

ARM DS-5 Development Studio

QUICK START GUIDE

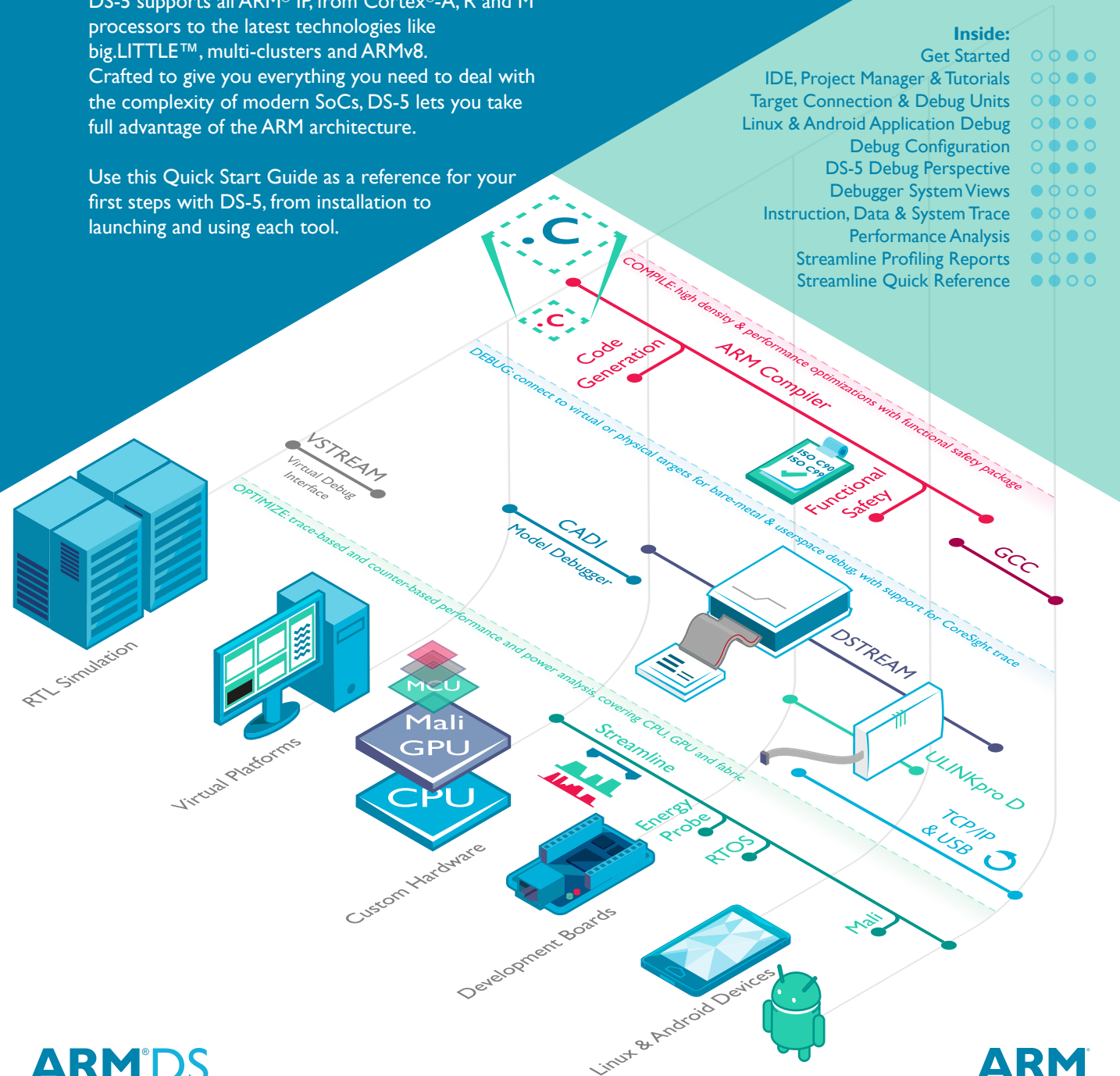
DS-5 supports all ARM® IP, from Cortex®-A, R and M processors to the latest technologies like big.LITTLE™, multi-clusters and ARMv8.

