Development Studio 5 (DS-5)

Development Tools for ARM Systems

Quick Start Guide

The **ARM® Development Studio 5 (DS-5™)** toolchain is a complete suite of software development tools for ARM processor-based systems.

DS-5 accelerates software development by providing an easy to use, robust, well integrated, and professionally maintained toolchain.

DS-5 covers all the development stages of ARM processor-based products, from platform bring-up to application profiling, while including a number of ARM Linux and Android™ specific features.

This Quick Start Guide is intended to help you through your first steps with DS-5, from installation to launching and using each of its component.

What's New!

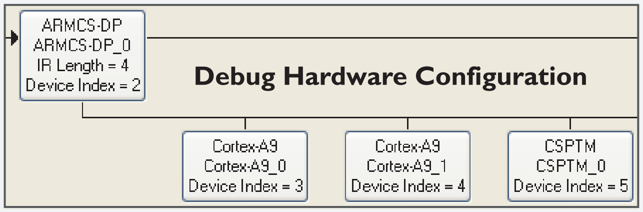
Extended Target Connection

DS-5 Debugger supports the connection to a variety of target types, from programmer's view models and RTL running in EDA environments, to physical implementations on FPGAs and ASICs.

For development of Linux and Android native applications and libraries, DS-5 also supports the connection to the gdbserver debug agent over a low-cost Ethernet or USB interface.

	Interface	Target Connection
ARM Fast Models	CADI	Virtual
RTL Simulators	VSTREAM	Virtual
Hardware Emulators	VSTREAM	Virtual
FPGAs and ASICs	DSTREAM	JTAG or Serial-Wire Debug
Linux applications	TCP/IP	Ethernet or Serial
Android applications	ADB	USB

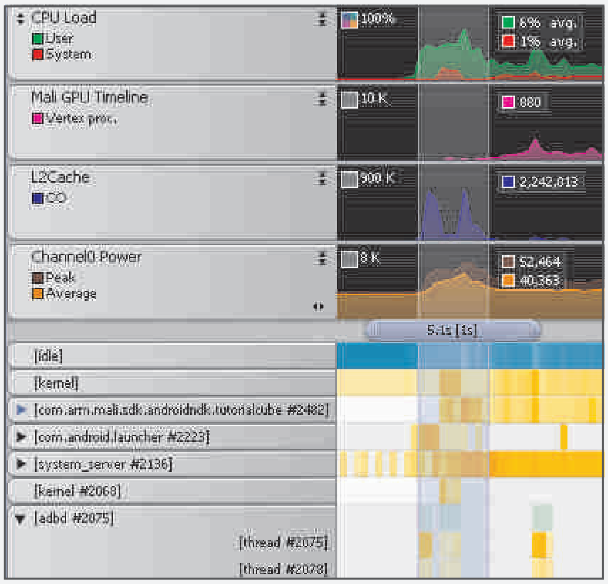
DS-5 includes target configuration utilities to facilitate the bring-up of new SoCs and boards, and to configure the CoreSight™ debug and trace fabric of the target.



GPU Profiling and Energy Analysis

The **ARM Energy Probe** adds energy analysis capability to the ARM Streamline™ performance analyzer, which enables developers to correlate the target's power consumption with software activity and events.

Streamline also supports complete system-level profiling, by merging information from the CPU, graphics processor, system caches and Fabric IP into one single report. This report visualizes the interactions between the different IP blocks.



Downloading DS-5

Download DS-5 and a 30-day free, fully featured, evaluation license from **www.arm.com/ds5**. Alternatively, request a DVD from your local ARM office or tools distributor.

DS-5 is regularly updated with new functionality. In order to use its latest features, just install the latest version available.

When you purchase DS-5 you are entitled to technical support and access to free updates for one year.

Running DS-5

Most of the functionality in DS-5 can be accessed from its Eclipse-based graphical environment.

On Windows hosts, launch **Eclipse for DS-5** from the **Start Menu ▶ Programs ▶ ARM DS-5**.

Installations on Linux hosts include a script to install and uninstall application menus. Alternatively, execute **eclipse** from **<install_directory>\bin**.

When launched, Eclipse prompts you to specify the location of a **Workspace**, where your Eclipse settings and projects will be stored. We recommend that you create one or more new directories to be used as DS-5 workspaces.

Welcome Screen and Licensing

At any time you can open the DS-5 welcome screen by selecting **Help ▶ Welcome**. This screen provides links to examples, docs, hints and tips, and support resources.

In order to license DS-5 follow the instructions on **Help ▶ ARM License Manager...** The license manager provides assistance to obtain and install a DS-5 license key associated with your host machine. Both node-locked and floating licenses are available.

Click on the **Go to the Workbench** link to close the welcome screen and start using DS-5.

Environment Variables

The DS-5 installer does not modify any environment variables in your system.

DS-5 only requires that the **path** environment variable points to the **<install_directory>\bin** directory. This is done automatically if you launch the tools from the **Eclipse for DS-5** or the **DS-5 Command Prompt** items in the Windows Start Menu.

Running the tools from a Linux console requires that you manually set the path environment variable.

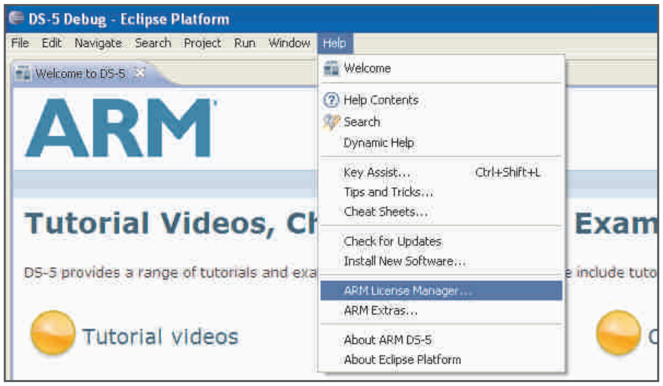
System Requirements

The minimum computer specification to run DS-5 is:

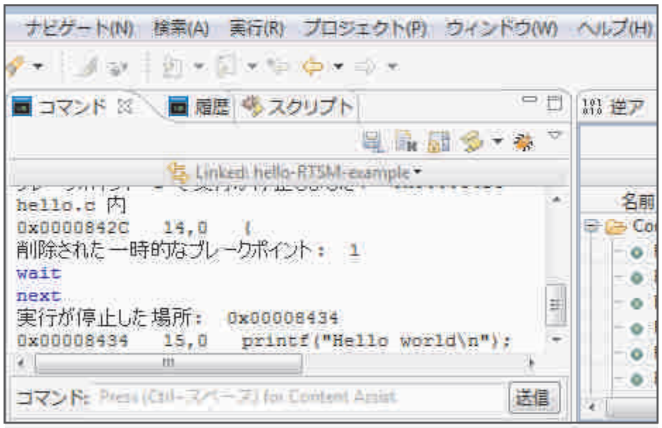
- 2 GHz dual-core host processor
- 2 GB of RAM (4 GB recommended)
- 2 GB of hard disk space.

DS-5 supports the following x86 host platforms:

- Windows XP Professional service pack 3 (32-bit)
- Windows 7 Professional and Enterprise (32/64-bit)
- Red Hat Enterprise Linux 5 Desktop and Workstation (32/64-bit)
- Ubuntu Desktop Edition 10.04 LTS (32-bit).



The DS-5 welcome screen links to useful information such as video tutorials, step-by-step guides, examples, and documentation.


ナビゲート(N) 検索(A) 実行(R) プロジェクト(P) ウィンドウ(W) ヘルプ(H)
コマンド
履歴
スクリプト
逆ア
Linked: hello-RTSM-example
名前
コマンド:
Press (Ctrl-スペース) for Content Assist
送信

```
hello.c 内
0x0000842C    14,0    {
削除された一時的なブレークポイント:  1
wait
next
実行が停止した場所:   0x00008434
0x00008434    15,0    printf("Hello world\n");
```

DS-5 includes support for Japanese localisation, both in its graphical user interface and command line.

