3. Programming

Once the machine code containing the user program is prepared on a personal computer, the user must load the code into the memory of the processor. Several methods for loading are available. Most universal programmers (devices that fit between a personal computer containing the machine code and the target microcontroller, and allow the transfer of the code into the microcontroller) support JTAG method. The company ST introduced another method called SWD (serial wire debug), which uses fewer lines than required for JTAG method. Additionally, SWD method is supported by demo board STM32F4DISCOVERY (roughly 15 €), and this board can be used to load the machine code into any board with microcontroller ST32F4xx. The company ST offers a programmer called ST-LINK / 2 which also supports both JATG and SWD methods and is reasonably priced (about 25 €).

3.1. Hardware – SWD

The board contains the connector for SWD bus to implement SWD method of programming. Fig. 1 gives the correct connections for the STM32F4DISCOVERY board used as programmer, and fig. 2 gives the correct connections with ST-LINK / 2.

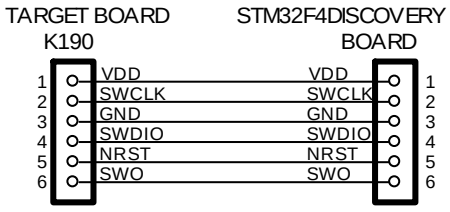

Figure 1: The connection between the target and the STM32F4DISCOVERY board.

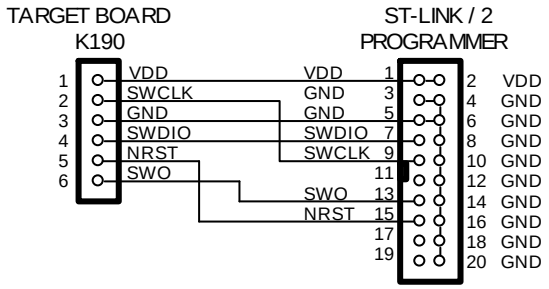

Figure 2: The connection between the target board and the ST-LINK / 2 programmer

The ST recommends connecting more than one GND line at the ST-LINK / 2 programmer. If flat cable is used then every second wire should be GND to reduce the crosstalk between wires.

3.2. Hardware – JTAG

The programming using JTAG bus is offered as a standard way of programming microcontrollers. However, JTAG is not implemented on present board, and will therefore not be described here.

3.3. Software – IAR

The IAR Embedded Workbench will be used for the preparation of program and compiling. The current version is 6. All programs will be prepared using the free version of the software. The current version IAR Embedded Workbench for ARM microcontrollers can be downloaded from IAR website, the website also provides instructions for the installation.

The files need to create the machine code for the microcontroller are at least:

- The user program, a text file written in "c" language.
- "startup_stm32f4xx.s", a text file written in assembly language; this file contains instructions for the initial set-up of the microcontroller and:
    - sets the initial SP (stack pointer) and PC (program counter) to point to program start,
    - sets the vector table entries with the exceptions ISR (interrupt service routine) address,
    - issues a call to the function for configuration of the system clock and branches to main in the C library (which eventually calls main()).
- "system_stm32f4xx.c", a text file written in "c"; this file contains two functions:
    - SystemInit(): Setups the system clock; this is called from "startup_stm32f4xx.s" and
    - SystemCoreClockUpdate(): Updates the variable SystemCoreClock to mirror the selected clock settings.
- "stm32f4xx.h", a text file written in "c"; this file defines names for registers and bits inside the STM32F4xx microcontroller.

Some other "c" header, source and definition files are called from the above files and are mandatory. The best is to copy all these files but the user program into a common folder and tell the IAR compiler the name of this folder. A possible content of such folder is listed in Fig. 3.

