Vybrid DS-5 Getting Started Guide

Rev 1.0

Revision History

Revision	Date	Changes
1.0	5/20/2013	Initial Release

1 Introduction

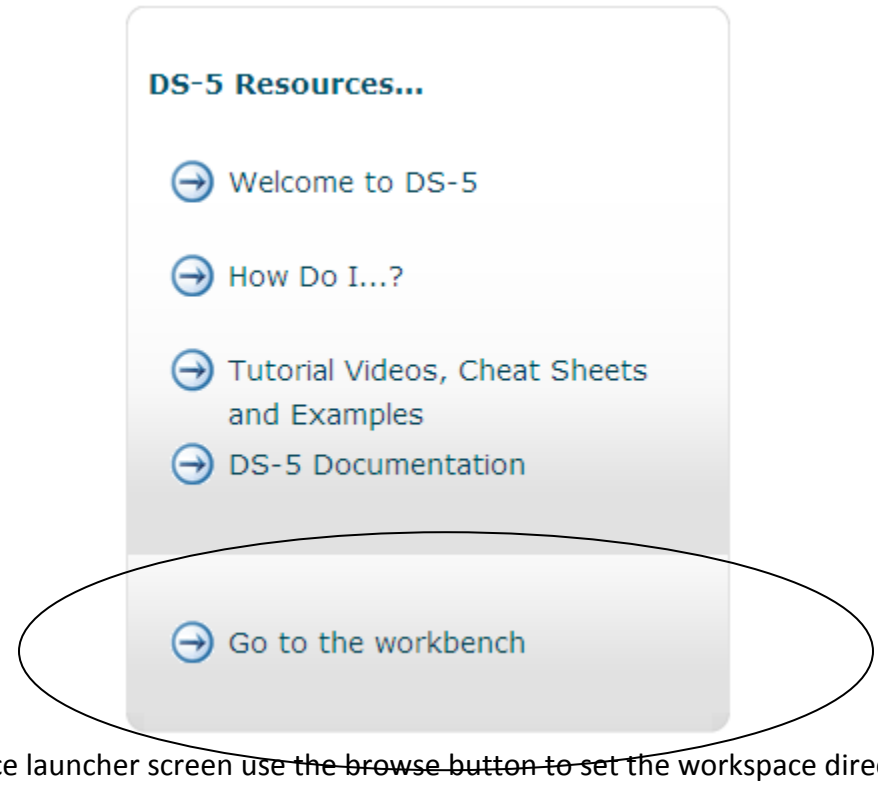
This document will detail the steps to getting started with DS-5 Eclipse debug environment using a Freescale Vybrid device. The examples and details will use a TWR-VF65GS10 Tower board. It will show the workspace environment, sample code project format, and debug options using A5 and M4 cores. A getting started with DS-5 video can be seen here:
http://www.youtube.com/watch?v=djExDHsBa5w

2 Download DS-5 from www.arm.com/ds5

The first version of DS-5 to have fully integrated support for Vybrid is v5.12. Earlier versions (5.11 specifically) did support Vybrid but there were some plug-ins required for debug connections and peripheral register access. It is recommended to have v5.12 or later installed.

3 Open DS-5 and configure the workspace

1. Open the ARM DS-5 IDE (**Start->Programs->ARM DS-5->Eclipse for DS-5**). If you see the welcome screen, click Go to the workbench

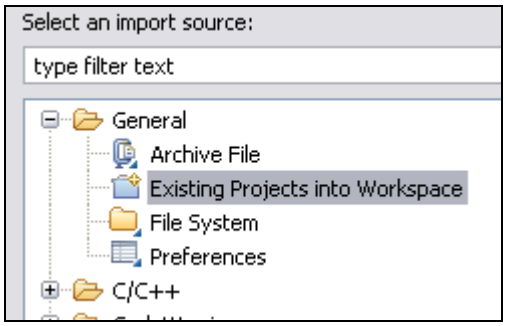


2. At the workspace launcher screen use the browse button to set the workspace directory to the Vybrid sample code folder. Typically this folder will be named vybrid-sc. Then click OK.
3. Make sure the C/C++ perspective button is selected.  These are located in the upper right hand corner. If the debug option is selected instead, just click on the C/C++ option to switch the view.

- Note: Available windows will change depending on the view that is selected

4 Import the Projects into the Workspace

1. If there are no projects visible in the Project Explorer window, click on **File->Import** in the menu bar. In the dialog box that comes up, select "Existing Projects into Workspace" under the General folder. Then click Next.



2. On the next screen, select the "Select root directory:" option, and click on Browse

3. Navigate to the root of the project patch (for example, **C:\vybrid-sc)** and hit OK. Note: if you setup your workspace correctly you should already be at this directory when you hit Browse.
4. Select all of the projects you wish to import into the workspace. Make sure to at least select **dual_core_ds5** and **dual_core_ds5_m4** hit **Finish**.

   Note: The sample code directory main folders are /build and /src. The projects reside in /build and source files are in /src.

5 A5 & M4 Project Settings and Differences

The **dual_core_ds5** project runs on the A5 core and the **dual_core_ds5_m4** project runs on the M4. Each has different project settings for compiling and debugging. They also work together to form one final application image. This will be covered in detail below.

In this example, the A5 debug session will be launched first, although it is not always required. For the example projects, the A5 code sets up the system clock, UART, clock gating, and interrupts so launching and executing this first is recommended. The M4 initialization is less involved, simply to prevent unintended clock updates or setup changes during A5 execution.

