3. Programming the STM32F4-Discovery

The programming environment including the settings for compiling and programming are described.

3.1. Hardware - The programming interface

A program for a microcontroller is prepared on a personal computer (PC) using a suitable set of programs. First the source code of the program is written, than this code is translated into the machine code for a particular microcontroller, the STM32F407VG in our case. The developer then loads the code into the memory of the microcontroller. Several methods for loading are known. Most universal programmers (devices that fit between a PC and the target microcontroller, and allow the transfer of the code into the microcontroller) support JTAG method. The company ST introduced a method called serial wire debug (SWD), which uses fewer wires than JTAG method. The method SWD is supported by STM32F4DISCOVERY board, and the programmer that comes between a USB connector and the SWD pins of the microcontroller is available on the Discovery board.

The STM32F4DISCOVERY board hosts additional connector for SWD signals and can also be used to program external free standing microcontroller of the same family.

3.2. Software – The IAR EWB for ARM microcontrollers

The complete procedure of writing, compiling and testing the user program can be done in a set of programs called the development suite. We are going to use the IAR (the name of the company) Electronic Work Bench (EWB) for ARM microcontrollers. The current version is 6.7. All demo programs will be prepared using the free version of the software; this version is size-limited, but our needs are way below the limits of the suite. The current version IAR EWB and instructions for the installation can be downloaded from IAR website.

The IAR EWB requires at least the following files to create the machine code for the microcontroller:

- The user program, a text file written in “C” language.
- The file “startup_stm32f4xx.s”, a text file written in assembly language; this file contains instructions for the initial set-up of the microcontroller (stack, program counter, interrupt vector table, initial system clock …; the description of the file is given in its header).
- The file “system_stm32f4xx.c”, a text file written in “C”; this file contains functions for the detailed microcontroller set-up (system clock, clock distribution …; the description of this file is given in its header).

  In our case the file may have different names; since the file contains clock configuration commands it is best to prepare several files in advance, each for a different configuration of the clock and then simply include in the process of compiling the file that corresponds to the current needs. We will run the microcontroller at its maximum speed, and the corresponding file is named “system_stm32f4xx_CLK168_HSE8.c”.

- The header file “stm32f4xx.h”; this file defines processor used, names the registers and bits inside the STM32F4xx microcontroller, and defines some register structures. This is the only file that must be included from the user program in the process of compilation. The file has been modified from the original (obtained from the ST site):
  - by uncommenting the line 68 (select the microcontroller STM32F40xx by default),
  - by uncommenting the line 88 (allow the use of standard peripheral driver) and
  - by changing the HSE frequency in line 100 (from 25000000 to 8000000, as used at the STM32F4-Discovery board).
- The header file “system_stm32f4xx.h”; allows the definition of some stuff for the use of CMSIS.
- The header file “stm32F4xx_conf.h”; defines and includes some files mandatory for the project.

The peripherals included in microcontrollers STM32F4xx are complex, and their operation is configured and monitored using several control registers for each individual peripheral, the details are given in reference manual RM0090. Bits in control registers define the operation of peripherals, and can be manipulated from the “C” language by simple writes into these registers. However, the programmer must know the exact location and function of each individual bit, which may be an overwhelming task for such a complex microcontroller. To overcome this problem a standard to ease the use of control registers has been developed. It is called ‘Cortex Microcontroller Standard Interface System’, or CMSIS for short. The standard defines a set of functions written in “C” language, each of them manipulating exactly the bits to define the operation of a peripheral. When using the CMSIS the programmer does not need to know the exact location and function of bits, he/she must only know what the peripheral is capable of and call appropriate functions, which are located in CMSIS library; functions take care of setting bits in control registers correctly. The standard covers several families of microcontrollers, and also eases porting of software from one type of microcontroller to another. The complete library for STM32F4xx series microcontrollers is available at ST site as “STM32F4xx_DSP_StdPeriph_Lib_V1.1.0” at the time of writing. The directory structure of the expanded library is given in Fig. 3.2.

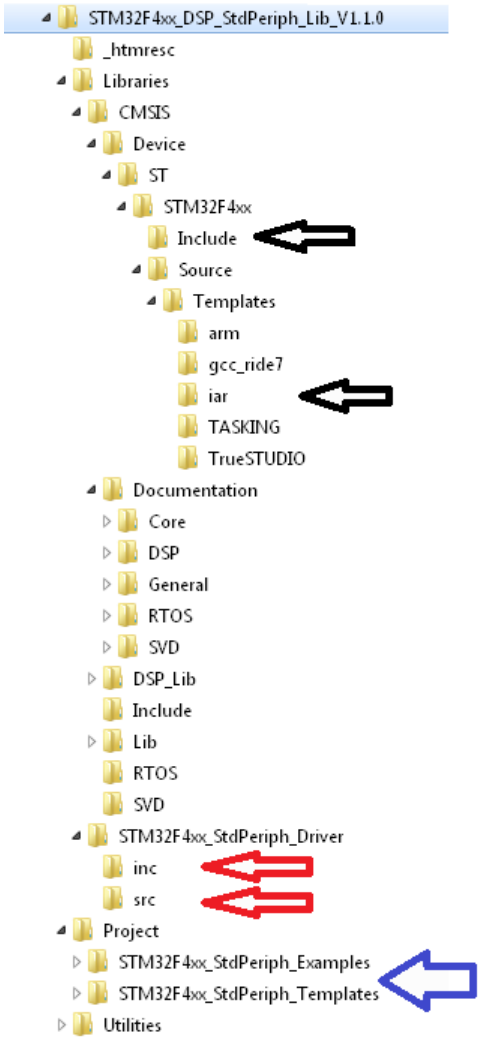

Fig. 3.2: The directory structure of the CMSIS library

The use of CMSIS standard is the reason that additional files are mandatory during the process of compilation beside the ones listed above. Two files are associated with every peripheral in the microcontroller:

