UM1677
User manual

Getting started with STM32F030 Value Line Discovery development tools

Introduction

This document describes the software, firmware environment and development recommendations required to build an application around the STM32F0308-DISCOVERY board (32F0308DISCOVERY) with demonstration firmware (STSW-STM32140).

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 1: Getting started*.
- *Section 2* describes the firmware applications package.
- *Section 4* presents development toolchain installation and overview of ST-LINK/V2 interface.
- *Section 5*, *Section 6*, *Section 7*, and Section 8 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

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:

- STM32F0308-DISCOVERY high-performance discovery board data brief
- STM32F0308-DISCOVERY peripherals firmware examples (AN4062)
- STM32F0xx reference manual (RM0360)
- STM32F030x4 STM32F030x6 STM32F030x8 datasheet

The above documents are available at www.st.com/stm32f0-discovery.

Contents

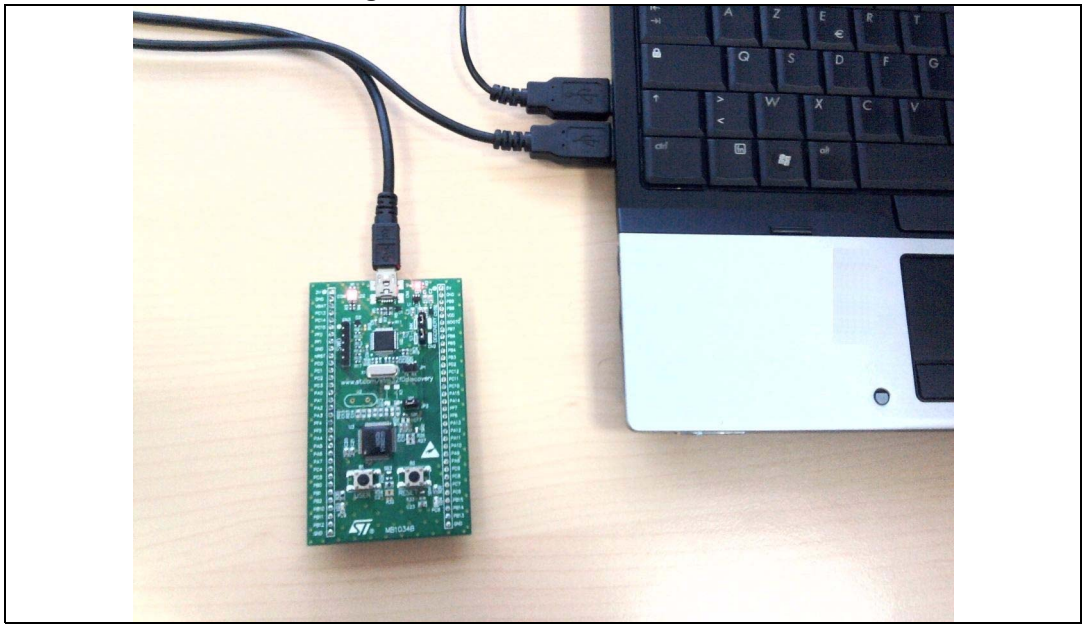
List of figures

1 Getting started

1.1 System requirements

Before running your application, you should establish the connection with the STM32F0308-DISCOVERY board as following.

Figure 1. Hardware environment



To run and develop any firmware applications on your STM32F0308-DISCOVERY 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.

1.2 Running the built-in demonstration

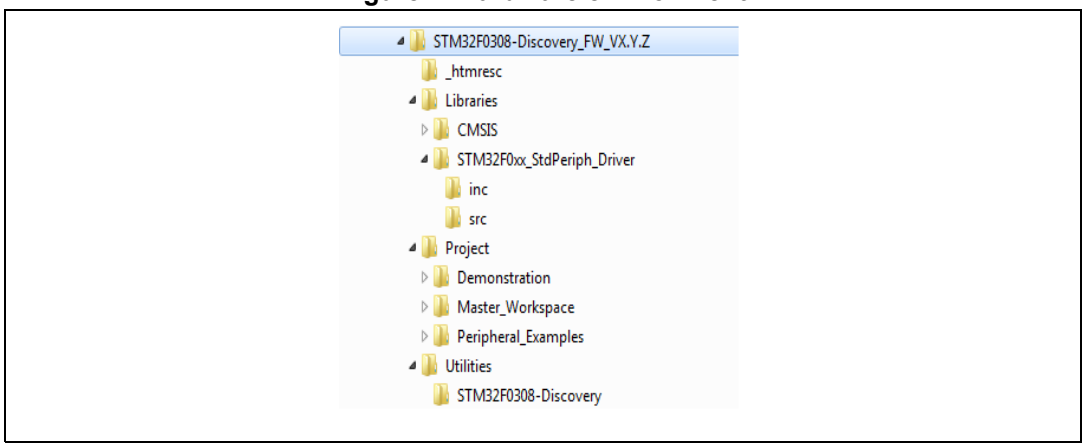
The board comes with the demonstration firmware preloaded in the Flash memory. Follow the steps below to run it:

