Freescale Semiconductor
User's Guide

Document Number: KSDK20MKS22FN256GSUG
Rev. 0, 12/2015

Getting Started with Kinetis SDK (KSDK) v.2.0 for the MKS22FN12 Devices

Contents

1 Overview

The Kinetis Software Development Kit (KSDK) provides comprehensive software support for Kinetis Microcontrollers. The KSDK includes a flexible set of peripheral drivers designed to speed up and simplify development of embedded applications. Along with the peripheral drivers, the KSDK provides an extensive and rich set of example applications covering everything from basic peripheral use case examples to full demo applications. Lastly, the KSDK contains the latest available RTOS kernels, a USB stack, and various other middlewares to support rapid development on Kinetis devices.

For supported toolchain versions, see the Kinetis SDK v.2.0.0 Release Notes (document KSDK200MKS22FN256RN).

For the latest version of this and other Kinetis SDK documents, see the Kinetis SDK homepage www.freescale.com/ksdk

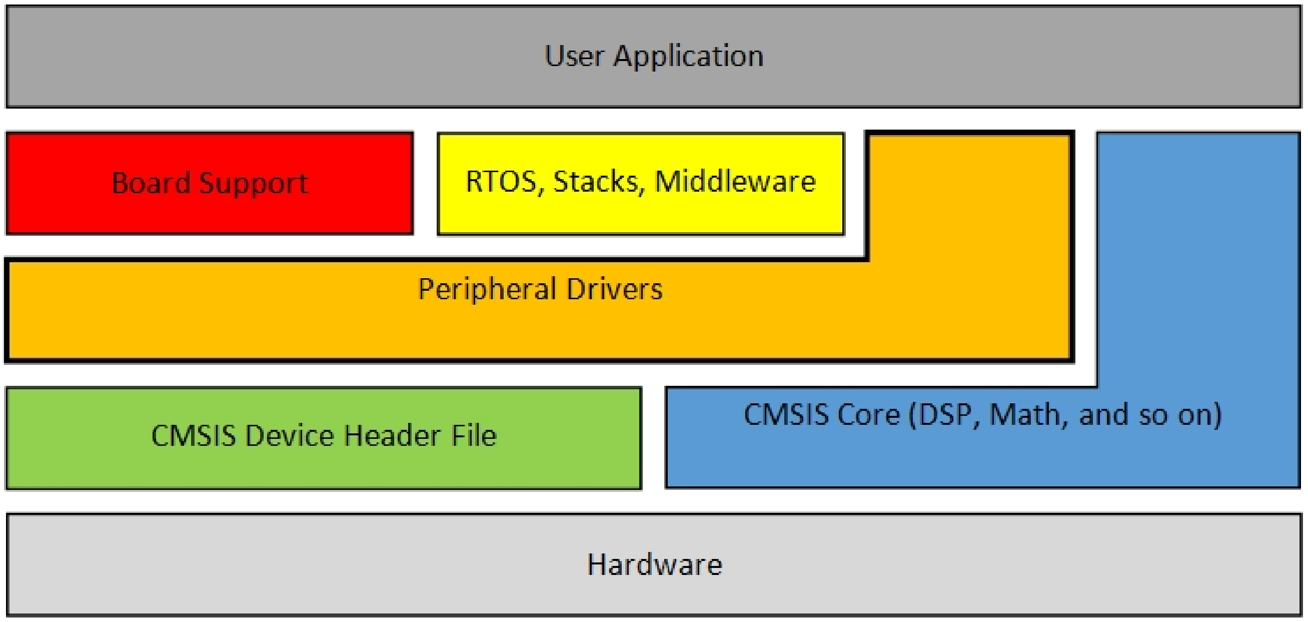


Figure 1. KSDK layers

2 KSDK Board Support Folders

KSDK board support provides example applications for Kinetis development and evaluation boards. Board support packages are found inside of the top level boards folder, and each supported board has its own folder (a KSDK package can support multiple boards). Within each <board_name> folder there are various sub-folders to classify the type of examples they contain. These include (but are not limited to):

- demo_apps: Full-featured applications intended to highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.
- driver_examples: Simple applications intended to concisely illustrate how to use the KSDK's peripheral drivers for a single use case. These applications typically only use a single peripheral, but there are cases where multiple are used (for example, ADC conversion using DMA).
- rtos_examples: Basic FreeRTOS examples showcasing the use of various RTOS objects (semaphores, queues, and so on) and interfacing with the KSDK's RTOS drivers
- usb: Applications that use the USB host/device/OTG stack.

2.1 Locating Example Application Source Files

When opening an example application in any of the supported IDEs, there are a variety of source files referenced. The KSDK devices folder is designed to be the "golden core" of the application and is, therefore, the central component to all example applications. Because it's a core component, all of the examples reference the same source files and, if one of these files is modified, it could potentially impact the behavior of other examples.

The main areas of the KSDK tree used in all example applications are:

- devices/<device_name>: The device's CMSIS header file, KSDK feature file and a few other things.
- devices/<device_name>/drivers: All of the peripheral drivers for your specific MCU.

- devices/<device_name>/<toolchain> Toolchain-specific startup code. Vector table definitions are here.
- devices/<device_name>/utilities: Items such as the debug console that are used by many of the example applications.

For examples containing middleware/stacks and/or a RTOS, there will be references to the appropriate source code. Middleware source files are located in the *middleware* folder and RTOSes are in the *rtos* folder. Again, the core files of each of these are shared, so modifying them could have potential impacts on other projects that depend on them.

2.2 Locating Example Application Source Files

When opening an example application in any of the supported IDEs, there are a variety of source files referenced. The KSDK *devices* folder is designed to be the "golden core" of the application and is, therefore, the central component to all example applications. Because it's a core component, all of the examples reference the same source files and, if one of these files is modified, it could potentially impact the behavior of other examples.

The main areas of the KSDK tree used in all example applications are:

- devices/<device_name>: The device's CMSIS header file, KSDK feature file and a few other things.
- devices/<device_name>/drivers: All of the peripheral drivers for your specific MCU.
- devices/<device_name>/<tool_name>: Toolchain-specific startup code. Vector table definitions are here.
- devices/<device_name>/utilities: Items such as the debug console that are used by many of the example applications.

For examples containing middleware/stacks and/or a RTOS, there will be references to the appropriate source code. Middleware source files are located in the *middleware* folder and RTOSes are in the *rtos* folder. Again, the core files of each of these are shared, so modifying them could have potential impacts on other projects that depend on them.

3 Run an example application using IAR

This section describes the steps required to build, run, and debug example applications provided in the Kinetis SDK. The hello_world demo application targeted for the MAPS-KS22 hardware platform is used as an example, although these steps can be applied to any example application in the KSDK.

3.1 Build an example application

The following steps will guide you through opening the hello_world example application. These steps may change slightly for other example applications as some of these applications may have additional layers of folders in their path.

1. If not already done, open the desired demo application workspace. Most example application workspace files can be located using the following path:

   <install_dir>/boards/<board_name>/example_type/*<application_name>/iar*

   Using the MAPS-KS22 board as an example, the hello_world workspace is located in*<install_dir>/boards/mapsks22/demo_apps/hello_world/iar/hello_world.eww*
2. Select the desired build target from the drop-down. For this example, select the "hello_world – Debug" target.

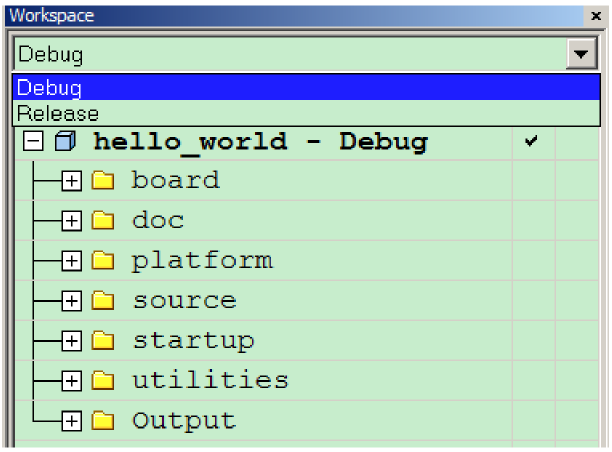
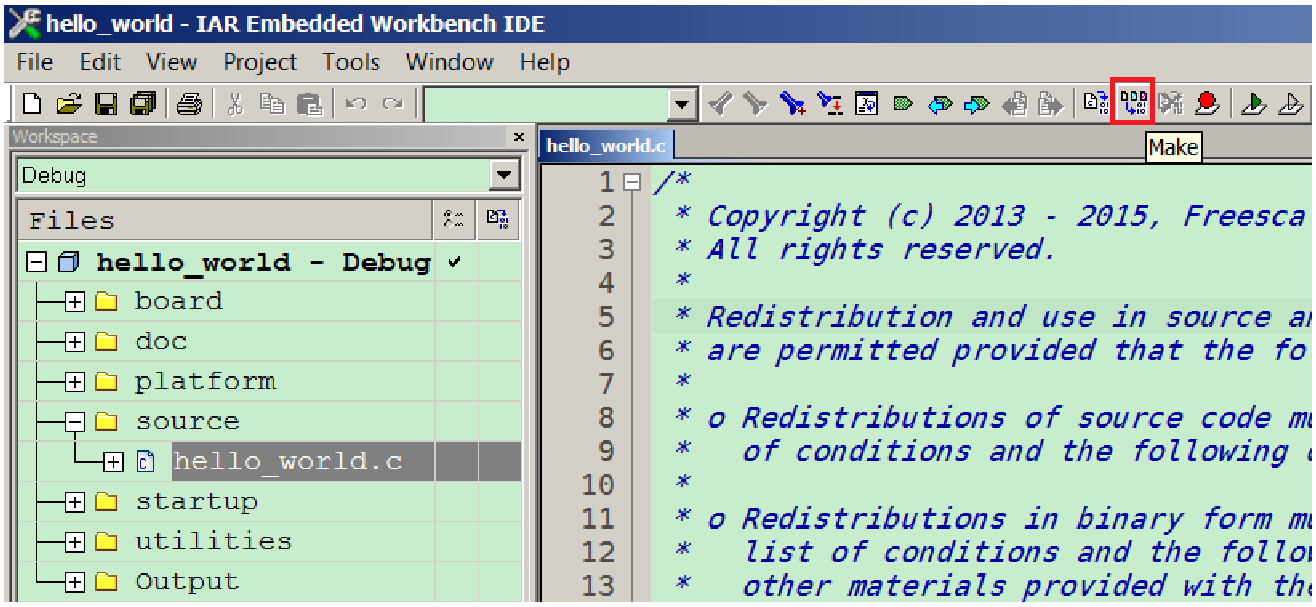


Figure 2. Demo build target selection

3. To build the demo application, click the "Make" button, highlighted in red below.



```
/*
 * Copyright (c) 2013 - 2015, Freesca
 * All rights reserved.
 *
 * Redistribution and use in source a
 * are permitted provided that the fo
 *
 * o Redistributions of source code m
 *   of conditions and the following
 *
 * o Redistributions in binary form m
 *   list of conditions and the follo
 *   other materials provided with th
```

Figure 3. Build the demo application

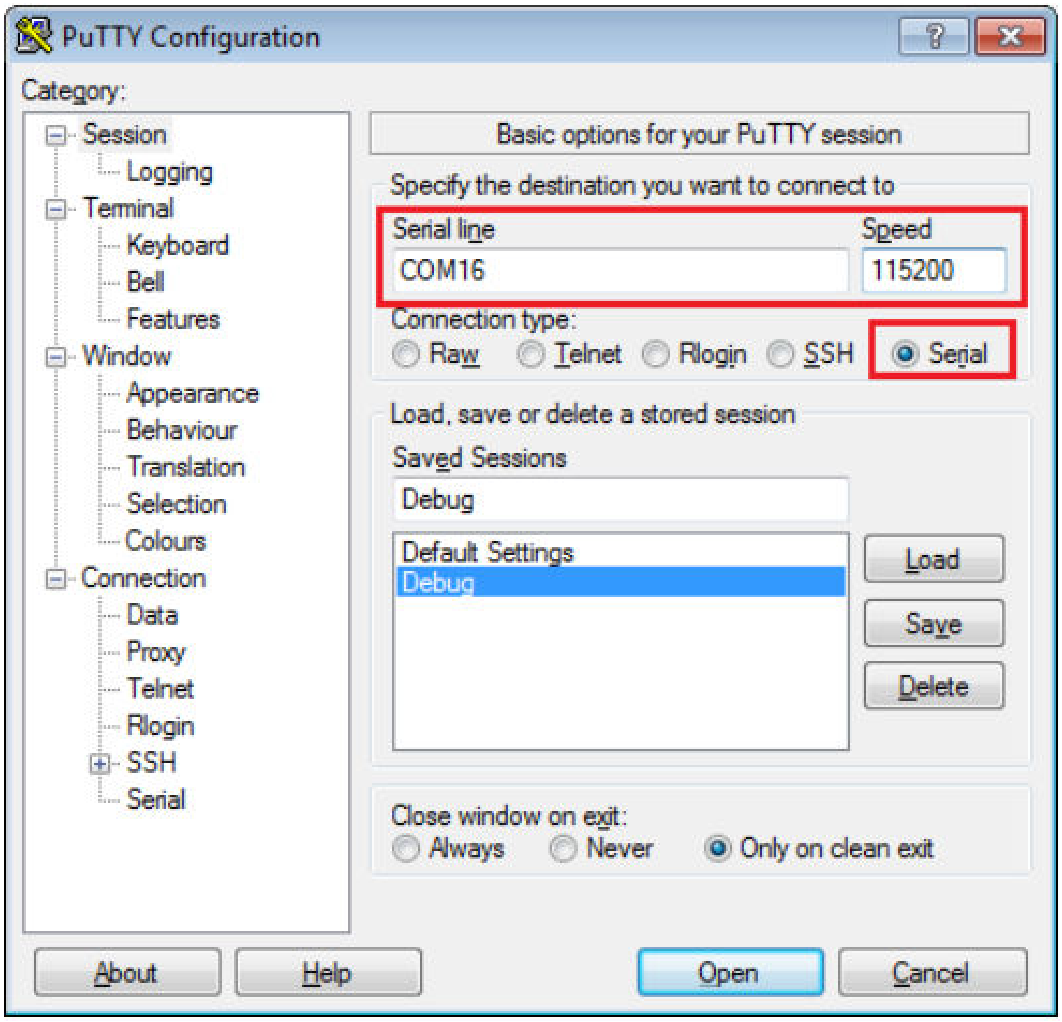
4. The build will complete without errors.

3.2 Run an example application

To download and run the application, perform these steps:

1. Reference the table in Appendix B to determine the debug interface that comes loaded on your specific hardware platform.
   - For boards with CMSIS-DAP/mbed/DAPLink interfaces, visit developer.mbed.org/handbook/Windows-serial-configuration and follow the instructions to install the Windows® operating system serial driver.
   - For boards with P&E Micro interfaces, visit www.pemicro.com/support/downloads_find.cfm and download the P&E Micro Hardware Interface Drivers package.
   - For the MRB-KW01 board, visit www.freescale.com/USB2SER to download the serial driver. This board does not support OpenSDA, so an external debug probe (such as a J-Link) is required. Steps below referencing OpenSDA do not apply as there is only a single USB connector for serial output.

