NXP Semiconductors, Inc.
User's Guide

Document Number: FRTOS6XV101GSUG
Rev. 0, 07/2016

Getting Started with FreeRTOS™ BSP for i.MX 6SoloX

1 Overview

The FreeRTOS™ BSP for i.MX 6SoloX is a Software Development Kit that provides a comprehensive software support for NXP i.MX 6SoloX processor. The FreeRTOS BSP includes a set of peripheral drivers that aim at encapsulating peripheral register access and give users the maximum flexibility at the same time. The FreeRTOS BSP also includes an open source event driven preemptive RTOS – FreeRTOS Operating System (OS) and an open source Multi-core communication stack – RPMsg.

Demo and example applications are provided to demonstrate peripheral drivers, FreeRTOS kernel, RPMsg usage and to highlight the main features of the i.MX 6SoloX processor. Figure 1 highlights the layers and features of the FreeRTOS BSP for i.MX 6SoloX.

For supported toolchain versions, see the *FreeRTOS BSP v.1.0.1 for i.MX 6SoloX Release Notes* (FRTOS1016XRN).

Contents

FreeRTOS OS (www.freertos.org) is the market leading real-time operating system, and the FreeRTOS BSP for i.MX 6SoloX provides the development environment on i.MX 6SoloX, including peripheral drivers, multi-core communication stack, and FreeRTOS OS integration. The following is the architecture of the FreeRTOS BSP.

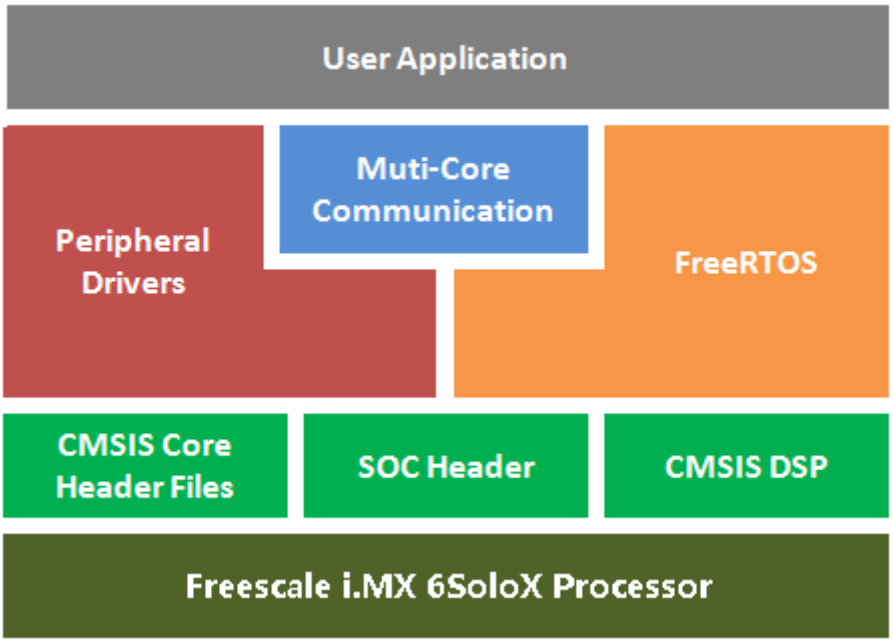

Figure 1 FreeRTOS BSP for i.MX 6SoloX architecture

The FreeRTOS BSP release provides two packages: The .exe package is a self-extract installer that can be used on Windows® OS, and the tarball is for installation on Linux® OS.

The organization of files in the FreeRTOS BSP for i.MX 6SoloX release package is focused on ease-of-use. The FreeRTOS BSP for i.MX 6SoloX folder hierarchy is organized at the top level with the folders below.

Deliverable	Location
Examples	<install_dir>/examples/...
Demo applications	<install_dir>/examples/<board_name>/demo_apps/...
Driver examples	<install_dir>/examples/<board_name>/driver_examples/...
Docmumentations	<install_dir>/doc/...
Middleware	<install_dir>/middleware/...
Peripheral Driver, Startup Code and Utilities	<install_dir>/platform/...
Cortex Microcontroller Software Interface Standard (CMSIS) ARM Cortex®-M header files, DSP library source and lib files.	<install_dir>/platform/CMSIS/...
Processor header file	<install_dir>/platform/devices/<device_name>/include/...
Linker script for each supported toolchain	<install_dir>/platform/devices/<device_name>/linker/...
CMSIS compliant Startup Code	<install_dir>/platform/devices/<device_name>/startup/...
Peripheral Drivers	<install_dir>/platform/drivers/...
Utilities such as debug console	<install_dir>/platform/utilities/...
FreeRTOS Kernel Code	<install_dir>/rtos/FreeRTOS/...
External useful tools	<install_dir>/tools/...

Figure 2 FreeRTOS BSP for i.MX 6SoloX directory structure

2 FreeRTOS BSP Demo and Example Applications

The FreeRTOS BSP provides two types of software applications:

- **Demos**: Applications intended to highlight key functions of the ARM® Cortex®-M4 Core in i.MX 6SoloX SoC, focusing on a particular use case on FreeRTOS OS.
- **Examples**: Simple applications intended to concisely illustrate how to use the peripheral drivers of the FreeRTOS BSP in the bare metal environment.

This section describes how the demo and example applications interact with other components of the FreeRTOS BSP. To get a comprehensive understanding of all FreeRTOS BSP components and folder structure, see the *FreeRTOS BSP i.MX 6SoloX API Reference Manual* (FRTOS6XAPIRM).

2.1 Locating demo source files

When opening a demo or example application in any of the supported toolchains, there are a variety of source files referenced. It is important to understand the location of these source files in the FreeRTOS BSP tree. Therefore, if needed, they can be copied or modified to help develop applications for custom hardware later on. Additionally, many files are shared and, if modified, impact other demos. As a result, the user should have a full grasp of the FreeRTOS BSP structure to fully understand the effect of manipulating the source files.

There are four main areas under install directory of the FreeRTOS BSP tree used to provide the full source code for each demo application:

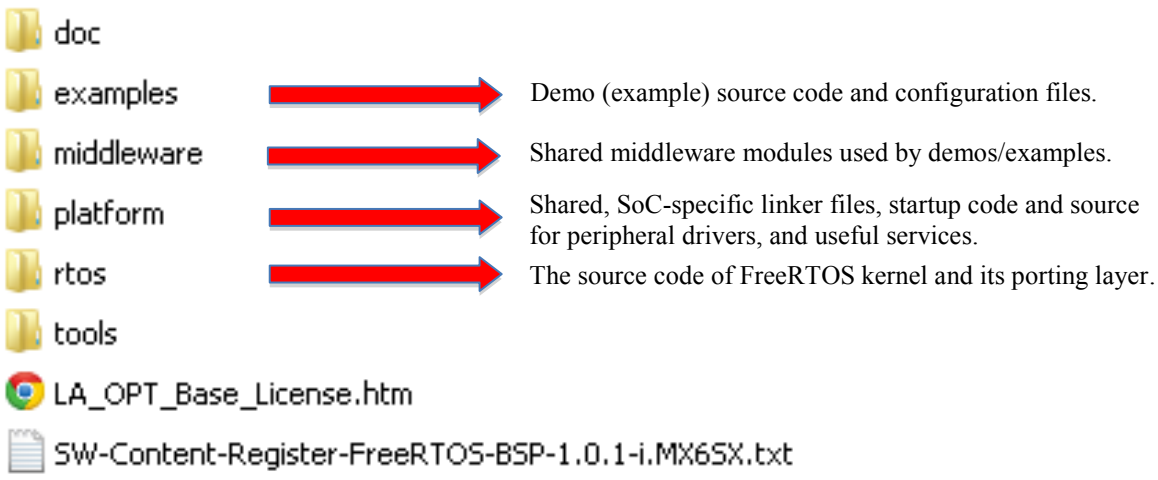

Figure 3 Root folder structure of FreeRTOS BSP for i.MX 6SoloX

2.2 FreeRTOS BSP examples folder

The examples folder contains all demos/examples source code and board configuration files. All applications are cataloged by board name. The directory structure of examples folder is shown as follows.

