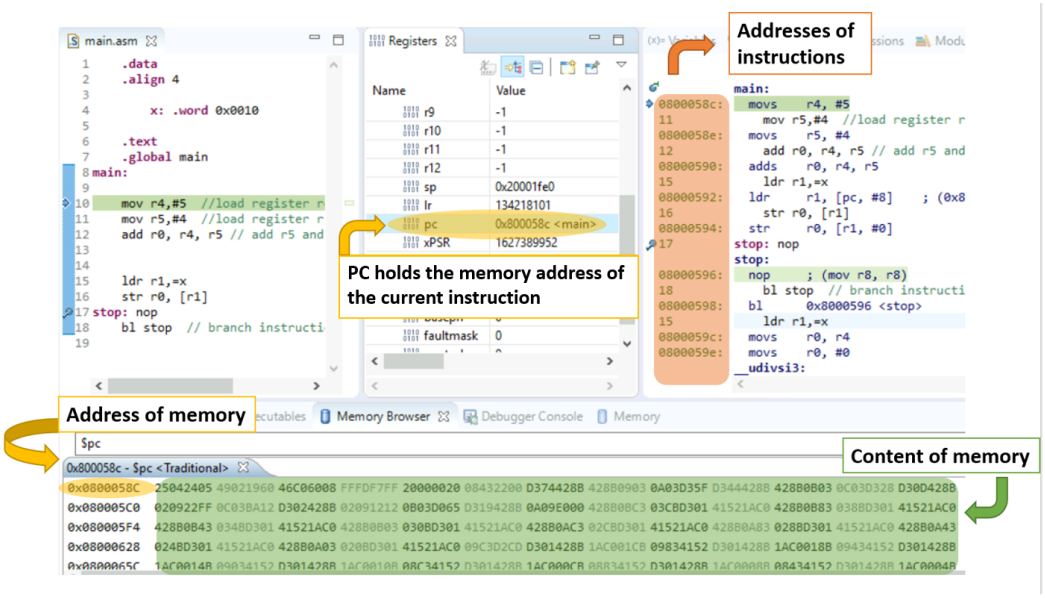

Figure 1.29: Disassembly window

Disassembly not showing in disassembly window: There is a known issue in the disassembly view. When you first enter debug mode, you will notice that the window does not update (picture on the left of Figure 1.30). The simple resolution is to close the current disassembly tab and open it from the main menu. `Windows->Show View->Disassembly` (picture on the right of Figure 1.30).

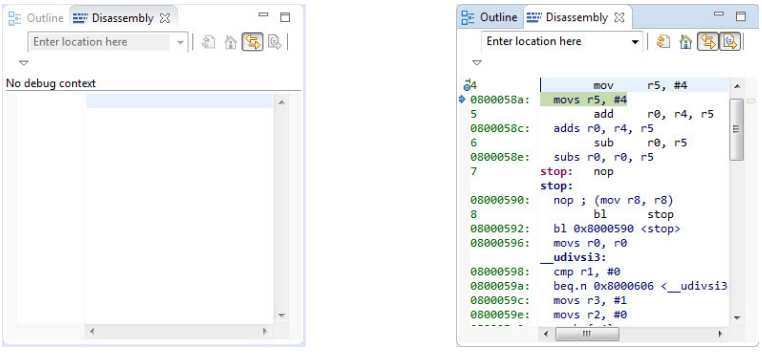

Figure 1.30: Disassembly windows