- The header file defines the data structures used in accessing the peripheral and names the registers and constants (“stm32f4xx_$$$$.h”).
- The source file contains the actual functions to access the registers responsible for the behavior of peripherals (“stm32f4xx_$$$$.c”).

Here “$$$$” stands for the name of the peripheral. The compiler must know the location of all header and

include files used, and this must be specified in the compiler setup before the compilation process.

All files mentioned are available in the CMSIS library, see Fig. 3.2 for the expanded directory structure of the library. The files “system_stm32f4xx.h” and “stm32f4xx.h” are located in directory “Include”, upper black arrow. The programming environment dependent file “startup_stm32f40x.s” is located in directory “IAR”, lower black arrow. Please note that these files are original, unmodified ones, and it is most convenient to store the modified versions to a separate directory and call them from there. CMSIS header (*.h) and source (*.c) files describing peripherals are located in directories “inc” and “src” respectively, both red arrows. Some examples on the use of CMSIS library are given in directory “STM32F4xx_StdPeriph_Examples”, and templates for different compilers are given in “STM32F4xx_StdPeriph_Templates”, blue arrow.

The process of downloading the compiled program into the microcontroller requires one additional file with the description of downloading procedure and the destination memory for the program, this file is called linker configuration file. There are three options to choose from as far as the destination memory is concerned: the file may be programmed into the flash memory, the file may be downloaded into the random access memory inside the microcontroller or the program may be loaded into an external RAM. All three configuration files have the extension “.icf”, and can be found in “STM32F4xx_StdPeriph_templates\EWARM”. Since we want to make programming permanent we will only use the file named “stm32f4xx_flash.icf”.

Two additional files were prepared for these experiments, “LCD2x16.c” and “dd.h”. The first contains functions to display character strings and numbers at the alphanumerical LCD display, the second one defines some constants and interfacing pins. Both files should be made available during the compilation process.

3.4. Creating of a new workspace and a project within

Programs for microcontroller are prepared in a workspace, an imaginary working environment containing all the user supplied stuff to compose the final machine code. The workspace is associated with a folder on a computer disk. The workspace contains projects; each project holds a complete set of files (or references to files common to several projects) that constitute one user program. Typically all files needed for a project are stored in a designated sub-folder within a folder for the workspace to avoid the naming confusion of individual files within the workspace.

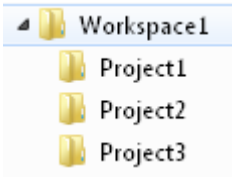

Figure 3.3: Folders created to host the workspace and three projects

The procedure for creating a new workspace and a project within will be described step-by-step. The directory structure to hold the workspace and projects is created first using the Windows Explorer,

Name	Date modified	Type	Size
dd	2.12.2013 13:57	H File	2 KB
LCD2x16	3.12.2013 8:50	C File	5 KB
startup_stm32f40xx	10.1.2013 11:54	S File	25 KB
stm32f4xx	17.12.2013 9:17	H File	533 KB
stm32f4xx_conf	5.12.2013 9:15	H File	4 KB
stm32f4xx_flash.icf	28.3.2013 9:09	ICF File	2 KB
system_stm32f4xx	10.1.2013 11:54	H File	3 KB
system_stm32f4xx_CLK168_HSE8	4.4.2013 11:03	C File	22 KB

Figure 3.4: The mandatory files are copied into the project directory

the directory structure is given in Fig. 3.3, for hosting three projects (Project1, Project2 and Project3) within a workspace called Workspace1. Additional directories can be added later to host more projects in the same workspace. A set of files as listed in Fig. 3.4. is copied into each of the project directories for easy access and inclusion into the user program. Note that some of these files are modified versions of the originals given in the CMSIS library.

3.4.1. Create a workspace

Initially the "IAR Embedded Workbench" is opened resulting in an empty space in the center of the program window and empty workspace in the left part of the window, Fig. 3.5. This state can be achieved also by clicking the "File" option from the menu, followed by "New" and "Workspace".