Figure 4 Examples folder structure

At the top level of each board folder, there is a common set of files used by the demos and examples. These files can be modified to perform operations such as changing the pin mux configuration. All board support files are provided as part of the FreeRTOS BSP:

- **board.c/h**: The header file contains board-specific configuration macros for things such as debug terminal configuration, push buttons, LEDs and other board-specific items. The C file contains clock and RDC initialization functions.
- **clock_freq.c/h**: Contains functions that are used to get current clock frequency of a specified peripheral such as GPT, I2C, or UART.
- **gpio_pins.c/h**: Definitions used by the FreeRTOS BSP GPIO driver for the platform's GPIO pins. These include push buttons and LEDs, but can include other items such as interrupt pins for external sensors.
- **pin_mux.c/h**: Contains peripheral-specific pin mux configurations. These functions can be called by the hardware_init() function or individually by the demo application.
- **Shared_clock_node**: Contains the index of shared clock between Cortex-A9 Core and Cortex-M4 Core. These values are used in dual-core power control mechanism.

2.3 FreeRTOS BSP middleware folder

The middleware folder contains the source code of all the middleware used by demos/examples. The open source multi-core communication stack – RPMsg is included in the middleware folder.

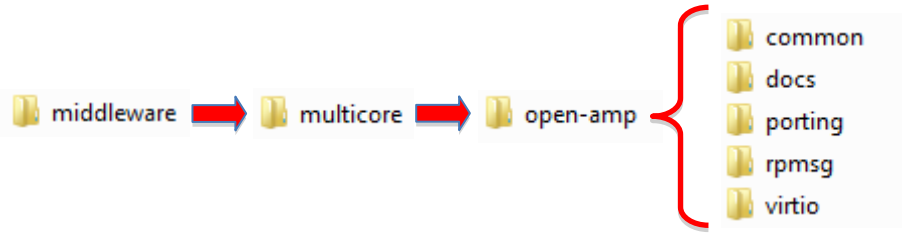

Figure 5 Middleware folder structure

2.4 FreeRTOS BSP platform folder

The platform folder contains the source code for the primary components including CMSIS header files, peripheral drivers, startup, utilities, and linker files.

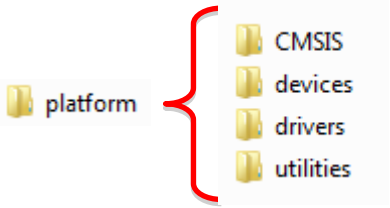

Figure 6 Platform folder structure

2.5 FreeRTOS BSP rtos folder

The rtos folder contains the source code of market leading RTOS – FreeRTOS OS:

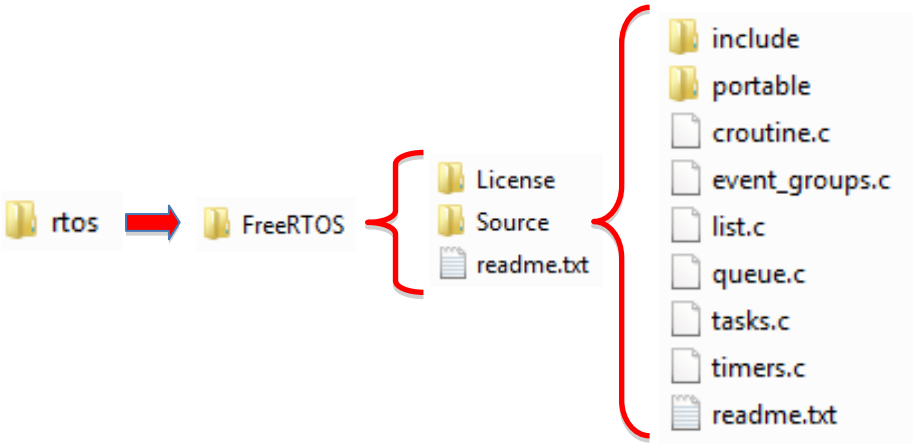

Figure 7 rtos folder structure

3 Building a Demo Using IAR Embedded Workbench IDE

This section describes the steps required to build demo applications provided in the FreeRTOS BSP. The hello_world demo for i.MX 6SoloX SABRE-SD board is used as an example, though these steps can be applied to any board, demo or example application in the FreeRTOS BSP.

1. Open the demo application workspace files located in the following path:

   <install_dir>/examples/<board_name>/demo_apps/<demo_name>/iar

   Using the i.MX 6SoloX SABRE-SD board as an example, the hello_world workspace is located in this folder:

   <install_dir>/examples/imx6sx_sdb_m4/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.

