UM1467
User manual

Getting started with software and firmware environments for the STM32F4DISCOVERY Kit

1 Introduction

This document describes the software, firmware environment and development recommendations required to build an application around the STM32F4DISCOVERY board.

It presents the firmware applications package provided within this board with details on its architecture and contents. It provides guidelines to novice users on how to build and run a sample application and allows them to create and build their own application.

This document is structured as follows:

- System requirements to use this board and how to run the built-in demonstration are provided in *Section 2: Getting started*.
- *Section 3* describes the firmware applications package.
- *Section 5* presents development toolchain installation and overview of ST-LINK/V2 interface.
- *Section 6*, *Section 7*, *Section 8*, and *Section 9* introduce how to use the following software development toolchains:
  - IAR Embedded Workbench® for ARM (EWARM) by IAR Systems
  - Microcontroller Development Kit for ARM (MDK-ARM) by Keil™
  - TrueSTUDIO® by Atollic
  - TASKING VX-toolset for ARM Cortex by Altium

Although this user manual cannot cover all the topics relevant to software development environments; it demonstrates the first basic steps necessary to get started with the compilers/debuggers.

Reference documents

- STM32F4DISCOVERY high-performance discovery board data brief
- STM32F4DISCOVERY peripherals firmware examples (AN3983)
- STM32F40x reference manual (RM0090)
- STM32F405xx STM32F407xx datasheet

The above documents are available at www.st.com/stm32f4-discovery.

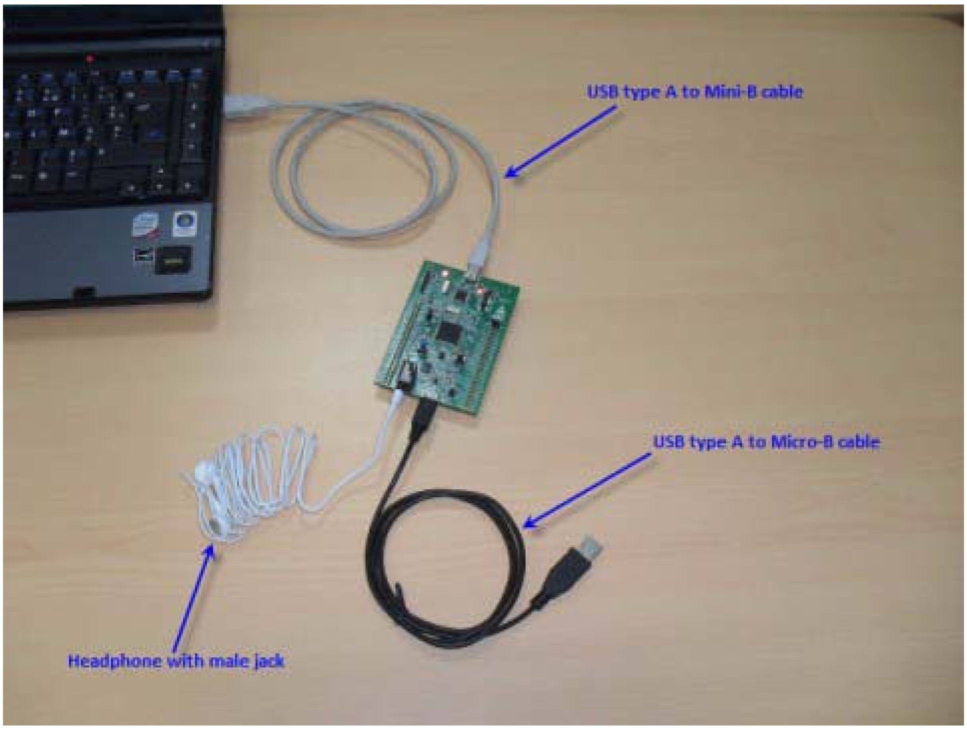
Contents

2 Getting started

2.1 System requirements

Before running your application, you should establish the connection with the STM32F4DISCOVERY board as following.

Figure 1. Hardware environment



To run and develop any firmware applications on your STM32F4DISCOVERY board, the minimum requirements are as follows:

- Windows PC (2000, XP, Vista, 7)
- 'USB type A to Mini-B' cable, used to power the board (through USB connector CN1) from host PC and connect to the embedded ST-LINK/V2 for debugging and programming

Additional hardware accessories will be needed to run some applications:

- 'USB type A to Micro-B' cable, used to connect the board (through USB connector CN5) as USB Device to host PC.
- Headphone with male jack connector.

2.2 Running the built-in demonstration

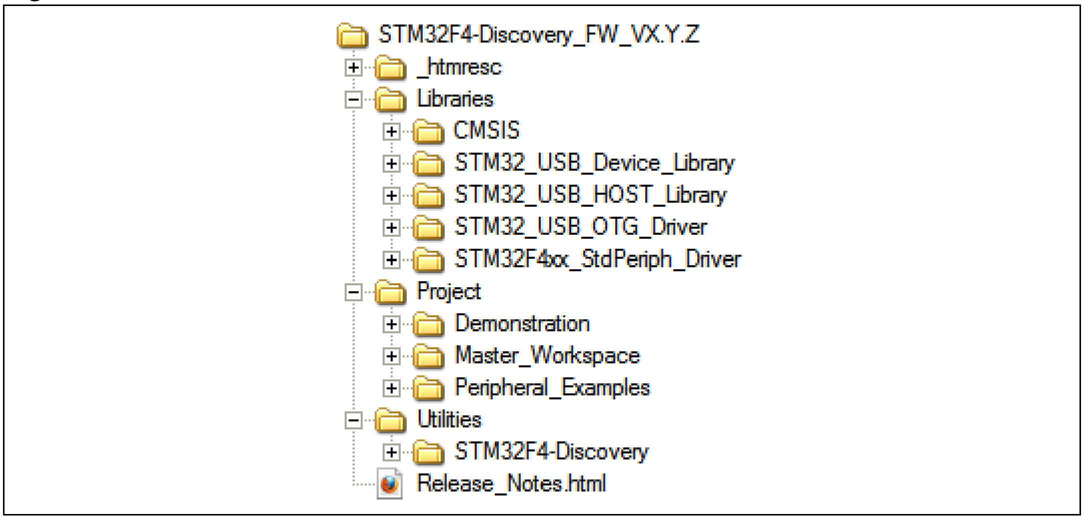
The board comes with the demonstration firmware preloaded in the Flash memory. Follow the steps below to run it:

- Check jumper position on the board, JP1 on, CN3 on (Discovery selected).
- Connect the STM32F4DISCOVERY board to a PC with a 'USB type A to Mini-B' cable through USB connector CN1 to power the board. Red LED LD2 (PWR) then lights up.
- Four LEDs between B1 and B2 are blinking.
- Press User Button B1 then MEMS sensor is enabled, move the board and observe the four LEDs blinking according to the motion direction and speed.
- If you connect a second 'USB type A to Micro-B' cable between PC and CN5 connector then the board is recognized as standard mouse and its motion will also control the PC cursor.

3 Description of the firmware package

The STM32F4DISCOVERY firmware applications are provided in one single package and supplied in one single zip file. The extraction of the zip file generates one folder, *STM32F4-Discovery_FW_VX.Y.Z*, which contains the following subfolders:

Figure 2. Hardware environment



1. VX.Y.Z refer to the package version, ex. V1.0.0

3.1 Libraries folder

This folder contains the Hardware Abstraction Layer (HAL) for STM32F4xx Devices.

3.1.1 CMSIS subfolder

This subfolder contains the STM32F4xx and Cortex-M4F CMSIS files.

Cortex-M4F CMSIS files consist of:

– *Core Peripheral Access Layer:* contains name definitions, address definitions and helper functions to access Cortex-M4F core registers and peripherals. It defines also a device independent interface for RTOS Kernels that includes debug channel definitions.

– *CMSIS DSP Software Library:* features a suite of common signal processing functions for use on Cortex-M processor based devices. The library is completely written in C and is fully CMSIS compliant. High performance is achieved through maximum use of Cortex-M4F intrinsics.

STM32F4xx CMSIS files consist of:

– *stm32f4xx.h:* this file contains the definitions of all peripheral registers, bits, and memory mapping for STM32F4xx devices. The file is the unique include file used in the application programmer C source code, usually in the main.c.

– *system_stm32f4xx.c/.h*: This file contains the system clock configuration for STM32F4xx devices. It exports `SystemInit()` function which sets up the system

clock source, PLL multiplier and divider factors, AHB/APBx prescalers and Flash settings. This function is called at startup just after reset and before connecting to the main program. The call is made inside the *startup_stm32f4xx.s* file.
– *startup_stm32f4xx.s*: Provides the Cortex-M4F startup code and interrupt vectors for all STM32F4xx device interrupt handlers.

3.1.2 STM32_USB_Device_Library subfolder

This subfolder contains USB Device Library Core and the class drivers.

The Core folder contains the USB Device library machines as defined by the revision 2.0 Universal Serial Bus Specification.

The Class folder contains all the files relative to the Device class implementation. It is compliant with the specification of the protocol built in these classes.

3.1.3 STM32_USB_HOST_Library subfolder

This subfolder contains USB Host Library Core and the class drivers.

The Core folder contains the USB Host library machines as defined by the revision 2.0 Universal Serial Bus Specification.

The Class folder contains all the files relative to the Host class implementation. It is compliant with the specification of the protocol built in these classes.

3.1.4 STM32_USB_OTG_Driver subfolder

This subfolder contains the low level drivers for STM32F4xx USB HS and FS cores. It provides an hardware abstraction layer, USB communication operations and interfaces used by the high level Host and Device Libraries to access the core.

3.1.5 STM32F4xx_StdPeriph_Driver subfolder

This subfolder contains sources of STM32F4xx peripheral drivers (excluding USB and Ethernet).

Each driver consists of a set of routines and data structures covering all peripheral functionalities. The development of each driver is driven by a common API (application programming interface) which standardizes the driver structure, the functions and the parameter names.

Each peripheral has a source code file, *stm32f4xx_ppp.c*, and a header file, *stm32f4xx_ppp.h*. The *stm32f4xx_ppp.c* file contains all the firmware functions required to use the PPP peripheral.

3.2 Project folder

This folder contains the source files of the STM32F4DISCOVERY firmware applications.

3.2.1 Demonstration subfolder

This subfolder contains the demonstration source files with preconfigured project for EWARM, MDK-ARM, TrueSTUDIO and TASKING toolchains.

A binary images (*.hex and *.dfu) of this demonstration is provided under Binary subfolder. You can use the STM32F4xx's embedded Bootloader or any in-system programming tool to reprogram the demonstration using this binary image.

3.2.2 Master_Workspace subfolder

This subfolder contains, for some toolchains, a multi-project workspace allowing you to manage all the available projects (provided under the subfolders listed below) from a single workspace window.

3.2.3 Peripheral_Examples subfolder

This subfolder contains a set of examples for some peripherals with preconfigured projects for EWARM, MDK-ARM, TrueSTUDIO and TASKING toolchains. See *Section 5* and *STM32F4DISCOVERY peripheral firmware examples,* AN3983, for further details.

3.3 Utilities folder

This folder contains the abstraction layer for the STM32F4DISCOVERY hardware. It provides the following drivers:

– *stm32f4_discovery.c*: provides functions to manage the user push button and 4 LEDs (LD3.LD6)
– *stm32f4_discovery_audio_codec.c/.h*: provides functions to manage the audio DAC (CS43L22)
– *stm32f4_discovery_lis302dl.c/.h*: provides functions to manage the MEMS accelerometer (LIS302DL).

4 Binary images for reprogramming firmware applications

This section describes how to use the provided binary images to reprogram the firmware applications. The STM32F4DISCOVERY firmware package contains binary images (*.hex and *.dfu) of the provided applications which allow to use the STM32F4xx's embedded Bootloader or any in-system programming tool to reprogram these applications easily.

Below are the steps to follow:

- Using “in-system programming tool”
  - Connect the STM32F4DISCOVERY board to a PC with a 'USB type A to Mini-B' cable through USB connector CN1 to power the board.
  - Make sure that the embedded ST-LINK/V2 is configured for in-system programming (both CN3 jumpers ON).
  - Use *.hex binary (for example, \Project\Demonstration\Binary\STM32F4-Discovery_Demonstration_V1.0.0.hex) with your preferred in-system programming tool to reprogram the demonstration firmware (ex. STM32 ST-LINK Utility, available for download from www.st.com).
- Using “Bootloader (USB FS Device in DFU mode)”
  - Configure the STM32F4DISCOVERY board to boot from “System Memory” (boot pins BOOT0:1 / BOOT1:0)
  - Set BOOT0 pin to high level: on the male header P2 place a jumper between BOOT0 pin and VDD pin
  - Set BOOT1(PB2) pin to low level: on the male header P1 place a jumper between PB2 pin and GND pin
  - Connect a 'USB type A to Mini-B' cable between PC and USB connector CN1 to power the board.
  - Connect a 'USB type A to Micro-B' cable between PC and USB connector CN5, the board will be detected as USB device.
  - Use *.dfu binary (for example, \Project\Demonstration\Binary\STM32F4-Discovery_Demonstration_V1.0.0.dfu) with “DFUse\DFUse Demonstration" tool (available for download from www.st.com) to reprogram the demonstration firmware.

