Freescale Semiconductor, Inc.

Document Number: KSDKK64UG
User's Guide
Rev. 1.0.0, 07/2014

Kinetis SDK K64 User's Guide

1 Introduction

This document describes the hardware and software environment setup for the Kinetis SDK (KSDK). It also explains how to build and run demo applications provided in the KSDK release package.

2 Overview

2.1 Kinetis SDK

Kinetis SDK is a software development that provides software support for the core and peripherals for Freescale devices with the ARM Cortex®-M core. KSDK includes a hardware abstraction layer (HAL) for each peripheral and peripheral drivers built on the HAL. Example applications are provided to demonstrate peripheral driver and HAL usage to highlight the main features of the targeted SoCs. Also, KSDK contains the latest available RTOS kernels, USB stacks, and other software components used on the supported evaluation boards.

Contents

2.2 Hardware requirement

TWR-K64F120M Tower System module or Freescale Freedom FRDM-K64F platform is required.

2.3 Toolchain requirement

To use the IAR for K64 development, apply a flash loader patch from IAR. Download this patch here, and unzip it under the IAR install folder as: C:\Program Files\IAR Systems\Embedded Workbench

3 Hardware configurations

This section describes how to set up the TWR-K64F120M Tower System module (RevB) and Freescale Freedom FRDM-K64F (RevD) platform for the KSDK application.

3.1 TWR-K64F120M Tower System module introduction

3.1.1 TWR-K64F120M Tower System module features

- K64FN1M0VMD12 MCU (120 MHz, 1024 KB Flash, 260 KB RAM, low power, 144 MAPBGA package)
- Dual-role USB interface with Micro-AB USB connector
- Onboard debug circuit: K20DX128VFM5 open (OpenSDA) with virtual serial port
- Three-axis accelerometer (MMA8451Q)
- Four (4) user-controllable LEDs
- Two (2) user push-button switches for GPIO interrupts (SW1/SW3)
- One potentiometer
- Independent, battery-operated power supply for the real-time clock (RTC)
- Standard SD Card slot

3.1.2 TWR-K64F120M Tower System module first look

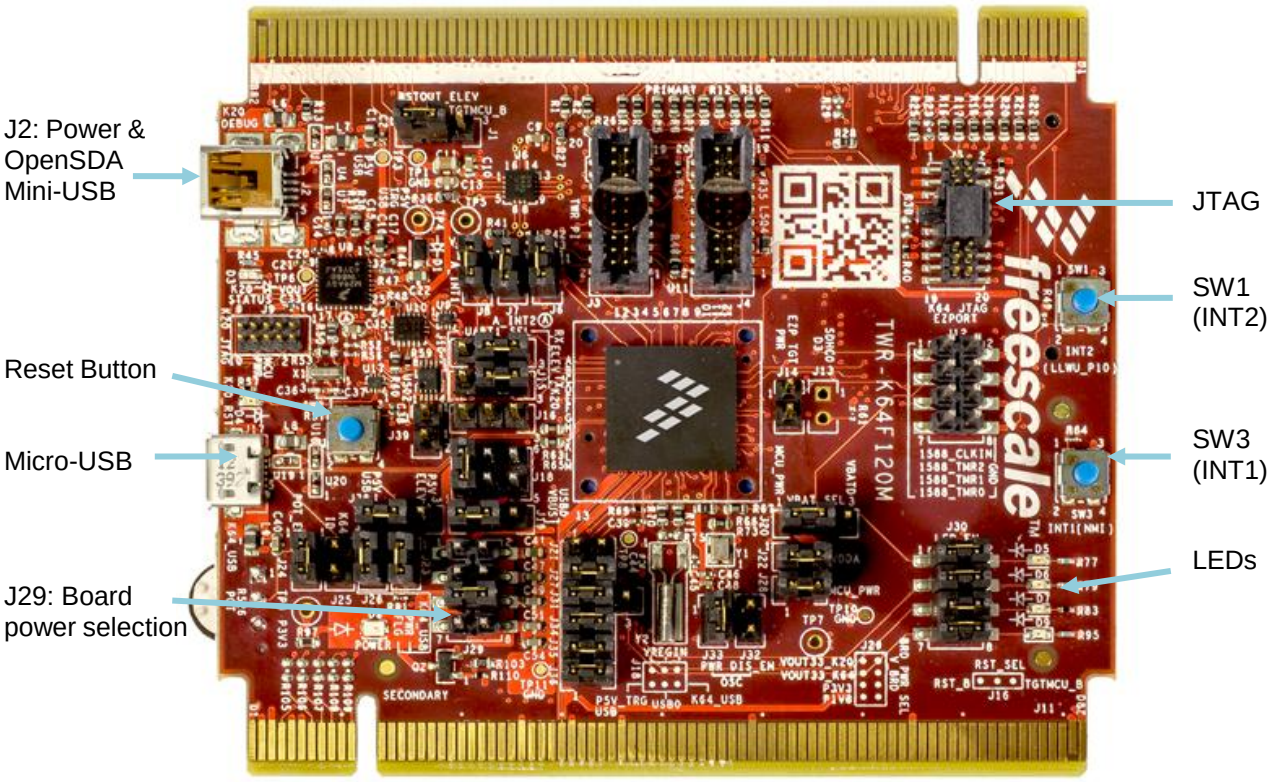

Figure-1 Front side of TWR-K64F120M Tower System module

Figure-2 Back side of TWR-K64F120M Tower System module

3.1.3 TWR-K64F120M Tower System module jumper settings

Ensure that the jumpers are set as shown in this table. For more details, see the *TWR-K64F120M Tower Module User's Guide* (document TWRK64F120M).

Table-1 TWR-K64F120M Tower System module jumper settings

Option	Jumper	Setting	Description
50 MHz Clock OSC power	J33	1-2	Enable V_BRD power supply to 50 MHz OSC
JTAG Board Power Selection	J14	OFF	Disconnect OSJTAG 5V output (P5V_TRG_USB) from JTAG port
3.3 V Voltage Regulator Input Selector	J18	1-2	Output of USB power switch controlled by the VTRG_EN signal from the K20 MCU. Provides input to 3.3 V regulator.
Board Power Selector	J29	1-2	Connect K20 USB regulator output (VOUT_3V3) to onboard supply (V_BRD)
MCU Power connection	J28	ON	Connect onboard 3.3 V or 1.8 V supply (V_BRD) to MCU VDD
MCU Power VDDA for current measurement	J22	ON	Connect MCU_PWR (3.3 V or 1.8 V) to VDDA and VREFH
VBAT Power Source	J20	1-2	Connect VBAT to on-board 3.3 V or 1.8 V supply
		1-2	Connect PTE6 to Yellow/Green LED (D5)

LED connections	J30	3-4	Connect PTE7 to Yellow LED (D6)
		5-6	Connect PTE8 to Orange LED (D7)
		7-8	Connect PTE9 to Blue LED (D9)
OpenSDA (K20) UART Selection	J10	2-3	Connect OpenSDA (K20) UART TX to UART1_RX
	J15	2-3	Connect OpenSDA (K20) UART RX to UART1_TX
SWD_CLK_TGTMCU Output Selection	J39	ON	Enable the SWD_CLK_TGTMCU connection between the OpenSDA and target MCU

3.2 Freescale Freedom FRDM-K64F platform

3.2.1 Freescale Freedom FRDM-K64F platform features

- K64FN1M0VLL12 MCU (120 MHz, 1024 KB Flash, 260 KB RAM, low power, 100 LQFP package)
- Dual-role USB interface with Micro-AB USB connector
- On-board debug circuit: K20DX128VFM5 open (OpenSDAv2) with virtual serial port
- Six-axis sensor with integrated linear accelerometer and magnetometer (FXOS8700CQ)
- Tri-color user-controllable LEDs
- Two (2) user push-button switches for GPIO interrupts (SW2/SW3)
- RF and Bluetooth® socket
- Onboard Ethernet Physical Interface (KSZ8081)
- Micro SD Socket

3.2.2 Freescale Freedom FRDM-K64F platform first look

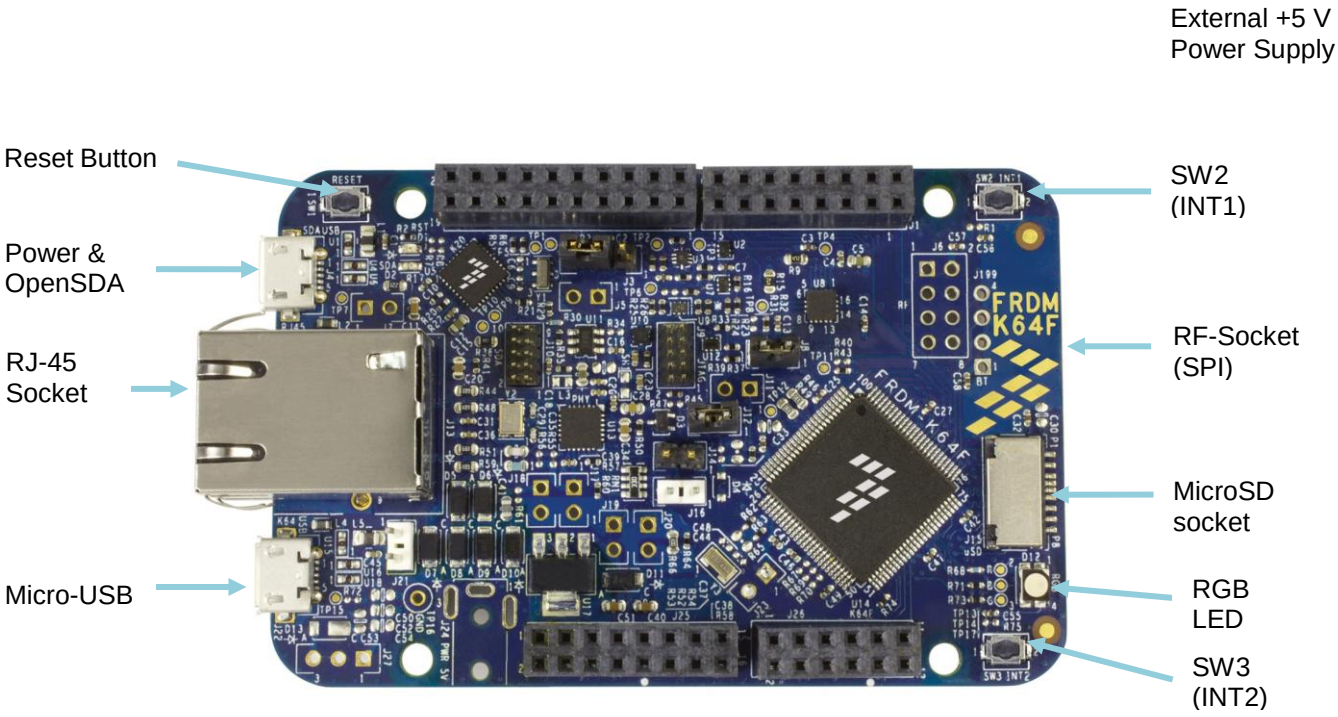

Figure-3 Front side of Freescale Freedom FRDM-K64F B457 platform

3.2.3 Freescale Freedom FRDM-K64F platform jumper settings

Ensure that the jumpers are set as shown in this table. For more details, see the *FRDM-K64 Freedom Module User's Guide (*document FRDMK64FUG).