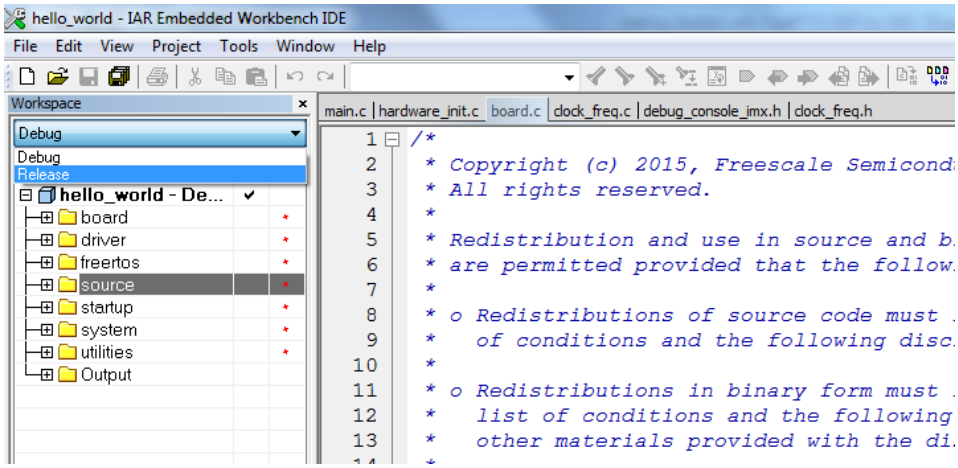

Figure 8 Demo build target selection

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

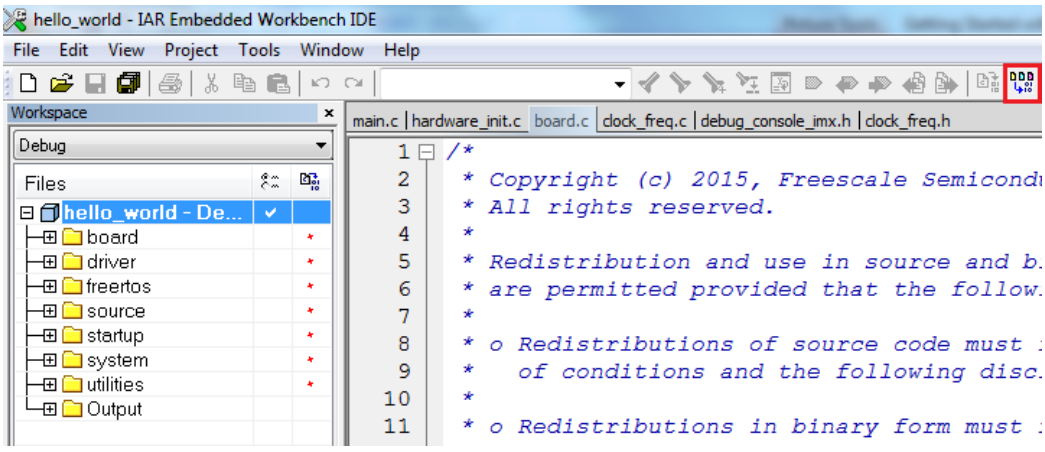

Figure 9 Build the demo application

4. The build completes without errors.

5. The build result can be found at *<install_dir>/examples/<board_name>/demo_apps/<demo_name>/iar/<build_conguration>*:

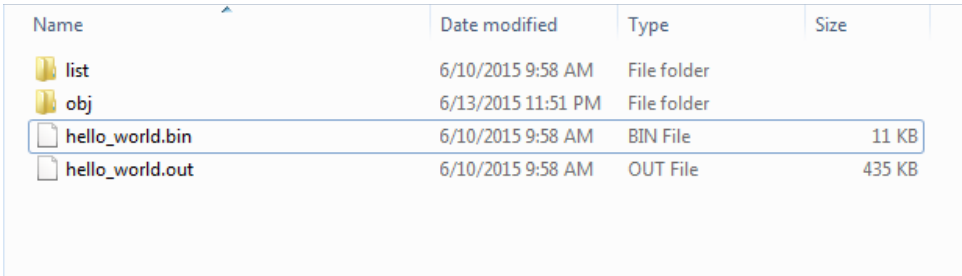

Figure 10 Build result

- The *.out file contains the debug information of the demo application. It can be used for software debugging.
- The *.bin file is the demo application binary file; it can be loaded and run on the target board using U-Boot;

4 Building a Demo Using DS-5 IDE

This section describes the steps required to build demo applications provided in the FreeRTOS BSP. The hello_world demo for i.MX 6SoloX SABRE-SD board is used as an example, though these steps can be applied to any board, demo or example application in the FreeRTOS BSP.

1. Open ARM DS-5 IDE installed on your PC:

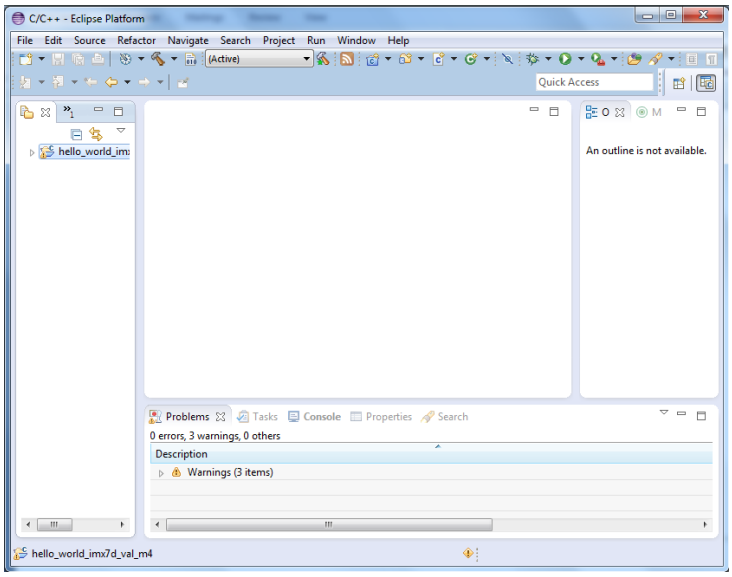

Figure 11 ARM DS-5 overview

2. Select “File->Import” from the DS-5 IDE menu. In the window that appears, expand the “General” folder and select “Existing Projects into Workspace”. Then, click the “Next” button.

