

arm

Arm Development Studio

Learning Outcomes

At the end of this module, you will be able to:



Arm Development Studio overview

Arm Development Studio featuring Keil MDK

Complete suite of tools for Arm Cortex-A/R/M based devices



Earliest Arm core support



No one knows Arm better



Accelerates time to market

Everything you need to develop Arm based projects from tiny sensors to server grade multi-core SoCs

- Includes Arm Compiler 6, Debugger, Streamline performance analyzer, Mali Graphics Debugger and Fixed Virtual Platforms
- Includes Keil MDK for power optimized microcontroller software development
- Supports all SoC configurations: from single core to complex multi-cluster and multi-core

Investment protection

- Early and dependable support for new Arm hardware IP
- Used in Arm for architecture validation
- Developed over many years of continuous improvement with billions of shipped products built
- Technical support from the experts in Arm technology

Innovate earlier and develop faster for competitive edge

- Develop software pre-silicon
- Supports full development workflow from emulators to production hardware
- Software building blocks for fast application development including CMSIS, middleware and RTOS
- Targeted performance analysis for rapid software optimization

Arm Development Studio

Create & build Code

- Latest Arm processor support
- Optimized code generation and embedded libraries

Debug Code

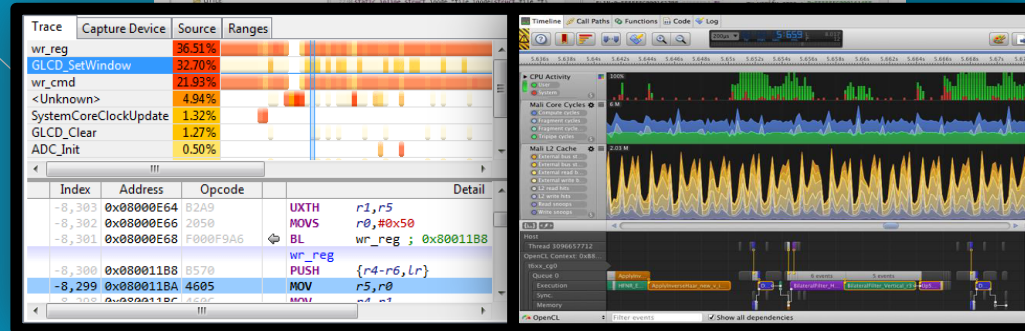
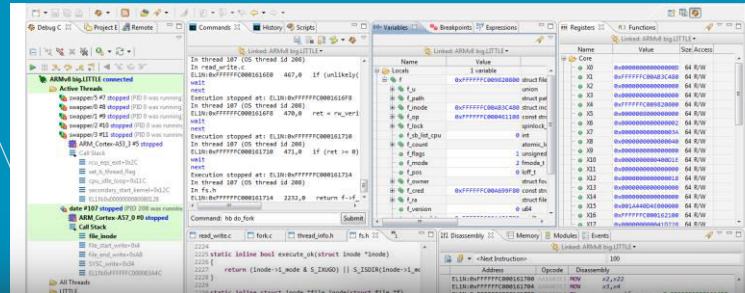
- Inspect CPU state and execution
- Set breakpoints, run scripts, debug exceptions
- Cache/MMU/MPU visibility
- AMP, SMP, big.LITTLE, DynamIQ

Visualize Code

- View code history up to breakpoint
- View code instrumentation output
- Make sense of OS level events and threads
- Track GPU calls and reconstruct frame buffers

Speed Up Code

- Tune code for optimal cache usage
- Thread/process level visualization
- GPU usage vs CPU usage
- Energy use over time