Name	Date modified	Type	Size
core_cm4.h	28.10.2011 10:31	C Header file	78 KB
core_cm4_simd.h	28.10.2011 10:31	C Header file	24 KB
core_cmFunc.h	28.10.2011 10:31	C Header file	16 KB
core_cmInstr.h	28.10.2011 10:31	C Header file	16 KB
misc	28.10.2011 10:31	C Source File	12 KB
misc.h	28.10.2011 10:31	C Header file	7 KB
startup_stm32f4xx	28.10.2011 10:31	S File	24 KB
stm32f4xx.h	28.10.2011 10:31	C Header file	519 KB
stm32f4xx_conf.h	19.6.2012 15:44	C Header file	4 KB
stm32f4xx_flash.icf	19.6.2012 15:44	ICF File	2 KB
stm32f4xx_rcc	28.10.2011 10:31	C Source File	73 KB
system_stm32f4xx	19.6.2012 15:44	C Source File	21 KB
system_stm32f4xx.h	4.6.2012 19:06	C Header file	3 KB

Figure 3: This is the content of a directory named »D:\STM32F407_exp\local includes«

Programs are written inside a »workspace«, which is typically located inside one single folder on a computer disk. The workspace contains "projects"; each project inside the workspace contains one user program. Typically each project is stored in a separate sub-folder to avoid the confusion with naming of the files, and holds one or more files that constitute a single user program.

3.4. Creating of a new workspace and a project within

The procedure for creating a new workspace and project within will be described step-by-step. Initially the “IAR Embedded Workbench” is opened resulting in an empty window in the center and empty workspace in the left part of the window, Fig. 4.

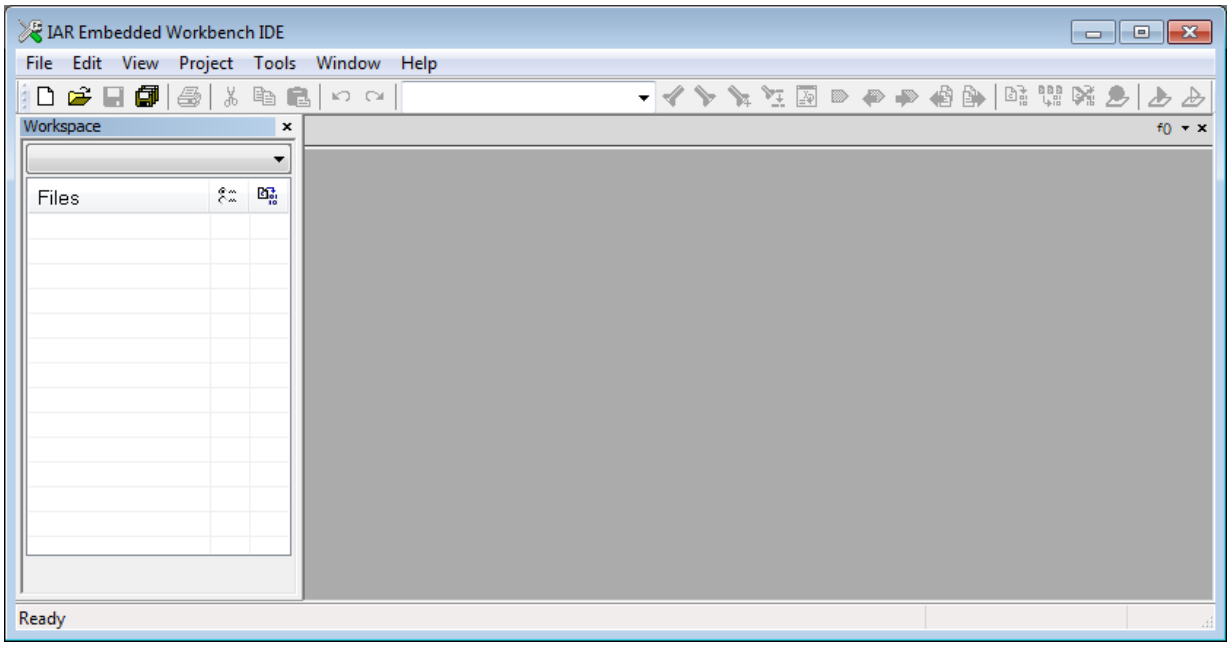

Figure 4: A newly opened »IAR Embedded Workbench« with an empty workspace on the left

3.4.1. Create new project

- Click “Project” in the menu, and then click “Create New Project …” to get a window as shown in Fig. 5.
- Select “Empty project” and click “OK” to get a window “Save As”. Select or create a folder for the newly created project, like “D:\my_projects\project_1”. 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 directory specified, and the Workspace window (Fig. 4) will contain the name of the newly created project.

