Lab 0: Getting started

ECSE 444 - Microprocessors

Fall 2018

Introduction

Welcome to the ECSE 444 Microprocessors course!

This lab will serve two purposes:

- Getting familiar with the IDE Keil μVision5, which is installed on all the TR 4160 computers and can also be downloaded from here and installed on your personal computer.
- Refreshing your ARM assembly and C programming knowledge via a few programming exercises.

Grading

Not graded!

Making a Keil μVision Project

In this section you will learn how to create a Keil μVision project. You will not necessarily have to create your projects in future labs, but it is still very useful to go through this procedure so that you get familiar with the IDE.

1. First, create a new folder named *GXX_Lab0*, where GXX is your group number. Within this folder, create three folders: *asm*, *src* and *inc*. The final folder structure is shown in Figure 1. All C code files (.c) will be saved in *src*, assembly files (.s) in *asm*, and header files (.h) in *inc*.

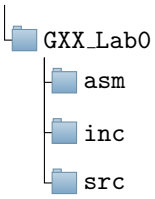

Figure 1: The project folder structure

2. Now open the Keil μVision5 software, and select *Project* $\rightarrow$ *New μVision Project...* as shown in Figure 2.

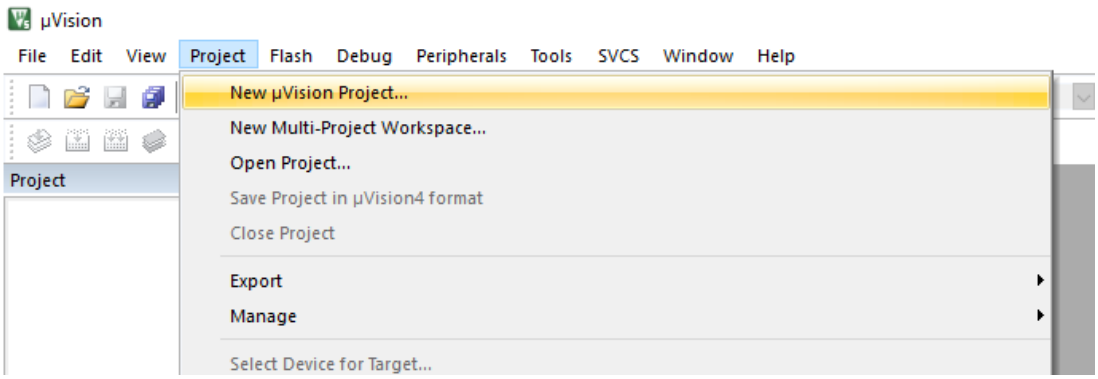

Figure 2: Create a new project.

3. In the dialogue box that pops up, navigate to the *GXX_Lab0* folder and name the project **GXX_Lab0** as shown in Figure 3.

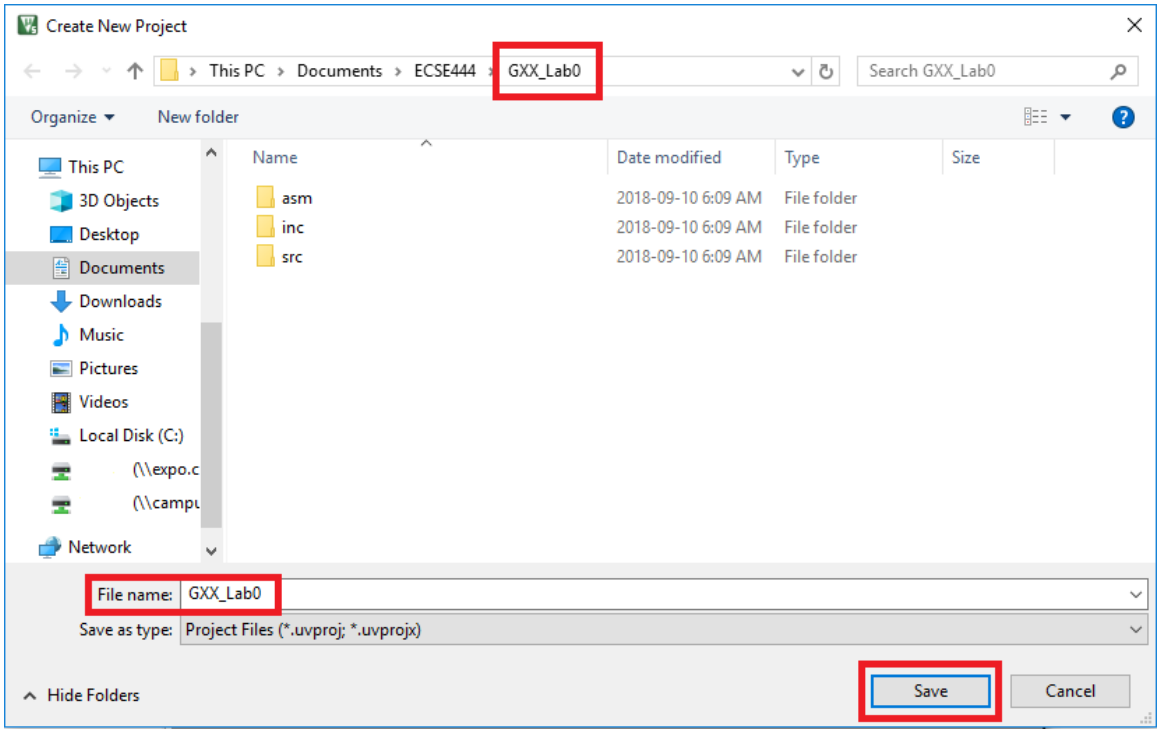

Figure 3: Name the project and save it in the *GXX_Lab0* folder.

4. We will now set the device to an ARM Cortex M4 processor with a floating point unit (FPU) as shown in Figure 4. We chose a cortex M4 here since the board that will be used in the remainder of the labs has an ARM Cortex M4 CPU.