Crafted to give you everything you need to deal with the complexity of modern SoCs, DS-5 lets you take full advantage of the ARM architecture.

Use this Quick Start Guide as a reference for your first steps with DS-5, from installation to launching and using each tool.

Inside:

- Get Started ○○○○
- IDE, Project Manager & Tutorials ○○○○
- Target Connection & Debug Units ○○○○
- Linux & Android Application Debug ○○○○
- Debug Configuration ○○○○
- DS-5 Debug Perspective ○○○○
- Debugger System Views ○○○○
- Instruction, Data & System Trace ○○○○
- Performance Analysis ○○○○
- Streamline Profiling Reports ○○○○
- Streamline Quick Reference ○○○○



Quick Start Guide

ARM DS-5 DEVELOPMENT STUDIO GET STARTED WITH DS-5

DS-5 Downloads

Start your next project with a 30-day free trial of DS-5 Professional Edition. Download DS-5 from ds.arm.com/downloads and use your evaluation license to explore the full features of DS-5 Pro.

The ds.arm.com website also provides documentation, tutorials and blogs to help you learn how to use DS-5 to its maximum potential.

First Time Setup

Install DS-5 following the on-screen prompts and launch the **Eclipse for DS-5** application. Create a new **Workspace** folder to store your projects and settings.

Welcome Screen and Licensing

Next, you'll see the **Welcome** screen, which gives you a quick way of importing the included example applications as well as tutorials, hints and tips. To close the welcome screen, click on the **Go to Workbench** link. You can get back to this screen at any time by selecting **Help - Welcome**.

Minimum System Requirements



DS-5 is supported on the x86 host platforms listed below with at least: 2 GHz dual-core host processor, 2 GB RAM (4 GB recommended), 2 GB hard disk space.

- Windows XP Professional SP3 (32-bit)
- Windows 7 Professional SP1 (32/64-bit)
- RHEL 5 Desktop & Workstation (32/64-bit)
- RHEL 6 Workstation (32/64-bit)
- Ubuntu Desktop Edition 12.04 LTS (32/64-bit)
- Mac OS X 10.7 (DS-5 Community Edition only)

To license your copy of DS-5, select **Help - ARM License Manager** and then click **Add License**. If you already have an account at ARM.com, you can use your existing username and password.

To upgrade or change a license, return to the **ARM License Manager** and either add a license file, or enter a valid serial number.

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 you to manually set the path environment variable.

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**.

Reposition any view by left-clicking on its tab and dragging it.

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 .zip 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**.

Source code editing features in DS-5 include C/C++/asm syntax coloring, code autocompletion, refactoring and function, class and variable definition.

```

118 ttb_zero_loop
119 STmia r1!, {r3-r6} ; Store out four entries
120 SUBS r2, r2, #1 ; Decrement counter
121
122
123 ; Entry for VA used by code and data
124 ; This region must be coherent
125 MOV r1, pc ; Make a copy of the PC
    
```

```

BP147_TZPC_BIT_6 | BP147_TZPC_BIT_7;
setDecodeRegionNS(0, tmp);
setDecodeRegionNS(1, tmp);

void setDecodeRegionNS(unsigned int region, unsigned int bits)
{
    bits = bits & 0xFF;

    switch (region)
    {
        case BP147_TZPC_REGION_0:
            tzpc.TZPCDECPROT0Set = bits;
            break;

        case BP147_TZPC_REGION_1:
            tzpc.TZPCDECPROT1Set = bits;
    }
}
    
```



DS-5 Example Projects

DS-5 comes complete with a range of examples to get you started including:

- **Startup** examples include reset, MMU and cache initialization code for ARM processors.
- **Fireworks** is a bare-metal example ported to the Snowball, Beagle and Panda boards.
- **Distribution** is a complete Linux distribution to enable driver and application development. **N.B.** requires additional download.
- **Gnometriz** is an open source Linux application similar to the game Tetris.
- **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.
- **Example_library** demonstrates how to create a simple shared library in C for ARM Linux.
- **Application Rewind 1 & 2** allow you to rewind through two simple examples to locate software bugs using reverse debugging techniques.