Note: Some peripherals, e.g., timers, may not run when the A5 is halted. Keep this in mind when debugging the M4 application.

There are two instructions to allow M4 core to begin execution. They can be provided by the A5 core, startup script, U-Boot, etc:

```
/* Set starting point (0x3f040400) for M4 code – Must be odd since M4 is thumb */
SRC->GPR[2] = (unsigned int)0x3f040401;

/* Enable M4 core */
CCM->CCOWR = 0x15a5a;
```

In the dual_core_ds5 project, these instructions are provided by the start_m4() function.

The information below shows the main differences between the A5 and M4 projects within DS-5.

1. Compiler & Assembler Settings

   - Right click project, select Properties->C/C++ Build->Settings. Notice the **Code Generation** box under ARM C Compiler and ARM Assembler

   For A5 Project:

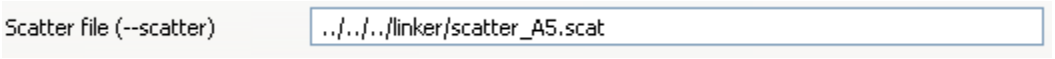
   For M4 Project:

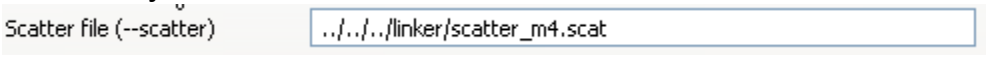
2. Scatter (Linker) Files

   - Right click project, select Properties->C/C++ Build->Settings->ARM Linker->Image Layout

   For A5 Project:

   For M4 Project:

3. Preprocessor defines

   - Right click project, select Properties->C/C++ Build->Settings->ARM C Compiler->Preprocessor. **ARMCC_A5** will be defined for the A5 project and **ARMCC_M4** will be defined for the M4 project. These preprocessor definitions will be used to specify the DS-5 tool and for specific application setup like interrupt routing.

4. Entry point

- Right click project, ->C/C++ Build->Settings->ARM C Compiler->Preprocessor

For A5 Project:

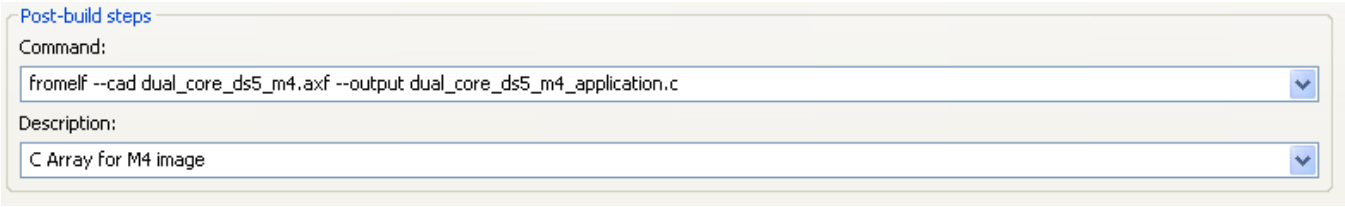
- See file: vectors_armcc.s

For M4 Project:

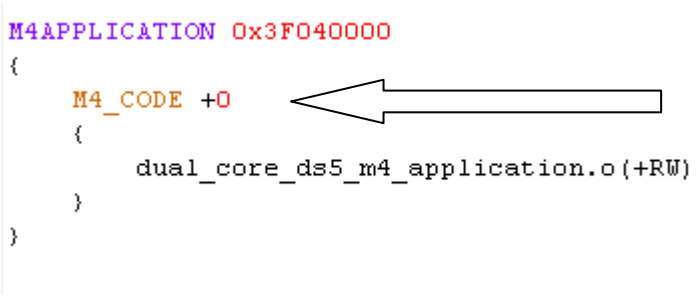
- See file: crt0_m4_ds5.s

6 A5 & M4 Project Linking into Single Application

When the M4 project is built, a C-array is generated from the elf file output file (.axf) and then linked to the A5 project. When the A5 project is built, it will include the C-array from the M4 build into its executable. The C-array is created after building the M4 project using the fromelf image converter feature, which is part of the DS-5 toolchain. The command can be viewed by right clicking the project and selecting Properties->C/C++ Build->Settings and clicking on the Build Steps tab:



The M4 C-Array is put into a specific location within the A5 executable using the scatter (linker) file.



This symbol (M4_CODE) is imported into the functional code so the A5 knows the starting execution address of the M4. The method to bring the symbol into the code can be seen in the dual_core_ds5.c file:

```
/* Scatter file symbol to tell us the M4 start address */
extern unsigned int Image$$M4_CODE$$Base[];
```

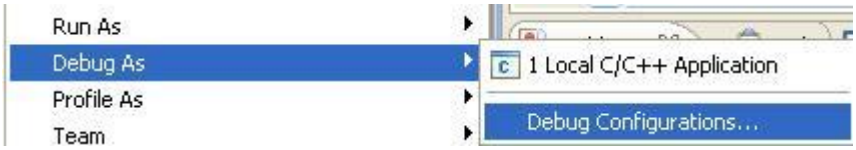
The scatter file is essentially a linker file with scatter-loading capabilities. The ARM info center has detailed information on scatter files:

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.kui0101a/armlink_babddhbf.htm

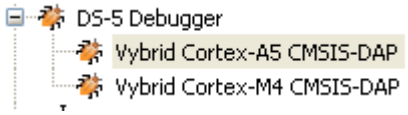
The A5 project links the C-array from the M4 project path by importing it into its source directory. This is setup already for this example. The M4 project must be built first since its output is linked into the A5 project. This was setup with the following steps: Right click on the dual_core_ds5 source folder and select New->File->Advanced, click 'Link to file in the file system' and point to the C-array in the Debug output folder of the M4 project.