5 ST-LINK/V2 installation and development

STM32F4DISCOVERY board includes an ST-LINK/V2 embedded debug tool interface that is supported by the following software toolchains:

- IAR™ Embedded Workbench for ARM **(EWARM)** available from **www.iar.com**

  The toolchain is installed by default in the *C:\Program Files\IAR Systems\Embedded Workbench 6.2* directory on the PC's local hard disk.

  After installing EWARM, install the ST-LINK/V2 driver by running the *ST-Link_V2_USB.exe* from *[IAR_INSTALL_DIRECTORY]\Embedded Workbench 6.2\arm\drivers\ST-Link \ST-Link_V2_USBdriver.exe*
- RealView Microcontroller Development Kit **(MDK-ARM)** toolchain available from **www.keil.com**

  The toolchain is installed by default in the *C:\Keil* directory on the PC's local hard disk; the installer creates a start menu µVision4 shortcut.

  When connecting the ST-LINK/V2 tool, the PC detects new hardware and asks to install the ST-LINK_V2_USB driver. The "Found New Hardware wizard" appears and guides you through the steps needed to install the driver from the recommended location.
- Atollic **TrueSTUDIO®** STM32 available from **www.atollic.com**

  The toolchain is installed by default in the *C:\Program Files\Atollic* directory on the PC's local hard disk.

  The *ST-Link_V2_USB.exe* is installed automatically when installing the software toolchain.
- Altium™ **TASKING** VX-toolset for ARM® Cortex-M available from **www.tasking.com**

  The toolchain is installed by default in the "*C:\Program Files\TASKING* directory on the PC's local hard disk. The *ST-Link_V2_USB.exe* is installed automatically when installing the software toolchain.

Note: *The embedded ST-LINK/V2 supports only SWD interface for STM32 devices.*

Refer to the firmware package release notes for the version of the supporting development toolchains.

6 Using IAR Embedded Workbench® for ARM

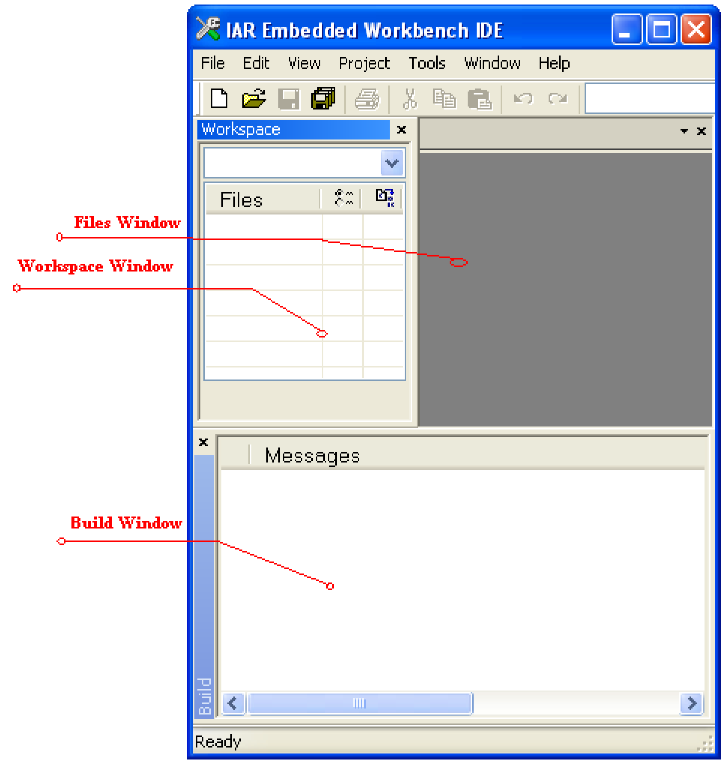
6.1 Building an existing EWARM project

The following is the procedure for building an existing EWARM project.

1. Open the IAR Embedded Workbench® for ARM (EWARM).

   Figure 3 shows the basic names of the windows referred to in this document.

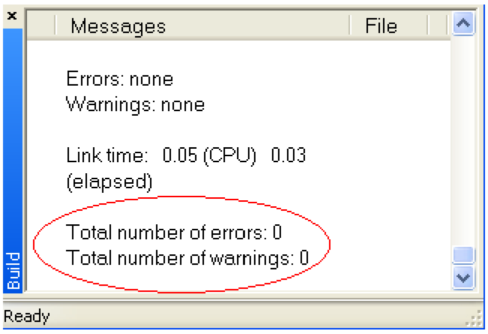
Figure 3. IAR Embedded Workbench IDE (Integrated Design Environment)



2. In the **File** menu, select **Open** and click **Workspace** to display the Open Workspace dialog box. Browse to select the *demonstration* workspace file and click **Open** to launch it in the Project window.
3. In the **Project** menu, select **Rebuild All** to compile your project.

4. If your project is successfully compiled, the following window in *Figure 4* is displayed.

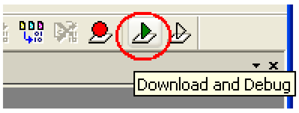
Figure 4. EWARM project successfully compiled



6.2 Debugging and running your EWARM project

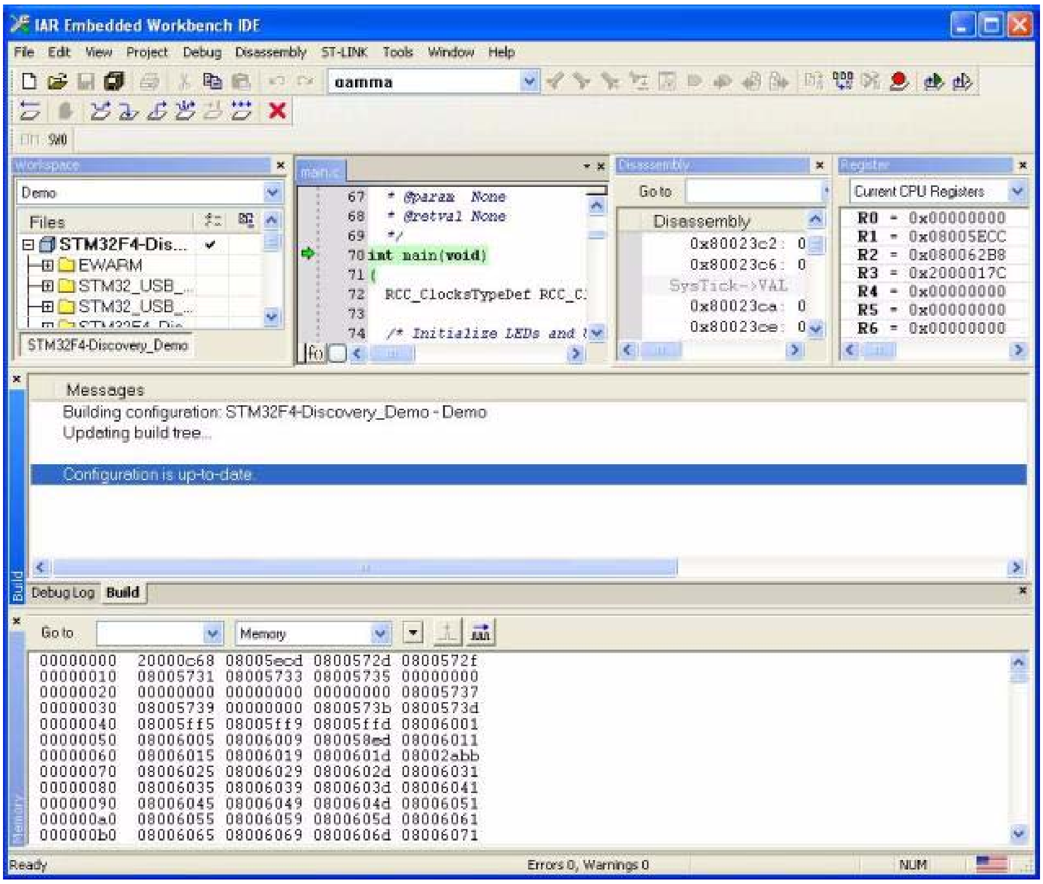
In the IAR Embedded Workbench IDE, from the **Project menu**, select **Download and Debug** or, alternatively, click the **Download and Debug** button the in toolbar, to program the Flash memory and begin debugging.

Figure 5. Download and Debug button



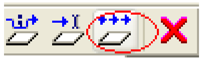
The debugger in the IAR Embedded Workbench can be used to debug source code at C and assembly levels, set breakpoints, monitor individual variables and watch events during the code execution.

Figure 6. IAR Embedded Workbench debugger screen



To run your application, from the **Debug** menu, select **Go**. Alternatively, click the **Go** button in the toolbar to run your application.

Figure 7. Go button



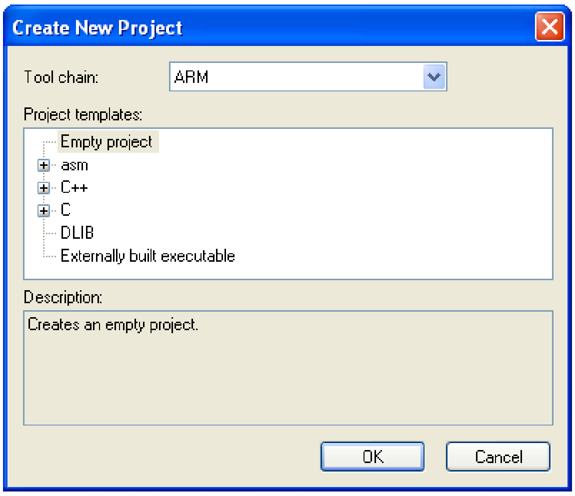
6.3 Creating your first application using the EWARM toolchain

6.3.1 Managing source files

Follow these steps to manage source files.

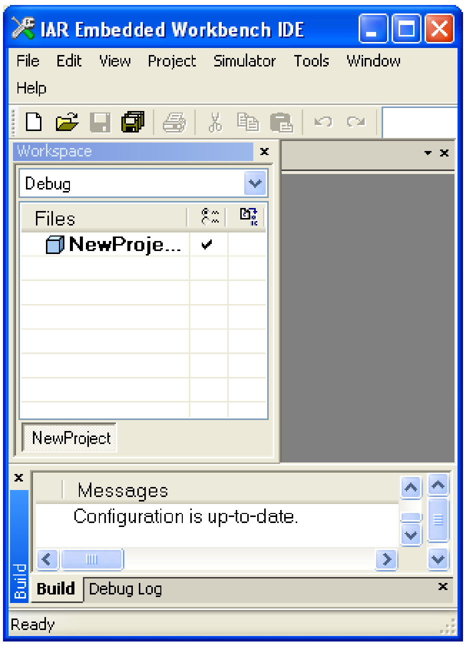
1. In the **Project** menu, select **Create New Project** and click **OK** to save your settings.

Figure 8. Create New Project dialog box



2. Name the project (for example, *NewProject.ewp*) and click **Save** to display the IDE interface.

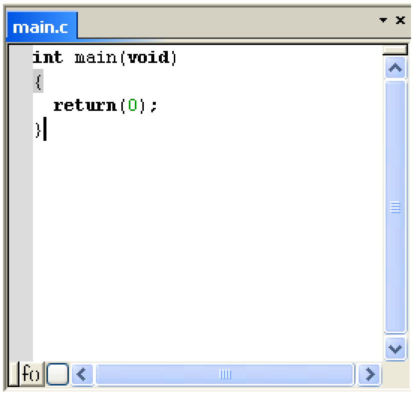
Figure 9. IDE interface



To create a new source file, in the **File menu**, open **New** and select **File** to open an empty editor window where you can enter your source code.

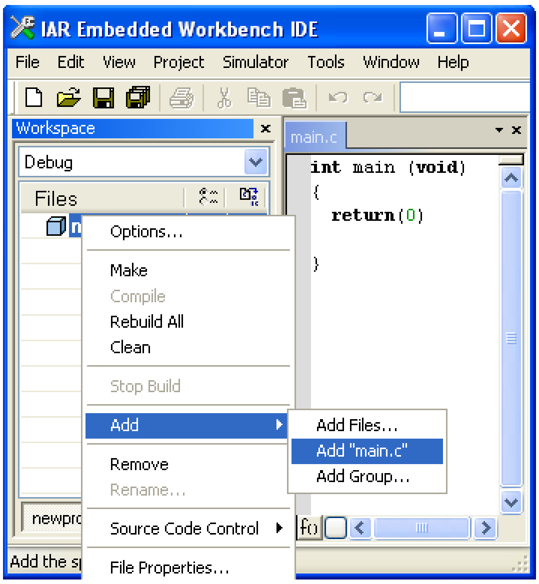
The IAR Embedded Workbench enables C color syntax highlighting when you save your file using the dialog **File > Save As…** under a filename with the ***.c** extension. In *Figure 10: main.c example file*, the file is saved as **main.c**.

Figure 10. main.c example file



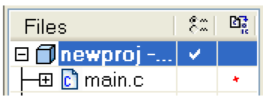
Once you have created your source file you can add this file to your project, by opening the **Project** menu, selecting **Add** and adding the selected file as in *Figure 11: Adding files to a project*.

Figure 11. Adding files to a project



If the file is added successfully, *Figure 12: New project file tree structure* is displayed.