ARM Compiler Tutorials

At ds.arm.com/tutorials, you can gain a deeper understanding of the ARM Compiler and the optimization techniques that it uses.

By understanding how to control optimizations such as NEON™ autovectorization, instruction scheduling and common subexpression elimination, you can get the most out of your ARM processor and ensure that you have maximum visibility during debug.

Other DS-5 tutorials available online include project management and code editing, Streamline performance optimization and bare metal, Linux and Android debugging.

Quick Start Guide

ARM DS-5 DEVELOPMENT STUDIO TARGET CONNECTION & DEBUG UNITS

DSTREAM: High-Performance Debug & Trace

The **DSTREAM™** Debug and Trace Unit provides DS-5 Debugger with 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.

ULINKpro D: Easy Debug

The **ARM Keil® ULINKpro™** family can be used with DS-5 for software debug on popular devices based on the ARM Cortex®-M, Cortex-R and Cortex-A processor series.

An ideal fit for equipment manufacturers using off-the-shelf processor devices, the ULINKpro family enables cost efficient run-control debug on devices ranging from MCUs to multicore application processors.

DS-5 Configuration Database

DS-5 provides **pre-configured JTAG and trace** for catalog 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**. You can also add custom configuration databases by selecting **Window - Preferences - DS-5 - Target Database**.

Debugging Custom Devices

DS-5 ships with powerful tools for custom device bring-up:

- **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. Manual configuration is also enabled.
- **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.



DSTREAM

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

Learn more: ds.arm.com/dstream/

ULINKpro D

Debug Interface	JTAG and Serial-Wire Debug
JTAG Speed	50 MHz
Download	1 MB/s
Trace	ETB trace support
Supported Devices	Up to quad-core Cortex-A9

Learn more: ds.arm.com/ulinkpro/

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 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:

- **Automated software image download:** DS-5 Debugger uses RSE connections to automate the download of images to the target, executing gdbserver and connecting to it.
- **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.
- **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.

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**.

Application Rewind - Go Back to the Bug

User-space Linux application development in DS-5 is assisted by **Application Rewind**, which allows you to “reverse debug” your software once an error has occurred.

Application Rewind runs on a lightweight debug agent on your target, so there's no need for JTAG or CoreSight instruction trace; simply connect over TCP/IP.

Read more: ds.arm.com/application-rewind

Fixed Virtual Platforms (FVPs)

Start bare metal and Linux software development without a hardware target.

DS-5 includes a sample FVP, which is a fast simulation model of a complete ARM-based SoC with a processor, system memory, and peripherals such as keyboard, mouse, UARTs, Ethernet, and LCD.

FVPs run at speeds comparable to the real hardware and significantly cut your development time.

To run the example applications, download the **ARM Linux distribution**. When launched, the model automatically boots Linux and enters a state in which you can load and debug your applications.

FVPs 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.

For 64-bit applications, the **ARMv8 Architecture Envelope Model (AEM)** is available for download for Linux, Android and bare-metal development.



Quick Start Guide

ARM DS-5 DEVELOPMENT STUDIO DEBUG CONFIGURATION

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.

Add ELF images to the configuration for source level debug.

Select **Linux OS awareness** or an **RTOS** per connection.

Use the DTSL options dialog to configure target specific options like PTM and ETM instruction and data trace, cross-triggers, clocks, power and ITM and STM instrumentation trace.

