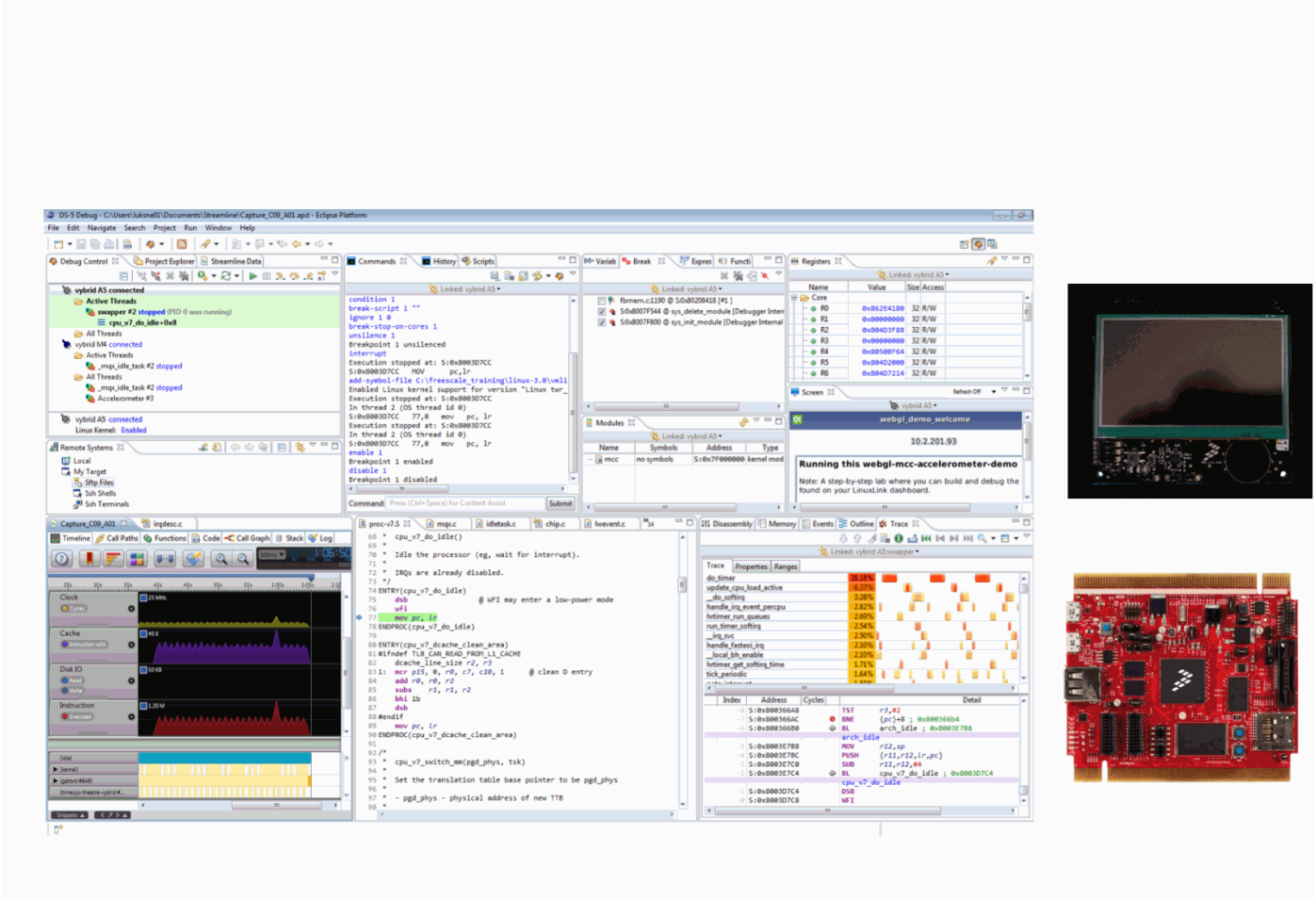
DS-5 Workshop: Linux Kernel and Application Debug, Trace and Profile on Vybrid



This workshop demonstrates the features of the DS-5 Debugger by simultaneously debugging and tracing Freescale MQX RTOS and the Linux kernel running on the Freescale Vybrid board.

It also demonstrates the ARM Streamline profiler using an application called threads. It will introduce some of the views of DS-5 Debugger and ARM Streamline and demonstrate their features. Some of the features are used in explicit instructions that you are expected to follow and are marked with this symbol ⇨. Other features are only mentioned and are not critical to the flow of the workshop. You don't have to use them, but you are generally encouraged to try them out anyway.

We will use a Vybrid board as the target, but DS-5 can also be used with any ARM Linux target that has networking or the Fixed Virtual Platforms (FVP) that come with DS-5. The Threads application that will be used is supplied as an example with DS-5.

(version 11 June 2013; v5.14.1 d1810; HW; HWCortex-A5; CMSIS-DAP; Vybrid Rev H; TWR-RGB)

Contents

Preparation

If the host and target have already been setup for you, you can skip down to *Starting DS-5* Eclipse below. Otherwise you'll need to start by installing some software.

Host Setup

DS-5 can be used on Windows and Linux hosts. If your host has not already been setup for you then you will need to follow the instructions in the appendix on page 28

Target Setup

For the workshop you'll require a Vybrid Tower stack (see Target Requirements on page 28) that has been setup with U-Boot, a Linux kernel and root file system on it. If you want to setup your own software stack please see Appendix B: Target software stack setup on page 30.

Demonstration setup (without a debug connection)

The WebGL accelerometer demonstration provided by Timesys is setup to run Linux on the Vybrid Cortex-A5 core. The Linux boot sequence is setup to load the accelerometer application onto the Cortex-M4 core and to start running the image. Through the use of a multi core communication (mcc) driver Linux applications can query the MQX accelerometer application for the inclination (x,y,z) of the board. In turn a web server (boa) on the target executes a WebGL script to draw any rotation of the picture displayed in your browser.

- ⇨ Plug in an Ethernet and connect it to your local PC. The board is set to static IP of 169.254.0.100. Set your PC to something similar like 169.254.0.1.
- ⇨ Point your host browser (it needs to be WebGL capable for example Google Chrome browser) at the webpage on the target http://169.254.0.100

You should see an image of the target with a slight jitter, as is depicted in the top right hand corner of the front page of the document. When you physically move the board the image on your host should also move. There is no jitter analysis added to the code.

Starting DS-5 Eclipse

- ⇨ **Start > All Programs > ARM DS-5 > Eclipse for DS-5** and choose a location for the workspace where Eclipse projects will be stored. The default workspace location is fine.
- ⇨ If this is the first time you've started DS-5 you will see the "Welcome to ARM DS-5" home page; click **Go to the workbench**. You can get the **Welcome** page back if you want it later by choosing **Help > Welcome**.

Installing a DS-5 License

Obtain and install license for the Vybrid Starter Kit by following the instructions in **Licence with Activation Code** section on http://ds.arm.com/vybrid/vybrid-tower-starter-kit.

Alternatively you can generate 30 day evaluation license from within DS-5 Eclipse Help Menu **ARM License Manager ... -> Add License ... -> Generate 30-day evaluation license.**

Eclipse views (a brief digression)

The Eclipse window is divided into a number of rectangular *panes*. Each pane contains one or more *views*. Each view has a tab at the top with the view's name. For example, in the screenshot on the cover, **Debug Control**, **Project Explorer** and **Remote Systems** are three views in the same pane. Views of source files are named after their files, for example **main.cpp**.

You can easily rearrange panes and views. Dragging the dividing lines between panes changes their size. Clicking on a view's tab brings it to the front. You can drag a view's tab to move the view to another pane or to split a pane vertically or horizontally. You can even drag a view out of the Eclipse window entirely to

create a new window. Double-clicking on the tabs of a pane will expand the pane to fill the window (maximize). Double-clicking it again will restore it down.

You can close a view by clicking the close button in its tab. If a view you want is not open, you can open it by using the **Window > Show view >** menu.

A particular grouping of views is called a perspective. For example, there is a **DS-5 Debug** perspective that we will be using, but there is also a C/C++ perspective and others. Eclipse provides ways to switch quickly between perspectives and to create your own custom perspectives from a group of views.

Some of the DS-5 views such as **Variables**, **Expressions**, **Registers, Functions** and **Modules** have columns. You can change the width of the columns by dragging the dividing lines in the column headers right or left. You can change which columns are displayed by right-clicking on the column headers and choosing from the context menu. In **Functions** and **Modules** you can sort by any of the columns by clicking on the column heading. If you shift-click a different columns you can set further minor sort keys which shows as the sort arrow and two or more dots.

There are more things you can do, such as minimizing panes and setting views to be pop-up "Fast Views", but that's probably enough to get you started.

Getting help (more digression)

You can press the **F1** key (or Shift+**F1** on Linux) at any time to get help about the current view. You can use **Help > Help Contents > ARM DS-5 Documentation** to view the documentation. You can access cheat sheets for various tasks using **Help > Cheat Sheets > ARM …**, for example importing the example projects. (But we won't be following those cheat sheets here.)

Importing threads

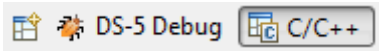
We need to import the two example projects **distribution** and **gnometris** if they have not been imported already.

⇨ Look in the **Project Explorer** view and if **threads** do not appear there follow the instructions for importing them in *Importing projects* in the appendix on page 28.

If **Project > Build Automatically** is checked, the **`threads`** project will be built automatically after it has been imported. The build produces a binary at the top level of the project: **`threads`**, which is the application we will run on the target. Copies of this files with the debug information removed are created in the **`stripped`** subdirectory of the project. The project contains a pre-built copy of the application which will be overwritten when you build the project.

Connecting to the Target

Next, we'll establish a connection to the target using Eclipse's Remote Systems Explorer (RSE) so that we can browse its file system and create a terminal connection. The Remote Systems view is part of the **DS-5 Debug** perspective, so we'll switch to that perspective now. It's possible that the RSE connection has already been created in the **Remote Systems** view for you.

⇨ Choose **Window > Open Perspective > DS-5 Debug**. If **DS-5 Debug** isn't listed then you are already in the **DS-5 Debug** perspective. You can also switch perspectives by using the buttons on the Perspective toolbar (

).

⇨ If the **Remote Systems** view is not open, you can open it by choosing **Window > Show View > Other... > Remote Systems > Remote Systems** in any perspective.

⇨ Click on the tab of the **Remote Systems** view to bring it to the front.

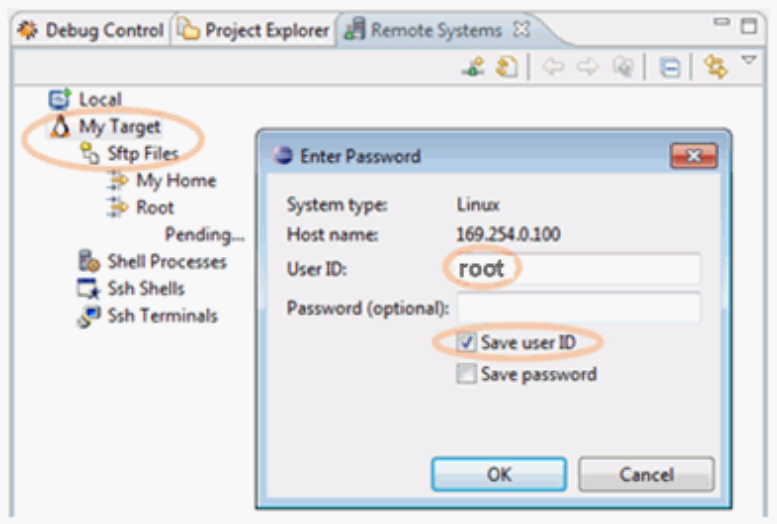
⇨ We won't be using the **Local** connection, so you can collapse it.

⇨ If there is no **My Target** connection, create one by following the instructions in the appendix Creating a Target Connection on page 29**Error! Bookmark not defined.**.

⇨ Browse the target's file system; Expand **My Target > Sftp files > Root**. If the connection has **Files** instead of **Sftp Files**, then the connection was not created correctly and you should **Disconnect** it, **Delete** it and recreate it.

⇨ Enter User ID=**root**, Password=**root**; check **Save user ID** and click the **OK** button. There will be a few authentication dialogs; accept them.

You can also expand **My Home** to browse the home directory of the user, which is **`/root`** for the root user.

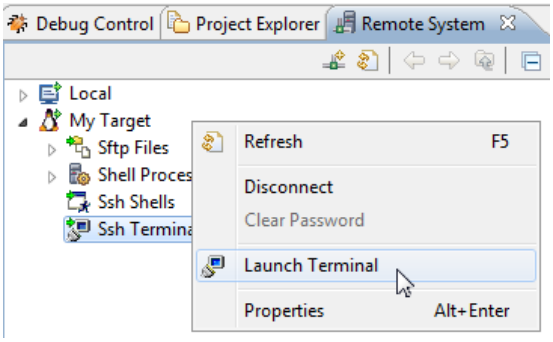


You can copy files to and from the target by dragging them between the **Remote Systems** view and the **Project Explorer** view or Windows Explorer windows. As we'll see below, it's also possible to copy files to the target automatically as part of the debug configuration.

If you want, you can double click a text file on the target, for example **`/etc/bash.bashrc`**, and view it or even edit it in Eclipse – but it's probably best if you don't save any changes unless you are sure that you know what you are doing.