Figure 12. New project file tree structure

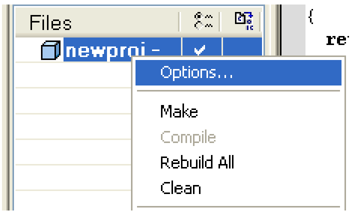


6.3.2 Configuring project options

Follow these steps to configure project options.

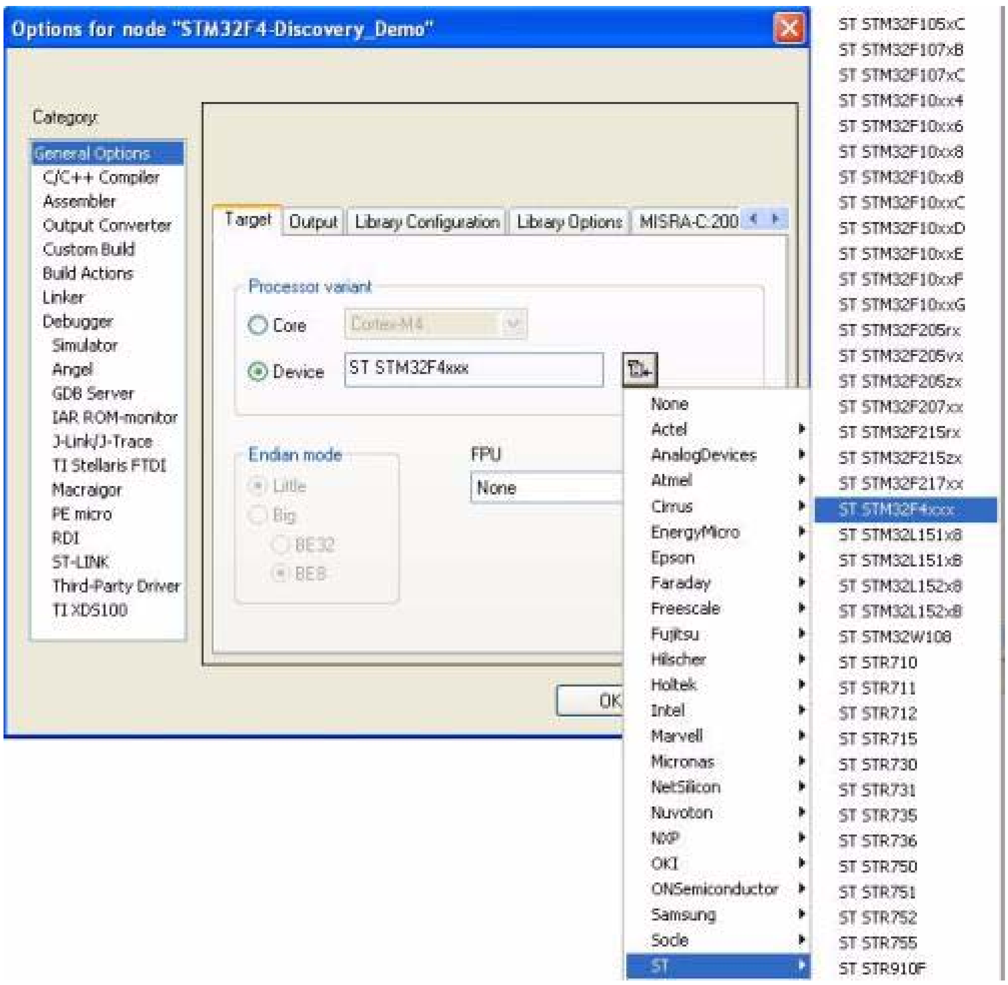
1. In the Project Editor, right-click on the project name and select **Options...** to display the Options dialog box as in *Figure 13*.

Figure 13. Configuring project options



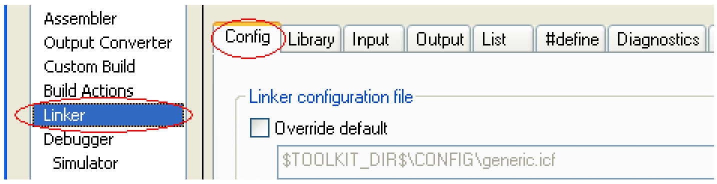
2. In the Options dialog box, select the **General Options** category, open the **Target** tab and select **Device - ST -STM32F4xx.**

Figure 14. General options > Target tab



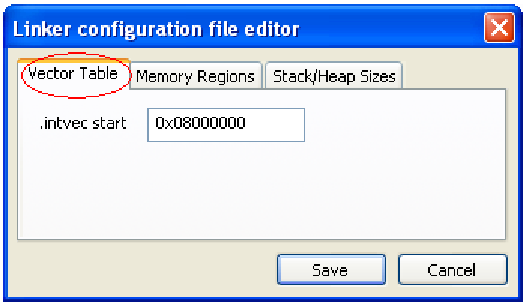
3. Select the **Linker** category, open the **Config** tab, in the **Linker configuration file** pane select **Override default** and click **Edit** to display the Linker configuration file editor.

Figure 15. Linker > Config tab



4. In the **Linker configuration file editor** dialog box, open the **Vector Table** tab and set the **.intvec.start** variable to `0x08000000`.

Figure 16. Linker configuration file editor dialog box > Vector Table tab



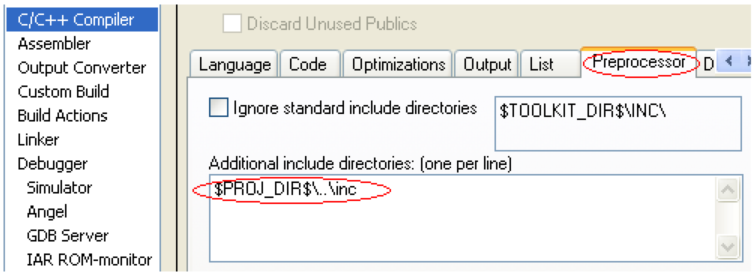
5. Open the **Memory Regions** tab, and enter the variables as shown in *Figure 17*.

Figure 17. Linker configuration file editor dialog box > Memory Regions tab

6. Click **Save** to save the linker settings automatically in the Project directory.

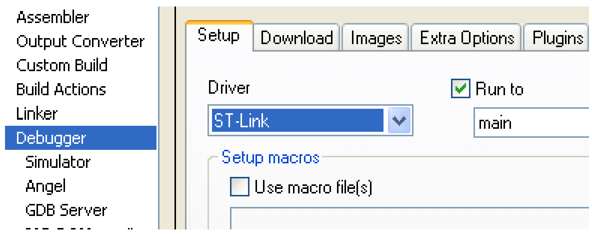
7. If your source files include header files, select the **C/C++ Compiler** category, open the **Preprocessor** tab, and specify their paths as shown in *Figure 18*. The path of the *include* directory is a relative path, and always starts with the project directory location referenced by `$PROJ_DIR$`

Figure 18. C/C++ Compiler > Preprocessor tab



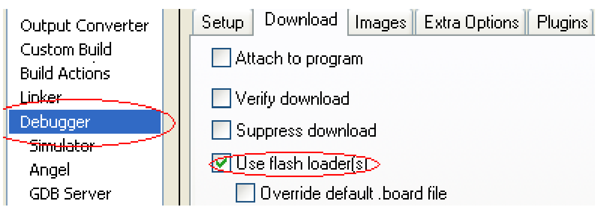
8. To set up the ST-Link embedded debug tool interface, select the **Debugger** category, open the **Setup tab** and from the drop-down **Driver** menu, select **ST-Link** as shown in *Figure 19*.

Figure 19. Debugger > Setup tab



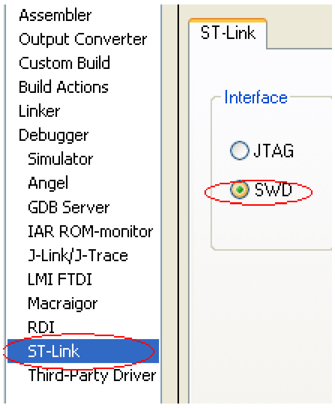
9. Open the **Debugger** tab and select **Use flash loader(s)** as shown in *Figure 20*.

Figure 20. Select Flash loaders



10. Select the **ST-Link** category, open the **ST-Link** tab and select **SWD** as the connection protocol as shown in *Figure 21*.

Figure 21. ST-Link communication protocol



11. Click **OK** to save the project settings.
12. To build your project, follow the instructions given in *Section 6.1: Building an existing EWARM project on page 11*.
13. Before running your application, establish the connection with the STM32F4DISCOVERY board as described in *Section 2: Getting started*.
14. To program the Flash memory and begin debugging, follow the instructions given in *Section 6.2: Debugging and running your EWARM project on page 12*.

7 Using MDK-ARM Microcontroller Development Kit by Keil™

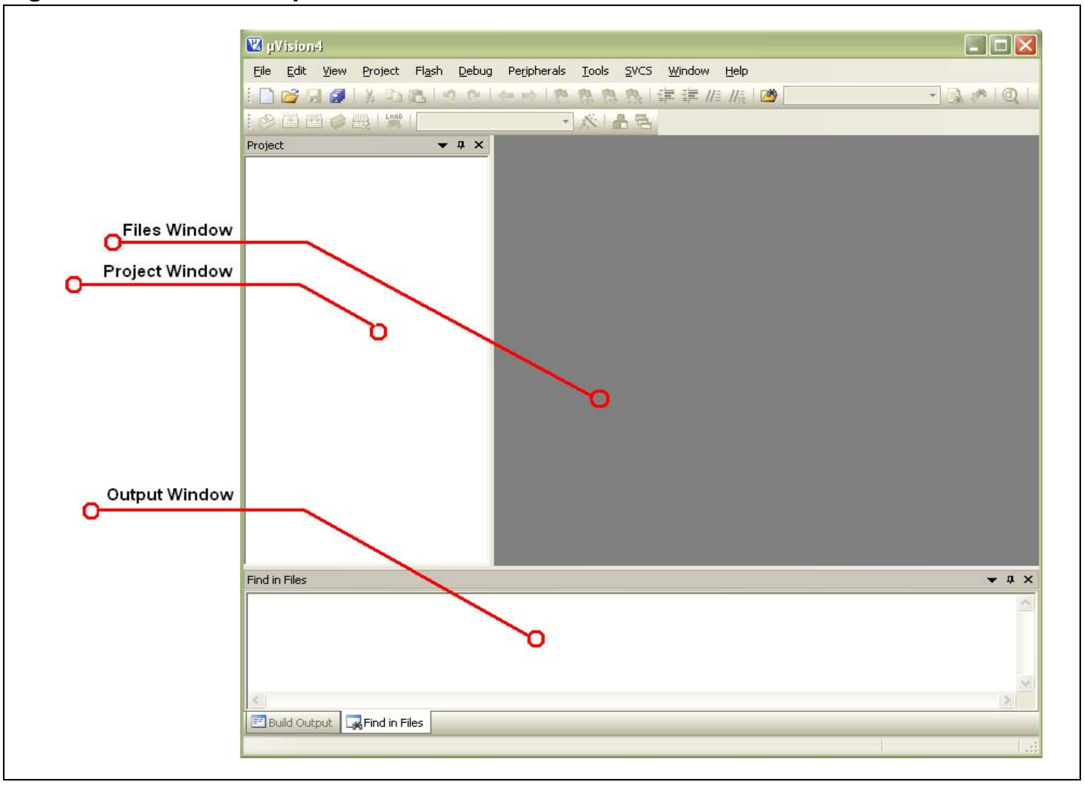
7.1 Building an existing MDK-ARM project

Follow these steps to build an existing MDK-ARM project.

1. Open the MDK-ARM µVision4 IDE, debugger, and simulation environment.

   Figure 22: MDK-ARM µVision4 IDE environment shows the basic names of the windows referred to in this section.

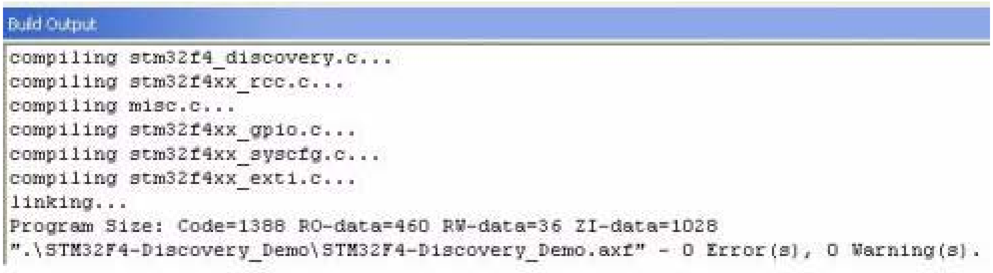
Figure 22. MDK-ARM µVision4 IDE environment



2. In the **Project** menu, select **Open Project...** to display the Select Project File dialog box. Browse to select the *STM32F4-Discovery.uvproj* project file and click **Open** to launch it in the Project window.
3. In the **Project** menu, select **Rebuild all target files** to compile your project.

4. If your project is successfully compiled, the following **Build Output** window (*Figure 23: Build Output - MDK-ARM µVision4 project successfully compiled*) is displayed.