Tools designed for the entire product lifecycle



Fast & Cycle Models

- Virtual platforms for architectural exploration and early software development



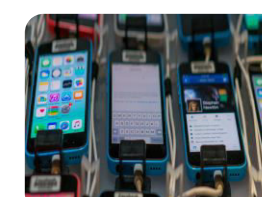
DSTREAM™

- Family of debug hardware for FPGA & board bring-up for complex SoCs



ULINKpro D

- Family of low-cost debug hardware for MCU & MPU

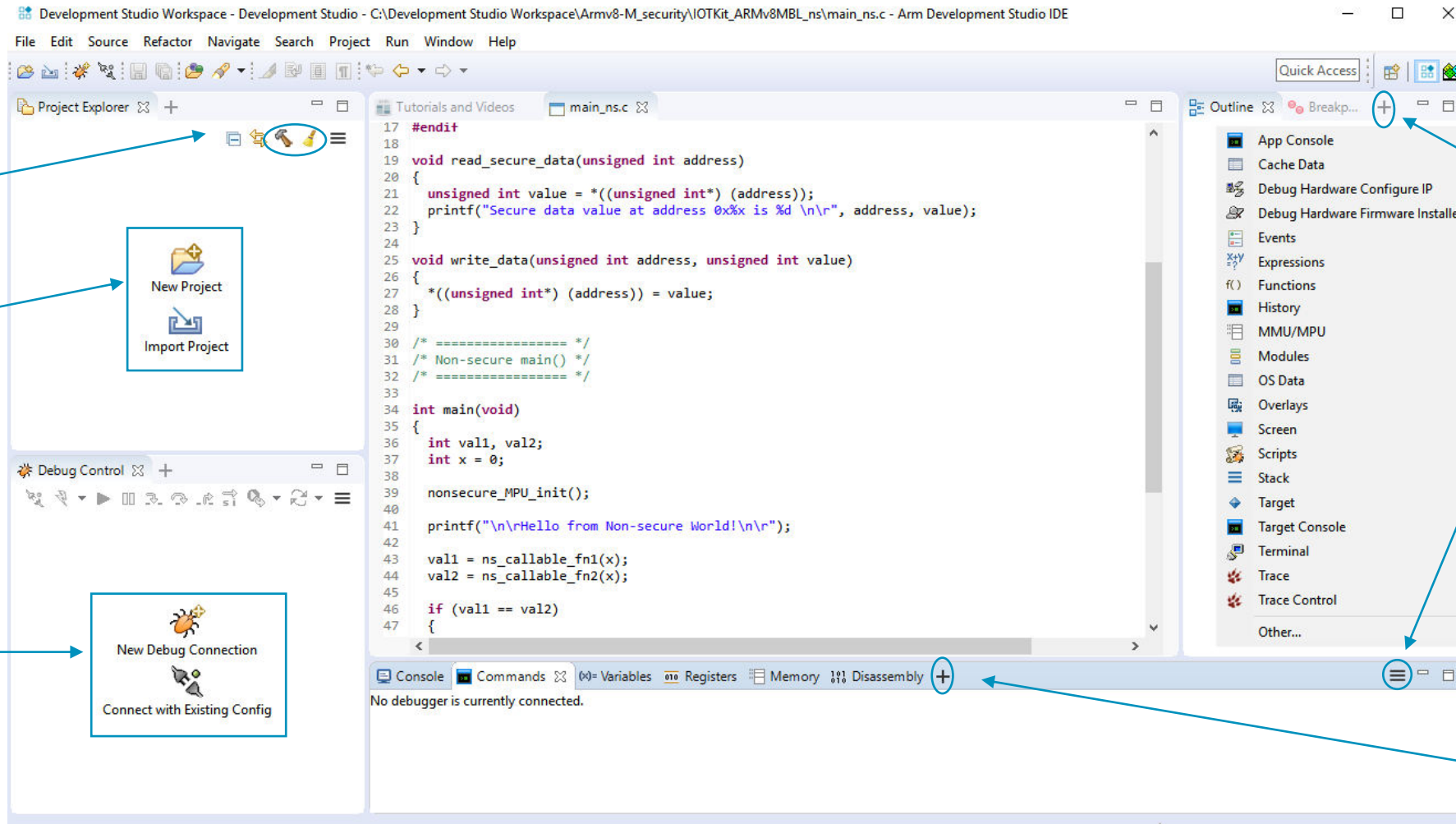


TCP/IP

- Application-level debug for BSP and reference design rollout

Arm Development Studio IDE

One perspective, simplified UI with easy and quick access

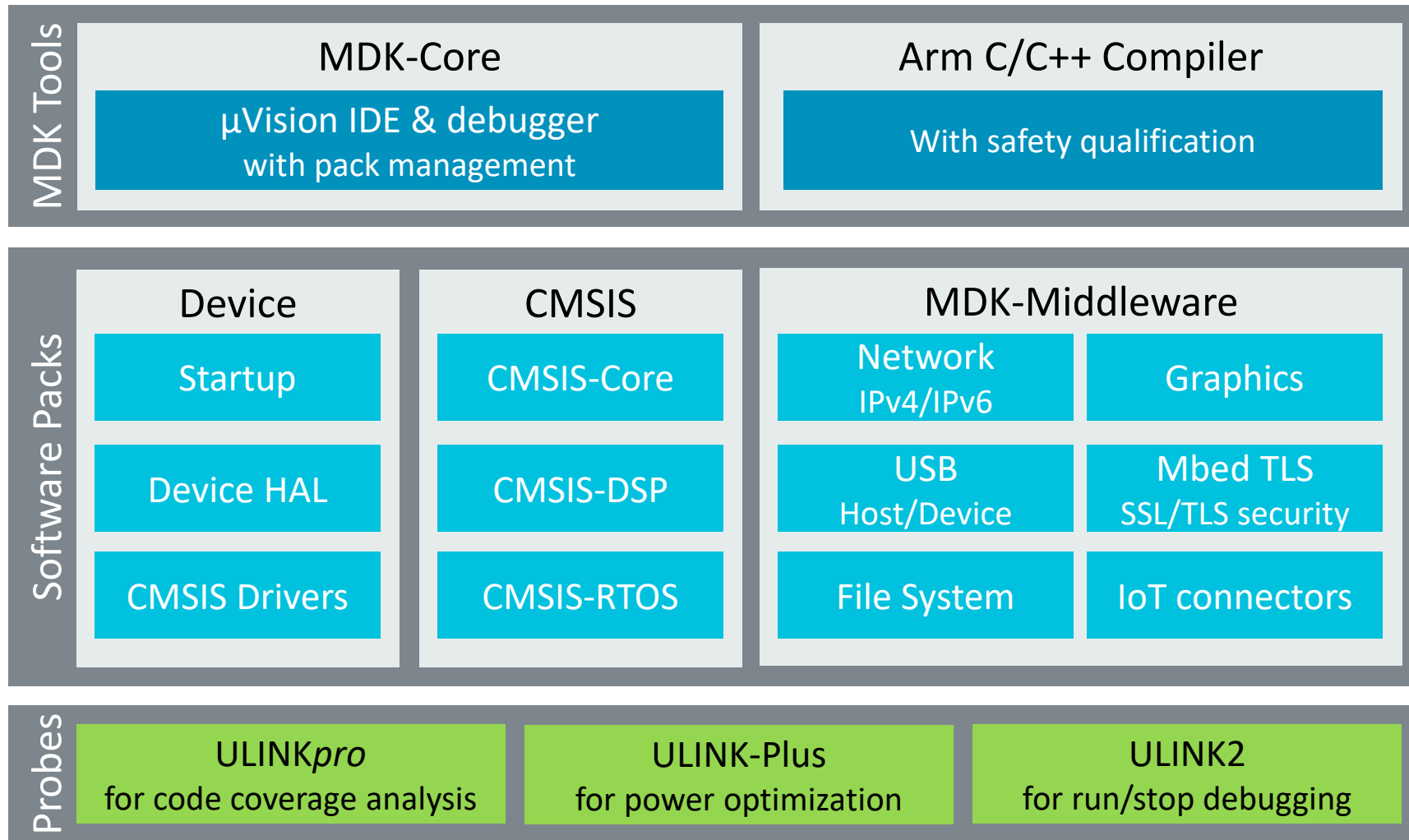


Quickly add or reopen new views

Hamburger icon (hidden menu options)

Quickly add or reopen new views

Keil MDK overview



arm

Arm Compiler 6

Arm Compiler 6: Leading-edge C/C++ toolchain



Comprehensive support



Real performance



Safety qualified

- The only embedded compiler you'll ever need
 - Validated support for new Arm technologies significantly
 - Architecture validation
 - Developed and supported by Arm architecture experts
- Efficient and accurate code for bare-metal and RTOS
 - Finely tuned Arm proprietary embedded libraries and linker
 - Optimized for a wide range of applications, not just a single benchmark
 - Support for the most recent language standards
- Path to functional safety certification
 - TÜV SÜD qualified for software development to multiple safety standards
 - Comprehensive Qualification Kit provides essential supporting documentation
 - Long Term Maintenance from stable & isolated code branch

Basic armclang options

```
armclang -c --target=arm-arm-none-eabi -mcpu=Cortex-A9 -g -O2 foo.c
```

- `--target=` selects the execution state