- Check the jumper position on the board, JP2 on, CN2 on (Discovery selected).
- Connect the STM32F0308-DISCOVERY board to a PC with a 'USB type A to Mini-B' cable through USB connector CN1 to power the board. Then red LEDs LED1 (PWR) and LED2 (COM) light up and green LED3 blinks.
- Press user button B1 (Button left corner of the board). The blinking of green LED3 changes according to clicks on user button B1.
- Each click on the USER push-button is confirmed by blue LED4.

2 Description of the firmware package

The STM32F0308-DISCOVERY firmware applications are provided in one single package and supplied in one single zip file. The extraction of the zip file generates one folder, *STM32F0308-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

2.1 Libraries folder

This folder contains the Hardware Abstraction Layer (HAL) for STM32F0xx devices.

2.1.1 CMSIS subfolder

This subfolder contains the STM32F0xx and Cortex-M0 CMSIS files.

Cortex-M0 CMSIS files consist of:

- *Core Peripheral Access Layer:* contains name definitions, address definitions and helper functions to access Cortex-M0 core registers and peripherals. It defines also a device independent interface for RTOS Kernels that includes debug channel definitions.

STM32F0xx CMSIS files consist of:

- *stm32f0xx.h:* contains the definitions of all peripheral registers, bits, and memory mapping for STM32F0xx devices. The file is the unique include file used in the application programmer C source code, usually in main.c.
- *system_stm32f0xx.c/.h*: contains the system clock configuration for STM32F0xx 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_stm32f030x8.s* file.
- *startup_stm32f030x8.s*: provides the Cortex-M0 startup code and interrupt vectors for all STM32F0xx device interrupt handlers.

2.1.2 STM32F0xx_StdPeriph_Driver subfolder

This subfolder contains sources of STM32F0xx peripheral drivers.

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, *stm32f0xx_ppp.c*, and a header file, *stm32f0xx_ppp.h*. The *stm32f0xx_ppp.c* file contains all the firmware functions required to use the PPP peripheral.

2.2 Project folder

This folder contains the source files of the STM32F030 Discovery firmware applications.

2.2.1 Demonstration subfolder

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

A binary image (*.hex) of this demonstration is provided under Binary subfolder. You can use any in-system programming tool to reprogram the demonstration using this binary image.

2.2.2 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.

2.2.3 Peripheral_Examples subfolder

This subfolder contains a set of examples for some peripherals with preconfigured projects for EWARM, MDK-ARM and TrueSTUDIO toolchains. See *Section 4* and *STM32F0308-DISCOVERY peripheral firmware examples,* AN4062, for further details.

2.3 Utilities folder

This folder contains the abstraction layer for the STM32F030 Discovery hardware. It provides the following drivers:

- *stm32f0308_discovery.c*: provides functions to manage the user push-button and 2 LEDs (LED3 and LED4).

3 Binary images for reprogramming firmware applications

This section describes how to use the provided binary images to reprogram the firmware applications. The STM32F030 Discovery firmware package contains binary images (*.hex) of the provided applications under Binary subfolder. You can use any in-system programming tool to reprogram the demonstration using this binary image.

to reprogram the firmware applications, use the “in-system programming tool” and:

1. Connect the STM32F030 Discovery board to a PC with a 'USB type A to Mini-B' cable through USB connector CN1 to power the board.
2. Make sure that the embedded ST-LINK/V2 is configured for in-system programming (both CN3 jumpers ON).
3. Use *.hex binary (for example, *\Project\Demonstration\Binary\STM32F0308-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).