Eclipse Perspectives and Views

A typical Eclipse workbench window contains one or more perspectives, each of them including a set of related views, editors, menus, and toolbars.

The main Eclipse perspectives in DS-5 are:

- **C/C++**: used to import and manage software projects, and to edit source files
- **DS-5 Debug**: graphical interface for the DS-5 Debugger.

DS-5 also includes other perspectives, such as **PyDev** for coding debugger Jython scripts.

To open an Eclipse view, go to **Window** ▶ **Show View**. Change the position of any view by left-clicking on its tab and dragging it. The new location can be adjacent to existing views or in a tabbed group.

Detach (undock) the view to drag it to a second monitor, or right-click on its name to add it to the **FastView** bar.

Import Example Projects

From the C/C++ perspective, select **File** ▶ **Import...** ▶ **General** ▶ **Existing Projects into Workspace**.

Activate **Select archive file** and choose an example zipfile from **<install_directory>\examples\example.zip**.

The examples are decompressed into your workbench directory and appear in the Project Explorer view.

Create and Build Projects

Right-click on the Project Explorer and select **New** ▶ **C or C++ Project**.

Select one of the six project templates in the DS-5 project wizard: bare-metal executable or library, Linux executable, shared library or static library, and makefile project. This creates a new project with working pre-defined compiler settings.

Template-based source and header files can easily be added to a project. Right click on the project in the Project Explorer view and select **New** ▶ **Source File.**

The DS-5 IDE includes powerful features to assist coding of C/C++ and assembler files. These include syntax highlighting, code auto-completion, and quick-jump to declaration of variables and functions (**F3** short-cut) .

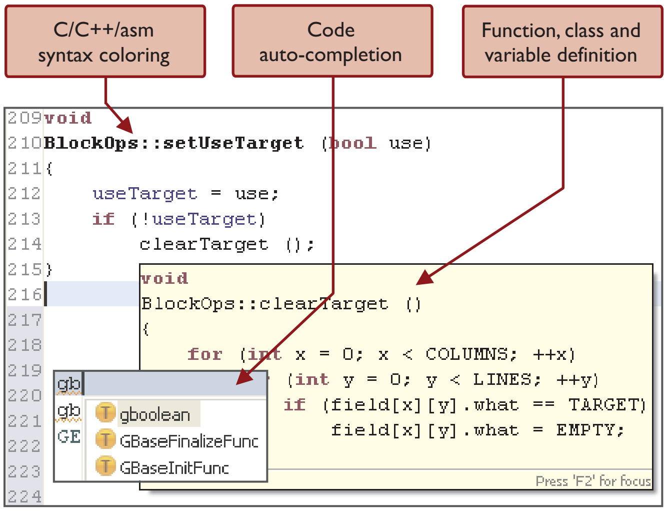
Find-in-files tools are grouped under the **Search** menu.

To make or build a project, select **Project** ▶ **Build**.

DS-5 Example Projects

DS-5 includes a range of examples to get you started:

- **Startup** examples include reset, MMU and cache initialization code for ARM and ARM Cortex™ processors.
- **Fireworks** is an example bare-metal application ported to the RTSMs, Beagle and Panda boards.
- **Hello World** bare-metal examples are ported to many of the boards in the configuration database.
- **Distribution** is a complete Linux distribution for the Beagle board and the Cortex-A8 Real-Time System Model, to enable driver and application development.
- **Gnometris** is an open source Linux application similar to the game Tetris.
- **Example_library** demonstrates how to create a simple shared library in C for ARM Linux.
- **Threads** is a simple multi-threaded Linux application that illustrates how the tools handle multiple threads.
- **Xaos** is an example fractal application, which comes with a pre-generated Streamline report.
- **Kernel_module** illustrates the kernel awareness features in DS-5 Debugger. Note that it needs to be built on a Linux host against your Linux kernel.



```
void
BlockOps::setUseTarget (bool use)
{
    useTarget = use;
    if (!useTarget)
        clearTarget ();
}
```

```
void
BlockOps::clearTarget ()
{
    for (int x = 0; x < COLUMNS; ++x)
        (int y = 0; y < LINES; ++y)
        if (field[x][y].what == TARGET)
            field[x][y].what = EMPTY;
```

Source code editing features in the DS-5 Eclipse IDE

Target Connection: Bare-Metal and Driver Debug

DSTREAM

The ARM **DSTREAM**^{TM} Debug and Trace Unit provides DS-5 Debugger with high-performance low-level debug and trace connection to hardware targets.

DSTREAM enables DS-5 Debugger to bring up a platform and debug bare metal software in stop-mode, with full access to the processor and system resources.

Thanks to its large buffer DSTREAM supports long-term off-chip trace. This enables the debug of complex, time-related software bugs, and performance optimization of critical code.

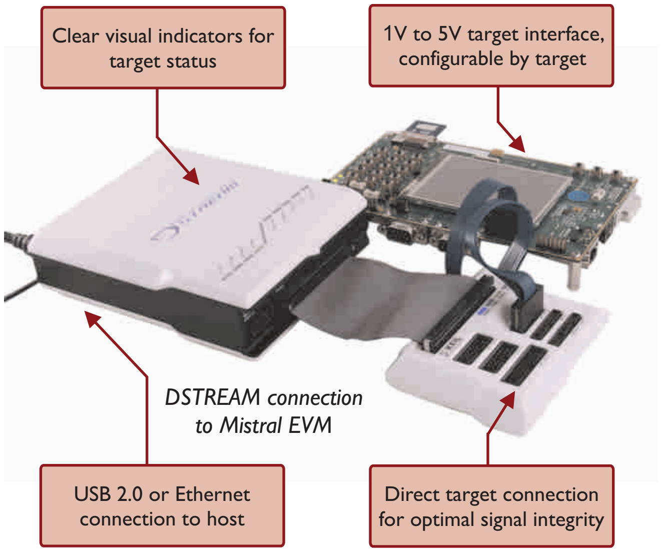
	Specification
Debug Interface	JTAG and Serial-Wire Debug
JTAG Speed	60 MHz
Download	2.5 MB/s
Trace Buffer	4 GB
Trace Speed	600 Mbps/pin (16-bit)
Trace Frequency	300 MHz DDR

DSTREAM features.

DS-5 Configuration Database

DS-5 provides **pre-configured JTAG and trace** for catalogue devices. Just choose a device on the Debug Configurations dialog to connect the debugger to one of these targets.

The default DS-5 configuration database is stored on **<install_directory>\sw\debugger\configdb**. However, you can add custom configuration databases by selecting **Window ▶ Preferences ▶ DS-5 ▶ Target Database.**


Clear visual indicators for target status
1V to 5V target interface, configurable by target
USB 2.0 or Ethernet connection to host
Direct target connection for optimal signal integrity

DSTREAM connection to Mistral EVM

Debugging Custom Devices

The tools required to bring-up new SoCs and boards, and add them to the configuration database are provided in the DS-5 installation:

- **CSAT**: This command line-based tool enables manual control of the registers of a CoreSight Debug Access Port and script access to the IP blocks connected to it.
- **Debug Hardware Configuration:** This graphical utility reads the contents of a CoreSight ROM Table to auto-configure the target connection. It also enables manual configuration if the table is incorrectly set up.
- **ConfigDB Import:** Use it to import the new debug hardware configuration into a configuration database. This utility outputs **DTSL Python scripts** (.py), which can be used to manually modify the configuration of each IP block with a simple scripting language.

Low-Level Debug Features

DS-5 Debugger implements support for the latest ARM processors and technology:

- **Multi-core debug:** DS-5 supports AMP and SMP configurations, as well as big.LITTLE systems
- **Linux kernel awareness:** visualize and access kernel threads and modules on single and multi-core targets
- **Hypervisor debug:** access memory in hypervisor mode and set virtual machine-dependent breakpoints
- **Direct memory access:** using the AHB-AP port of the CoreSight DAP for fast memory download and debug of running systems.

Target Connection: Linux and Android Application Debug

Connecting to the Target

All you need to debug and analyze Linux applications on an ARM processor-based target is a **TCP/IP** connection between the target and the host computer.