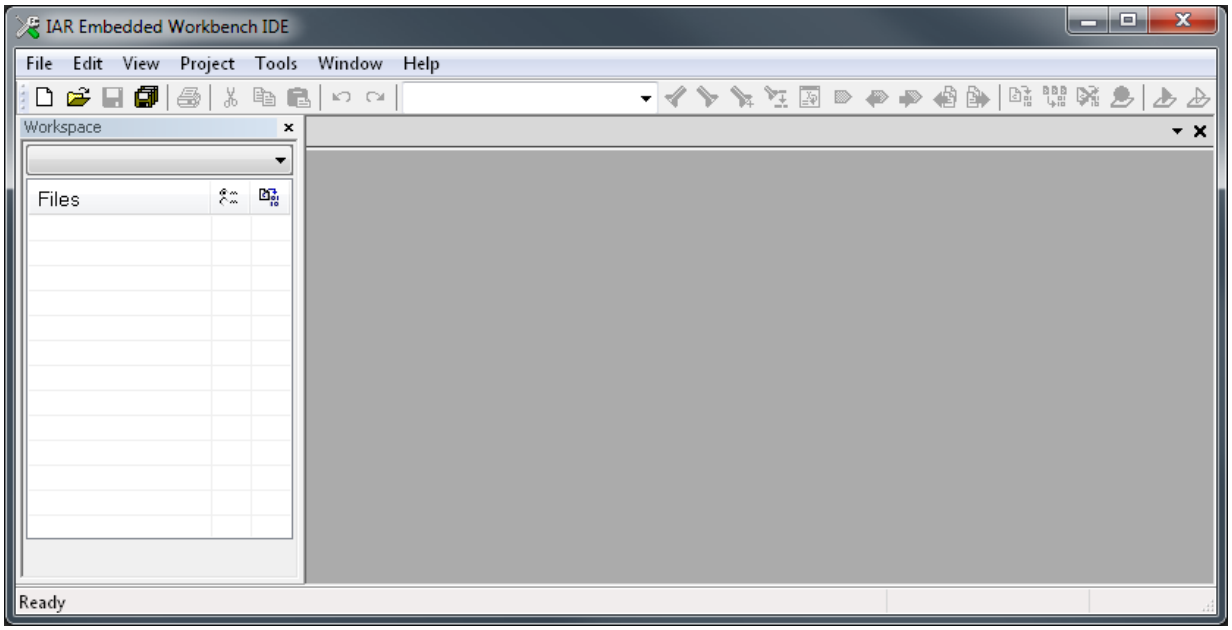

Figure 3.5: A newly opened »IAR Embedded Workbench« with an empty workspace on the left

3.4.2. Create a project

- Click "Project" option in the menu, and then click "Create New Project …" to get a window as shown in Fig. 3.6.

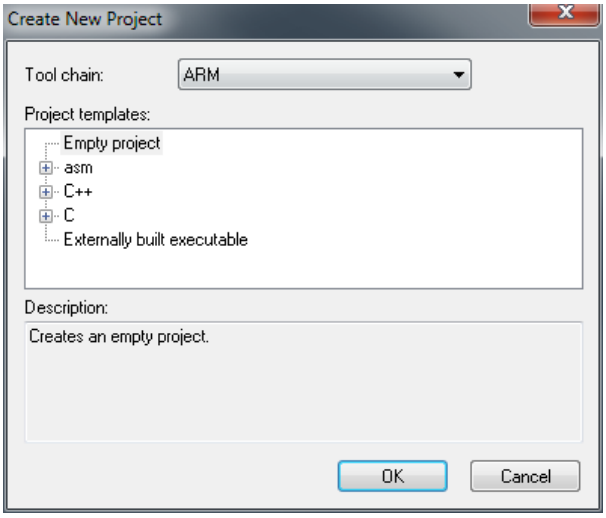

Figure 3.6: A window to create a new project

- Select “Empty project” and click “OK” to get a window “Save As”. Navigate to the folder where the project is to be saved (The “Workspace1\Project1” in our case). Under “File name:” type in the name of new project, like “MyFirstProject”, and click “OK”. The skeleton of the new project will be created in the stated directory, and the Workspace window (Fig. 3.5, left part of the window) of IAW EWB will contain the name of the newly created project.
- Save the newly created workspace by clicking “File” option in the menu and then clicking “Save Workspace”, navigate to the appropriate folder for the workspace (“Workspace1” in our case), state the name (“MyFirstWorkspace” for instance) of the newly created workspace in “File name:” and confirm saving by clicking “Save”.

3.4.3. Add files to project

Every project must contain at least files listed in the Fig. 3.4. Some of them are included from the user program, some are specified at the time of compilation, and some must be added in the project itself. The last are “startup_stm32f4xx.s”, “system_stm32f4xx_CLK168_HSE8.c” and the “C” file containing the user program.

- The first two files are added to a project by right-clicking the name of the project in the workspace window and selecting option “Add” -> “Add Files” from a drop-down menu. This opens a window where appropriate files can be selected (from the “Workspace1\Project1” folder in our case).
- Additionally, a new file with the user program must be created. A new file is created by clicking the leftmost icon in the toolbar (a symbol for empty pages). An empty page appears in the right IAR window, and the user can write new text onto the page. The user program must start with the line “#include “stm32f4xx.h”” to allow the use of predefined names for register in the microcontroller. The file must include at least the function “void main (void)”. The file should be saved in the folder of the choice (“Workspace1\Project1” in our case) and can have any name, like “MyFirstFile.c”. This file must also be added to the project using the same procedure as stated previously.

The result is the IAR EWB window shown in Fig. 3.7.