Table-2 Freescale Freedom FRDM-K64 platform jumper settings

Option	Jumper	Setting	Description
Optional USB Host Functionality	J21	OFF	Disable USB Host functionality
VBAT Power Source	J23	OFF	Disconnect K64 3.3 V supply to VBAT
OpenSDAv2 debug source select	J11	1-2	Connect MCU with OpenSDAv2 debug interface
RST Push-Button Bypass	J25	1-2	Default setting for RST button
OpenSDA connections	J12	1-2	Connect OpenSDA DIO signal to target MCU
OpenSDA connections	J8	1-2	Connect OpenSDA CLK signal to target MCU
KSZ8081 interrupt connections	J14	OFF	Disconnect KSZ8081 interrupt signal to target MCU

4 Build and run a KSDK demo application

This section describes the configuration process for IAR Embedded Workbench, GCC ARM Embedded Tool, Keil uVision IDE, and Kinetis Design Studio IDE to build, run, and debug demo applications provided in the Freescale KSDK.

The hello_world demo application, targeted for the TWR-K64F120M Tower System module hardware platform or Freescale Freedom FRDM-K64F platform, is used as an example.

Note that the SDK libraries must be built before building a demo application. To build the SDK libraries, see Appendix A of this document.

4.1 IAR Embedded Workbench

4.1.1 Build a demo application

Because the platform driver library is already included in the lib folder of the demo application project, you can open the demo application project and build the demo applications directly whenever the ksdk_platform_lib.a is ready.

Demo applications workspace files are located in:

`<install_dir>/demos/<demo_name>/<compiler>/<board_name>/<demo_name>.eww`

If the hello_world demo application is used as an example, the IAR workspace file is located here: `<install_dir>/demos/hello_world/iar/twrk64f120m/hello_world.eww`

Alternatively, it is located here:

`<install_dir>/demos/hello_world/iar/frdmk64f120m/hello_world.eww`

To build a demo application, click on the “Make” button:

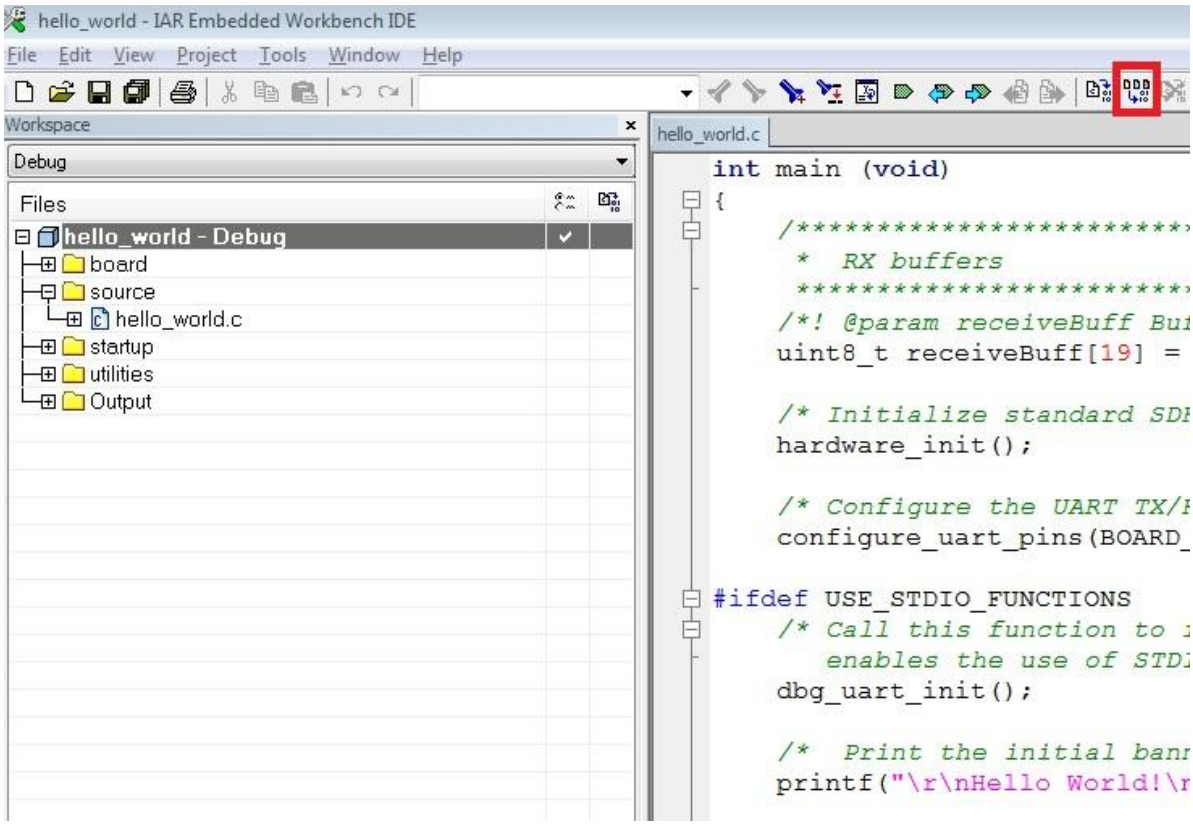

Figure-4 Build hello_world demo application

When the build is complete, IAR displays this information in the build window:

Figure-5 Build hello_world demo successfully

4.1.2 Run a demo application

Downloading and debugging KSDK demo applications in IAR Embedded Workbench is a standard process for all applications. Follow these steps to download and run the application:

1. If your target platform is TWR-K64F Tower System module, use the PE micro debugger. If your target platform is the Freescale Freedom FRDM-K64F platform, use the factory installed CMSIS-DAP debugger firmware. If you want to update firmware, check the Appendix C for CMSIS-DAP firmware update steps.
2. Download and install the latest CMSIS-DAP mbed serial drivers from: *mbed.org/handbook/Windows-serial-configuration#1-download-the-mbed-windows-serial-port*. Note that only the Windows® operating system needs drivers. The mbed drivers work by default on Linux and Mac systems.
3. After the successful installation of the serial driver, connect the CMSIS-DAP USB connector, J26 on the Freescale Freedom FRDM-K64F platform, to the USB port on a PC.

4. Open the terminal application on a PC, such as PuTTy, and connect to the mbed COM port number. Configure the terminal settings as shown here:
    a. 115200 baud rate
    b. No parity
    c. 8 data bits
    d. 1 stop bits

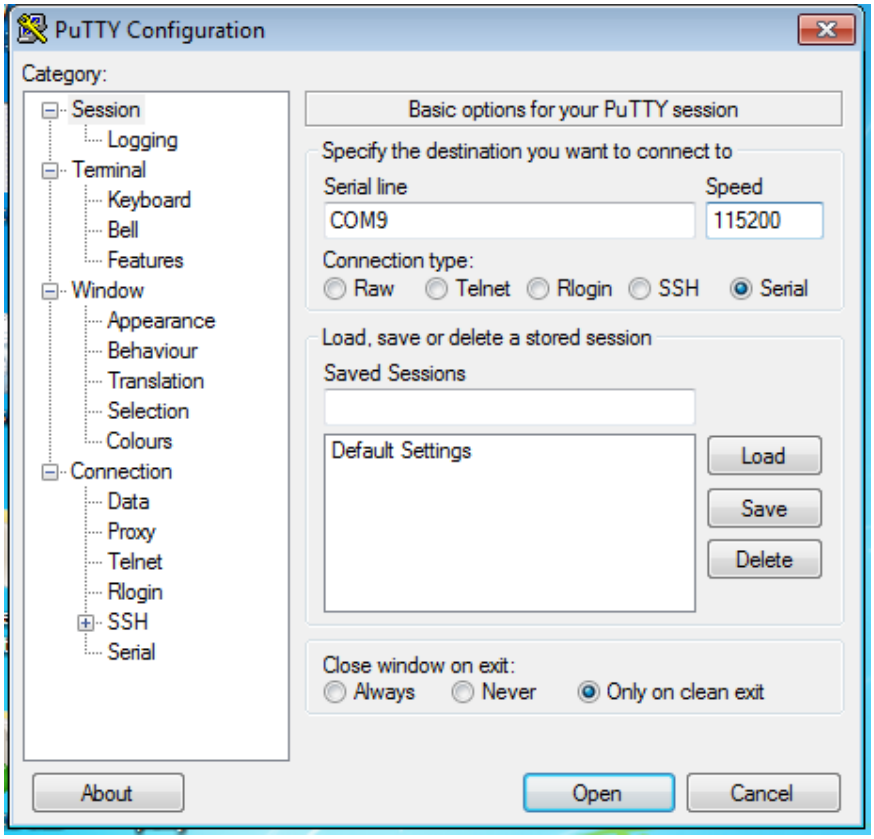

Figure-6 Terminal (PuTTY) configurations

5. Configure the Debugger in IAR as below.
   - Select the appropriate debugger in the debugger setup:

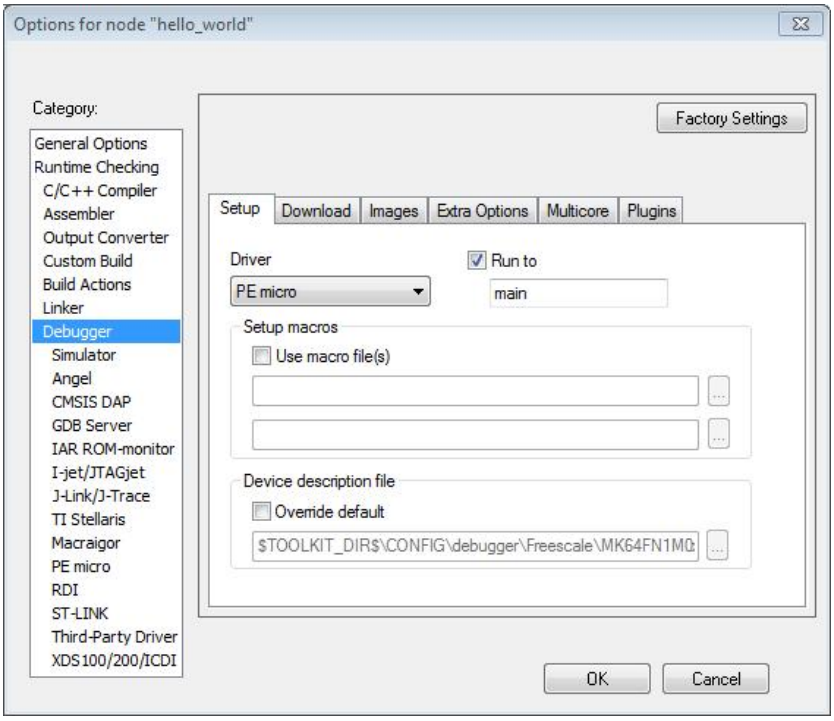

Figure-7 Debugger configuration for TWR-K64F120M Tower system module

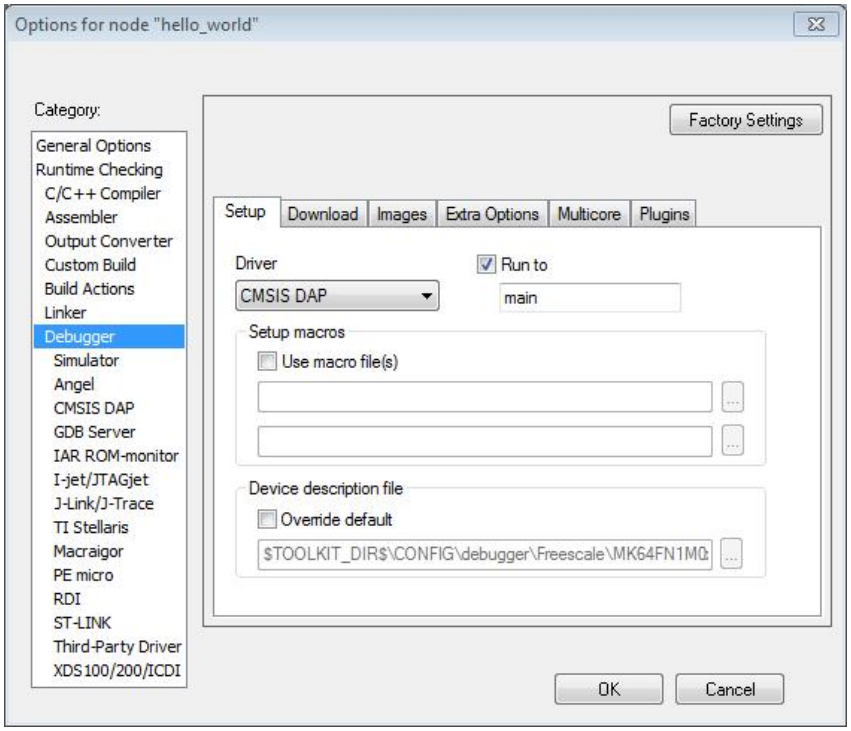

Figure-8 Debugger Configurations for Freescale Freedom FRDM-K64F platform

- SWD should be configured as the debugger interface in the debugger specific category.