Figure 23. Build Output - MDK-ARM µVision4 project successfully compiled



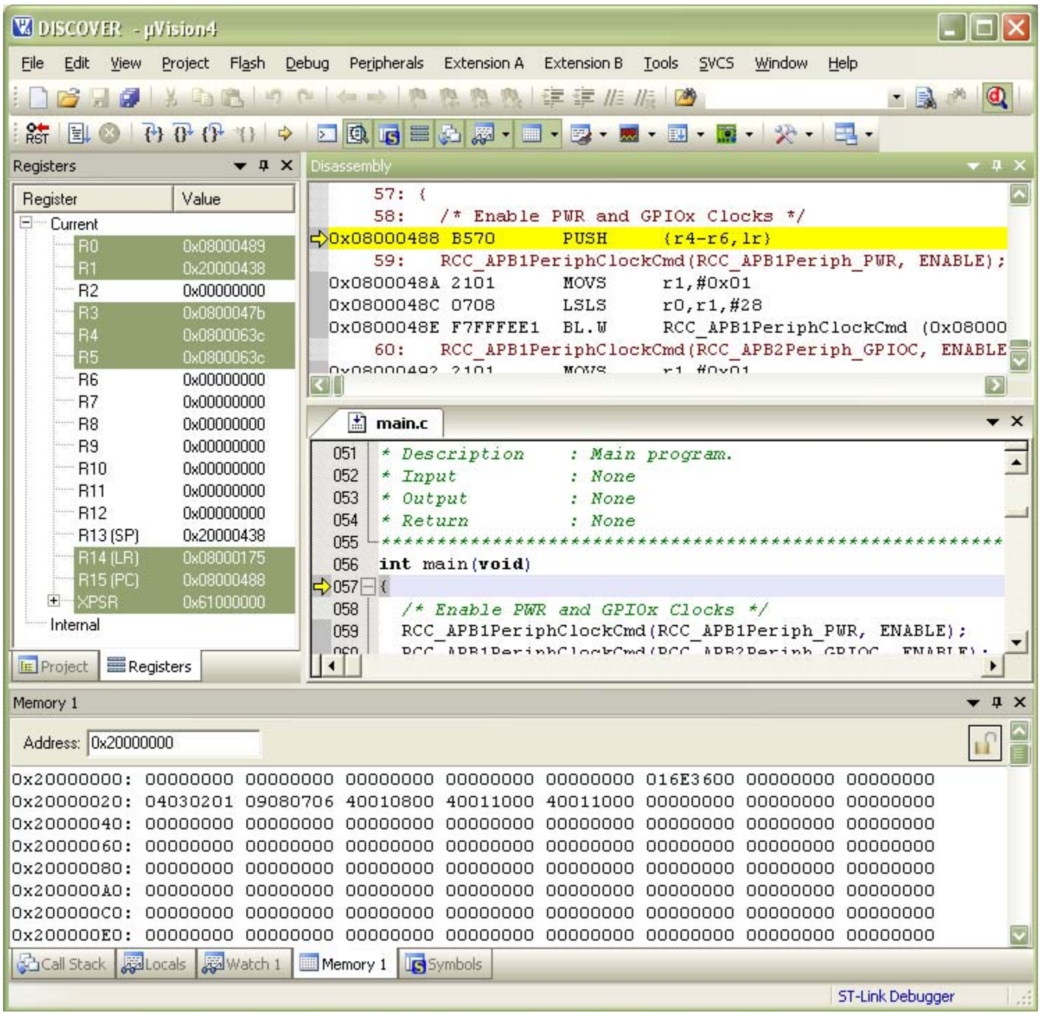
7.2 Debugging and running your MDK-ARM project

In the MDK-ARM µVision4 IDE, click the magnifying glass to program the Flash memory and begin debugging as shown below in *Figure 24*.

Figure 24. Starting a MDK-ARM µVision4 debugging session

The debugger in the MDK-ARM IDE can be used to debug source code at C and assembly levels, set breakpoints, monitor individual variables and watch events during the code execution as shown below in *Figure 25*.

Figure 25. MDK-ARM IDE workspace



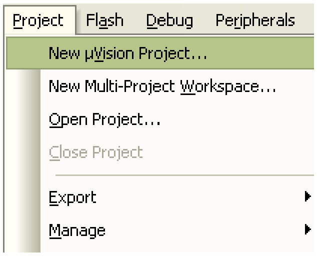
7.3 Creating your first application using the MDK-ARM toolchain

7.3.1 Managing source files

Follow these steps to manage source files.

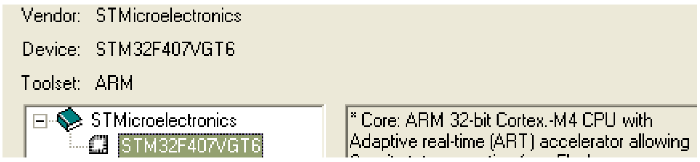
1. In the **Project** menu, select **New µVision Project...** to display the Create Project File dialog box. Name the new project and click **Save**.

Figure 26. Creating a new project



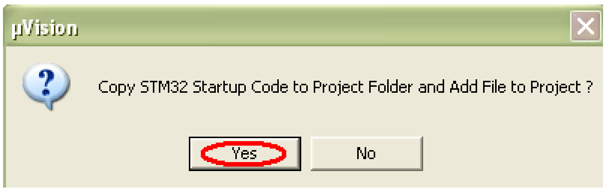
2. When a new project is saved, the IDE displays the *Device selection dialog box*. Select the device used for testing. In this example, we will use the STMicroelectronics device mounted on the STM32F4DISCOVERY board. In this case, double-click on **STMicroelectronics**, select the **STM32F407VGT6** device and click **OK** to save your settings.

Figure 27. Device selection dialog box



3. Click **Yes** to copy the STM32 Startup Code to the project folder and add the file to the project as shown in *Figure 28*.

Figure 28. Copy the STM32 Startup Code dialog box

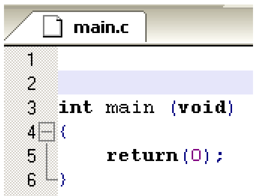


Note: The default STM32 startup file includes the SystemInit function. You can either comment out this file to not use it or add the system_stm32f4xx.c file from the STM32f4xx firmware library.

To create a new source file, in the **File menu**, select **New** to open an empty editor window where you can enter your source code.

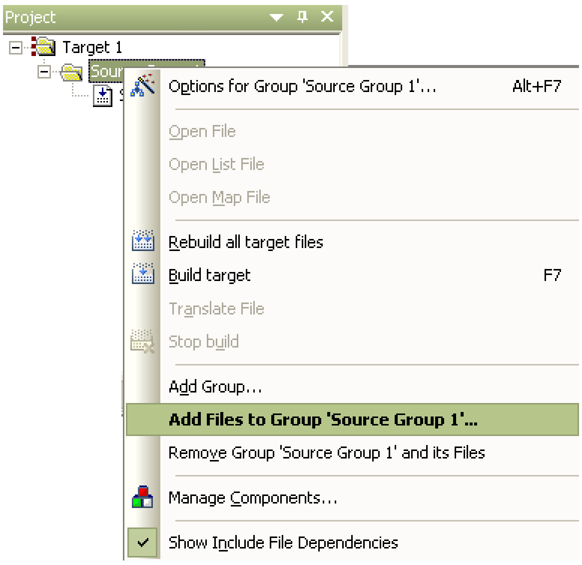
The MDK-ARM toolchain enables C color syntax highlighting when you save your file using the dialog **File > Save As…** under a filename with the ***.c** extension. In this example (*Figure 29*), the file is saved as **main.c**.

Figure 29. main.c example file



MDK-ARM offers several ways to add source files to a project. For example, you can select the file group in the **Project Window > Files** page and right-click to open a contextual menu. Select the **Add Files...** option, and browse to select the *main.c* file previously created.

Figure 30. Adding source files



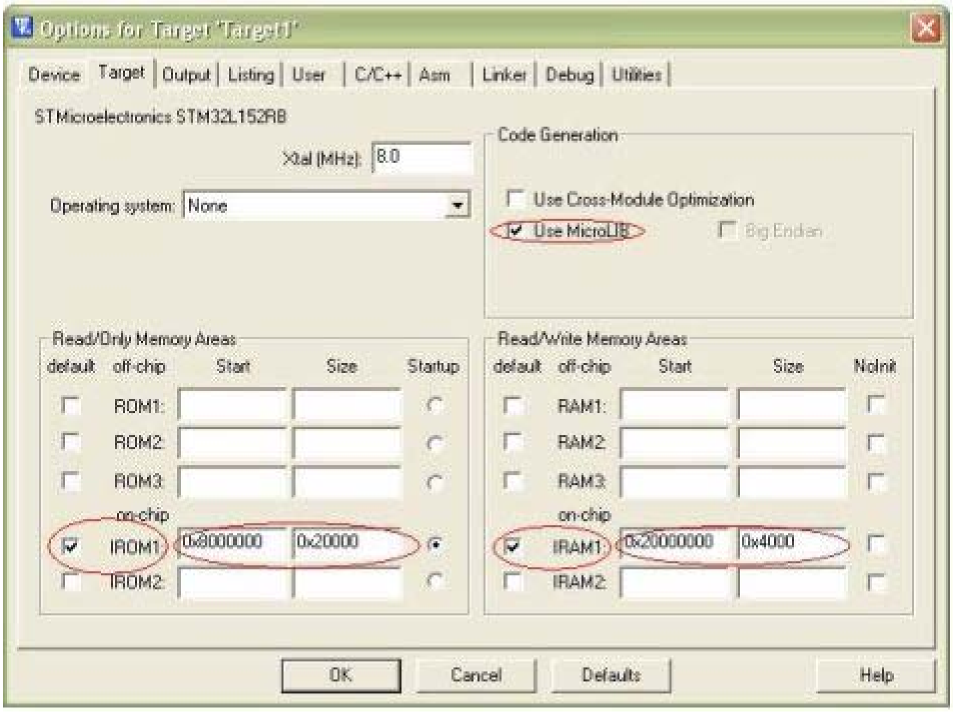
If the file is added successfully, the following window is displayed.

Figure 31. New project file tree structure

7.3.2 Configuring project options

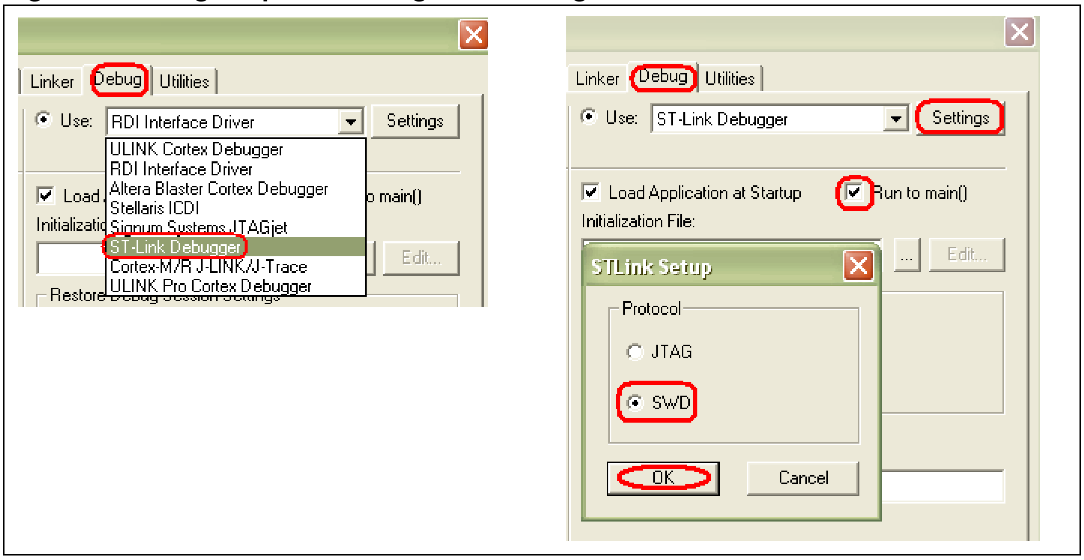
1. In the **Project** menu, select **Options for Target 1** to display the Target Options dialog box.
2. Open the Target tab and enter IROM1 and IARM1 start and size settings as shown in *Figure 32*.