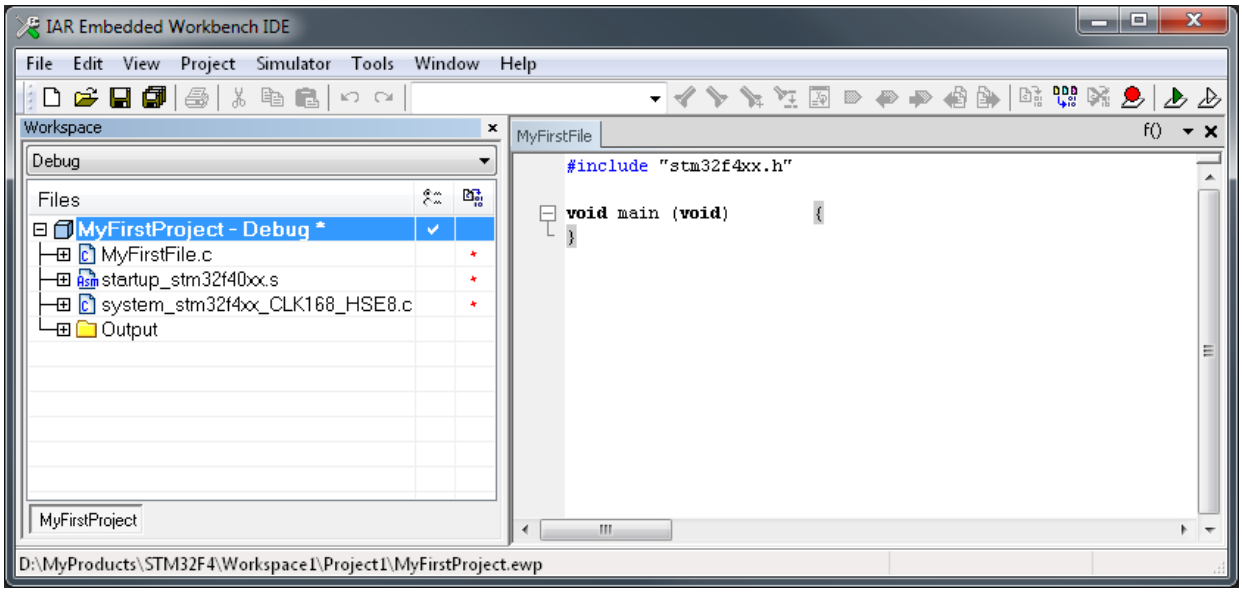

Figure 3.7: A fully defined minimal project within a workspace

3.4.4. **Define options for a project**

The process of altering the user provided source code into machine code consists of compiling and linking. The IAR Embedded Workbench does both, but needs additional information for the process. This information can be supplied by clicking the option “Project” in the menu and then “Options..”. This brings out a window where options for the operation of the compiler and linker can be configured.

- The IAR Embedded Workbench needs to know the microcontroller to compile for. Under “General Options” click “Device” and select the appropriate ST microcontroller (STM32F407VG) as shown in Fig. 3.8.

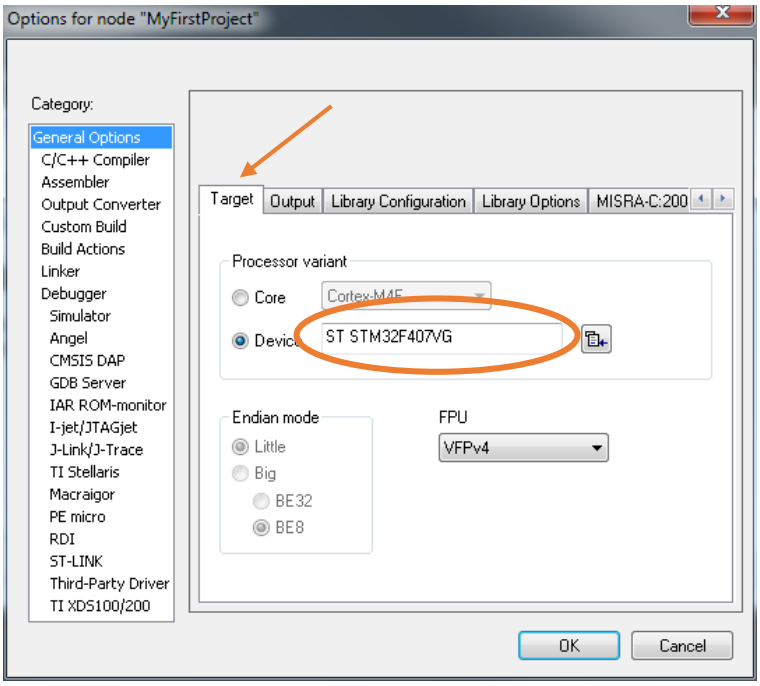

Figure 3.8: Select the microcontroller

- Enable the CMSIS standard and libraries by ticking the appropriate tick box, Fig. 3.9.