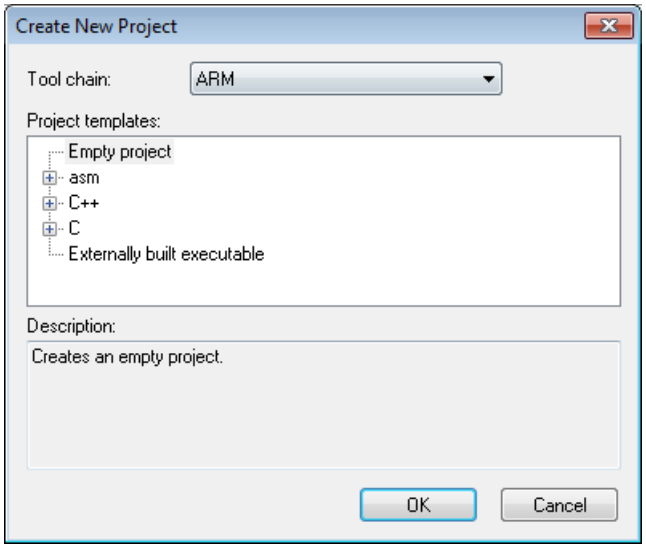

Figure 5: A window to create new project

- Save the newly created workspace by clicking “File” option in the menu and then clicking “Save Workspace”, select the appropriate folder for the workspace (“d:\my_projects” in our case), state the name (“MyFirstWorkspace” for instance) of the newly created workspace in “File name:” and confirm saving by clicking “Save”.

3.4.2. Add files to project

Every project must contain at least “startup_stm32f4xx.s”, “system_stm32f4xx.c” and the user “c” file containing at least “void main(void)” function.

- The first two files must be 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 »D:\STM32F407_experiments\local includes« folder in our case).
- Additionally, a new user file with the user program must be created. A new file is created by clicking the first icon in the toolbar on the top-left (white rectangle). An empty page appears in the IAR window, and the user can write new text onto the page. The user file must contain the line “#include “stm32f4xx.h”” at the beginning to allow the use of predefined names for register in the microcontroller. The file must include the function “void main (void)” among others. The file should be saved in the folder of the choice (“D:\my_projects\project_1” in our case) and can have any name. This file must also be added to the project using the same procedure as stated before.