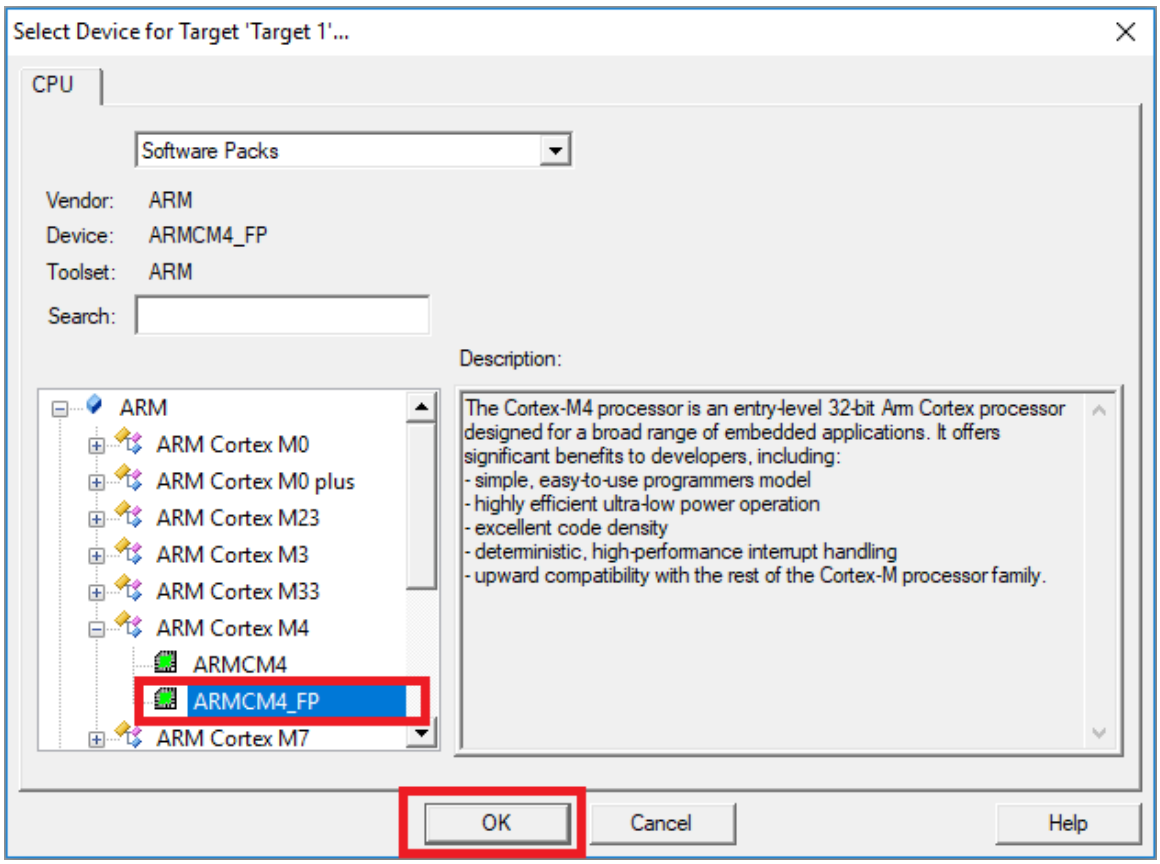

Figure 4: Choose the device.

5. We are now asked what software to add to the project. Select the options as shown in Figure 5, and click *OK*. This software is essentially a collection of low level drivers that initializes the ARM core and allows us to interact with the hardware at a higher level of abstraction.

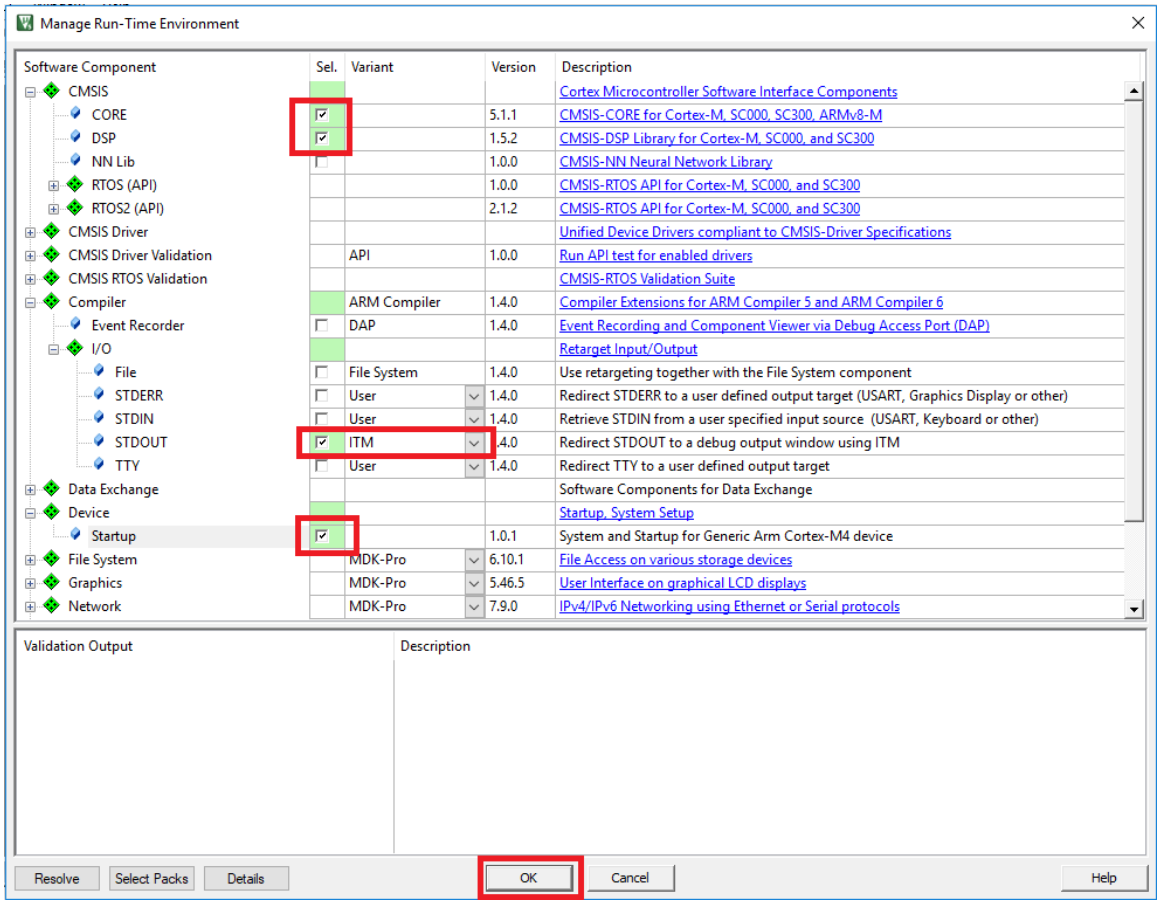

Figure 5: Select software.

6. The project is now created as shown in Figure 6. We have the option of adding more groups of files to our project as shown in Figure 7. Add one more group to the project. This is useful when we want to manage our C and assembly files in different groups, or even if we want to manage C files written for different tasks separately.

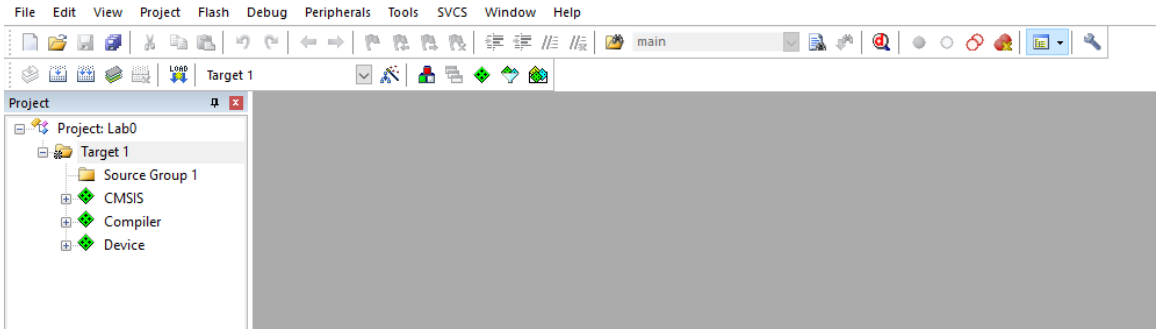

Figure 6: The project is created!