4 ST-LINK/V2 installation and development

STM32F030 Discovery 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 6.5* 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 6.5\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.

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.

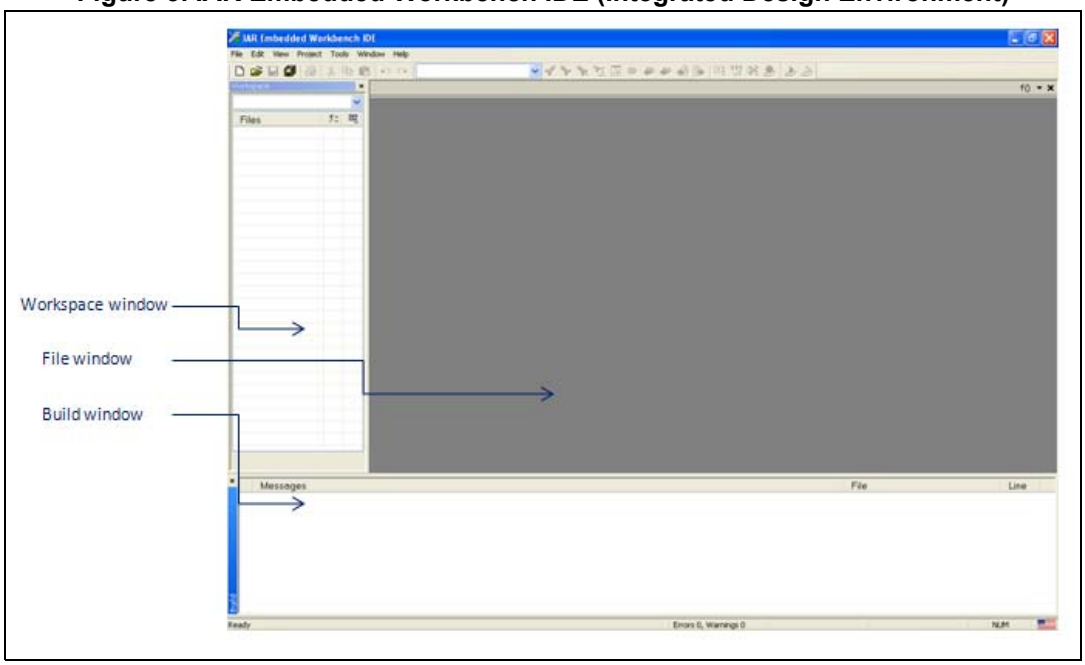
5 Using IAR Embedded Workbench® for ARM

5.1 Building an existing EWARM project

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

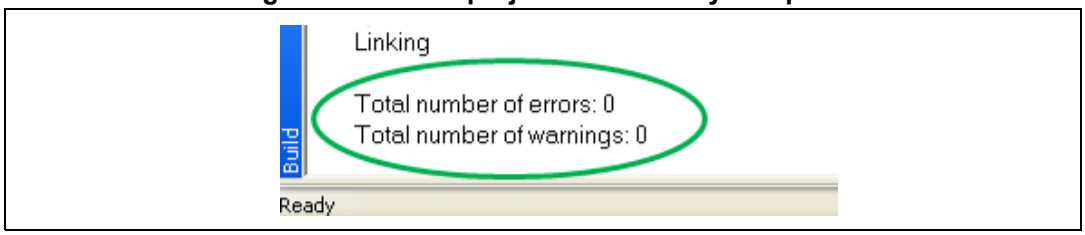
1. Open the IAR Embedded Workbench® for ARM (EWARM).
   Figure 3 shows the basic names of the windows referred to in this document.