Debug and Trace Services Layer (DTSL)



DTSL is the 'glue' between the debugger and the target. This layer of the debugger is accessible from python scripts which are used to configure the debug and trace settings for individual components of the SoC and the overall topology.

These scripts are auto-generated by the debugger and can also be modified to extend debugger functionality or work around broken components on the SoC. Every device in the DS-5 database has a corresponding DTSL Python script.

When used in conjunction with the optional DSTREAM Template Development Kit (TDK) it is even possible to make the DS-5 debugger work with new user defined IP blocks.

The screenshot shows the DS-5 Debugger interface. The main window displays the 'Debug Configurations' dialog for a configuration named 'TrustZone-versatile-A9x4-example'. The 'Connection' tab is active, showing a list of target connections. A callout box points to the 'Debug Cortex-A9x4 SMP' connection. The 'DTSL Options' tab is also visible, showing options for 'DS-5 Debugger will connect to a DSTREAM to debug a bare me...'. A callout box points to the 'DTSL Options' tab. The 'Files' tab is active, showing a list of files. A callout box points to the 'Files' tab, indicating that the user should select an application and check 'Load symbols' to enable source-level debug. The 'DTSL Configuration for DSTREAM' dialog is open, showing the 'Trace Capture' tab. The 'Name of configuration' is 'default'. The 'Trace Capture' section has three radio buttons: 'None', 'On Chip Trace Buffer (ETB)', and 'DSTREAM 4GB Trace Buffer' (selected). The 'Clear Trace Buffer on connect' and 'Start Trace Buffer on connect' checkboxes are checked. The 'Trace full action' dropdown menu is set to 'Trace wraps on full and continues to store data'. Callout boxes provide instructions: 'Create a new DS-5 Debug configuration and customize its name' (pointing to the configuration name), 'Select a target from the configuration database, or a gdbserver connection' (pointing to the connection list), 'Customize and control trace options from the DTSL configuration' (pointing to the DTSL Options tab), and 'Browse through the multiple tabs to configure different IP blocks' (pointing to the DTSL Configuration dialog).

Create a new DS-5 Debug configuration and customize its name

Select a target from the configuration database, or a gdbserver connection

In the Files tab, select an application and check "Load symbols" to enable source-level debug.

Customize and control trace options from the DTSL configuration

Browse through the multiple tabs to configure different IP blocks

DS-5 includes a powerful graphical C/C++ debugger, opened 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 driven by 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. Each view has an options toolbar with further functions which can be explored.

The screenshot shows the ARM DS-5 Debug Perspective in Eclipse IDE. The interface is divided into several panes:

- Debug Console:** Shows active threads for 'xaos-FVP-example connected'. Thread 1156 #5 is stopped on breakpoint #2 at address 0x54C.
- Commands:** Shows the command 'continue' and execution details: 'Execution stopped at breakpoint 2.14: 0x000508A8 In thread 5 (OS thread id 1156) 0x000508A8 206,0 while (BTEST && iter);'.
- Registers:** Shows the 'anti.c' namespace with variables like 'antialias_filter.getinstance'.
- Disassembly:** Shows assembly instructions for 'Next Instruction 100':

Address	Opcode	Disassembly
0x000508A4	E50B9034	STR r9, [
0x000508A8	BNE	manc
0x000508AC	E1A03009	MOV r3, r
0x000508B0	E1A09004	MOV r9, r
0x000508B4	E1A04003	MOV r4, r
- Functions:** Lists functions like 'main', 'start', and 'call_gmon_start' with their start and end addresses.
- Variables:** Shows local variables 'zre' (value: -1.23945434) and 'zim' (value: 0.72901964).
- Breakpoints:** Shows a breakpoint at 'docalc.c:206 [#2 A32 (ARM), 2 hits]'.
- Memory:** Shows memory addresses like '0x0005A124' and '0xE92D4818'.
- Source Code:** Shows the C code for 'docalc.c' with a 'while' loop and an 'if' statement.