2. Connect the development platform to your PC via USB cable between the OpenSDA USB connector and the PC USB connector.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port (to determine the COM port number, see Appendix A). Configure the terminal with these settings:
   a. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUD variable in board.h file)
   b. No parity
   c. 8 data bits
   d. 1 stop bit



Serial line: COM16
Speed: 115200
Connection type: Raw Telnet Rlogin SSH [✓ Serial]
Saved Sessions: Debug
Default Settings
Debug
Close window on exit: Always Never [✓ Only on clean exit]

Figure 4. Terminal (PuTTY) configuration

4. Click the "Download and Debug" button to download the application to the target.

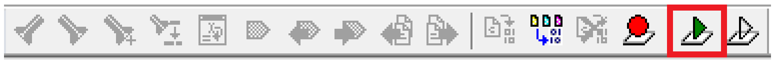
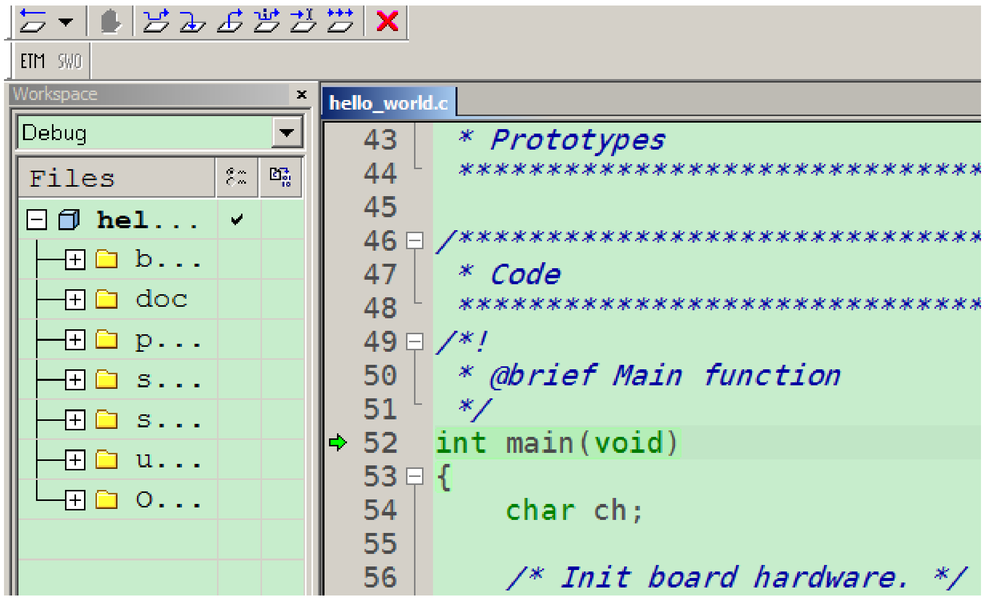


Figure 5. Download and Debug button

5. The application is then downloaded to the target and automatically runs to the main() function.



```
 * Prototypes
 ******************************

/******************************
 * Code
 ******************************
/*!
 * @brief Main function
 */
int main(void)
{
    char ch;

    /* Init board hardware. */
```

Figure 6. Stop at main() when running debugging

6. Run the code by clicking the "Go" button to start the application.

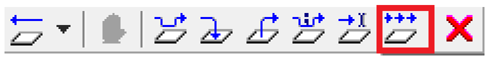
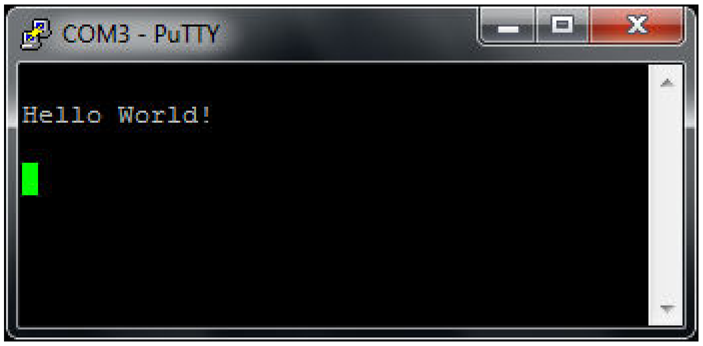


Figure 7. Go button

7. The hello_world application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.



```
Hello World!
```

Figure 8. Text display of the hello_world demo

4 Run a demo using Keil® MDK/μVision

This section describes the steps required to build, run, and debug example applications provided in the Kinetis SDK. The hello_world demo application targeted for the MAPS-KS22 hardware platform is used as an example, although these steps can be applied to any demo or example application in the KSDK.

4.1 Install CMSIS device pack

After the MDK tools are installed, Cortex Microcontroller Software Interface Standard (CMSIS) device packs must be installed to fully support the device from a debug perspective. These packs include things such as memory map information, register definitions and flash programming algorithms. Follow these steps to install the appropriate CMSIS pack.

1. Open the MDK IDE, which is called µVision. In the IDE, select the “Pack Installer” icon.

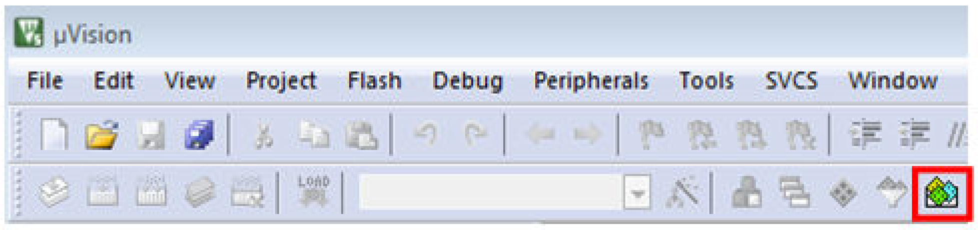


Figure 9. Launch the Pack installer

2. After the installation finishes, close the Pack Installer window and return to the µVision IDE.

4.2 Build an example application

- If not already done, open the desired example application workspace in: <install_dir>/boards/<board_name>/ <*example_type*>/<application_name>/mdk

  The workspace file is named <demo_name>.uvmpw, so for this specific example, the actual path is:

  <install_dir>/boards/mapsks22/demo_apps/hello_world/iar/hello_world.uvmpw
- To build the demo project, select the "Rebuild" button, highlighted in red.

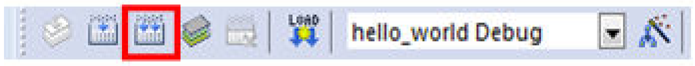


Figure 10. Build the demo

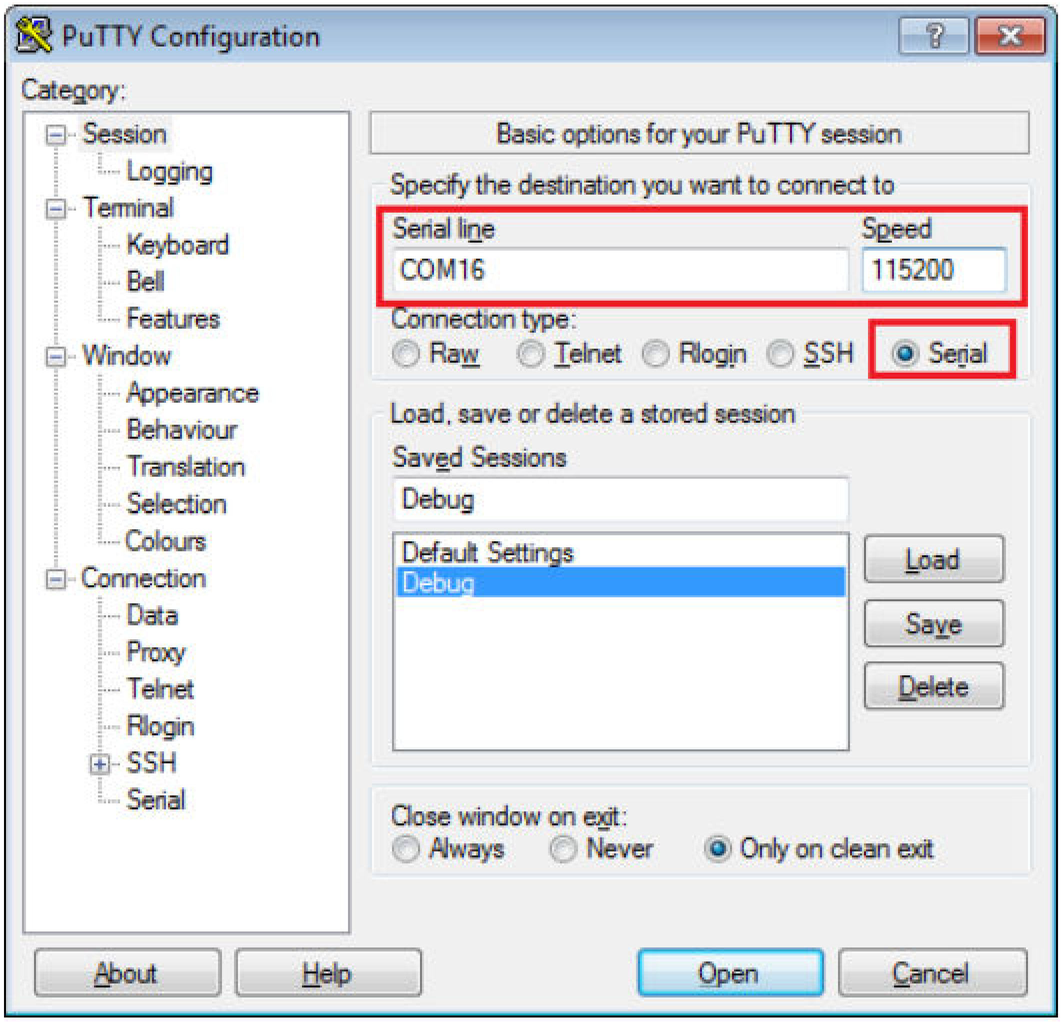
- The build will complete without errors.

4.3 Run an example application

To download and run the application, perform these steps:

1. Reference the table in Appendix B to determine the debug interface that comes loaded on your specific hardware platform.
   - For boards with the CMSIS-DAP/mbed/DAPLink interface, visit mbed Windows serial configuration.
   - For boards with a P&E Micro interface, visit www.pemicro.com/support/downloads_find.cfm and download and install the P&E Micro Hardware Interface Drivers package.
   - For the MRB-KW01 board, visit www.freescale.com/USB2SER to download the serial driver. This board does not support the OpenSDA. Therefore, an external debug probe (such as a J-Link) is required. Steps below referencing the OpenSDA do not apply because there is only a single USB connector for serial output.
2. Connect the development platform to your PC via USB cable between the OpenSDA USB connector and the PC USB connector.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see Appendix A). Configure the terminal with these settings:
   a. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUD variable in board.h file)

b. No parity
c. 8 data bits
d. 1 stop bit


PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

Serial line: COM16

Speed: 115200

Connection type: Raw Telnet Rlogin SSH [✓ Serial]

Load, save or delete a stored session

Saved Sessions: Debug

Default Settings
Debug

Close window on exit: Always Never [✓ Only on clean exit]

Figure 11. Terminal (PuTTY) configurations

4. After the application is properly built, click the "Download" button to download the application to the target.

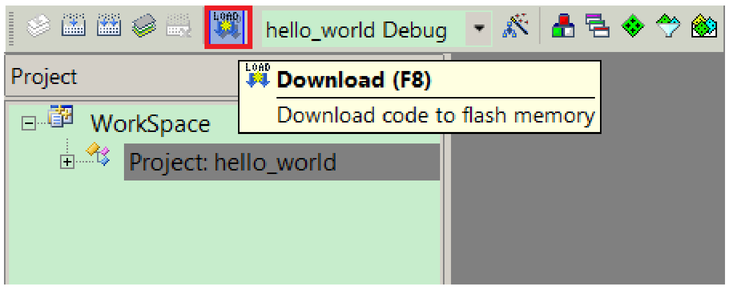

Download (F8)
Download code to flash memory

Figure 12. Download button

5. After clicking the “Download” button, the application downloads to the target and should be running. To debug the application, click the “Start/Stop Debug Session” button, highlighted in red.

6. Run the code by clicking the “Run” button to start the application.

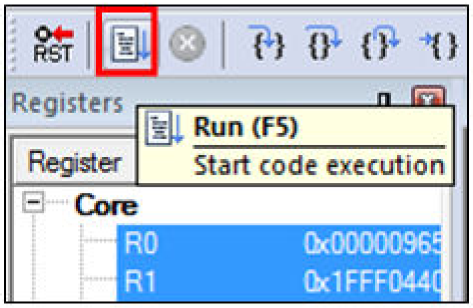
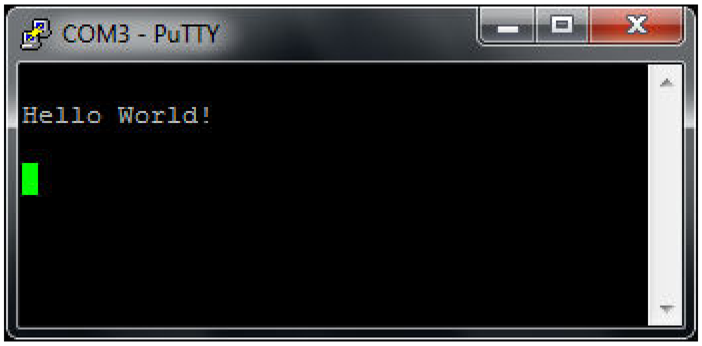


Figure 13. Go button

The hello_world application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.



```
Hello World!
```

Figure 14. Text display of the hello_world demo

5 Run a demo using Kinetis Design Studio IDE

This section describes the steps required to configure Kinetis Design Studio (KDS) IDE to build, run, and debug example applications. The hello_world demo application targeted for the MAPS-KS22 hardware platform is used as an example, though these steps can be applied to any example application in the KSDK.

5.1 Select the workspace location

The first time that KDS IDE launches, it prompts the user to select a workspace location. KDS IDE is built on top of Eclipse, which uses workspace to store information about its current configuration, and in some use cases, source files for the projects in the workspace. The location of the workspace can be anywhere, but it is recommended that the workspace be outside of the KSDK tree.

5.2 Build an example application

NOTE

The steps required for the Linux® OS and Mac® OS are identical to those for the Windows® operating system. The only difference is that the IDE looks slightly different.

1. Select "File->Import" from the KDS IDE menu. In the window that appears, expand the "Project of Projects" folder and select "Existing Projects Sets". Then, click the "Next" button.

