Document Number: MQXGSDS5
Rev. 07, 04/2015

Getting Started with ARM® Development Studio 5 (DS-5™) with Freescale MQX™ RTOS

PRODUCT:	Freescale MQX™ RTOS
PRODUCT VERSION:	4.2.0
DESCRIPTION:	Using ARM® Development Studio 5 (DS-5™) with Freescale MQX™ RTOS
RELEASE DATE:	April, 2015

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

Document Number: MQXGSDS5
Rev. 07, 04/2015

Table of Contents

1 Read Me First

This document describes how to build, debug, and run Freescale MQX™ RTOS programs in the ARM® Development Studio 5 (DS-5™) development suite.

See *Getting Started with Freescale MQX™ RTOS* and other user documentation included within the latest Freescale MQX RTOS distribution for further information on board specific build targets, jumper and HW settings, MQX RTOS API documentation, etc.

2 Building the MQX RTOS library and example application.

This chapter concentrates on steps specific to Development Studio 5 (DS-5) tool chain only. For details on generic build process and compile time configuration, see Chapter 2 of the *Getting Started with Freescale MQX™ RTOS*.

First, install the MQX RTOS Eclipse plugin using ***Help\Install New Software\Add\Archive…*** menu. Then, select the archive : `<mqx_install_dir>/tools/ds5/ds5_update_site.zip`, install this archive by following the DS-5 package install wizard.

For MQX RTOS v4.2.0 each example/demo application comes along with one workspace file which contains the path to the example/demo project file and all the dependent MQX library project files. Import the workspace file into the DS-5 workspace by following these steps for example with MQX RTOS.

Then, using ***File\Import\MQX\Import Working Sets*** menu, import the workspace file in the folder – for example with MQX RTOS example:

`<mqx_install_dir>/mqx/examples/<example>/build/ds5/<example>_<board_name>/<example>_<board_name>.wsd`

The MQX RTOS library projects will be imported to DS-5 working space together with build configurations settings.

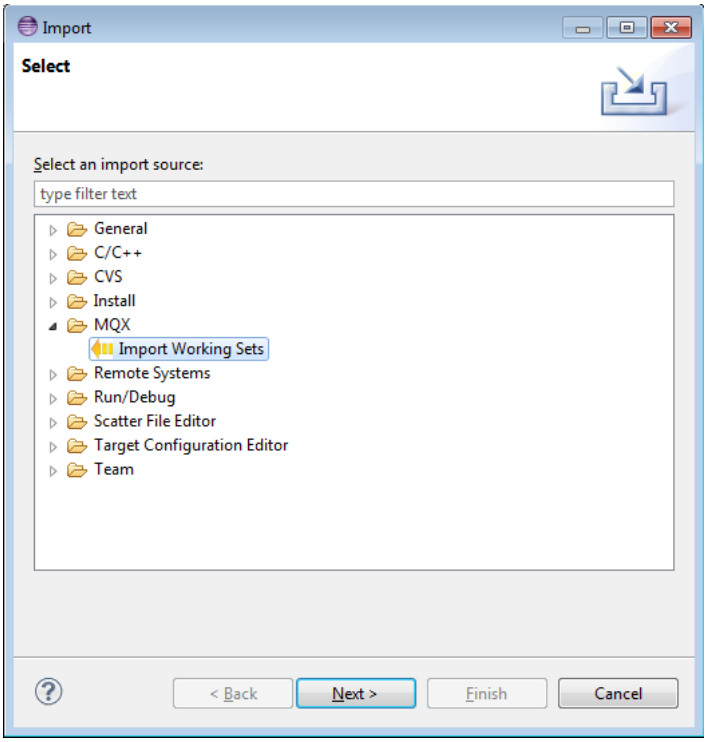

Figure 1: MQX RTOS import working sets

Before building and running any application, all libraries that the application requires must be built successfully. In this case, for example `hello` application will be used in the next chapter to demonstrate a generic build process, so in the imported workspace, you have to finish building

bsp_twrvf65gs10_a5 and *psp_twrvf65gs10_a5* libraries. These libraries come along with the application in the imported workspace.

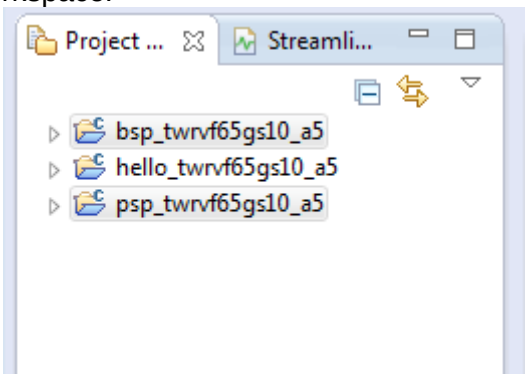

Figure 2: Hello application example workspace

2.1 Vybrid-kit special licenses

In addition to the regular DS5 licenses there are two special Vybrid licenses available.