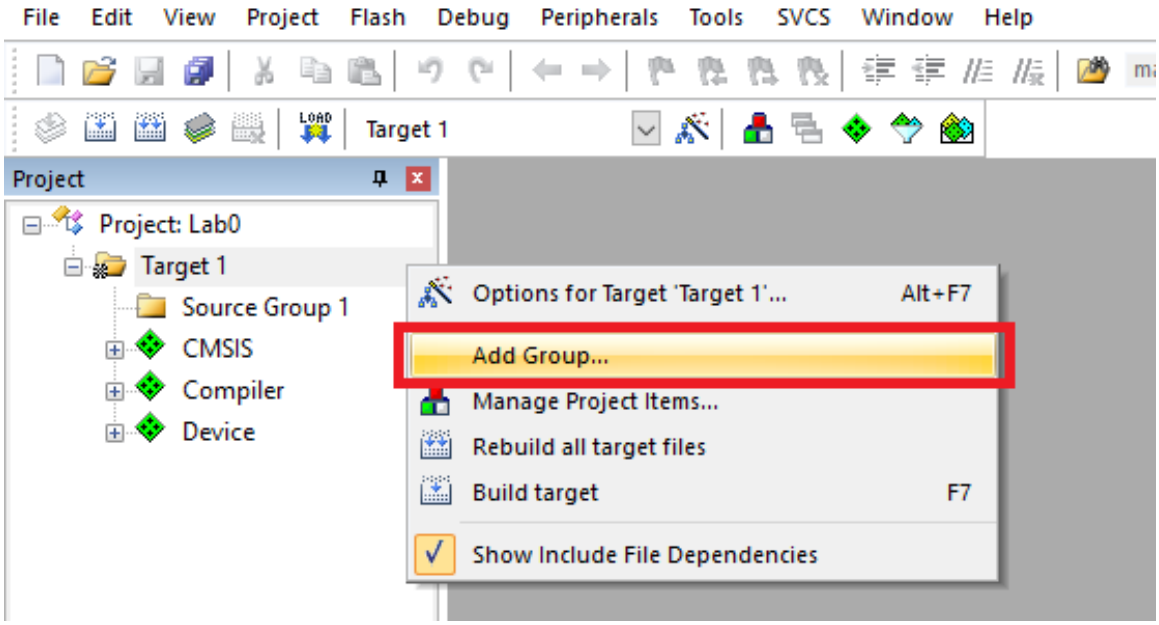

Figure 7: Add a new group.

7. We also have the option of renaming *Target 1*, *Source Group 1* and *New Group* as shown in Figure 8 (by *s l o w l y* left clicking on the name twice). Let's go ahead and rename them.

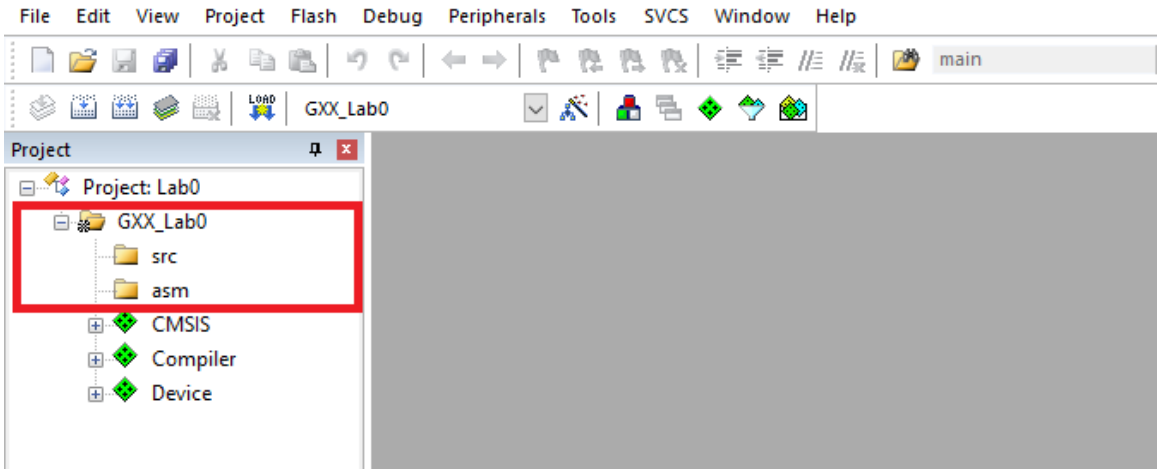

Figure 8: Rename the groups and the target.

8. Lastly, we have to change some project options. The menu can be accessed as shown in Figure 9. We need to tell the compiler in which directories header files will be found. Fill in the field as shown in Figure 10 (the text says " .*inc; .\RTE* "). We also need to run the project in simulation mode since we don't actually have the boards as yet. This is done as shown in Figure 11.

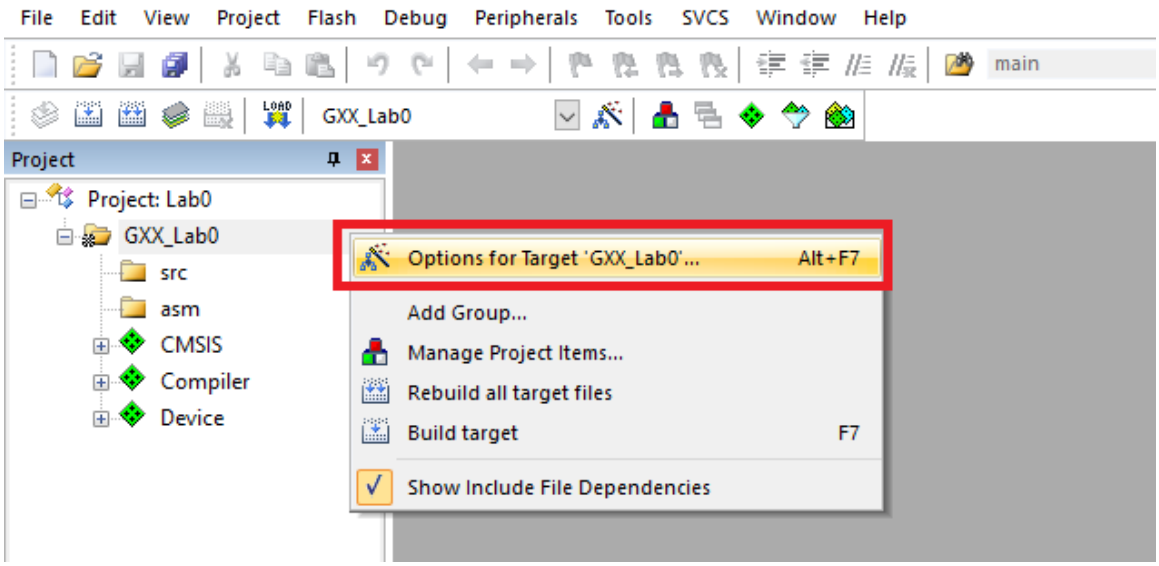

Figure 9: Accessing the project options.