Figure 32. Target Options dialog box - Target tab



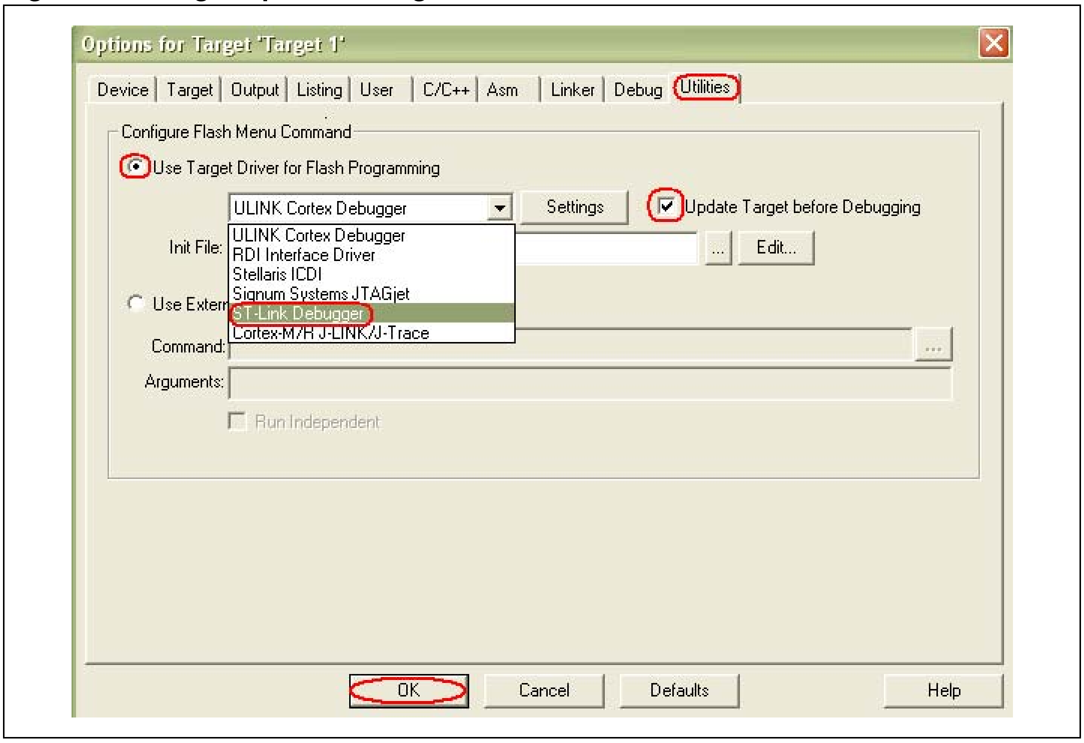
3. Open the **Debug** tab, click **Use** and select the **ST-Link Debugger**. Then, click **Settings** and select the **SWD** protocol. Click **OK** to save the ST-Link setup settings.
4. Select **Run to main()**.

Figure 33. Target Options dialog box - Debug tab



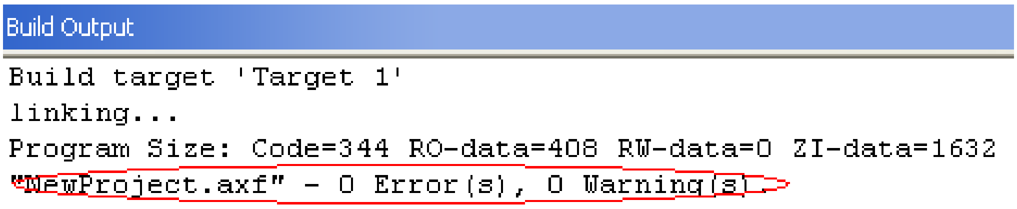
5. Open the **Utilities** tab, select **Use Target Driver for Flash Programming** and select the **ST-Link Debugger** from the drop-down menu.
6. Verify that the **Update Target before Debugging option** is selected.
7. Click **OK** to save your settings.

Figure 34. Target Options dialog box - Utilities tab



8. In the **Project** menu, select **Build Target**.
9. If your project is successfully built, the following window is displayed.

Figure 35. MDK-ARM µVision4 project successfully built



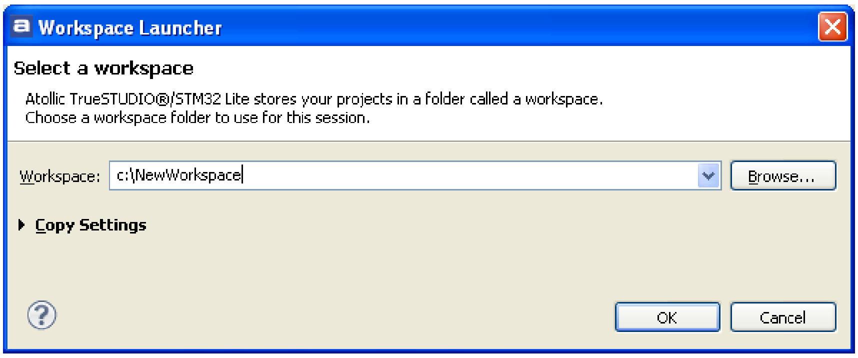
10. Before running your application, establish the connection with the STM32F4DISCOVERY board as described in *Section 2: Getting started*.
11. To program the Flash memory and begin debugging, follow the instructions given in *Section 6.2: Debugging and running your EWARM project on page 12*.

8 Using Atollic TrueSTUDIO®

8.1 Building an existing TrueSTUDIO project

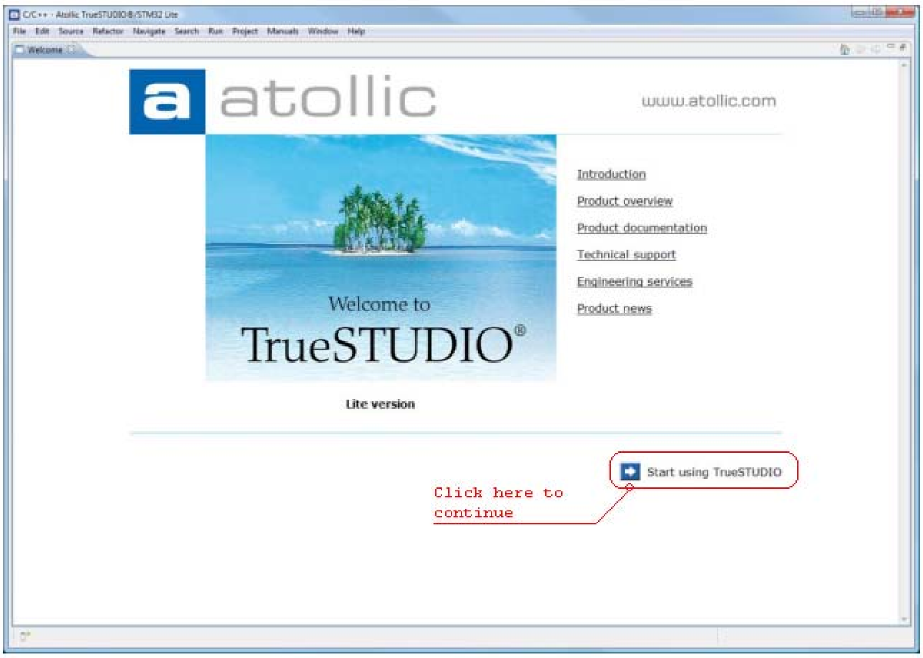
1. Open the **TrueSTUDIO®/STM32** product folder and select the **Atollic TrueSTUDIO® STM32** product name. The program launches and asks for the Workspace location.

Figure 36. TrueSTUDIO workspace launcher dialog box



2. Browse to select the STM32F4DISCOVERY Demonstration TrueSTUDIO workspace and click **OK** to save your settings and to display the Welcome screen. To start using Atollic TrueSTUDIO®, click **Start using TrueSTUDIO**.

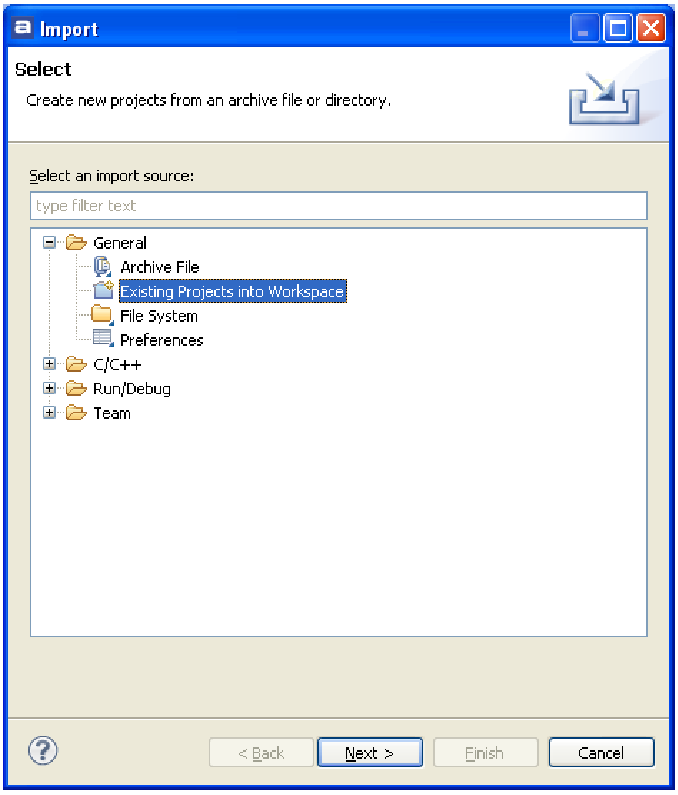
Figure 37. Atollic TrueSTUDIO®/STM32 Lite welcome screen



3. The TrueSTUDIO Discovery workspace contains a demo project for the STM32F4DISCOVERY kit. To load this project, in the File menu, select **Import...** to display the **Import** dialog box.

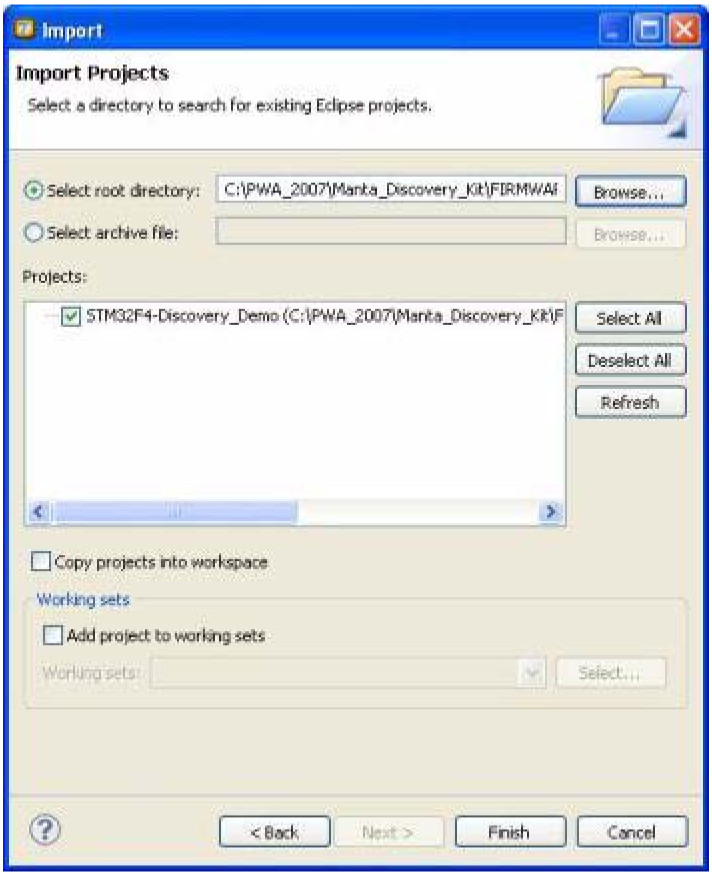
4. In the **Import** window, open **General**, select **Existing Projects into Workspace** and click **Next**.

Figure 38. Atollic TrueSTUDIO®/STM32 Lite import source select dialog box



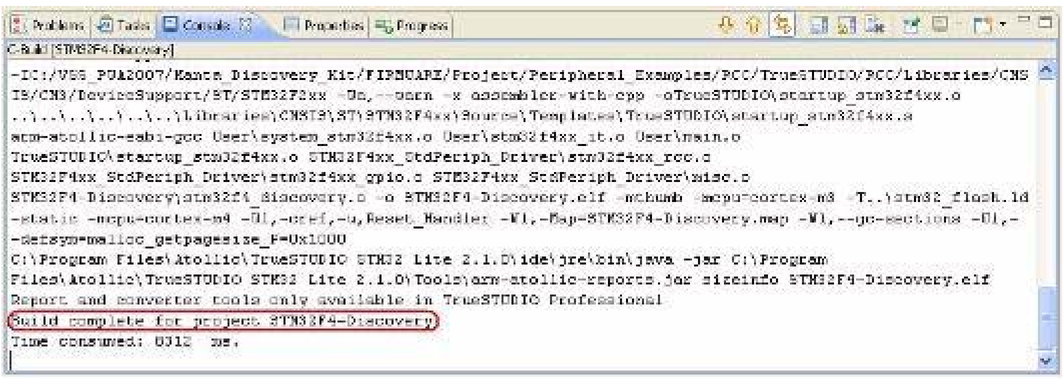
5. Click **Select root directory**, browse to the TrueSTUDIO workspace folder and select the **STM32F4-Discovery** project.

Figure 39. Atollic TrueSTUDIO®/STM32 Lite import projects dialog box



6. In the **Projects** pane, select the **STM32F4-Discovery** and click **Finish**.
7. In the **Project Explorer**, select the **STM32F4-Discovery project**. Open the **Project** menu, and click **Build Project**.
8. If your project is successfully compiled, the following window is displayed.