Android middleware and native application development normally uses an **Android™ Debug Bridge (ADB)** connection over USB.

DS-5 Debugger supports both connection types, and can use them to automatically download applications into the target file system, launch the **gdbserver** debug agent on the target and connect to it.

DS-5 requires gdbserver v6.8 or later. The installation includes gdbserver v7.0, which enables debugging NEON™ code and multi-threaded Android libraries.

Linux Target File System

DS-5 integrates a **Remote System Explorer (RSE)** to access the Linux file system on hardware targets that support the Secure Shell (SSH). The target must include a secure shell daemon (**sshd**) and **sftp-server**.

RSE enables the following functionality:

- **SFTP Files View**: Access the target's Linux file system using an FTP connection. Navigate, copy, paste, drag and drop, and edit files on the target
- **Shell Processes View**: Displays a list of current processes running on the target
- **SSH Shell/Terminal View**: Opens a shell or a terminal window connected to the target in order to execute Linux commands - for example to run an application
- **Automated software download**: DS-5 Debugger uses RSE connections to automate the download of images to the target, executing gdbserver, and connect to it.

Configure Remote System Explorer

Open RSE by selecting **Window ▶ Show View ▶ Other... ▶ Remote Systems ▶ Remote Systems**.

Click on the **New Connection** button . Select Linux and click Next.

Fill the Host name box with the IP address of the target system, or its network name if it is registered on a DNS server. Define a name for the connection and click Next.

In the next configuration steps choose the Secure Shell (ssh) option, then click Finish.

Real-Time System Models

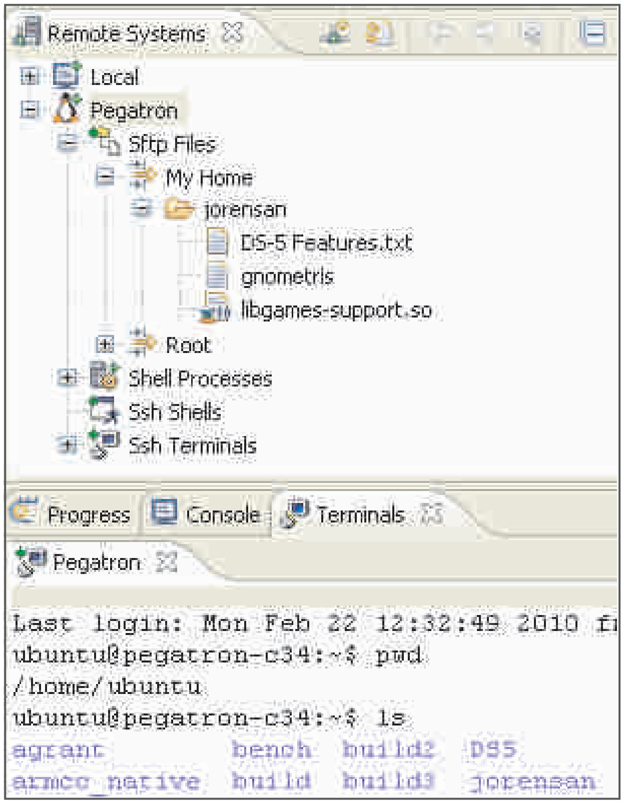
DS-5 includes Real-Time System Models (**RTSMs**) of Cortex-A8 and Cortex-A9 processor-based SoCs.

An RTSM is a fast simulation model of a complete SoC with a processor, system memory, and peripherals such as keyboard, mouse, UARTs, Ethernet, and LCD.

RTSMs enable ARM bare metal and Linux software development without any hardware, and deliver simulation speeds of over 250MHz on a typical desktop.

DS-5 includes an example **ARM Linux distribution** ported to the Cortex-A8 RTSM. When launched, the model automatically boots Linux and enters a state in which you can load and debug your applications.

RTSMs are configured to mount a **Virtual File System (VFS)**. The processor running in simulation can access files in the host computer's file system as if they were part of its ARM Linux file system.



```
Last login: Mon Feb 22 12:32:49 2010 fr
ubuntu@pegatron-c34:~$ pwd
/home/ubuntu
ubuntu@pegatron-c34:~$ ls
agrant          bench   build2   DS5
armco_native    build   build3   jorensan
```

Remote System Explorer SFTP and Terminal views.

Launching DS-5 Debugger

Click on the down arrow of the Debug button on the menu bar, or right-click on the Debug Control View to open the **Debug Configurations** dialog.

Right-click on **DS-5 Debugger** and select **New**, or browse existing debug configurations. If you have imported the DS-5 examples into the Workbench, the Debug Configurations Dialog is populated with pre-configured target connections.

Existing debug configurations can be launched by double clicking on the Debug Control View.

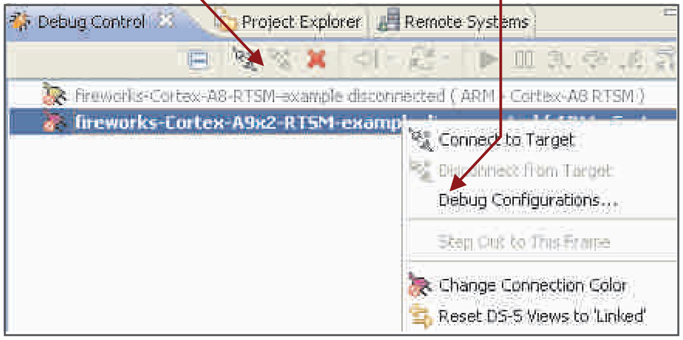
Adding Custom Plug-ins

Eclipse plug-ins can connect to the DS-5 Debugger using a Java-level API. Its target application is RTOS awareness plug-ins, but could be used to visualize graphically any target data structures. The API enables:

- Read memory from the target
- Decode the address of symbols (e.g. variables)
- Add custom tabs to the Debug Launch dialog
- Get notification of the target's status (e.g. stopped).

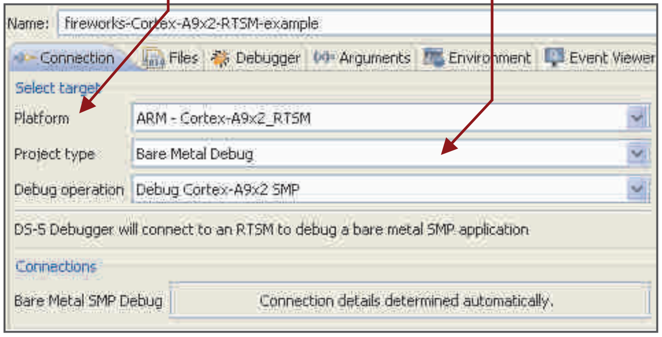
Use the Debug Control View to manage your debug configurations

Open the debugger launcher to configure the connection's properties



Select a target from the configuration database or a gdbserver connection

The project types available depend on the type of target selected



Name: fireworks-Cortex-A9x2-RTSM-example

Select target

Platform: ARM - Cortex-A9x2_RTSM

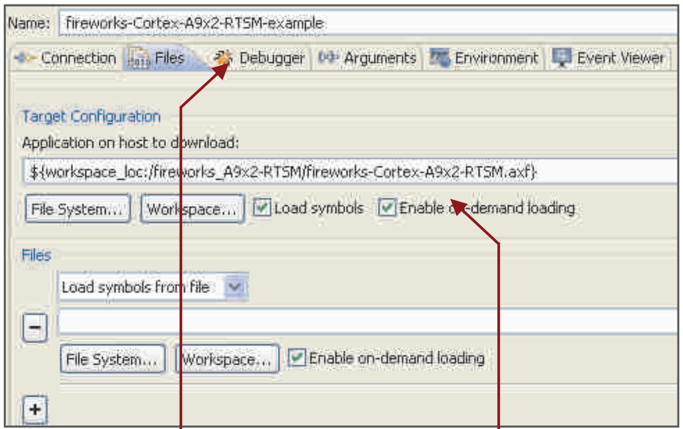
Project type: Bare Metal Debug

Debug operation: Debug Cortex-A9x2 SMP

DS-5 Debugger will connect to an RTSM to debug a bare metal SMP application

Connections

Bare Metal SMP Debug: Connection details determined automatically.