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 an *example* or *demonstration* or *template* 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



If you need to change your project settings (Include and preprocessor defines), you need just to go through project options:

– For Include directories

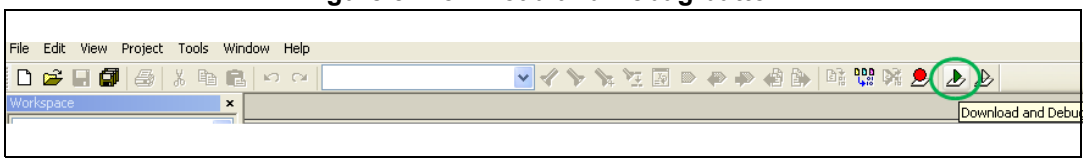
Project>Options…>C/C++ compiler>

– For pre-processor defines

Project>Options…C/C++ compiler>pre-processor>

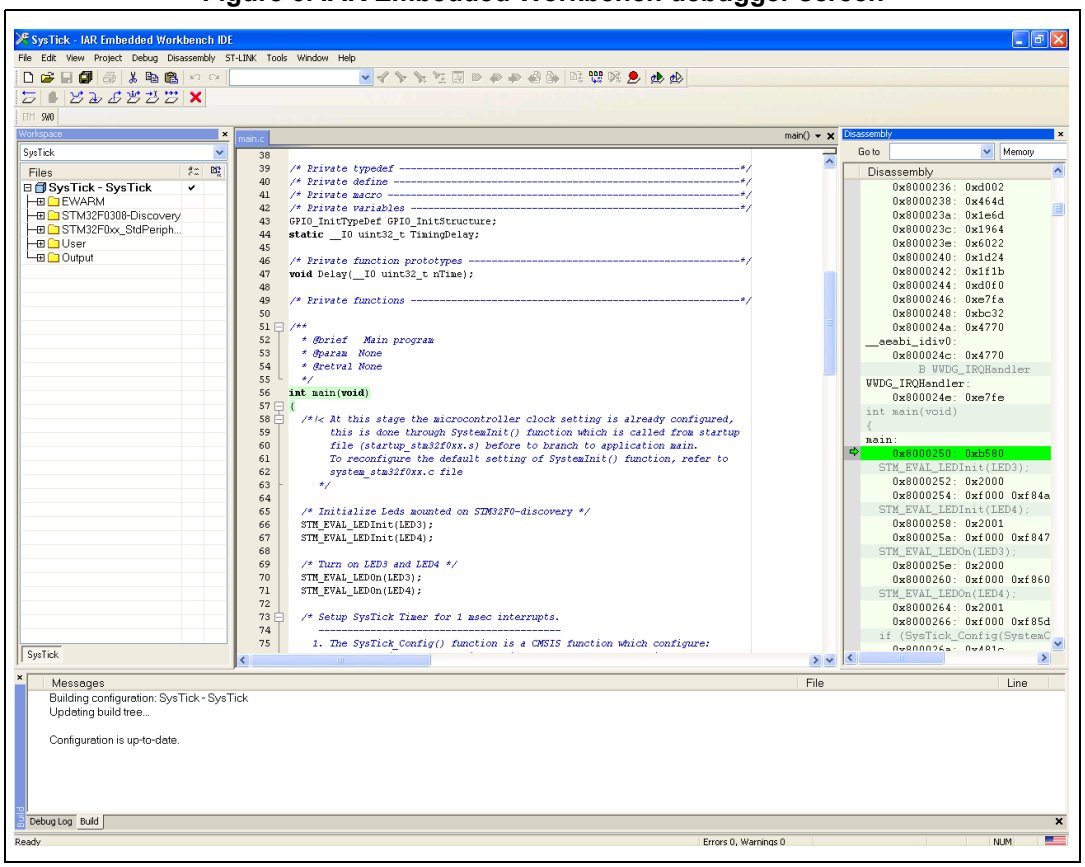
5. In the IAR Embedded Workbench IDE, from the **Project menu**, select **Download and Debug** or, alternatively, click the **Download and Debug** button the in toolbar, to program the Flash memory and begin debugging.

Figure 5. Download and Debug button



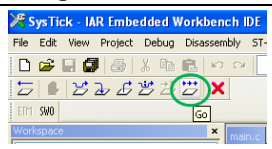
6. The debugger in the IAR Embedded Workbench can be used to debug source code at C and assembly levels, set breakpoints, monitor individual variables and watch events during the code execution.