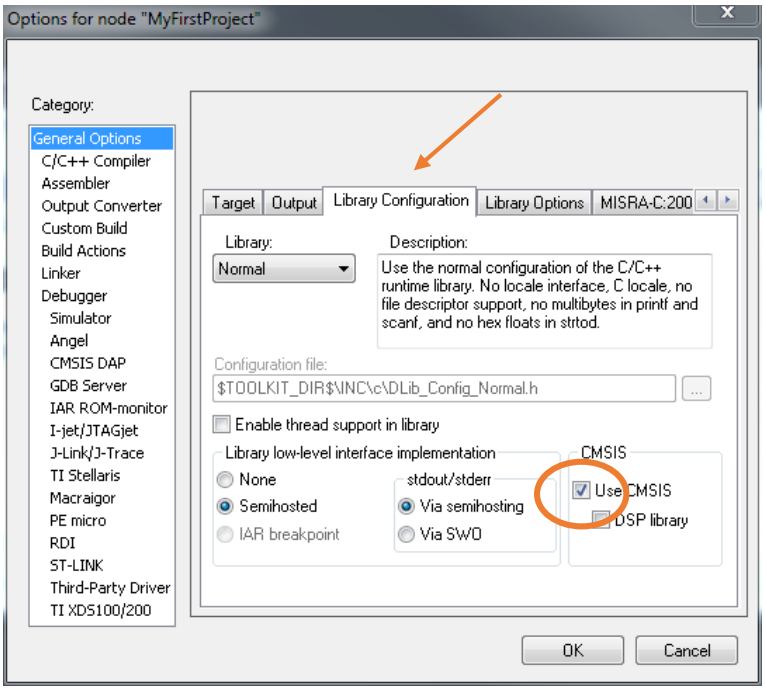

Figure 3.9: Enable the use of CMSIS standard

- Set the level of optimization for compiler, Fig. 3.10. The default is “Low”, but we can safely select “Medium” to speed-up the execution of the program. Selecting the option “High” enables the compiler to actually modify the user program in order to speed-up the execution, and is to be used with caution.

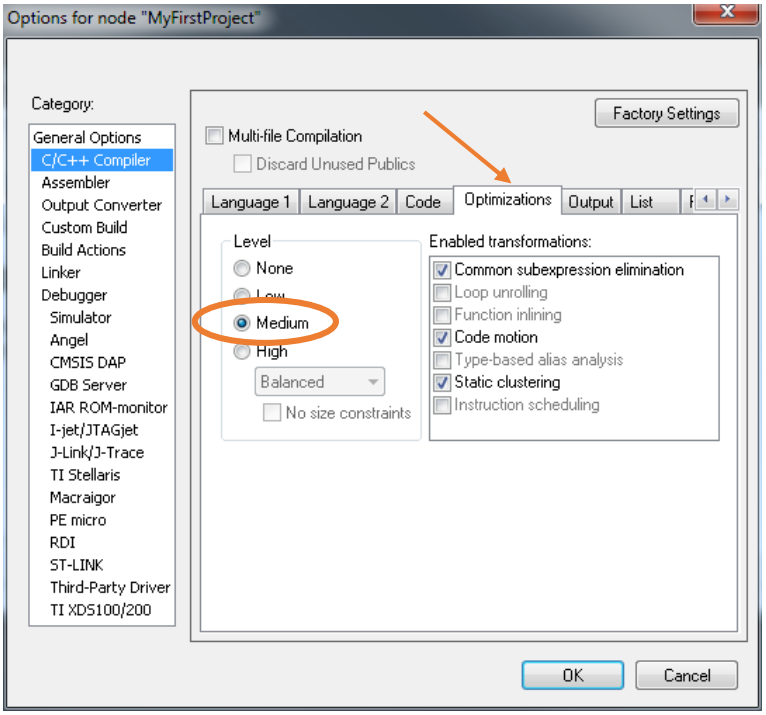

Figure 3.10: Enable optimization, level “Medium”

- The IAR Embedded Workbench needs to know the location of header and include files conforming to CMSIS standard to handle the peripherals, Fig. 3.11. The folders for these files are specified relatively to the directory where the project is located (variable $PROJ_DIR$). From this directory

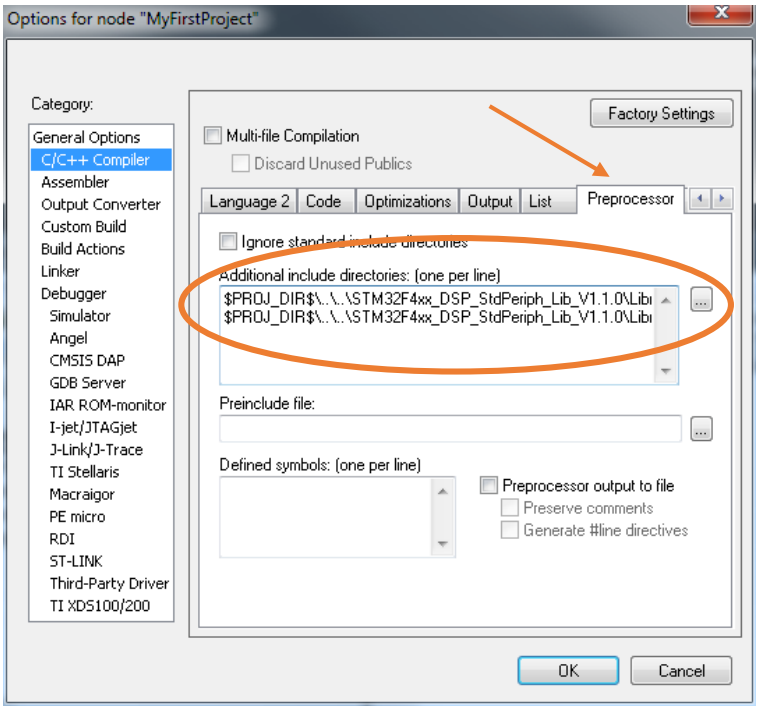

Figure 3.11: State the location of CMSIS header and include files

the relative path leads two levels up ("\..\..") and then downwards to directory named "\STM32F4xx_DSP_StdPeriph_Lib_V1.1.0\Libraries\STM32F4xx_StdPeriph_Driver\inc" and "\STM32F4xx_DSP_StdPeriph_Lib_V1.1.0\Libraries\STM32F4xx_StdPeriph_Driver\src", see Fig. 3.2 for reference.