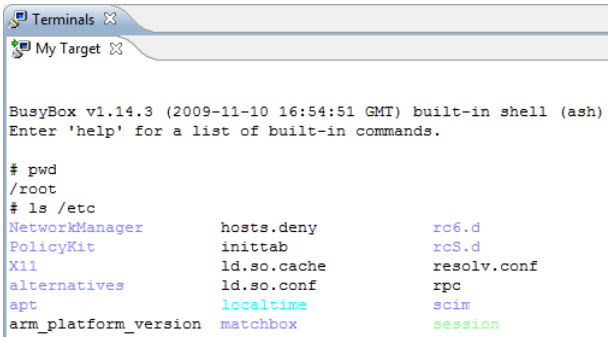
Now we'll create a terminal connection so that we can execute commands easily on the target. You can Collapse the **Sftp Files** to get them out of the way.

- ⇨ Create a Terminal by right-clicking on **Ssh Terminals** and choosing **Launch Terminal**.



This will open a **Terminals** view in Eclipse that can be used to execute commands on the target. (This is different from, but confusingly similar to the **Serial** view). The picture below shows the output from the example target image included with DS-5. If your target is running a different distribution the output will be different.

You can use **Launch Terminal** more than once if you want to have multiple terminal sessions to the target.



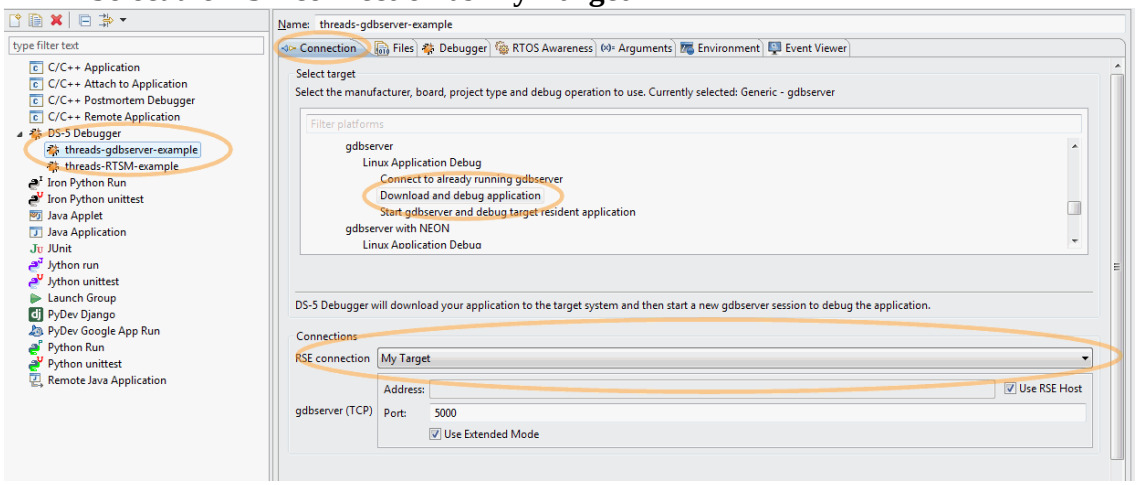
Debug a Linux application - Threads example

As a start and an introduction to the debugger let's debug a Linux example that is provided in the DS-5 examples.

The Threads example project includes two debug configurations, **threads-gdbserver-example,** and **threads-RTSM-example**, for running the application on the Real Time System Model (RTSM) or in later editions renamed to Fixed Virtual Platform (FVP). You can try the RTSM debug configuration later if you want, but we are going to ignore them for now and make our own.

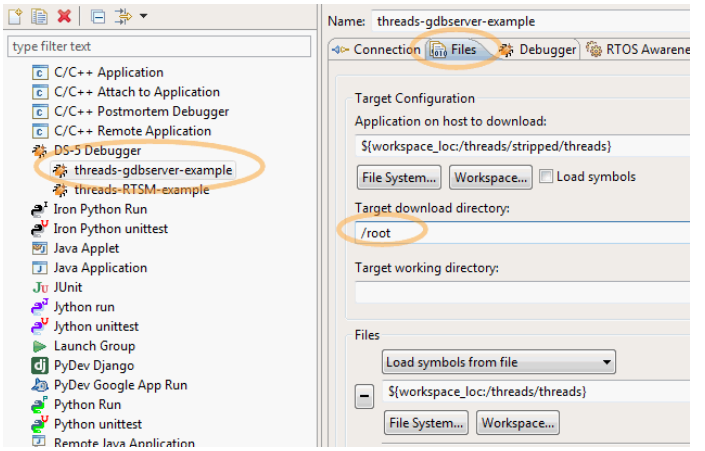
Let's go edit the **threads-gdbserver-example** and modify it for our target

- ⇨ Click **Run > Debug Configurations**
- ⇨ Expand the **DS-5 Debugger** section if it's not already expanded
- ⇨ Click on the **threads-gdbserver-example** debug configuration
- ⇨ Select the **RSE connection** as **My Target**



Note the other settings that have already been selected.

- ⇨ In the **Files** tab enter the target download directory as **/root**
- ⇨ Select the **RSE connection** as **My Target**



⇨ Inspect the **Debugge**r tab. Here we can set the behaviour of the debugger on at connection time for example stop at function `main()` or execute to script of commands etc.

⇨ Click **Debug**

Congratulations you are debugging your first Linux application using DS-5. Now let's do some debugging, the DS-5 documentation contains some helpful information which you can find from the **Help** menu, let's go find some debug hints for this application.

⇨ Click **Help -> Welcome to ARM DS-5 -> Examples**

⇨ Search for the **Debugging Multi-Threaded ARM Linux Applications** example. Note you may find it useful to undock the Welcome window from Eclipse by dragging it outside of the Eclipse boundaries.

⇨ Scroll to the section **Debugging the threads application** and follow the instruction.

⇨ Once you are finished click the **Disconnect** ( ) button in the **Debug Control** view.

Debug MQX on the Cortex-M4 using CMSIS-DAP

This section will demonstrate how to debug the accelerometer application running on the Cortex-M4 core. We will also demonstrate the MQX operation system support in the DS-5 debugger and a few other features

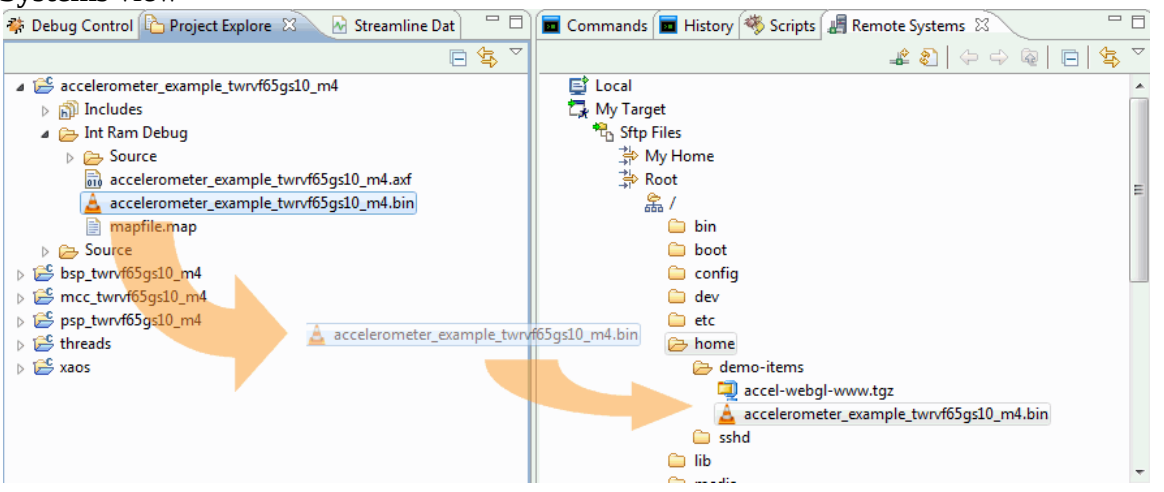
Note: this to work with your target you need to have the CMSIS-DAP firmware installed on your target.

First we need to import, build and copy the MQX accelerometer application to the target

⇨ If you haven't done so already follow the instructions in the appendix Importing and Build MQX Projects on page 29 to build the example

⇨ Open the **Remote Systems** view; drag the tab of the **Remote Systems** view so that you can see it and the **Project Explorer** view at the same time.

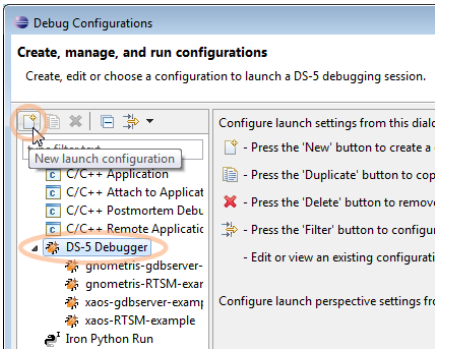
⇨ From the **Project Explorer** drag the file "`accelerometer_example_twrvf65gs10_m4/Int Ram Debug/accelerometer_example_twrvf65gs10_m4.bin`" to the **/home/demo-items/** in the **Remote Systems** view



⇨ **Reboot** the board, so that the new binary we copied onto the target is loaded by Linux on the Cortex-M4.

⇨ Choose **Run > Debug Configurations...** then expand **DS-5 Debugger**

⇨ Select **DS-5 Debugger** and click the **New launch configuration** button ( ) to create a new debug configuration. (You can also double-click **DS-5 Debugger** or right-click it and choose **New** instead.)



⇨ Give the debug configuration a name, I used ***MQX Vybrid Cortex-M4 Debug***

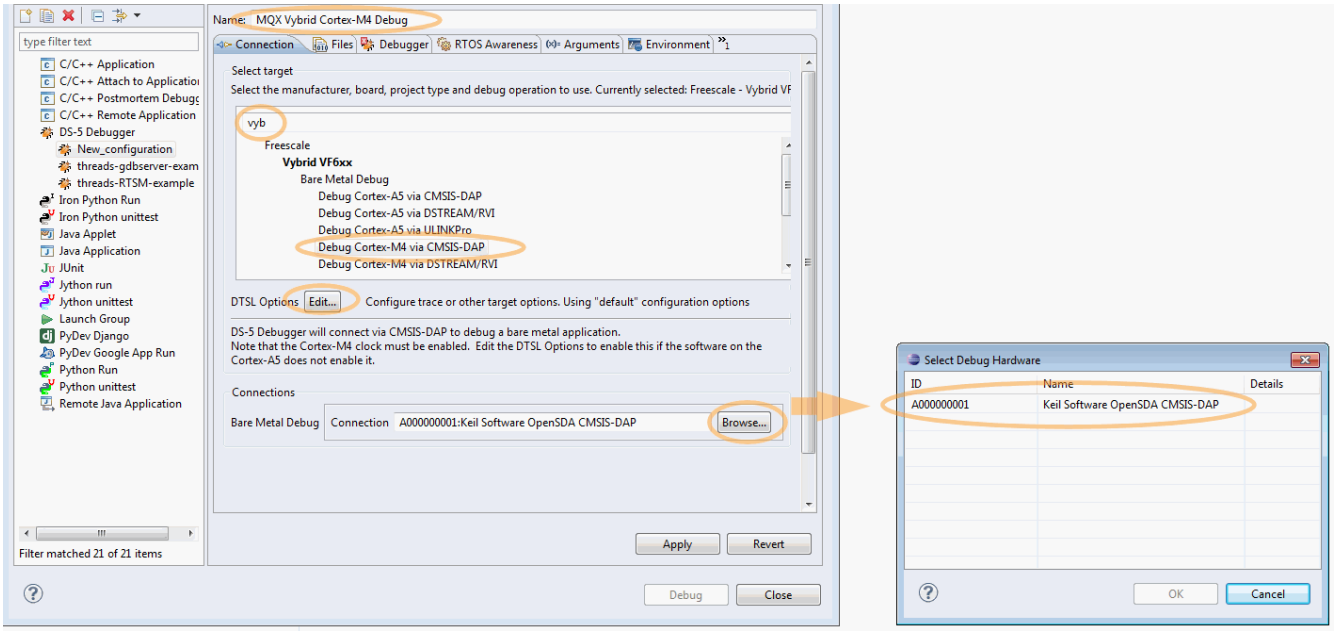
In the **Connection** pane of the debug configuration we need to specify the platform and choose the Bare Metal debug connection method. The platforms list contains the large number of platforms that DS-5 supports by default arranged as a tree by vendor.

⇨ Type ***vyb*** in the Filter platforms filter box so that the only the matching platforms are shown.

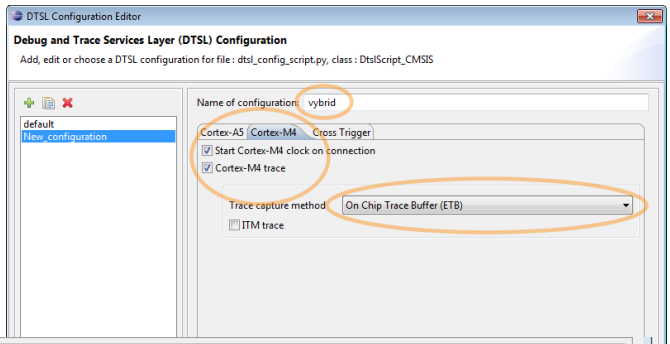
⇨ Expand **Freescale > Vybrid VF6xx >Bare Metal Debug** in the platforms list.