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

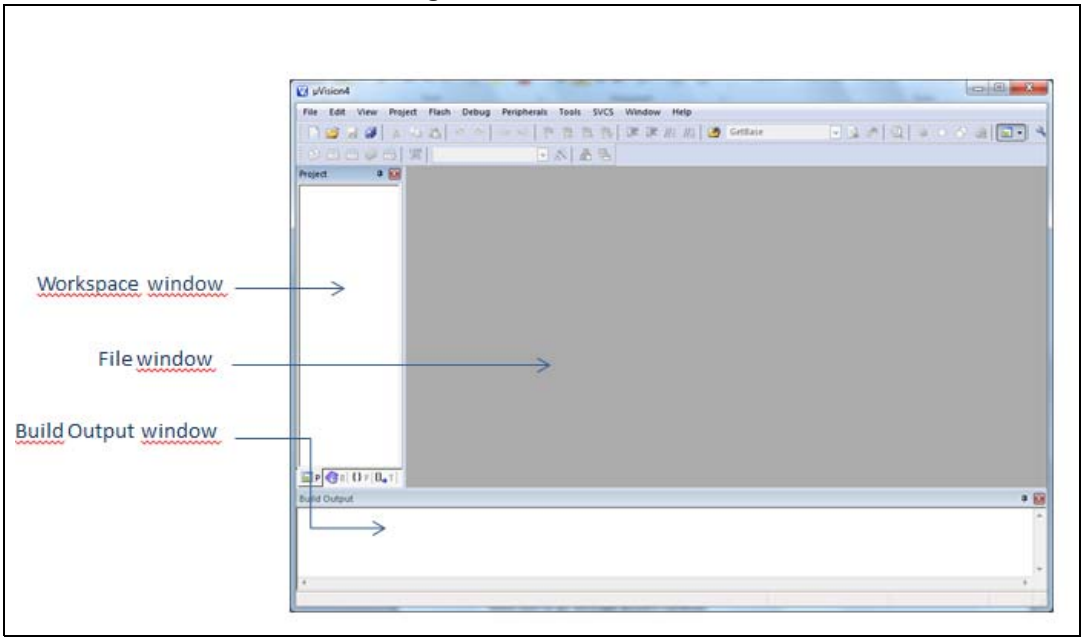


6 MDK-ARM toolchain

1. Open Keil MDK-ARM Microcontroller Kit,

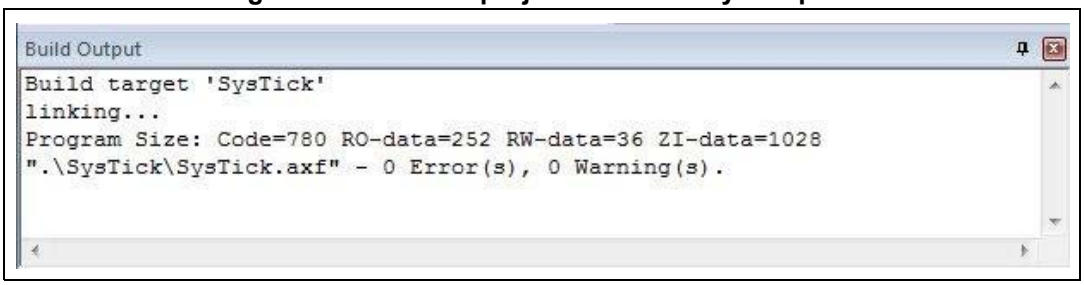
Figure 8 shows the basic names of the "Keil uVision4"windows referred to in this document.

Figure 8. uVision4 IDE



2. In the **Project** menu, select **Open Project**... Browse to select either an example or demonstration or template 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 window in Figure 3 is displayed

Figure 9. MDK-ARM project successfully compiled



If you need to change your project settings (Include and preprocessor defines), you need just to go through project options:

– For Include directories'

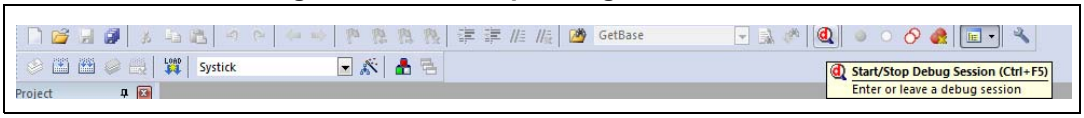
Project>Options for Target > C/C++ > Include Paths