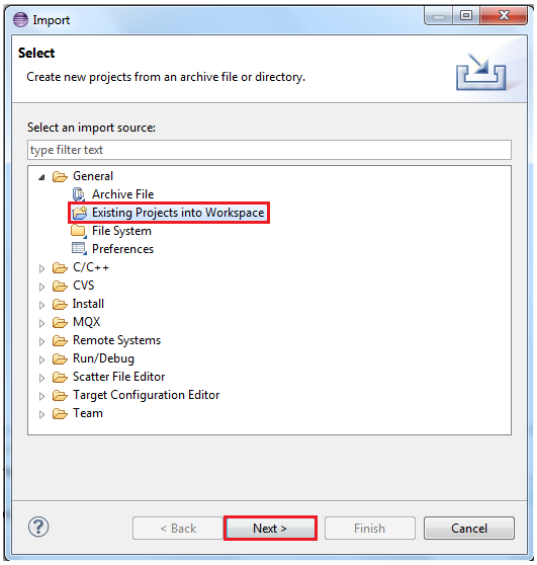

Figure 12 Selection of the correct import type in DS-5 IDE

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

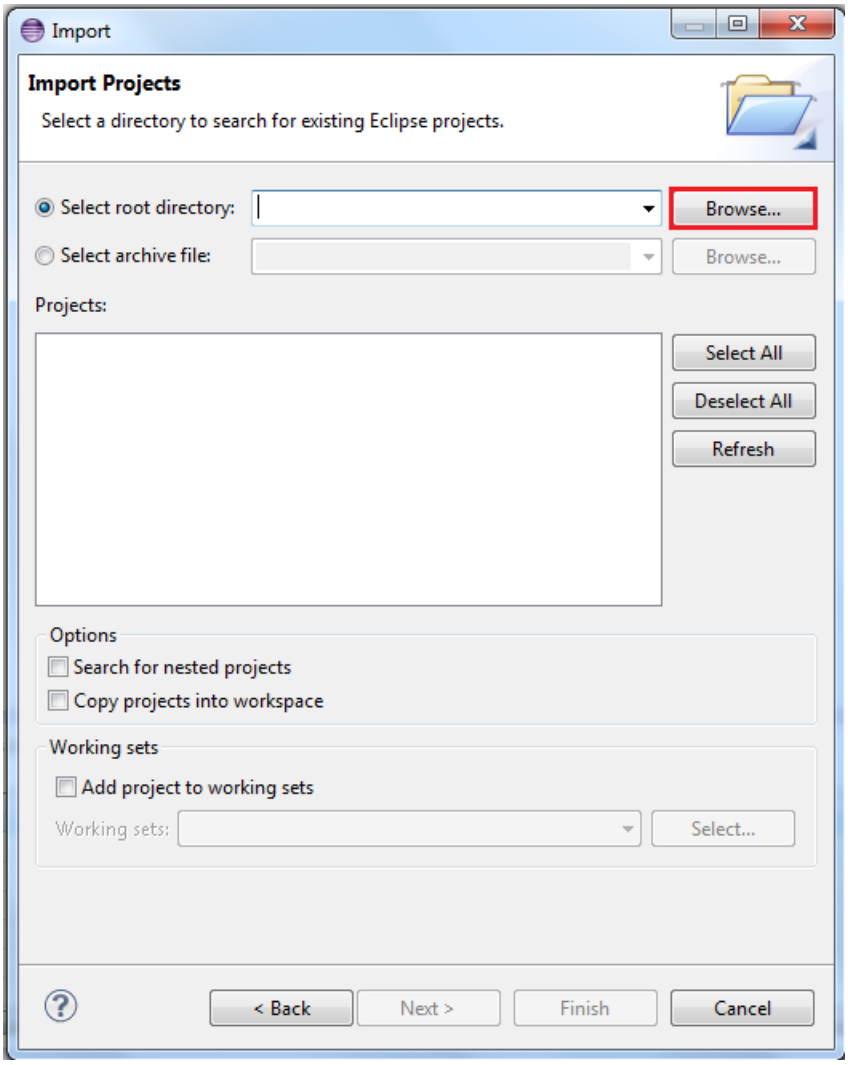

Figure 13 Projects directory selection window

4. Point to the hello_world demo for the appropriate device, which can be found using this path:

   <install_dir>/examples/<board_name>/demo_apps/hello_world/ds5

   For this example, the specific location is:

   <install_dir>/ examples/imx6sx_sdb_m4/demo_apps/hello_world/ds5

5. After pointing to the correct directory, your “Import Projects” window should look like the figure below. Click the “Finish” button.

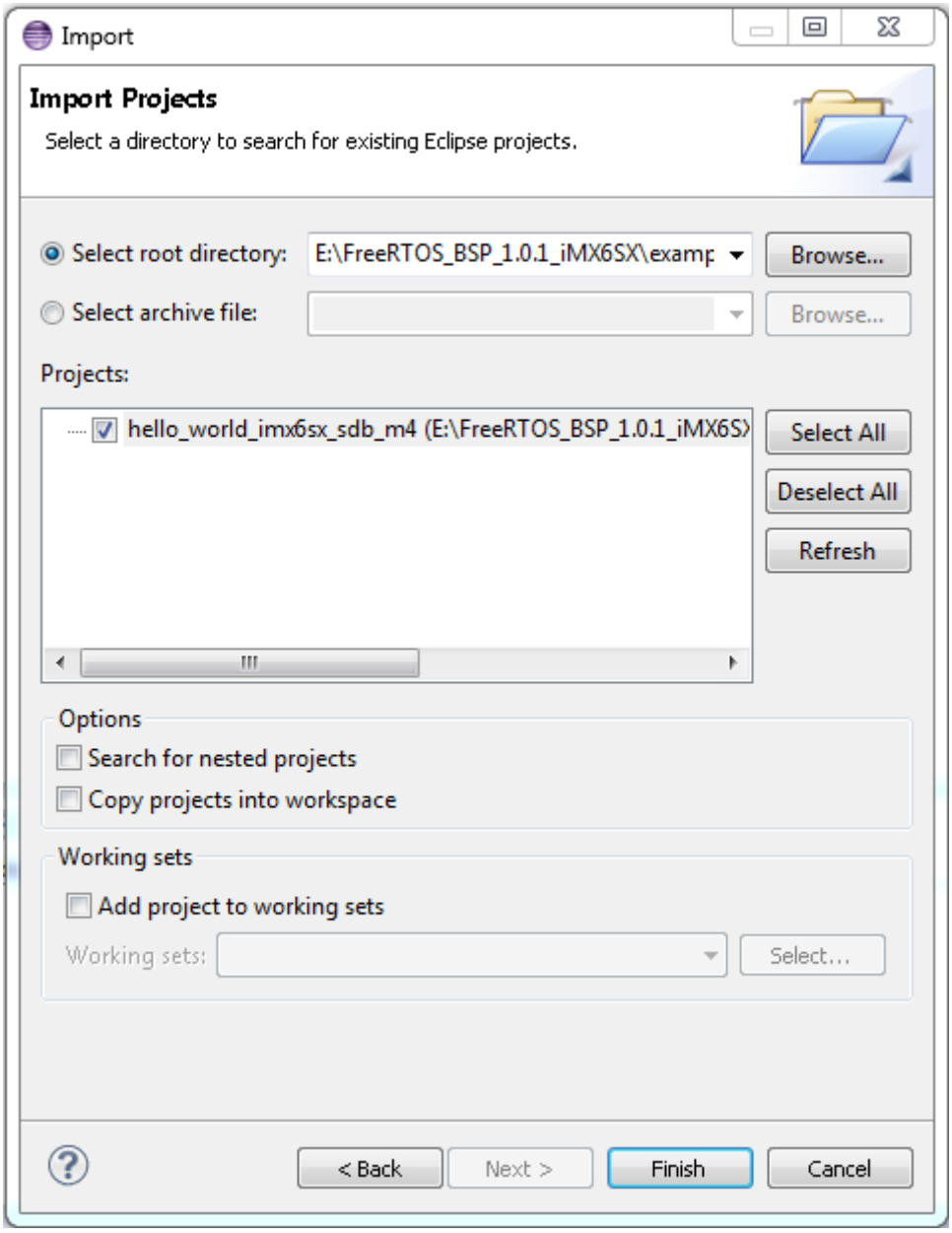

Figure 14 Select hello_world project

6. There are two project configurations (build targets) supported for each 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.
7. 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.

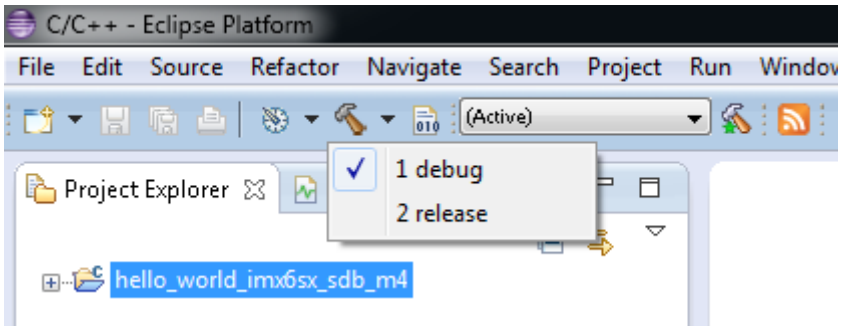

Figure 15 Selection of the build target in DS-5 IDE

8. The demo starts building after the build target is selected. To rebuild the demo in the future, click the hammer icon (assuming the same build target is chosen).
9. The build result can be found at *<install_dir>/examples/<board_name>/demo_apps/<demo_name>/ds5/<build_conguration>*.
    - The *.axf file contains the debug information of the demo application, it can be used for software debugging;
    - The *.bin file is the demo application binary file; it can be loaded and run on the target board using U-Boot;

5 Building a Demo Using ARM® GCC

This section describes the steps to configure the command line ARM GCC tools to build demo applications. The hello_world demo application targeted for the i.MX 6SoloX SABRE-SD board hardware platform is used as an example, though these steps can be applied to any board, demo or example application in the FreeRTOS BSP.