⇨ Select **Debug Cortex-M4 via CMSIS-DAP**

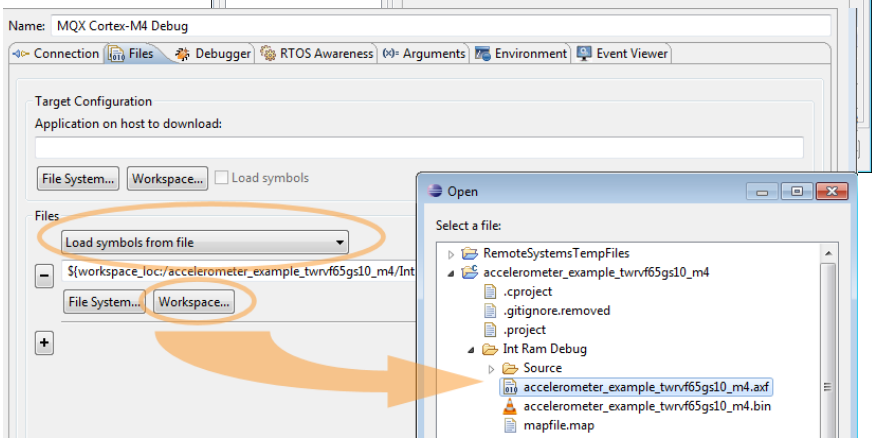
⇨ Click the **Browse** button and choose the *Keil Software OpenSDA CMSIS-DAP* connection and click **OK**. Your Connection number may be different than shown.



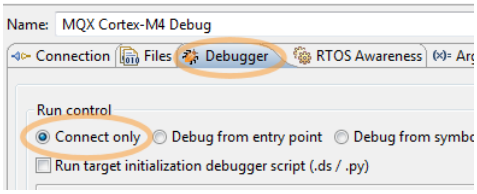
The tracing options are configured in a separate DTSL configuration. "DTSL" stands for Debug and Trace Services Layer.



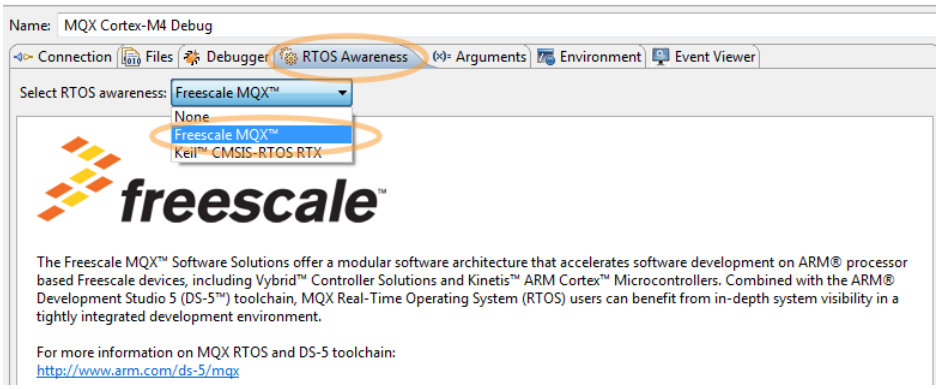
- ⇨ Click the **DTSL Options Edit…** button to open the **DTSL Configuration Editor** dialog box.
- ⇨ Click the Add button (✚) to create a new DTSL configuration.
- ⇨ Give the new DTSL configuration a name, I used ***Vybrid***
- ⇨ In the **Cortex-M4** pane check the boxes for **Start Cortex-M4 clock on connection** and **Cortex-M4 Trace**
- ⇨ Change the **Trace capture method** to **On Chip Trace Buffer (ETB)**.
- ⇨ Click **Apply** and **OK**



- ⇨ In the **Files** pane choose **Load symbols from file** and select from the **Workspace** the accelerometer example binary "`accelerometer_example_twrvf65gs10_m4/Int Ram Debug/accelerometer_example_twrvf65gs10_m4.axf`"
- ⇨ In the **Debugger** pane choose **Connect only**



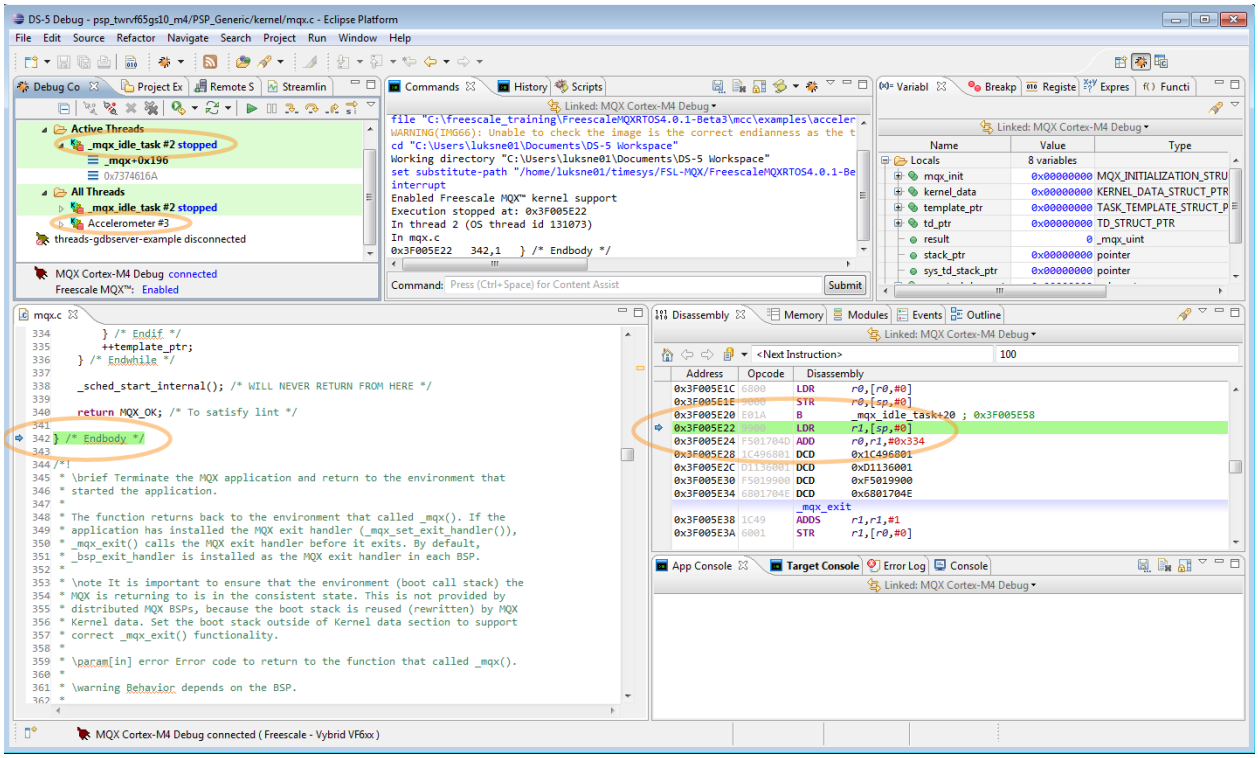
- ⇨ In the **RTOS Awareness** pane choose **Freescale MQX**
- ⇨ Click **Apply** and **Debug** to start the debug session



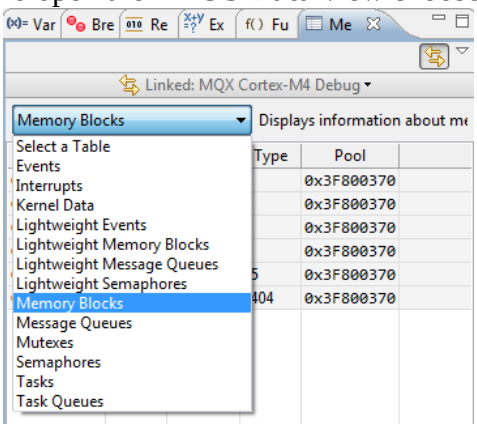
The debug configuration is set connect to the target only, and as the target was running at connection time it will be in a running state now.

- ⇨ Click the Interrupt button ( ) in the **Debug Control** view to stop the target:

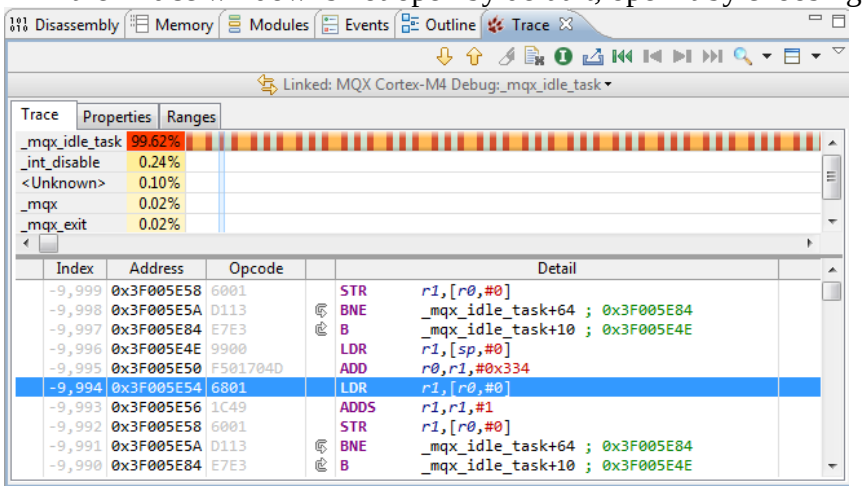
The target will be interrupted and you should see the DS-5 views like the **Source Code**, **Disassembly, Variables** and **Debug Control** views being updated. Fold open **All Threads** in the **Debug Control** view to see al the current threads running on the host operating system.



⇨ To open the **RTOS Data** view choose **Window > Show View > RTOS Data**

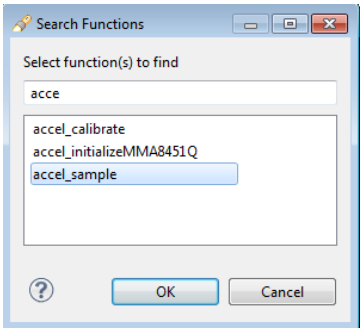


⇨ If the **Trace** window is not open by default, open it by choosing **Window > Show View > Trace**



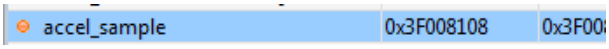
The **Functions** view shows information about all of the functions in the debug symbols we have loaded including the start and end addresses. You can change the sorting of the **Functions** view by clicking on the column headings. You can use the **Filters...** command in the view's drop-down menu ( ) to control which images and compilation units are shown.

⇨ Click on the Search button ( ) of the **Functions** view to open the **Search Functions** dialog then type some of the function's name, select **`accel_sample`** and click **OK**:



The entry for **`accel_sample`** is selected.

⇨ We can set a breakpoint on it by double clicking on function name and the dot next to the function will turn red



There are also Search buttons ( ) in the **Variables**, **Expressions**, **Registers**, **Disassembly** and **Memory** views.

⇨ Click the Continue button ( ) to put the target into a running state

⇨ Open http://169.254.0.100 in your web browser so that Linux will query MQX for accelerometer measurements

The breakpoint should be hit as soon as the browser refreshes, and we can now continue debugging the application ☺

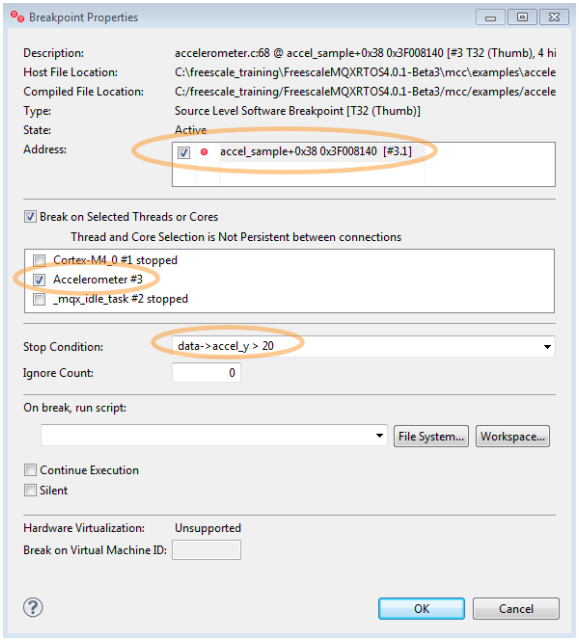
⇨ As an exercise try to place an advanced breakpoint that will only be hit if you tilt the board to far to the side (HINT: right click on the breakpoint and select properties. The answer is in the next step, so don't read on if you want to try it yourself)

⇨ **Delete** the breakpoint on **`accel_sample`** by double click on the break point in the source code.

⇨ Right click on line 68 in accelerometer.c and choose **Toggle Breakpoint**

⇨ Right click on the breakpoint next to line 68 and choose **Breakpoint Properties...**

⇨ Enable **Break on Selected Threads or Cores**



We don't really need to enable break on selected threads for this example as the only thread that will run this code is the accelerometer thread.

⇨ Set the **Stop Condition** as `data-> accel_y > 20`

⇨ Click the Continue button ( ) to put the target into a running state

⇨ Restart the web browser and point it to `http://169.254.0.100`

⇨ Looking directly at the LCD on the target tilt the board backwards. The breakpoint should be hit when you tilt the board far enough - 45° should be enough.

The jitter on the target will be slower and the movement of the picture in your browser will be less responsive. The reason is that we have set a software breakpoint in SRAM which is slowing down the execution. Hardware breakpoints may only be set below addresses 0x20000000 in FLASH for the Cortex-M4.

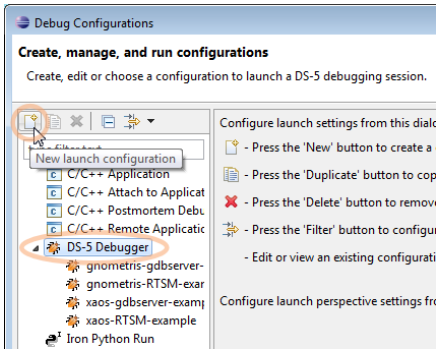
⇨ With the target running, click the **Disconnect From Target** button ( ) in the **Debugger Control** view to disconnect from the debug session.