Name: fireworks-Cortex-A9x2-RTSM-example

Target Configuration

Application on host to download:

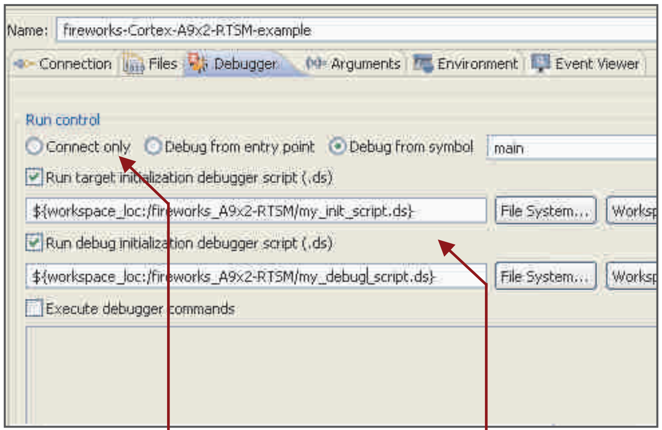
${workspace_loc:/fireworks_A9x2-RTSM/fireworks-Cortex-A9x2-RTSM.axf}

[✓] Load symbols [✓] Enable on-demand loading

Files

Load symbols from file

[✓] Enable on-demand loading



Name: fireworks-Cortex-A9x2-RTSM-example

Run control

Connect only Debug from entry point [✓ Debug from symbol] main

[✓] Run target initialization debugger script (.ds)

${workspace_loc:/fireworks_A9x2-RTSM/my_init_script.ds}

[✓] Run debug initialization debugger script (.ds)

${workspace_loc:/fireworks_A9x2-RTSM/my_debug_script.ds}

[ ] Execute debugger commands

Use the other tabs in the dialog to configure connection options

Load an application's symbols to enable source-level debug

Use this option to attach the debugger without stopping the target

Use debugger scripts to initialize the target and the debug session

DS-5 Debug Perspective

DS-5 includes a fully featured graphical C/C++ debugger. Open it by launching the debugger from the Debug Configurations dialog, or by clicking **Window ▶ Open Perspective ▶ Other... ▶ DS-5 Debug**.

Alternatively, the debugger can be used from the host's command line interface, or from a **Jython script**.

Each debug view is self-contained, so there is no need to configure any options using the Eclipse main menu. Note that some of the debugger views, for example the Trace view, are only enabled for JTAG target connections.

Take time to explore the toolbars of each of the views. Press the down arrows ▽ to see more buttons and options.

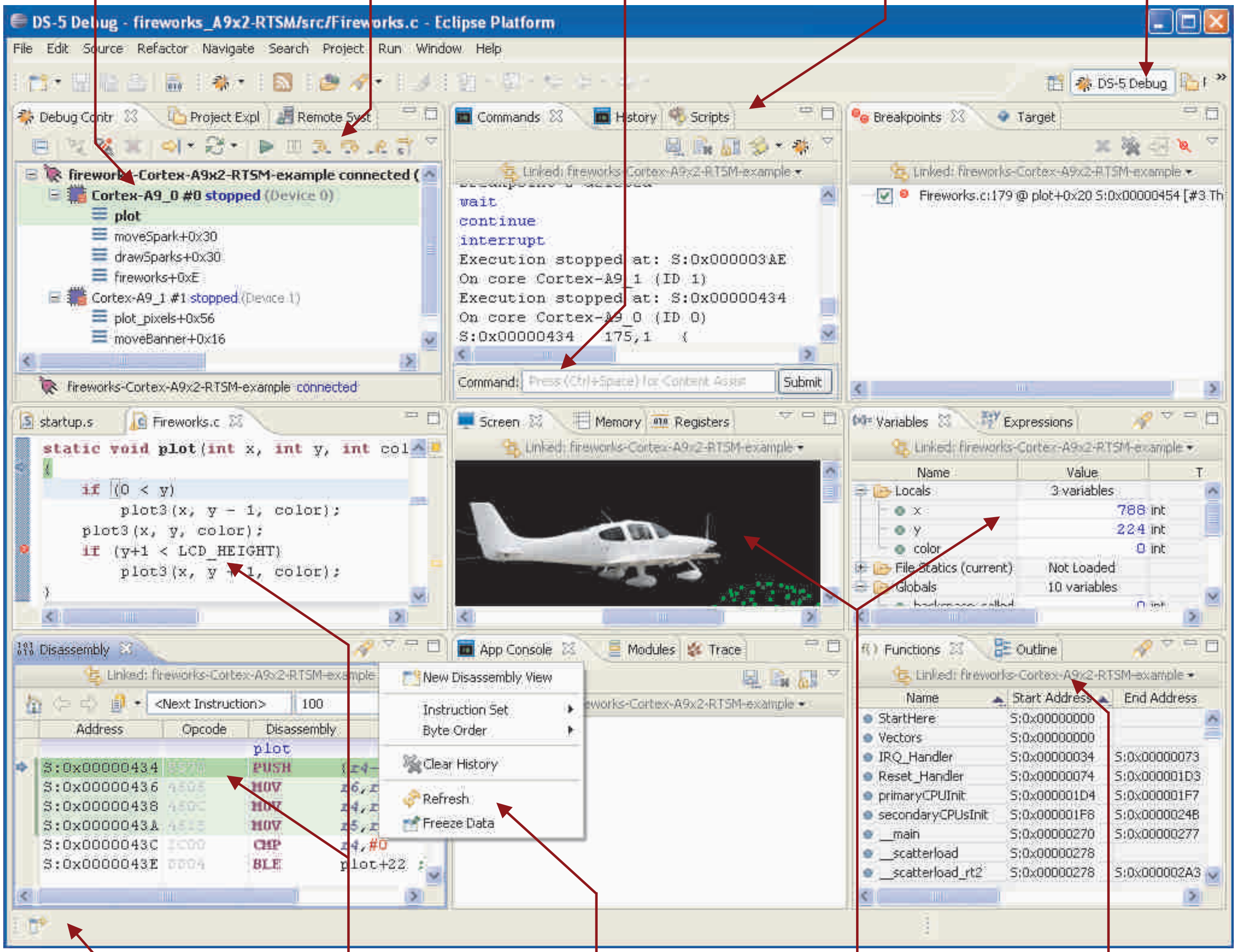
Click to focus the system views on a connection, core, thread or stack frame

Control target execution with source and assembly level stepping

Type GDB-style commands with auto-completion and on-line help

Drag & drop lines from history view to scripts view to create a script

Switch between the C/C++ and DS-5 Debug perspectives



```
fireworks-Cortex-A9x2-RTSM-example connected (
  Cortex-A9_0 #0 stopped (Device 0)
    plot
    moveSpark+0x30
    drawSparks+0x30
    fireworks+0xE
  Cortex-A9_1 #1 stopped (Device 1)
    plot_pixels+0x56
    moveBanner+0x16
fireworks-Cortex-A9x2-RTSM-example connected
```

```
wait
continue
interrupt
Execution stopped at: S:0x000003AE
On core Cortex-A9_1 (ID 1)
Execution stopped at: S:0x00000434
On core Cortex-A9_0 (ID 0)
S:0x00000434    175,1    {
```

```
Fireworks.c:179 @ plot+0x20 S:0x00000454 [#3 Th
```

```
static void plot(int x, int y, int col
{
    if (0 < y)
        plot3(x, y - 1, color);
    plot3(x, y, color);
    if (y+1 < LCD_HEIGHT)
        plot3(x, y + 1, color);
}
```

Name	Value	T
Locals	3 variables	
x	788	int
y	224	int
color	0	int
File Statics (current)	Not Loaded	
Globals	10 variables	

Address	Opcode	Disassembly
		plot
S:0x00000434	[illegible]	PUSH {r4-
S:0x00000436	[illegible]	MOV r6,r
S:0x00000438	[illegible]	MOV r4,r
S:0x0000043A	[illegible]	MOV r5,r
S:0x0000043C	[illegible]	CMP r4,#0
S:0x0000043E	[illegible]	BLE plot+22