- Vybrid edition license (http://ds.arm.com/vybrid/vybrid-edition/)
- Vybrid-tower-starter-kit license (ds.arm.com/vybrid/vybrid-tower-starter-kit/)

If you are using any of these licenses please be aware of the following limitations:

- The licenses are “node locked” and the tool cannot be accessed with the Windows® OS Remote Desktop.
- Code size limitation (1 MB for Vybrid-edition, 256 KB for Vybrid-tower-starter-kit)
- Vectorization – this feature is not supported in older DS5.14 with the special Vybrid license. Users have to turn off this feature manually (uncheck checkbox in IDE or use “--vectorize” option). This results in a less optimized code. The DS5.15 users are not affected.
- To use the command line build tools (with make), an additional compiler option is needed and the scatter file needs to be updated. Add the following options to the command line and update the scatter file by adding the same option to the end of the first line.
- Vybrid edition: "--tool_variant=vf6xx_tk"
- Vybrid-tower-starter-kit: "--tool_variant=vf6xx_sk"

3 Running and Debugging MQX RTOS application

The description bellow is provided for Vybrid microcontrollers BSPs - twrvf65gs10_a5 and twrvf65gs10_m4 and Hello World example application. The twrvf65gs10_a5 BSP runs on primary Vybrid ARM® Cortex®-A5 core and twrvf65gs10_m4 runs auxiliary Vybrid ARM® Cortex®-M4 core. The same procedure applies for all other BSPs and example applications distributed in the MQX RTOS release package.

3.1 Debugging Primary Core - MQX RTOS Hello World program

- Connect a serial cable to the TWR-SER or TWR-SER2 board DB9 connector. Set the communication speed to 115200.
- Select menu ***File/Import/General/Existing Projects into Workspace*** and import Hello World example application.

<mqx_install_dir>/mqx/examples/hello/build/ds5/hello_twrvf65gs10_a5/.project

- Click the compile button  to build application ***Int. RAM Debug target***.
- Click the arrow next to the Debug button and select ***Debug Configurations***.

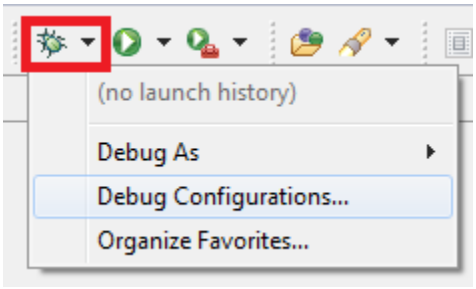

Figure 3: Debug configurations

- A dialog box will come up. Select the ***hello_twrvf65gs10_a5_Int_Ram_Debug*** configuration, then choose the ***Vybrid VF6xx / Bare Metal Debug/ Debug Cortex-A5 CMSIS-DAP*** and CMSIS-DAP Target connection. Then click the **Debug** button in the lower right corner.

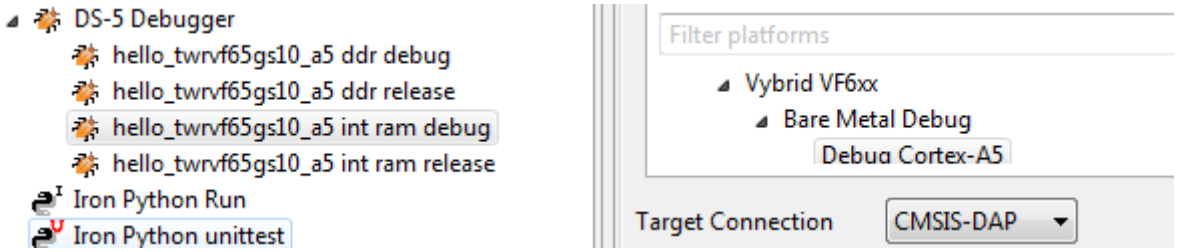

Figure 4: Debug

- The Development Studio 5 (DS-5) will switch to Debug Perspective automatically. Then, the project will be loaded to the device and execution will stop in the main() function.

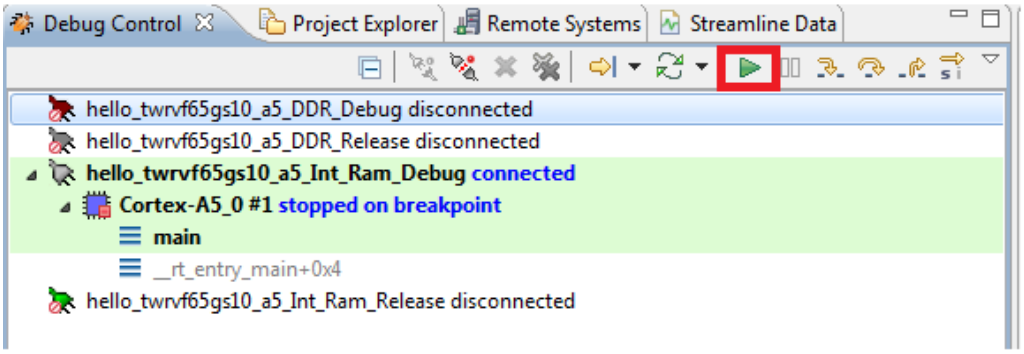

Figure 5: Debug control

- Press the **Run** button to continue in the Hello World program execution.
- The program prints Hello World on serial console terminal.