– For pre-processor defines

Project>Options for Target > C/C++ > Preprocessor symbols > Define

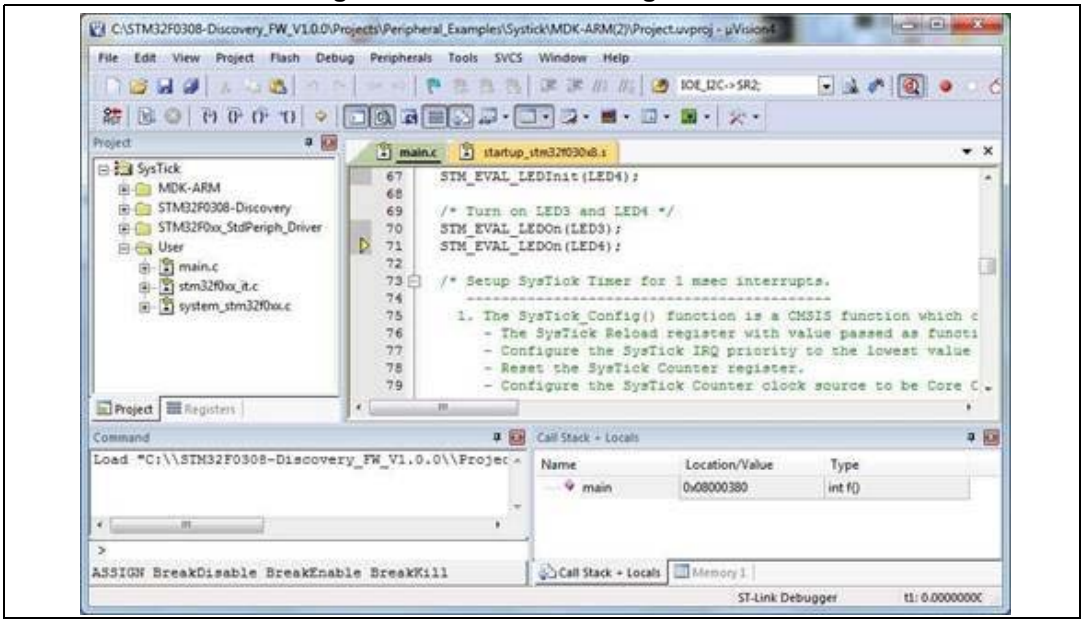
5. In the MDK-ARM IDE, from the **Debug** menu, select **Start/Stop Debug Session** or, alternatively, click the **Start/Stop Debug Session** button the in toolbar, to program the Flash memory and begin debugging.

Figure 10. Start/Stop Debug Session button



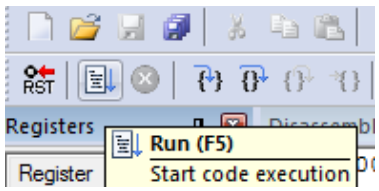
6. The debugger in the MDK-ARM can be used to debug source code at C and assembly levels, set breakpoints, monitor individual variables and watch events during the code execution.

Figure 11. MDK-ARM debuger screen



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

Figure 12. Run button

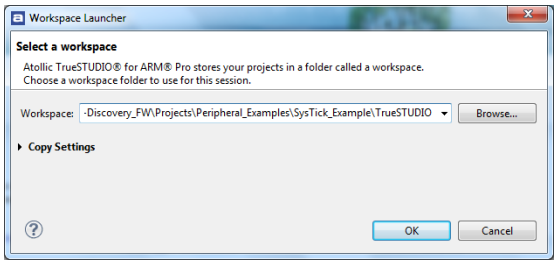


7 TrueSTUDIO toolchain

Follow these steps to use the TrueSTUDIO® toolchain:

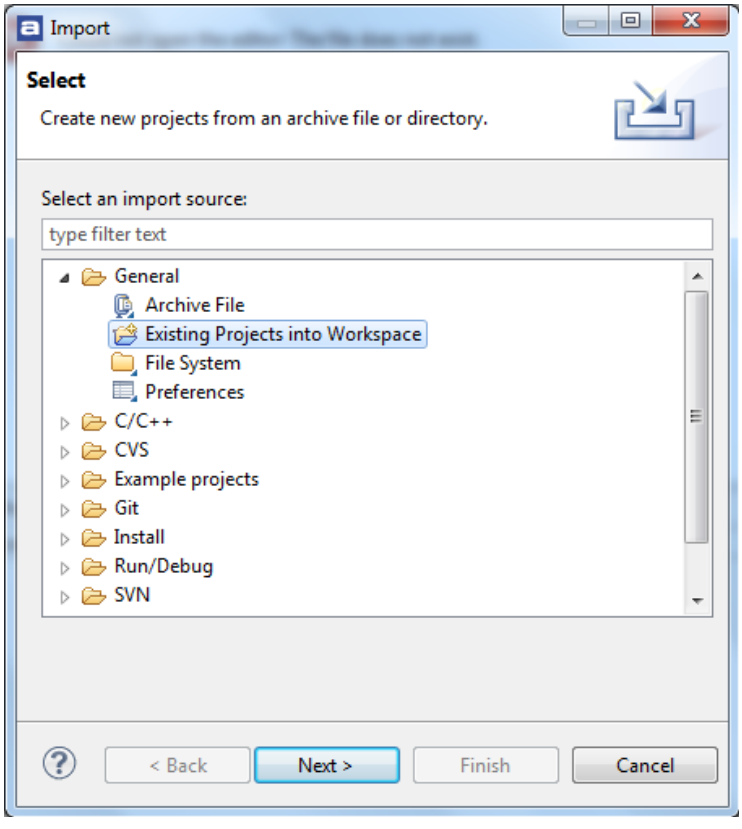
1. Open Atollic TrueSTUDIO® for ARM product. The program launches and asks for the Workspace location.

Figure 13. TrueSTUDIO® workspace launcher dialog box



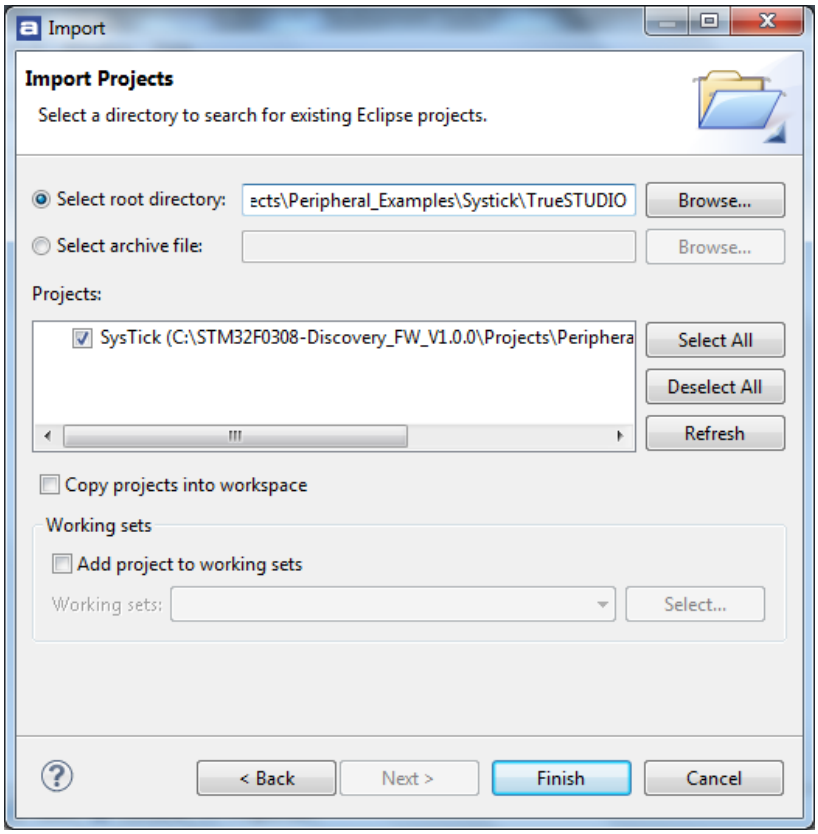
2. Browse to select a TrueSTUDIO® workspace of either an *example* or *demonstration* or *template* workspace file and click OK to load it.
3. To load an existing project in the selected workspace, select **Import** from the **File** menu to display the **Import** dialog box.
4. In the **Import** window, open **General**, select **Existing Projects** into Workspace and click **Next**.