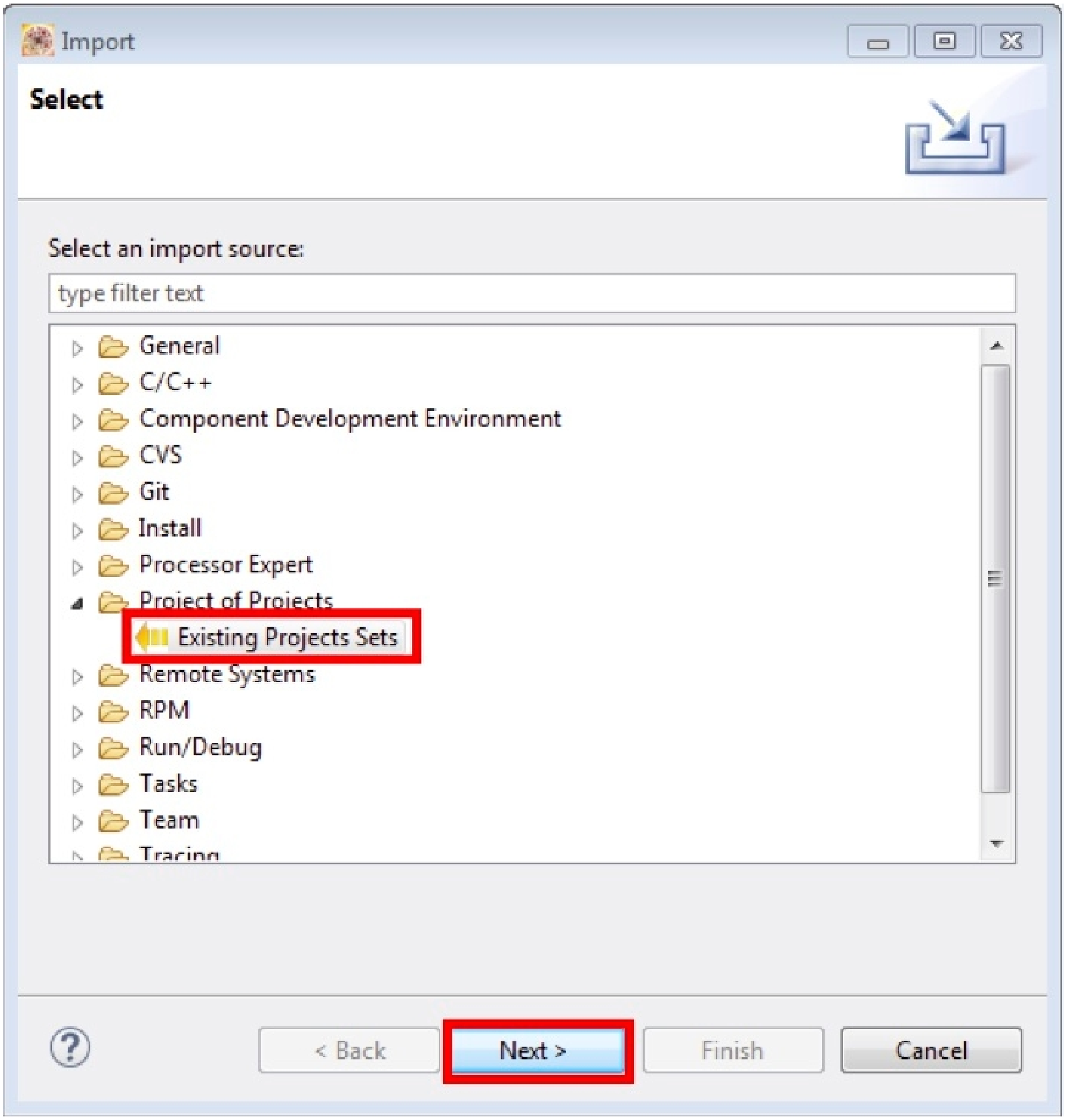


Figure 15. Selection of the correct import type in KDS IDE

2. Click the "Browse" button next to the "Import from file:" option.

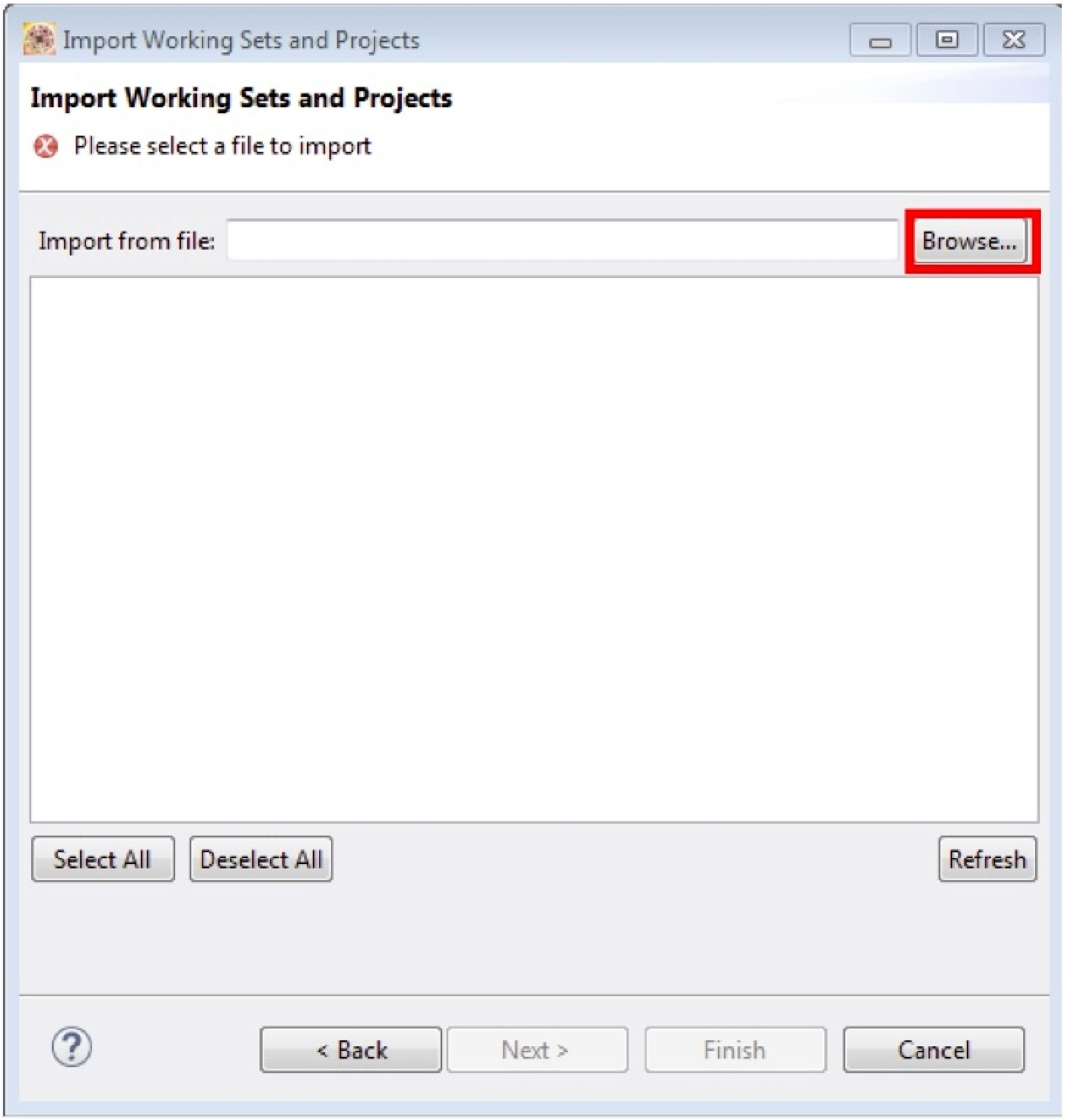

Import Working Sets and Projects
Import Working Sets and Projects
Please select a file to import
Import from file:
Browse...
Select All
Deselect All
Refresh
< Back
Next >
Finish
Cancel

Figure 16. Projects directory selection window

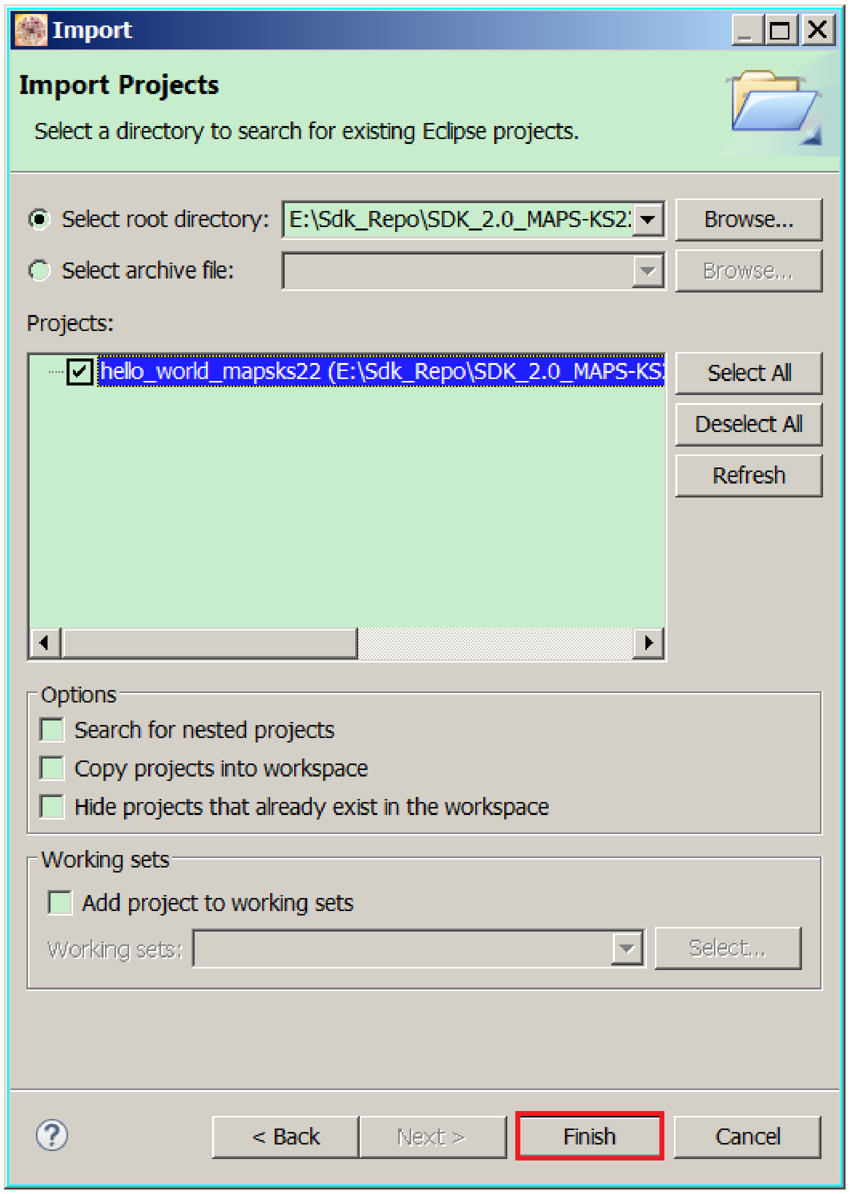
3. Point to the example application project, which can be found using this path:

   <install_dir>/boards/<board_name>/<example_type>/<application_name>/kds

   For this example, the specific location is:

   <install_dir>/boards/mapsks22/demo_apps/hello_world/kds

4. After pointing to the correct directory, your "Import Working Sets and Projects" window should look like the figure below. Click the "Finish" button.



Select root directory: E:\Sdk_Repo\SDK_2.0_MAPS-KS2

Select archive file:

Projects:

- [✓] hello_world_mapsks22 (E:\Sdk_Repo\SDK_2.0_MAPS-KS

Options

- [ ] Search for nested projects
- [ ] Copy projects into workspace
- [ ] Hide projects that already exist in the workspace

Working sets

- [ ] Add project to working sets

Working sets:

Figure 17. Select KS22F12 platform library project

5. There are two project configurations (build targets) supported for each KSDK project:
   - Debug – Compiler optimization is set to low, and debug information is generated for the executable. This target should be selected for development and debug.
   - Release – Compiler optimization is set to high, and debug information is not generated. This target should be selected for final application deployment.
6. Choose the appropriate build target, "Debug" or "Release", by clicking the downward facing arrow next to the hammer icon, as shown below. For this example, select the "Debug" target.

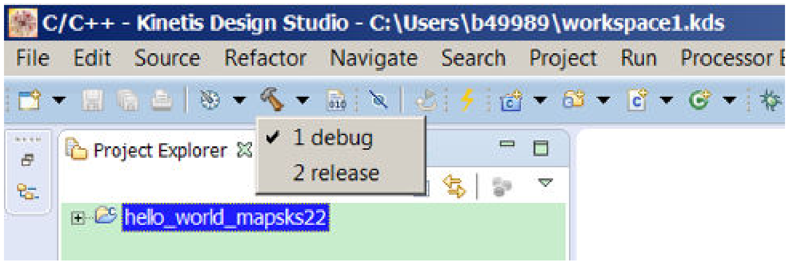

C/C++ - Kinetis Design Studio - C:\Users\b49989\workspace1.kds
File Edit Source Refactor Navigate Search Project Run Processor E
Project Explorer
✔ 1 debug
2 release
hello_world_mapsks22

Figure 18. Selection of the build target in KDS IDE

The library starts building after the build target is selected. To rebuild the library in the future, click the hammer icon (assuming the same build target is chosen).