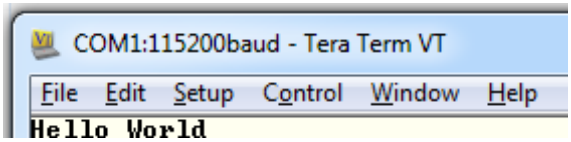

Figure 6: Hello World console

- To debug the application again, push ***Interrupt*** button to stop program execution**.** Then press ***Disconnect from Target*** and ***Connect from Target*** buttons in ***Debug Control*** Menu.

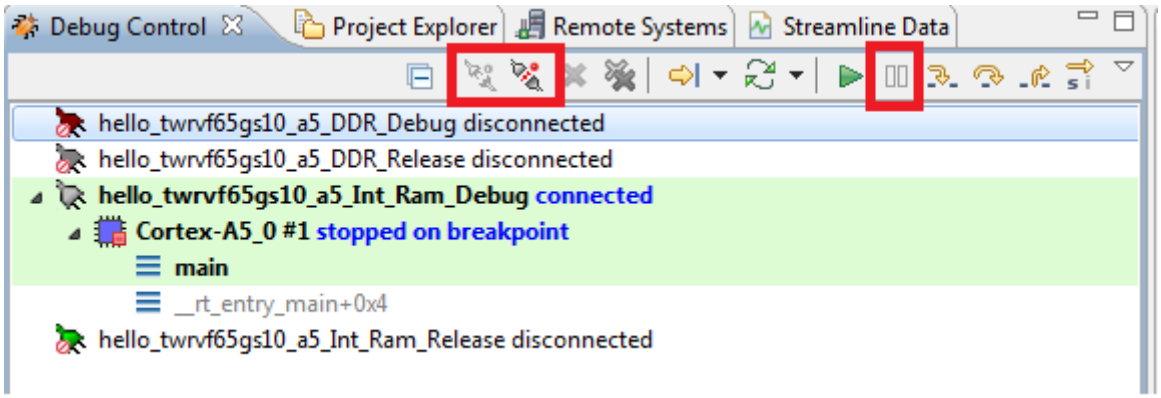

Figure 7: Debug again

Note: If subsequent connections to the target fail, it is recommended to reset the board by using the Reset button on TWR or by using PWR down/up sequence on the TWR elevator.

3.2 Run MQX RTOS Hello World program on auxiliary core on dual core system (Vybrid ARM Cortex-M core)

- Before loading the application to the primary core as described in the previous chapter, it is important to check the **Enable secondary clock on connection** in the **DTSL Options** menu (accessible from the **Debug Configuration** dialogue).

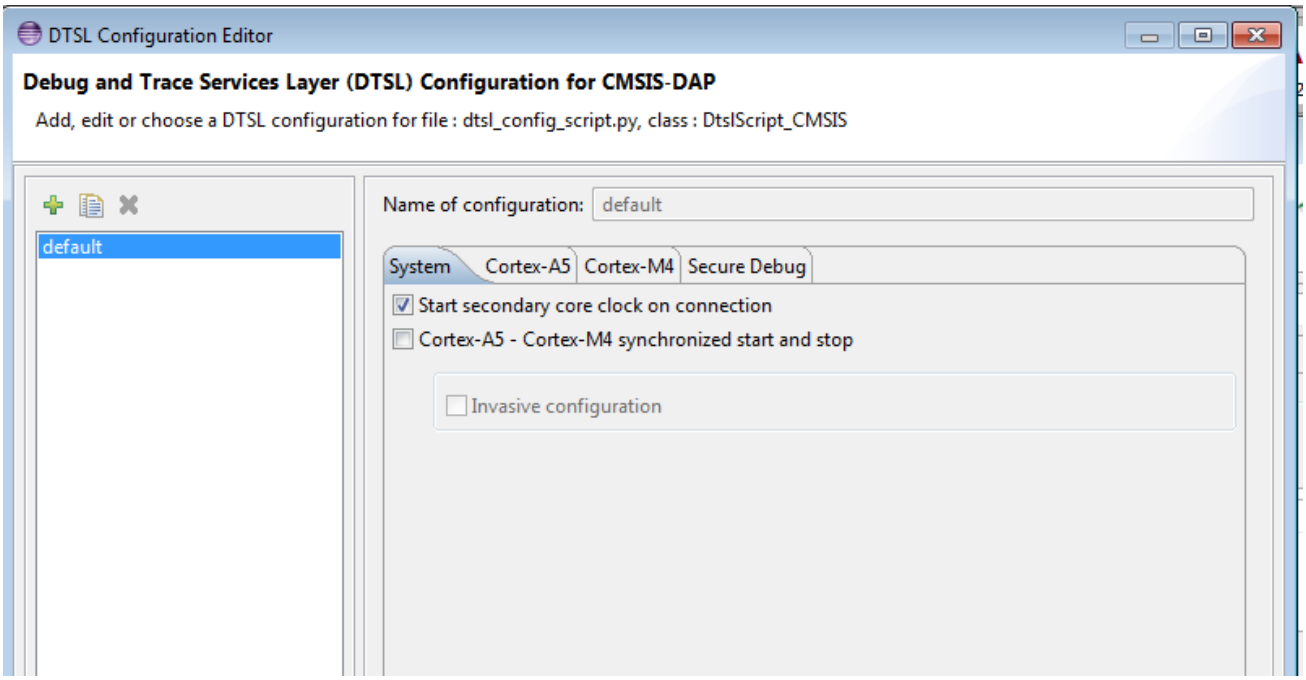

Figure 8: DTSL configuration

- Run the primary core application and stop the execution using the ***Interrupt*** button .
- Switch to **C/C++ Perspective** using the button in right top corner .
- Select menu ***File/Import/General/Existing Projects into Workspace*** and import Hello World example application for Cortex-M4 auxiliary core to your workspace:

  <mqx_install_dir>/mqx/examples/hello/ds5/hello_twrvf65gs10_m4/.project

- Click the compile button and build application in the selected target. By default, the project is compiled in the ***Int Ram Debug*** target.
- Switch back to **Debug Perspective** using the button in right top corner .
- Highlight the ***hello_twrvf65gs10_m4_Int_Ram_Debug*** target.

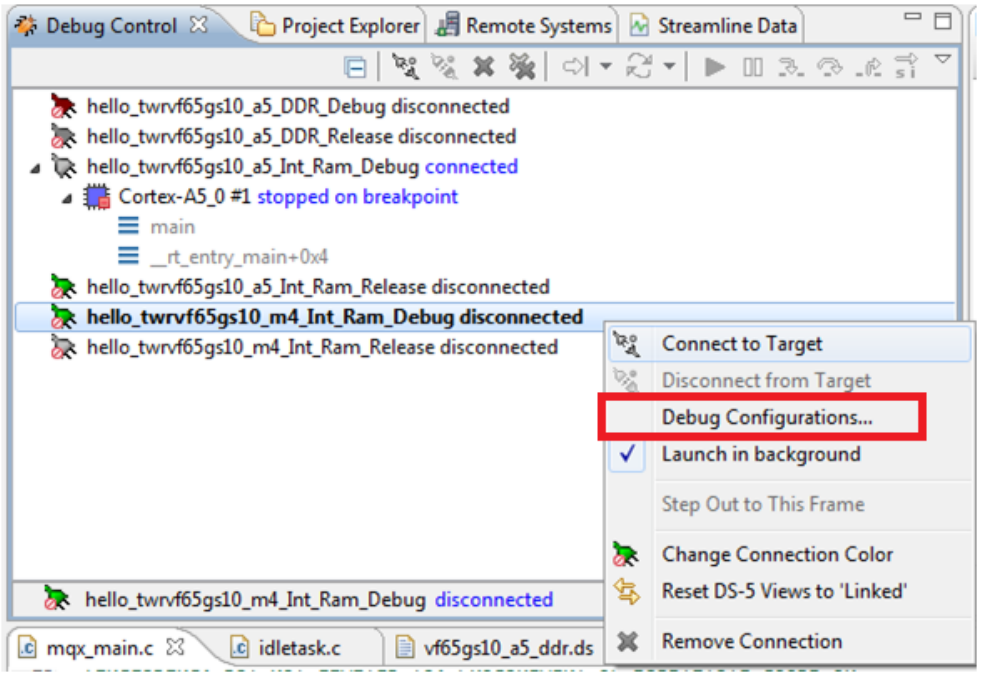

Figure 9: Debug configurations

- Select **Debug Configurations** and verify that the **Enable secondary clock on connection** option in the **DTSL Options** menu is still checked.