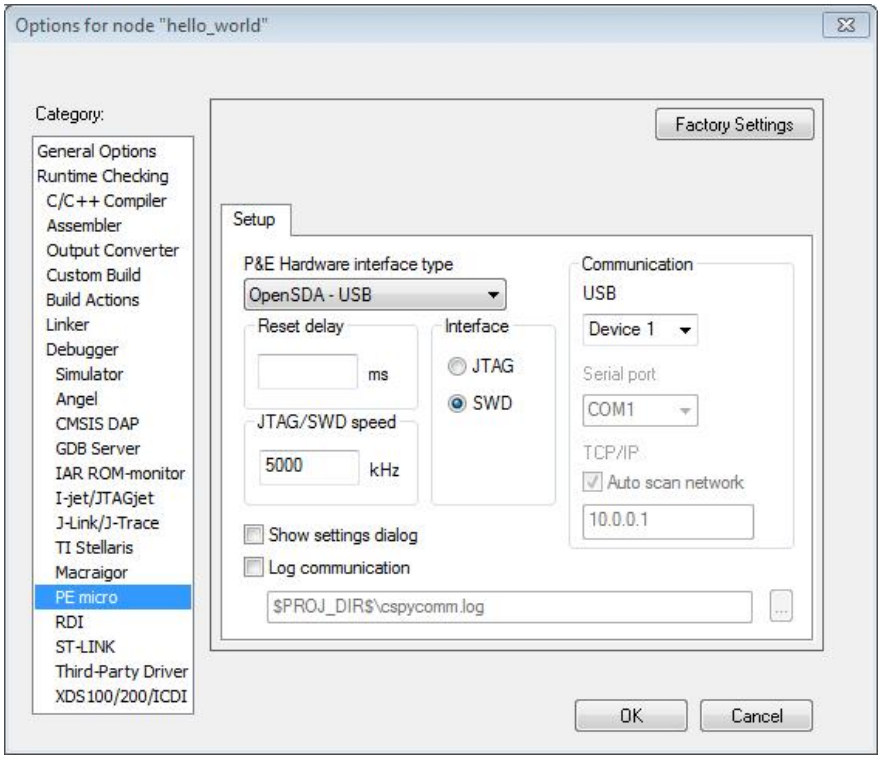

Figure-9 Debugger configuration for TWR-K64F120M Tower system module (PE Micro)

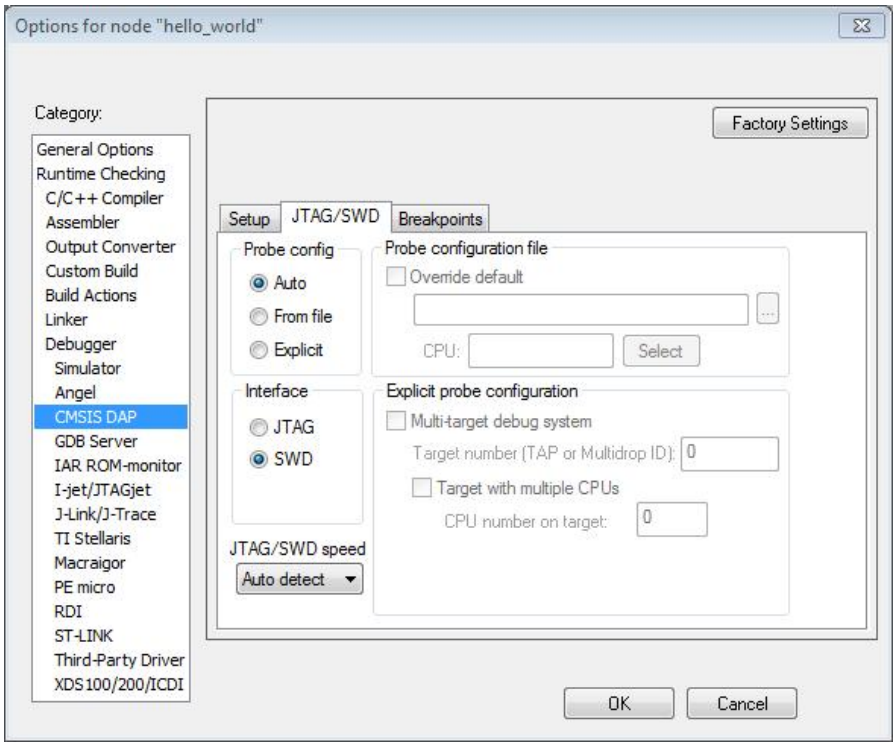

Figure-10 Debugger Configurations for Freescale Freedom FRDM-K64F platform (CMSIS DAP)

6. After the terminal has successfully connected, click on the “Download and Debug” button to download the application to the target device.

Figure-11 Download and debug button

7. Next, the debugger stops executing at the start of the main() function:

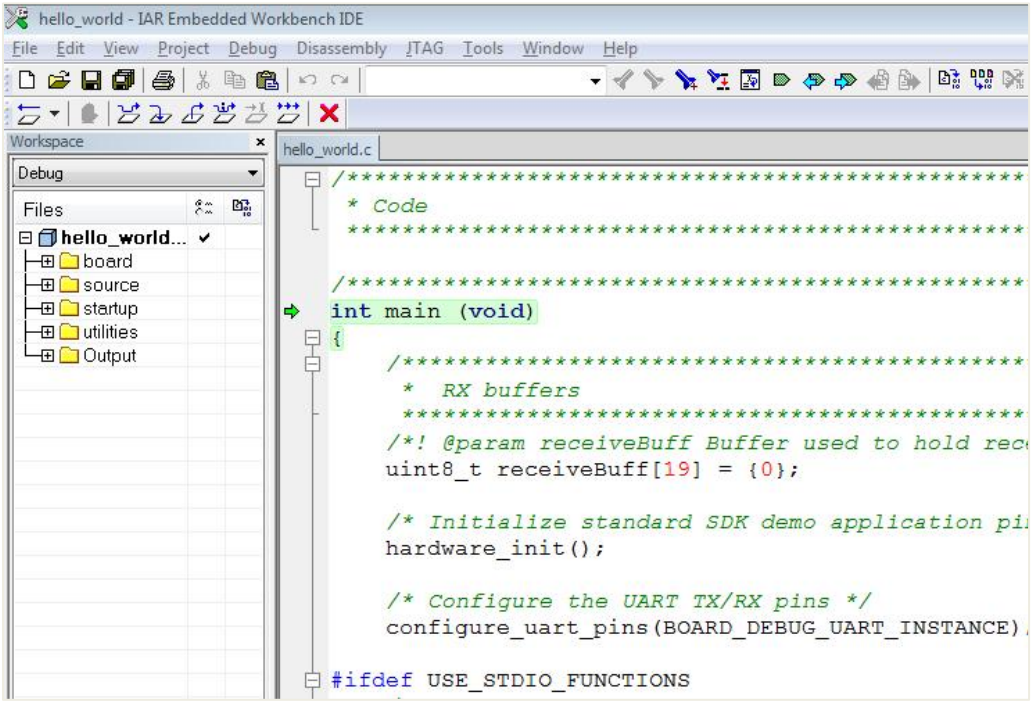

Figure-12 Stop at main() when run debugging

Resume application execution by clicking on the “Go” button:

Figure-13 Go

9. The hello_world application should now be running and the following banner should be displayed on the terminal. If this is not the case, check the terminal settings and terminal connections.

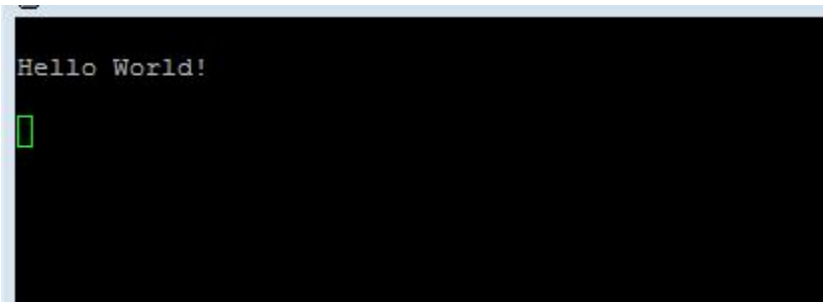

Figure-14 Hello_world demo running

4.2 GCC ARM Embedded Tool

4.2.1 Environment setup

4.2.1.1 Install GCC ARM v4.8.3 2014q1 embedded toolchain

1. Download the Windows installer.
2. Install the toolchain in the /Program Files/ location, which is usually on your "C:\" drive.

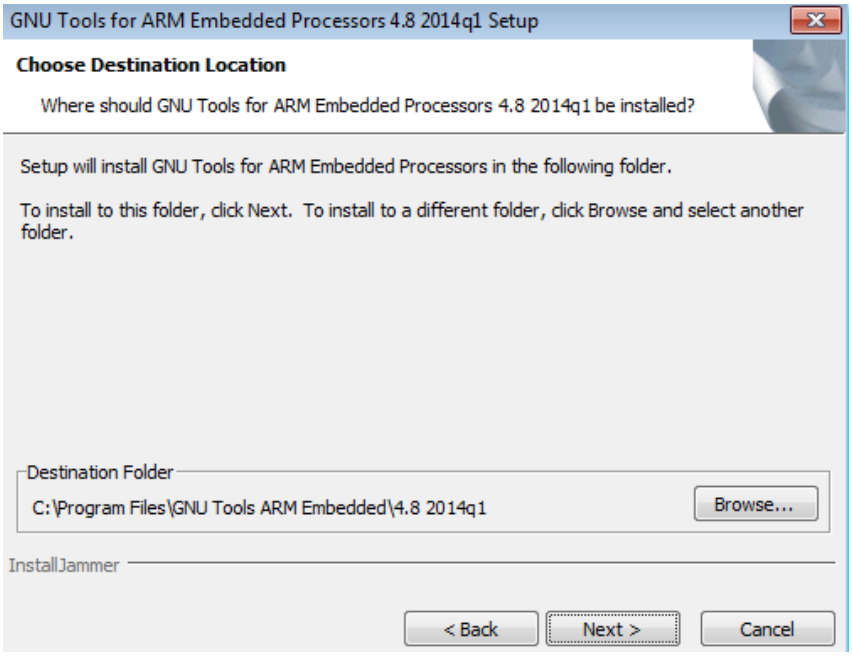

Figure-15 Install GCC ARM embedded toolchain

4.2.1.2 Install MinGW and MSYS

1. Download the MinGW installer, which is located here.
2. Run mingw-get-setup.exe and select the installation path, such as: C:/MINGW.
3. Select the mingw32-base and the msys-base under the Basic Setup as shown in Figure-16 MinGW Installer.