5.1 Setting up the toolchain

This section contains the steps to install the necessary components required to build and run a FreeRTOS BSP demo application with the ARM GCC toolchain, as supported by the FreeRTOS BSP. 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.

5.1.1 Installing the GCC ARM Embedded tool chain

Download and run the installer from launchpad.net/gcc-arm-embedded. This is the actual toolset, such as compiler and linker.

5.1.2 Installing 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 FreeRTOS BSP does not use the MinGW build tools, but does leverage the base installation 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.

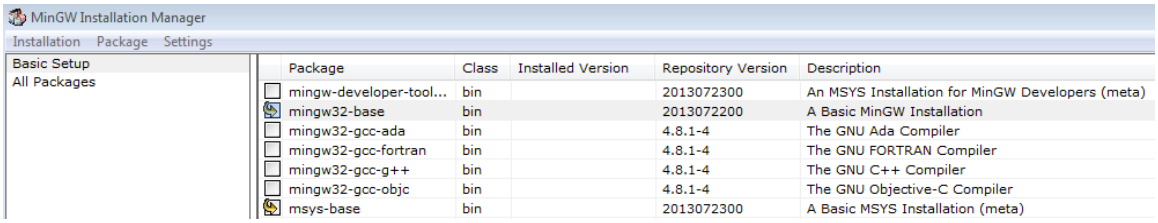

Figure 16 Setting up MinGW and MSYS

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

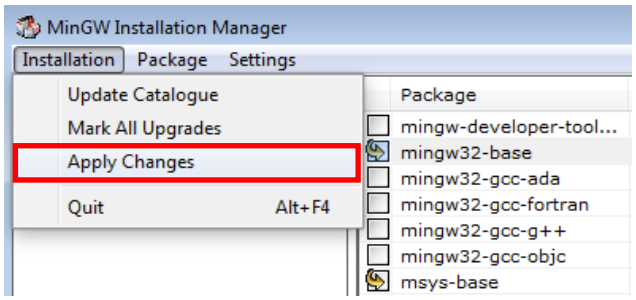

Figure 17 MinGW and MSYS installation complete

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 FreeRTOS BSP), remove it to ensure that the new GCC build system works correctly.

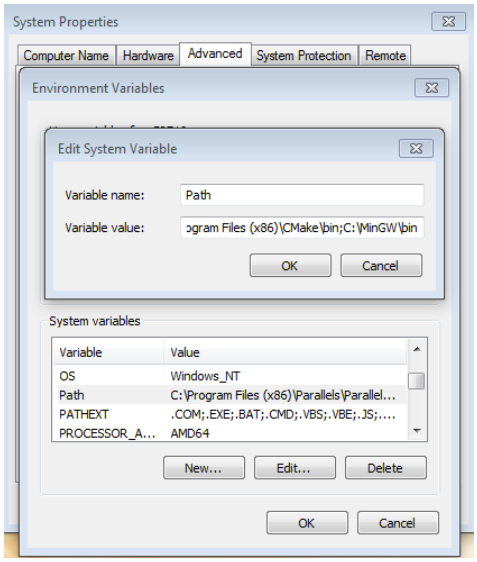

Figure 18 Adding Path to systems environment

5.1.3 Adding a new system environment variable 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 2015q1

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

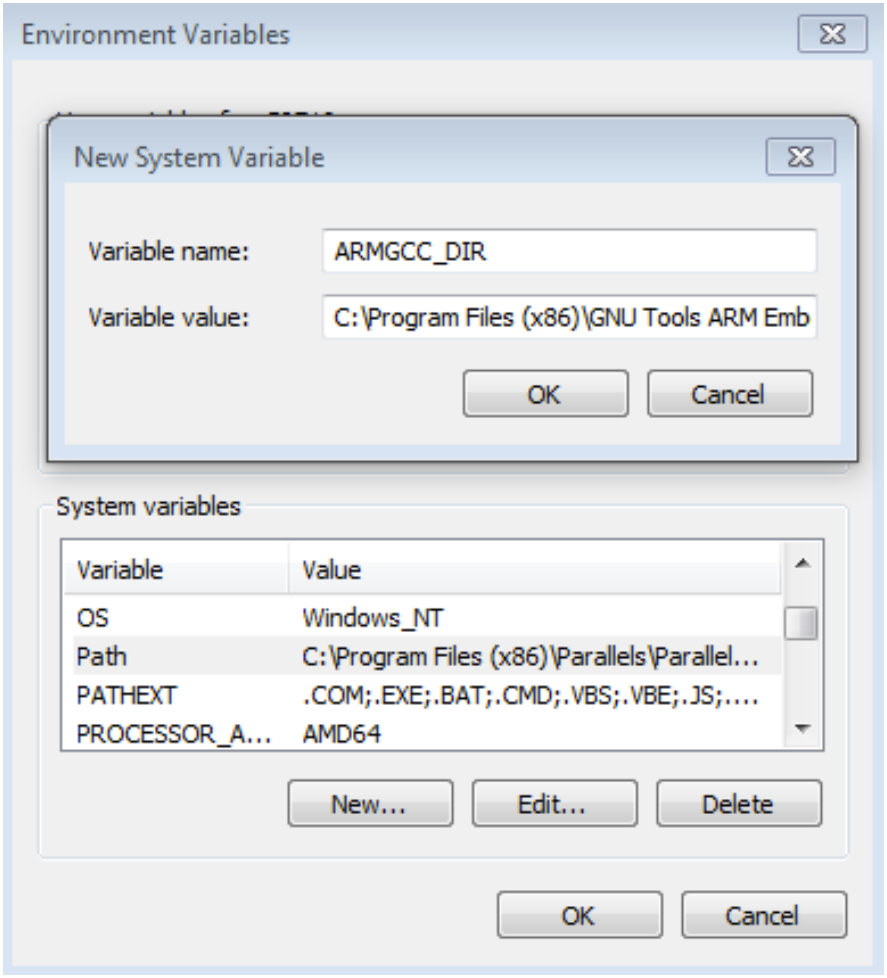

Figure 19 Adding ARMGCC_DIR system variable

5.1.4 Installing 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. It's up to the user to select whether it's installed into the PATH for all users or just the current user. In this example, the assumption is that it's installed for all users.