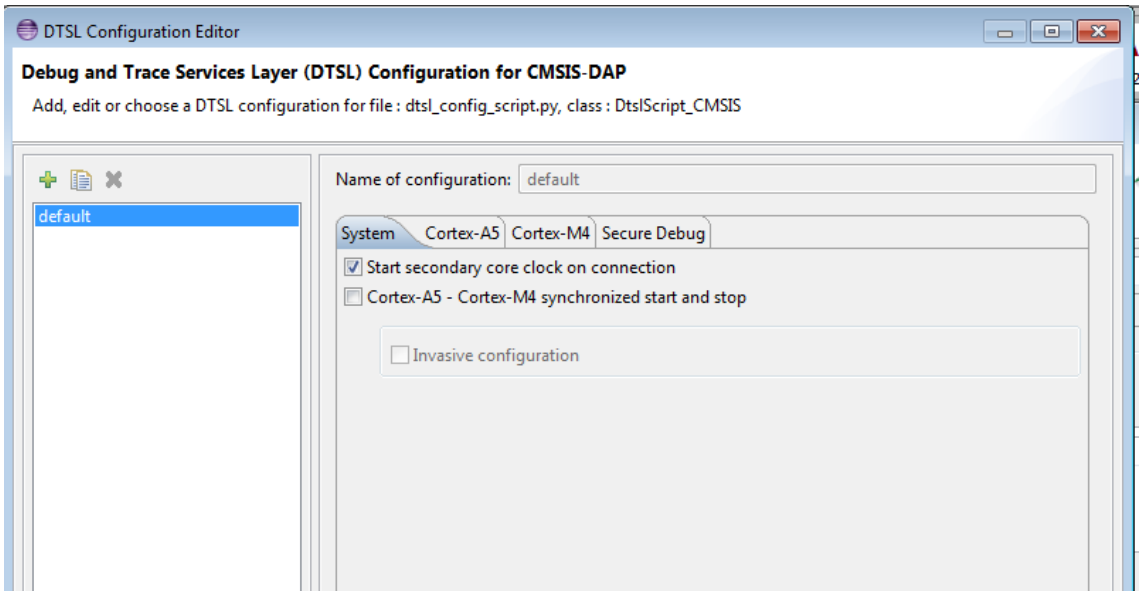

Figure 10: DTSL configuration

- Confirm the selection and then click the **Debug** button in the lower right corner.
- Project will be loaded to device and execution. CorexM4 core execution will stop in the main() function
- Press the run button and the program running on the auxiliary Cortex-M4 core will print “Hello World” on the serial console.

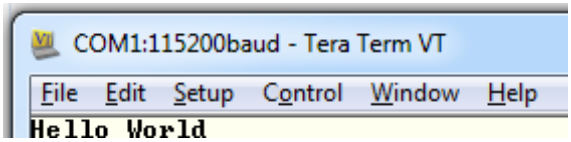

Figure 11: Hello World console

Note: It is always necessary to execute MQX RTOS application on primary core first. The MQX RTOS primary core application startup sequence contains settings required by Cortex-M4 core (clock setup etc).

3.3 Multi-core debugging

This chapter describes the basics of multi-core debugging with MQX RTOS. Description is provided for Vybrid microcontroller BSPs - twrvf65gs10_a5/twrvf65gs10_m4 and Multicore Communication (MCC) “pingpong” example application. The ARM Cortex-A5 is primary core in this setup while the Cortex-M4 is set to auxiliary core.

- Select menu ***File/Import/General/Existing Projects into Workspace*** and import Cortex-A5 and Cortex-M4 MCC library projects as follows:
  - *<mqx_install_dir>*/mcc/build/ds5/mcc_twrvf65gs10_a5/.project
  - *<mqx_install_dir>*/mcc/build/ds5/mcc_twrvf65gs10_m4/.project
- Select the *mcc_twrvf65gs10_a5* project in the Project Explorer View and then click the compile button  to build the ***Debug target***.
- Select the *mcc_twrvf65gs10_m4* project in the Project Explorer View and then click the compile button  to build the ***Debug target***.
- Select menu ***File/Import/General/Existing Projects into Workspace*** and import MCC Pingpong example applications for both Cortex-A5 and Cortex-M4 core as follows:

<mqx_install_dir>/mcc/examples/pingpong/ds5/pingpong_example_twrvf65gs10_a5/.project

<mqx_install_dir>/mcc/examples/pingpong/ds5/pingpong_example_twrvf65gs10_m4/.project

- Select the *pingpong_example_twrvf65gs10_a5* project in the Project Explorer View and then click the compile button  to build the ***DDR Debug target***.
- Select the *pingpong_example_twrvf65gs10_m4* project in the Project Explorer View and then click the compile button  to build the ***Int Ram Debug target***.
- Click the arrow next to the Debug button and select ***Debug Configurations***:

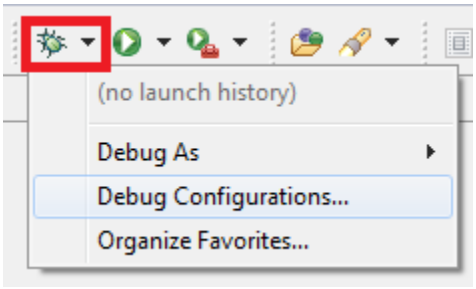

Figure 12: Arrow button

- A dialog box will come up. Select the ***pingpong_example_twrvf65gs10_a5_ddr_debug*** configuration and the ***Cortex-A5*** *with* ***CMSIS-DAP*** debug target connection. Before clicking the **Debug** button in the lower right corner, check the **Enable secondary clock on connection** in the **DTSL Options** menu.