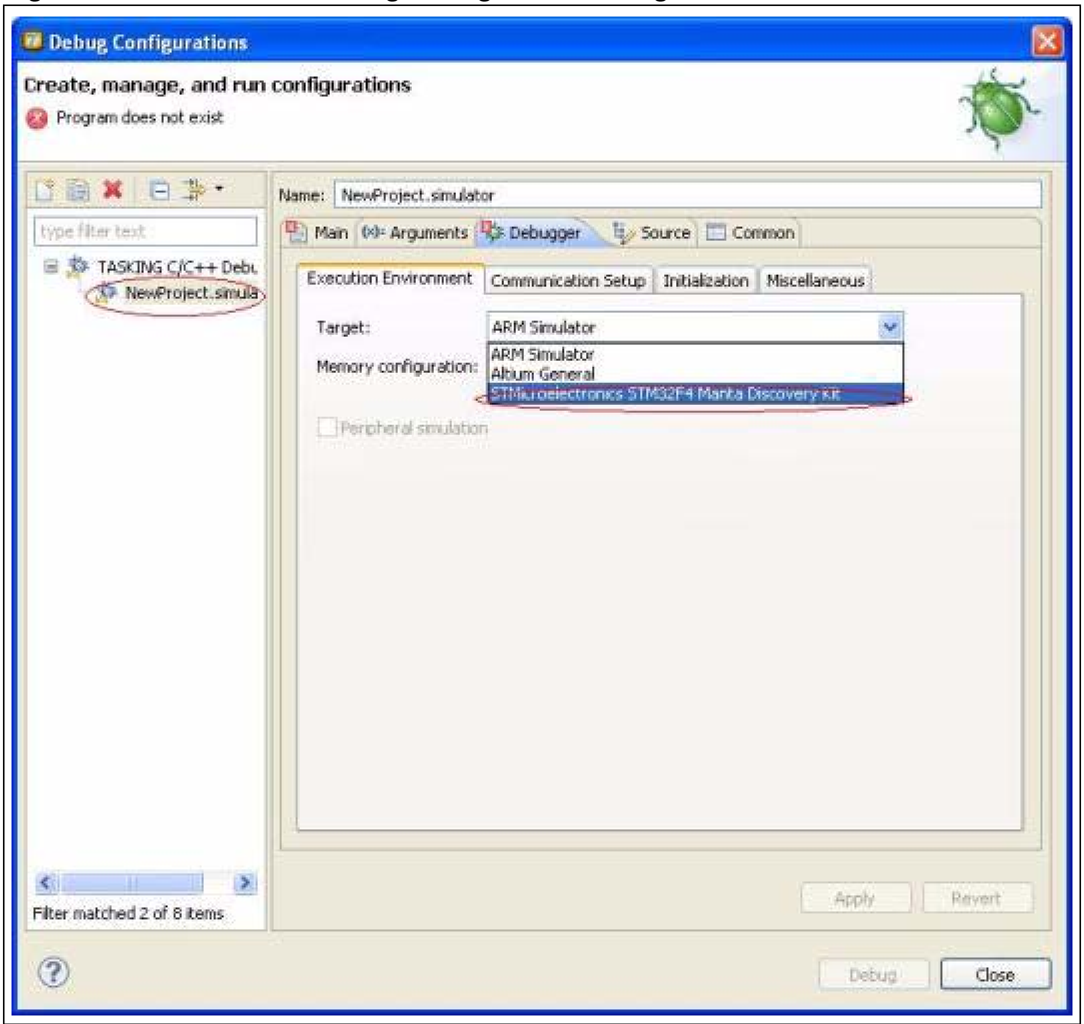
Figure 40. TrueSTUDIO® project successfully compiled



8.2 Debugging and running your TrueSTUDIO project

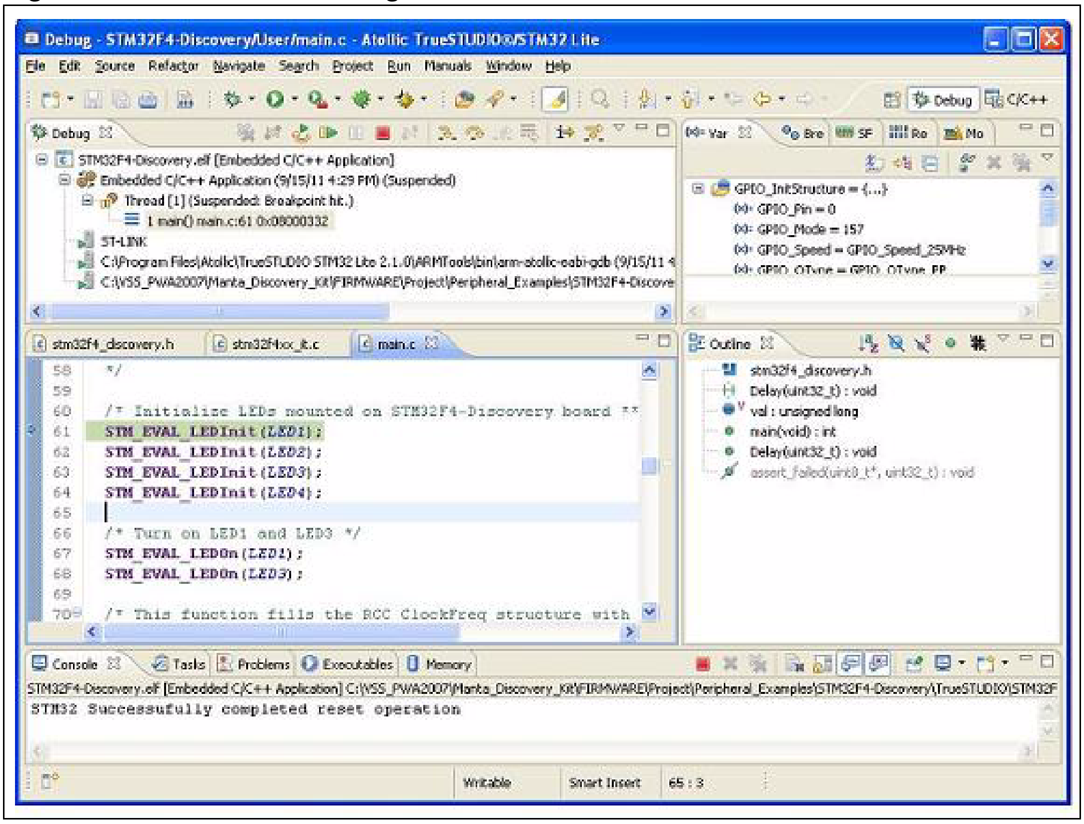
In the **Project Explorer**, select the **STM32F4-Discovery project** and press **F11** to display the **Debug Configuration** dialog box.

Figure 41. TrueSTUDIO Debug Configuration dialog box



9. In the **Main** tab, configure the project as shown in *Figure 41* and click **OK** to save your settings and to program the Flash memory and begin debugging.

Figure 42. TrueSTUDIO Debug window



The debugger in the Atollic TrueSTUDIO can be used to debug source code at C and assembly levels, set breakpoints, monitor individual variables and watch events during the code execution.

To run your application, from the **Run** menu, select **Resume**, or alternatively click the **Resume** button in the toolbar.

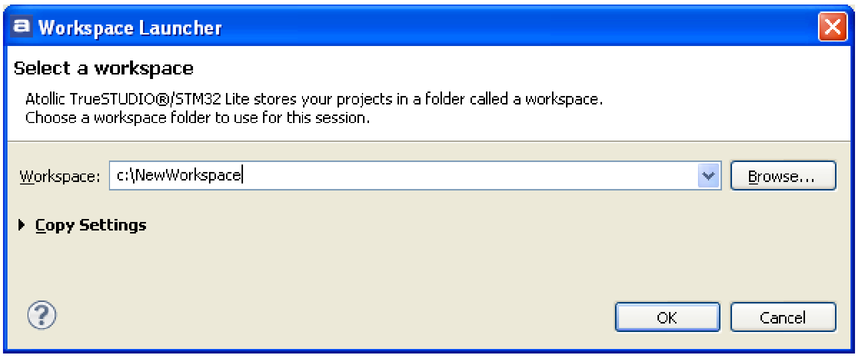
8.3 Creating your first application using TrueSTUDIO toolchain

TrueSTUDIO includes a dedicated connection to the STM32F4DISCOVERY board. When choosing this connection, all required files (startup file, firmware library, etc.) are added to the workspace and sample files are generated in the project folder to simplify development. The debug settings are automatically configured by selecting STM32F4DISCOVERY as the evaluation board.

Follow these steps to create your first application using TrueSTUDIO toolchain.

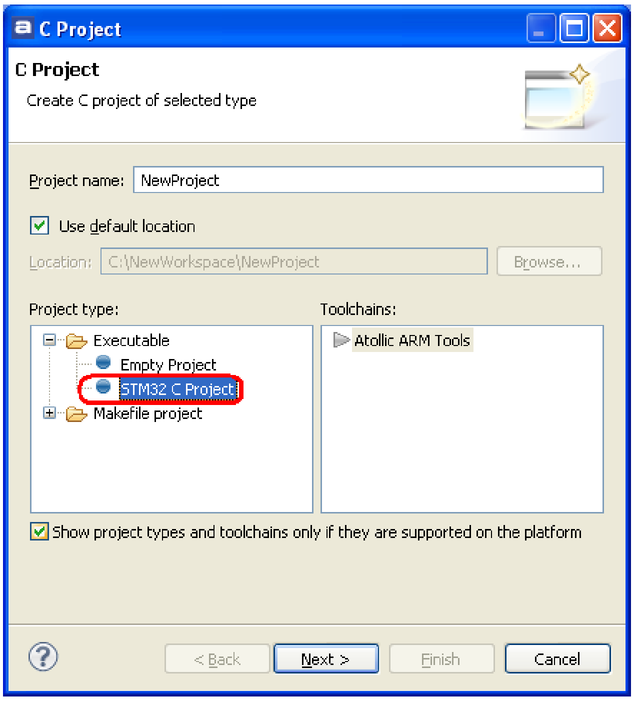
1. Open the **TrueSTUDIO®/STM32** product folder and select the **Atollic TrueSTUDIO® STM32** product name. The program launches and asks for the Workspace location. Browse to select an existing workspace, or enter a new workspace location and click **OK** to confirm.

Figure 43. TrueSTUDIO workspace launcher dialog box



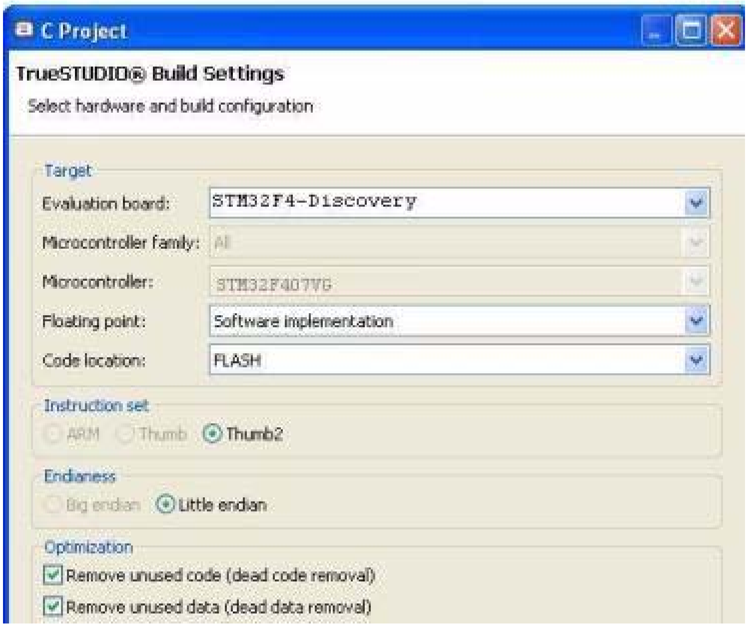
2. When the Atollic TrueSTUDIO® displays its Welcome window, click **Start using TrueSTUDIO** to open the main window. In the **File** menu, select **New** and click **C Project**.
3. Name the new project, in the **Project** type pane select **STM32 C Project** and click **Next**.

Figure 44. TrueSTUDIO® C Project dialog box



4. In the **TrueSTUDIO® Build Settings** dialog box, select **STM32F4-Discovery** as **Evaluation board**, configure the other settings as shown in *Figure 45* and click **Next**.