Name	Start Address	End Address
StartHere	S:0x00000000	
Vectors	S:0x00000000	
IRQ_Handler	S:0x00000034	S:0x00000073
Reset_Handler	S:0x00000074	S:0x000001D3
primaryCPUInit	S:0x000001D4	S:0x000001F7
secondaryCPUsInit	S:0x000001F8	S:0x0000024B
__main	S:0x00000270	S:0x00000277
__scatterload	S:0x00000278	
__scatterload_rt2	S:0x00000278	S:0x000002A3

Fast View bar for minimizing often-used debug views

Source and assembly code views synchronized with color coding

Click the down arrow to access the sub-menu, and to freeze and replicate views

Screen buffer and system views have multiple formatting options

Link the view to the active connection, or to a fixed connection or processor

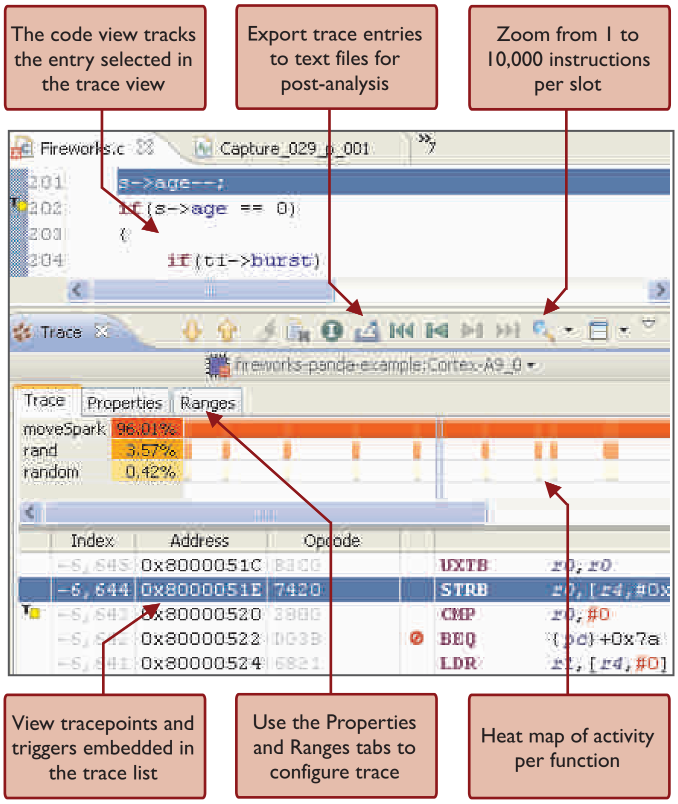
Instruction Trace

Instruction trace enables the non-intrusive debug of random time-related issues, which are difficult to replicate by stepping through the code, and the performance analysis of critical areas of software.

When trace is enabled, the debugger records the last instructions executed by the target. When the execution stops, the trace view is filled with a history of instructions around a trigger point, synchronized with the code view.

Instruction trace requires a JTAG connection to devices featuring an **Embedded Trace Macrocell** (ETM) or **Program Trace Macrocell** (PTM).

The number of instructions recorded is limited by the size of the on-chip **Embedded Trace Buffer** (ETB). If you need a larger buffer, trace off-chip to the 4GB buffer on DSTREAM.


The code view tracks the entry selected in the trace view
Export trace entries to text files for post-analysis
Zoom from 1 to 10,000 instructions per slot
Fireworks.c
Capture_029_p_001
Trace
fireworks-panda-example:Cortex-A9_0
Trace
Properties
Ranges
View tracepoints and triggers embedded in the trace list
Use the Properties and Ranges tabs to configure trace
Heat map of activity per function

```
s->age--;
if(s->age == 0)
{
    if(ti->burst)
```

```
moveSpark  96.01%
rand        3.57%
random      0.42%
```

```
Index     Address     Opcode
-6,645    0x8000051C  B3C0    UXTB   r0,r0
-6,644    0x8000051E  7420    STRB   r0,[r4,#0x
-6,643    0x80000520  2800    CMP    r0,#0
-6,642    0x80000522  D03B    BEQ    {pc}+0x7a
-6,641    0x80000524  6821    LDR    r1,[r4,#0]
```

Trace View.

Using Tracepoints and Filters

By default, DS-5 traces all the instructions executed by the processor into a circular buffer. The contents of the trace buffer are collected, decompressed and displayed when the target stops - for example, when hitting a breakpoint.

Trace start and **trace stop points** enable you to restrict trace to certain areas of the code. This is useful to optimize the analysis of certain functions when tracing to an ETB, or to prevent ETM FIFO overflows when the trace port is slow.

Ranges (also called filters) can be used in conjunction with trace start and trace stop points. They enable you to select an address range outside of which trace is always disabled.

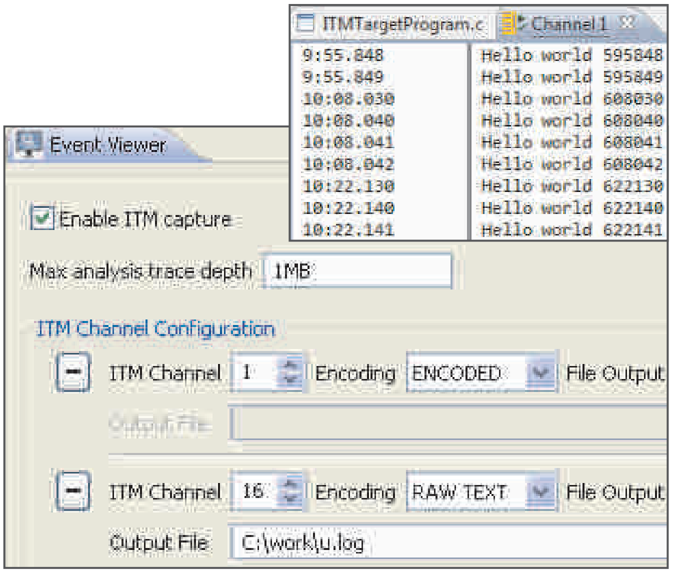
Triggers are highlighted on the trace view, and can be used to identify and search interesting areas in the code. In addition, triggers enable you to configure the tools to collect trace around, before, or after a certain instruction. Just check the **Stop on Trigger** box in the trace view's Properties tab to trace around a trigger point.

System Trace

The **Instrumentation Trace Macrocell** (ITM) and **System Trace Macrocell** (STM) provide high-bandwidth, low latency "printf-style" instrumentation links.

The output of ITM and STM is routed to the on-chip trace bus, and can be collected by the debugger and displayed on its **Event Viewer** or exported to a text file.

The Event Viewer can be configured from the Event Viewer tab of the **Debug Configurations** dialog.



```
9:55.848   Hello world 595848
9:55.849   Hello world 595849
10:08.030  Hello world 608030
10:08.040  Hello world 608040
10:08.041  Hello world 608041
10:08.042  Hello world 608042
10:22.130  Hello world 622130
10:22.140  Hello world 622140
10:22.141  Hello world 622141
```

☑ Enable ITM capture

Max analysis trace depth: 1MB

ITM Channel: 1 | Encoding: ENCODED | File Output

Output File:

ITM Channel: 16 | Encoding: RAW TEXT | File Output

Output File: C:\work\u.log

Event Viewer configuration and output.

Debugger System Views and Target Control

System Views

The **variables view** shows the value of all the application's variables under three groups: locals, file statics, and globals. It also displays each variable's type, size, value, and location (either register or memory).

If you are interested in a defined set of variables or registers, just type them or drag them to an **expressions view**. This view also offers the flexibility to watch any C-like expression.

For bare metal development, use the **register view** to view the processor, co-processor, NEON and peripheral registers with the same format as in the target's documentation.

The **functions view** lists all the functions in the target's application, and can be used to set breakpoints and trace points, or navigate to the source code of a function.

Using the System Views

All the system views use color-coding to highlight content that has recently changed, and most support **drag & drop**. The code view shows variable values by hovering over them.