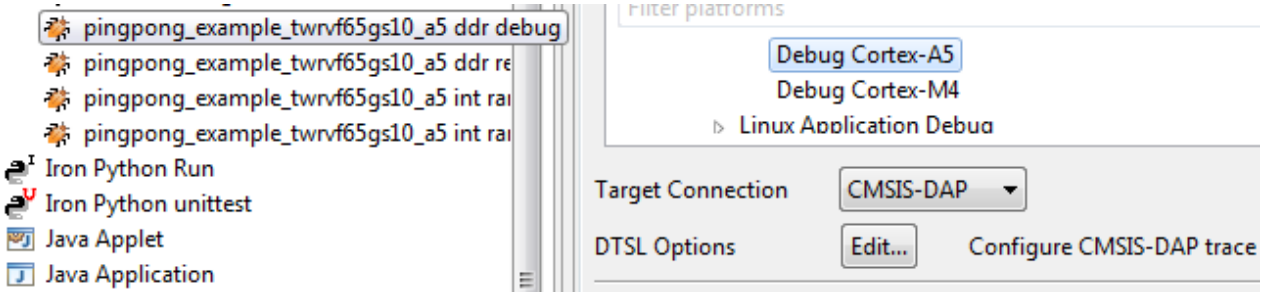

Figure 13: DTSL options menu

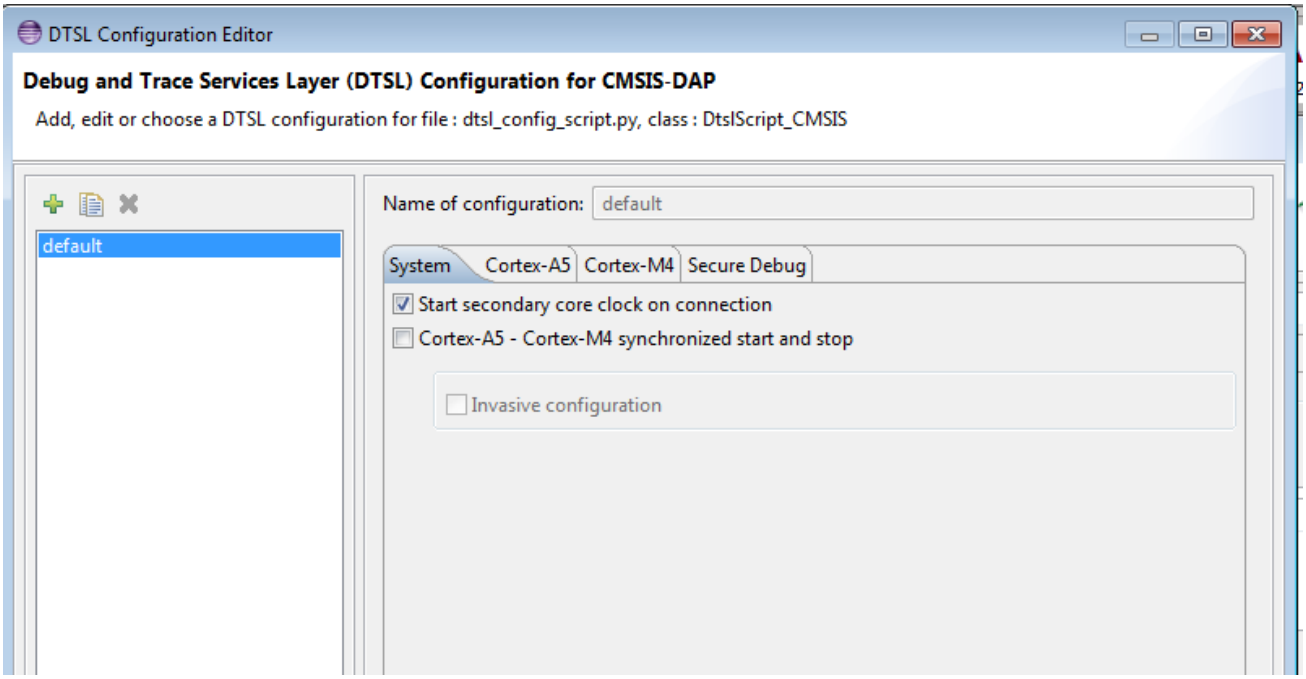

Figure 14: DTSL configuration

- The Development Studio 5 (DS-5) will switch to Debug Perspective automatically. Then, the project will be loaded to the device and execution will stop in the main() function.

- Press the **Run** button to continue in the *pingpong_example_twrvf65gs10_a5* program execution.

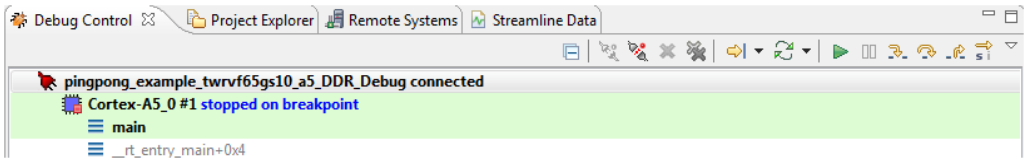

Figure 15: Debug control

- The program will print “Main task started, MCC version is xxx.xxx” on the serial console terminal.

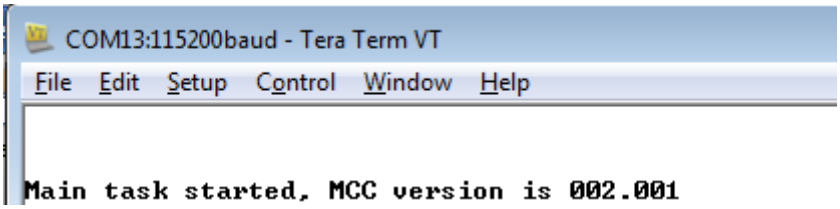

Figure 16: Main task started

- Once the Cortex-A5 code application is running, start the execution of the auxiliary core (Cortex-M4).
- Select menu ***Run/Debug Configuration***s.
- When the Debug Configuration dialogue occurs, select the ***pingpong_example_twrvf65gs10_m4_Int_Ram_Debug*** configuration and the ***Vybrid Cortex-M4 CMSIS-DAP*** debug connection.
- Then click the ***Debug*** button in the lower right corner.