7 Debug Settings and Code Execution

1. Right-click on the A5 project and select "**Clean Project**." Do the same for the M4 project.
2. Select the dual_core_m4 project by single clicking on it. Then, go to the menu "Project" and click on "**Build Project**" or hit the Build icon  . The active project in the main project window will be the one that is built. Next select the dual_core_ds5 project and build it.
   →Note: the M4 project must be built before the A5 project.
3. Right-Click on the dual_core_a5 project in the project panel and select **Debug As -> Debug Configurations**

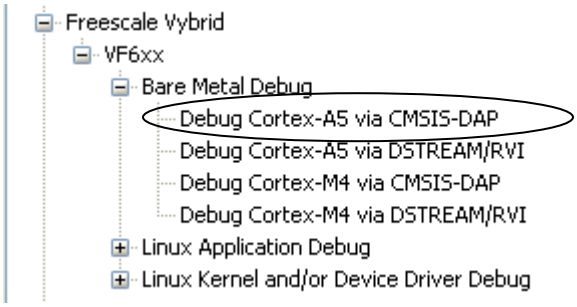


Here you will see options for both cores:

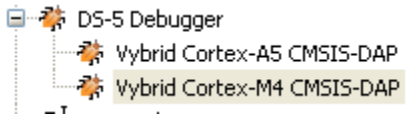


Note: If the debug connections are not available, click on DS-5 Debugger and click the New Launch Configuration icon on the upper left hand side of the Debug Configurations window.

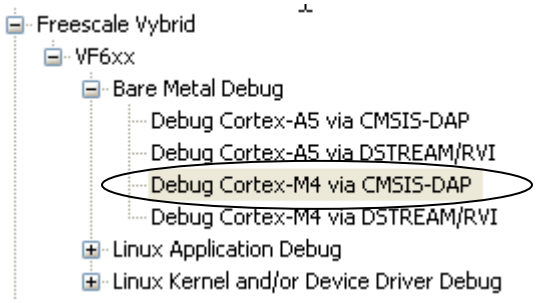
Next, click on **Vybrid Cortex-A5 CMSIS-DAP** and then on the right side under the Connection tab, navigate to the A5 CMSIS-DAP connection



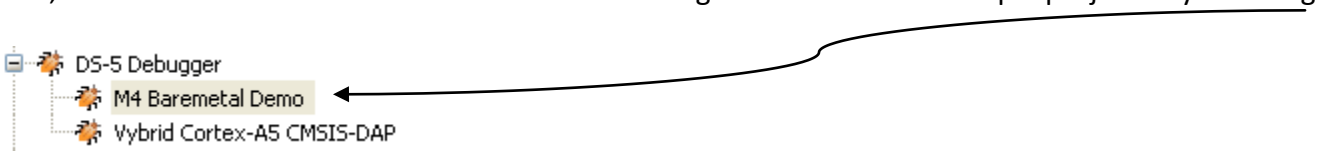
This will also need to be done for the M4 as well.



And under the Connection tab:



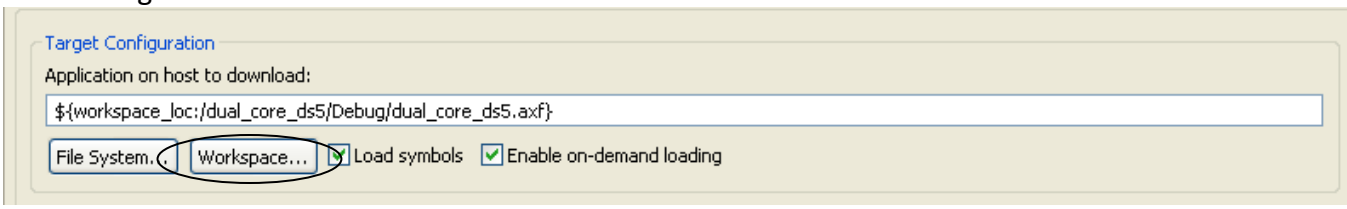
Note, that the name of the connection should be changed to match the example project to your liking.



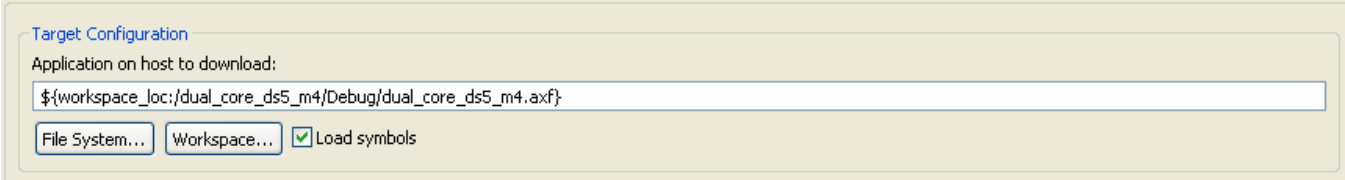
Updating the name of the connection is recommended if there are multiple projects in the workspace since each setup is tied to a particular output file, and potentially any startup or initialization scripts.

Under the **Files** tab for each connection, navigate to the correct elf (.axf) output file for that connection. This file will reside in the /Debug output folder. Click on “apply” to save your changes. The .axf will only be available after the project is built successfully.