System views can be **frozen** or **replicated** for comparing data between two points in time. Most views also accept C-like expressions. For example, you can set the value of a register with an expression such as **(*my_ptr) & 0xffffff**.

The way the debugger accesses the memory system (e.g. as bytes, words, or multiple words) is configured via the **memory map**. View the memory map with the **info mem** command, and configure it with the **mem** command.

Breakpoints and Watchpoints

Set **new breakpoints or tracepoints** by double clicking on the left column of the source code or disassembly views.

Set **new watchpoints** (data breakpoints) by right-clicking on a cell of the memory view or a global variable in the variable or expression view.

Edit the properties of breakpoints to assign:

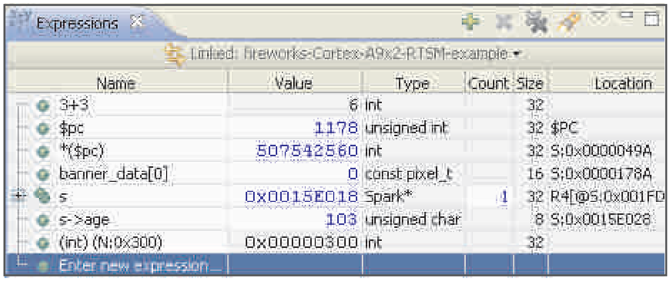
- **Condition:** the breakpoint only stops the execution if the condition (a C-like expression) is evaluated true
- **Ignore count**: number of times the breakpoint is hit until the application stops
- **Run script**: the debugger performs an action (e.g log a message) automatically when the breakpoint is hit
- **Thread and virtual machine-specific breakpoints** stop the processor only when hit from a certain thread.

Address Modifiers

You can use address modifiers to select a certain type of memory or target access. For instance, if you type **S:0x0**, the debugger reads the content of memory at address 0x0 in TrustZone secure mode.

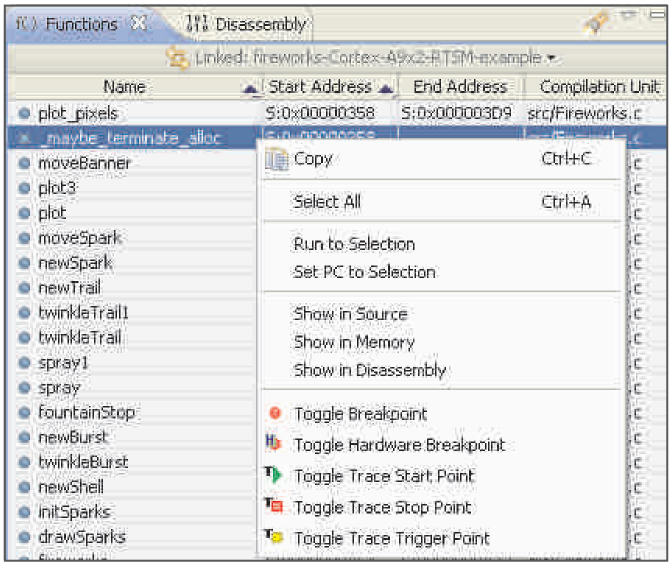
This is a list of the available address modifiers:

- **P:** Physical memory (MMU disabled)
- **S:** TrustZone secure mode
- **N:** TrustZone non-secure mode
- **H:** Hypervisor mode
- **DAP:** Direct memory access via the AHB-AP port of a CoreSight DAP. This port enables debugger access to physical memory with a running target.



Name	Value	Type	Count	Size	Location
3+3	6	int		32	
$pc	1178	unsigned int		32	$PC
*($pc)	507542560	int		32	S:0x0000049A
banner_data[0]	0	const pixel_t		16	S:0x0000178A
s	0X0015E018	Spark*	1	32	R4[@S:0x001FD
s->age	103	unsigned char		8	S:0x0015E028
(int) (N:0x300)	0x00000300	int		32	
Enter new expression...					

Expressions view.



Name	Start Address	End Address	Compilation Unit
plot_pixels	S:0x00000358	S:0x000003D9	src/Fireworks.c
_maybe_terminate_alloc			
moveBanner			
plot3			
plot			
moveSpark			
newSpark			
newTrail			
twinkleTrail1			
twinkleTrail			
spray1			
spray			
fountainStop			
newBurst			
twinkleBurst			
newShell			
initSparks			
drawSparks			

Functions View.

Starting a Performance Analysis Session

ARM **Streamline** Performance Analyzer is a system-level performance analysis tool for Linux and Android systems. It uses sample-based profiling, Linux kernel trace and software annotation techniques.

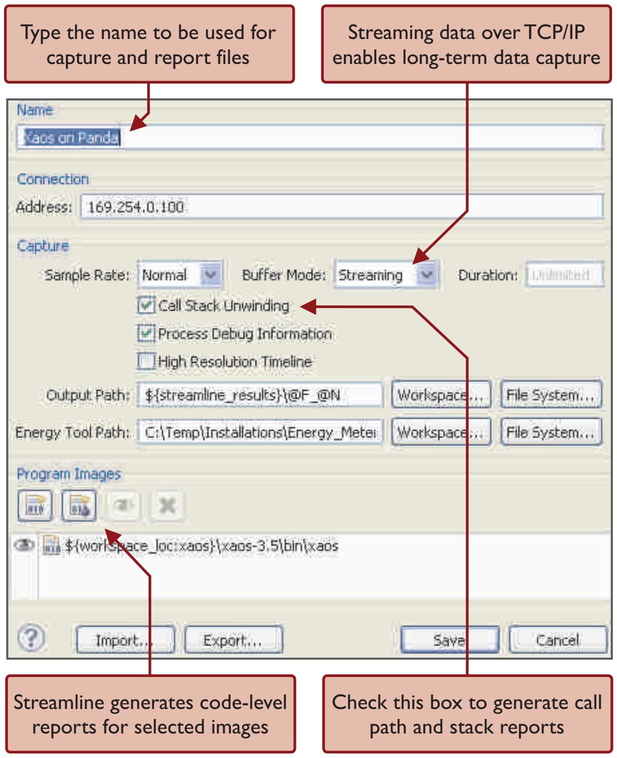
Setting up your system

To get the most out of Streamline you need:

- **TCP/IP connection** to the target, using an Ethernet link (Linux targets) or ADB over USB (Android targets)
- **Build the gator module** with the target's Linux kernel and copy the gator demon to the same directory. Gator is licensed under GPL, and works on Linux 2.6.32 or later
- **Enable frame pointers** when you build your application and libraries for Streamline to generate call-path reports
- **Annotate your code** with a printf-style interface to the gator driver, to highlight important software events
- **Add custom counters** to Streamline by pointing to their address in the memory map in the gator driver.

Launching Streamline

In Eclipse, open the **ARM Streamline Data** view in the Streamline perspective. Alternatively, start a capture and generate reports from the host's command line with the **streamline** command.


Type the name to be used for capture and report files
Streaming data over TCP/IP enables long-term data capture
Streamline generates code-level reports for selected images
Check this box to generate call path and stack reports
Program Images
Import...
Export...
Save
Cancel
Workspace...
File System...

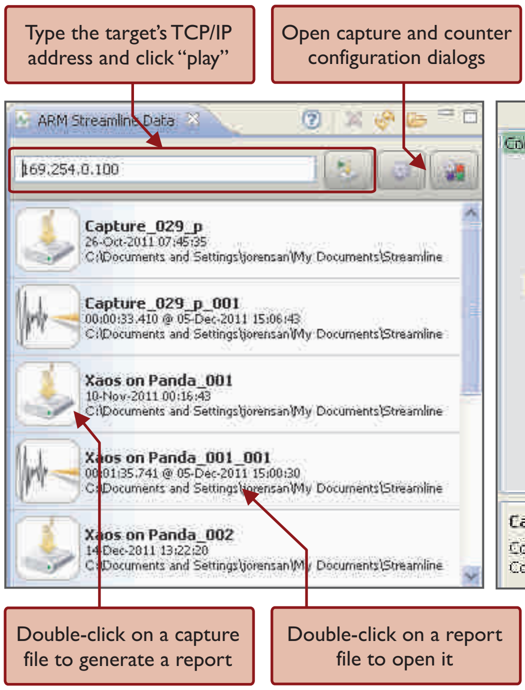
Name: Xaos on Panda

Connection
Address: 169.254.0.100

Capture
Sample Rate: Normal   Buffer Mode: Streaming   Duration: Unlimited
[✓] Call Stack Unwinding
[✓] Process Debug Information
[ ] High Resolution Timeline
Output Path: ${streamline_results}\@F_@N
Energy Tool Path: C:\Temp\Installations\Energy_Meter

${workspace_loc:xaos}\xaos-3.5\bin\xaos

Streamline Capture Options Dialog.