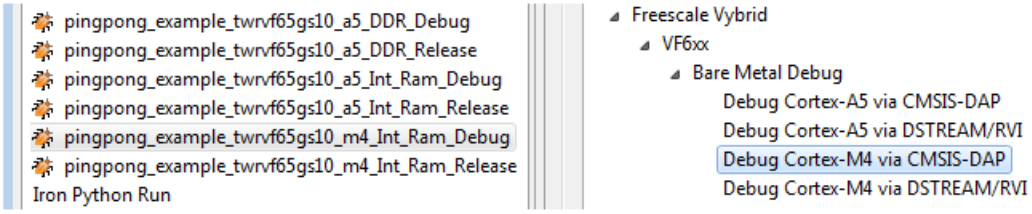

Figure 17: Debug button

- The Cortex-M4 project will be loaded to the device and execution will stop in the main () function. Press the ***Run*** button to continue in the *pingpong_example_twrvf65gs10_m4* program execution.

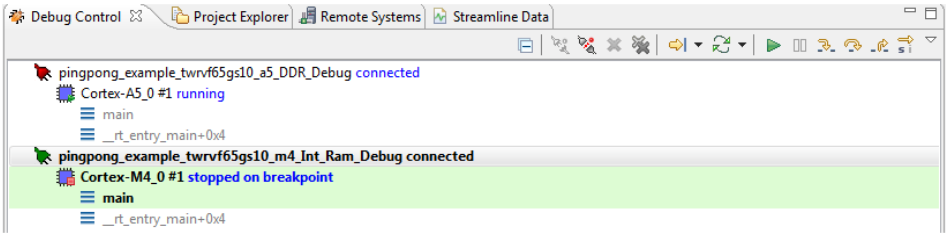

Figure 18: Run button

- The responder will be started and message “pingpong” between the cores will be initialized. See the console log.

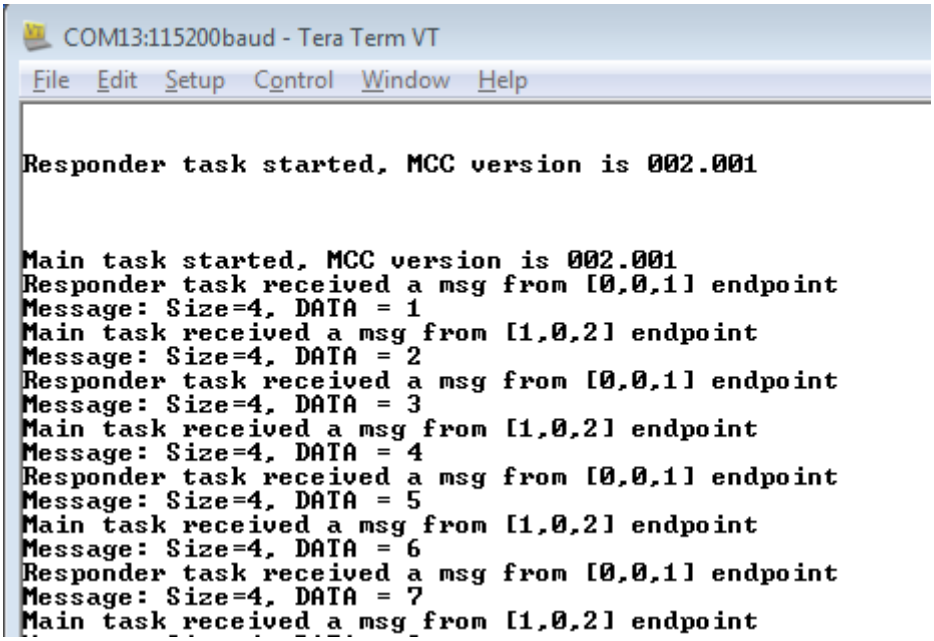

Figure 19: Pingpong message

3.4 Debugging the Application loaded by MQX RTOS Boot Loader

This chapter describes debugging the application which was loaded to the processor memory by MQX RTOS Boot Loader. The similar approach can be used for debugging an application loaded by a different boot loader e.g., U-Boot. This chapter also briefly describes steps required for preparing bootable SD Card image and application images in DS-5 tool set. For details about the Vybrid Boot Loader usage, see Readme.txt located in the MQX RTOS Boot Loader application folder (*<mqx_install_dir>/mqx/examples/bootloader_vybrid/Readme.txt)*

Building Boot Loader and creating bootable SD card

- First, import the MQX RTOS Boot Loader project to your workspace by using the ***File\Import\General\Existing Projects*** **into Workspace** menu.
- Select the bootloader_vybrid from your MQX RTOS installation directory:

<mqx_install_dir>/mqx/examples/bootloader_vybrid/ds5/bootloader_vybrid_twrvf65gs10_a5

- Select Int Ram Debug target and click the compile button  .
- Use DS5 "*C:\Program Files\DS-5\bin\fromelf*" utility to create the binary image:

```
fromelf.exe --bin --output=bootloader_vybrid_twrvf65gs10_a5.bin
bootloader_vybrid_twrvf65gs10_a5.axf
```

- Follow *<mqx_install_dir>\mqx\examples\bootloader_vybrid\Readme.txt* description and use the prepare binary image to prepare the bootable SD Card.