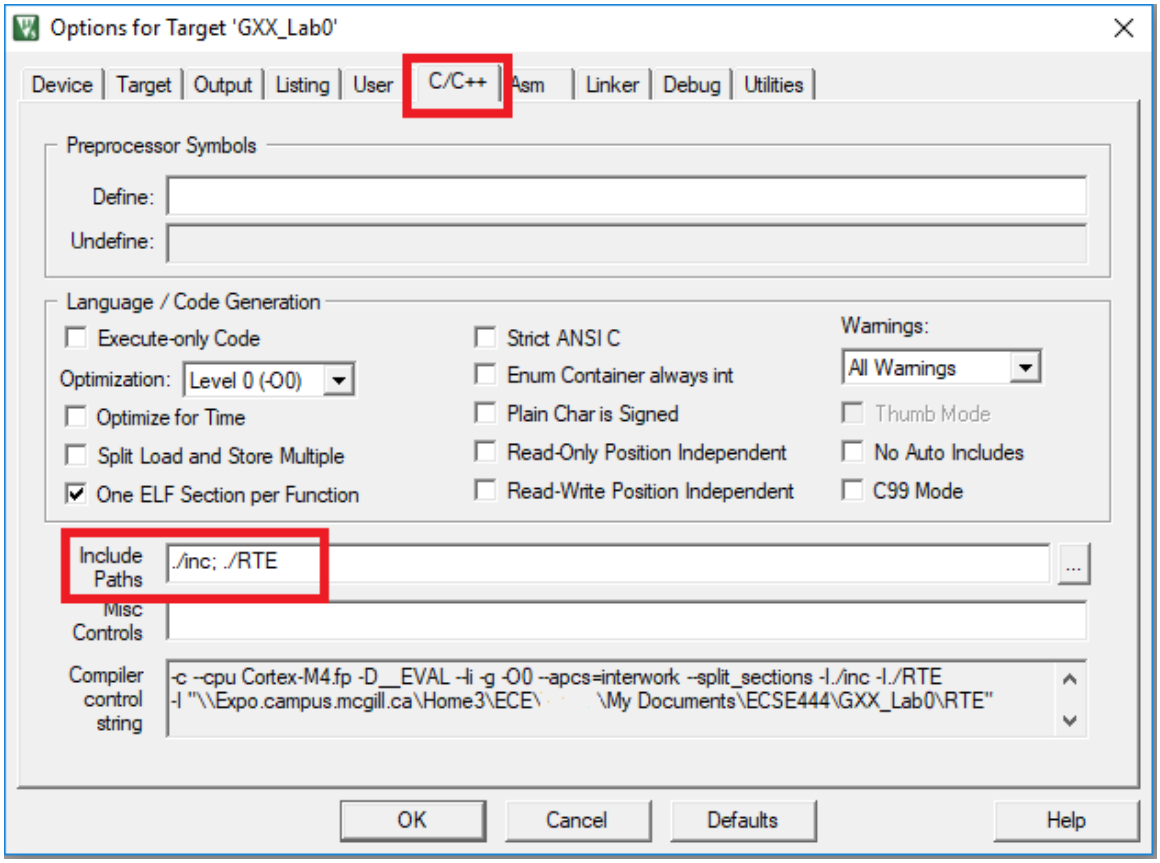

Figure 10: Set the include directories

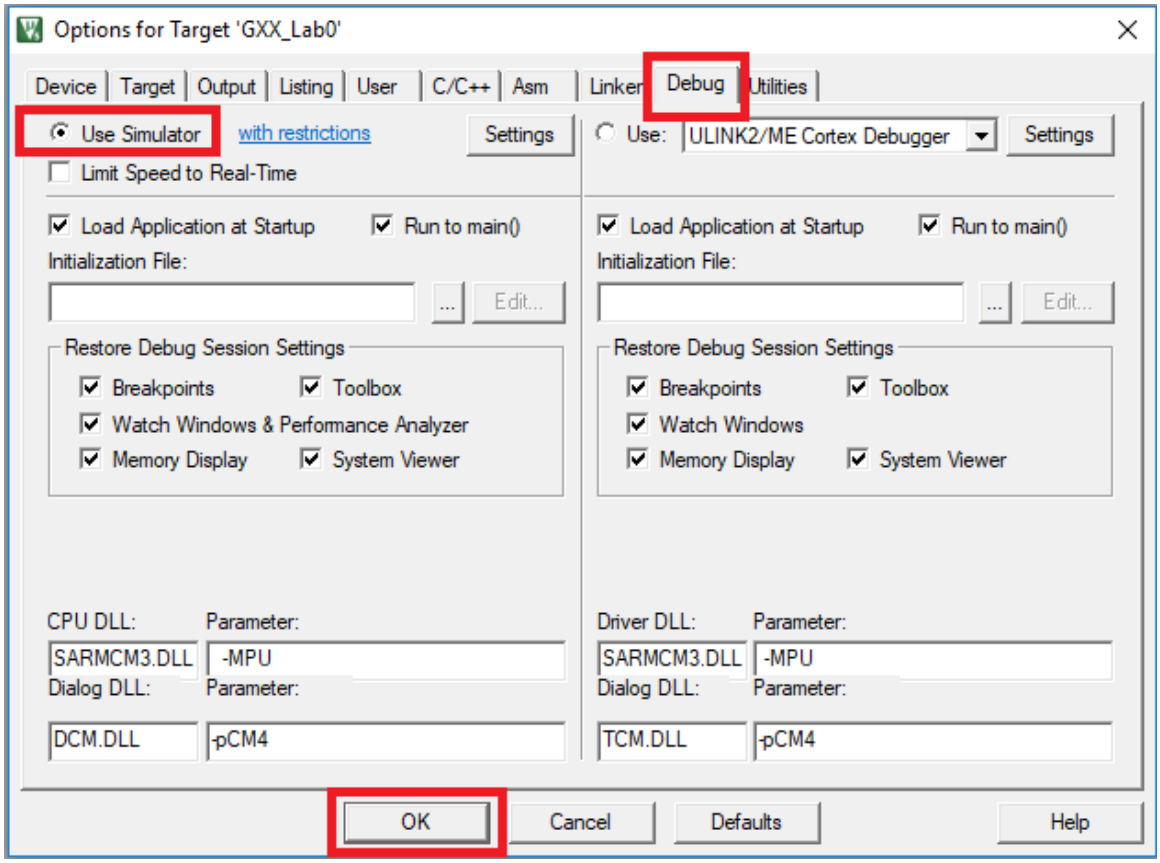

Figure 11: Enable simulator mode.

That completes the project build! Do not lose heart if you have been unsuccessful in this endeavour; a blank project is available for you to download from myCourses, and you can always come back to this handout and try again later.

Hello World

Now we will walk through writing our first program - a simple “Hello world” C program. We will walk through how to build and execute the program in debug mode.

1. Add a new file **main.c** to the *src* group as shown in Figure 12 and Figure 13.

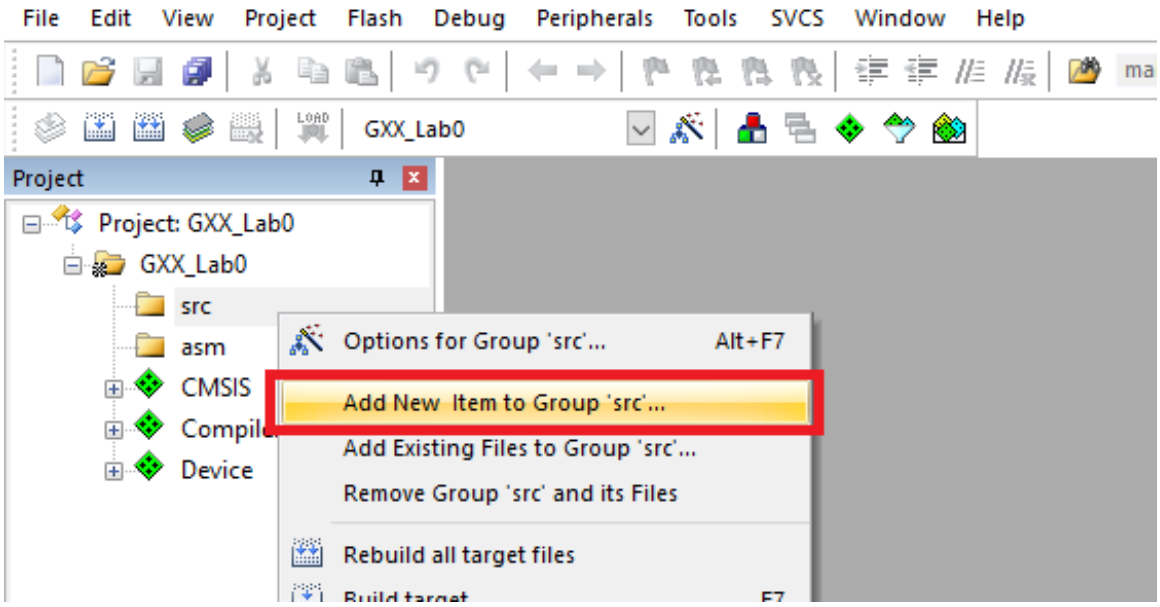

Figure 12: Adding a file to a group.