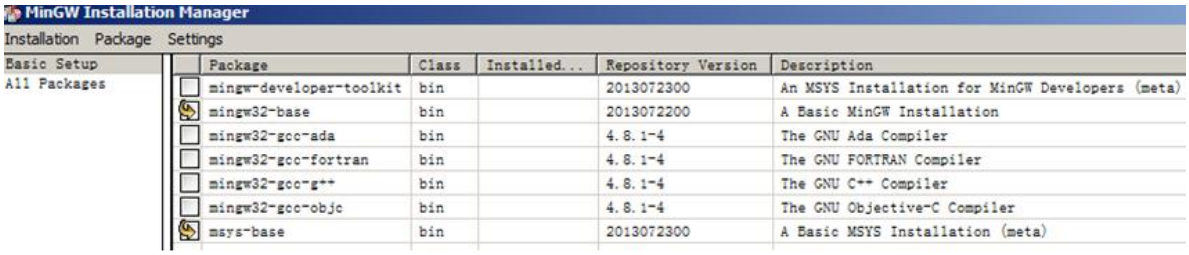

Figure-16 MinGW Installer

4. Click on “Apply Changes” from the “Installation” menu to install packages, as shown in Figure 17.

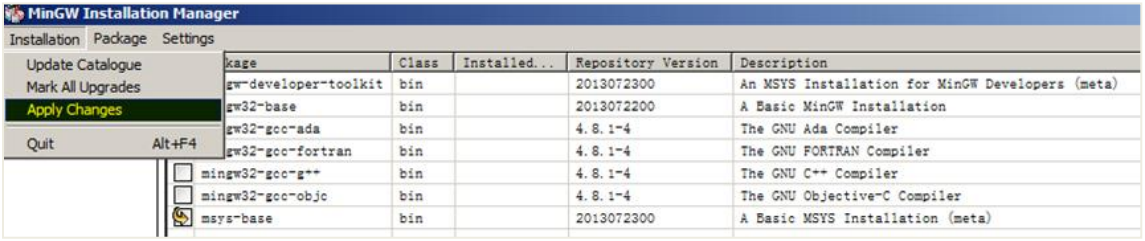

Figure-17 MinGW Installer apply change

4.2.1.3 Configure system environment

- Update the system environment variable “Path” to include the MINGW installation folder, such as the <drive>\MINGW\msys\1.0\bin;<drive>:\MINGW\bin.

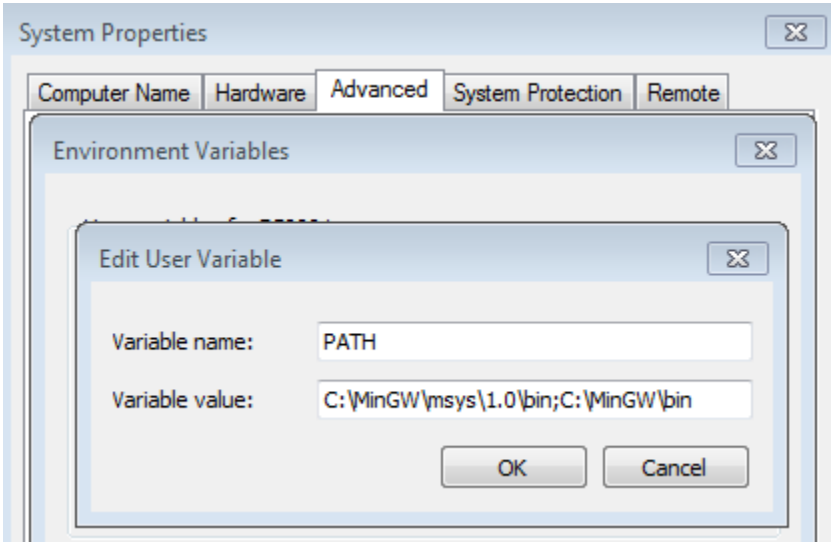

Figure-18 Environment variable update

- Run the GCC Command prompt in Start>All Programs> GNU Tools ARM Embedded 4.8.3 2014q1.

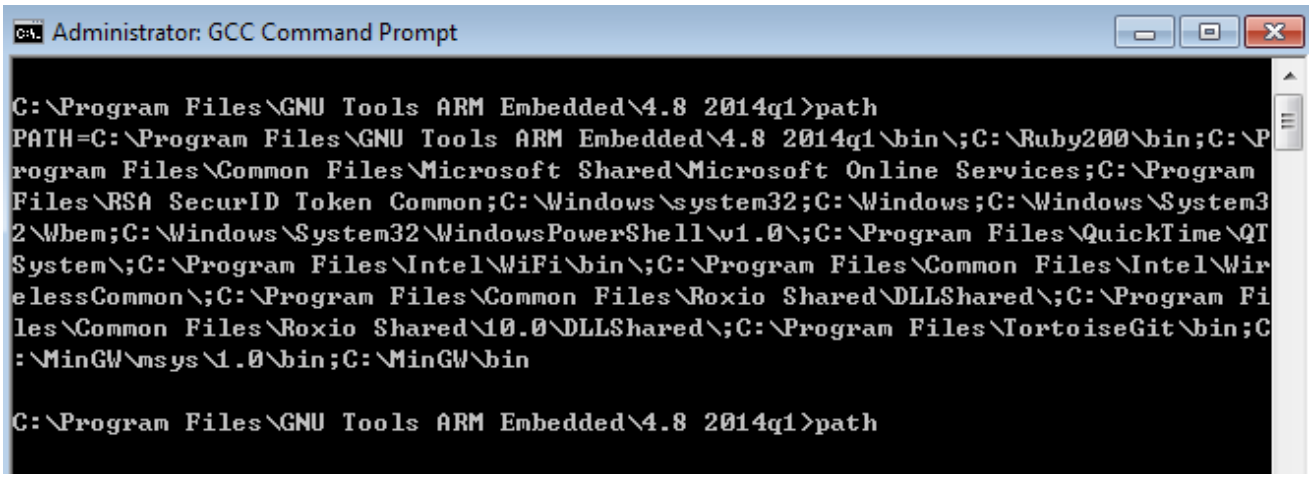

Figure-19 PATH environment

- When using a Windows command line, add the environment variable, ARMGCC_DIR, which is also the short name of the ARM GCC installation path.

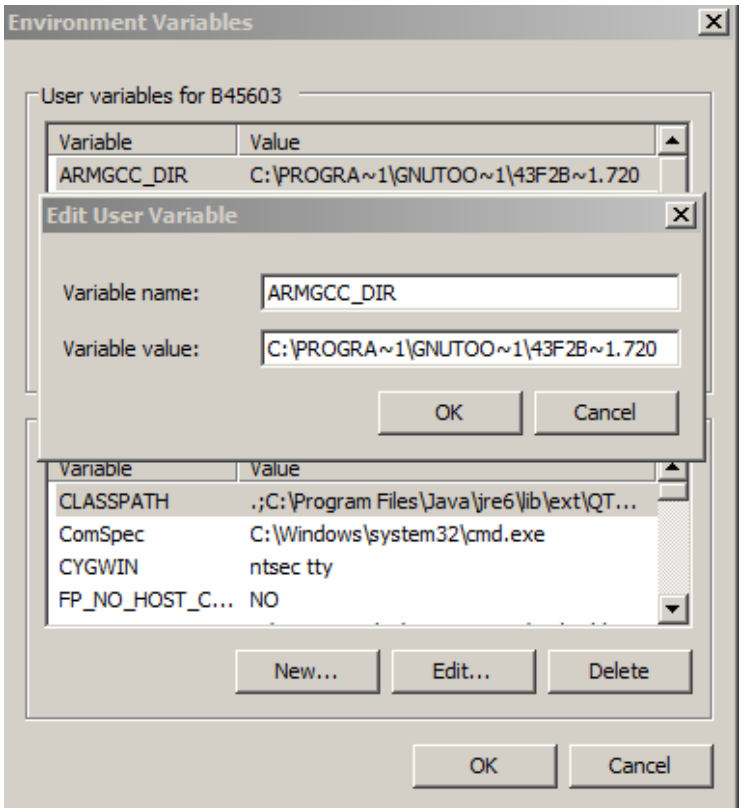

Figure-20 Add environment variable

- When using GIT Bash or Cygwin, set the environment variable to "export ARMGCC_DIR=C:/PROGRA~/GNUTOO~1/4298B~1.820".

Note

Use a forward slash '/' as a separator.

When using the KDS GCC toolchain, add the system environment variable, KDSGCC_DIR, which is also the path of KDS GCC toolchain. Use a short name and run the “mingw32-make toolchain=kdsgcc”.

```
>/mk/common.mk
```

4.2.2 Build the platform driver library

Before building and debugging any demo application in KSDK, the driver library project should be built to generate the library archives: ksdk_platform_lib.a. Because this library contains all binary codes for HAL and the peripheral drivers specific to the chip, each SoC has its own platform.a library archives.

To build the platform library, change the current directory in GCC command prompt to:

```
<install_dir>/lib/ksdk_platform_lib/gcc/<device_name>/
For example: <install_dir>/lib/ksdk_platform_lib/gcc/K64F12/
```

Run the command mingw32-make build=debug or mingw32-make build=release.

When the build is complete, the ksdk_platform_lib.a is generated in the directory according to the build target:

```
Debug - <install_dir>/lib/ksdk_platform_lib/gcc/<device_name>/Debug
Release - <install_dir>/lib/ksdk_platform_lib/gcc/<device_name>/Release
```

Demo applications use the ksdk_platform_lib.a to call the HAL and peripheral driver functions.

4.2.3 Build a demo application

Change the directory in GCC command prompt to this:

```
<install_dir>/demos/hello_world/gcc/twrk64f120m
```

Alternatively, change the directory in the GCC command prompt to this:

```
<install_dir>/demos/hello_world/gcc/frdmk64f120m
```

Run the mingw32-make build=debug target=flash command.

When the build is complete, the hello_world.elf and the hello_world.bin are generated in this path:

```
<install_dir>/demos/hello_world/gcc/twrk64f120m/Flash_Debug
```

Alternatively, they are generated in this path:

```
<install_dir>/demos/hello_world/gcc/frdmk64f120m/Flash_Debug
```

4.2.4 Run a demo application

This section describes how to download and debug the applications by using J-Link.

Note

To use an external debugger, such as J-Link, you may need to disconnect the OpenSDA SWD from the K64. To do this on the Freescale Freedom

FRDM-K64F platform, cut the shorting trace which connects the pins of jumper holes on connectors J8 and J12.

You can debug the application by using Eclipse with CDT or command line. This example shows how to use the GDB command line.

- Download and install J-Link software and documentation package for Windows operating system here. You may need a J-Link serial number to download that software.
- K64 is now supported in the Segger GDB server. Setup the GDB server by running the GDB server JLinkGDBServer.exe.