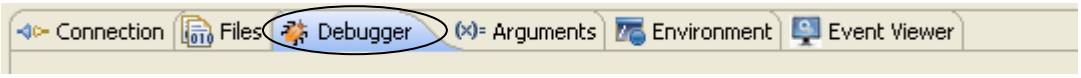
For A5 Target:



Also for the M4 target:

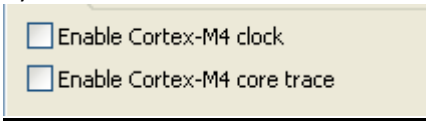


The Debugger tab shows the entry point, and any startup scripts for that particular connection.



Lastly, back on the connection tab, there is an option to automatically enable the M4 clock. Click the Edit… button under DTSL Options

We will leave this unchecked because we enable the M4 clock in software (Example 1). It will be useful to enable this when doing a dual core debug session using connections to both cores (Example 2).



Starting Debug Session

Two methods are detailed below. One establishes a debugger connection with the A5 and then the function start_m4() will begin the M4 code execution. The second example establishes debug connections for both cores.

Note: For quick access, there will be a history of debug connections available after they are launched by clicking the down arrow next to the debug icon. 

Example 1: Single debug connection running both cores

1. Power down the Tower board and remove the SD card. Plug power back into board. We do not want the default program from the SD card for an A5-only debug session. The default program running from the SD card will run code on the M4 core and we don't want to do this yet.

2. Right click on the A5 project and navigate to Debug As->Debug Configurations…, select the proper connection, and hit the **Debug** button to establish a connection with the Tower board.

3. Wait for the debugger to connect to the board and download the application. The Debug perspective will open and the code is ready to run from the entry point (or main() depending on debug settings).

4. If you don't have a **terminal** open already, then open HyperTerm or another terminal application configured for **115200 8-N-1**. Make sure your serial cable is connected to the tower and your PC.

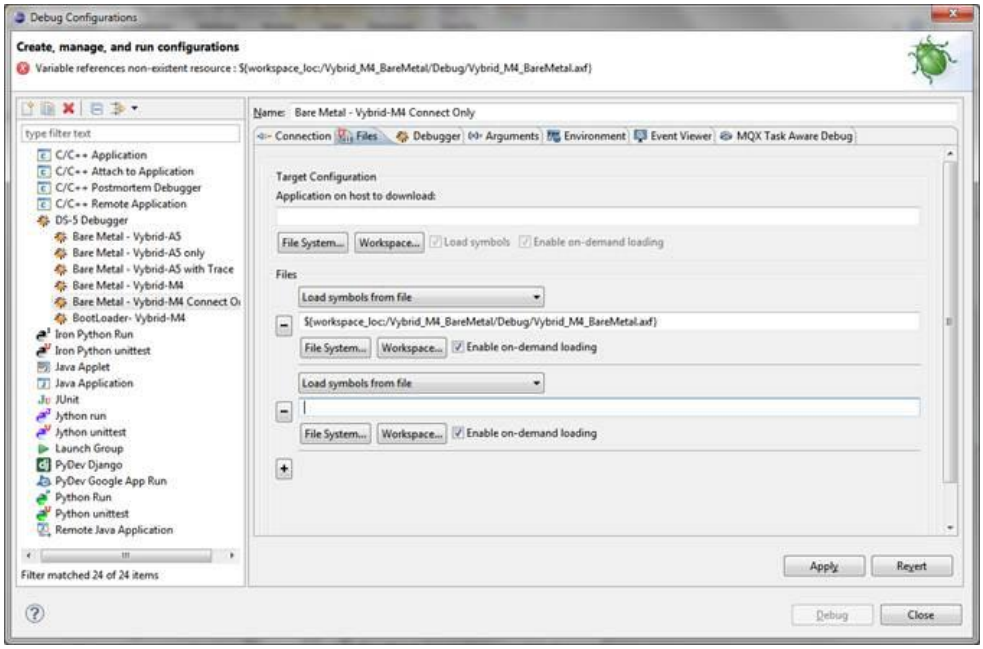
    - TIP: DS-5 provides a terminal program.
        - Go to Window->Show View->Other->Terminal

5. At this point, hitting the Run  button will start the A5. The A5, in turn, will start the M4 using the **start_m4()** function. Notice the LEDs toggling at different rates and the terminal printouts from both cores. Remember the M4 C-array output file is included in the A5 download. In addition, the start address from the scatter file symbol was imported into the code.

6. Click Pause  , then disconnect button  . Now the A5 debug session is stopped. Why are two LEDs still blinking? It's because the M4 core is still running! Press the reset button on the board before continuing to the next step.

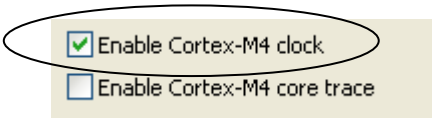
TIP: The M4 can connect without downloading.

- In M4 debug target settings, go to Files tab. Leave "Application on host to download" section blank, but then add your application to the "Load symbols from file" section. Finally, in Debugger tab, set Run Control to "Connect Only". This allows debugging an application that is already running on the M4 core.