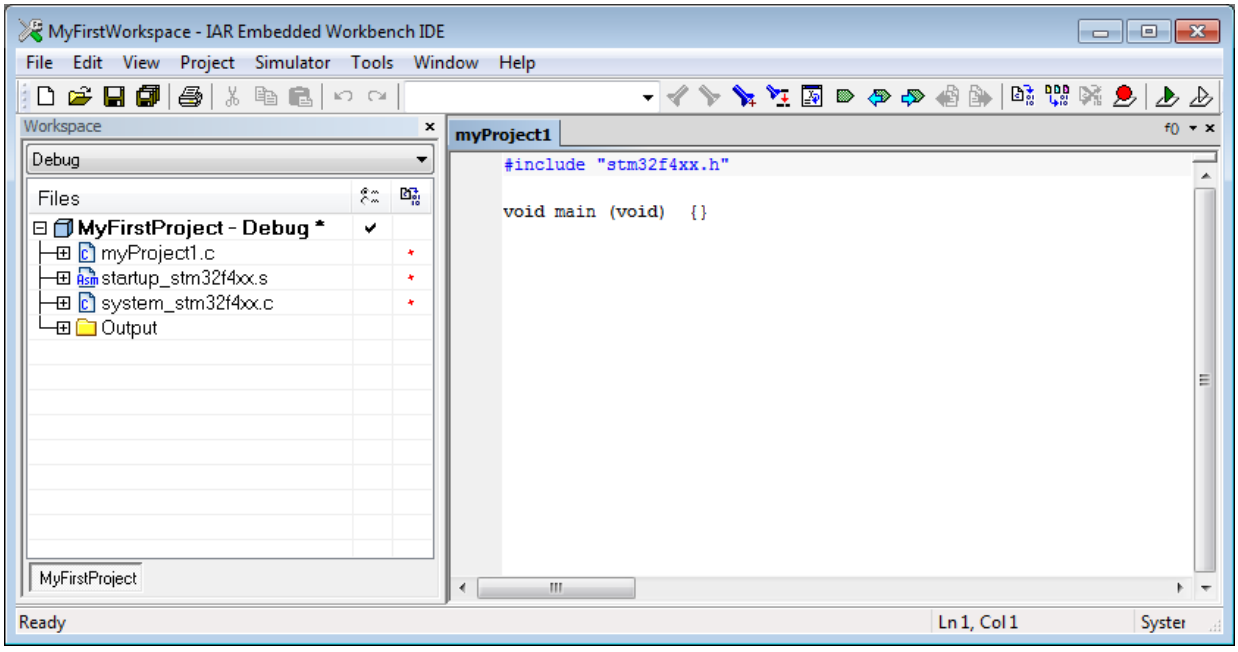

Figure 6: A fully defined minimal project within a workspace

3.4.3. Define options for a project

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

- The IAR Embedded Workbench needs to know the microcontroller to compile for. Under "General Options" click "Device" and select the appropriate ST microcontroller as shown in Fig. 7.

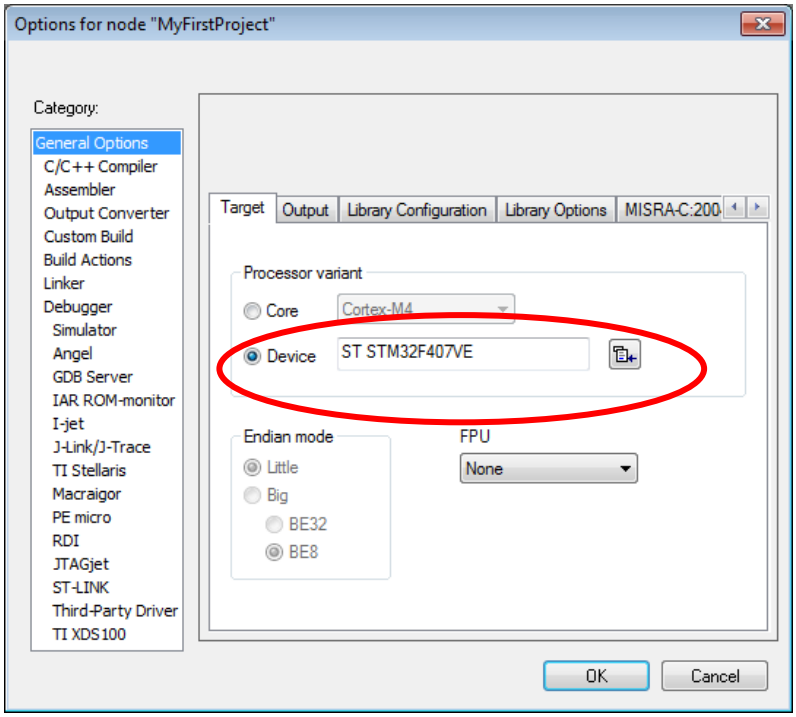

Figure 7: Select the microcontroller

- The IAR Embedded Workbench needs to know the location of files to include in the project, in our case »D:\STM32F407_experiments\local includes«.