Debug Linux kernel on the Cortex-A5 using CMSIS-DAP

To develop, port and debug the Linux kernel on a platform, you will need to set breakpoints, view registers, view memory, single-step at source level and so on - all the normal facilities provided by a debugger. You will probably also need to do these both before the MMU is enabled (with a physical memory map), and after the MMU is enabled (with a virtual memory map). DS-5 allows you to do all this and more, not just for single-core platforms, but for SMP platforms too. In this workshop, we'll debug the Linux kernel after it has booted and also debug the multi core communication (MCC) driver. See Appendix @@@ for instructions to debug the kernel start up.

DS-5 Debugger has a slick Debug Configuration dialog in Eclipse that makes it easy to configure a debugging session to a target. Predefined debug configuration types include "Bare Metal Debug", "Linux Application Debug", and "Linux Kernel debug". The latter is the topic of this workshop. This **Linux Kernel debug** configuration type is primarily designed for post-MMU debug to provide full kernel awareness but, with some extra controls, can also be used for pre-MMU debug too. This makes it possible to debug the Linux kernel, all the way from its entry point, through the pre-MMU stages, and then seamlessly through the MMU enable stage to post-MMU debug with full kernel awareness, all with source-level symbols and all without the need for tedious disconnecting/reconfiguring/reconnecting!

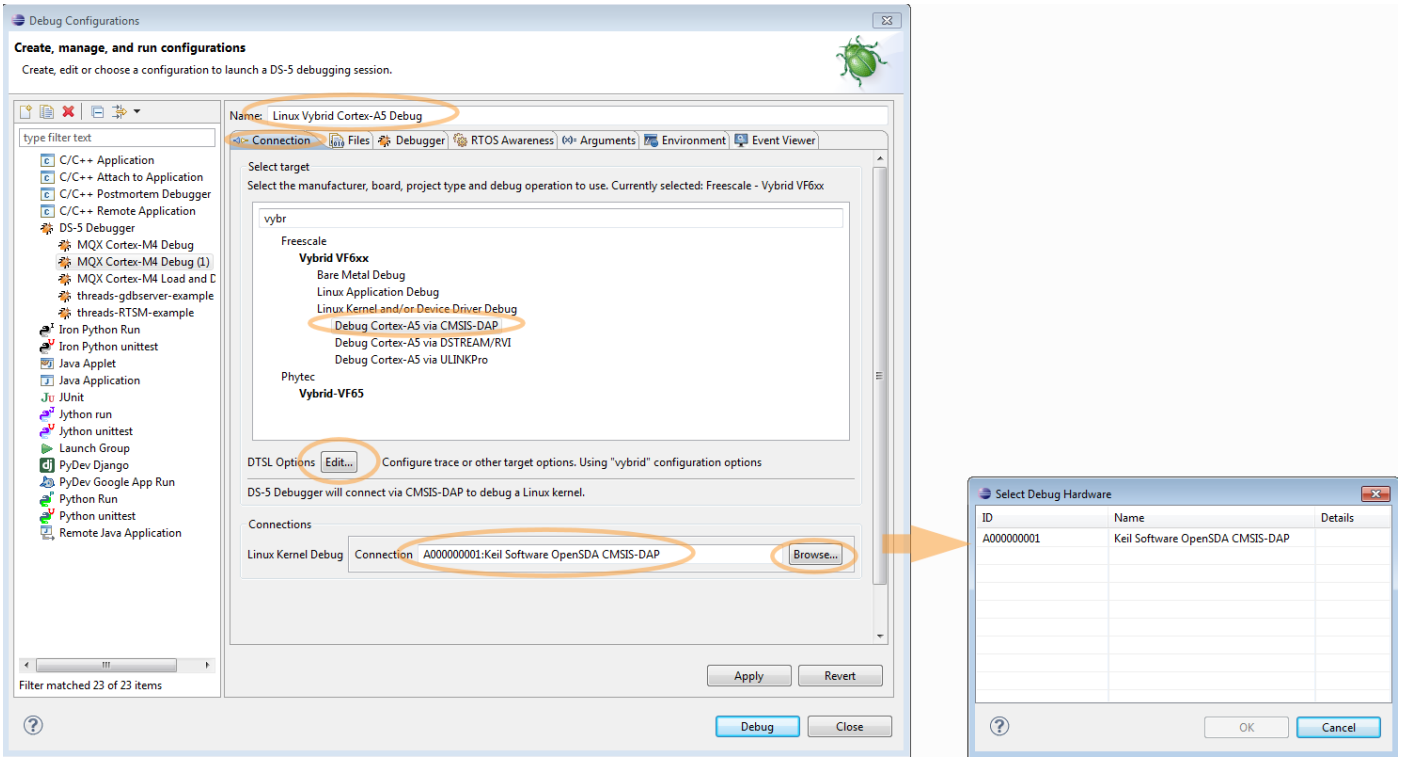
Now it's time to examine or create a debug configuration for the kernel when the MMU is on.



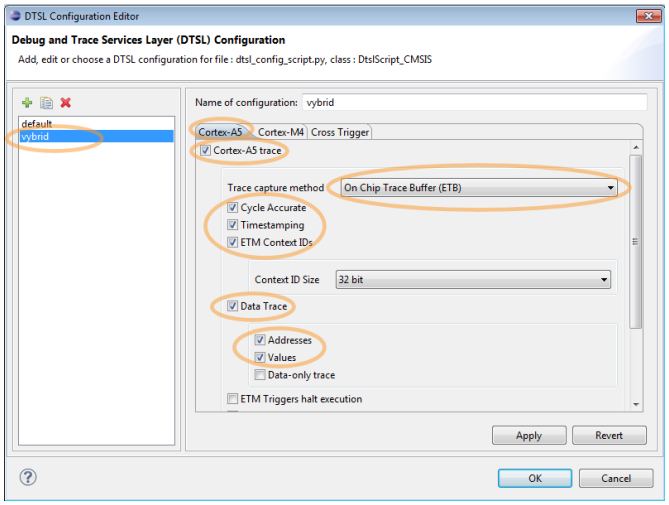
- ⇨ Choose **Run > Debug Configurations...** then expand **DS-5 Debugger**.
- ⇨ Select **DS-5 Debugger** and click the **New launch configuration** button ( ) to create a new debug configuration. (You can also double-click **DS-5 Debugger** or right-click it and choose **New** instead.)
- ⇨ Give the debug configuration a name, I used ***Linux Vybrid Cortex-A5 Debug***.

In the **Connection** pane of the debug configuration we need to specify the platform and choose the Linux Kernel debug connection method. The platforms list contains the large number of platforms that DS-5 supports by default arranged as a tree by vendor.

- ⇨ Type ***vyb*** in the Filter platforms filter box so that the only the matching platforms are shown.
- ⇨ Expand **Freescale > Vybrid VF6xx >Linux Kernel and/or Device Driver Debug** in the platforms list.
- ⇨ Select **Debug Cortex-A5 via CMSIS-DAP**
- ⇨ Click the **Browse** button and choose the *Keil Software OpenSDA CMSIS-DAP* connection and click **OK**. Your Connection number may be different than shown.



The tracing options are configured in a separate DTSL configuration. "DTSL" stands for Debug and Trace Services Layer. We've already created a configuration for Vybrid to enable trace on the Cortex-M4. Now let's go modify that configuration to enable trace from the Cortex-A5.



- ⇨ Click the **DTSL Options Edit…** button to open the **DTSL Configuration Editor** dialog box.
- ⇨ Select the ***vybrid*** connection you created in the previous section
- ⇨ In the **Cortex-A5** pace check the boxes for **Cortex-A5 Trace**, **Cycle Accurate**, **Timestamping**, **ETM Context ID's**, **Data Trace**, **Addresses** and **Values**
- ⇨ Change the **Trace capture method** to **On Chip Trace Buffer (ETB)**.
- ⇨ Click **Apply** and **OK**

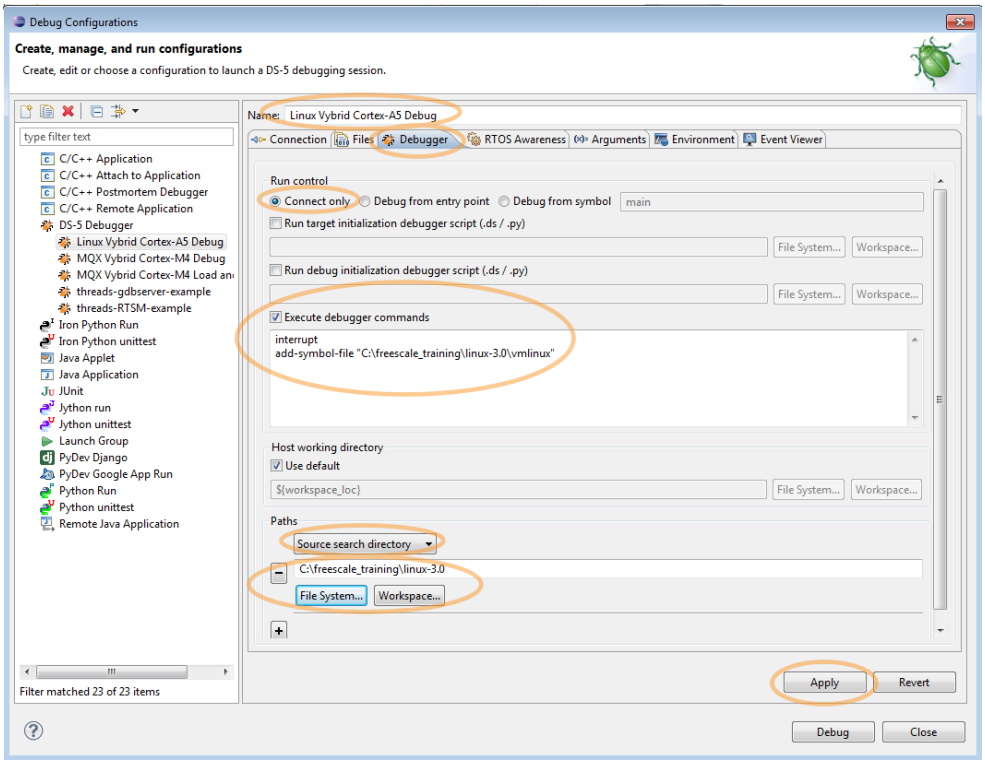
We'll leave the **Files** pane of the debug configuration blank so that we can be sure that the target is stopped when we load the debug symbols. We'll load them in a script in the **Debugger** pane.

In the **Debugger** pane of the debug configuration:

- ⇨ Choose **Connect only** so that DS-5 will just attach to the target which is already running the kernel.
- ⇨ Check **Execute debugger commands** and type these commands into the field (you'll need to change the path to vmlinux to match that of your host):

```
interrupt
add-symbol-file "c:\training\linux-3.0\vmlinux"
```

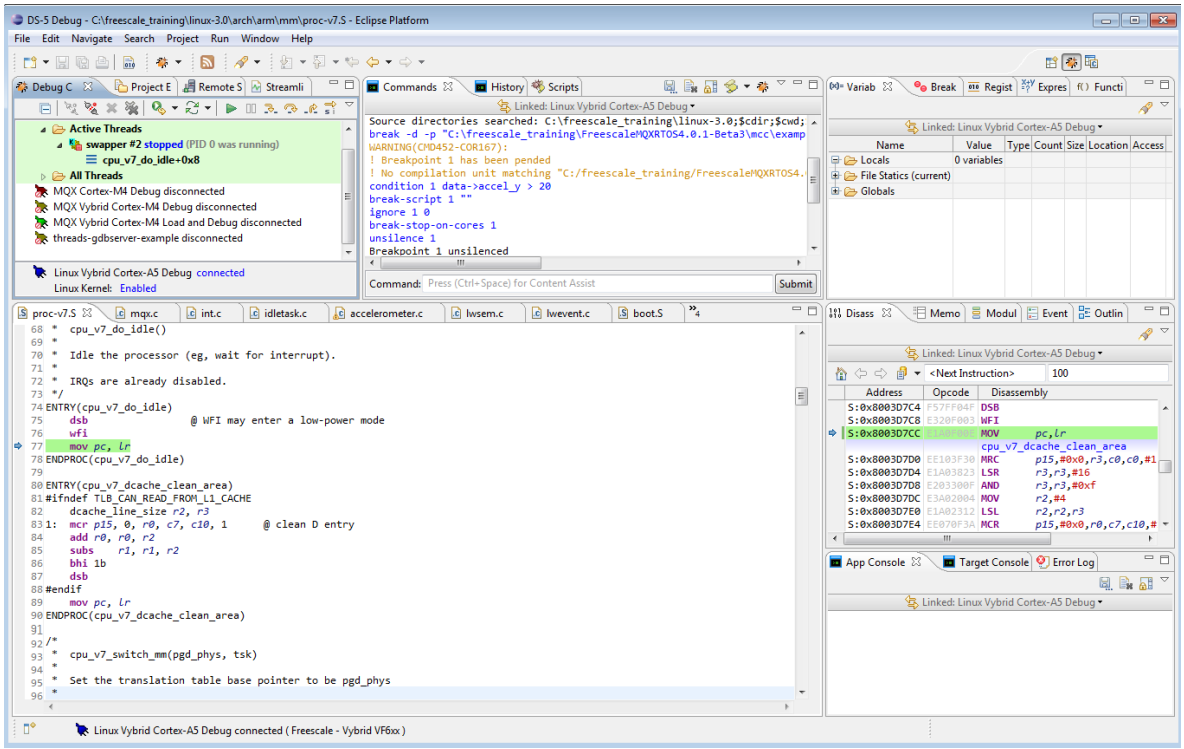
- ⇨ In the **Paths** list select **Source search directory** as " `c:\training\linux-3.0`". You'll need to modify this path to match that for your host. If you built the kernel on your own host, you would not need this step.