Figure 45. TrueSTUDIO® Build Settings dialog box

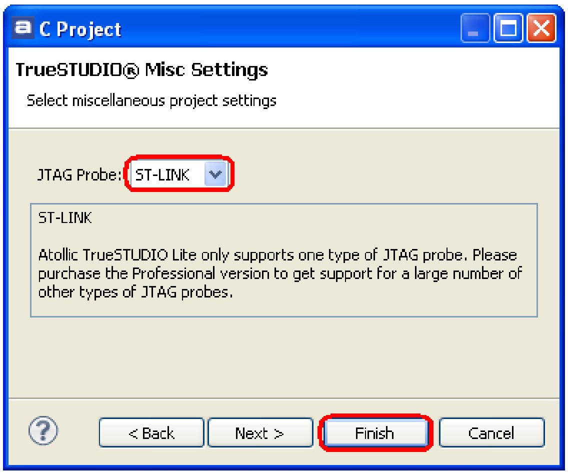


Note: *Choosing STM32F4DISCOVERY as the evaluation board, will configure the project as follows:*

- *Microcontroller: STM32F407VGT6*
- *Debug probe: ST-LINK*
- *Connection: Serial Wire Debug (SWD).*

5. Verify that the **JTAG Probe** is **ST-LINK** and click **Finish** to confirm your settings.

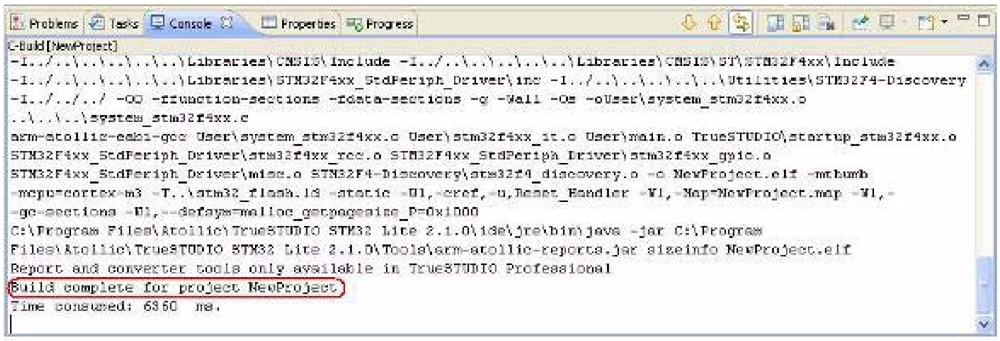
Figure 46. TrueSTUDIO® Misc Settings dialog box



6. Your project is successfully created. Atollic TrueSTUDIO® generates target specific sample files (main.c, stm32f4xx_it.c...) in the Project folder to simplify development. You can tailor this project to your needs by modifying these sample files.

7. To build your project, in the **Project** menu, click **Build Project**.
8. Your project is successfully compiled.

Figure 47. TrueSTUDIO® project successfully built



9. Before running your application, establish the connection with the STM32F4DISCOVERY board as described in *Section 2: Getting started*. To program the Flash memory and begin debugging, follow the instructions given in *Section 8.2: Debugging and running your TrueSTUDIO project on page 31*.

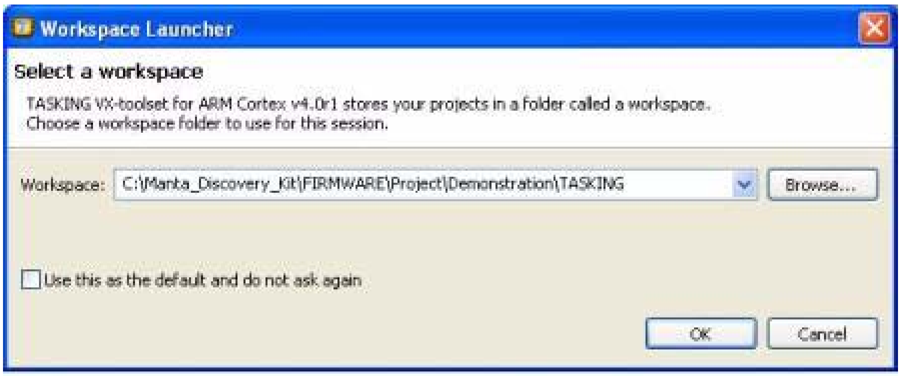
9 Using TASKING

9.1 Building an existing TASKING project

Follow these steps to build an existing TASKING project.

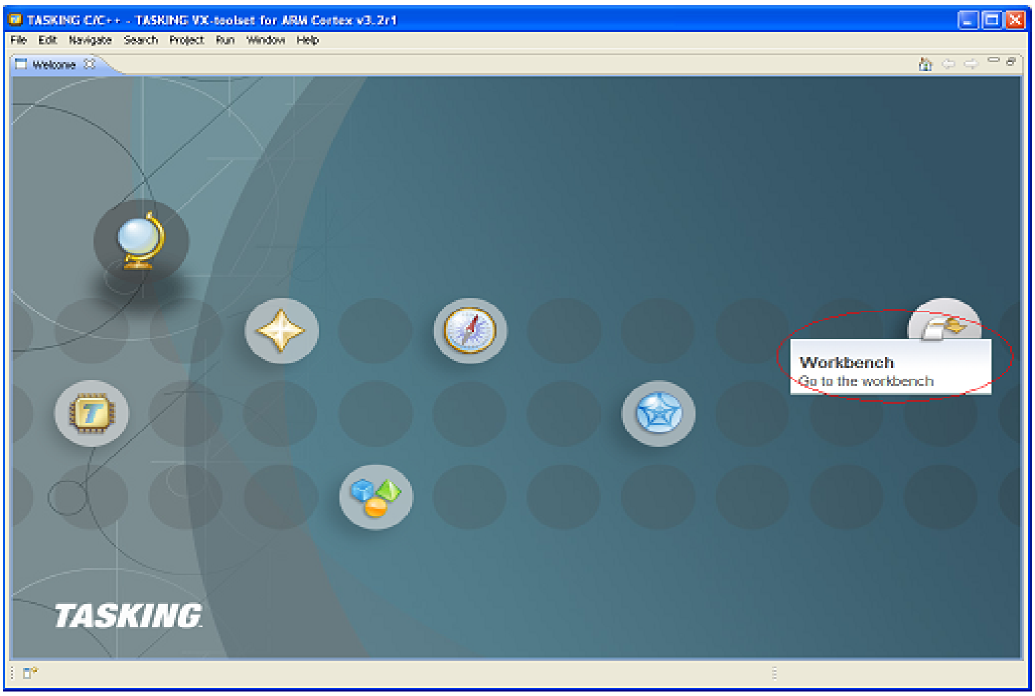
1. Open the **TASKING VX-toolset for ARM Cortex** IDE. The program launches and asks for the Workspace location.

Figure 48. TASKING workspace launcher dialog box



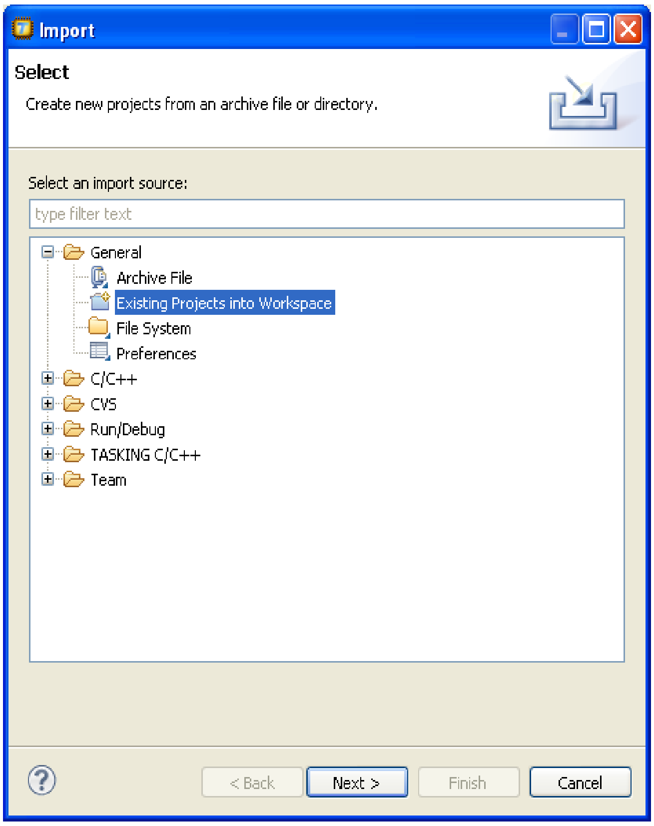
2. Browse to select the STM32F4DISCOVERY Demonstration TASKING workspace and click **OK** to save your settings and to display the Welcome screen. To start using TASKING, click **Go to the workbench**.

Figure 49. TASKING VX-Toolset for ARM Cortex welcome screen



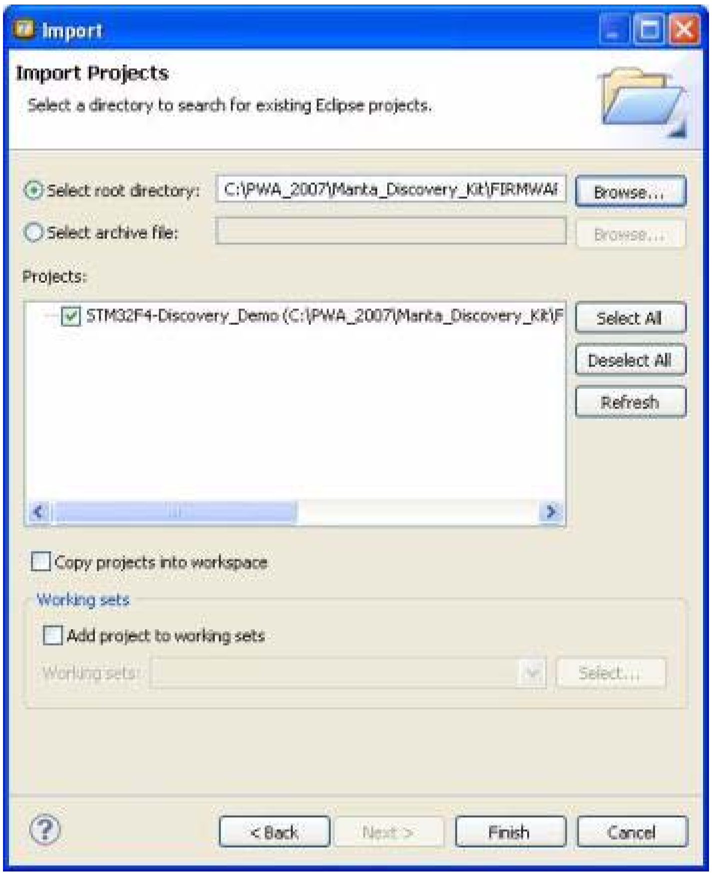
3. The TASKING Discovery workspace contains a demo project for the STM32F4DISCOVERY kit. To load this project, in the File menu, select **Import...** to display the **Import** dialog box.
4. In the **Import** window, open **General**, select **Existing Projects into Workspace** and click **Next**.

Figure 50. TASKING import source select dialog box



5. Click **Select root directory**, browse to the TASKING workspace folder and select the **STM32F4-Discovery** project.

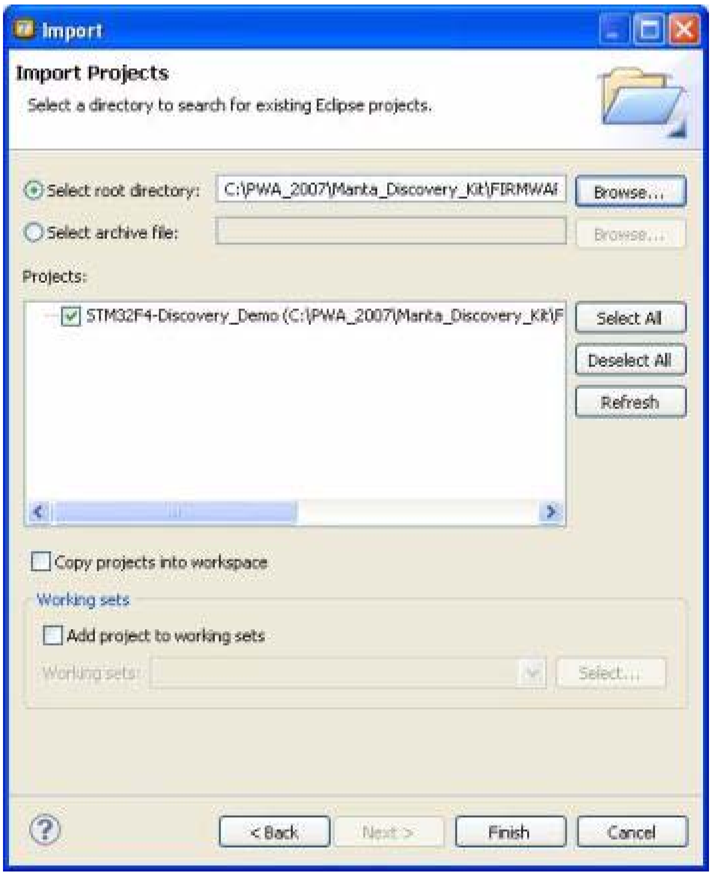
Figure 51. TASKING import projects dialog box



6. In the **Projects** window, select the **STM32F4-Discovery** and click **Finish**.
7. In the **Project Explorer**, select the **STM32F4-Discovery** project. Open the **Project** menu, and click **Build Project**.