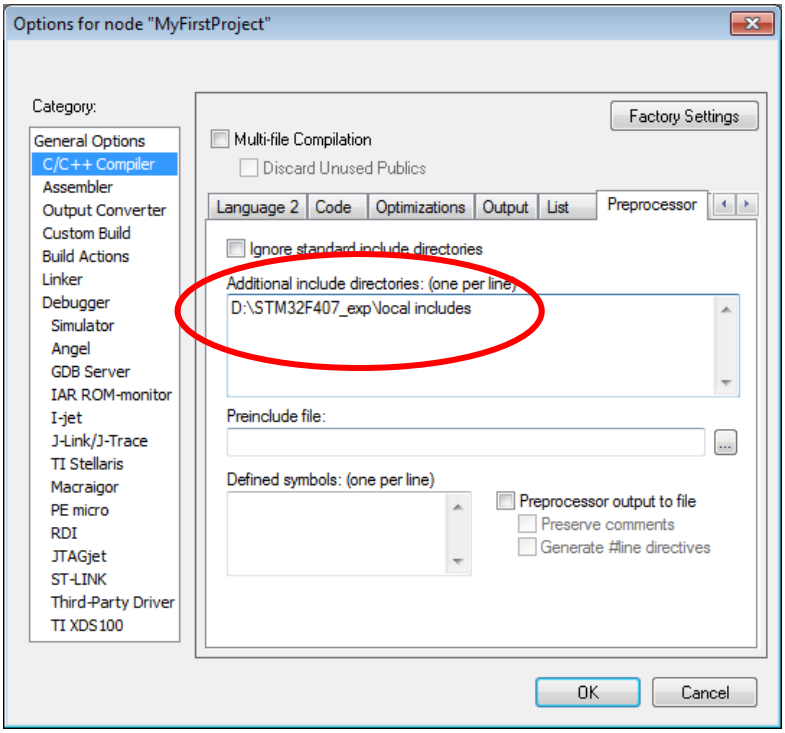

Figure 8: State the name of the folder with files to include

- The machine code can be prepared to run from flash memory, from RAM or from different sources, but the linker must know the desired location of machine code. We define if by selecting the appropriate linker configuration file “stm32f4xx_flash.icf” as shown in Fig. 9.

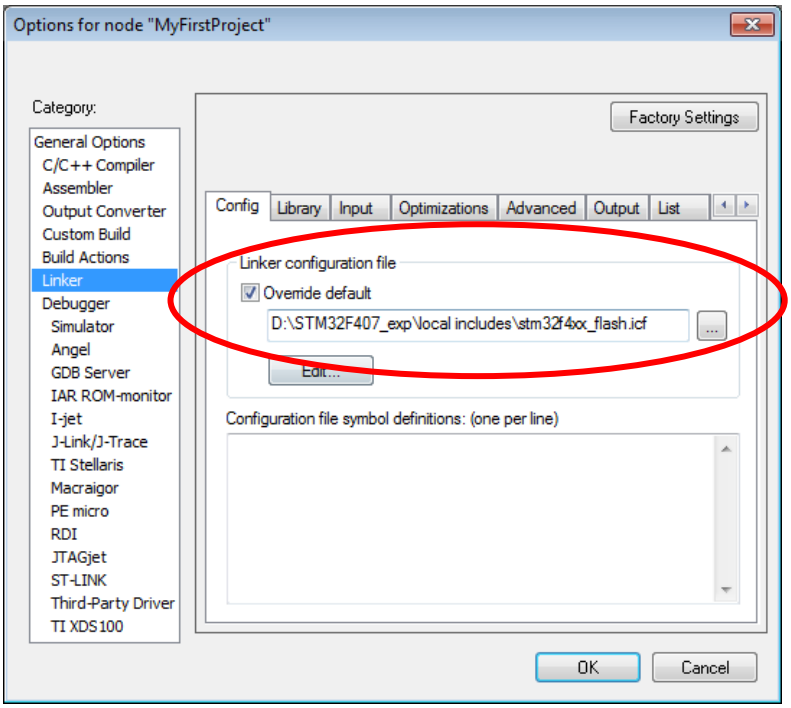

Figure 9: Define linker configuration file to select the location of machine code in the microcontroller

- We will use the ST-LINK / 2 to download the machine code and debug it, either the one embedded onto the DISCOVERY board or the stand-alone version, so we need to tell the debugger out hardware, see Fig. 10.