- The machine code can be prepared to run from flash memory, from RAM or from external RAM, but the linker must know the desired destination for the code. We define if by selecting the appropriate linker configuration file "stm32f4xx_flash.icf" as shown in Fig. 3.12.

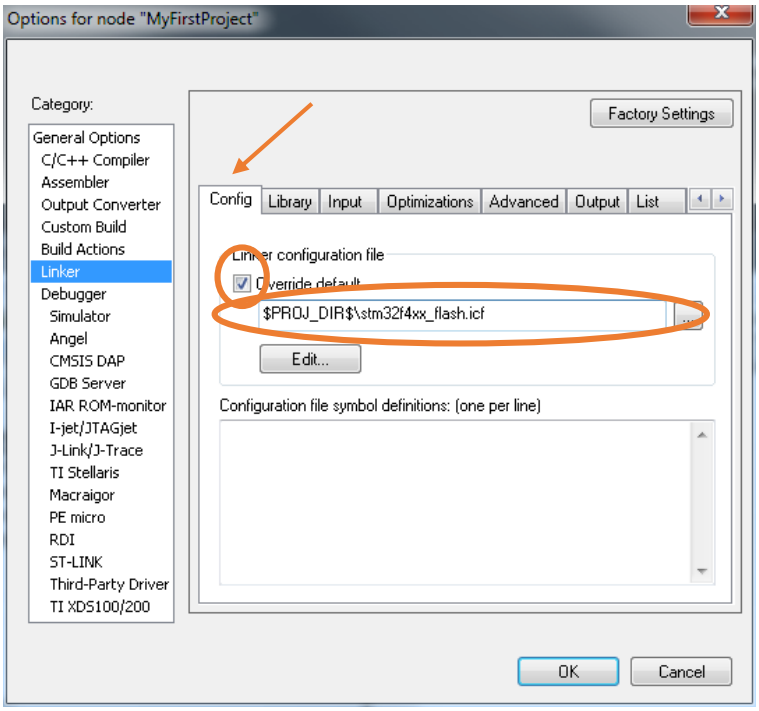

Figure 3.12: The machine code will be downloaded into the flash memory, and it must be linked accordingly

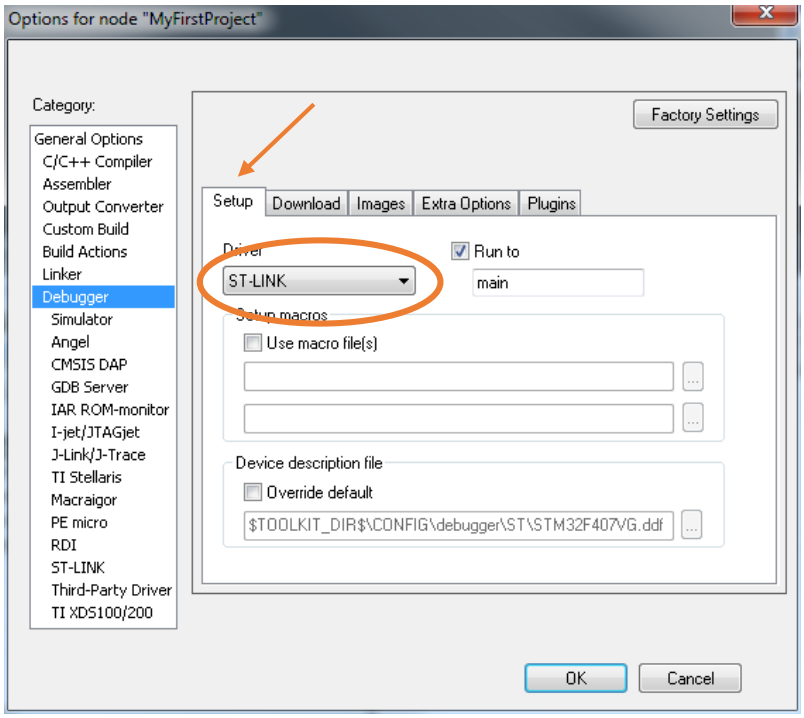

Figure 3.13: ST-LINK / 2 programmer is used for downloading and debugging

- We will use the ST-LINK / 2 programmer embedded onto the STM32F4-Discovery board to download and debug the machine code, so we need to tell the IAR EWB about our selection, see Fig. 3.13.
- Since we want to load the machine code into the flash memory the programmer must use the appropriate protocol for the download, namely the flash loader, see Fig. 3.14.