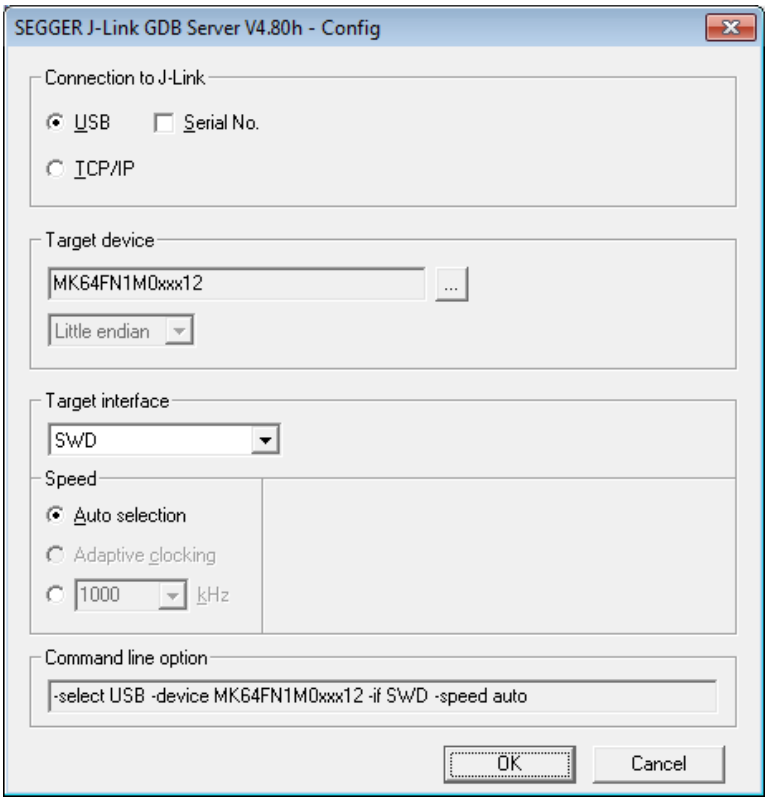

Figure-21 J-Link GDB server GUI

- Select the connection options for your board. For K64, select the MK64FN1M0xxx12 device as the target device. After configuration, click on the "OK" button to connect to the board.

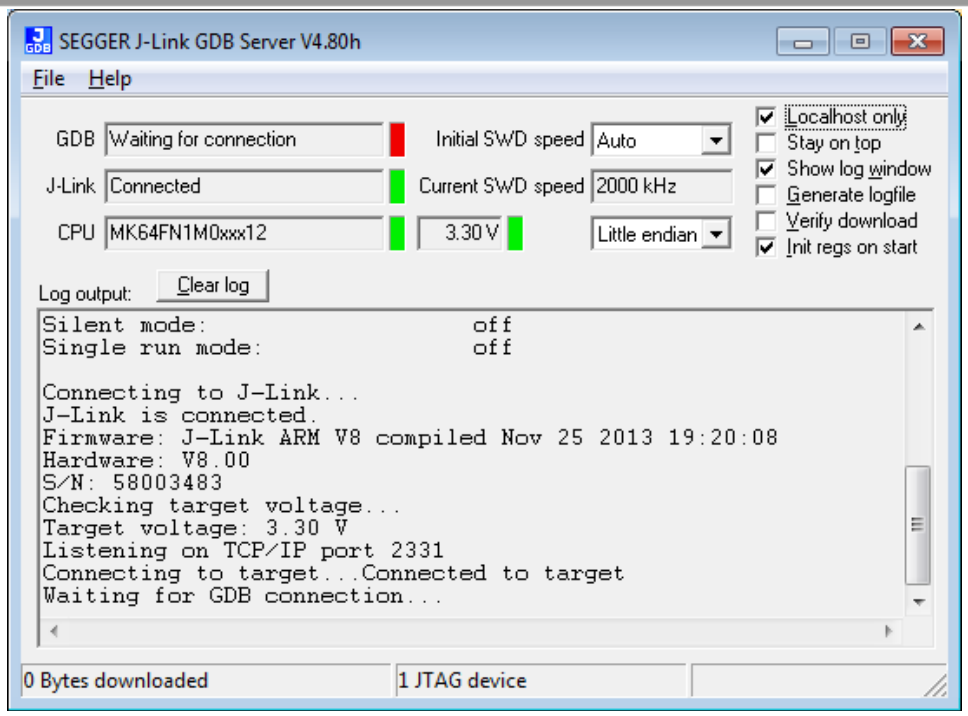

Figure-22 JLinkGDBServer connection

- Download and debug from command line

  Change the directory like this:

```
<install_dir>/demos/hello_world/gcc/twrk64f120m/Flash_Debug
```

  Alternatively, change the directory like this:

```
<install_dir>/demos/hello_world/gcc/frdmk64f120m/Flash_Debug
```

- Run the arm-none-eabi-gdb to start the GDB. After the GDB is launched, run these commands to load and start the application:

```
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
0x00000000 in __isr_vector ()
(gdb) monitor reset
Resetting target
(gdb) monitor halt
(gdb) monitor load
(gdb) load
Loading section .text, size 0x495c lma 0x0
Loading section .ARM_exidx, size 0x8 lma 0x495c
Loading section .data, size 0xac lma 0x4964
Start address 0x8d1, load size 18960
Transfer rate: 1089 KB/sec, 4740 bytes/write.
(gdb) monitor reg pc = (0x00000004)
Writing register (PC = 0x000008D1)
(gdb) monitor reg sp = (0x00000000)
Writing register (SP = 0x1FFF2588)
(gdb) monitor go
(gdb)
```

Figure-23 Run arm-none-eabi-gdb

- Debugging

  Use the GDB general debug command to do debug. To quit the debugging, use the monitor halt to halt the target CPU, then quit the debug.

4.3 KEIL uVision IDE

4.3.1 Build a demo application

Since the platform driver library is already included in the lib folder and the ksdk_platform_lib.lib is generated, users can directly open the demo application project and build demo applications.

Demo application workspace files are located in:

`<install_dir>/demos/<demo_name>/<compiler>/<board_name>/<demo_name>.uvproj`

Take the hello_world demo application of FRDM-K64F for example; the KEIL workspace file is in:
`<install_dir>/demos/hello_world/uv4/frdmk64f120m`

To build a demo application, click the "build button":

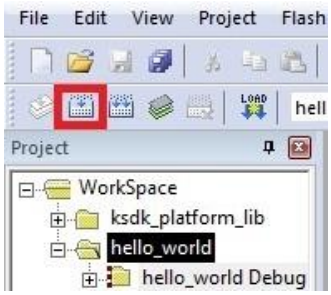

Figure-24 Build target files in a demo application

To rebuild all target files in a demo application, click the "rebuild button":

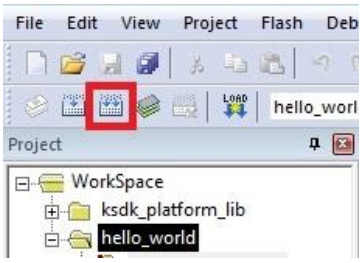

Figure-25 Rebuild all target files in a demo application

When the build is complete, KEIL displays this information in the build output window:

```
Build Project 'hello_world' - Target 'hello_world Debug'
compiling fsl_misc_utilities.c...
compiling fsl_debug_console.c...
assembling startup_MK64F12.s...
compiling system_MK64F12.c...
compiling startup.c...
compiling hello_world.c...
compiling gpio_pins.c...
compiling pin_mux.c...
compiling hardware_init.c...
linking...
Program Size: Code=18008 RO-data=1192 RW-data=40 ZI-data=49344
"debug\hello_world.out" - 0 Error(s), 0 Warning(s).
```

Figure-26 Build hello_world demo application successfully

4.3.2 Run a demo application

1. Driver installation, CMSIS-DAP setup and serial terminal configuration are the same as section 4.1.2. The TWR-K64F Tower System module uses PE Micro, FRDM-K64F uses CMSIS-DAP.
2. Ensure the debugger configuration is correct in the project options:
    a. Press “ALT+F7” or select “Project > Options for Target <project_name>,” you will get the Option for Target window as shown in Figure-26.
    b. In the “Options for Target …” dialog box, select the Debug tab and ensure that the simulator is not selected and the correct debug driver is selected. If the target is a TWR-K64F120M Tower System platform, the PE Micro Debugger should be selected. If the target is a Freescale Freedom FRDM-K22F platform, the CMSIS-DAP should be selected. Next, click on the “Settings button” next to the debug driver selection drop-down box. Ensure that the settings are correct for the appropriate development platform.