Callout boxes provide the following information:

- 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 help
- Drag & drop lines from history view to scripts view to create a script
- Switch between C/C++ and DS-5 Debugger perspectives
- Fast View bar for minimizing often-used debug views
- Source and assembly code views synchronized with color coding
- Click the down arrow for the sub-menu, 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

Quick Start Guide

ARM DS-5 DEVELOPMENT STUDIO DEBUGGER SYSTEM VIEWS

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**.

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) & 0xffff**.

The way the debugger accesses the memory system is configured via the **memory map**. View the memory map with the **info mem** command and configure it with **mem**.

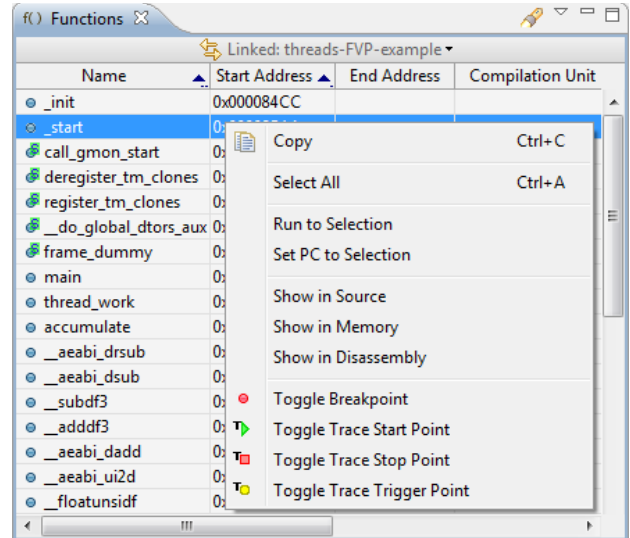
Breakpoints and Watchpoints

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. For instance, if you type **S:0x0**, the debugger reads the memory at address 0x0 in TrustZone™ secure mode.

The following address modifiers are supported in DS-5:

- **P:** Physical memory (MMU disabled)
- **S:** TrustZone secure mode
- **N:** TrustZone non-secure mode
- **H:** Hypervisor mode

You can also debug a running target with direct access to memory via the CoreSight Debug Access Ports. Use the prefix **APB:**, **AHB:** or **AXI:** to select a particular DAP interface to memory.

For **AArch64**, memory spaces can be accessed for **ELIN:**, **EL1S:**, **EL2:** and **EL3:** exception levels.

Instruction and Data 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.

Data trace optionally records address and register values from load and store instructions. Combined with the instruction trace this gives a complete record of execution.

When trace is enabled, DS-5 begins to capture instructions and data. Trace can be downloaded from the trace buffer at any time, or once the capture is complete. Trace data is synchronized with the code view.

System Trace

The **Instrumentation Trace Macrocell (ITM)** and **System Trace Macrocell (STM)** provide high-bandwidth, low latency “printf-style” instrumentation links. ITM and STM output is routed to the on-chip trace bus, and can be collected by the debugger and displayed in its **Event Viewer** or exported to a text file. Using the CoreSight **Global Timestamp**, trace from all these sources can be synchronized and cross referenced.

Using Tracepoints and Filters

By default, DS-5 traces everything executed by a processor into a circular buffer.

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 in the trace view and the tools can be configured to collect trace around, before, or after the trigger. Just check the **ETM Triggers Halt Execution** box in the debug configuration menu to trace around a trigger point.

Trace Requirements

Trace typically requires a JTAG connection to devices featuring an **Embedded Trace Macrocell (ETM)** or **Program Trace Macrocell (PTM)**. Data trace is not available on all cores. The amount of trace 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 in DSTREAM.

CoreSight Library & Snapshot Viewer



The **CoreSight on-target access library** enables the software running on your ARM target to interact directly with CoreSight devices, enabling “flight-recorder” trace. This removes the need to attach an external debugger, allowing you to debug software crashes in the field. DS-5 includes an example application to illustrate the use of the CoreSight library on a hardware target. The collected data can then be exported for use in DS-5 Debugger.