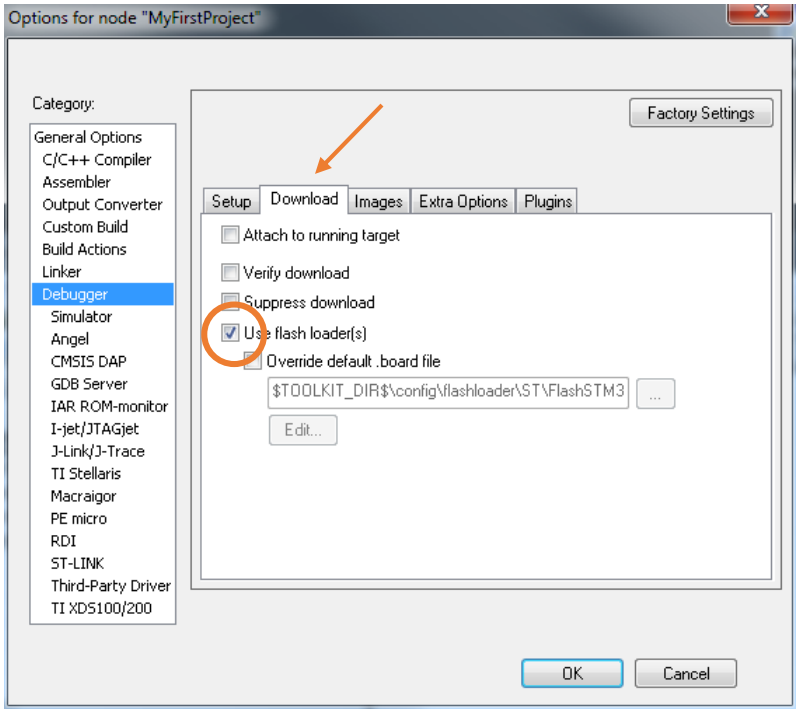

Figure 3.14: The programmer should load the machine code into flash memory

- The target microcontroller is connected using a ST-LINK and SWD protocol, Fig. 3.15.

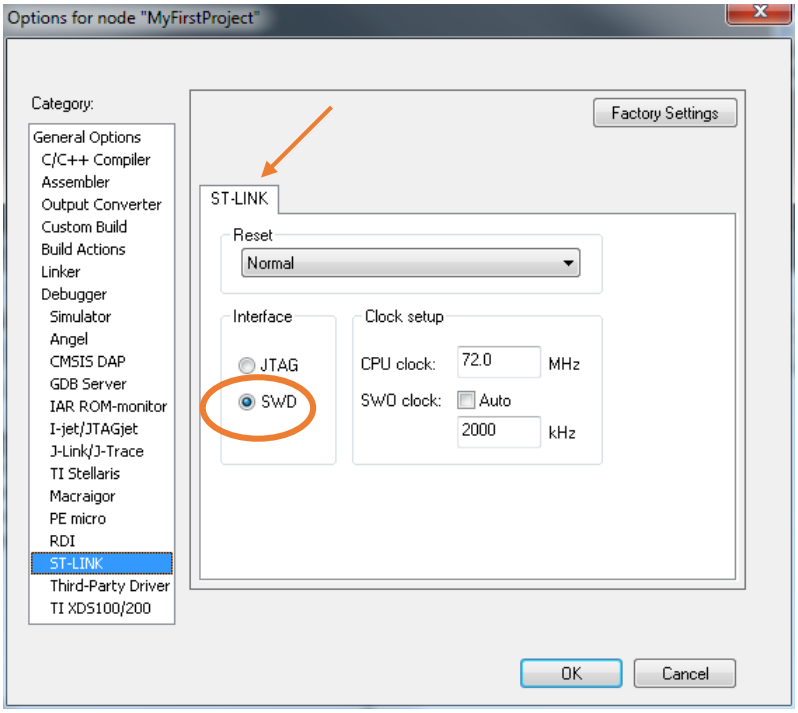

Figure 3.15: Use SWD as the preferred protocol for downloading the code

3.5. Compiling and debugging a project

Once the user program is typed-in and saved, and all the above settings are in place, one can run the compiler and linker to convert the user program into the machine code.

- The compilation and linking can be initiated either from the menu ("Project" -> "Compile") or simply by pressing F7. There is also an icon on the toolbar to initiate the compilation (Fig. 3.7, fifth icon from the right on the toolbar).
- The compilation, linking and download to the target processor can be initiated by simply clicking the second icon on the right from the toolbar. If either compiler or linker finds errors during the process, they issue warning and/or error messages and stop the process.

The debugging of the program can be either pure simulation in software of the personal computer or running in real hardware.

- When the first is desired, the step described in Fig. 3.13 must be different: the "Simulator" must be selected instead of "ST-LINK". The machine code will not be sent to the microcontroller, but kept inside the personal computer instead. The personal computer will read the machine code step by step and simulate the behavior of real microcontroller. This mode is essential for testing algorithms for all possible values of input variables, but cannot simulate the real hardware (interrupt requests, timers/counters, ADCs, DACs, …).
- When the second is desired, the real hardware with the microcontroller is mandatory. In this case the program and hardware can be fully tested in real life situation, as we are going to do.

The download of the program is initiated by pressing the button with the green triangle pointing right, or can be initiated using the menu. The suite then enters the debug mode, and Fig. 3.16 gives a sample of the debug session in progress.

The execution of the program is started by clicking the "Run" button, the one with three arrows to the right, just left to the button with the red "X". Clicking the button with a hand stops the execution, and clicking the button with a single arrow pointing to the left resets the microcontroller. Various options for single-stepping are available, the currently executed line is marked with a green arrow left of the listing. The simulation/emulation stops when the user clicks the button with the red "X".