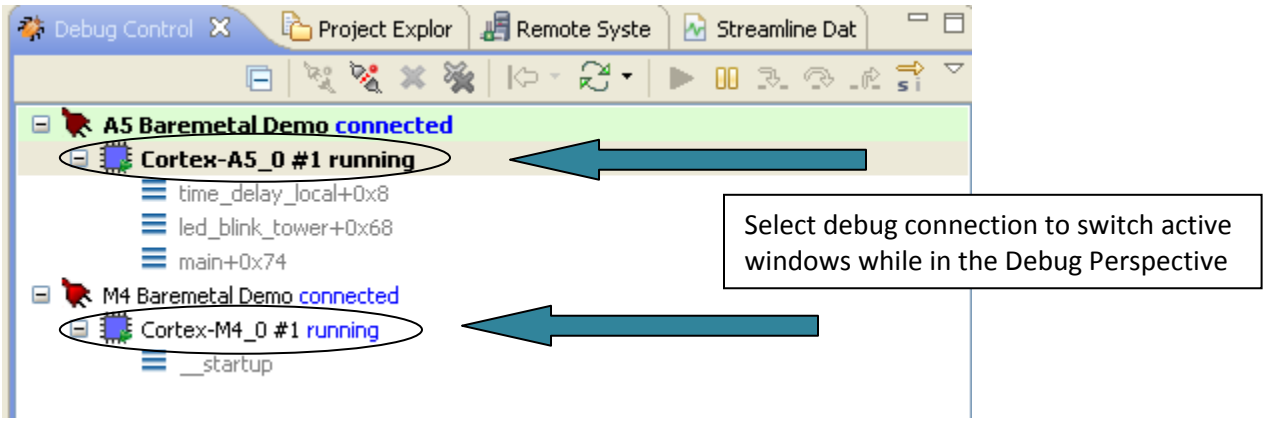
Example 2: Dual debug connection running both cores

7. Go back to the C/C++ perspective  and comment out the line "start_m4();" in file **dual_core_ds5.c** (line 24) and click Save. Rebuild the target by pressing the hammer icon  .
8. In the Debug Configurations… window Enable M4 clocking in the A5 debug connection.

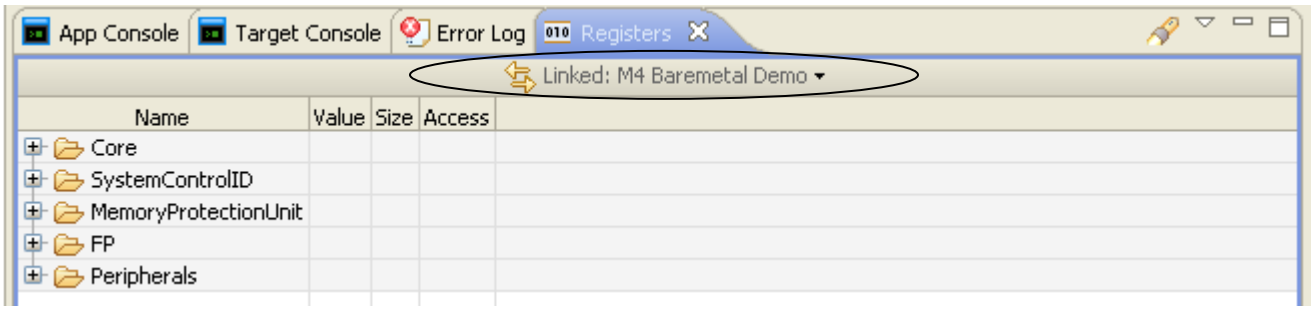


9. Click the down arrow next to the Debug Icon  to re-connect the A5 debug session. Click Run after it connects. Notice the LEDs and the printouts, this time without the M4 printout or additional LED toggling. Now launch the M4 connection (while A5 is running) and click Run. Both cores are now running with separate debug connections.

10. Based on the active selection in the Debug Control tab, the active program counter, registers, and other debugger windows will be updated. Certain views can remain open even if debugger views are switched.

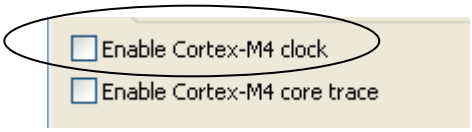


11. Select a connection then click the pause button. Do this for both cores. Notice that the debug windows update based on the debug connection selected above. If necessary, we can lock certain windows to keep them active for both connections. Simply click on the 'Linked' area as show below to switch from Linked view.



12. To create a secondary Registers view for the other core, click the down arrow in the upper right corner and select **New Registers View**. This new view can be linked to the other core in a similar fashion. Also, these windows can be moved and resized so they can be visible at the same time. Just click the Registers tab and drag.

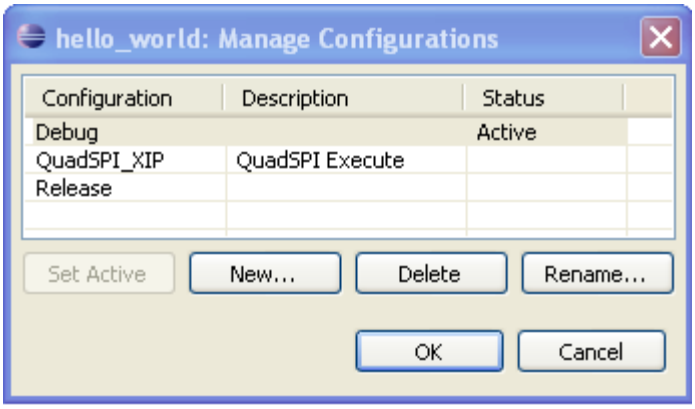
13. If future debug sessions are to be done with A5 only debug connection, then disable (uncheck) Enable Cortex-M4 clock within the A5 debug connection.