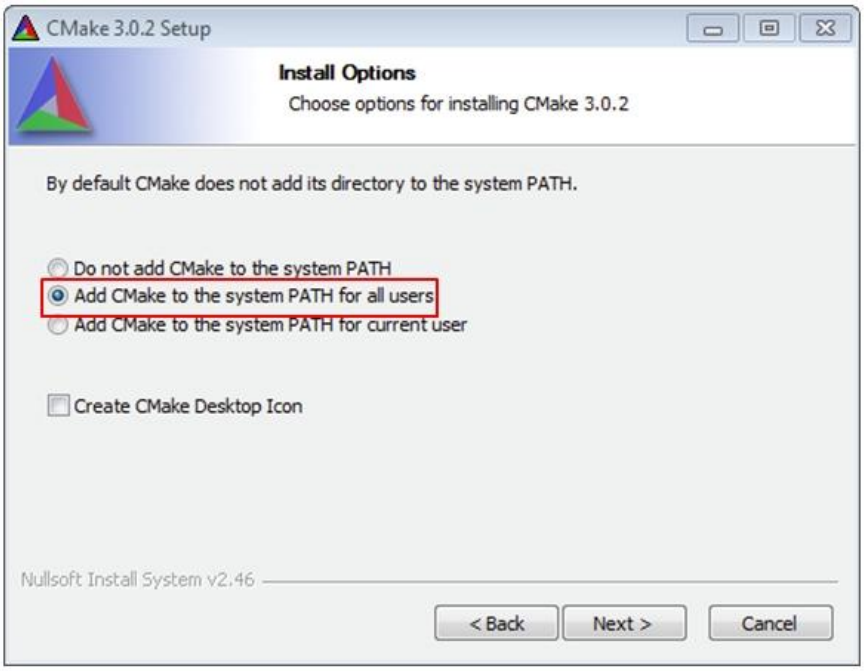

Figure 20 Installing CMake

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

5.2 Building the hello world demo application

To build the demo application, follow these instructions:

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".

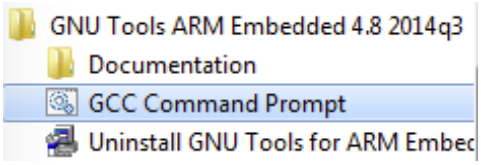

Figure 21 Launch command prompt

2. Change the directory of the command window to the hello world demo application directory in the FreeRTOS BSP: *<install_dir>/examples/<board_name>/demo_apps/hello_world/armgcc/*

3. There are two project configurations (build targets) supported for each FreeRTOS BSP 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.

   There are batch files provided to build both configurations. For this example, the "Debug" target is built and "build_debug.bat" is typed on the command line. If the "Release" target is desired, type the "build_release.bat" instead.

   Alternatively, if using the command line is not desired, you can double click on the batch files from Windows operating system Explorer.

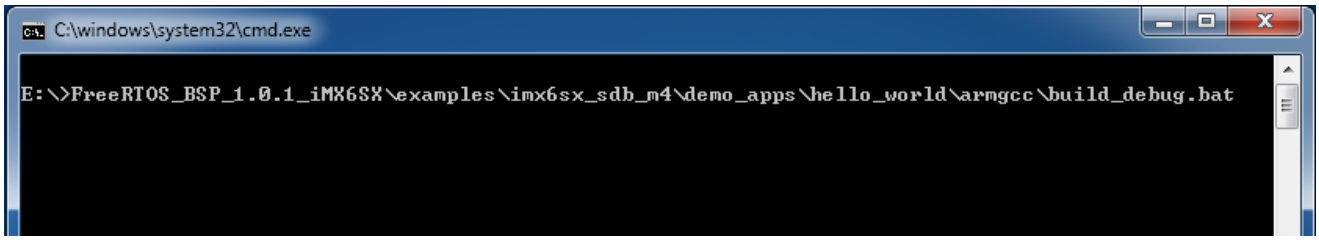

Figure 22 Build debug version of platform library

4. When the build finishes, the output looks like the image below.

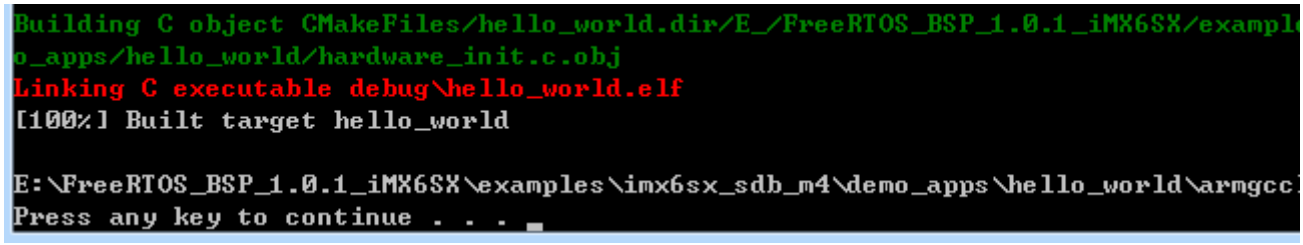

Figure 23 Hello world application builds successful

5. The demo application is generated in one of these directories, according to the build target:

   <install_dir>/examples/imx6sx_sdb_m4/demo_apps/hello_world/armgcc/debug

   <install_dir>/examples/imx6sx_sdb_m4/demo_apps/hello_world/armgcc/release

6. The build result can be found at *<install_dir>/examples/<board_name>/demo_apps/<demo_name>/armgcc/<build_conguration>*.
   - The *.elf file contains the debug information of the demo application, it can be used for software debugging;
   - *.bin file is the demo application binary file; it can be loaded and run on the target board using U-Boot;

6 Running Application with U-Boot

This section describes the steps to run application using a SD card with prebuilt U-Boot image for i.MX processor. The prebuilt U-Boot image can be found in Linux BSP bundle or package for i.MX 6SoloX processor. For more information about how to write the U-Boot image to SD card and create FAT file system partition, see the Linux BSP package.

6.1 Running application on TCM with U-Boot

1. Preparing an SD card with prebuilt U-Boot image from Linux BSP packet for i.MX 6SoloX processor.
2. Insert the SD card to the PC, and copy the application image (for example hello_world) you want run to the FAT partition of the SD card.