The execution of the running program can be stopped at a certain line of the program using breakpoints. A breakpoint can be set onto a specific line of code using the double click in front of the line, which brings a red dot beside the line with the breakpoint. Please note that the compiler optimizes the user supplied source code, and some lines of the user program can be omitted and embedded in nearby lines. Breakpoints cannot be placed onto the missing lines, and in such case the suite places a breakpoint to the nearest existing line of the user program.

The content of all registers is available for inspection and alteration when the program is stopped either manually or after a breakpoint. One needs to click option "View" in the main menu and then "Registers" in the drop-down menu. This opens a new window inside the IAR EWB window with the content of the CPU registers, see Fig. 3.16, right. The user can select to inspect or modify registers belonging to peripherals by clicking the drop-down box at the top of this window and selecting the desired peripheral. Initially, the content of registers is shown in hex notation, but can be broken down to bits by clicking the + sign in front of the register name.

The value of variables used in program can be shown by clicking the "Watch" option in the "View" menu. This opens a new window, and the user can write the name of the desired variable into an

empty slot. The value of the variable is returned to the right of the name when the program is stopped. It is shown in hex notation initially, but can be converted to decimal, ASCII or binary on demand. Due to optimization during the compiling of the program some variables might not be available for watching. In such case the value will not be returned, but a message will be displayed instead. Variables can be also inspected or modified when “View” -> “Quick View” is selected, but with different options.

Figure 3.16 gives an example of the IAR Embedded Workbench in the debug mode showing the project window (where green arrow is used to point to the line to be executed when simulation is started) and register window (showing the content of register associated with port A in hex notation.

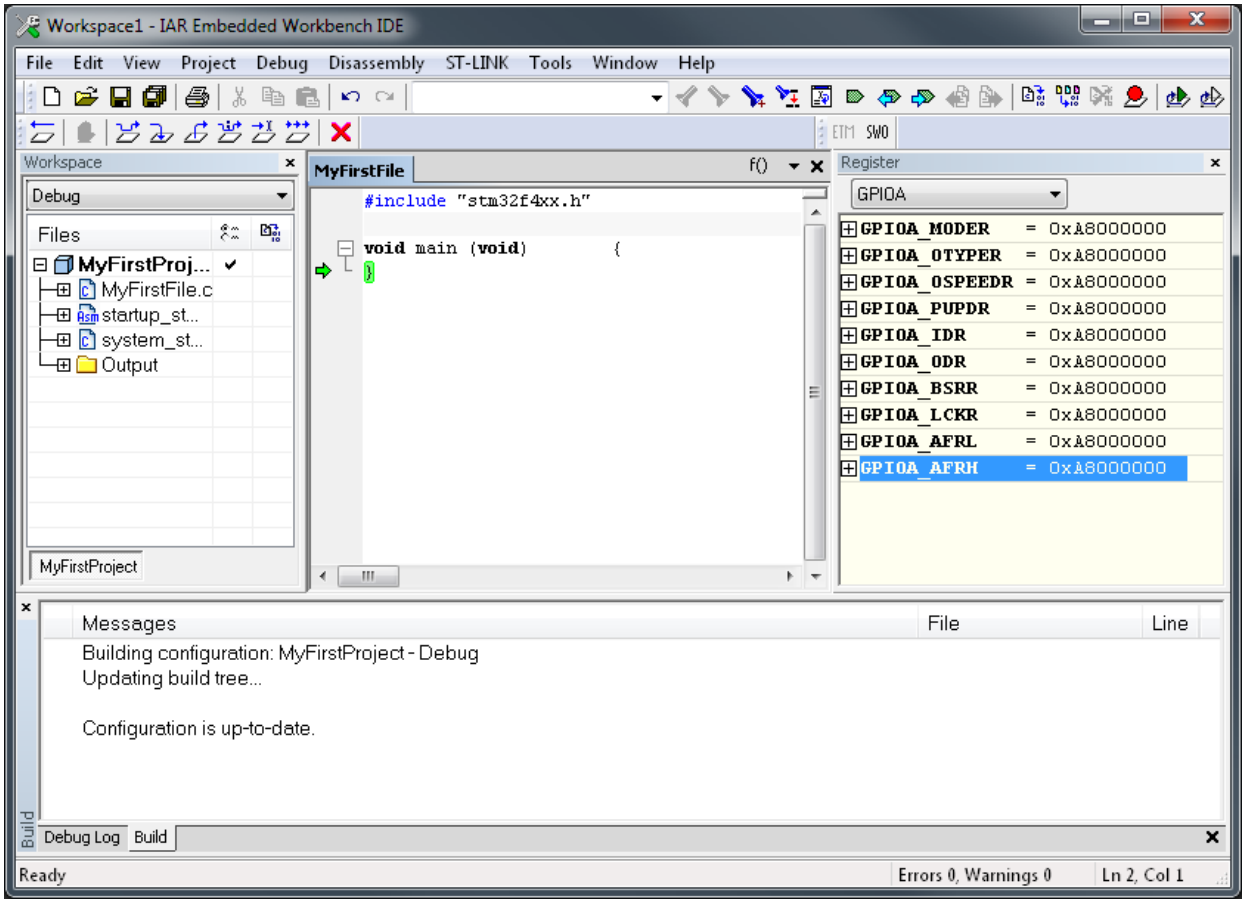

Figure 3.16: An example of a debug session in progress