Since we modified the global trace configuration for the Vybrid connection, we'd need to disconnect and reconnect the connection to the **Cortex-M4** core. If you have not done so before you debug the **Cortex-A5** you will run into an error message.

- ⇨ If you are connected to the **Cortex-M4** disconnect the session now.
- ⇨ Click **Apply** and **Debug** to start the debug session

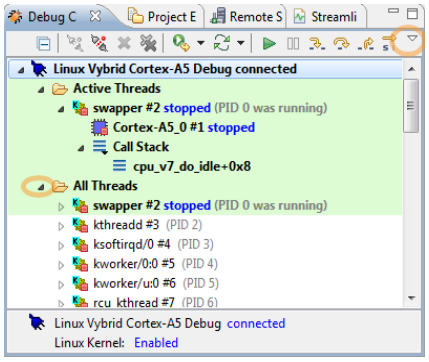
Once you are connected you will be presented with a similar screen to the one below. TIP you can reset your perspective to the default settings by selecting **Window > Reset Perspective**



In the **Debug Control** view we can now also see the processes and threads instead of just the cores that we saw when debugging the bare-metal U-Boot.

⇨ Expand the **All Threads** folder in the **Debug Control** view to see all the processes and threads:

You can expand each thread to see the thread's stack. You can change the debugger's focus from one thread to another by clicking on the different threads and stack frames. You can change the way child threads are displayed by choosing **Flat** or **Hierarchical** from the **Thread Presentation** submenu of the **Debug Control** view's drop-down menu ( ). From the drop-down menu you can also select **Always Show Cores** to display the cores ( ) like in bare-metal debugging as well.



Kernel space threads are shown with a ( ) icon and user space threads are shown with a ( ) icon.

Reading the thread information for all threads takes some time, so it's a good idea to leave the **All Threads** folder collapsed when you don't need it.

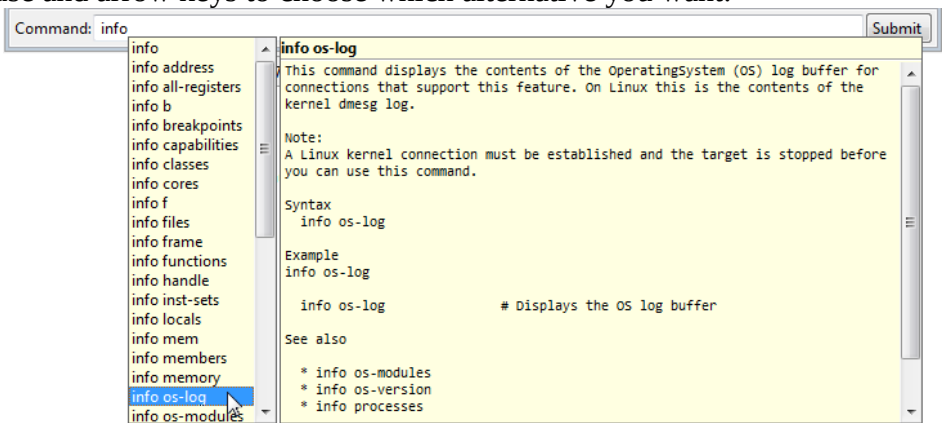
⇨ Collapse the **All Threads** folder to hide the non-active threads.

Kernel **`printk`** functions print messages are normally directed to the serial port to aid with the debug of kernel start-up. At very early stage in the kernel boot process the serial port driver for the kernel may not be initialised yet. Additionally, in some cases a physical serial port connection may not be available.

You can use the **`info`** command to get information about the kernel even if the target doesn't have a serial console. You can use Content Assist (Ctrl+Space) to get help on commands as you are typing them.

INFO: In Windows, if foreign language support is enabled, the Ctrl+Space key combination is used to change between languages inside a text box (for example English to Chinese). You can change the key combination to something else in Eclipse by going to **Window > Preferences > General > Keys > Content Assist** to change the key combination to something else.

⇨ Type **`info`** into the **Command** field of the **Commands** view and then type Ctrl+Space. This activates the Content Assist which shows the possible completions and help for each. You can use the mouse and arrow keys to choose which alternative you want:

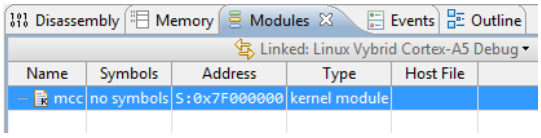


⇨ Try the **`info os-version`** and **`info os-log`** commands. The **`info os-log`** command shows the kernel message buffer. (**`printk`**; **`dmesg`**);

The other views you encountered (except for the **RTOS Data** view) are available for the Linux kernel debug connections. These include the **Functions**, **Variables**, **Expressions**, **Breakpoints** views and many. For more information on these views please see the the DS-5 Documentation **Help > Help Contents > ARM DS-5 Documentation.**

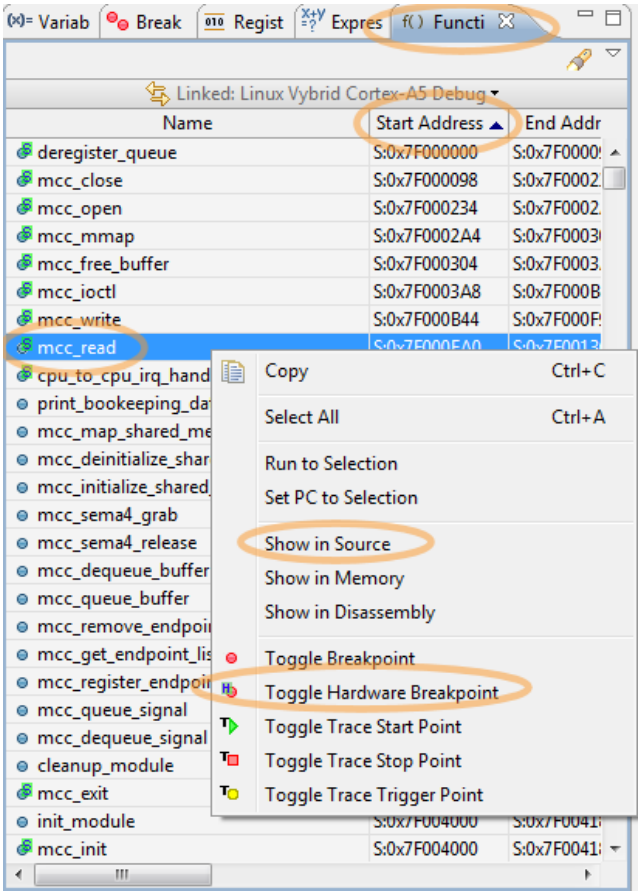
As an exercise let's go debug the MCC driver which should be loaded by the Linux start up scripts.

- If the target is running click the Interrupt button ( ) to stop the target.
- Select **Window > Show View > Modules** to open the modules view
- Right click on the mcc entry in the **Modules** view and select **Add Symbol File...** to add the symbols which will enable you to debug the module. The location of the symbols will depend on your host setup for example `"mcc-kmod-1.02\mcc-kmod-1.02\mcc.ko"` inside the Timesys build directory.



From the modules view we can see that the mcc driver is loaded into memory at location S:0x7F000000. As I don't know the code, it might be handy for me to see which functions are compiled into the driver. In the **Functions** view

- Click on the **Start Address** column to sort the view by start address so we can see a list of the functions in the mcc driver.
- Inspect the source code of the `mcc_read()` function by right clicking on the function name and choosing **Show in Source**
- Set a hardware breakpoint on `mcc_read` by selecting **Toggle Hardware Breakpoint**
- Click the Continue button ( ) to put the target into a running state
- Refresh your browser screen to get the breakpoint to be hit – we can now debug the driver



As an exercise try to find the accelerometer data that is being received by the mcc_read() function.

You can have more than one debug connection at the same time.

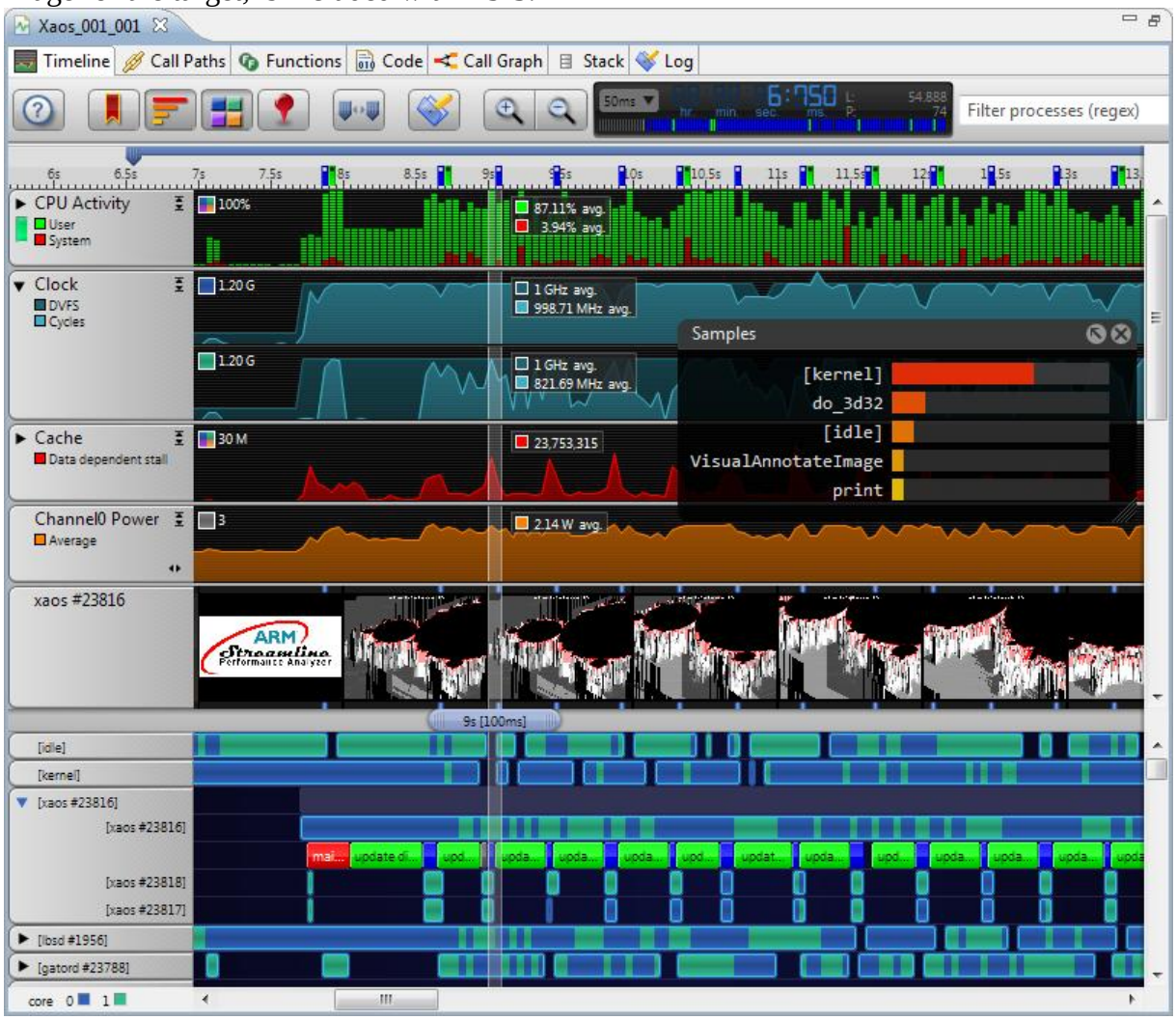
- Make a connection to the Cortex-M4 by right clicking on your ***Cortex-M4 Vybrid Debug*** configuration in the **Debug Control** view.

Now you will have two debug connections at the same time. If you select (left click) a connection in the **Debug Control** view the other views in the debugger will change to match that connection. Some windows (like the **Registers** view) can be pinned to a specific connection by selecting the connection at the top of the window.

ARM Streamline – Profiling on Vybrid

ARM® Streamline™ is a graphical performance analysis tool. Combining a Linux kernel driver, target daemon, and an Eclipse-based user interface, it transforms sampled data and system trace into reports that present the data in both visual and statistical forms. ARM Streamline uses hardware performance counters along with kernel metrics to provide an accurate representation of system resource use.

This part of the workshop demonstrates the use and features of the ARM Streamline performance analyser to inspect applications running on an ARM Linux target. The workshop details the Linux configuration and setup required to enable application profiling. It demonstrates how use the Streamline report to analyse and investigate application performance and power. All of the software you need, including the ARM Linux image for the target, is included with DS-5.

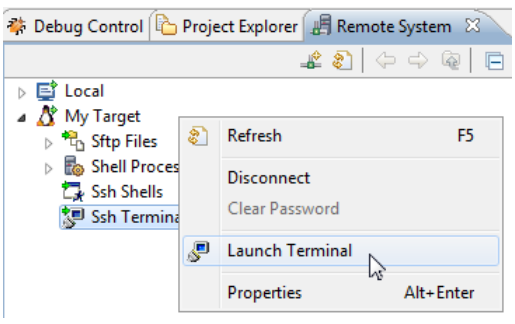


Target setup

If enabled in the build instructions (see Appendix B: Target software stack setup on page 30) the DS-5 Streamline driver and user space modules are included in the Timesys build for Vybrid.