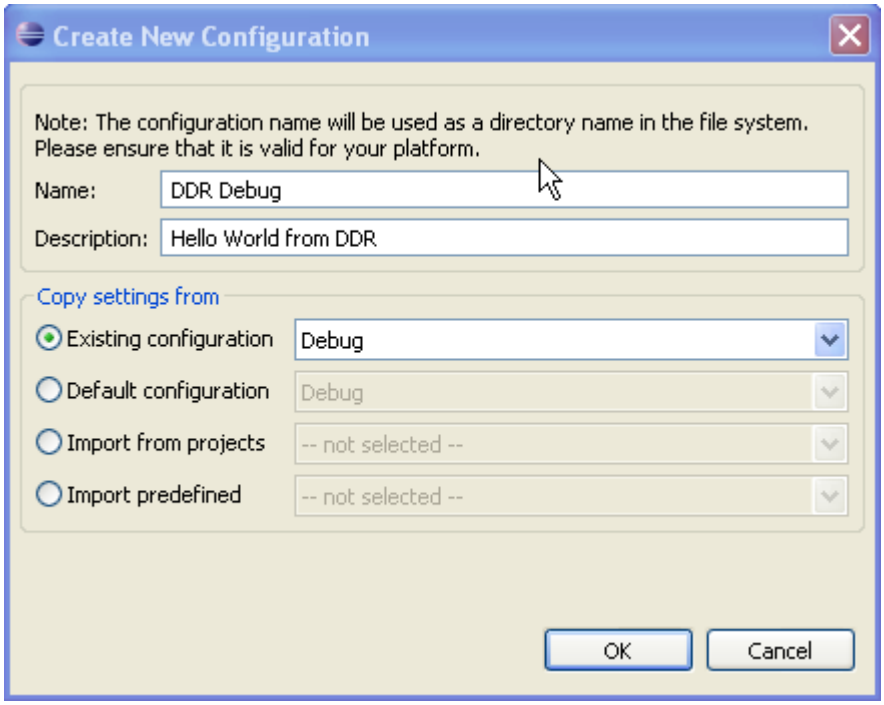


14. Click on the **disconnect** button  for each connection to end the current debug session.

7.1 Creating a new Debug Configuration (Example: DDR)

Right click project (hello_world example shown), select Build Configurations->Manage. Here, you can add a new entry (New…) and give it a new name, like DDR Debug.

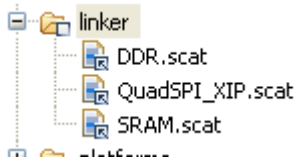




Use default (Copy settings from Debug Configuration) click OK. Click OK again to exit Manage Configurations.

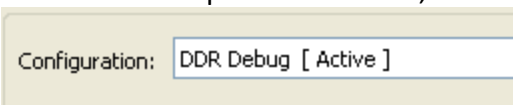
Right click hello_world project again. Select Build Configurations->Set Active. Select the DDR Debug configuration just created.

Now we need to add a new scatter (linker) file and new initialization script. Drag and drop the DDR.scat file into the /linker folder. This does not yet make the target use this scatter file, this is done in the next step (Project Properties).



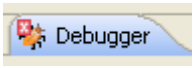
Right click on project, select Properties->Settings->ARM Linker->Image Layout. Change
../../../linker/SRAM.scat
To
../../../linker/DDR.scat

Notice at the top of the window, DDR is the active configuration. Click OK.



Compile the DDR target by pressing the hammer icon. We need to make sure it compiles without errors prior to creating the new connection so we can point the new connection to the output (.axf) file.

Update debugger script. Go to Debug Configurations  (right click down arrow). Right click DS-5 Debugger and create a new connection.

Navigate to Freescale Vybrid VF6xx Bare Metal Debug and choose connection target (likely to be Debug Cortex-A5 via CMSIS-DAP for Tower board). Give the connection a new name in the Name box at the top of the Debug Configurations window. Click the Debugger tab  and click the box for Run target initialization debugger script (.ds / .py). Click Workspace button, select the hello_world (or the current target) folder and select file vf65gs10_a5_ddr.ds. If this file is not available, then use the File System button to find it.

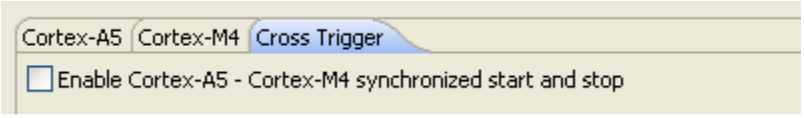
Note: This file is re-used from MQX for Vybrid

While still in the Debugger Configurations, click the Files tab  and under Target Configuration click the Workspace… button. Select the hello_world.axf under DDR Debug folder (or the name that was selected for the debug configuration). Click OK.

Now you have new target and debug configuration to build and debug hello_world from DDR memory. These steps can be applied to any target to create a DDR setup.

7.2 Cross Triggering (CTI), available in DS-5 v5.14 and Later

Cross triggering allows both cores to stop when a breakpoint is set. This can easily be enabled by going to Debug Configurations  (right click down arrow). Click the Edit button next to DTSL options and you will notice new tab named Cross Trigger. Simply check the box to enable this feature.



Also note, under the Cortex-A5 and Cortex-M4 tabs there are options to enable trace.