8. If your project is successfully compiled, the following window is displayed.

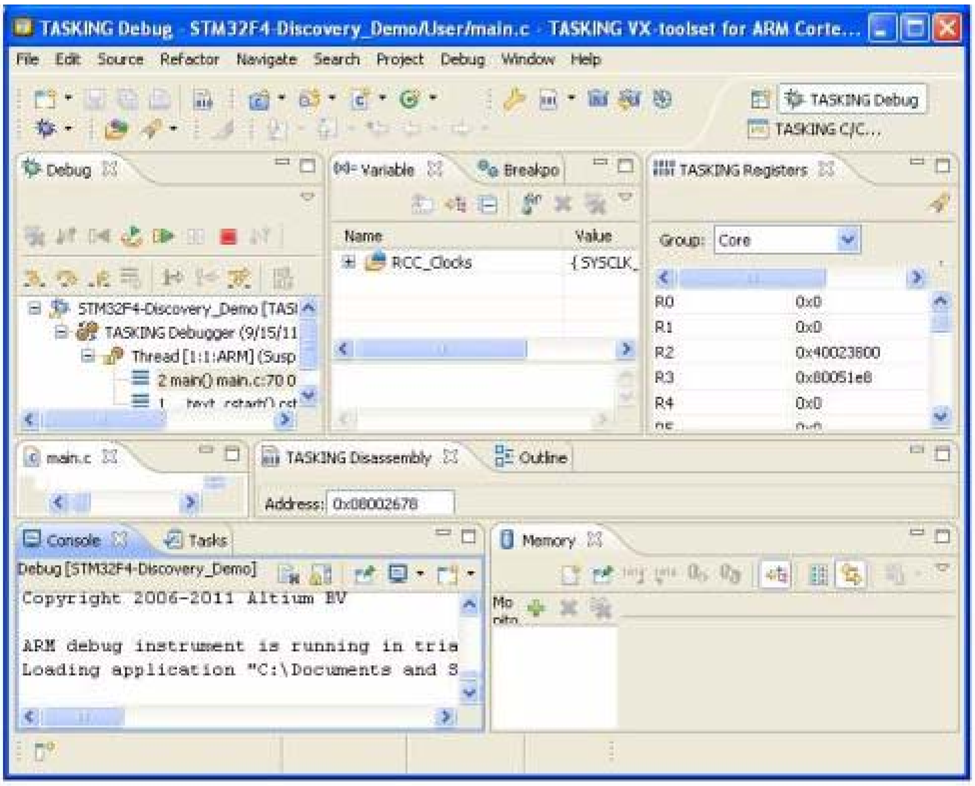
Figure 52. TASKING project successfully compiled



9.2 Debugging and running your TASKING project

Figure 53 shows the first step for debugging and running your TASKING project. From the project toolbar menu select **Debug** > **Debug STM32F4-Discovery_Demo.**

Figure 53. TASKING debug window



The debugger in TASKING can be used to debug source code at C and assembly levels, set breakpoints, monitor individual variables and watch events during the code execution.

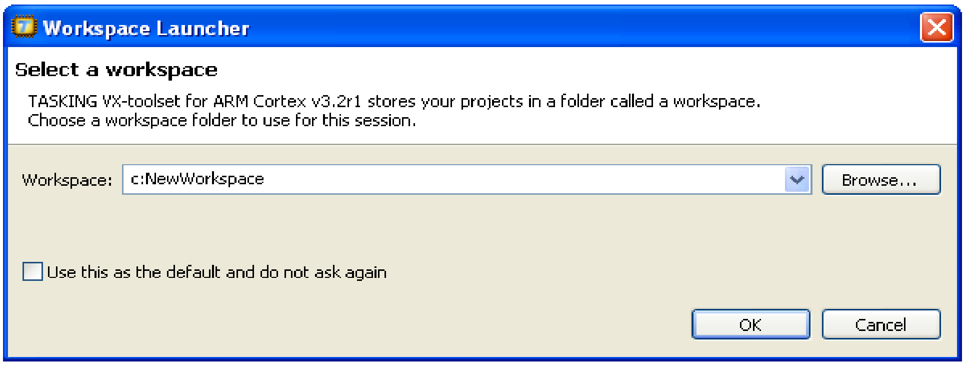
To run your application, from the **Run** menu, select **Resume**, or alternatively click the **Resume** button in the toolbar.

9.3 Creating your first application using TASKING toolchain

The debug session is launched as follows:

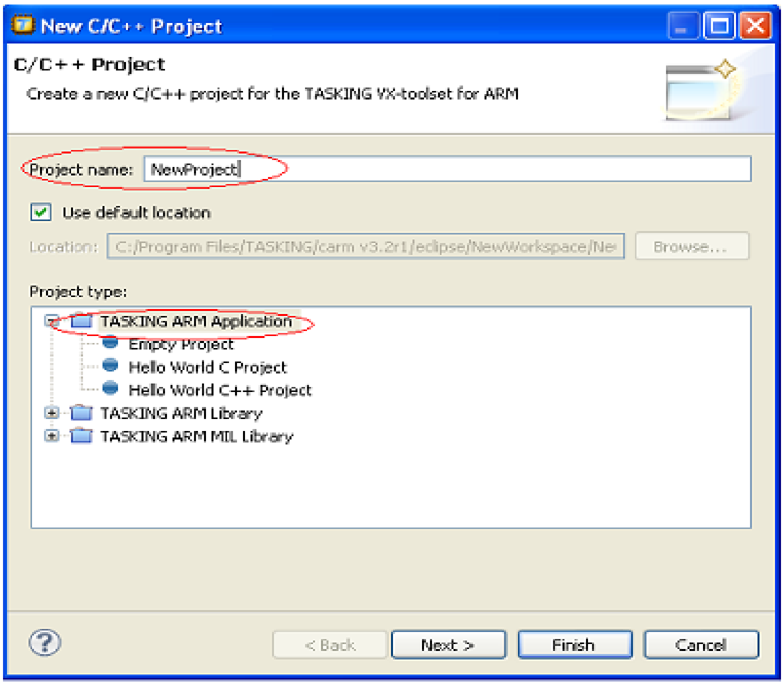
1. Open TASKING VX-Toolset for ARM Cortex. The program launches and asks for the Workspace location. Browse to select an existing workspace, or enter a new workspace location and click **OK** to confirm.

Figure 54. TASKING workspace launcher dialog box



2. When TASKING displays its Welcome window, click **Go to workbench** to open the main window. In the **File** menu, select **New** > TASKING VX-toolset for ARM C/C++ **Project**.
3. In the **New C/C++ Project** dialog box**,** enter the new **Project name**; then in the **Project type** box, select **TASKING ARM Application** and click **Next.**

Figure 55. TASKING New C/C++ Project dialog box



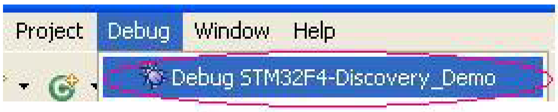
The list of the supported devices is shown, select STMicroelectronics > STM32F407VG > STM32F407VGT6 as shown below in *Figure 56*.

Figure 56. Processor selection

To configure the project for Manta DISCOVERY board, select **Debug** > **Debug configurations** and choose **STMicroelectronics STM32F4 Manta Discovery Kit**. Choosing **STMicroelectronics STM32F4 Manta Discovery Kit** as the evaluation board, will add automatically the needed linker file and will configure the project as follows:

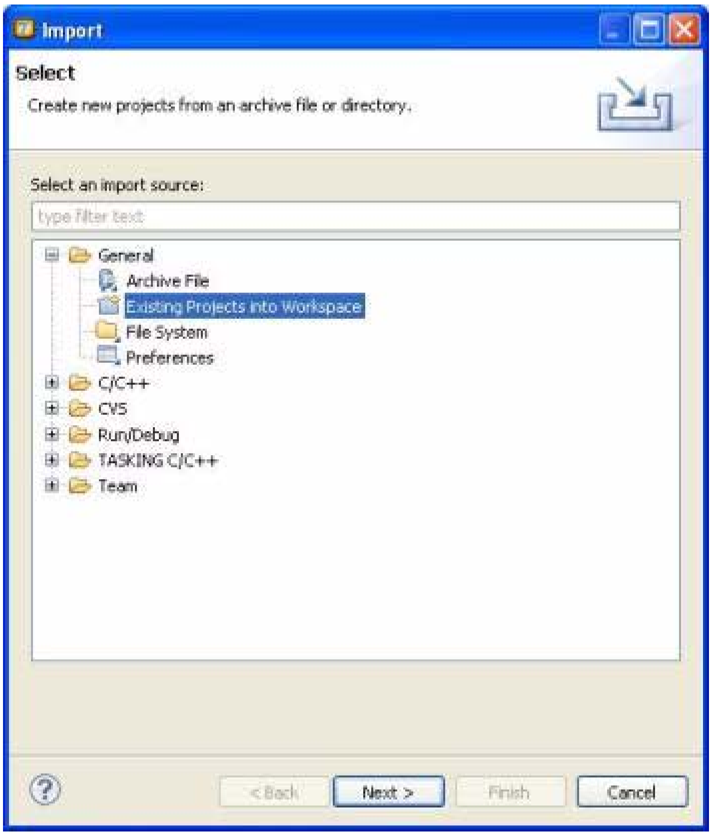
- *Microcontroller: STM32F407VGT6*
- *Debug probe: ST-LINK*
- *Connection: Serial Wire Debugging (SWD).*

Figure 57. Debug configuration



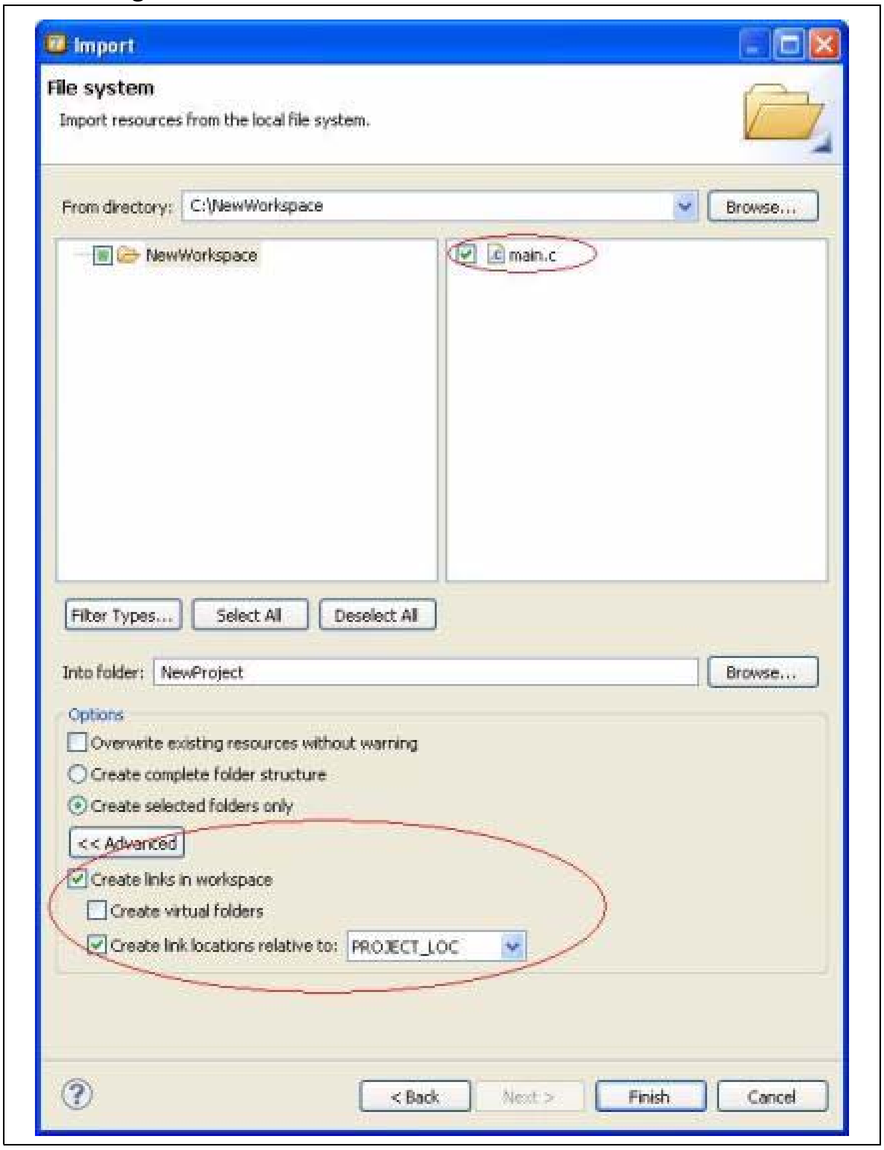
4. To add source file to your project right click on the project from the C/C++ project window and select **Import**.
5. The **Import** dialog box is displayed, select **General** and the desired file as shown in *Figure 58: TASKING Import dialog box*

Figure 58. TASKING Import dialog box



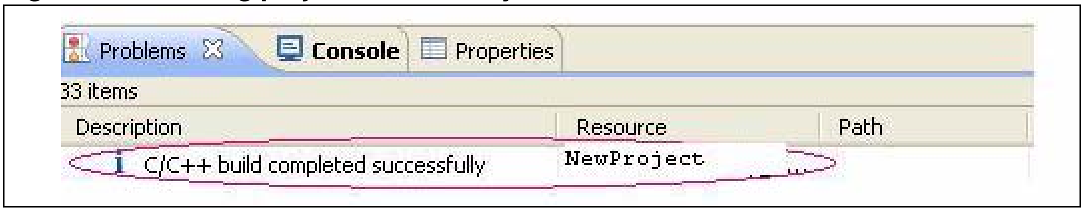
6. Click **Next**.Fill the displayed window as following and then browse to your source file.

Figure 59. Adding a new source file window.



7. Select **main.c** file and click **Finish**.
8. To build your project click on **Project** > **Build Project** from the toolbar menu.
9. Your Project is successfully compiled.

Figure 60. Tasking project successfully built



10. Before running your application, establish the connection with the STM32F4DISCOVERY board as described in *Section 2: Getting started*.

10 Revision history

Table 1. Document revision history

Date	Revision	Changes
23-Sep-2011	1	Initial release.