5.3 Run an example application

NOTE

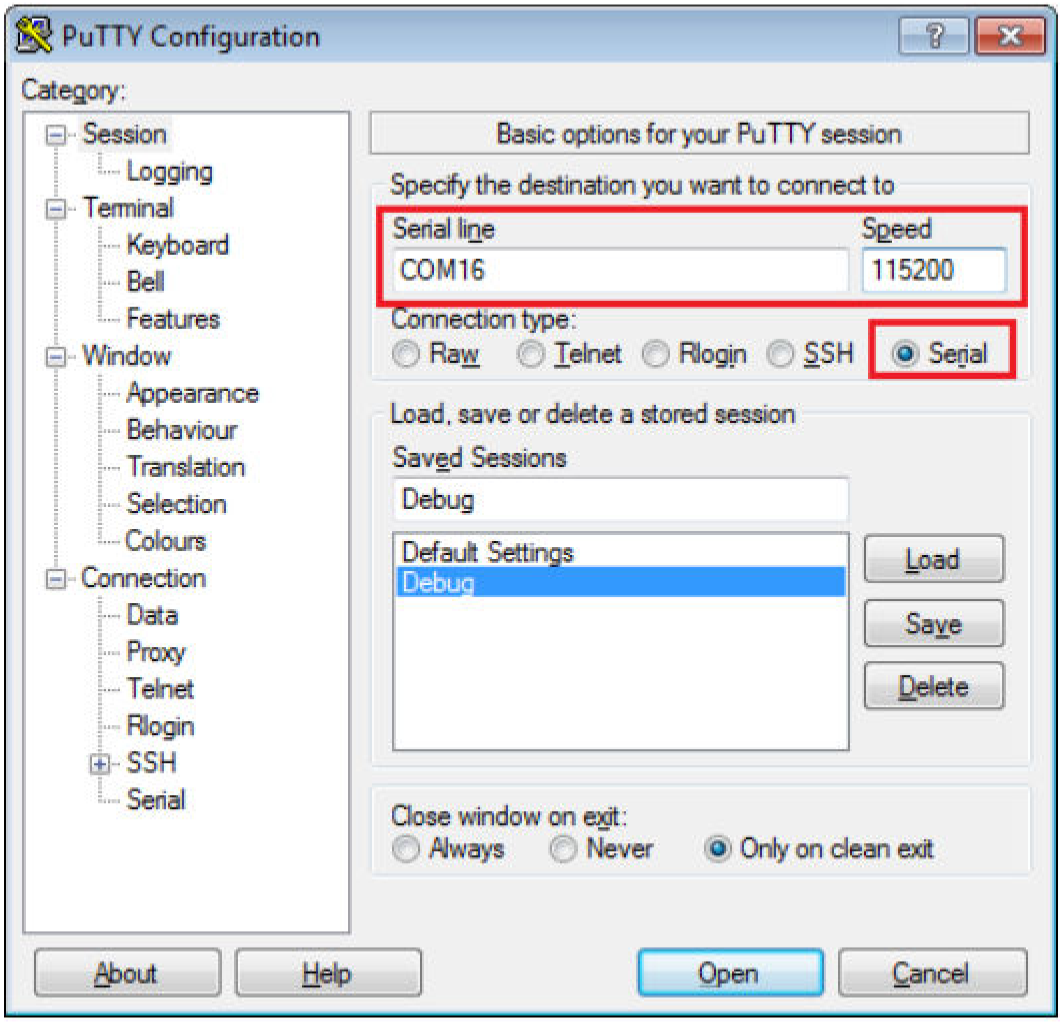
The steps required for the Linux OS and Mac OS are identical to those for the Windows operating system. The only difference is that the IDE looks slightly different. Any platform-specific steps are listed accordingly.

To download and run the application, perform these steps:

1. Reference the table in Appendix B to determine the debug interface that comes loaded on your specific hardware platform.
2. Connect the development platform to your PC via USB cable between the OpenSDA USB connector and the PC USB connector.
3. In the Windows operating system environment, open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see Appendix A). For Linux OS, open your terminal application and connect to the appropriate device.

   Configure the terminal with these settings:

   a. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUD variable in board.h file)
   b. No parity
   c. 8 data bits
   d. 1 stop bit



Specify the destination you want to connect to

Serial line: COM16

Speed: 115200

Connection type: Raw Telnet Rlogin SSH [✓ Serial]

Load, save or delete a stored session

Saved Sessions: Debug

Default Settings
Debug

Close window on exit: Always Never [✓ Only on clean exit]

Figure 19. Terminal (PuTTY) configurations

4. For Linux OS users only, run the following commands in your terminal. These install libudev onto your system, which is required by KDS IDE to launch the debugger.

```
user@ubuntu:~$ sudo apt-get install libudev-dev libudev1
```

```
user@ubuntu:~$ sudo ln -s /usr/lib/x86_64-linux-gnu/libudev.so /usr/lib/x86_64-linux-
gnu/libudev.so.0
```

5. Ensure that the debugger configuration is correct for the target you're attempting to connect to. Consult Appendix B for more information about the default debugger application on the various hardware platforms supported by the KSDK.
   a. To check the available debugger configurations, click the small downward arrow next to the green “Debug” button and select “Debug Configurations”.

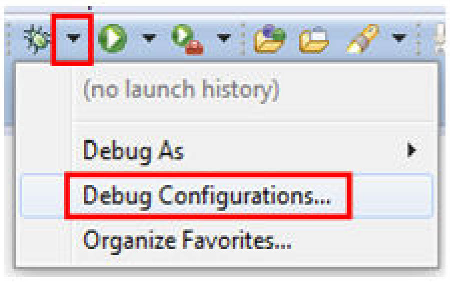


Figure 20. Debug Configurations dialog button

b. In the Debug Configurations dialog box, select the debug configuration that corresponds to the hardware platform you're using.

   After selecting the debugger interface, click the "Debug" button to launch the debugger.

6. The application is downloaded to the target and automatically run to main():

7. Start the application by clicking the "Resume" button:

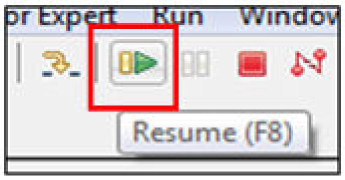
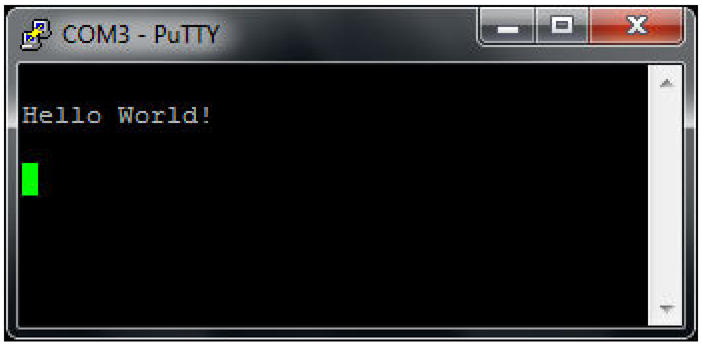


Figure 21. Resume button

The hello_world application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.



```
Hello World!
```

Figure 22. Text display of the hello_world demo

6 Run a demo using Atollic® TrueSTUDIO®

This section describes the steps to configure Atollic TrueSTUDIO to build, run, and debug example applications provided in the KSDK. The hello_world example application targeted for the MAPS-KS22 hardware platform used as an example, though these steps can be applied to any demo or example application in the KSDK.

6.1 Select the workspace location

The first time that TrueSTUDIO launches, it prompts the user to select a workspace location. TrueSTUDIO uses Eclipse, which uses workspace to store information about its current configuration, and in some use cases, source files for the projects in the workspace. The location of the workspace can be anywhere, but it is recommended that the workspace be outside of the KSDK tree.

6.2 Build an example application

1. Select “File -> Import” from the TrueSTUDIO menu. Expand the “General” folder and select “Existing Projects into Workspace”. Then, click the “Next” button.

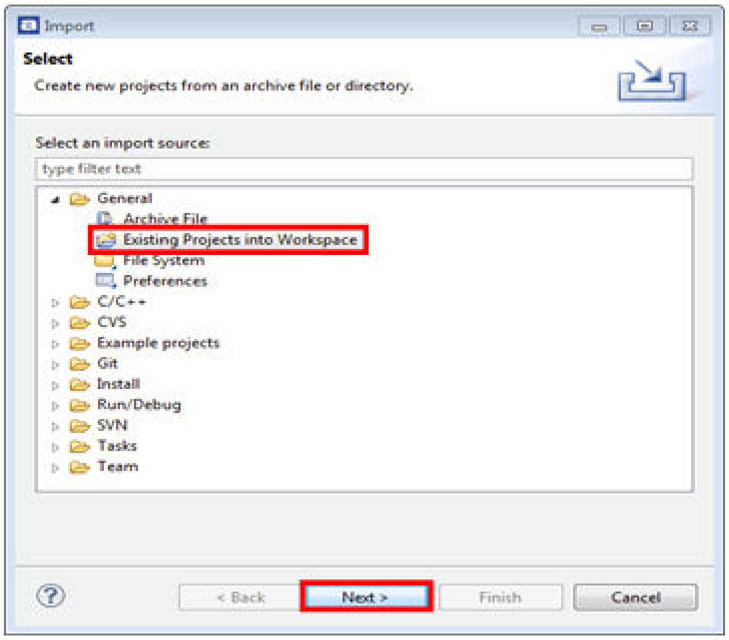


Figure 23. Selection of the correct import type in TrueSTUDIO

2. Click the “Browse” button next to the “Select root directory:” option.

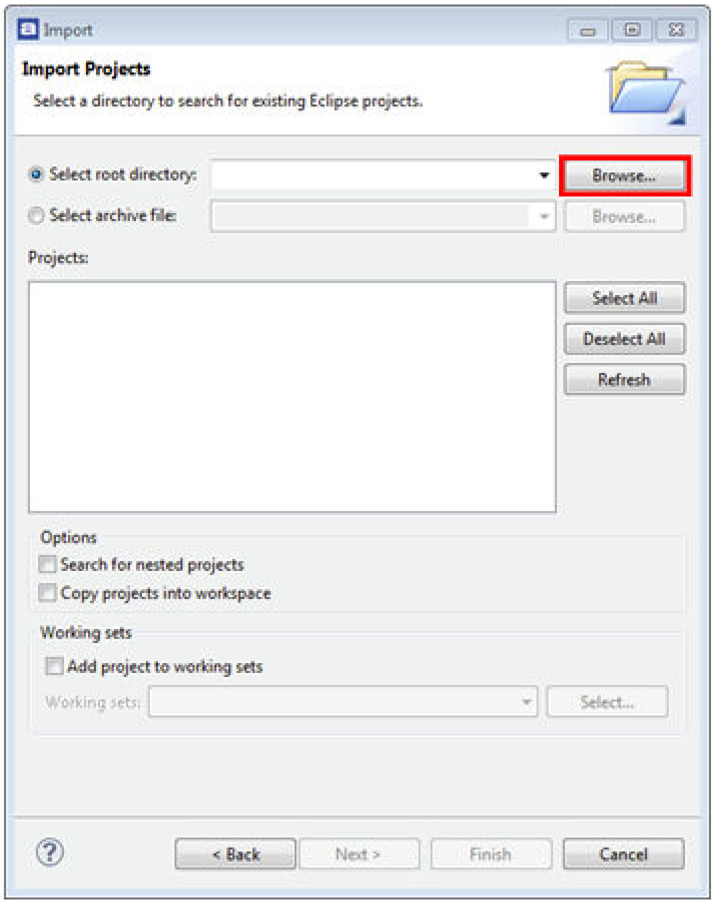


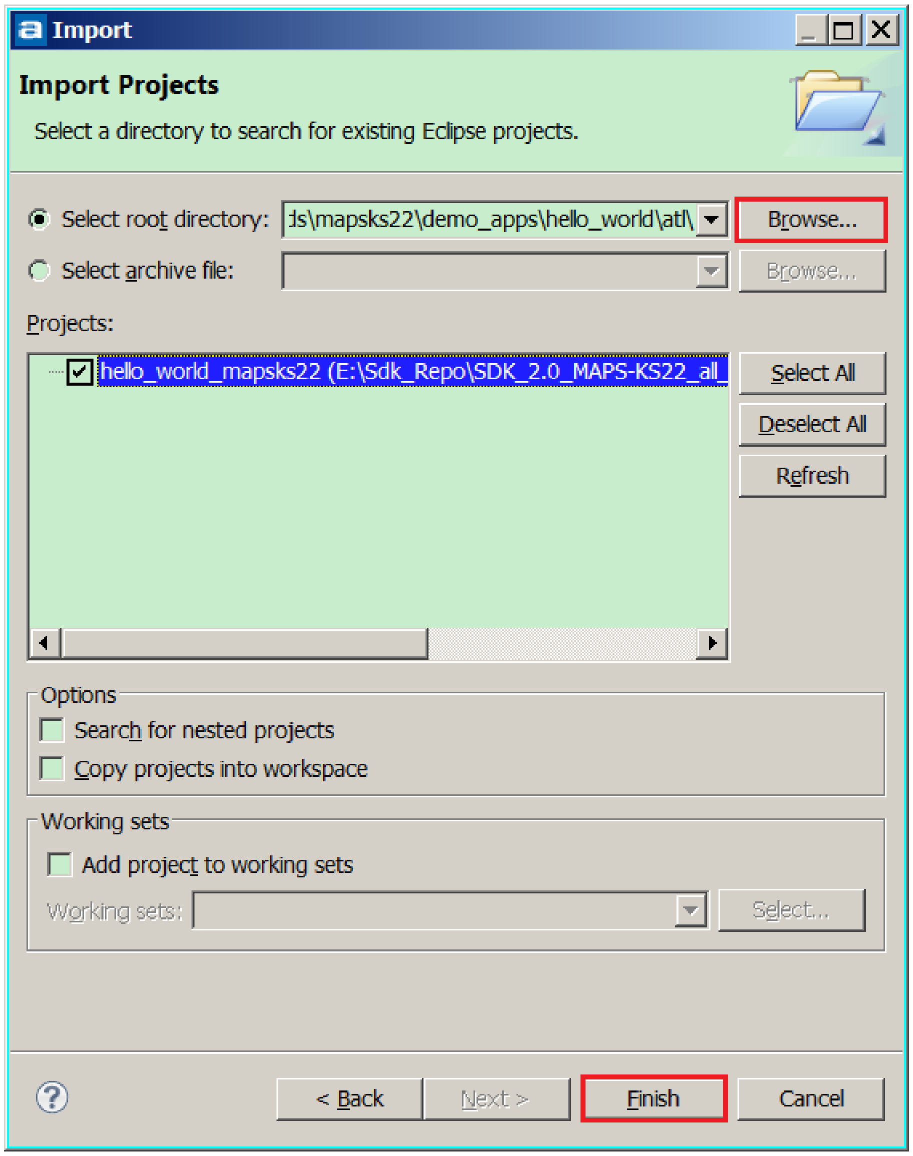
Figure 24. Projects directory selection window

3. Point to the example application project for the appropriate device, which can be found using this path:

   <install_dir>/boards/<board_name>/<example_type>/<application_name>/atl

   For this example, the specific location is:

   <install_dir>/boards/mapsks22/demo_apps/hello_world/atl
4. After pointing to the correct directory, your “Import Projects” window should look like this figure. Click the “Finish” button.


Import

Import Projects

Select a directory to search for existing Eclipse projects.