7.3 General Debugging Tips and Recommendations

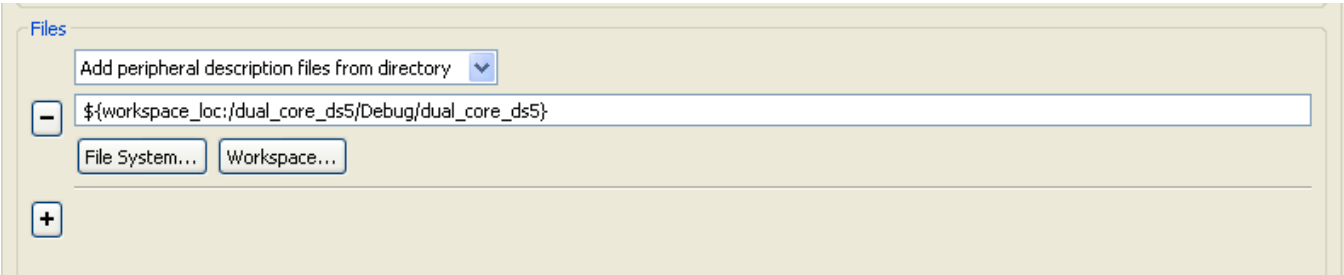
- Be aware of the default program running from the SD card. This may affect the M4 debug connection. Removing the SD card will prevent the default program from running and changing some default settings. If a program was running previously on the M4, it will not re-execute without a reset or a new debugger connection.

- Workspace settings update: By default, files are not saved when building the project. To enable this by default within the workspace, go to Window->Preferences->General->Workspace and check the box for 'Save automatically before build'.

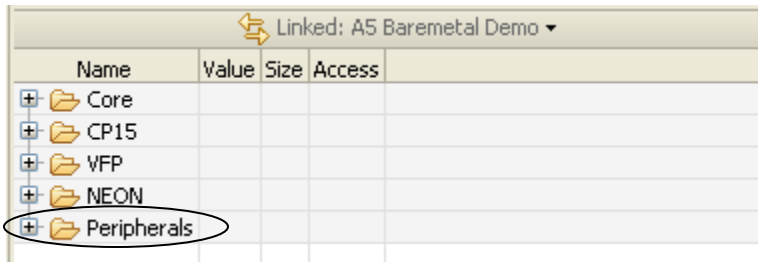
- Hitting pause before hitting disconnect on an active debug connection is recommended.

8 DS-5 Vybrid Peripheral Registers

The core registers are visible in the Register window for both A5 and M4 connections. To get additional peripheral registers into the Register view, the SVD file needs to be imported manually. This process is required for DS-5 v5.12 and prior. The required file is **MVF50GS10MK50.svd.xml**. This file is brought into the workspace using the Debug Configurations. Go to Run->Debug Configurations->DS-5 Debugger->Vybrid Cortex-A5 CMSIS-DAP, click on the Files tab:



Point to the folder where the SVD file resides and click OK. Now you should see the Peripheral register in the Registers window:



Now you have full debug control over both cores, register views of the entire memory map, and a base framework for future development. Happy debugging!

==========

Misc Info:
C:\Program Files\DS-5\sw\debugger\configdb\Boards\Freescale_Vybrid\VF6xx contains the debugger scripts

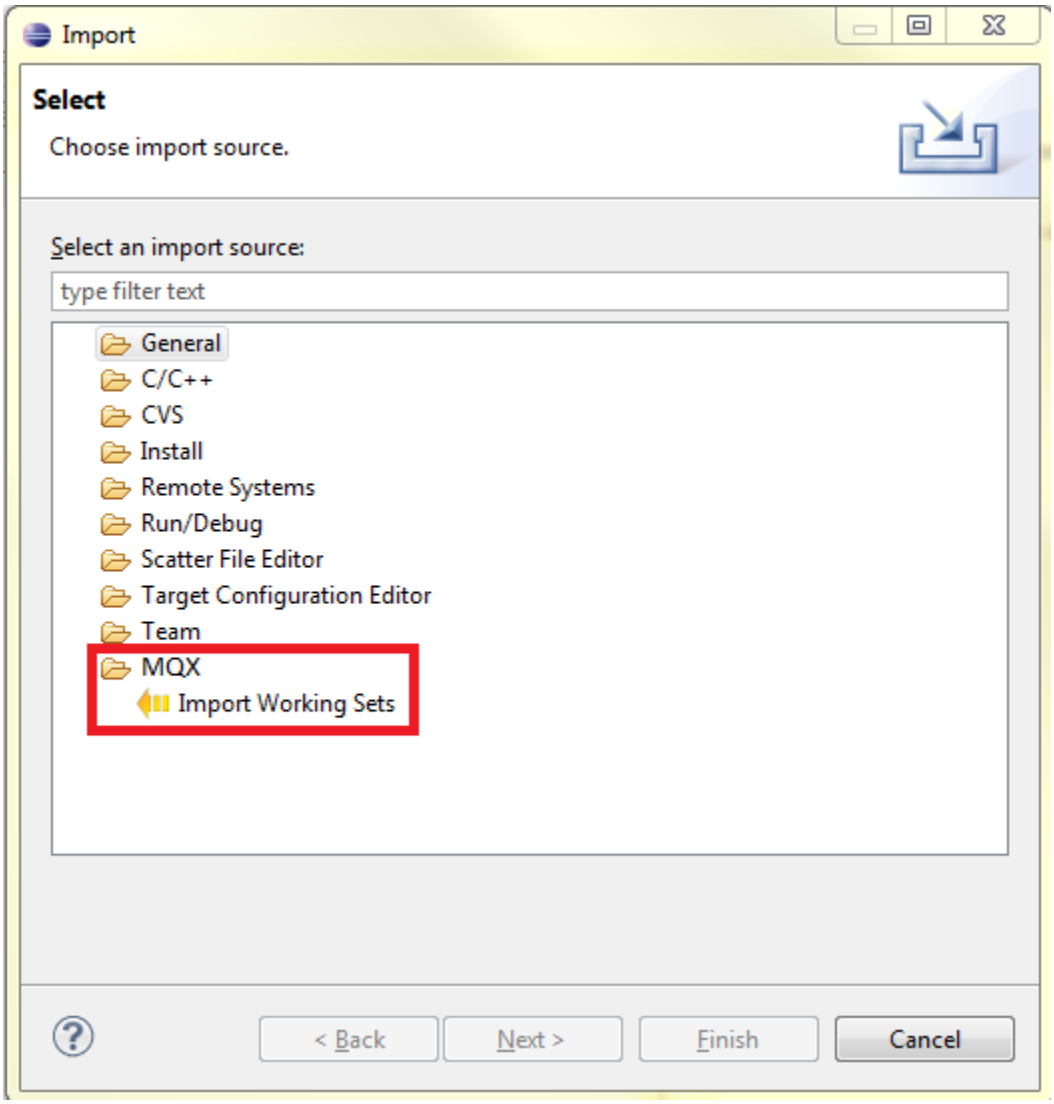
The .py contains the low level connectivity commands