Type the target's TCP/IP address and click "play"
Open capture and counter configuration dialogs
Double-click on a capture file to generate a report
Double-click on a report file to open it
ARM Streamline Data

169.254.0.100

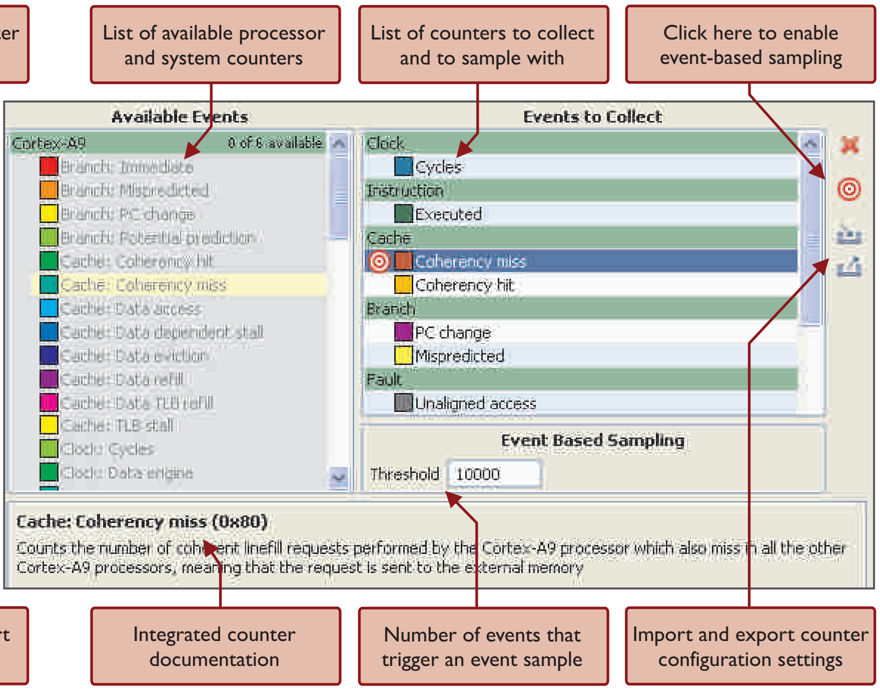
Capture_029_p
26-Oct-2011 07:45:35
C:\Documents and Settings\jorensan\My Documents\Streamline

Capture_029_p_001
00:00:33.410 @ 05-Dec-2011 15:06:43
C:\Documents and Settings\jorensan\My Documents\Streamline

Xaos on Panda_001
10-Nov-2011 00:16:43
C:\Documents and Settings\jorensan\My Documents\Streamline

Xaos on Panda_001_001
00:01:35.741 @ 05-Dec-2011 15:00:30
C:\Documents and Settings\jorensan\My Documents\Streamline

Xaos on Panda_002
14-Dec-2011 13:22:20
C:\Documents and Settings\jorensan\My Documents\Streamline


List of available processor and system counters
List of counters to collect and to sample with
Click here to enable event-based sampling
Integrated counter documentation
Number of events that trigger an event sample
Import and export counter configuration settings
Available Events
Events to Collect
Event Based Sampling

Cortex-A9   0 of 6 available
Branch: Immediate
Branch: Mispredicted
Branch: PC change
Branch: Potential prediction
Cache: Coherency hit
Cache: Coherency miss
Cache: Data access
Cache: Data dependent stall
Cache: Data eviction
Cache: Data refill
Cache: Data TLB refill
Cache: TLB stall
Clock: Cycles
Clock: Data engine

Clock
Cycles
Instruction
Executed
Cache
Coherency miss
Coherency hit
Branch
PC change
Mispredicted
Fault
Unaligned access

Threshold: 10000

Cache: Coherency miss (0x80)
Counts the number of coherent linefill requests performed by the Cortex-A9 processor which also miss in all the other Cortex-A9 processors, meaning that the request is sent to the external memory

Streamline Profiling Reports

Navigating a Streamline report

A Streamline report provides a great depth of performance-related information on seven panes: Timeline, Call Paths, Functions, Code, Call Graph, Stack Analysis and Logs.

Start with the Timeline and Functions for an overview of the execution. Use the other panes to focus your analysis.

Use the tabs on the top to navigate between the panes. Alternatively, right-click on a function name to highlight the function on another pane.

Sampling on events

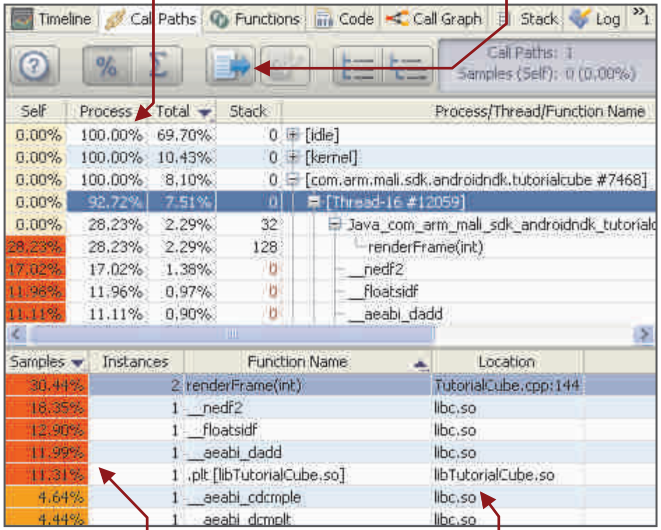
By default, the Streamline profiling reports are generated by sampling the program counter every 1ms or 10ms.

When you enable event-based sampling, Streamline takes samples when an event counter reaches the Threshold value selected. These samples fall on the instruction that caused the last event, and are used to fill the profiling reports with event-based data instead of time-based data.

For example, event-based sampling can be used to explore which parts of the code are causing cache misses or branch mispredictions. This feature requires PMU sampling support on the target and its Linux distribution.

Call tree shows processor time spent on each individual call path

All table-based reports can be exported to text files



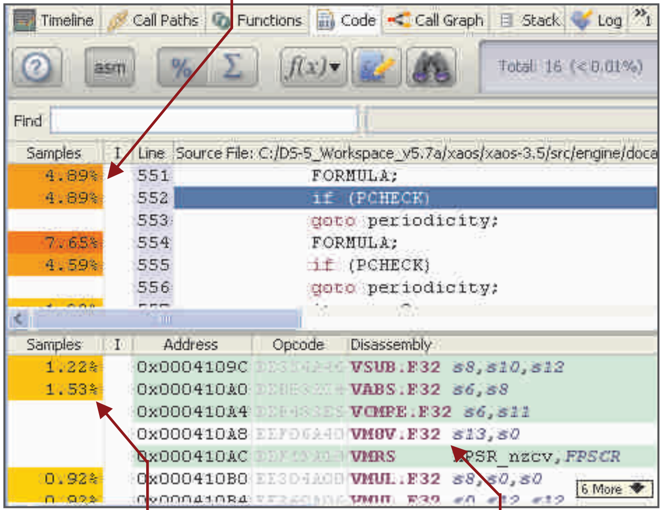
Self	Process	Total	Stack	Process/Thread/Function Name
0.00%	100.00%	69.70%	0	[idle]
0.00%	100.00%	10.43%	0	[kernel]
0.00%	100.00%	8.10%	0	[com.arm.mali.sdk.androidndk.tutorialcube #7468]
0.00%	92.72%	7.51%	0	[Thread-16 #12059]
0.00%	28.23%	2.29%	32	Java_com_arm_mali_sdk_androidndk_tutorialc
28.23%	28.23%	2.29%	128	renderFrame(int)
17.02%	17.02%	1.38%	0	__nedf2
11.96%	11.96%	0.97%	0	__floatsidf
11.11%	11.11%	0.90%	0	__aeabi_dadd