 - `arch64-arm-none-eabi` for v8A AArch64
 - `arm-arm-none-eabi` for v8A AArch32, Cortex-M, and Cortex-R, and older CPUs
- Additionally require
 - `-march` and/or `-mcpu` to select specific architecture or core
 - `-mcpu=list` shows all available options (for specified target)
- `-On` to select optimization level ($0 \leq n \leq 3$)
 - Also `-Os`, `-Oz`, `-Ofast`, `-Omax`
 - `-O0` default (extremely poor optimization)
 - `-flto` to enable link-time optimization (implied with `-Omax`)
- `-g` to enable debug info

Basic armlink options

- `--cpu=` selects the CPU (or architecture)
 - Mainly used as a check when merging projects
- `--scatter` to specify scatter description file
 - Defines platform memory map
- `--entry` to specify entry point (symbol or address)
- `--lto` Link-Time Optimization (armclang objects built with `-flto`)

Arm Compiler 5 (armcc)

- Legacy compiler, no longer in development
 - No support for any Armv8 architectures
 - However 20+ years of history, many projects exist based on this compiler
- Recommend all projects start with (or migrate to) Arm Compiler 6 (armclang)
 - Thorough migration [documentation](#) available
 - For many users, C/C++ source code will simply be directly portable
 - Changes only to command options
 - armcc source containing embedded assembler must be re-written to GNU assembler (gas) format
 - Recommend rewriting with compiler [intrinsic](#)s

arm

Debug and trace

Arm Debugger key points

Full support of Arm IP

- Arm Debugger is developed alongside Arm IP
- Support for all Arm cores, including leading-edge support for cores that are not yet public
- Comprehensive support for all Arm debug and trace IP