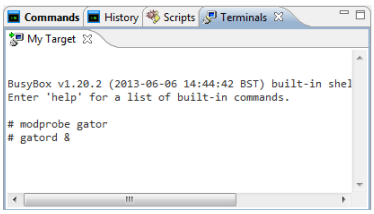
Now we'll load the Streamline kernel driver (gator) and start the user space daemon so that we can do some profiling. In the **Remote Systems** view

⇨ Create a Terminal by right-clicking on **Ssh Terminals** and choosing **Launch Terminal**.



In the **Terminals** view, type

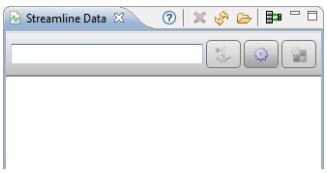
- `modprobe gator`
- `gatord &`



Set the Capture Options

If it's not already open, open the **Streamline Data** view:

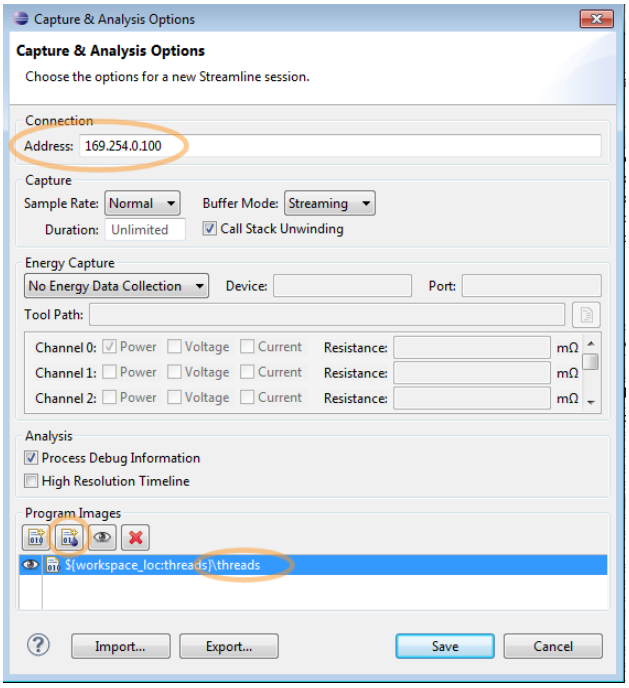
- Select **Window > Show View > Streamline Data**.



Streamline data and reports are shown in the **Streamline Data** view. It will be empty, unless a previous profiling session has been run, in which case saved captures and reports may be present in the view. You can select any existing captures and reports and click the Delete button ( ) to get rid of them. You can get help by clicking the Show Help button ( ).

The **Streamline Data** view has an **Address** field for specifying the name or IP address of the target, a Start Capture button ( ), a Change Capture Options button ( ) and a Counter Configuration button ( ).

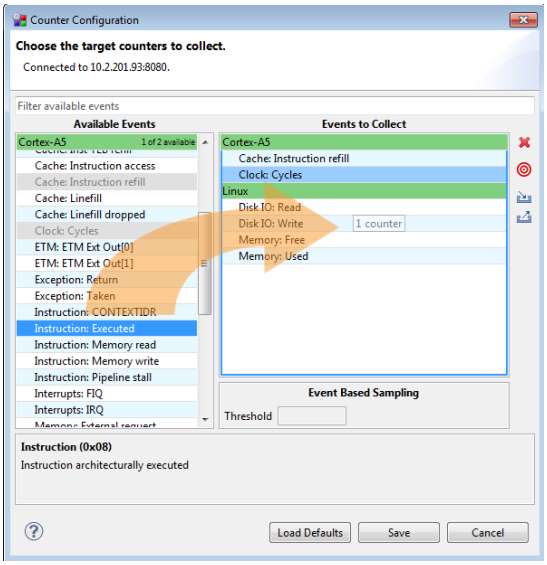
- Click the Capture Options button ( ):
- Set the **Address** field to the IP address of the target `169.254.0.100`.
- In the **Program Images** section, click the Add ELF image from Workspace... (second) button ( ) and choose the `threads` binary with debug information from the workspace



- Click **Save** to save the capture options.

Configure Counters

Advanced ARM processors have a performance monitor unit (PMU) in hardware. Exactly which events can be counted and how many events can be counted at once depends on which ARM processor is being used. The operating system allows access the performance counters for debug and profiling purposes. The Cortex-A board has a cycle counter and 3 configurable event counters.



Streamline defaults to capturing data from various counters in the PMU, kernel and L2 cache controller, depending on the target's hardware and kernel.

- Click the Counter configuration button ( ) to open the **Counter Configuration** dialog where you can examine and change the counters that you wish to capture.

You can hover your mouse over a counter to see a description of the event being counted.

- If they are not already in the **Events to Collect** list, add these counters:

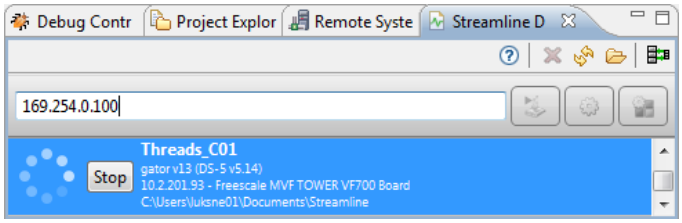
- **Cache: Instruction refill**: to see the number of instruction fetches that cause a cache refill
- **Clock: Cycles:** to see the number of core clock cycles
- **Instructions: Executed:** to see the number of instructions executed

⇨ Click **Save** to save the counter configuration. The counter configuration is saved on the target.

INFO: If you have your own hardware or software counters (for example, number of packets processed) you can modify the gator software and use Streamline to capture and to display charts of your own custom counters. The counters of the L2C-310 cache controller are a good example to start from.

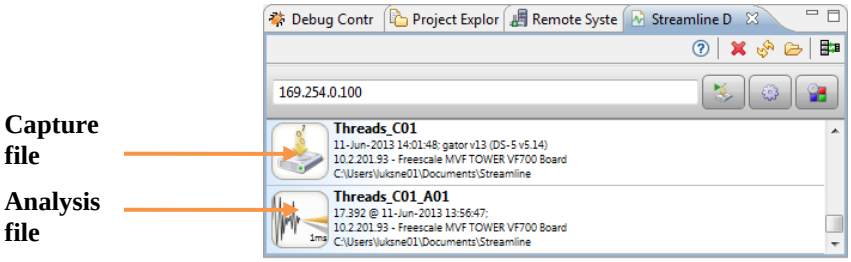
Capture some profile data

⇨ Click the start capture button ( ) to collect data from the target. You are now prompted to specify the name and location of the capture file where the profile data is stored on the host. By default it will be in the directory `...\My Documents\Streamline`. You can change the file name, for instance we are profiling the threads application, so I called it ***Threads_C01.apc***



⇨ In the **Terminal** start the threads application, run it a few times in succession by running `"/root/threads &"`

⇨ Click the **Stop** button in the **Streamline Data** view to stop capturing data and generate a profile report. Streamline will show a spinning progress wheel as it analyses the captured data.



⇨ Click the **Stop** button in the **Streamline Data** view to stop capturing data and generate a profile report. Streamline will show a spinning progress wheel as it analyses the captured data.

Examine the Report

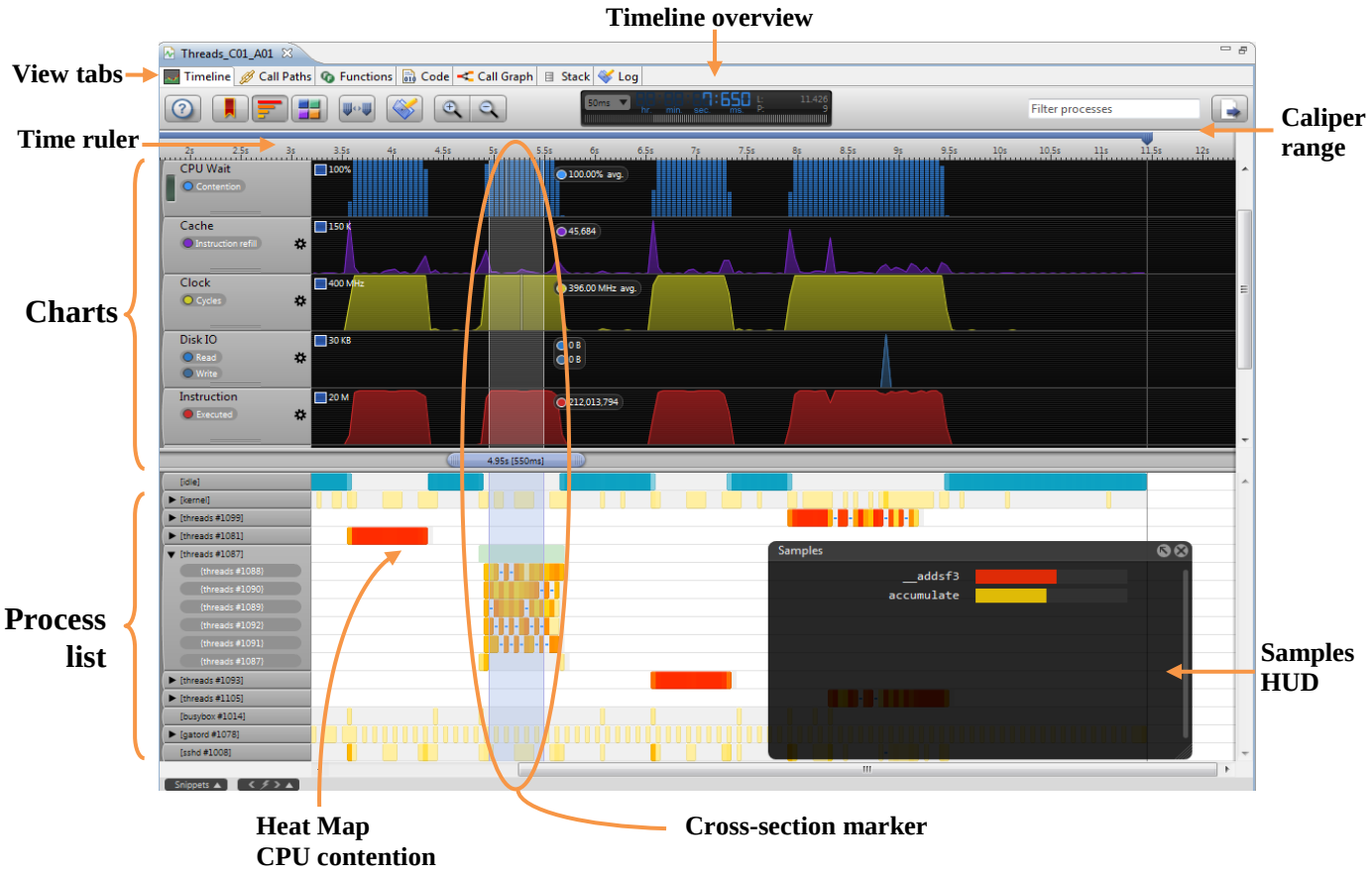
When the analysis is finished, the report view will open with a name determined by the capture's name, for example **Threads_C01_A01** for the first analysis of the **Threads_C01** capture.

⇨ Double click the **Threads_C01_A01** tab of the report view to maximize it.

All of the Streamlines views can be exported to text files by using the Export button ( ) except for the Code and Call Graph views

Timeline view – a first look at the Streamline report

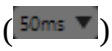
The **Timeline** view is the first view presented when a report is opened. It displays information to give an overall view of the system performance. The graphs and percentages in your report may be different than the ones shown here depending on how long you captured data for.



The top section of the **Timeline** view, below the time ruler, shows the system performance charts. Streamline captures performance data from the hardware and operating system, and displays the samples or "snapshots" of the chosen counters in the system performance charts. The order of the charts can be rearranged by dragging them by the gray legend area at the left end up or down. Just to the right of the legend is an indication of the full vertical scale of each chart (for example, 30 M). You can get help on any of the charts by clicking the **Help** button ( ) in the top left of the **Timeline** view and select **Timeline view charts**.

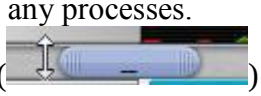
Above the time ruler are the buttons and the Timeline overview. The Timeline overview shows the current zoom level as a drop-down menu (50ms). The large time in the center shows the time

of the current mouse position as you move the mouse. The Timeline overview also shows the length of the capture after the **L** and the number of processes after the **P**. Along the bottom of the Timeline overview is a bar that represents the entire capture. Clicking on the bottom bar will jump scroll the Timeline to that point.

⇨ Click the zoom buttons () to zoom in and out one level at a time (by a factor of 2 or 2.5). You can also use the drop-down menu () in the Timeline overview to change the zoom.

By default the finest resolution is 1 ms. If you use the **High Resolution Timeline** option when doing the analysis the finest resolution is 1 µs. You can also zoom in and out by typing **I** or **+** and **O** or **-**.

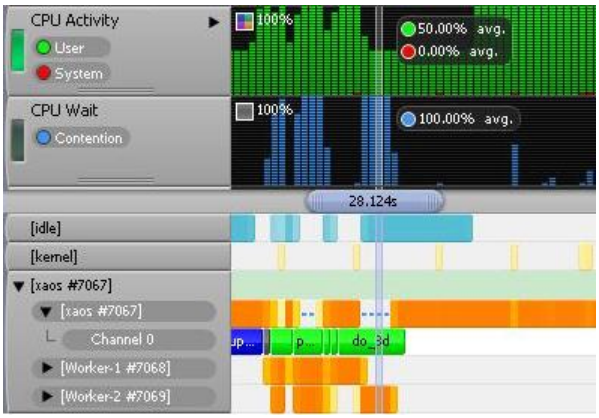
Between the charts and process list is the cross-section thumb bar (). If the cross-section thumb bar is at the bottom of the view, you won't be able to see any processes.