Select root directory: ds\mapsks22\demo_apps\hello_world\atl\

Select archive file:

Projects:

☑ hello_world_mapsks22 (E:\Sdk_Repo\SDK_2.0_MAPS-KS22_all_

Options

☐ Search for nested projects

☐ Copy projects into workspace

Working sets

☐ Add project to working sets

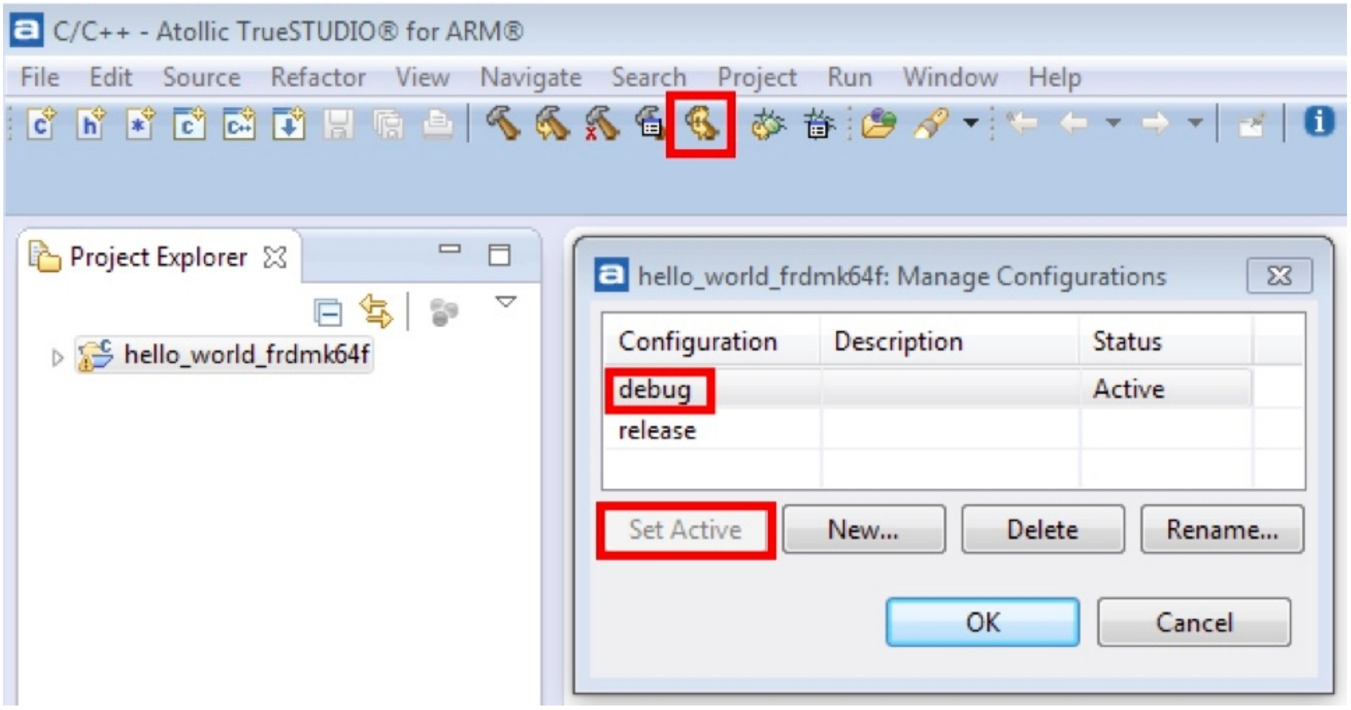
Working sets:

Figure 25. Select the KS22F12 platform library project

NOTE

Do not select the "Copy projects..." option.

5. There are two project configurations (build targets) supported for each KSDK project:
    - Debug – Compiler optimization is set to low, and debug information is generated for the executable. This target should be selected for development and debug.
    - Release – Compiler optimization is set to high, and debug information is not generated. This target should be selected for final application deployment.
6. Choose the appropriate build target, “Debug” or “Release”, by clicking the “Manage build configurations” icon, as shown below. For this example, select the “Debug” target and click “Set Active”. Since the default configuration is to use the Debug target, there should not be a change required.


C/C++ - Atollic TrueSTUDIO® for ARM®
File Edit Source Refactor View Navigate Search Project Run Window Help
Project Explorer
hello_world_frdmk64f
hello_world_frdmk64f: Manage Configurations
Set Active
New...
Delete
Rename...
OK
Cancel

Configuration	Description	Status
debug		Active
release		

Figure 26. Selection of build target in TrueSTUDIO

7. Click the "Build" icon to build the application.

6.3 Run an example application

The Atollic tools require either a J-Link or P&E Micro debug interface. As a result, some hardware platforms require an update to the OpenSDA debug firmware found on the board. To determine the default debug interface of your board, see Appendix B. If the default interface is not J-Link or P&E Micro, see Appendix C for instructions on how to install one of these debug interfaces.

This section describes steps to run a demo application using a J-Link debugger, although the P&E Micro interface is also supported.

In order to perform this exercise with the J-Link interface, two things must be done:

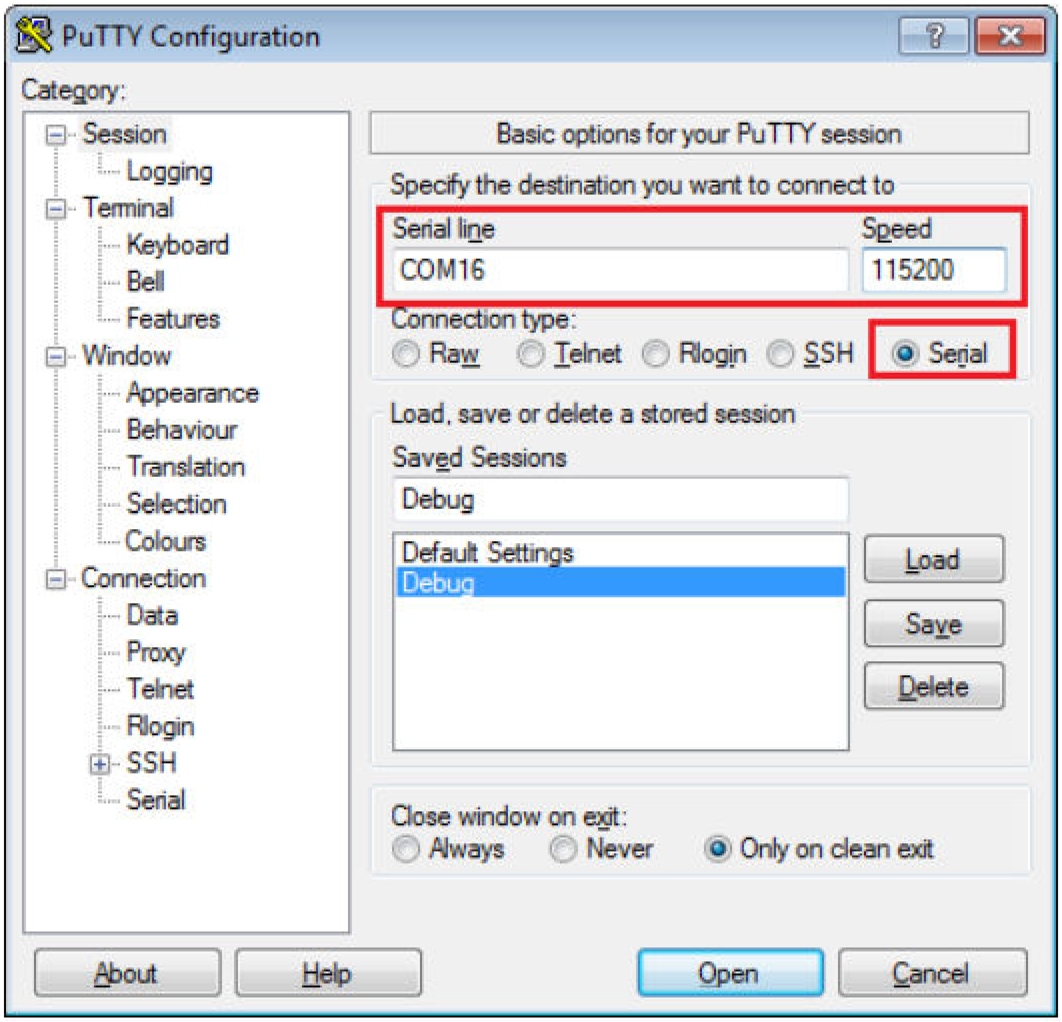
- Install the J-Link software (drivers and utilities), which can be downloaded from segger.com/downloads.html.
- Make sure that either:
  - The OpenSDA interface on your board is programmed with the J-Link OpenSDA firmware. To determine if your board supports OpenSDA, see Appendix B. For instructions on reprogramming the OpenSDA interface, see Appendix C. If your board does not support OpenSDA, then a standalone J-Link pod is required.
  - A standalone J-Link pod is connected to the debug interface of your board. Note that some hardware platforms require hardware modification in order to function correctly with an external debug interface.

The P&E Micro interface can also be used. To use this interface:

- Install the P&E Micro Hardware Interface Drivers, which can be downloaded from www.pemicro.com/support/downloads_find.cfm.
- If your board does not come loaded with a P&E Micro interface, if supported, reprogram the OpenSDA interface with P&E Micro OpenSDA firmware. To determine if your board supports OpenSDA, see Appendix B. For instructions on reprogramming the OpenSDA interface, see Appendix C.

After the debug interface is configured and ready to use to download and run the application:

1. Connect the development platform to your PC via USB cable between the OpenSDA USB connector (may be named OSJTAG for some boards) and the PC USB connector.
2. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see Appendix A). Configure the terminal with these settings:
   a. 115200 or baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUD variable in board.h file)
   b. No parity
   c. 8 data bits
   d. 1 stop bit



Specify the destination you want to connect to
Serial line: COM16
Speed: 115200
Connection type: Raw Telnet Rlogin SSH [✓ Serial]
Load, save or delete a stored session
Saved Sessions: Debug
Default Settings
Debug
Close window on exit: Always Never [✓ Only on clean exit]

Figure 27. Terminal (PuTTY) configurations

3. Ensure that the debugger configuration is correct for the target you are attempting to connect to.
   a. To check the debugger configurations, click the “Configure Debug” icon.

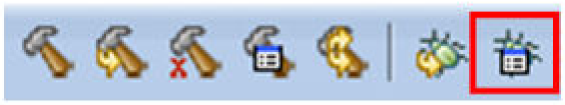


Figure 28. Debug configurations dialog button

b. In the Debug Configurations window, select debug configuration that corresponds to the hardware platform you’re using. The Atollic tools require either a J-Link or P&E Micro debug interface, so some hardware platforms require an update to the OpenSDA debug firmware. To determine the default debug interface of your board, see Appendix B. If the default interface is not J-Link or P&E Micro, see Appendix C for instructions on how to install one of these debug interfaces.

   Important: This example assumes the J-Link interface has been installed on the .

c. Select the J-Link “Debug” interface and click the “Debug” button.

4. The application is downloaded to the target and automatically runs to main():

5. Run the code by clicking the "Resume" button to start the application.

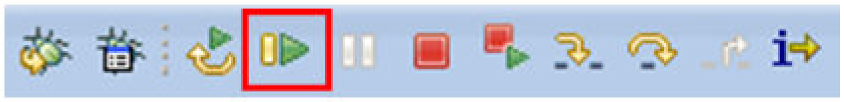
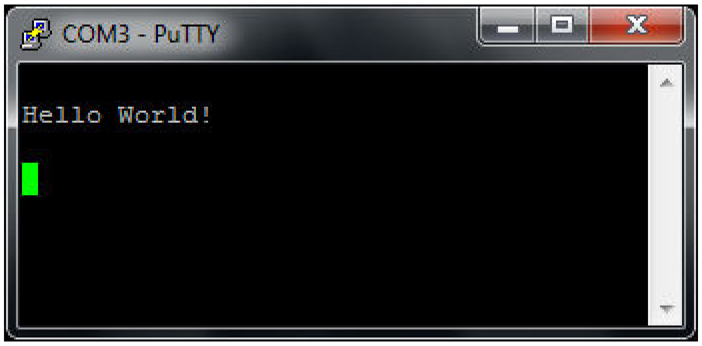


Figure 29. Resume button

The hello_world application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.



```
Hello World!
```

Figure 30. Text display of the hello_world demo

7 Run a demo using ARM GCC

This section describes the steps to configure the command line ARM GCC tools to build, run, and debug demo applications and necessary driver libraries provided in the KSDK. The hello_world demo application targeted for the is used as an example, though these steps can be applied to any board, demo or example application in the KSDK.

7.1 Set up toolchain

This section contains the steps to install the necessary components required to build and run a KSDK demo application with the ARM GCC toolchain, as supported by the KSDK. There are many ways to use ARM GCC tools, but this example focuses on a Windows operating system environment. Though not discussed here, ARM GCC tools can also be used with both Linux OS and Mac OSX.

7.1.1 Install GCC ARM Embedded tool chain

Download and run the installer from launchpad.net/gcc-arm-embedded. This is the actual toolset (i.e., compiler, linker, etc.). The GCC toolchain should correspond to the latest supported version, as described in the

7.1.2 Install MinGW

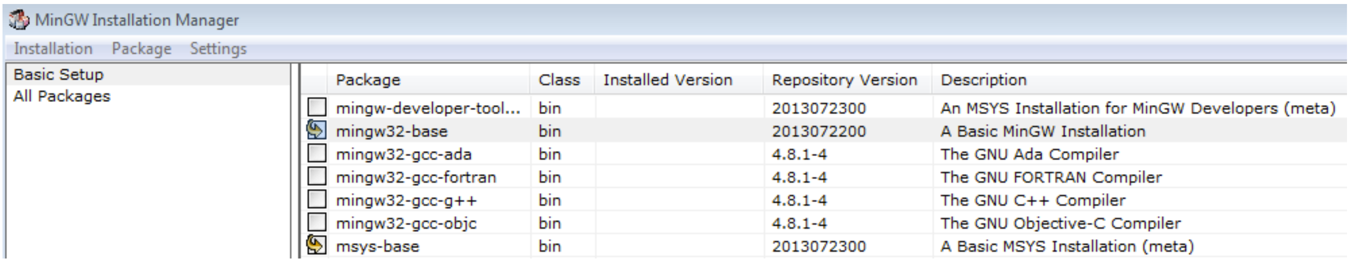
The Minimalist GNU for Windows (MinGW) development tools provide a set of tools that are not dependent on third party C-Runtime DLLs (such as Cygwin). The build environment used by the KSDK does not utilize the MinGW build tools, but does leverage the base install of both MinGW and MSYS. MSYS provides a basic shell with a Unix-like interface and tools.

1. Download the latest MinGW mingw-get-setup installer from sourceforge.net/projects/mingw/files/Installer/.
2. Run the installer. The recommended installation path is C:\MinGW, however, you may install to any location.

NOTE

The installation path cannot contain any spaces.