Building and Debugging the Application images

- Build the applications you want to run on A5 and M4 cores and convert them to binary format (.bin) by using `fromelf` DS-5 utility.
- Store the binary images on the root directory on bootable SD card.
- Copy setup.ini to the SD Card and modify according to Readme.txt description.
- Remove the SD Card from the PC and plug it into Micro SD Card slot on your Vybrid board.
- Power up the Vybrid board. MQX RTOS Boot Loader will print out the following message on the default console (RS232 TWR-SER) and start execution of M4 and A5 applications.

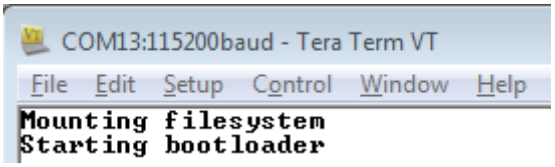

Figure 20: Mounting filesystem message

- To debug the running application, click the arrow next to the Debug button and select ***Debug Configurations***.

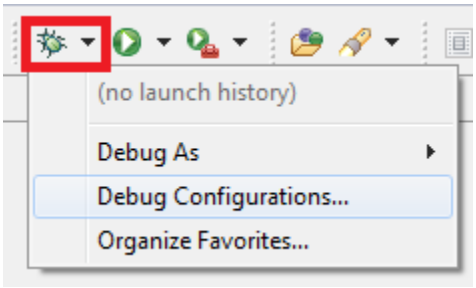

Figure 21: Debug configurations

- Then, select the application and target you want to debug and select ***Connect only***.
- Finally, click the **Debug** button in the lower right corner.

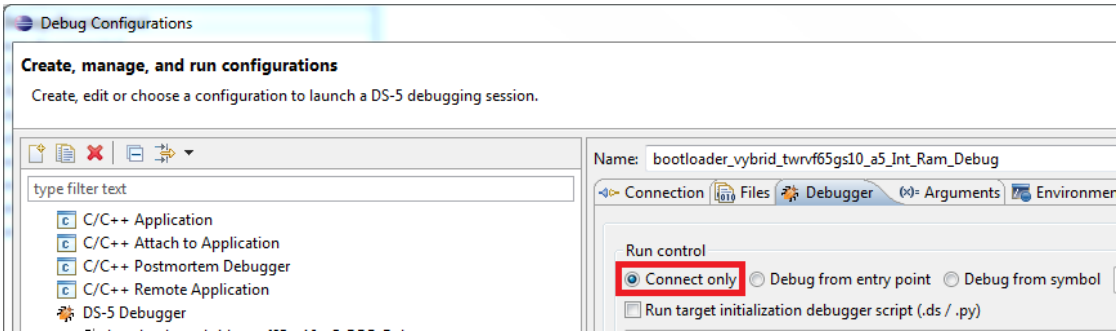

Figure 22: Create, manage, and run configurations

- The debugger will connect to the selected application. You can stop the selected core and debug the booted image.

4 MQX RTOS Task Aware Debugging in Development Studio 5 (DS-5) IDE

MQX RTOS Task Aware Debugging plug-in (TAD) is an optional extension to a debugger tool which helps to visualize internal MQX RTOS data structures, task-specific information, I/O device drivers, and other MQX RTOS context data.

The TAD plug-in is distributed separately from MQX RTOS release and directly from ARM® Ltd. For detailed documentation, contact your ARM distributor.

Example of available DS-5 TAD menus:

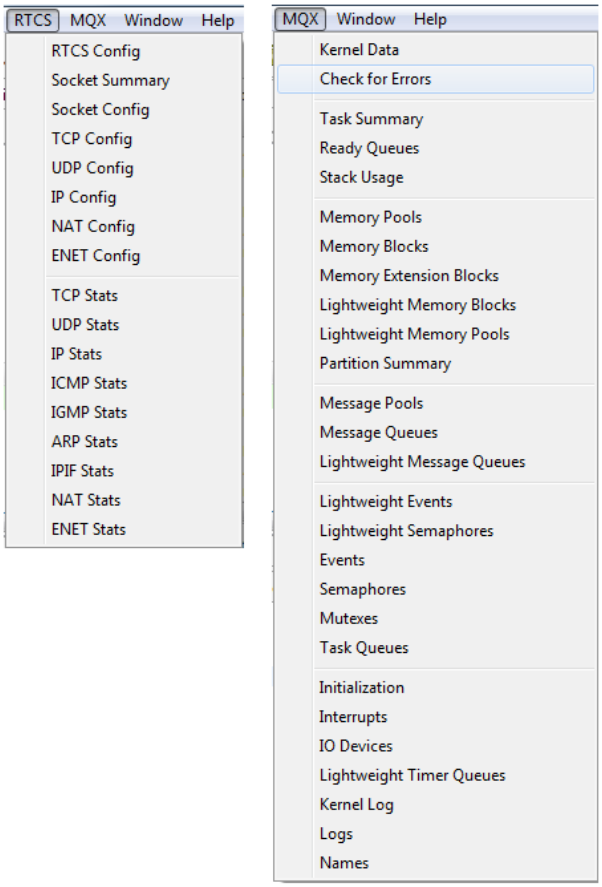

Figure 23: Example menus