⇨ You can drag the cross-section thumb bar up and down () to adjust how many charts and processes are shown.

By default the process list shows a "heat map" colored by **CPU Activity** in red, orange and yellow. Red indicates that this is a "hot spot" i.e. an area where a lot of time was spent during the execution. These are typically the parts of the code which one would inspect and look to optimize first.

If a process has multiple threads it will have an expansion triangle to the left of the process name that will toggle between showing the individual threads and the aggregate of all the threads. You can see the context switches between processes and threads.

Process/thread contention is shown by in the process list. This is when a process or thread is runnable but other threads are using all of the cores. Contention is also shown in the **CPU Wait/Contention** chart.



We can change to coloring the processes and threads to be "by contention":

⇨ Click on the button at the left end of the **CPU Wait/Contention** chart.

Now the process list has been recolored to show red when processes are waiting to run.

⇨ Click on the button at the left end of the **CPU Activity** chart to restore the normal coloring by CPU use.

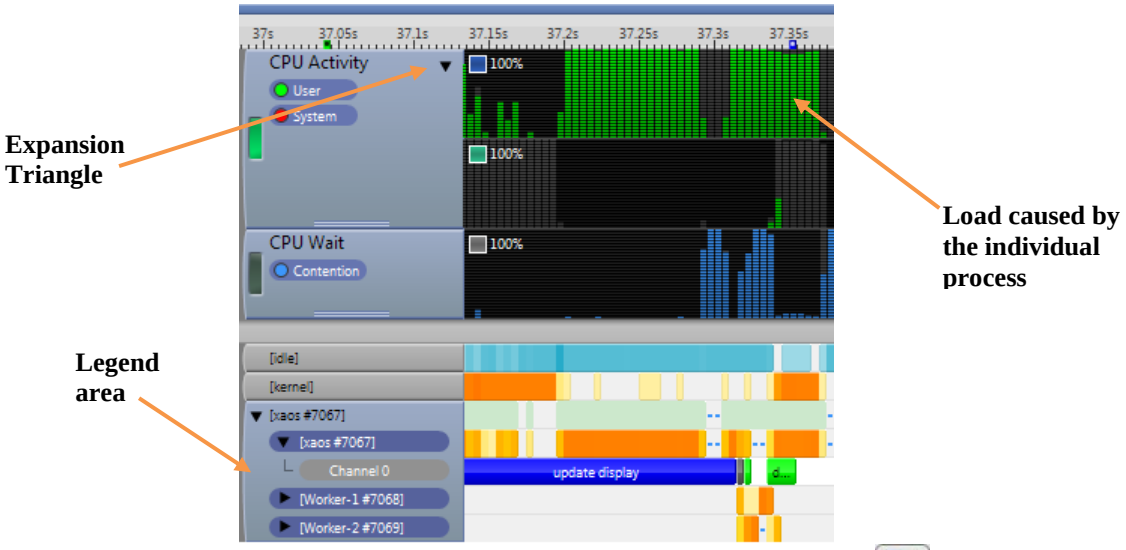
You can see how a particular process contributes to a chart by selecting it.

⇨ Click on the expansion triangle at the left end of the **CPU Activity** chart to show both cores.

⇨ Click on the legend area of the **threads** process.

The threads process will become selected and the **CPU Activity** chart will adjust to show the load caused by just the threads process. You can see the original values of the **CPU Activity** chart as "ghosts".

⇨ Click on the legend area of the threads process again to deselect it.



If the target has multiple cores, you can click the Toggle X-Ray button ( ) to turn on X-Ray mode which changes the coloring of the processes and threads to show which core they executed on. There is a legend at the bottom left of the view to show which core is which color (core 0 1). Hovering over a colored process bar will also show which core was active in a tooltip.

Let's find out more about where the time is being spent:

- ⇨ Zoom into an area where you can see switching between threads, the [kernel] and [idle] similar to the pictures above and click to set the cross-section marker.
- ⇨ Show the Samples HUD (heads up display) by clicking the Samples HUD button ( ) or typing **S**.

The Samples HUD displays a colored histogram of where time was spent in the selected cross-section. If you have supplied debug symbols for the sampled functions then the function names will appear in the Samples HUD. For samples without symbols then the name of the process or shared library will be shown surrounded in square brackets `[]` to indicate that the time was spent inside some unknown function.

- ⇨ Click on the **Timeline** anywhere under the ruler and to the right of the labels to change the selected cross-section being inspected. The entries in the Samples HUD will change accordingly.

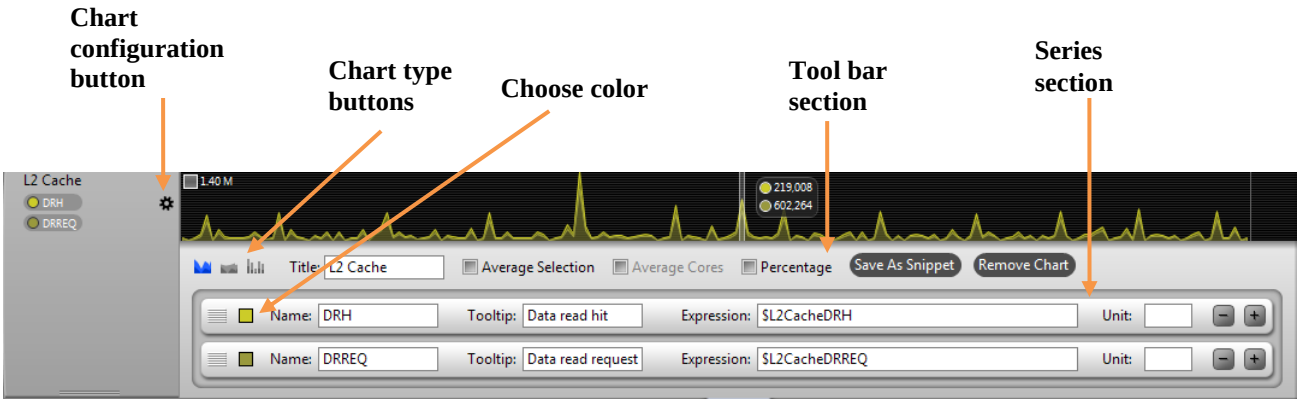
The initial width of the cross-section is determined by the current zoom level. The time of the beginning of the cross-section is shown in the "thumb" of the cross-section bar (2.3s). You can move the cross-section by dragging the thumb right and left and by typing the right and left arrow keys. You can grow the cross-section by dragging the right or left edge of the cross-section thumb. The duration of the cross-section is shown in square brackets (2.5s [0.7s]).

At the top of the **Timeline** view, above the time ruler, is the caliper range ( ). You can set the caliper range by dragging either end, or by right-clicking and choosing **Set left caliper**, **Set right caliper** or **Reset calipers**. All of the samples outside the caliper range are ignored in all of the other views. You can also reset the calipers to include all of the captured data by clicking the Reset calipers button ( ) at the top of the view.

Configuring Charts

You can control many aspects of the display of the charts in the **Timeline** view by using each chart's configuration panel. You can control which counters are displayed and how they are grouped. This panel is used to change all aspects of the chart from color to the counters that the chart uses, giving the option to display the data in a more convenient way.

⇨ Click on the chart configuration button ( ) to open the chart configuration panel for L2 Cache

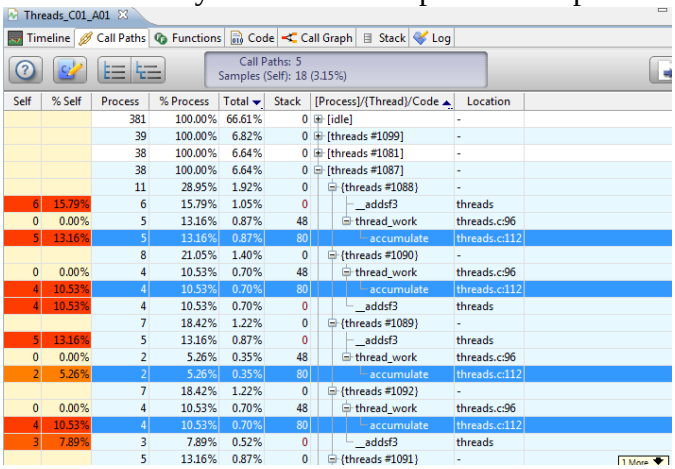


You can enter complex expressions into the expressions view. For more information press **F1** and goto **Chart configuration series options**.

Call Paths view

⇨ In the Samples HUD in the **Timeline** view, you can right click on a function name to bring up navigation options. Find a cross-section where the **accumulate** function is listed in the Samples HUD, then click on the function name to take you that function in the **Call Paths** view.

The **Call Paths** view displays the call paths taken by an application that were sampled. They are listed per process in the view. If you followed the previous steps it should look similar to the screenshot below.

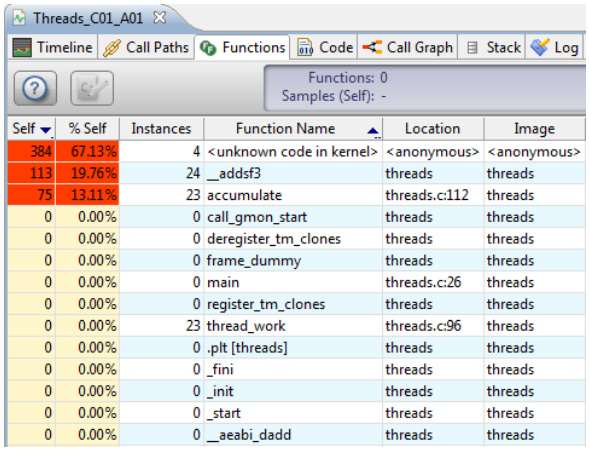


Functions view

The **Functions** view displays a list of all the functions that were sampled during the profile session. The functions view is the quickest way to find which functions are your hot functions. If you've set the caliper range in the **Timeline** view then only the samples within the callipers will be reflected in the **Functions** view.

⇨ Select the **Functions** tab at the top of your report.

⇨ You can **Shift-** or **Ctrl-click** to select multiple functions to see a total count of the self time in the **Totals** panel in the top right of the view.



The **Instances** column shows the number of times a function appears in separate call paths.

Next let's go investigate the code inside one of our hot functions;

⇨ Right click on the `accumulate` function and choose **Select in Code View** from the context menu.

Code view

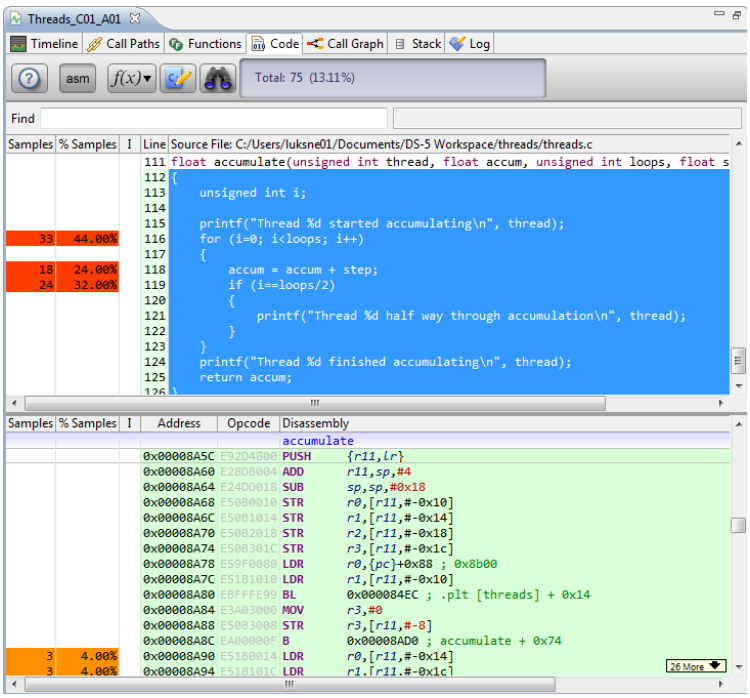
Now the **`accumulate`** function will be highlighted in blue in the **Code** view.

(If the source file has not been found by default you can locate it by pressing the () button or the "Click here to locate it." link to browse the file system. You can use the displayed path name to help you locate the file. This may happen if you used a pre-built copy of threads instead of building it on your host or if the source files are in a different location now compared to when you built the application.)

Lines in the source code where many samples were taken are highlighted in red. Lines of code that don't have any percentages next to were not sampled at all in this profiling run. Profiling for longer periods of time or at a higher sampling rate may give a better resolution in the source code.

You can also open up the disassembly for the code you are viewing.

⇨ Click the Toggle Asm button () in the top right hand corner of the report. You can select a line or lines of source code and the corresponding disassembly lines will be highlighted in the bottom. Vice versa, you can also make a selection in the disassembly pane and the source above will be highlighted, along with the other disassembly lines for that source line.



Notice the **26 More** indicator in the screenshot. It indicates that there are two more disassembly instructions highlighted above. It's useful when compiler has moved the code out of order. You can click on the indicator to scroll them into view.

Call Graph view

The **Call Graph** view is a graphical display showing which functions called each other. Hover above a function name to see the number of samples taken on that function.

⇨ Use the mini-map in the bottom left hand corner to scroll around the map. If the mini-map is in your way you can disable it by selecting () in the top left corner.

⇨ Enable **Uncalled** functions by clicking ().These functions were not in the call chain of any sample.

⇨ Enable **System** functions by clicking (). This displays runtime library functions and shared libraries without symbols.