3. Ensure that the “mingw32-base” and “msys-base” are selected under Basic Setup.



	Package	Class	Installed Version	Repository Version	Description
☐	mingw-developer-tool...	bin		2013072300	An MSYS Installation for MinGW Developers (meta)
✓	mingw32-base	bin		2013072200	A Basic MinGW Installation
☐	mingw32-gcc-ada	bin		4.8.1-4	The GNU Ada Compiler
☐	mingw32-gcc-fortran	bin		4.8.1-4	The GNU FORTRAN Compiler
☐	mingw32-gcc-g++	bin		4.8.1-4	The GNU C++ Compiler
☐	mingw32-gcc-objc	bin		4.8.1-4	The GNU Objective-C Compiler
✓	msys-base	bin		2013072300	A Basic MSYS Installation (meta)

Figure 31. Setup MinGW and MSYS

4. Click “Apply Changes” in the “Installation” menu and follow the remaining instructions to complete the installation.

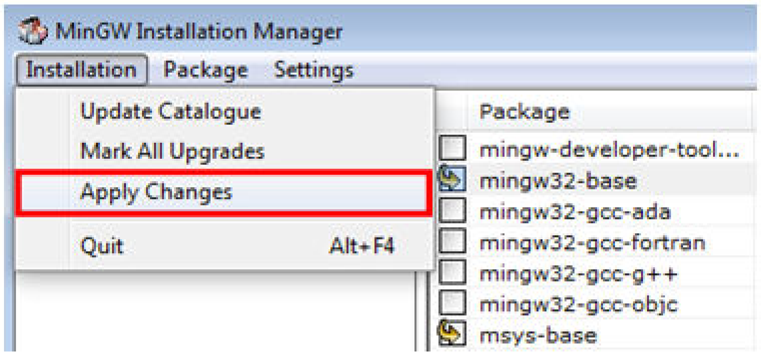


Figure 32. Complete MinGW and MSYS installation

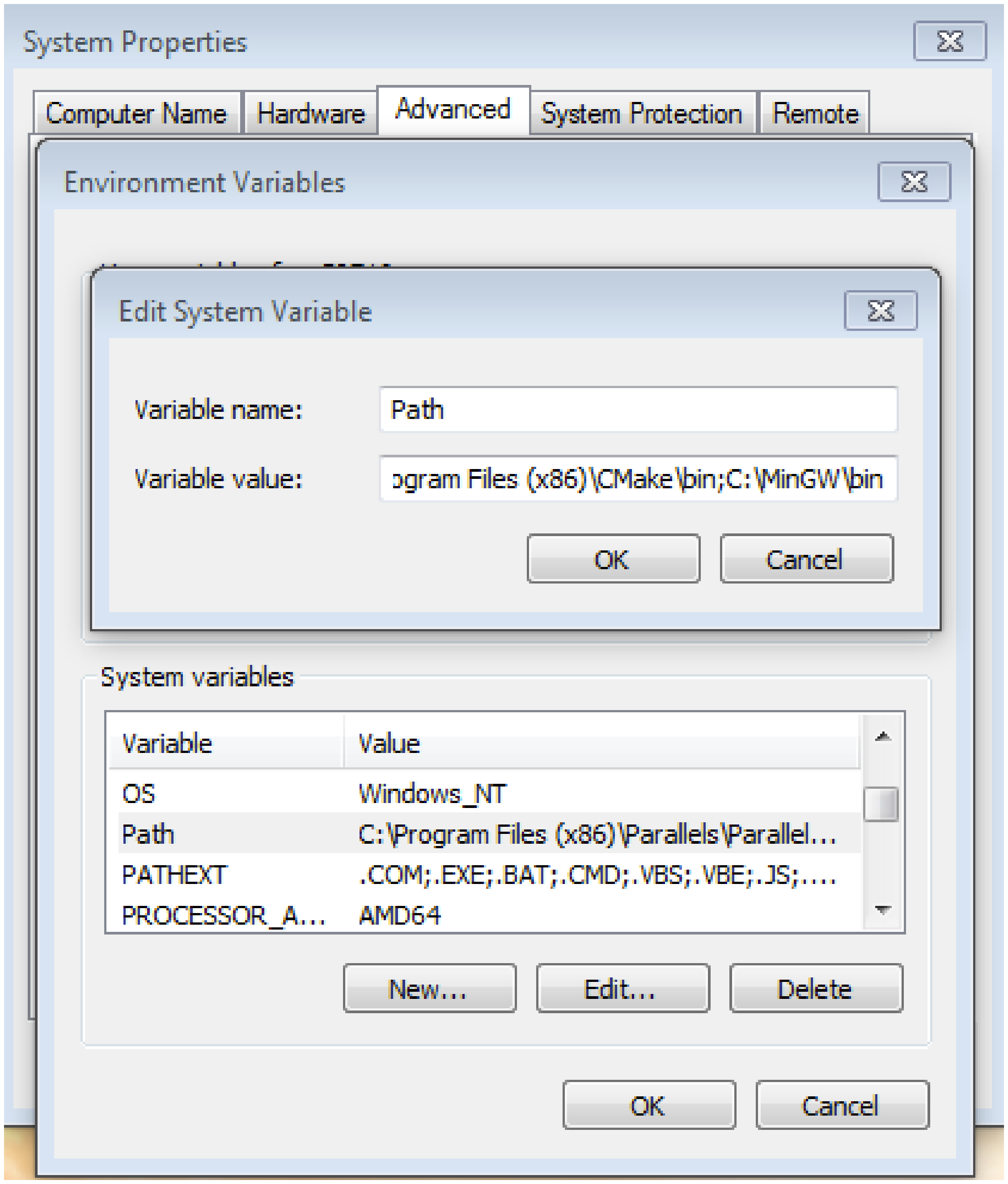
5. Add the appropriate item to the Windows operating system path environment variable. It can be found under *Control Panel -> System and Security -> System -> Advanced System Settings* in the "Environment Variables..." section. The path is:

<mingw_install_dir>\bin

Assuming the default installation path, C:\MinGW, an example is shown below. If the path is not set correctly, the toolchain does not work.

NOTE

If you have "C:\MinGW\msys\x.x\bin" in your PATH variable (as required by KSDK 1.0.0), remove it to ensure that the new GCC build system works correctly.


Edit System Variable

Variable name: Path

Variable value: ogram Files (x86)\CMake\bin;C:\MinGW\bin

System variables

Variable	Value
OS	Windows_NT
Path	C:\Program Files (x86)\Parallels\Parallel...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
PROCESSOR_A...	AMD64

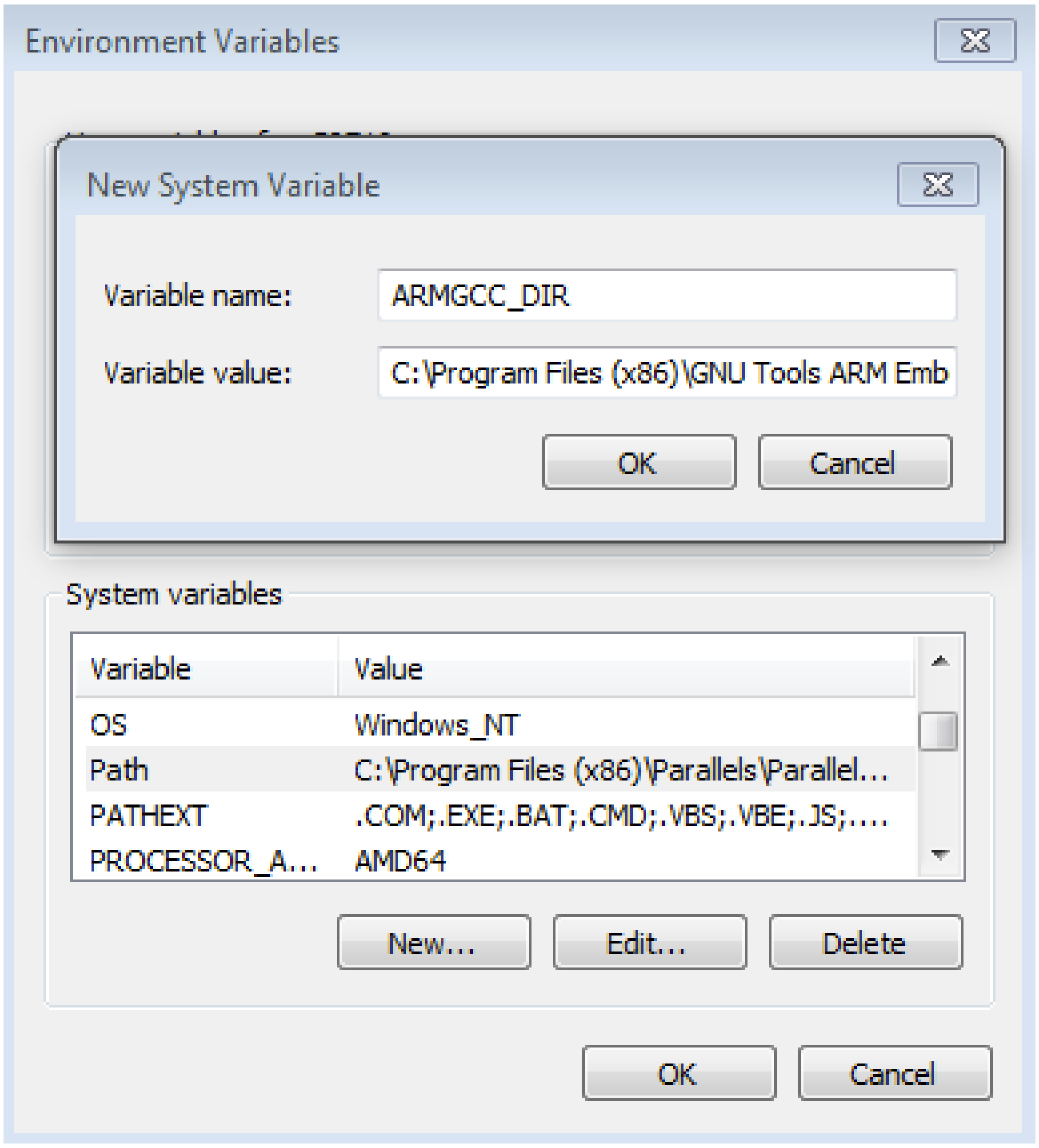
Figure 33. Add Path to systems environment

7.1.3 Add a new system environment for ARMGCC_DIR

Create a new *system* environment variable and name it ARMGCC_DIR. The value of this variable should point to the ARM GCC Embedded tool chain installation path, which, for this example, is:

C:\Program Files (x86)\GNU Tools ARM Embedded\4.9 2015q3

Reference the installation folder of the GNU ARM GCC Embedded tools for the exact path name of your installation.



Variable name: ARMGCC_DIR

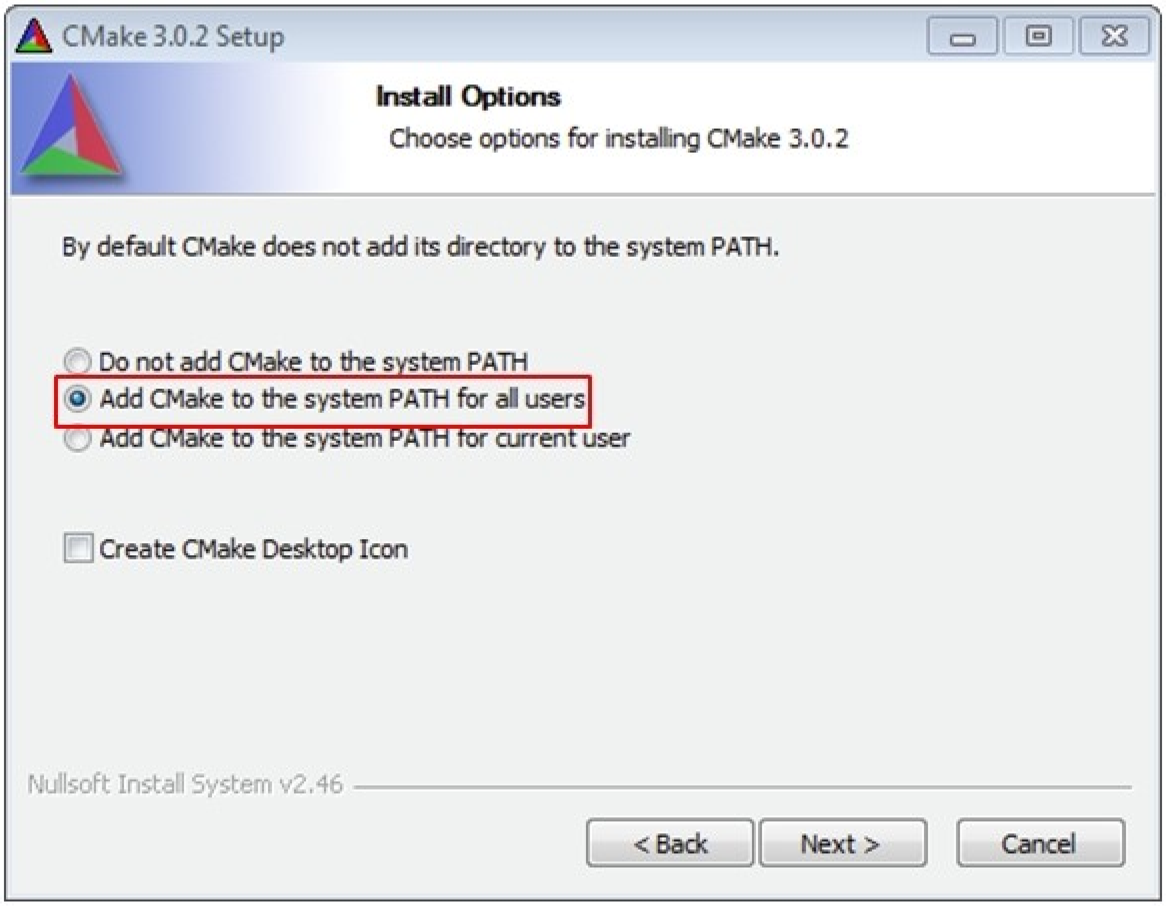
Variable value: C:\Program Files (x86)\GNU Tools ARM Emb

Variable	Value
OS	Windows_NT
Path	C:\Program Files (x86)\Parallels\Parallel...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
PROCESSOR_A...	AMD64

Figure 34. Add ARMGCC_DIR system variable

7.1.4 Install CMake

1. Download CMake 3.0.x from www.cmake.org/cmake/resources/software.html.
2. Install CMake, ensuring that the option "Add CMake to system PATH" is selected when installing. The user chooses to select whether it is installed into the PATH for all users or just the current user. In this example, it is installed for all users.


CMake 3.0.2 Setup
Install Options
Choose options for installing CMake 3.0.2

By default CMake does not add its directory to the system PATH.

Do not add CMake to the system PATH
[✓ Add CMake to the system PATH for all users]
Add CMake to the system PATH for current user

Create CMake Desktop Icon

Nullsoft Install System v2.46
< Back
Next >
Cancel

Figure 35. Install CMake

3. Follow the remaining instructions of the installer.
4. You may need to reboot your system for the PATH changes to take effect.

7.2 Build an example application

To build an example application, follow these steps.

1. Open a GCC ARM Embedded tool chain command window. To launch the window, from the Windows operating system Start menu, go to “Programs -> GNU Tools ARM Embedded <version>” and select “GCC Command Prompt”.