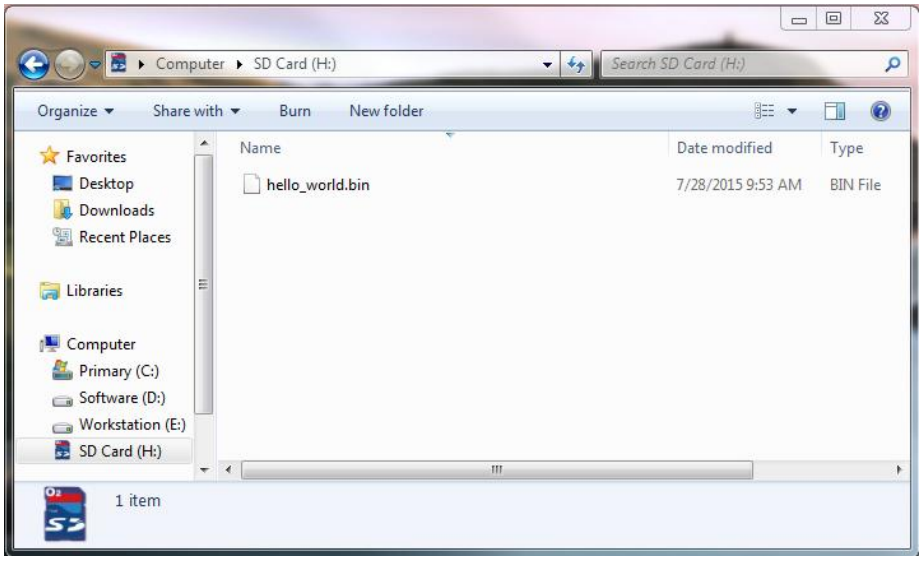

Figure 24 Copying firmware image to SD Card FAT partition

3. Safely remove the SD card from the PC.
4. Insert the SD card to the target board. Make sure to use the default boot SD slot and double check the dip switch configuration. The default configuration of the SABRE-SD board boots from SD4, and on SABRE-AI board there is only one SD slot which is used for boot.
5. Connect the “DEBUG UART” slot on the board with your PC through the USB cable. The Windows OS installs the USB driver automatically, and the Ubuntu OS finds the serial devices as well.
6. On Windows OS, open the device manager, find “USB serial Port” in “Ports (COM and LPT)”. Assume that the ports are COM9 and COM10. The smaller numbered port (COM9) is for the debug message from Cortex A9 and the larger numbered port (COM10) is for Cortex-M4.
   On Ubuntu OS, find the TTY device with name /dev/ttyUSB* to determine your debug port. Similar to Windows OS, the smaller number is for Cortex®-A9 and the bigger number is for Cortex-M4.

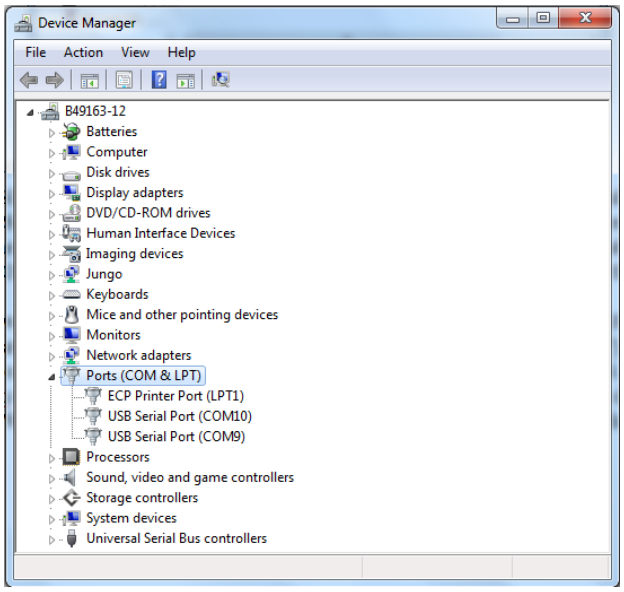

Figure 25 Determining the COM Port of target board

7. Open your favorite serial terminals for the serial devices, setting the speed to 115200 bps, data bits 8, no parity, and power on the board.
8. On the COM9 terminal, press any key within 3 seconds of booting, and U-Boot enters command line mode, and then you can run your application from TCM with following commands:
    a. **fatload mmc n:1 0x80000000 hello_world.bin**: Load the application image from the SD card to DDR RAM. (**n=2** on SABRE-SD board and **n=0** on SABRE-AI board)
    b. **dcache flush**: Flush cached content to DDR RAM.
    c. cp.b 0x80000000 0x7F8000 0x8000:Copy Cortex-M4 image from DDR RAM to TCM.
    d. **dcache flush**: Flush cached content to TCM.
    e. **bootaux 0x7F8000**: Start the Cortex-M4 core from the TCM.