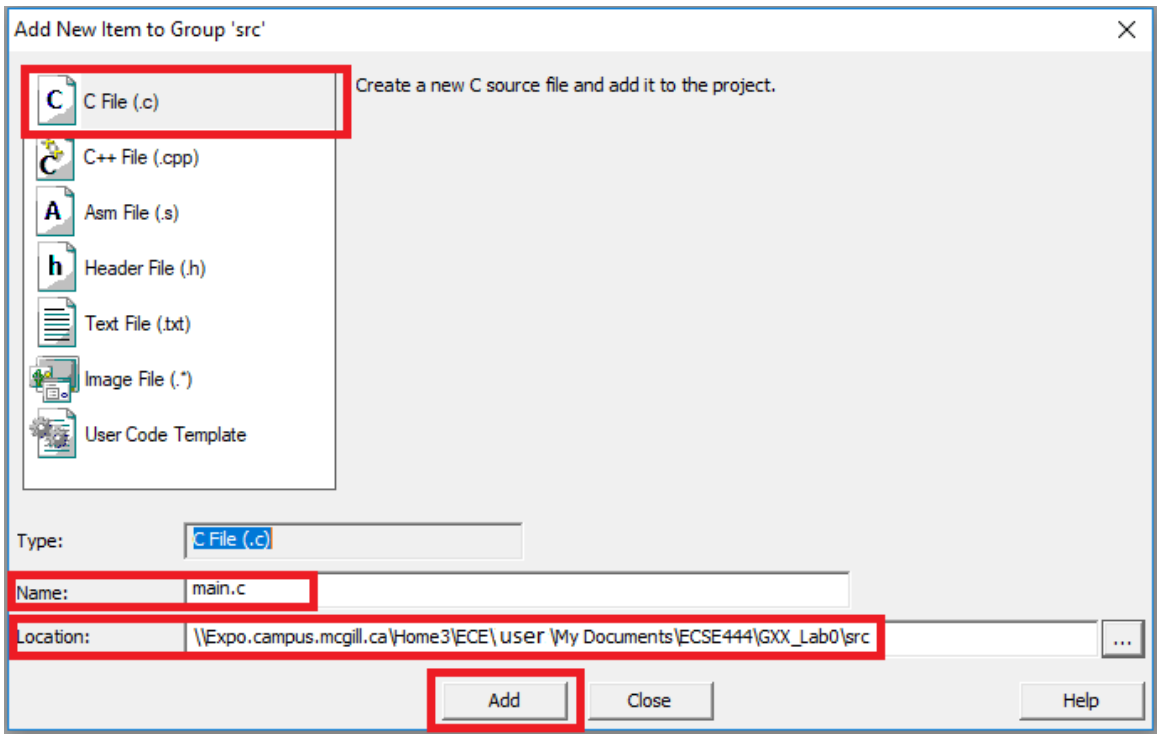

Figure 13: Adding the **main.c** file.

2. Write the “Hello world” program as shown in Figure 14 and save the file.

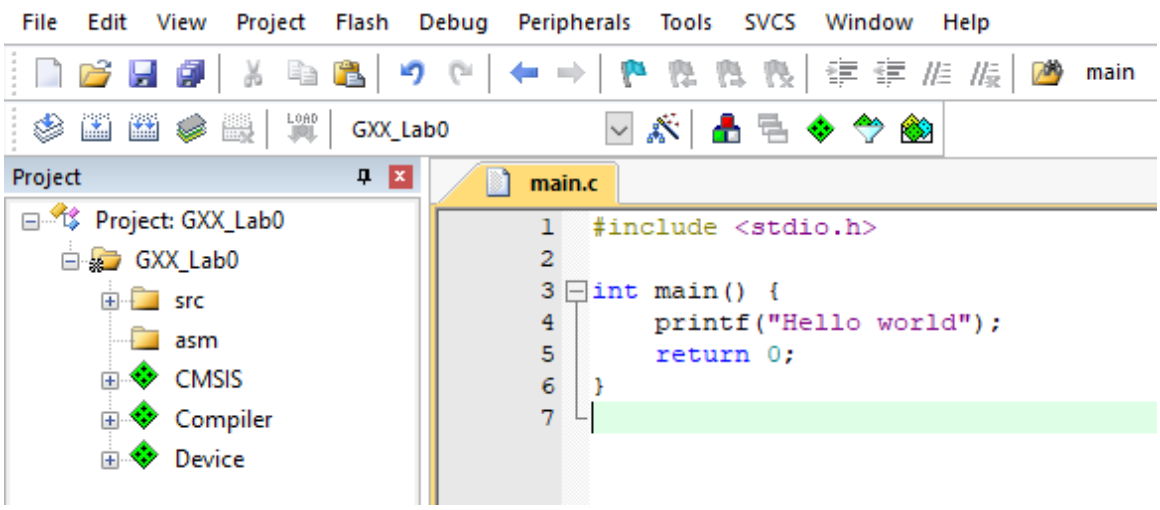

Figure 14: “Hello world” program code.

3. We will now build the project using the 'build' button or the keyboard shortcut F7, as shown in Figure 15. If everything goes well, the build output window will match what is seen in the figure.

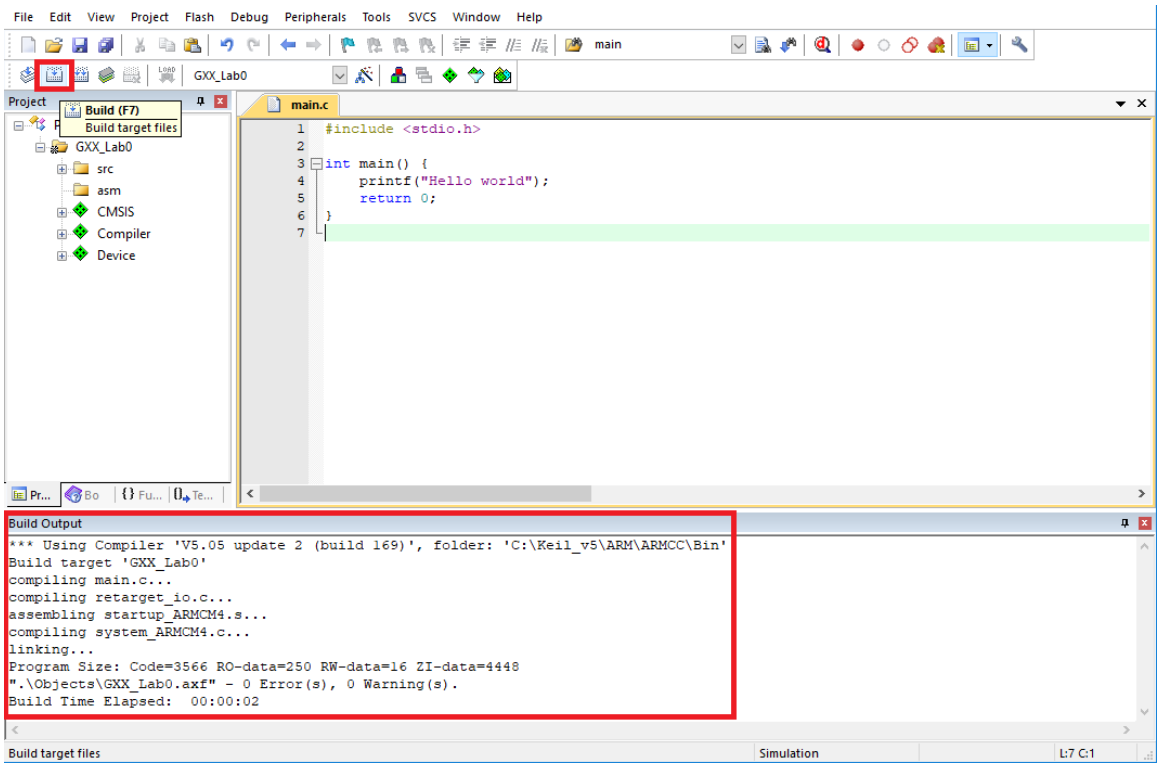

Figure 15: Building the project.