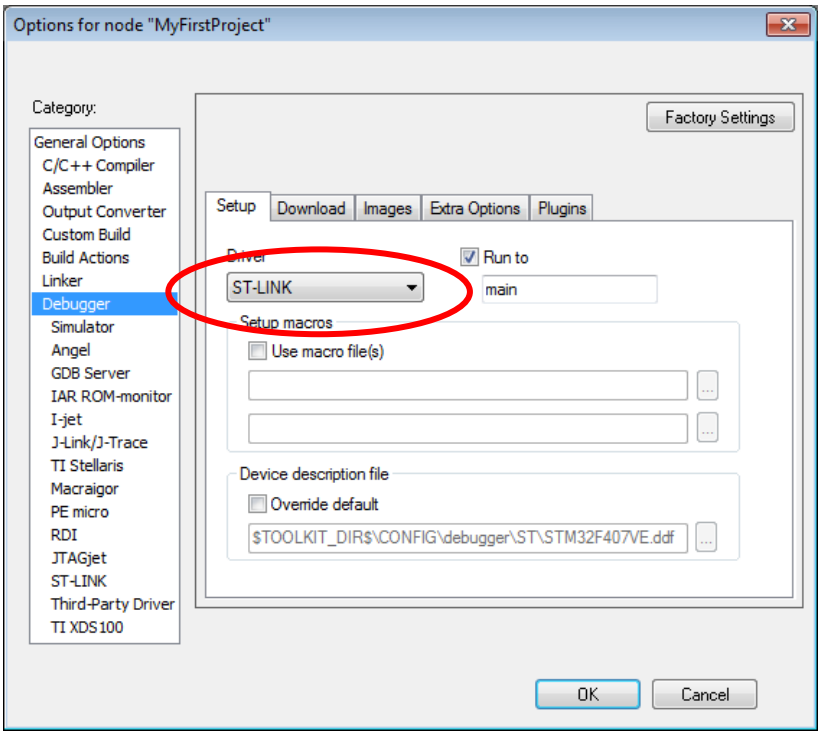

Figure 10: We use ST-LINK / 2 for downloading and debugging

- Since we are to download into flash memory of the microcontroller we need to use protocol for loading into the flash, see Fig. 11.