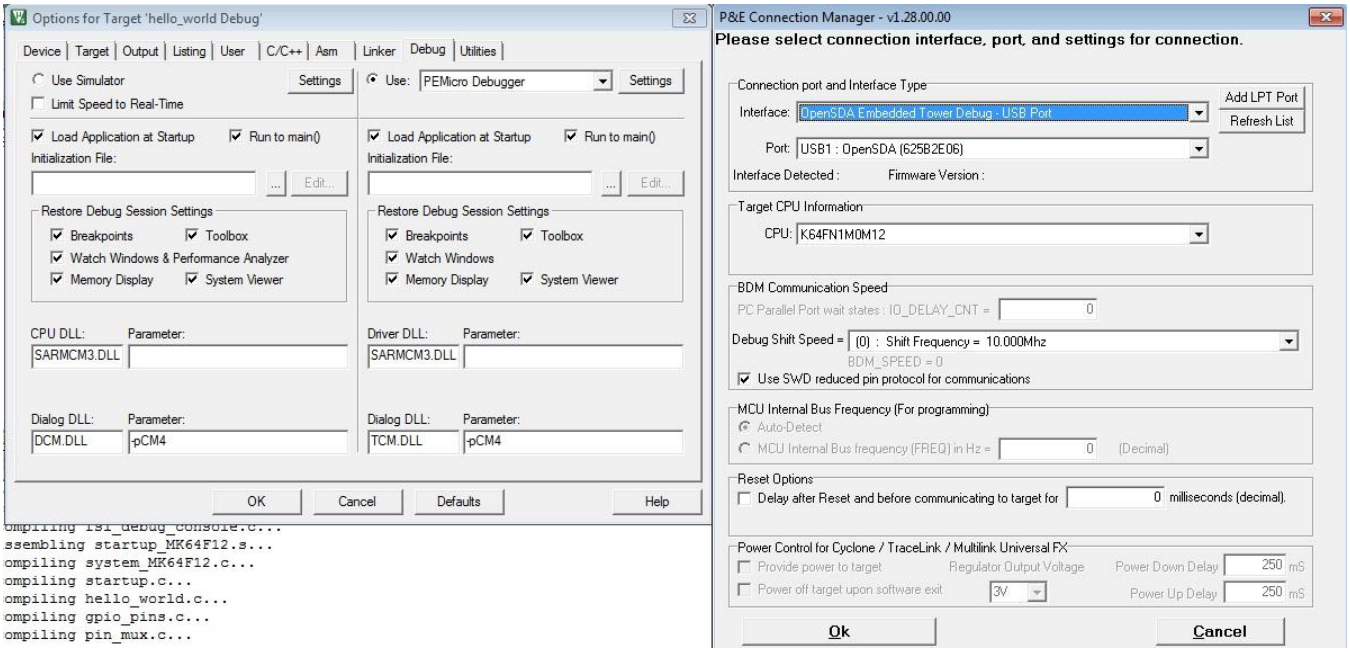

Figure-27 Debugger configurations for TWR-K64F120M Tower system module

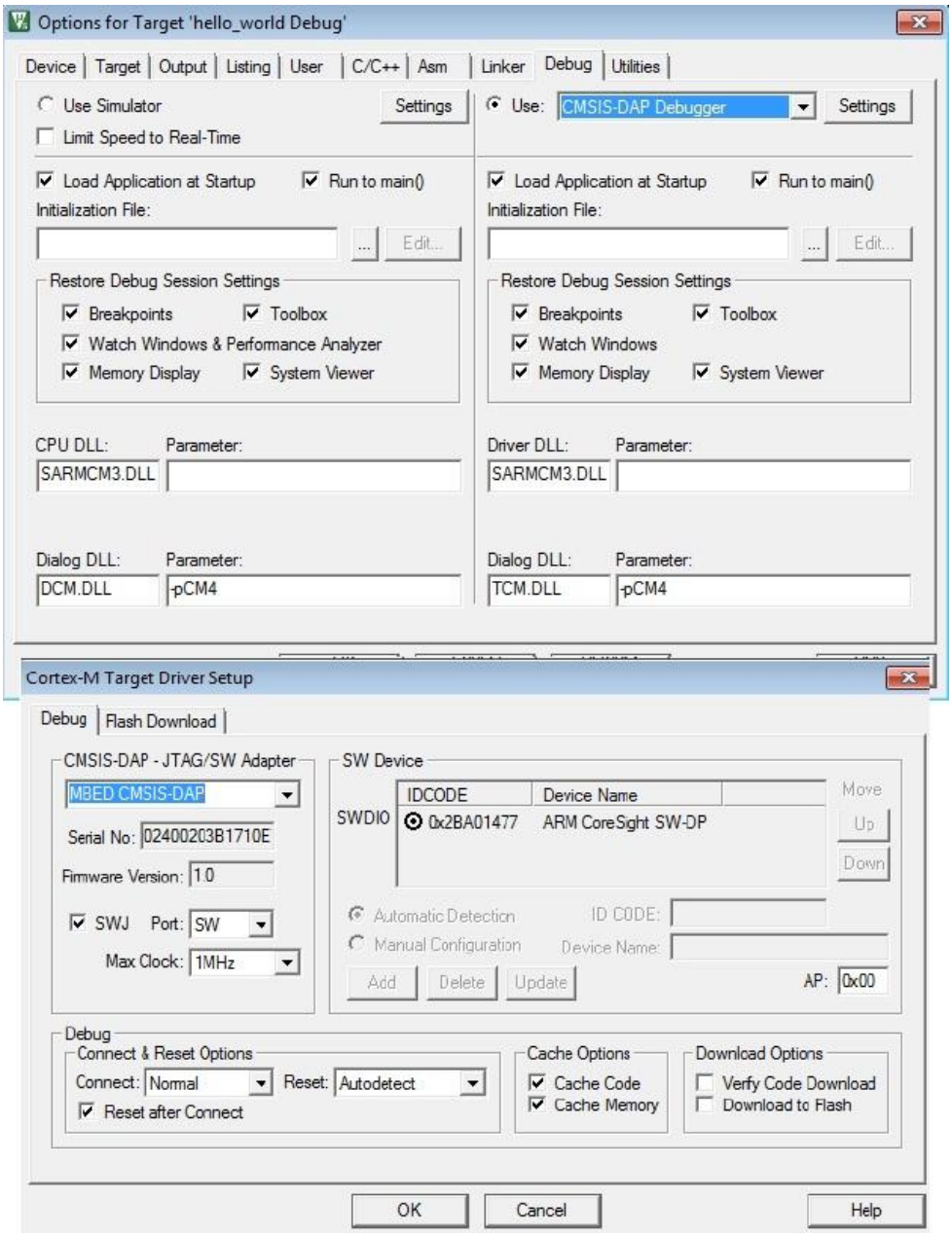

Figure-28 Debugger Configurations for Freescale Freedom FRDM-K64F platform (CMSIS DAP)

3. Configure the terminal settings as shown in Section 4.1.2-4.
4. After the terminal has successfully connected, click on the “Start/Stop Debug Session” button or press CTRL+F5 to enter debug session.

Figure-29 Start debug session

5. The debugger stops executing at the start of the main() function.

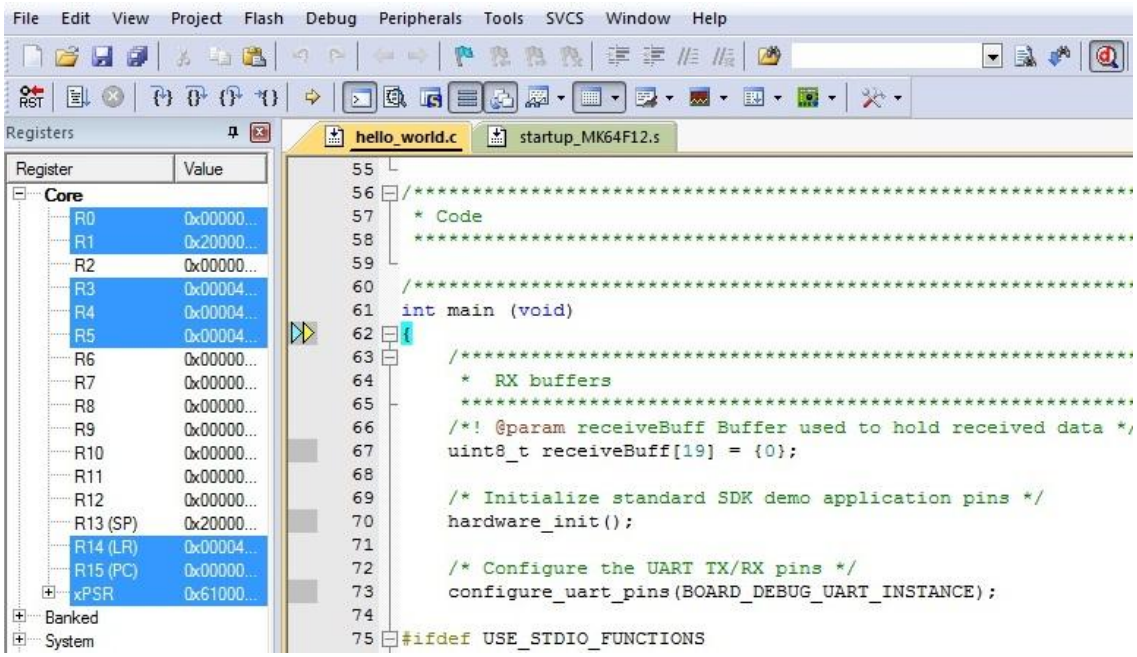

Figure-30 Stop at main() when run debugging

6. You can resume application execution by clicking on the “Run” button or press F5 to execute the application.

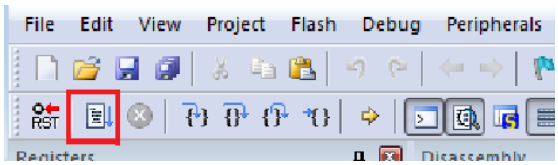

Figure-31 Run button

7. The hello_world application should now be running in PuTTy. If this is not the case, check the terminal settings and the terminal connections.

4.4 Kinetis Design Studio IDE

4.4.1 Environment Setup

Install the Kinetis Design Studio IDE (KDS) and fix the environment variable (only for Windows users. See Appendix B for details).

Before using KDS with KSDK, apply an update. Without the update, KDS cannot generate KSDK-compatible projects. To install the update, follow these instructions:

1. Select Help > Install New Software.

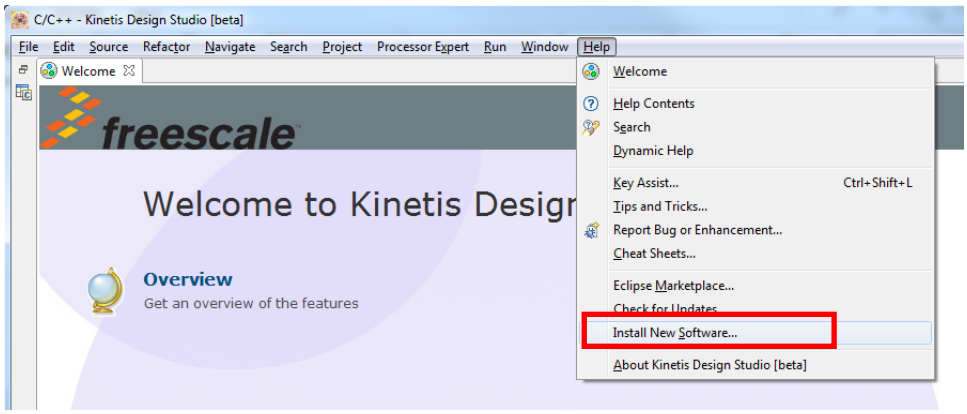

Figure 32 Install new software

2. In the "Install New Software" dialog box, select the "Add…" button in the upper right corner. In the "Add Repository" dialog, select "Archive…"

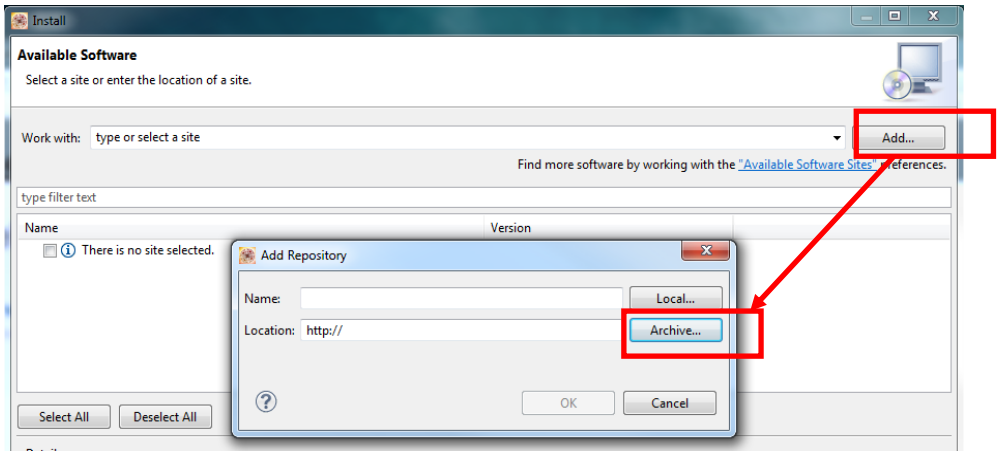

Figure 33 Add repository

3. In the dialog box that appears, browse the KSDK install directory. From the top-level, enter the tools/eclipse_update folder and select the *SDK_version_Update_for_Eclipse.zip* file.
4. Click on "Open", and then "OK" in the "Add Repository" dialog box. The KSDK update now shows up as "Processor Expert Software" → "Eclipse Update for KSDK 1.0.0-GA".

5. Check the box to the left of the KSDK Eclipse update and click “Next” in the lower right corner. Follow the remaining instructions to finish the installation of the update.

Once the update is applied, restart KDS for the changes to take effect.

4.4.2 Build a demo application

If the platform driver is already included in the lib folder and the ksdk_platform_lib.a is generated, users can directly open the demo application project and build it in KDS.

Demo application workspace files are located in:

```
<install_dir>/demos/<demo_name>/<compiler>/<board_name>
```

Take the hello_world demo application of FRDM-K64F for example. The KDS workspace file is located at:

```
<install_dir> /demos/hello_world/kds/frdmk64f120m
```

You can click File->Import to open the project as shown in Figure-34. Use browse to select root directory and KDS will automatically open the project in sub directories.

Note

Be sure not to select "Copy projects into workspace"

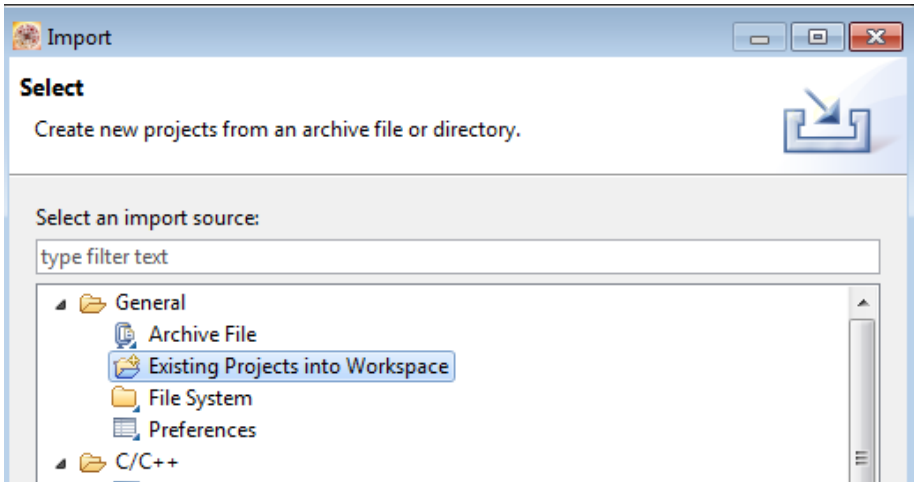

Figure-34 KDS project import

To build a demo application, click on the “build button”:

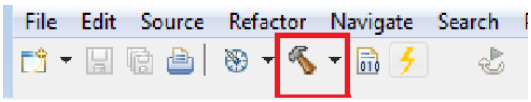

Figure-35 Build test demo application

When the build is complete, KDS displays this information in the build output window:

```
'Invoking: Cross ARM C Linker'
arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb -mfloat-abi=hard -mfpu=fpv4-sp-d16 -O0
'Finished building target: hello_world_twrk64f120m.elf'
' '

15:57:46 Build Finished (took 12s.397ms)
```

Figure-36 Build hello_world demo application successfully

4.4.3 Run a demo application

1. The serial terminal configuration is the same as Section 4.1.2.

   Because the OpenSDA JLink debug adapter is recommended for KDS, update the board debugger firmware using these steps:

   The J-Link driver installation and firmware download can refer to
   http://www.segger.com/opensda.html

   Press the reset button, then power-on, we can see a disk driver on file explorer.