9 Create Working Set to Build MQX Libraries

Instead of importing and building each MQX project separately, a process exists to create a working set for all MQX library projects and build them with one button click.

First, install the MQX eclipse plugin using ***Help\Install New Software\Add\Archive***… menu. Select the following archive : `<mqx_install_dir>/tools/ds5/ds5_update_site.zip`

To rebuild the MQX libraries import the `<mqx_install_dir>/config/<board>.wsd` working set description file using ***File\Import\MQX\Import Working Sets*** menu. The MQX library projects will be imported to DS-5 working space together with build configurations settings.

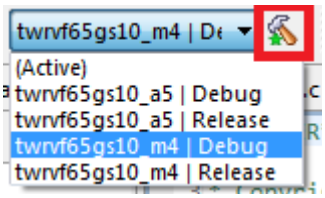


Note: The MQX folder may appear as 'Other'

Following projects will be imported to your workspace

```
<mqx_install_dir>/mqx/build/ds5/bsp_<board>/.project
<mqx_install_dir>/mqx/build/ds5/psp_<board>/.project
<mqx_install_dir>/mfs/build/ds5/mfs_<board>/.project
<mqx_install_dir>/rtcs/build/ds5/rtcs_<board>/.project
<mqx_install_dir>/usb/host/build/ds5/usbh_<board>/.project
<mqx_install_dir>/usb/device/build/ds5/usbd_<board>/.project
<mqx_install_dir>/shell/build/ds5/shell_<board>/.project
```

Select the target and platform and build the libraries - hit the compile all button (hammer with green start) . All projects will be built in selected configuration. The "Debug" configuration is dedicated for easy application debugging while the "Release" target has compiler and linker optimization set to maximum



10 TimeStorm Plugin Information

The TimeStorm plugin allows the Timesys GCC toolchain integration into DS-5. This is required for Linux application development and debug, and easy integration into the Timesys Desktop Factory. This only works in the Linux environment. For Windows GCC toolchain options, see the Third Party Compiler Options section below. To install, download either tsplugins_x86.sh (32-bit) or tsplugins.sh (64-bit) installer. After downloading, change permissions:

$ chmod +x tsplugins.sh

And then install:

$ sudo ./tsplugins.sh

To remove Timestorm plugin, in Linux terminal, navigate to /usr/local/DS-5/sw/eclipse/eclipse/plugins

$ sudo rm –r com.timesys.*

11 Third Party Compiler Options

DS-5 is supplied with ARM Compiler for compiling bare-metal applications. ARM Compiler is license managed and not all editions of DS-5 include a license for it. If your edition of DS-5 does not include a license for ARM Compiler, or you are developing applications for a non-bare-metal case, such as Linux applications, then you might require the use of a third-party compiler.

GCC is one such toolchain that you can use as an alternative to ARM Compiler. The GCC toolchain allows you to compile bare-metal, Linux kernel and Linux applications. Linaro provide pre-built versions of GCC on their website. For example:

GCC for Linux kernel and applications (2013.02 release):

Installer for Windows host:
https://launchpad.net/linaro-toolchain-binaries/trunk/2013.02/+download/gcc-linaro-arm-linux-gnueabihf-4.7-2013.02-01-20130221_win32.exe

Compressed archive for Linux host:
https://launchpad.net/linaro-toolchain-binaries/trunk/2013.02/+download/gcc-linaro-arm-linux-gnueabihf-4.7-2013.02-01-20130221_linux.tar.bz2

GCC for bare-metal (2012.06 release):

Installer for Windows host:
https://launchpad.net/gcc-arm-embedded/4.6/4.6-2012-q2-update/+download/gcc-arm-none-eabi-4_6-2012q2-20120614.exe

Compressed archive for Linux host:
https://launchpad.net/gcc-arm-embedded/4.6/4.6-2012-q2-update/+download/gcc-arm-none-eabi-4_6-2012q2-20120614.tar.bz2

For ease of use and to be able to use the tools from Eclipse IDE, ensure that the installed / extracted binaries are present on the PATH environment variable, for example "C:\Program Files (x86)\Linaro\gcc-linaro-arm-linux-gnueabihf-4.7-2013.02-01\bin". Newer releases of GCC might be made available on this website from time-to-time. Older releases are also available.
The pre-built bare-metal examples supplied with DS-5 are built with ARM Compiler. The pre-built Linux application examples are built with Linaro's GCC 2012.05 release.