Stack view

The **Stack** view is designed to give the user an idea of the amount of maximum stack usage in a particular thread. In this view you can also see and sort the stack usage of all the functions used by the application.

Log view and Annotations

The **Log** view displays a list of all the text and visual annotations that were generated by the application along with a time stamp for each and the time delta from the previous entry. You can filter the list on the various columns using regular expressions. When you select an entry for a visual annotation the image is displayed. For more information see the Streamline Documentation by pressing **F1**.

Appendix A: Setup

Host Setup

DS-5 can be used on Windows and Linux hosts. If your host has not already been setup for you then you will need to:

- Download and install DS-5 by following the **Download Now** link from http://ds.arm.com/vybrid/vybrid-tower-starter-kit.
- Obtain and install license by following the instructions in **Licence with Activation Code** section on http://ds.arm.com/vybrid/vybrid-tower-starter-kit. Alternatively you can generate 30 day evaluation license from within DS-5 Eclipse Help Menu **ARM License Manager ... -> Add License ... -> Generate 30-day evaluation license.**
- To follow the workshop you will also need a WebGL enabled browser, for example Google Chrome.
- Adjust the host's networking so that it can communicate with the target. TCP/IP (for example Ethernet) is used to communicate with the target when doing application level debug and Streamline profiling. The rest of this workshop assumes that the host's IP address is **169.254.0.1** and that the target's IP address is **169.254.0.100**. If your addresses are different you will need to make the appropriate adjustments to the instructions in this workshop.

Target Requirements

To run the workshop you'll need to build a software stack from Timesys (See Appendix B: Target software stack setup on page 30).

The hardware requirements to run the demo and the workshop are listed below

- TWR-VF65SG10 Tower Board (power / serial cables)
- TWR SER or TWR SER2
- TWR-LCD-RGB (could probably get away without it)
- Ethernet cable

Importing projects

If you already have an Eclipse project with the same name, for example if you've already imported an older version, you won't be able to import it again. You can either rename or delete the existing Eclipse projects if you want to import them again. If you delete them, you should check the **Delete project contents on disk** checkbox. If you have deleted the Eclipse project, but the project folder is still in the workspace directory, for example if you didn't check **Delete project contents on disk**, then you will need to delete the project folder "by hand" before you can import it again.

Import threads project

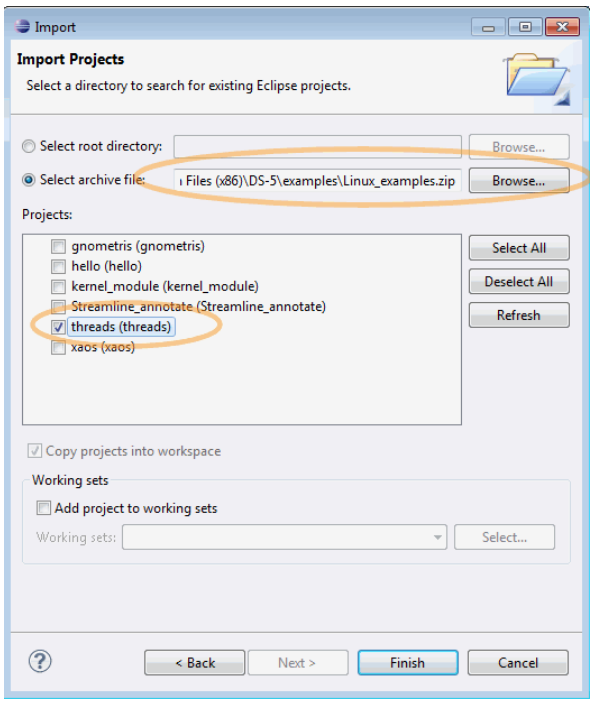
Now import the **threads** project which is in a different `.zip` file:

⇨ Choose **File > Import... > General > Existing Projects into Workspace > Next >**

⇨ Choose **Select archive file** and use the **Browse...** button to select `C:\Program Files\DS-5\examples\Linux_examples.zip`

⇨ Click the **Deselect All** button; then check only **gnometris**. If you're doing the Streamline section of the workshop, you can also check **xaos** to import it now or wait until later.

⇨ Click the **Finish** button.



Importing and Build MQX Projects

Freescale MQX RTOS (see Freescale MQX RTOS 4.0.1-beta3 on the timesys website) includes project build settings for DS-5. Follow the instructions below to import and build the accelerometer example application.

- Choose **File > Import... > General > Existing Projects into Workspace > Next >**
- Choose **Select root directory** and use the **Browse...** button to select the path to your MQX installation for example `C:\FreescaleMQXRTOS4.0.1-Beta3`
- Click the **Deselect All** button; then check only the following projects
    a. accelerometer_example_twrvf65gs10_m4
    b. bsp_twrvf65gs10_m4
    c. mcc_twrvf65gs10_m4
    d. psp_twrvf65gs10_m4
- Make sure that the **Copy projects into workspace** is ***not selected***
- Click the **Finish** button

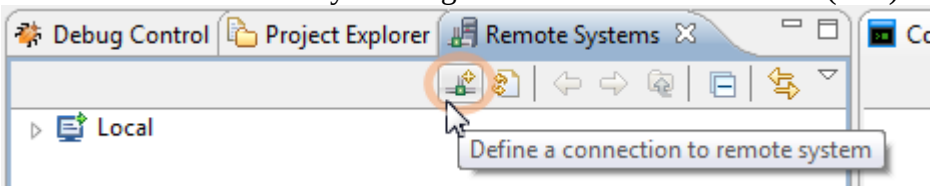
In the Project Explorer

- Right click on the each project ***(in this order bsp, psp, mcc, accelerometer)*** and choose **Build Project**

Creating a Target Connection

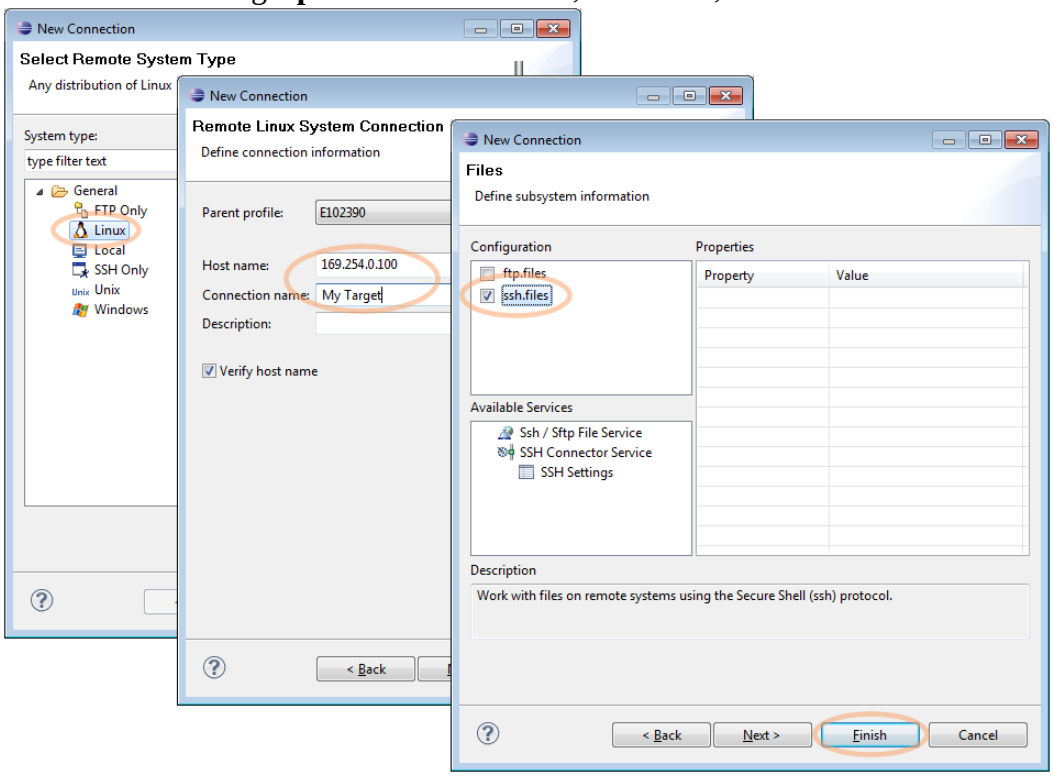
We will establish a connection to the target using Eclipse's Remote Systems Explorer (RSE) so that we can browse its file system and create a terminal connection.

- If the **Remote Systems** view is not open, you can open it by choosing **Window > Show View > Other... > Remote Systems > Remote Systems** in any perspective.
- Click on the tab of the **Remote Systems** view to bring it to the front.
- Create a new connection by clicking the New Connection button ( )



- Choose **General > Linux >** "**Next >**"; put the target's IP address, `169.254.0.100` in the **Host name** field and use **My Target** as the **Connection name**; click "**Next >**". (If target is using a different IP address, for example, if your target is connected to a network with a DHCP server, you'll need to determine its IP address by using the serial console or a monitor, keyboard and mouse.)

⇨ Check **ssh.files** then click the **Finish** button. If you click "**Next >**" instead of **Finish**, you want the rest of the default settings: **processes.shell.linux**, **ssh.shells**, and **ssh.terminals**.



The username and password to the board is *root*. Now you can resume where you left off.

Appendix B: Timesys software stack setup on Vybrid

The Linux kernel and MQX software can be downloaded from the Timesys website. You'll need an account to gain access to the factory installer. In order to follow the instructions in this workbook you'll need to build the Linux kernel with **`CONFIG_DEBUG_KERNEL=y and CONFIG_DEBUG_INFO=y`**. Additionally we'll install and enable a few extra components into the default recipe to demonstrate more features of the tool suite.

Building the Software stack

1) Download the Timesys SDK installer (PRO customers only), filename `twr_vf600-factory-installer.sh`, from https://linuxlink.timesys.com/webshare/2/index.pt/timesys/factory/Starting_Point_Build/16/output/

2) Install the factory onto your own PC. (See Host requirements https://linuxlink.timesys.com/docs/wiki/factory/FactoryGSG ; latest releases of Ubuntu and Fedora is supported).

    a. Install the factory by running "`sh ./twr_vf600-factory-installer.sh`"

    b. Run "`make checksystem`" from the root of the installation to scan for the required software

    c. Run "`make menuconfig`"

        i. Select and enable `Target Software -> Software Packages -> Development -> Miscellaneous -> gator`

        ii. Select and enable `Target Software -> Kernel -> Include uncompressed kernel image within the RFS`

        iii. Select and enable `Target Software -> Software Packages -> Networking -> SSH Servers -> openssh`

iv. Disable dropbear `Target Software -> Software Packages -> Networking -> SSH Servers -> dropbear`

v. Exit and save

d. Run "`make kernel-menuconfig`"

i. Select and enable `Kernel hacking -> Kernel debugging -> Compile the kernel with debug information`

ii. Exit and save

e. Run "`make`" in the root directory to build the source

3) Note ... the Streamline gator version is still at 5.9 at the time of writing. It is being updated and those changes should reflect in due time.

4) Package up the following to distribute or use with the workshop

a. vmlinux `[ build_armv7l-timesys-linux-gnueabi/linux-3.0/vmlinux ]`

b. Kernel source `[ build_armv7l-timesys-linux-gnueabi/packages/kernel-source-3.0-armv7l-timesys-linux-gnueabi.tgz ]`

c. Mcc driver source and debug symbols `[build_armv7l-timesys-linux-gnueabi/mcc-kmod-1.02/ ]`

Install images to SDCard & setup boot arguments

1) Follow these instructions to prepare the SD Card partitions and install the binaries https://linuxlink.timesys.com/docs/gsg/twr_vf600#SECTION00041000000000000000

2) Connect your serial cable (BAUD 115200), boot the board, stop in u-boot and set the boot arguments

a. Run `"setenv bootargs mem=128M console=ttymxc1,115200 root=/dev/mmcblk0p2 rw rootwait jtag=on ip=169.254.0.100"`

i. Note you may need to modify the console settings above for your setup / revision of the board.

b. Run "`saveenv`" to save environment

3) Boot the board

a. If you have a touch screen connected, on first boot you'll need to calibrate the screen by touching the cross hairs

b. Once booted run `"passwd root"` and set the password to `"root"`

Appendix C: Debug on Fixed Virtual Platforms (FVP)

ARM DS-5 also ships with a set of Fixed Virtual Platforms (FVP's – previously known as RTSM's) which are based on ARM Versatile express development boards The FVP's can be used for early development when real hardware is not yet available, or when real hardware in short supply. Using the DS-5 debug environment you can seamlessly switch from a virtual to a physical debug environment when your hardware becomes available. Note: to use this feature you require a DS-5 Professional edition. If you don't have this you can obtain a 30-day evaluation license from the DS-5 Eclipse Help Menu **ARM License Manager ... -> Add License ... -> Generate 30-day evaluation license.**

As an example lets debug the threads application we used in Debug your first Linux example over Ethernet on a Cortex-A8 FVP that has been setup to boot Linux. You can also debug the Linux kernel and any other bare metal code on the FVP, but we'll leave that for another day.

⇨ If you have not already done so, follow /Importing Threads on page 28 to import it into your workspace. The project contains a pre-setup debug configuration to launch the FVP and to start a debug connection to the threads application.

⇨ Select **Run -> Debug Configurations**

⇨ Select **DS-5 Debugger -> threads-RTSM-example**

⇨ Click **Debug** to launch the debug session

The FVP will launch a simulated LCD screen and Telnet serial terminal which will display the boot messages. Once the target has booted DS-5 will copy the threads binary to the target (pre-setup in the debug configuration) and the program will be stopped at `main()`. You can now follow the same debug instructions as you did in Debug your first Linux example over Ethernet on page 4