4. We will now enter the *Debug* mode via the 'debug' button or the keyboard shortcut ctrl+F5 as shown in Figure 16.

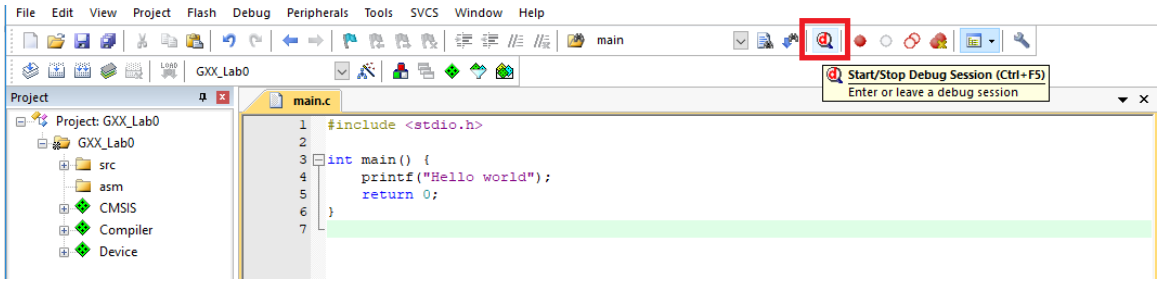

Figure 16: Entering the debug mode.

5. Once in the debug mode, you will be able to see the processor registers, the disassembly window, and other useful windows to debug your applications. We can open the printf viewer as shown in Figure 17.

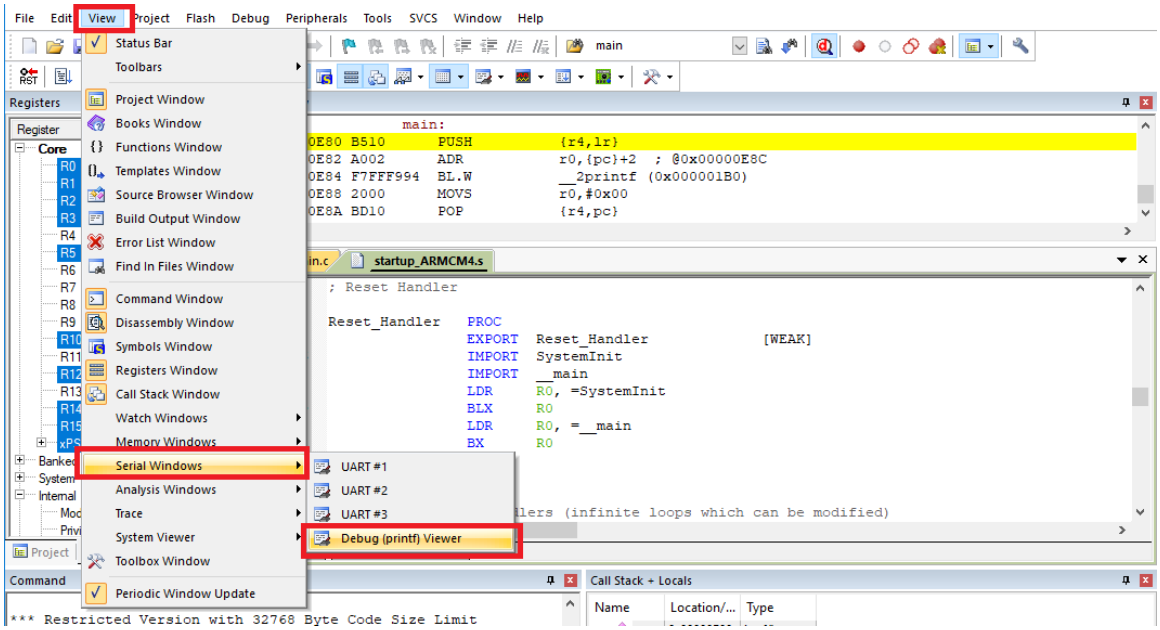

Figure 17: Opening the printf viewer.

6. Finally, execute your code by clicking the 'run' button or using the keyboard shortcut F5 as shown in Figure 18. You should see "Hello world" printed in the printf viewer. We can exit debug mode by clicking on the debug button.

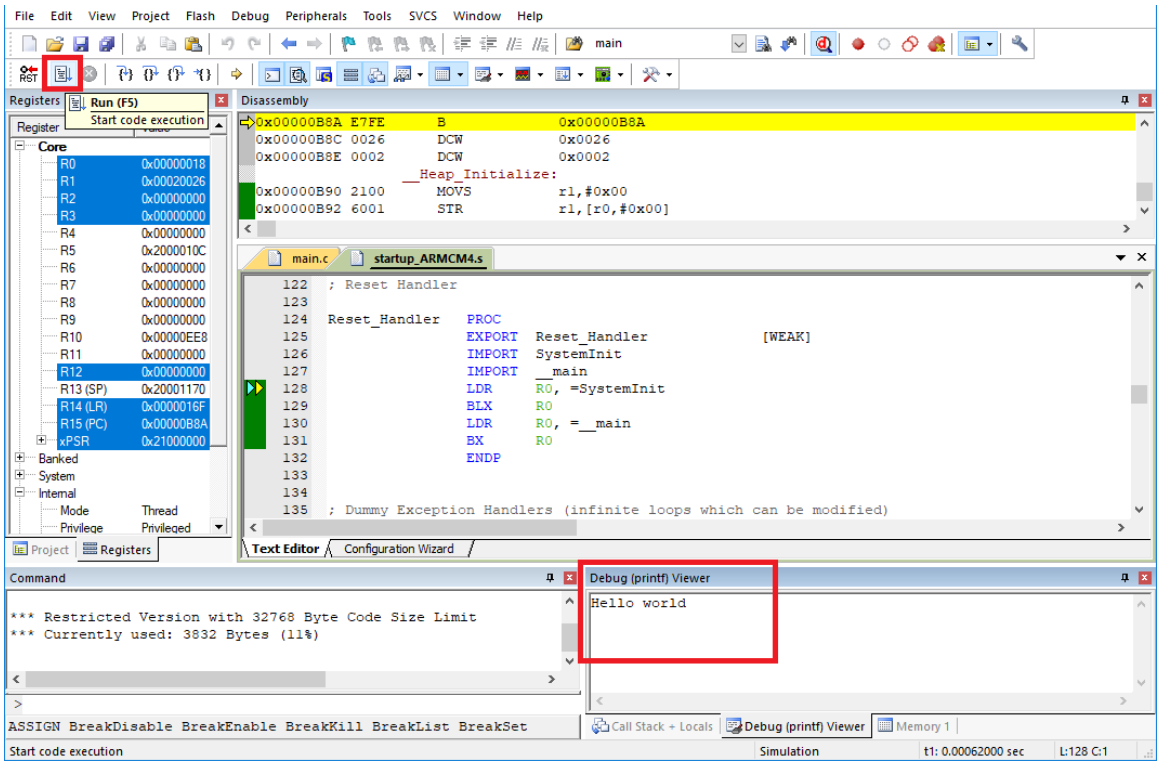

Figure 18: Executing the code.