Samples	Instances	Function Name	Location
30.44%	2	renderFrame(int)	TutorialCube.cpp:144
18.35%	1	__nedf2	libc.so
12.90%	1	__floatsidf	libc.so
11.99%	1	__aeabi_dadd	libc.so
11.31%	1	.plt [libTutorialCube.so]	libTutorialCube.so
4.64%	1	__aeabi_cdcmple	libc.so
4.44%	1	__aeabi_dcmplt	libc.so

Color-coding assists with the identification of hot spots

Profiling report for the selected process, thread or call path

Color coding makes it easy to identify hot spots



Source File: C:/DS-5_Workspace_v5.7a/xaos/xaos-3.5/src/engine/doca

Samples	I	Line	Source
4.89%		551	FORMULA;
4.89%		552	if (PCHECK)
		553	goto periodicity;
7.65%		554	FORMULA;
4.59%		555	if (PCHECK)
		556	goto periodicity;

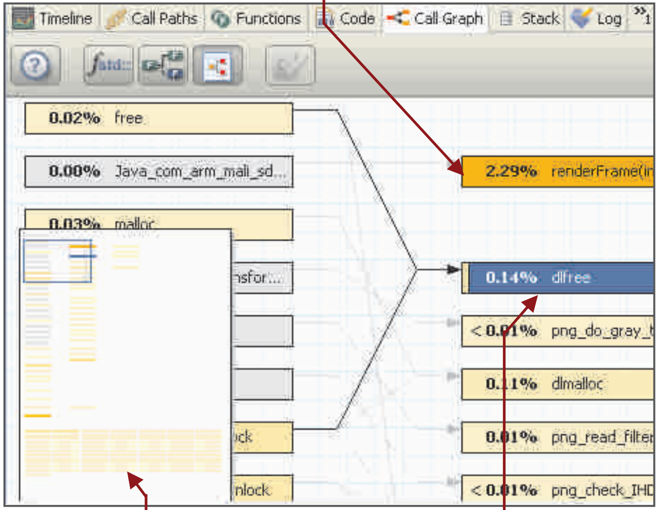
Samples	I	Address	Opcode	Disassembly
1.22%		0x0004109C	[illegible]	VSUB.F32 s8,s10,s12
1.53%		0x000410A0	[illegible]	VABS.F32 s6,s8
		0x000410A4	[illegible]	VCMPE.F32 s6,s11
		0x000410A8	[illegible]	VMOV.F32 s13,s0
		0x000410AC	[illegible]	VMRS APSR_nzcv,FPSCR
0.92%		0x000410B0	[illegible]	VMUL.F32 s8,s0,s0

Profiling data is reported down to source code or assembler level

Select a line of code to see its matching assembler code

Open a source file in the code view to deepen the analysis

Function usage highlighted with color coding



The mini-map shows all the functions in the code

Click on a function to highlight callers and callees

The call graph displays the relationships between functions

Streamline Timeline View

Charts show Linux events, processor and system counters

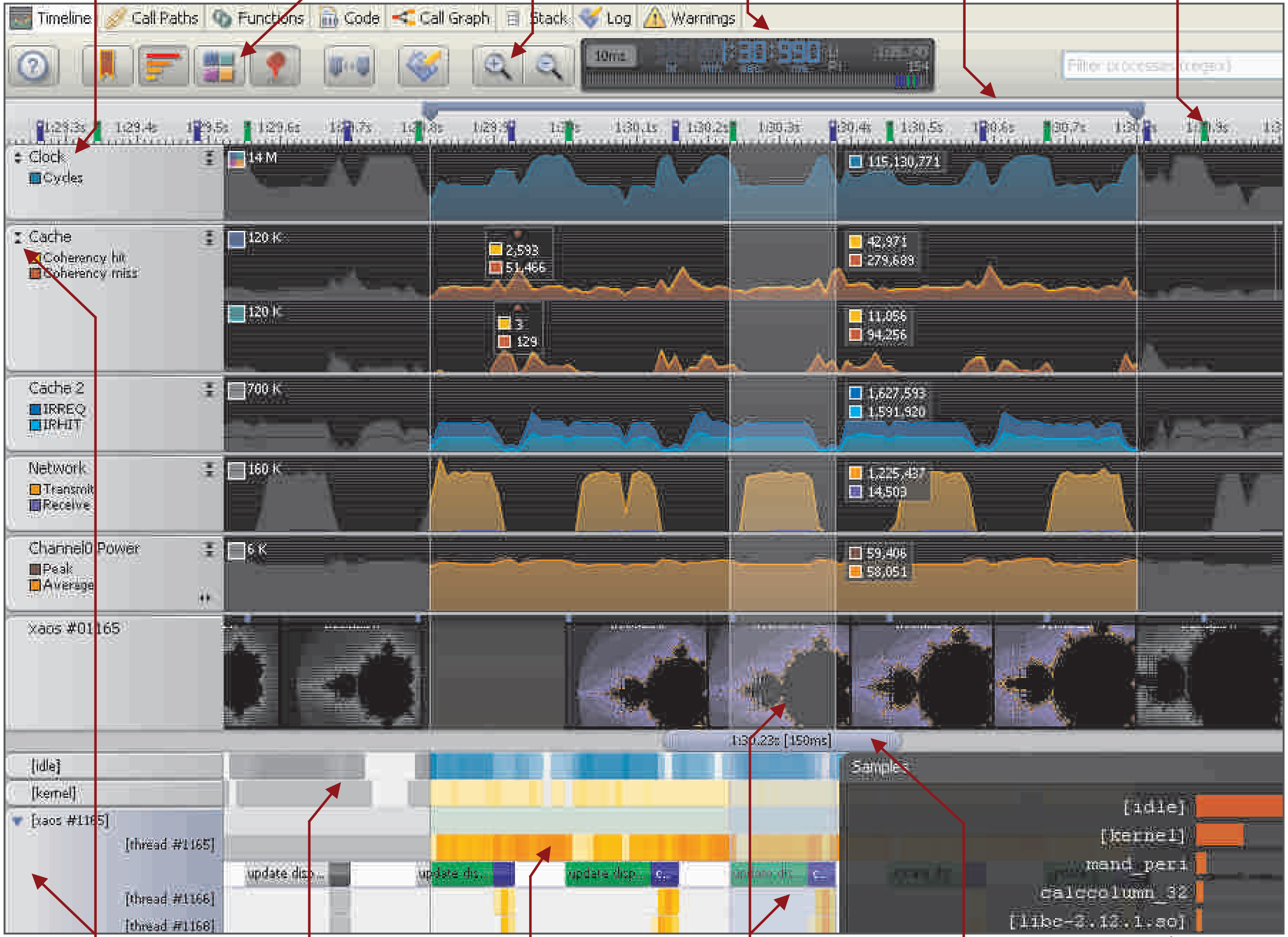
Activate the X-Ray mode to show core to thread affinity

Zoom from 1s to 100us or less per slot

Navigate the view easily using the timeline overview

Use the calipers to filter the captured data on time

Highlight points of interest with time bookmarks



Expand charts per core and threads per process

The process pane shows thread and process activity

Visualize thread heat map or core to thread affinity (X-Ray mode)

Mark software events with text and visual annotations

Aggregate for a period of time counter and sample information

Correlate aggregated counters with a function-level report

Find out more about DS-5

Web resources include product download, information and documentation, knowledge articles, support forums and sales contacts. For more information see www.arm.com/ds5 and www.arm.com/dstream.

Please send any feedback on the products or this guide to feedback-ds5@arm.com.

ARM®