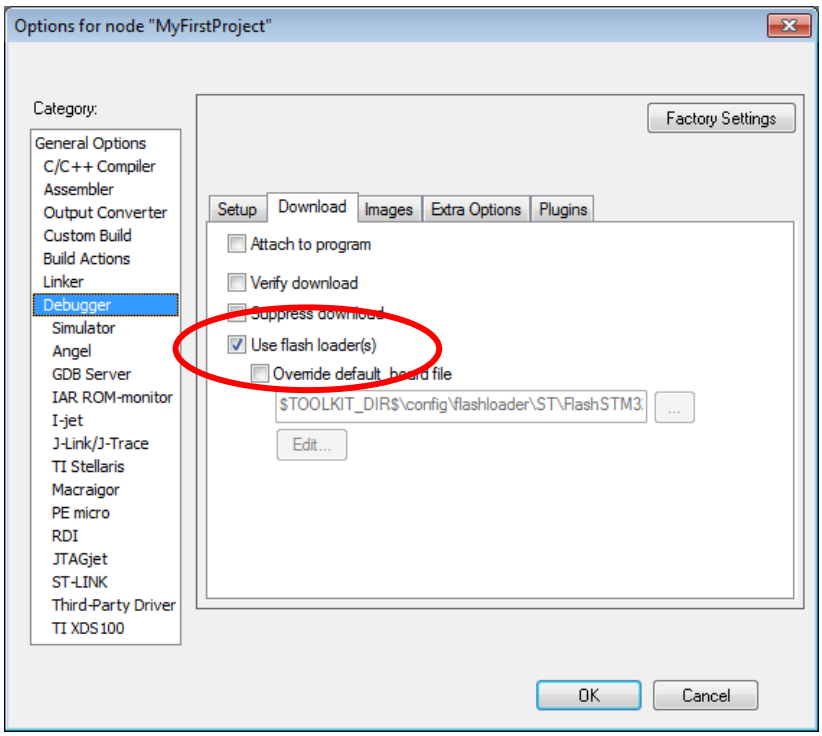

Figure 11: Use special protocol for loading the code into the flash memory

- The target microcontroller is connected using a SWD protocol, so state this in »Options«.

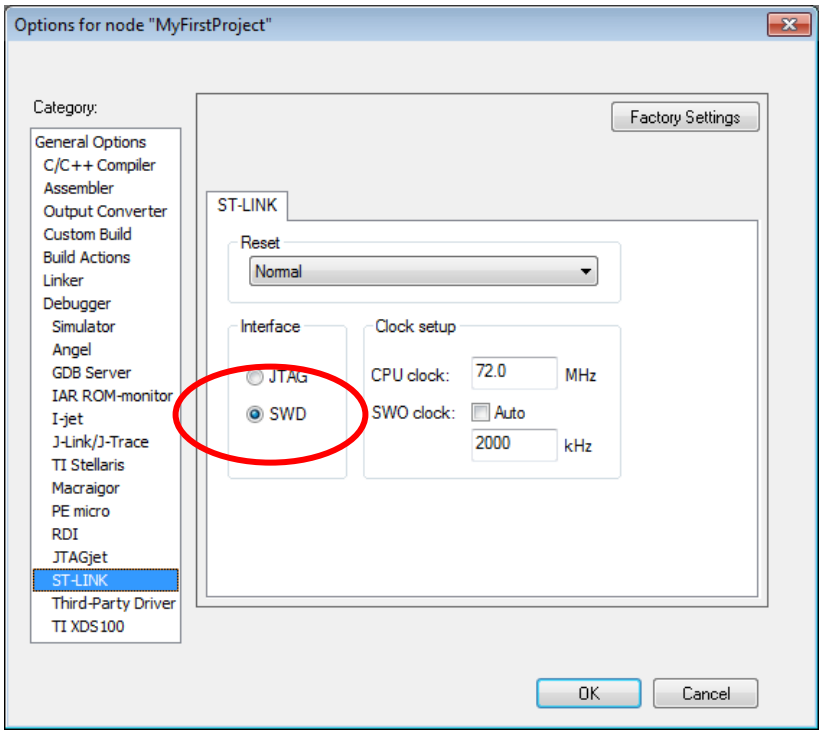

Figure 12: Use SWD as the preferred protocol for downloading the code

3.5. Compiling and debugging a project

Once the user program is written and saved, and all the above definitions are finished, one can compile and link the 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. 6, 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 the compiler or linker fined errors during their work, then 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. 10 must be different: the "Simulator" must be selected instead of "STR-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 hardware. This mode is very usable for testing algorithms, 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. The testing of hardware using specific values of variables might be more difficult than testing in simulator alone.

In both cases breakpoints in program execution are available. They 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.

The content of all registers is available for inspection and modification when the program is not running. 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 Embedded Workbench window with the content of all CPU registers. The user can select to inspect or modify other hardware by clicking at the top of this window ("Current CPU Registers") 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 13 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; most of values

are shown in hex notation, GPIOA_ODR is shown in decimal notation, GPIOA_AFRL is shown in binary notation, and GPIO_AFRH is expanded into individual bits).