For easy use of the CoreSight on-target access library, DS-5 Debugger contains **DS-5 snapshot viewer**, which enables you to open register values, memory values and debug symbols captured from software in the field. For instance, this allows you to analyze trace data collected during a crash dump.

Use the Capture Device, Source and Ranges tabs to configure trace

Heat map of activity and percentage activity to identify local hotspots

Trace	Capture Device	Source	Ranges
wr_reg		36.51%	
GLCD_SetWindow		32.70%	
wr_cmd		21.93%	
<Unknown>		4.94%	
SystemCoreClockUpdate		1.32%	
GLCD_Clear		1.27%	
ADC_Init		0.50%	

Index	Address	Opcode	Detail
-8,303	0x08000E64	B2A9	UXTH r1,r5
-8,302	0x08000E66	2050	MOVS r0,#0x50
-8,301	0x08000E68	F000F9A6	BL wr_reg ; 0x80011B8
-8,300	0x080011B8	B570	PUSH {r4-r6,lr}
-8,299	0x080011BA	4605	MOV r5,r0

From the Trace view, synchronize other views using timestamps, view address and values for memory accesses, and view tracepoints and triggers embedded in the trace list

Quick Start Guide

ARM DS-5 DEVELOPMENT STUDIO PERFORMANCE ANALYSIS

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 started with Streamline you need:

- **TCP/IP connection** to the target, using WiFi, Ethernet or ADB over USB (Android targets).
- **Enable Linux kernel config options** required by gator.
- **Build the gator.ko module** and gator daemon from `<install_dir>\arm\gator\` with the target's Linux kernel.

In addition you can get more out of Streamline if you:

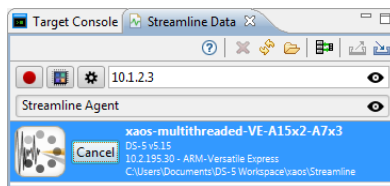
- **Add frame pointers** to your application and libraries using the `-fno-omit-frame-pointer` compiler flag. This allows Streamline to generate call-path reports.
- **Annotate your code** with a printf-style interface to the gator driver, to highlight important software events.
- **Add custom counters** to the gator driver.
- **Connect an ARM Energy Probe or NI DAQ** to the target to measure voltage and current consumption.

Launching Streamline

In Eclipse, open the **ARM Streamline Data View**. By default Streamline generates profiling reports by sampling the program counter at a constant time interval.

When you enable **event-based sampling**, Streamline takes samples when an event counter reaches the threshold value selected. For example, event-based sampling can be used to explore which parts of the code are causing cache misses.

The simplified Streamline Data view generates and analyzes reports with a double-click



Counter Configuration

Open the **counter configuration** panel to view and select from available counters, as well as importing custom counters. For event-based sampling, the event threshold can also be set.

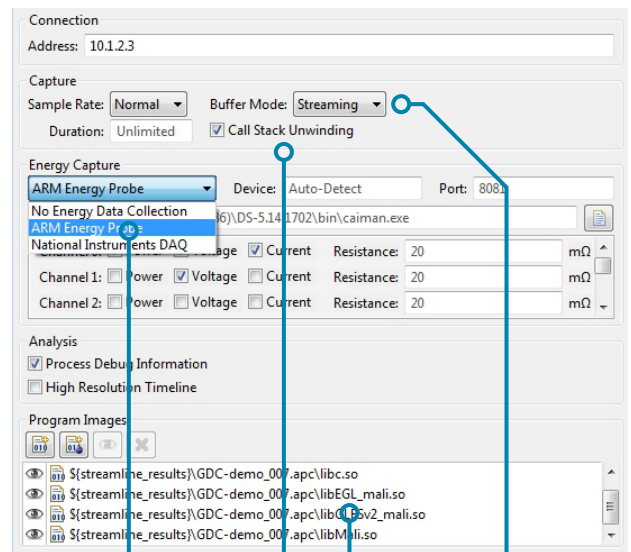
Streamline for RTOS



New to DS-5 is the ability to profile and optimize RTOS and bare-metal software on Cortex-M3 and M4 processors using Streamline.