Mixing C and assembly

We will now write a simple program to add two floating point numbers and print the result. We will write the addition subroutine in both C and assembly.

1. Create a new file **add_asm.s** in the *asm* group as shown in Figure 19 and Figure 20. Write the code shown in Figure 21 and save the file.

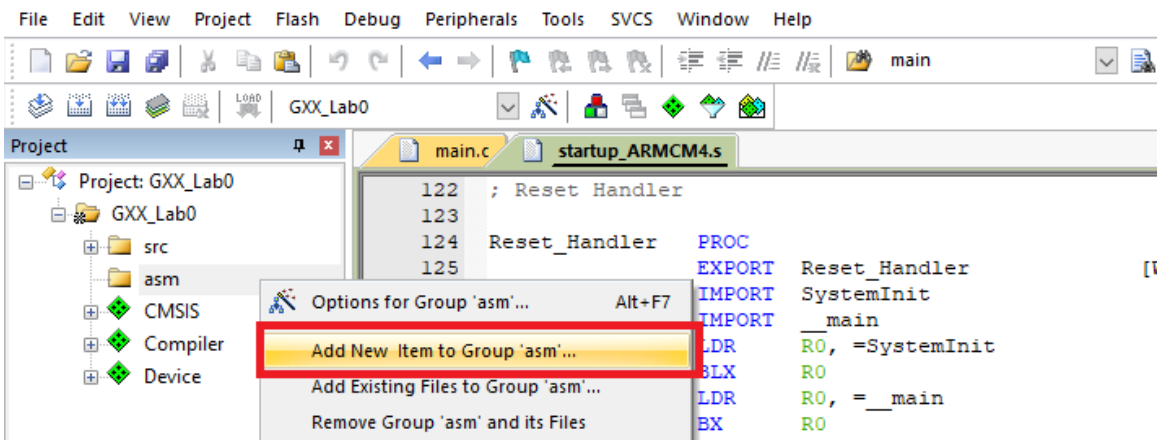

Figure 19: Adding a file to the *asm* group.

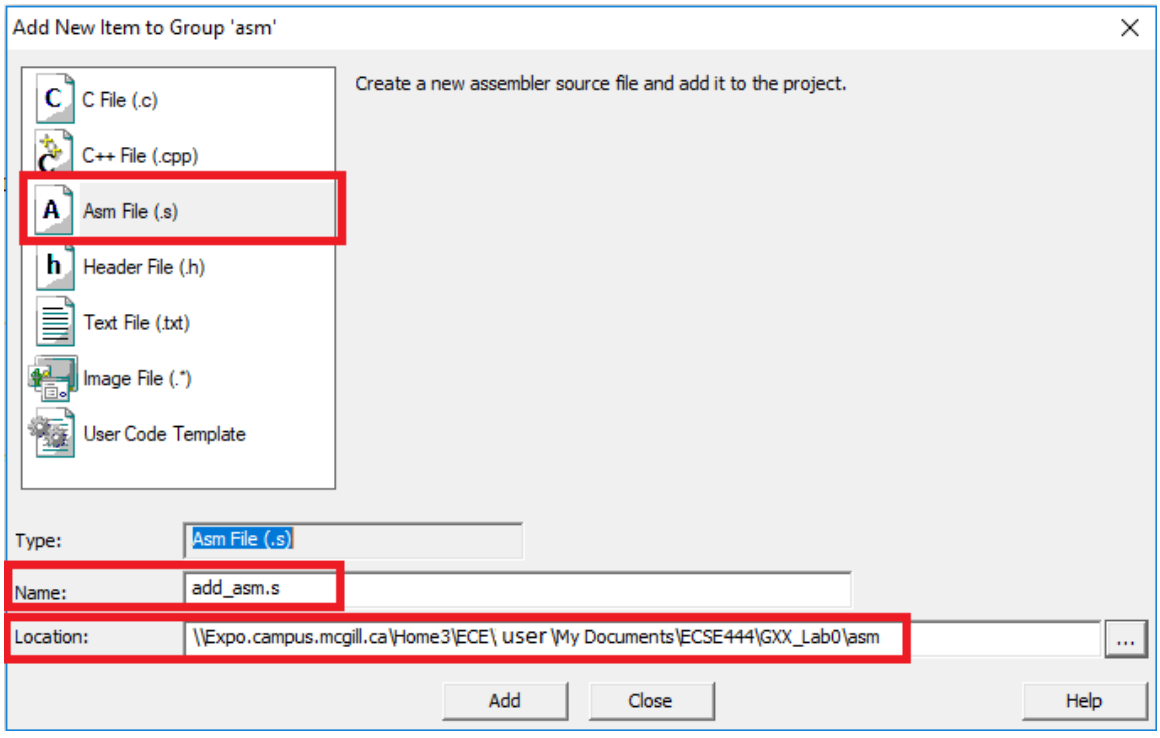

Figure 20: Adding the **add_asm.s** file.

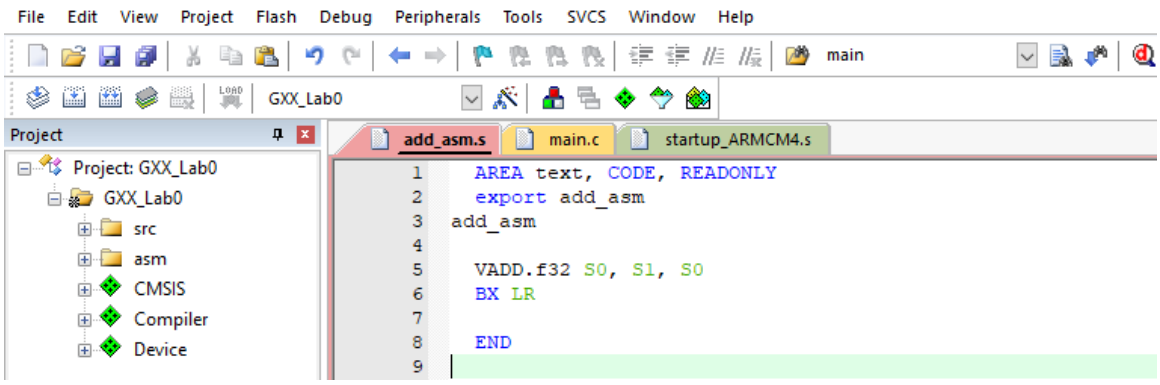

Figure 21: Assembly code for floating point addition.

2. Create a new file **add_c.c** in the *src* group in the same fashion as in Figure 12 and Figure 13 (but the filename is different!). Write the code shown in Figure 22 and save the file.