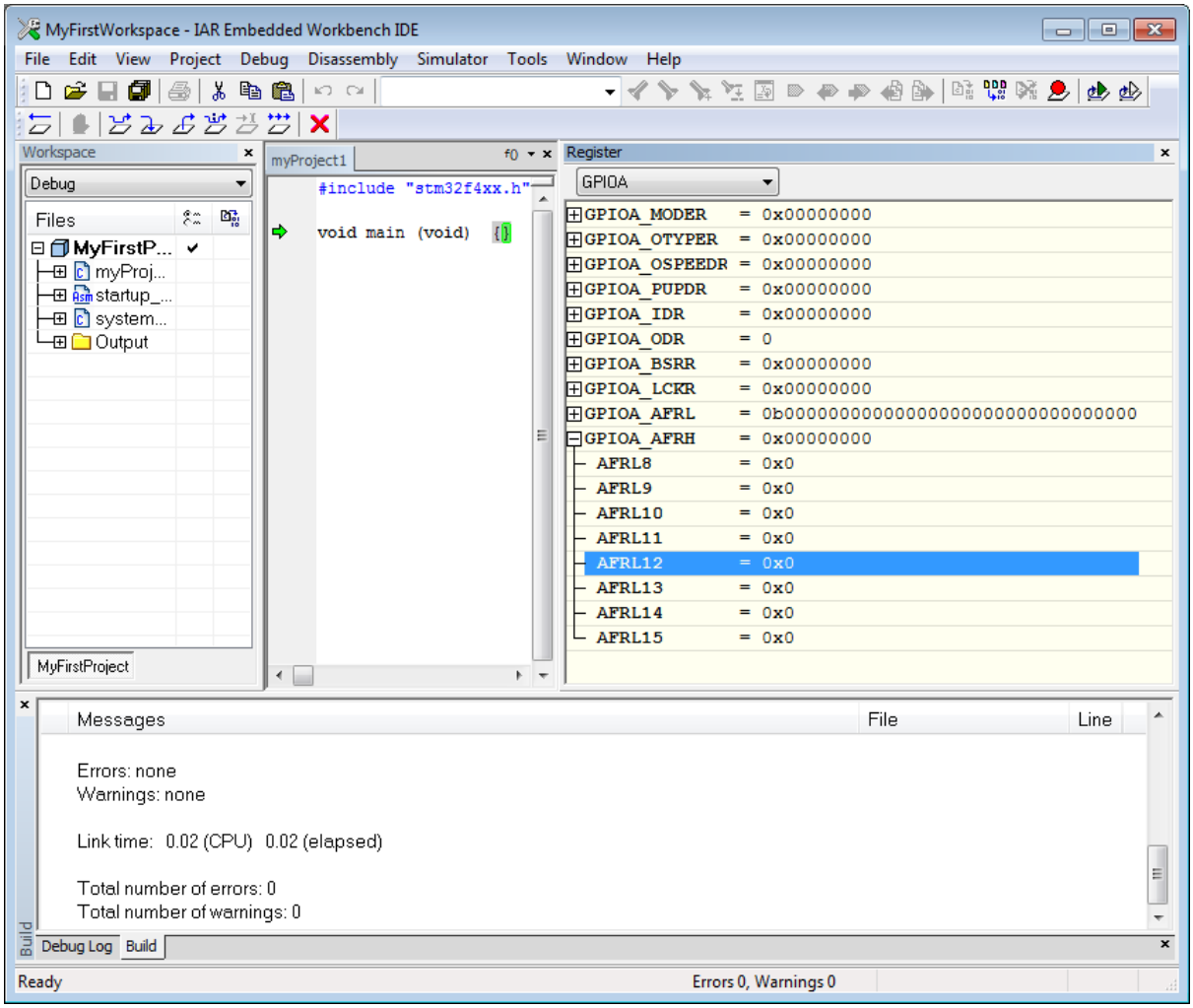

Figure 13: An example of a debug session in progress