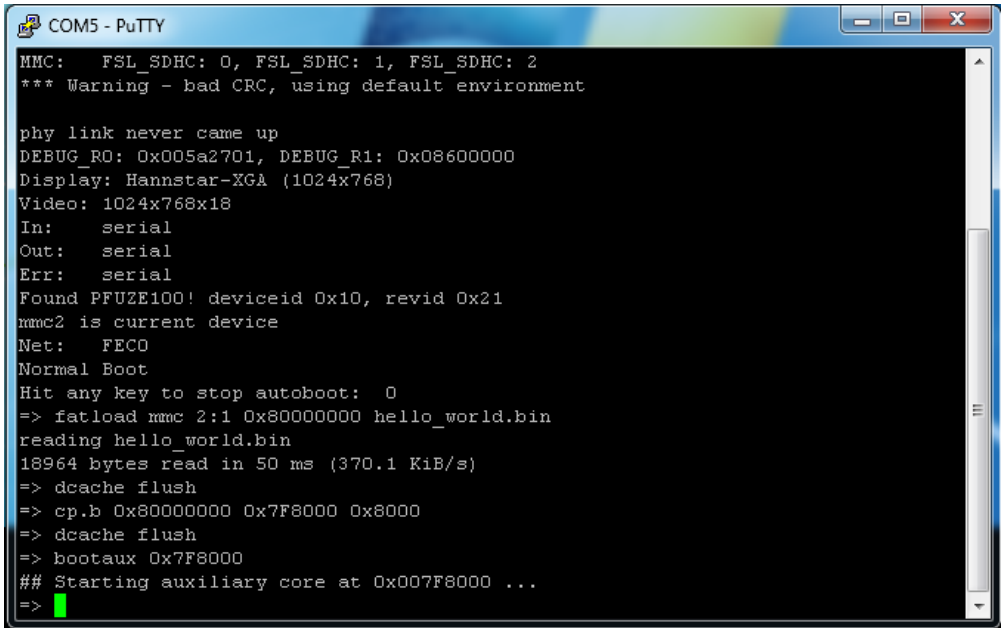

Figure 26 U-Boot cmd to run application on TCM

9. Now, you can see that your application is started on Cortex-M4 Core through COM 10:

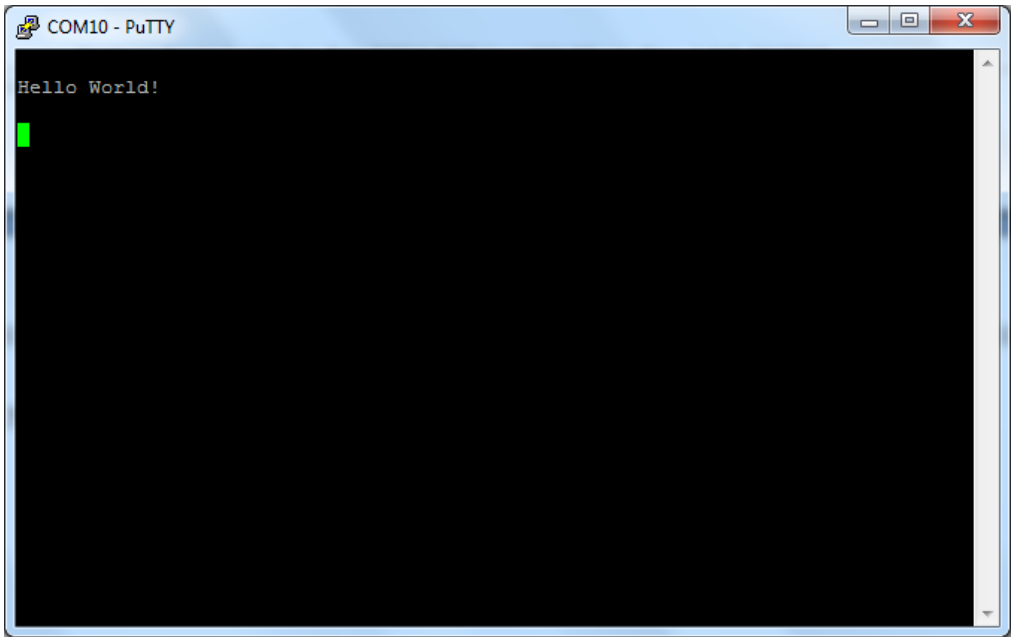

Figure 27 FreeRTOS OS hello world demo running on Cortex-M4 Core TCM

6.2 Running application on DDR/OCRAM with U-Boot

Some applications with the names ended by “ddr” or "ocram" in FreeRTOS BSP should be run in DDR or OCRAM.

To run application from DDR/OCRAM, please follow the steps below:

1. The first 7 steps are same to running application from TCM as mentioned above.
2. After finish the first 7 steps, power on the board and enter to U-Boot command line mode, then you can run with following commands:
    - DDR
        a. **fatload mmc n:1 0x9ff00000 hello_world_ddr.bin:** Load the application image from the SD card to DDR RAM. (**n=2** on SABRE-SD board and **n=0** on SABRE-AI board)
        b. **dcache flush**: Flush cached content to DDR RAM.
        c. **bootaux 0x9ff00000:** Start the M4 core from the DDR.
    - OCRAM
        - **fatload mmc n:1 0x00910000 hello_world_ocram.bin:** Load the application image from the SD card to OCRAM. (**n=2** on SABRE-SD board and **n=0** on SABRE-AI board)
        - **dcache flush:** Flush cached content to OCRAM.
        - **bootaux 0x00910000:** Start the M4 core from the OCRAM.

    The application is started on ARM Cortex-M4 Core through COM 10.

6.3 Running application on QSPI with U-Boot

Some applications with the names ended by "qspi" in FreeRTOS BSP should also boot from external storage device like QSPI Flash.

To run application from QSPI Flash, please follow the steps below:

1. The first 7 steps are same to running application from TCM as mentioned above. Make sure that the U-Boot has QSPI enabled.
2. After finish the first 7 steps, power on the board and enter to U-Boot command line mode, then you can write image and run it from QSPI Flash with the following commands:
    - For i.MX 6SoloX SABRE-SD board:
        - **fatload mmc 2:1 0x80000000 hello_world_qspi.bin:** Load the flash image file from the SD card to DDR RAM.
        - **sf probe 1:0:** Load the SPI flash driver.
        - You should get the message "**SF: Detected N25Q256 with page size 256 Bytes, erase size 4 KiB, total 32 MiB**".
        - **sf erase 0x0 0x10000:** Erase the first 64 KB in the flash.
        - **sf write 0x80000000 0x0 0x10000:** Burn the image from DDR RAM to the QSPI Flash.
        - **bootaux 0x78000000:** Start the M4 core at the flash head.
    - For i.MX 6SoloX SABRE-AI board:
        - **fatload mmc 0:1 0x80000000 hello_world_qspi.bin:** Load the flash image file from the SD card to DDR RAM.
        - **sf probe 1:0:** Load the SPI flash driver.
        - You should get the message "**SF: Detected N25Q256 with page size 256 Bytes, erase size 4 KiB, total 32 MiB**".
        - **sf erase 0x0 0x10000:** Erase the first 64 KB in the flash.
        - **sf write 0x80000000 0x0 0x10000:** Burn the image from DDR RAM to the QSPI Flash.
        - **bootaux 0x68000000:** Start the M4 core at the flash head.

    The application is started on ARM Cortex-M4 Core through COM 10.

7 Debugging Application with Trace32 Debugger

Although IAR and DS5 are supported to build the FreeRTOS BSP and examples, the debugger part of these IDE tools are not enabled. Only TRACE32 is able to debug the programs built from ARM-GCC, IAR and DS5.

1. To debug the program with TRACE32, a script file is used. Find it at `<install_dir>/tools/trace32/attach_imx6sx_m4.cmm`.
2. Make sure that TRACE32 ICD (In-Circuit-Debugger) for ARM is installed, and your Lauterbach debugger device supports Cortex-A9 and Cortex-M4 debugging.
3. Build a FreeRTOS application of the RAM target, and change the default ELF load path in `attach_imx6sx_m4.cmm`: *data.load.elf "<your ELF path>" /verify*.
4. Connect the TRACE32 debugger device to your PC and the board (through JTAG). Run the TRACE32 ICD ARM debugger, and load `attach_imx6sx_m4.cmm` by choosing "**File -> Run Batchfile**".

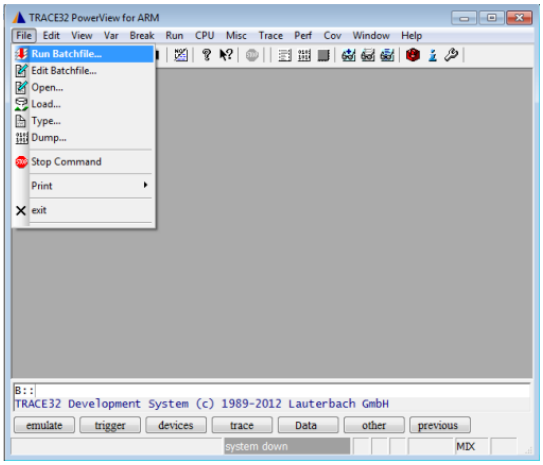

Figure 28 Running the batch file in Trace32

5. Now you can run (GO) and debug the program with the single step (Step, Over, Next, Return) or break points.

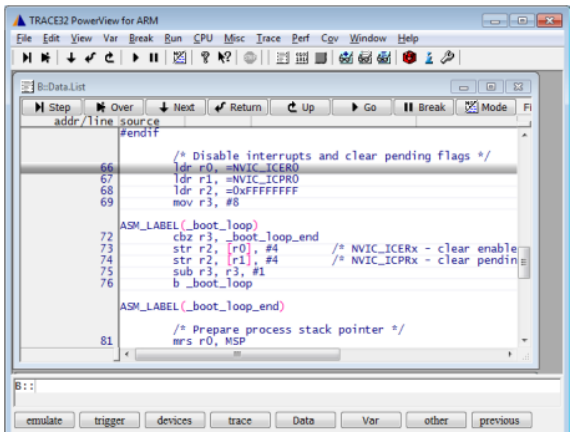

Figure 29 Starting debugging

8 Revision History

This table summarizes the revisions made to this document.

Revision number	Date	Substantive changes
0	07/2016	Initial release.

How to Reach Us:

Home Page:
www.nxp.com

Web Support:
www.nxp.com/support

Document Number: FRTOS6XV101GSUG
Rev. 0
07/2016