Making use of the **Data Watchpoint and Trace (DWT)** and **Instrumentation Trace Macrocell (ITM)**, Streamline connects through DSTREAM to collect system information with near-zero intrusion.

Turning on event based sampling is reflected in the Streamline report



Select energy measurement tool and set shunt resistor values

Streaming data over TCP/IP enables long-term data capture

Call path and stack reports are enabled by checking call stack unwinding

Streamline generates source-code level reports for selected images

Navigating a Streamline report

Streamline provides a comprehensive, intuitive report on overall **system performance**, from big, LITTLE CPU clusters to interconnect fabric, GPUs and also MCUs. Streamline enables analysis of the interaction between software and all the critical hardware components in complex SoCs.

Start with the **Timeline** pane which provides a complete time orientated view of performance and highlights areas of interest. Drill down to function level using the other views.

Use the tabs to navigate between the panes. Or, right-click on a function name to highlight it in another pane.

Detailed analysis

Average performance and hot functions can be assessed immediately in the **Call Paths** and **Code** views. Anomalies can be identified visually from charts, with reference to the Filmstrip or with a user defined printf-like **Annotation**.

Once identified, the calipers can be used to isolate a region of the timeline and recalculate the other reports.

The process heat map can be switched to focus on kernel scheduling or GPU activity by clicking on the green button. CPU Wait charts reveal why the Linux kernel rescheduled a thread, for example, due to blocking a I/O operation.

Start from a high-level analysis and drill down to source code level

Isolate and expand on a period of time with the caliper tool

Switch process/thread heat map between alternative sources

Heat map of activity can be broken down per cluster or core in X-ray mode

Samples	% Samples	I	Line	Source File
5,321	26.05%	137	GLCD_SetCursor(320, 240);	C:/Documents/DS-5_Workspaces/DS-5 Workspace
9,164	44.86%	139	for(i=0; i<DISP_SIZE; i++)	
1,122	5.49%	143	if(val) GLCD_WritePixel(val);	
4,823	23.61%	144	else GLCD_WritePixel(backBuf[i]);	

Samples	% Samples	I	Address	Opcode	Disassembly
					WriteFB
			0x08000DA6	E92D41F0	PUSH {r4-r8,lr}
			0x0800DAA	2400	MOVS r4,#0
			0x0800D4C	4878	LDR r0,{pc}+0x1e4 ; 0x800f90
			0x0800DAE	6800	LDR r0,[r0,#0]
			0x0800DB0	8B90	CBNZ r0,0x0800E18 ; WriteFB +
			0x0800DB2	2001	MOVNS r0,#1
			0x0800DB4	4976	LDR r1,{pc}+0x1dc ; 0x800f90
			0x0800DB6	6008	STR r0,[r1,#0]
			0x0800DB8	E44E76A0	MOV r6,#0x10