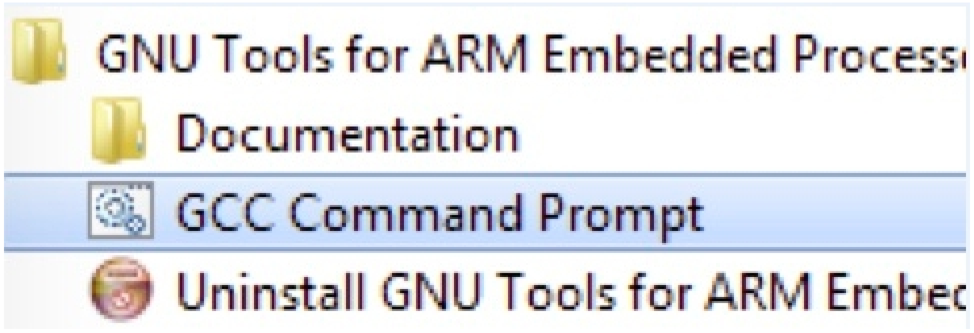

GNU Tools for ARM Embedded Process
Documentation
GCC Command Prompt
Uninstall GNU Tools for ARM Embec

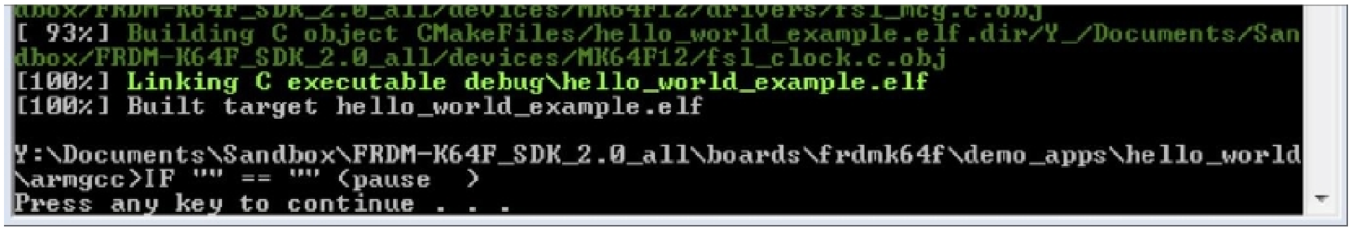
Figure 36. Launch command prompt

2. Change the directory to the example application project directory, which has a path like this:

<install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc

For this example, the exact path is: *<install_dir>/examples/mapsks22/demo_apps/hello_world/armgcc*

3. Type “build_debug.bat” on the command line or double click on the "build_debug.bat" file in Windows Explorer to perform the build. The output is shown in this figure:



```
[ 93%] Building C object CMakeFiles/hello_world_example.elf.dir/Y_/Documents/San
dbox/FRDM-K64F_SDK_2.0_all/devices/MK64F12/fsl_clock.c.obj
[100%] Linking C executable debug\hello_world_example.elf
[100%] Built target hello_world_example.elf

Y:\Documents\Sandbox\FRDM-K64F_SDK_2.0_all\boards\frdmk64f\demo_apps\hello_world
\armgcc>IF "" == "" (pause  )
Press any key to continue . . .
```

Figure 37. hello_world demo build successful

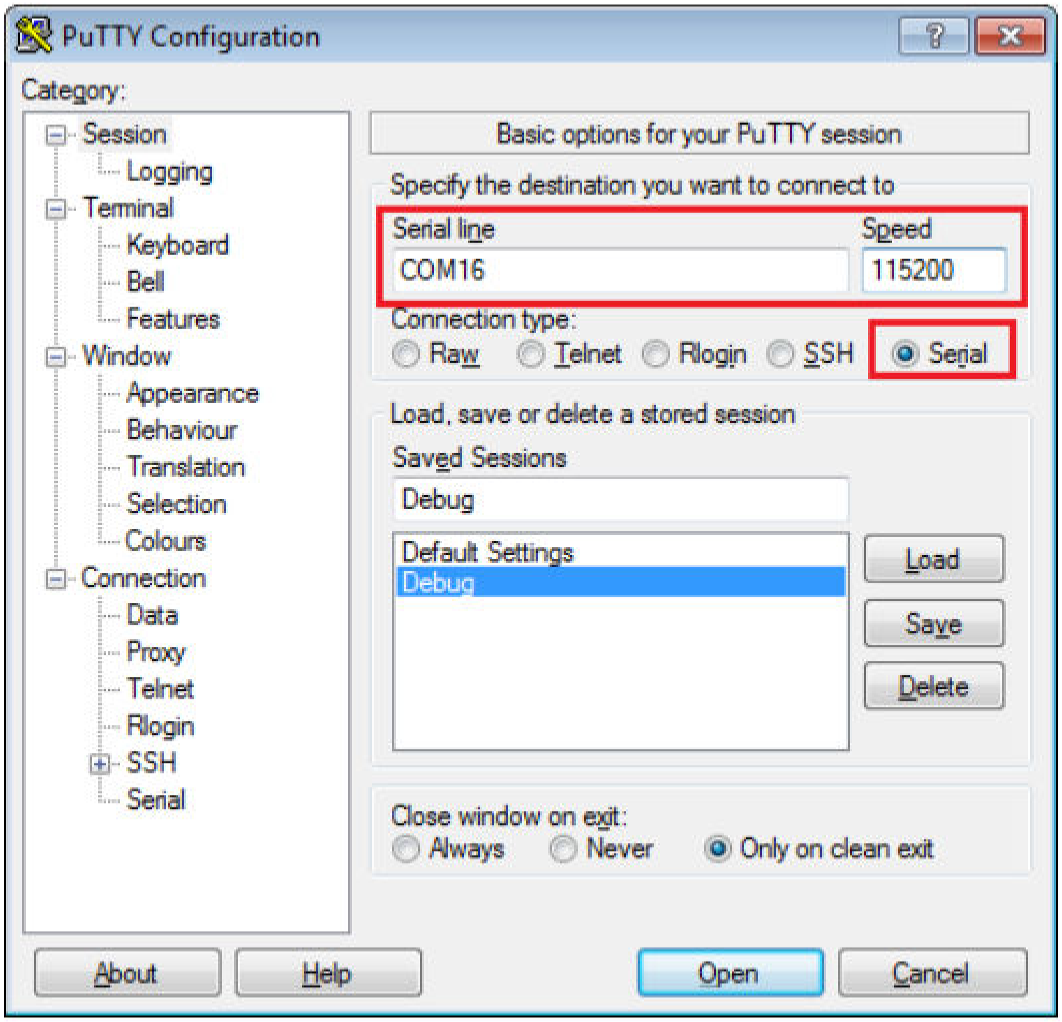
7.3 Run an example application

This section describes steps to run a demo application using J-Link GDB Server application. To perform this exercise, two things must be done:

- Make sure that either:
  - The OpenSDA interface on your board is programmed with the J-Link OpenSDA firmware. To determine if your board supports OpenSDA, see Appendix B. For instructions on reprogramming the OpenSDA interface, see Appendix C. If your board does not support OpenSDA, then a standalone J-Link pod is required.
  - You have a standalone J-Link pod that is connected to the debug interface of your board. Note that some hardware platforms require hardware modification in order to function correctly with an external debug interface.

After the J-Link interface is configured and connected, follow these steps to download and run the demo application:

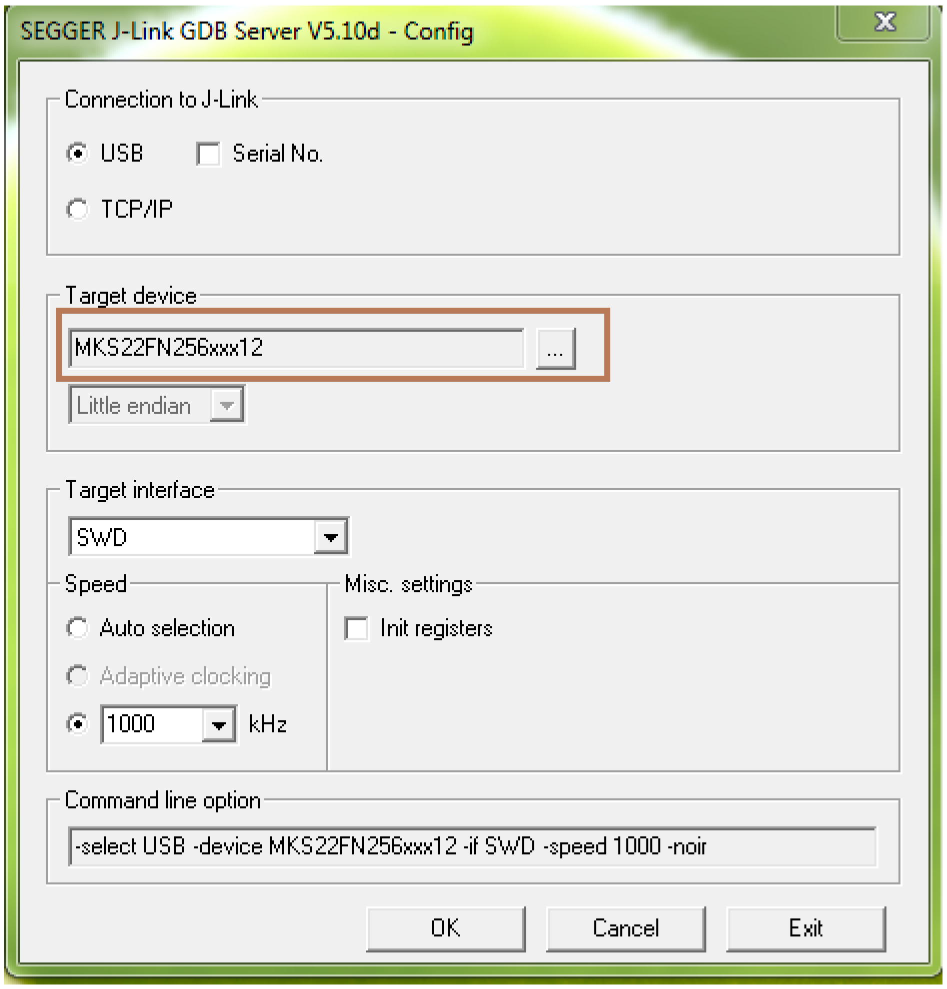
1. Connect the development platform to your PC via USB cable between the OpenSDA USB connector and the PC USB connector. If using a standalone J-Link debug pod, also connect it to the SWD/JTAG connector of the board.
2. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see Appendix A). Configure the terminal with these settings:
   a. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUD variable in board.h file)
   b. No parity
   c. 8 data bits
   d. 1 stop bit



Specify the destination you want to connect to
Serial line: COM16
Speed: 115200
Connection type: Raw Telnet Rlogin SSH [✓ Serial]
Load, save or delete a stored session
Saved Sessions: Debug
Default Settings
Debug
Close window on exit: Always Never [✓ Only on clean exit]

Figure 38. Terminal (PuTTY) configurations

3. Open the J-Link GDB Server application. Assuming the J-Link software is installed, the application can be launched by going to the Windows operating system Start menu and selecting “Programs -> SEGGER -> J-Link <version> J-Link GDB Server”.
4. Modify the settings as shown below. The target device selection chosen for this example is the MKS22FN256xxx12.


SEGGER J-Link GDB Server V5.10d - Config

Connection to J-Link

[✓ USB] Serial No.

TCP/IP

Target device

MKS22FN256xxx12

Little endian

Target interface

SWD

Speed

Auto selection

Adaptive clocking

[✓ 1000] kHz

Misc. settings

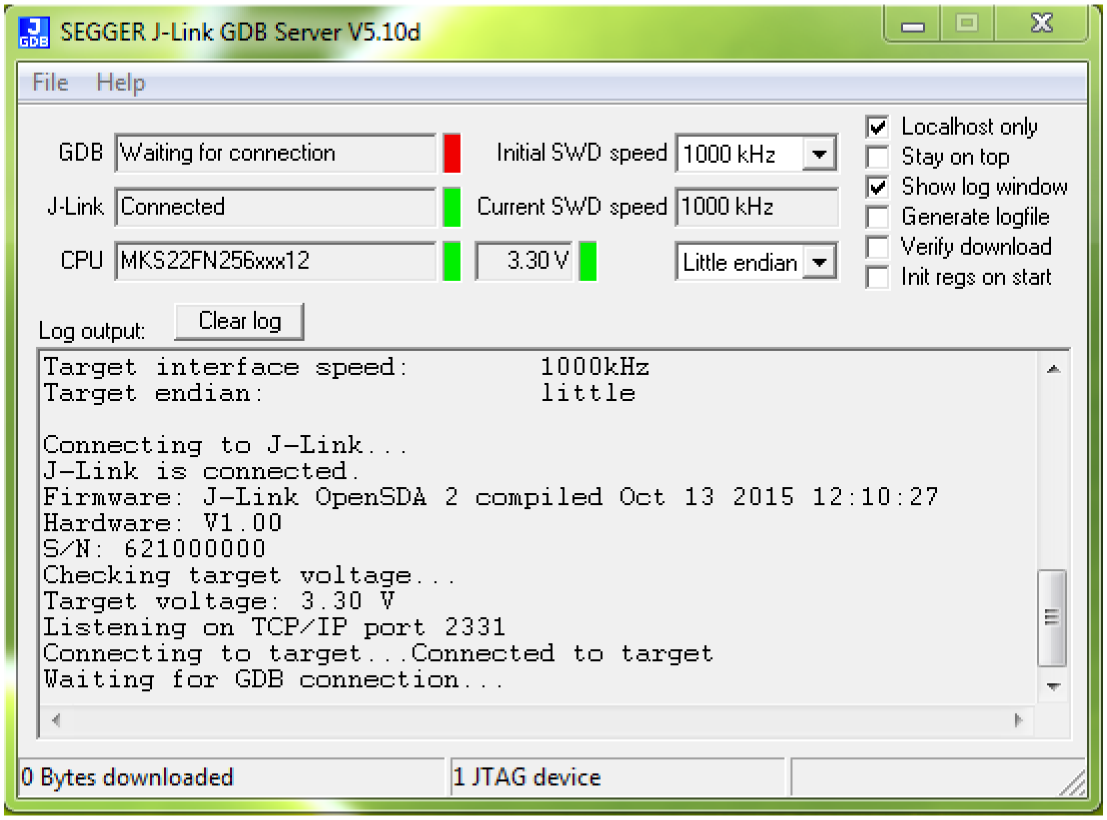
Init registers

Command line option

-select USB -device MKS22FN256xxx12 -if SWD -speed 1000 -noir

Figure 39. SEGGER J-Link GDB Server configuration

5. After it is connected, the screen should resemble this figure:



GDB: Waiting for connection 🟥
J-Link: Connected 🟩
CPU: MKS22FN256xxx12 🟩 3.30 V 🟩
Initial SWD speed: 1000 kHz
Current SWD speed: 1000 kHz
Little endian
☑ Localhost only
☐ Stay on top
☑ Show log window
☐ Generate logfile
☐ Verify download
☐ Init regs on start

Log output:

```
Target interface speed:        1000kHz
Target endian:                 little

Connecting to J-Link...
J-Link is connected.
Firmware: J-Link OpenSDA 2 compiled Oct 13 2015 12:10:27
Hardware: V1.00
S/N: 621000000
Checking target voltage...
Target voltage: 3.30 V
Listening on TCP/IP port 2331
Connecting to target...Connected to target
Waiting for GDB connection...
```

0 Bytes downloaded | 1 JTAG device

Figure 40. SEGGER J-Link GDB Server screen after successful connection

6. If not already running, open a GCC ARM Embedded tool chain command window. To launch the window, from the Windows operating system Start menu, go to “Programs -> GNU Tools ARM Embedded <version>” and select “GCC Command Prompt”.