Figure 14. Atollic TrueSTUDIO® import source select dialog box



5. Click Select root directory, browse to the TrueSTUDIO® workspace folder and select the following shownin *Figure 15*:

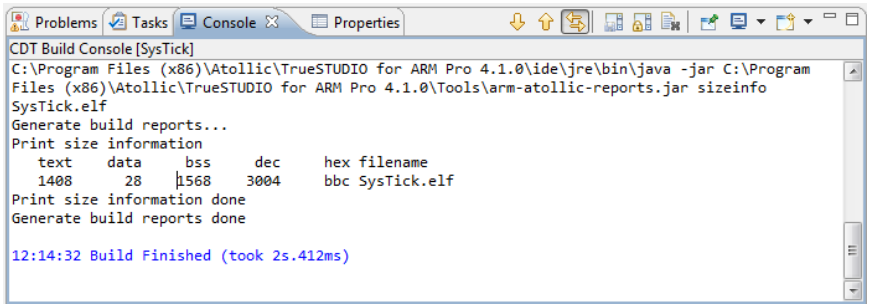
Figure 15. Atollic TrueSTUDIO® import projects dialog box



6. In the Projects panel, select the project and click Finish.
7. In the Project Explorer, select the project, open the Project menu, and click Build Project.

8. If your project is successfully compiled, the following messages will be displayed on the **Console** window.

Figure 16. TrueSTUDIO® project successfully compiled



If you need to change the project settings (Include directories and preprocessor defines), you need just to go through Project>Properties, select C/C++ Build>Settings from the left panel:

– For Include directories

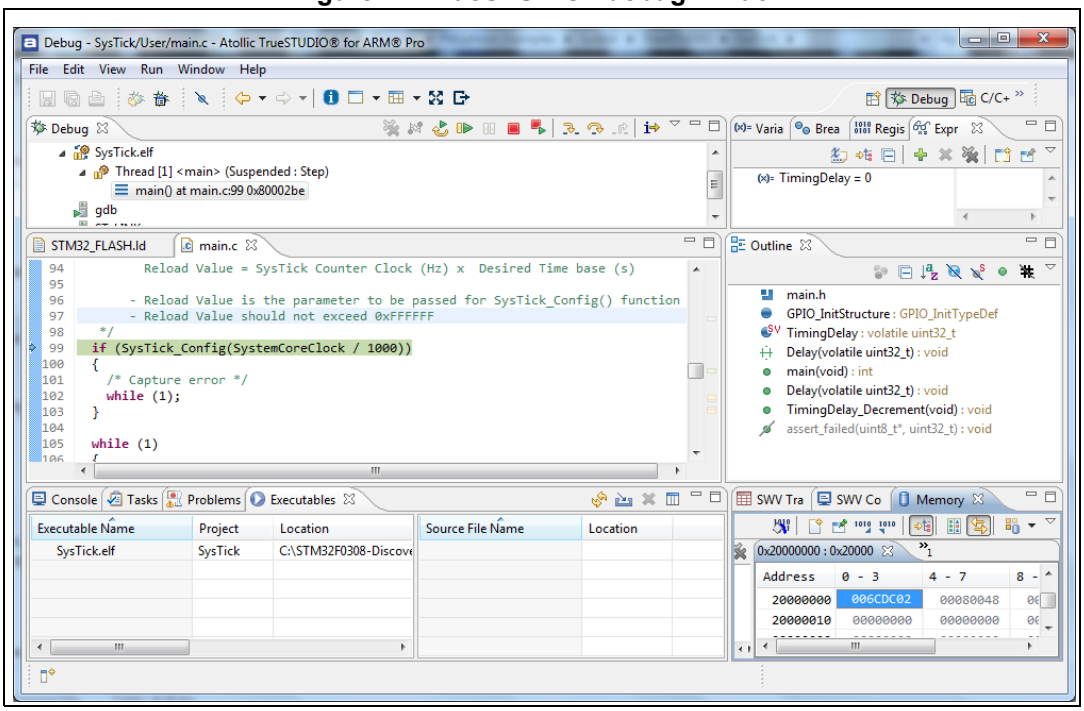
C Compiler>Directories>Include path

– For pre-processor defines

C Compiler>Symbols> Defined symbols

9. To debug and run the application, select the project In the **Project Explorer** and press **F11** to start a debug session. (See *Figure 17*.)

Figure 17. 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 Revision history

Table 1. Document revision history

Date	Revision	Changes
03-Oct-2013	1	Initial release.