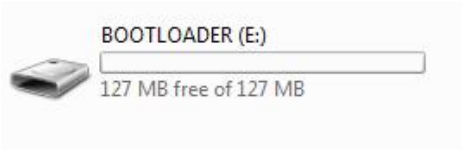

Figure-37 Display the BOOTLOADER disk driver

   Open this disk driver, then copy the firmware into the disk driver.

2. In the Project Explorer, right click on the C project containing the embedded application that users want to debug. Or in the tool bar, click Run->Debug configuration.

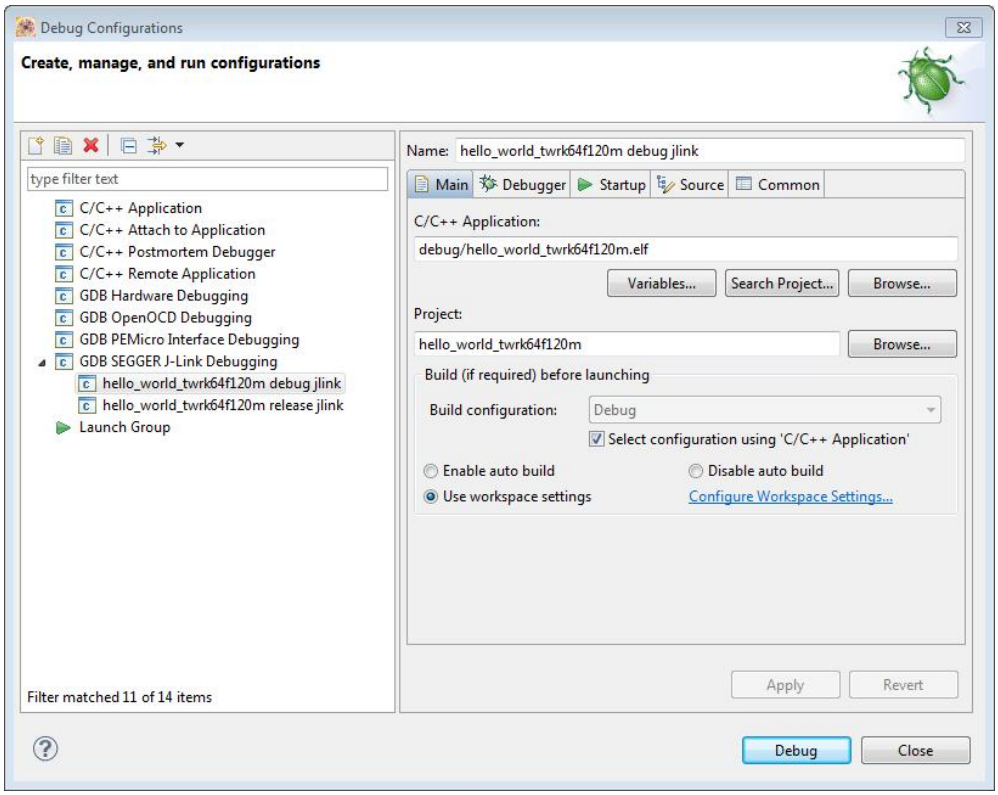

Figure-38 Open the debug configuration

3. Click the *GDB SEGGERJ-Link Debugging*, then select a debug configuration. The debug configuration has been set properly.

4. Click the Debug button. This will launch the debugger. You will be prompted to open the Kinetis Design Studio IDE's Debug perspective. Select "Yes" to switch perspective. The Debug Perspective will open.

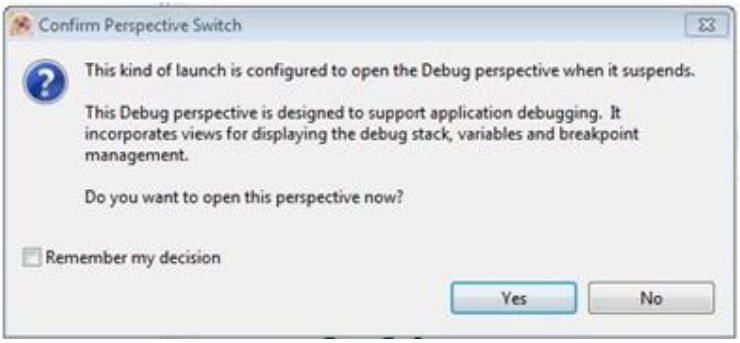

Figure-39 Confirm debug perspective

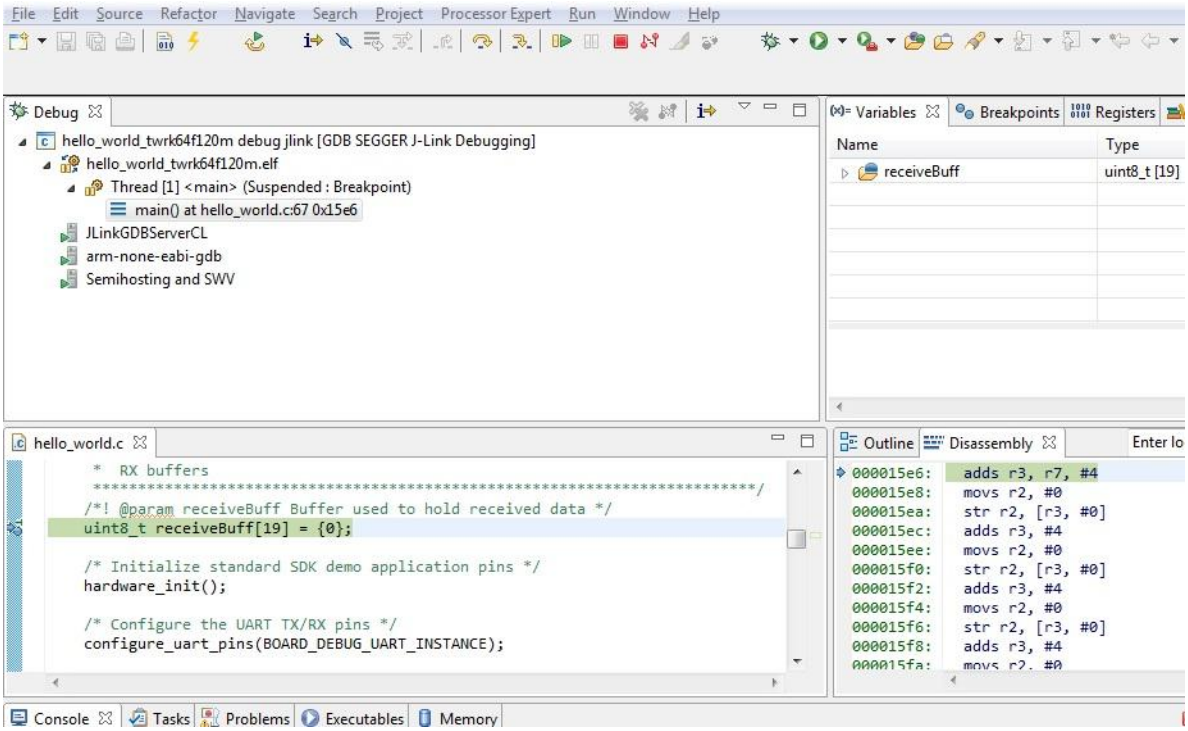

Figure-40 Debug perspective

5. Resume application execution by clicking on the “Resume” button, or press F8 to execute the application.

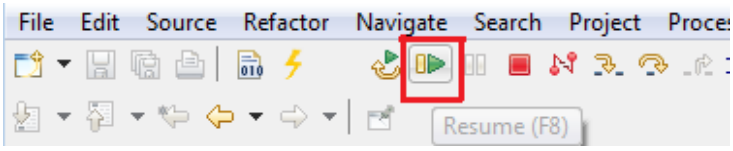

Figure-41 Run button

6. The hello_world application should now be running in PuTTy and the following banner should be displayed in the terminal. If this is not the case, check your terminal settings and the terminal connections.

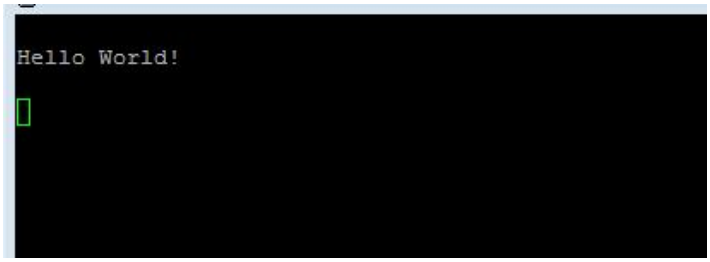

Figure-42 Hello_world demo running

Appendix A: Build the platform driver library

Before building and debugging any demo application in KSDK, the driver library project should be built to generate the library archive: ksdk_platform_lib.a. Because this library contains all binary codes for HAL and the peripheral drivers specific to the chip, each SoC has its own ksdk_platform.a library archive.

IAR: To build the platform library, open the workspace file in IAR. The platform driver library project is located in:

```
<install_dir>/lib/ksdk_platform_lib/iar/<device_name>
```

The workspace file is named ksdk_platform_lib.eww:

```
<install_dir>/lib/ksdk_platform_lib/iar/<device_name>/ksdk_platform_lib.eww
```

The project file is named ksdk_platform_lib.ewp:

```
<install_dir>/lib/ksdk_platform_lib/iar/<device_name>/ksdk_platform_lib.ewp
```

To build the platform driver library for the K64, open the workspace file in IAR:

```
< install _dir>/lib/ksdk_platform_lib/iar/K64F12/ksdk_platform_lib.eww
```

In the IAR Embedded Workbench project file, two compiler/linker configurations (build "targets") are supported:

- Debug - the compiler optimization is set to low. The debug information is generated for the binary. This target should be used for developing and debugging.
- Release - the compiler optimization is set to maximum. The debug information is not generated. This target should be used for the final application release.

Note:

Code is downloaded to Flash instead of RAM in both Debug and Release configurations.