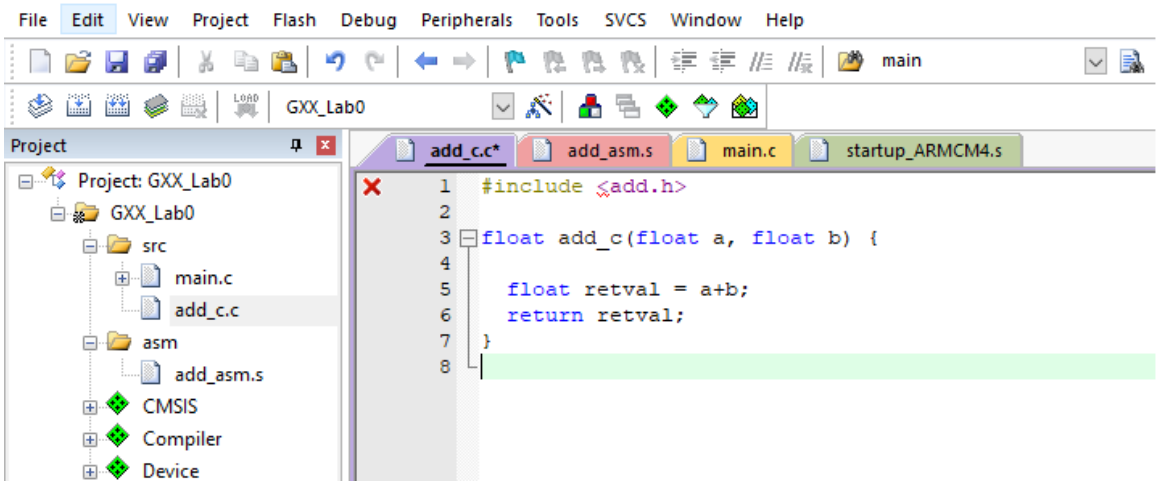

Figure 22: C code for floating point addition.

3. Create a new file **add.h**, fill in the code provided and save it in the *inc* folder as shown in Figure 23. This file does not need to be added to the project, simply saving it in the *inc* folder is enough.

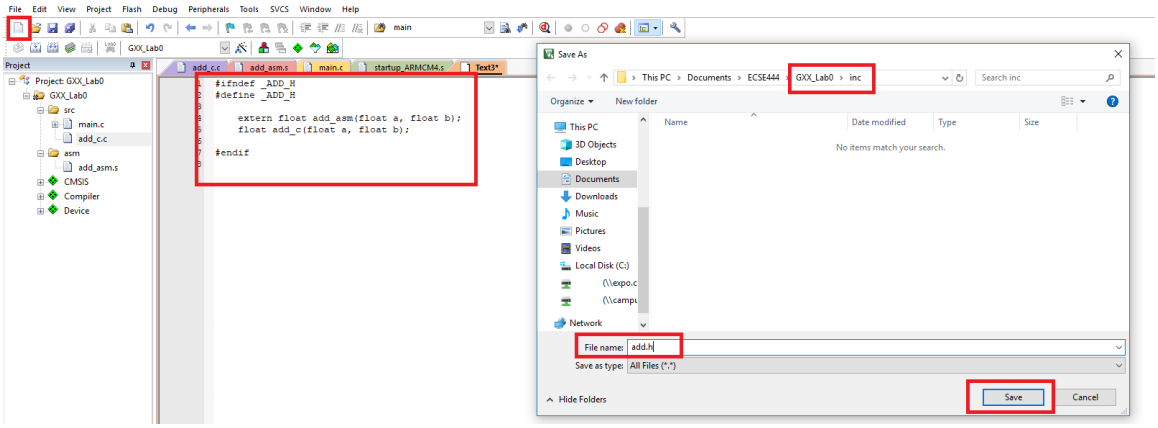

Figure 23: Creating the header file.

4. Finally, modify the **main.c** file as shown in Figure 24 and save it.

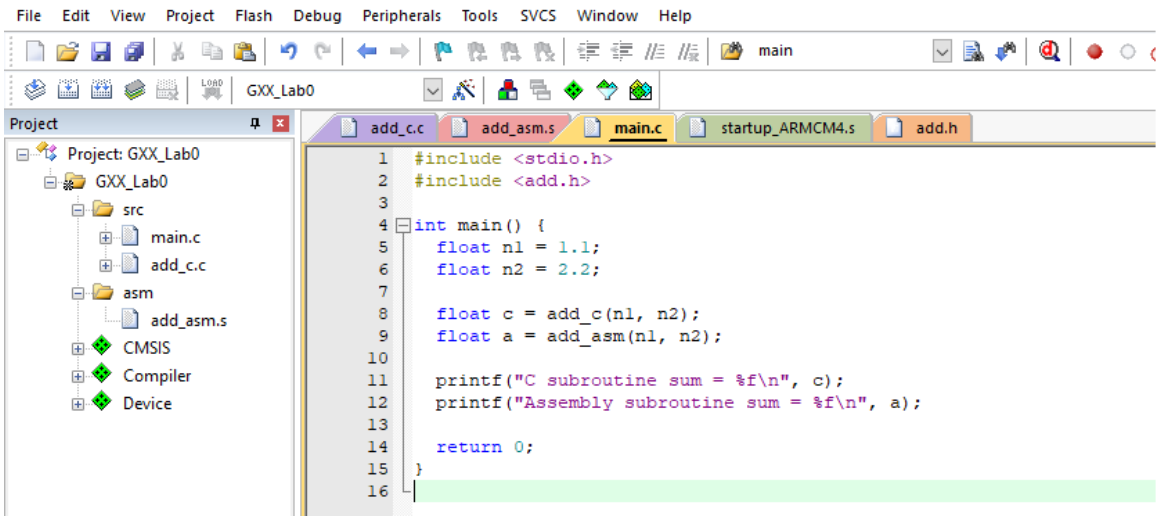

Figure 24: Modify the **main.c** file.

5. Now build the project (F7) and enter debug mode (ctrl+F5). When you run the project, you should see the expected output as shown in Figure 25.

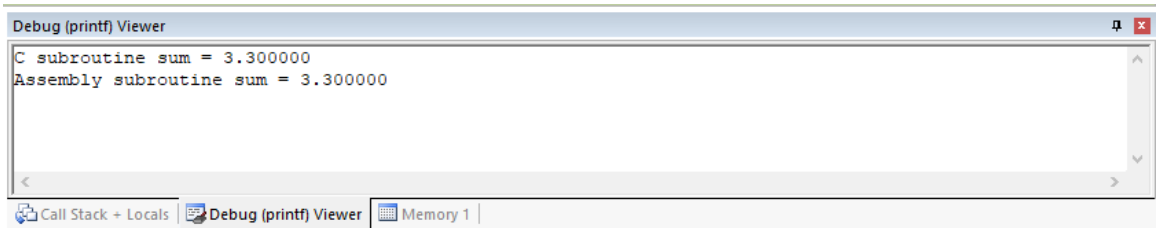

Figure 25: The program output.

Moving average filter

Now that we have been introduced to the Keil μVision IDE and become comfortable with the basic features, it is time to refresh your C and assembly programming skills by completing a simple programming task.

Raw data acquired from sensors is usually contains undesirable high frequency noise. The simple moving average (SMA) filter is a low-pass filter that can be used to smooth out the noisy data signal. It replaces each data point by the average of D samples centered around the data point. A SMA filter is characterized by the number D, also called its depth.

Given a signal $x[n]$ with N samples, the filtered signal $x_f[n]$ can be created by:

$$x_f[n] = \frac{x[n - \frac{D}{2}] + \cdots + x[n-1] + x[n] + x[n+1] + \cdots + x[n + \frac{D}{2}]}{D}$$

The task is to make a C and an assembly version of the SMA filter. Your subroutines should be able to operate on floating point data, do not need to have a return value, and have the following arguments:

- A pointer to the original signal $x[n]$.
- A pointer to the where the filtered signal $x_f[n]$ will be stored.
- The number of samples N in the original signal.
- The filter depth D.

You may make the following assumptions:

- $x[n] = 0$ when $n < 0$ or $n \geq N$.
- If D is even, then the extra sample will come from the preceding samples. For example:

$$x_f[n] = \frac{x[n-2] + x[n-1] + x[n] + x[n+1] + x[n+2]}{5} \qquad \text{when } D = 5$$

$$x_f[n] = \frac{x[n-2] + x[n-1] + x[n] + x[n+1]}{4} \qquad \text{when } D = 4$$