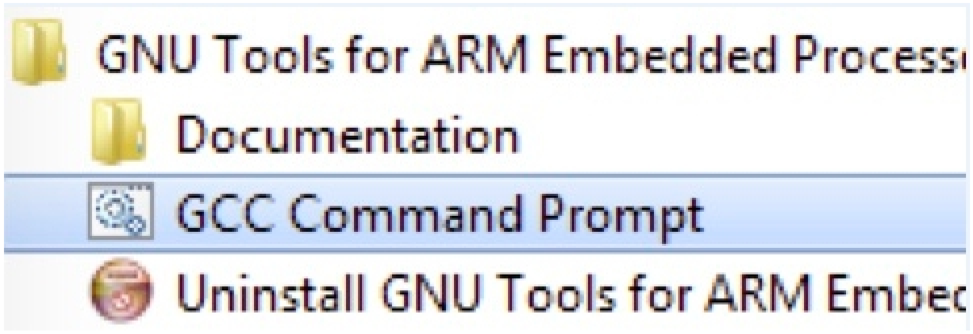

GNU Tools for ARM Embedded Process
Documentation
GCC Command Prompt
Uninstall GNU Tools for ARM Embec

Figure 41. Launch command prompt

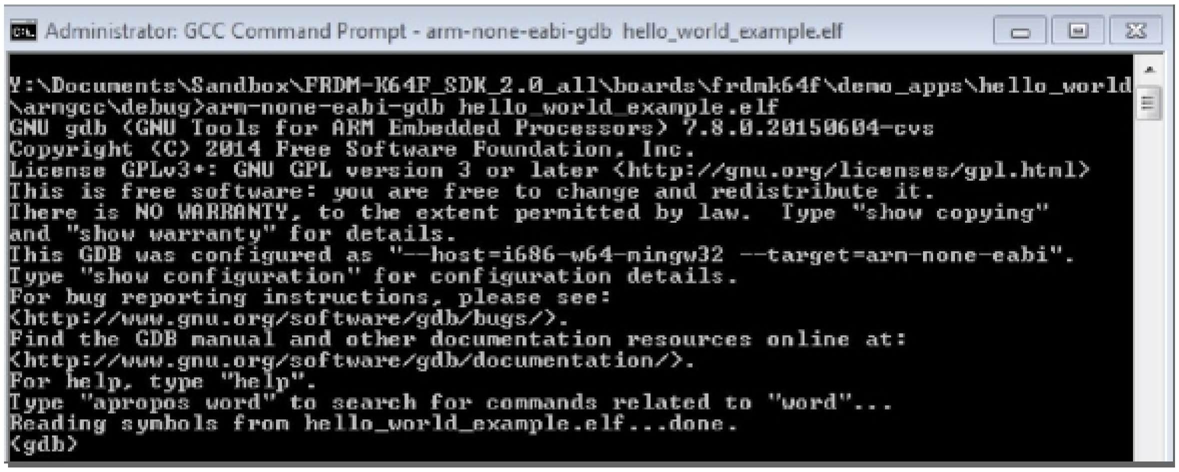
7. Change to the directory that contains the example application output. The output can be found in using one of these paths, depending on the build target selected:

   <install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc/debug

   <install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc/release

   For this example, the path is:

   <install_dir>/boards/mapsks22/demo_apps/hello_world/armgcc/debug
8. Run the command “arm-none-eabi-gdb.exe <application_name>.elf”. For this example, it is “arm-none-eabi-gdb.exe hello_world.elf”.



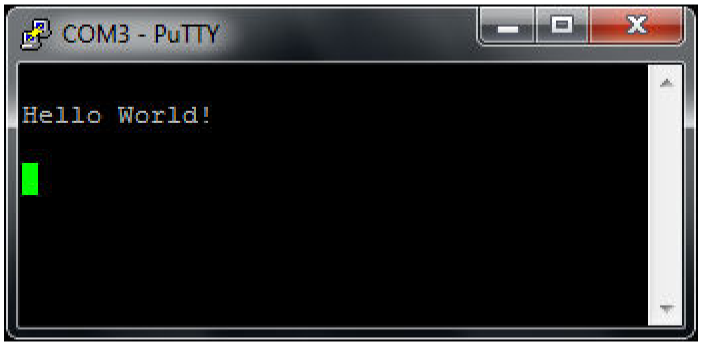
```
Administrator: GCC Command Prompt - arm-none-eabi-gdb  hello_world_example.elf

Y:\Documents\Sandbox\FRDM-K64F_SDK_2.0_all\boards\frdmk64f\demo_apps\hello_world
\armgcc\debug>arm-none-eabi-gdb hello_world_example.elf
GNU gdb (GNU Tools for ARM Embedded Processors) 7.8.0.20150604-cvs
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from hello_world_example.elf...done.
(gdb)
```

Figure 42. Run arm-none-eabi-gdb

9. Run these commands:
   a. "target remote localhost:2331"
   b. "monitor reset"
   c. "monitor halt"
   d. "load"
   e. "monitor reset"
10. The application is now downloaded and halted at the reset vector. Execute the “monitor go” command to start the demo application.

    The hello_world application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.



```
COM3 - PuTTY

Hello World!
```

Figure 43. Text display of the hello_world demo

8 Appendix A - How to determine COM port

This section describes the steps necessary to determine the debug COM port number of your Freescale hardware development platform. All Freescale boards ship with a factory programmed, on-board debug interface, whether it’s based on OpenSDA or the legacy P&E Micro OSJTAG interface. To determine what your specific board ships with, see Appendix B.

1. To determine the COM port, open the Windows operating system Device Manager. This can be achieved by going to the Windows operating system Start menu and typing “Device Manager” in the search bar, as shown below:

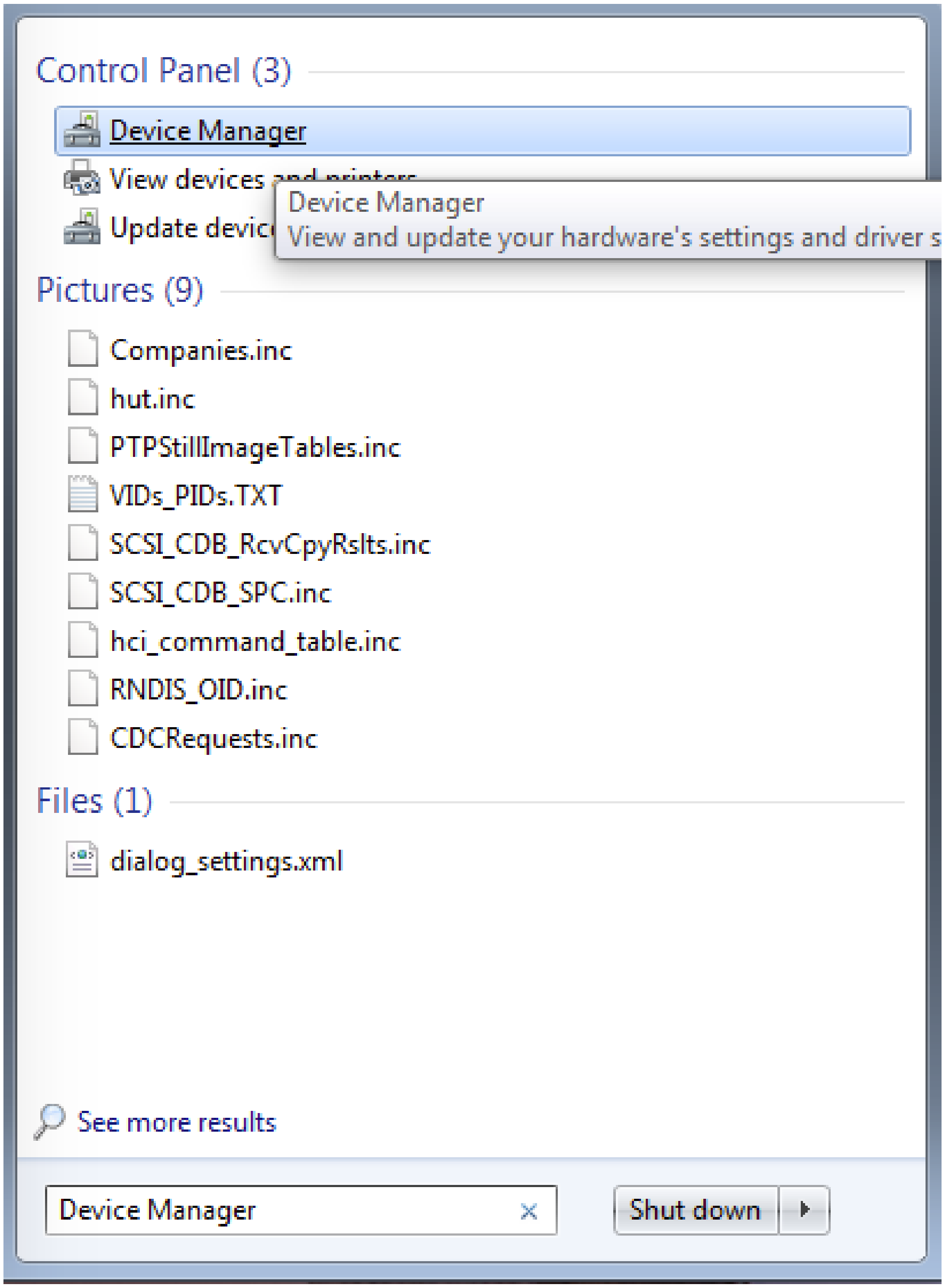


Figure 44. Device manager

2. In the Device Manager, expand the “Ports (COM & LPT)” section to view the available ports. Depending on the Freescale board you’re using (see Appendix B), the COM port can be named differently:

9 Appendix B - Default debug interfaces

The Kinetis SDK supports various Kinetis hardware platforms that come loaded with a variety of factory programmed debug interface configurations. The following table lists the hardware platforms supported by the KSDK, their default debug interface, and any version information that helps differentiate a specific interface configuration.

All recent and future Freescale hardware platforms support the configurable OpenSDA standard.

Table 1. Hardware platforms supported by KSDK

Hardware Platform	Default Interface	OpenSDA details
MAPS-KS22	J-Link	OpenSDA v2.0

10 Appendix C - Updating OpenSDA firmware

Any Freescale hardware platform that comes with an OpenSDA-compatible debug interface has the ability to update the OpenSDA firmware. This typically means switching from the default application (P&E Micro) to a SEGGER J-Link. This section contains the steps to switch the OpenSDA firmware to a J-Link interface. However, the steps can be applied to also restoring the original image.

For reference, OpenSDA firmware files can be found at the links below:

- J-Link: Download appropriate image from www.segger.com/opensda.html**.** Chose the appropriate J-Link binary based on the table in Appendix B. Any OpenSDA v1.0 interface should use the standard OpenSDA download (i.e., the one with no version). For OpenSDA 2.0 or 2.1, select the corresponding binary.
- CMSIS-DAP/mbed/DAPLink: This interface is provided to support the ARM mbed initiative. Navigate to developer.mbed.org/platforms and select your hardware platform. On the specific platform/board page, there is a link to the firmware image and instructions on how to load it, though the instructions are the same as below.
- P&E Micro: Downloading P&E Micro OpenSDA firmware images requires registration with P&E Micro (www.pemicro.com).

These steps show how to update the OpenSDA firmware on your board for Windows operating system and Linux OS users:.

1. Unplug the board's USB cable.
2. Press the board's "Reset" button. While still holding the button, plug the board back in to the USB cable.
3. When the board re-enumerates, it shows up as a disk drive called "BOOTLOADER".

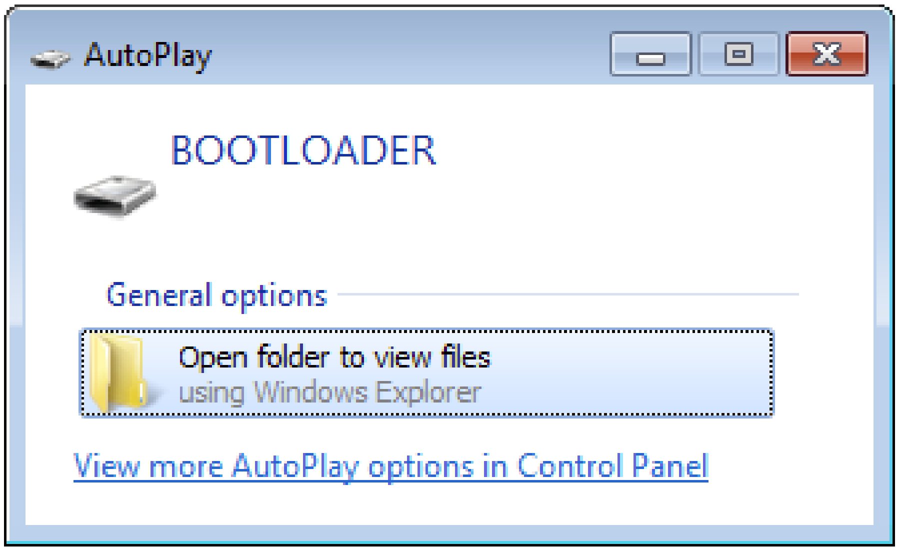

AutoPlay
BOOTLOADER
General options
Open folder to view files
using Windows Explorer
View more AutoPlay options in Control Panel

Figure 45. BOOTLOADER drive

4. Drag the new firmware image onto the BOOTLOADER drive in Windows operating system Explorer, similar to how you would drag and drop a file onto a normal USB flash drive.

NOTE

If for any reason the firmware update fails, the board can always re-enter bootloader mode by holding down the "Reset" button and power cycling.

These steps show how to update the OpenSDA firmware on your board for Mac OS users.

NOTE

The USB-KW019032 board has a specific OpenSDA interface, which is not compatible with the J-Link and P&E Micro OpenSDA firmware image.

1. Unplug the board's USB cable.
2. Press the board's "Reset" button. While still holding the button, plug the board back in to the USB cable.
3. For boards with OpenSDA v2.0 or v2.1, it shows up as a disk drive called "BOOTLOADER" in Finder. Boards with OpenSDA v1.0 may or may not show up depending on the bootloader version. If you see the drive in Finder, you may proceed to the next step. If you do not see the drive in Finder, use a PC with Windows® OS 7 or an earlier version to either update the OpenSDA firmware or update the OpenSDA bootloader to version 1.11 or later. The bootloader update instructions and image can be obtained from P&E Microcomputer website.
4. For OpenSDA v2.1 and OpenSDA v1.0 (with bootloader 1.11 or later) users, drag the new firmware image onto the BOOTLOADER drive in Finder, similar to how you would drag and drop the file onto a normal USB Flash drive.
5. For OpenSDA v2.0 users, type these commands in a Terminal window:

```
> sudo mount -u -w -o sync /Volumes/BOOTLOADER
> cp -X  <path to update file> /Volumes/BOOTLOADER
```

NOTE

If for any reason the firmware update fails, the board can always re-enter bootloader mode by holding down the "Reset" button and power cycling.

11 Revision History

This table summarizes revisions to this document.

Table 2. Revision History

Revision number	Date	Substantive changes
0	12/2015	Initial release

Document Number KSDK20MKS22FN256GSUG
Revision 0, 12/2015