Choose the appropriate build target: "Debug" or "Release", then click on the "Make" button" (highlighted by a red rectangle below):

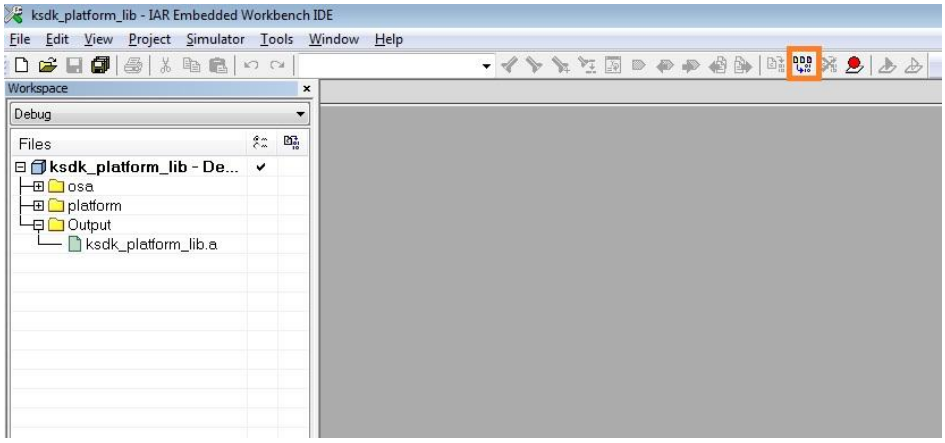

Figure-43 Platform driver library build

When the build is complete, the ksdk_platform_lib.a is generated in the directory according to the build target:

```
Debug - <install_dir>/lib/ksdk_platform_lib/iar/<device_name>/Debug
Release - <install_dir>/lib/ksdk_platform_lib/iar/<device_name>/Release
```

Demo applications use the ksdk_platform_lib.a to call the functions of the HAL and the peripheral drivers.

KEIL: To build the KEIL platform driver library, open the workspace file in KEIL. The platform driver library project is located in:

```
<install_dir>/lib/ksdk_platform_lib/uv4/<device_name>
```

The project file is named ksdk_platform_lib.uvproj:

```
<install_dir>/lib/ksdk_platform_lib/uv4/<device_name>/ksdk_platform_lib.uvproj
```

To build the platform driver library for the K64, open the project file in KEIL:

```
< install _dir>/lib/ksdk_platform_lib/uv4/K64F12/ksdk_platform_lib. Uvproj
```

In the KEIL project file, two compiler/linker configurations (build "targets") are supported:

- Debug - the compiler optimization is set to low. The debug information is generated for the binary. This target should be used for developing and debugging.
- Release - the compiler optimization is set to maximum. The debug information is not generated. This target should be used for the final application release.

Note:

Code is downloaded to Flash instead of RAM in both Debug and Release configurations.

Choose the appropriate build target: "Debug" or "Release," then click on the "Build" button:

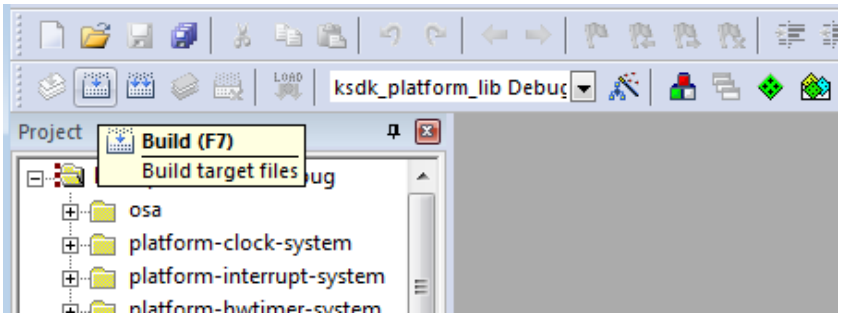

Figure-44 KEIL Platform driver library build

When the build is complete, the ksdk_platform_lib.lib is generated in the directory according to the build target:

```
Debug - <install_dir>/lib/ksdk_platform_lib/uv4/<device_name>/Debug
Release - <install_dir>/lib/ksdk_platform_lib/uv4/<device_name>/Release
```

Demo applications use the ksdk_platform_lib.lib to call the functions of the HAL and the peripheral drivers.

KDS: To build the KDS platform driver library, open the workspace file in KDS. The platform driver library project is located in:

```
<install_dir>/lib/ksdk_platform_lib/kds/<device_name>
```

To build the platform driver library for the K64, open the project file in KDS:

```
< install _dir>/lib/ksdk_platform_lib/kds/K64F12/
```

You can use File>Import to open the project.

In the KDS file, two compiler/linker configurations (build "targets") are supported, you can click the arrow in Figure-45 to view it.

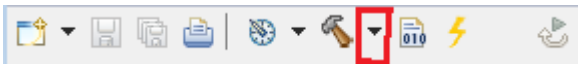

Figure-45 KDS Complier/Linker selection

- Debug - the compiler optimization is set to low. The debug information is generated for the binary. This target should be used for developing and debugging.
- Release - the compiler optimization is set to maximum. The debug information is not generated. This target should be used for the final application release.

Note:

Code is downloaded to Flash instead of RAM in both Debug and Release configurations.

Choose the appropriate build target: "Debug" or "Release," then click on the "Build" button.

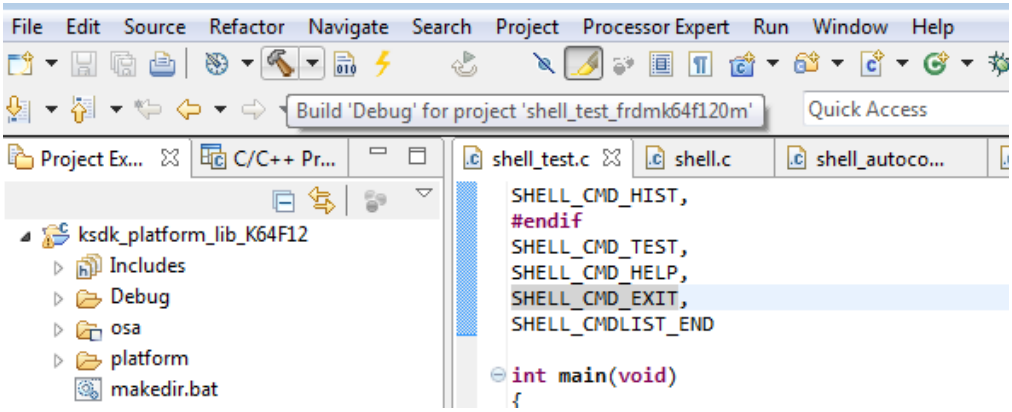

Figure-46 KDS Platform driver library build

When the build is complete, the ksdk_platform_lib.a is generated in the directory according to the build target:

```
Debug - <install_dir>/lib/ksdk_platform_lib/kds/<device_name>/Debug
Release - <install_dir>/lib/ksdk_platform_lib/kds/<device_name>/Release
```

Demo applications use the ksdk_platform_lib.a to call the functions of the HAL and the peripheral drivers.

Appendix B: Kinetis Design Studio Environment Variable Fix

First, finish the installation of the KDS-VX.X.exe in your PC, then:

- In the startup menu, right click “Computer” and choose “Properties.”
- In the left column, click “Advanced system settings.” You should get the following window:

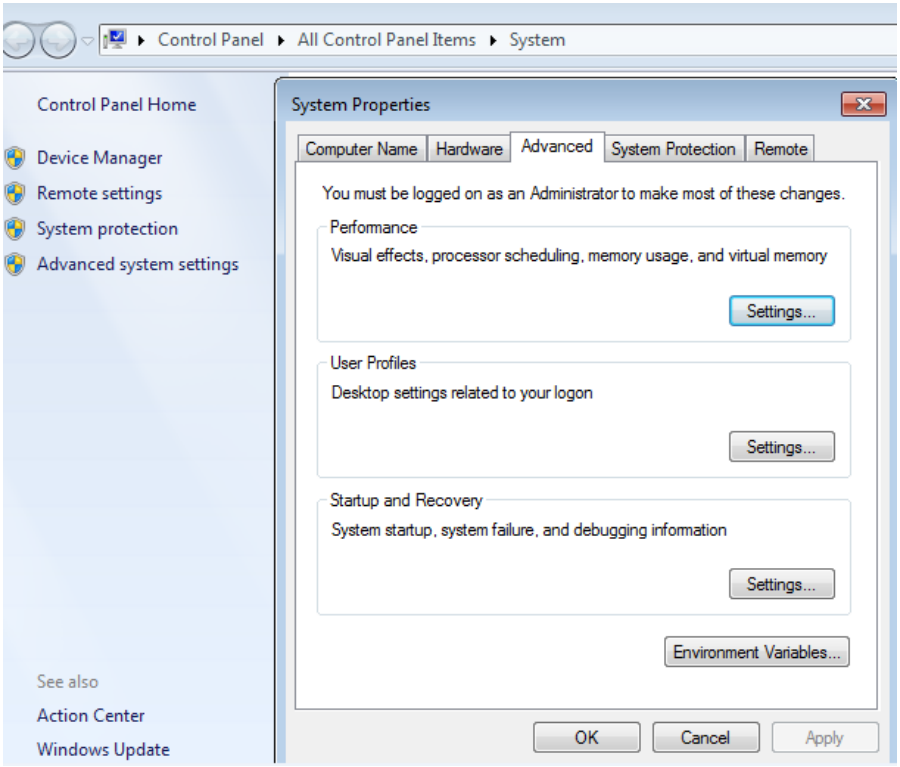

Figure-47 Computer system setting

Click the “Environment Variables” and make sure the KSDK_PATH is correct in the “User variables for <user name>”. See Figure-48 for details.

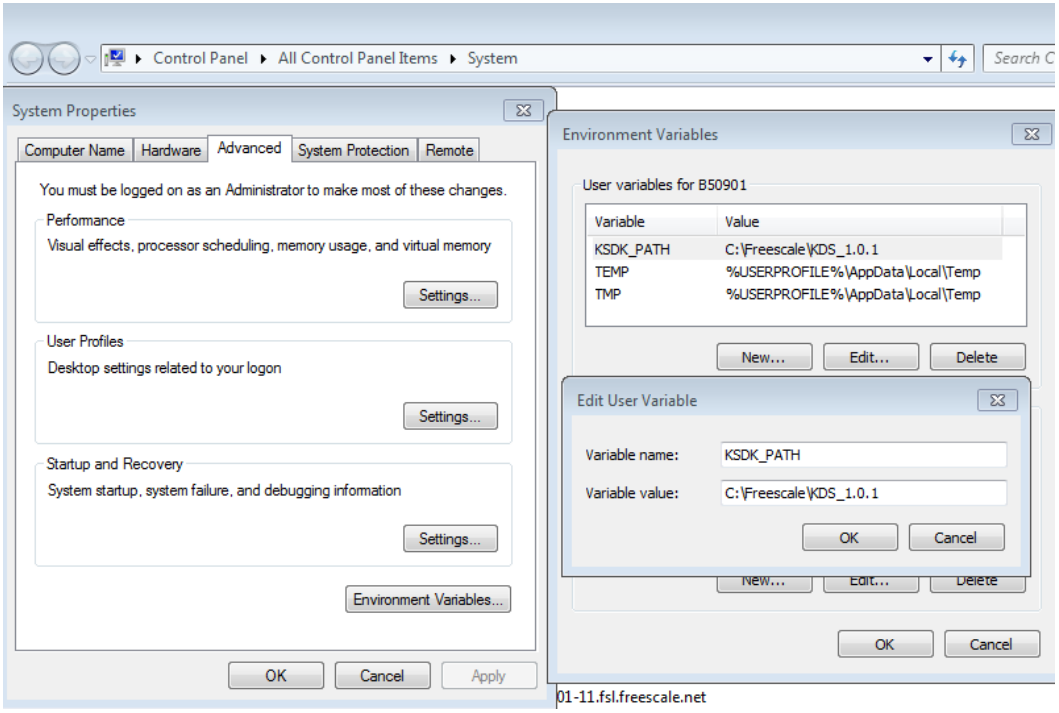

Figure-48 Environment variables for KSDK

5 Revision history

This table summarizes revisions to this document.

Revision History		
Location	**Date**	**Change description**
Entire document	7/2014	Initial release – 1.0.0

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: KSDKK64UG
Rev. 1.0.0
07/2014