Self	% Self	Instances	Function Name	Location	Image
24,088	65.24%	2	WriteFB	framebuf.c:129	Framebuffer_MCBSMT32E.axf
5,090	13.79%	2	GLCD_WritePixel	glcd.h:183	Framebuffer_MCBSMT32E.axf
3,239	8.77%	1	FrameBuf_LoadBackG	framebuf.c:206	Framebuffer_MCBSMT32E.axf
1,580	4.28%	2	wr_reg	glcd.h:163	Framebuffer_MCBSMT32E.axf
1,542	4.18%	2	wr_dat	glcd.h:153	Framebuffer_MCBSMT32E.axf
640	1.73%	1	ExtSRAM_Test	ext_sram.c:153	Framebuffer_MCBSMT32E.axf
234	0.63%	1	FrameBuf_PutChar	framebuf.c:99	Framebuffer_MCBSMT32E.axf
142	0.38%	1	_m_memset_w	Framebuffer_MCBSMT32E.axf	Framebuffer_MCBSMT32E.axf
104	0.28%	1	running_light	led.c:58	Framebuffer_MCBSMT32E.axf
47	0.13%	1	os_idle_demon	RTX_Conf_CM.c:219	Framebuffer_MCBSMT32E.axf
45	0.12%	1	led_out	led.c:52	Framebuffer_MCBSMT32E.axf
18	0.05%	1	delay	GLCD.c:52	Framebuffer_MCBSMT32E.axf
12	0.03%	1	SystemCoreClockUpdate	system_stm32f10x.c:307	Framebuffer_MCBSMT32E.axf

Streamline Quick Reference Hints & Tips

Connect an ARM Energy Probe for energy measurements

Use X-ray mode for per core and per cluster thread activity

Capture GPU activity and data from Mali OpenGL ES and OpenCL drivers

Filter threads to reconfigure the heatmap and charts

Hover over charts to use the quick access tooltip for data at a glance

Quickly zoom to pre-defined time levels, or use the magnifier

Blue dashes show CPU contention, red show delay caused by IO

Highlight a subset of processes to focus on their overall impact

Expand charts per core and threads per process, and configure expressions

Easily create charts for counter ratios with the Snippets menu

All brand names or product names are the property of their respective holders. Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder. The products described in this document are subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given in good faith. All warranties implied or expressed, including but not limited to implied warranties of satisfactory quality or fitness for purpose are excluded. This document is intended only to provide information to the reader about the product. To the extent permitted by local laws ARM shall not be liable for any loss or damage arising from the use of any information in this document or any error or omission in such information. Copyright © 2014 ARM Ltd.

X-ON Electronics

Largest Supplier of Electrical and Electronic Components

Click to view similar products for [panasonic](#) manufacturer:

Other Similar products are found below :

[ECE-A1HKAR47](#) [ELK-EA102FA](#) [ELC-09D151F](#) [EEC-S0HD224H](#) [ELL-5PS3R3N](#) [HC2-H-DC48V-F](#) [HL2-HP-AC120V-F](#) [HL2-H-DC12V-F](#) [HL2-HP-DC12V-F](#) [HL2-HP-DC6V-F](#) [HL2-HP-DC24V-F](#) [HL2-H-DC110V-F](#) [HC4-H-DC24V](#) [HL2-HTM-DC24V-F](#) [HL2-HTM-AC24V-F](#) [HC4-H-AC24V](#) [HC4-H-AC120V](#) [HC4-H-DC12V](#) [EEC-RG0V155H](#) [AZH2031](#) [RP-SDMF64DA1](#) [EEF-UD0K101R](#) [EVM-F6SA00B55](#) [RP-SMLE08DA1](#) [ELC-12D101E](#) [ERA-3YEB272V](#) [EEC-RF0V684](#) [ERA-3YEB153V](#) [ELC-3FN2R2N](#) [ERA-3YEB512V](#) [ERJ-1GEJ564C](#) [ERZ-V20R391](#) [ETQ-P3W3R3WFN](#) [ELL-ATV681M](#) [ELK-EA100FA](#) [EEF-UD0J101R](#) [LC-R121R3P](#) [ERA-3YEB303V](#) [ERZ-V05V680CB](#) [EEF-UE0K101R](#) [ELK-E101FA](#) [EEC-S0HD224V](#) [EVQ-PAC05R](#) [ELK-EA222FA](#) [LT4H-DC24V](#) [LT4HL8-AC24V](#) [LT4HW-AC24V](#) [LT4HWT8-AC240V](#) [LT4HWT-AC240VS](#) [CX-444-P-Z](#)