Flexible architecture

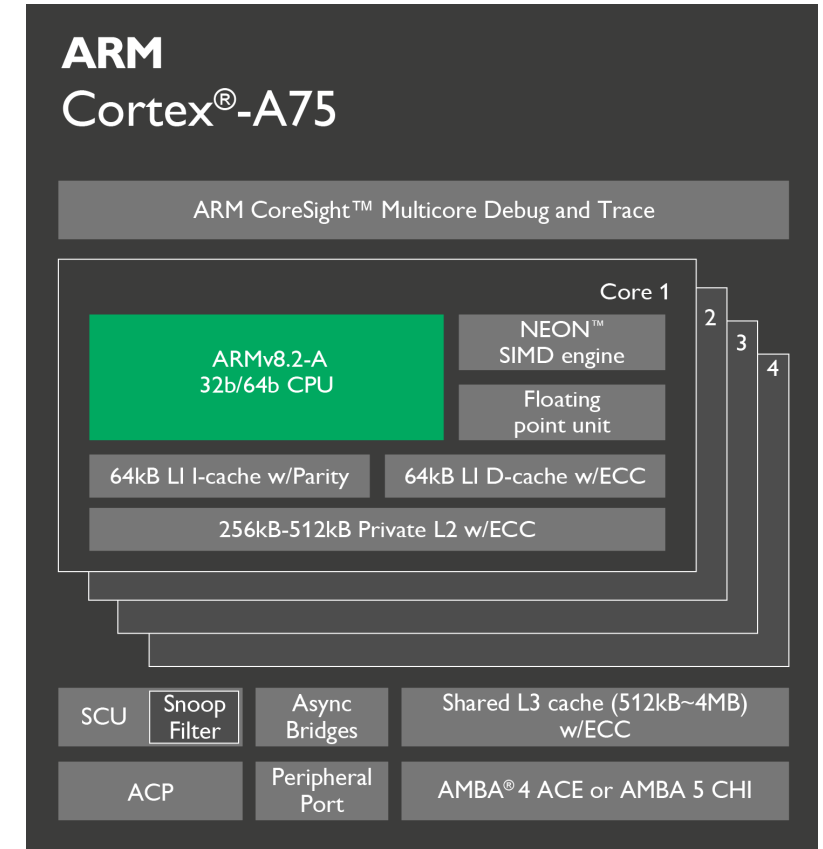
- Support for target complexity, challenging implementations, non-Arm IP
- Extension APIs – customize debugger functionality, integrate with other tools

Designed for productivity

- CMSIS-Pack compatibility
- Integrated with Eclipse and Python – both widely used
- Debugger views and functionality optimized for fast, efficient, and clear operation
- Advanced bring-up tools, expert support world-wide
- Comprehensive scripting support with shallow learning curve

Comprehensive support for Arm cores

- Arm Development Studio is developed alongside Arm cores
- Debug support for cores and architectural extensions is developed as soon as it is practical
- Development Studio is used as part of the normal core development and validation process
- Debugger testing is carried out against Fast Models and early FPGA implementations of all Arm cores
- Tools available for leading-edge partners working with future cores which are not yet public



Advanced User Interface Design

Simplicity, visibility, and performance

Register Set: My Registers

Name	Value	Size	Access
Core 5 of 47 registers			
R0	0x000000AA	32	R/W
R1	0x000001CC	32	R/W
R2	0x00000000	32	R/W
SP	0x800C9578	32	R/W
PC	0x80000486	32	R/W
CP15 3 of 134 registers			
MMU 3 of 32 registers			
TLBTR	0x00000400	32	RO
TTBR0	0x80500000	32	R/W
TTBR1	0x00000000	32	R/W

Breakpoints Registers OS Data: Tasks Expressions Functions

Name	Value	Type	Size	Location	Access
UART0->UARTFR & 0x10	16	volatile const uint32_t	32		RO
UART0->UARTDR	0	volatile uint32_t	32	S:0x1C090000	R/W
line	""	char*	32	\$R5	R/W
cnt	0	int	32	\$R6	R/W
Enter new expression here					

Stack

Function Prototype	Source/Line
<code>_svcMessageGet()</code>	rt_CMSIS.c:1817
<code>osMessageGet()</code>	rt_CMSIS.c:1817
<No Function Name>	

Name	Value	Type	Count	Size	Location	Access
a1	0x8030611C	osMessageQId	1	32	\$R4	R/W
a2	4294967295	uint32_t		32	\$R5	R/W

Debug Cont... Project Expl... Remote Syst...

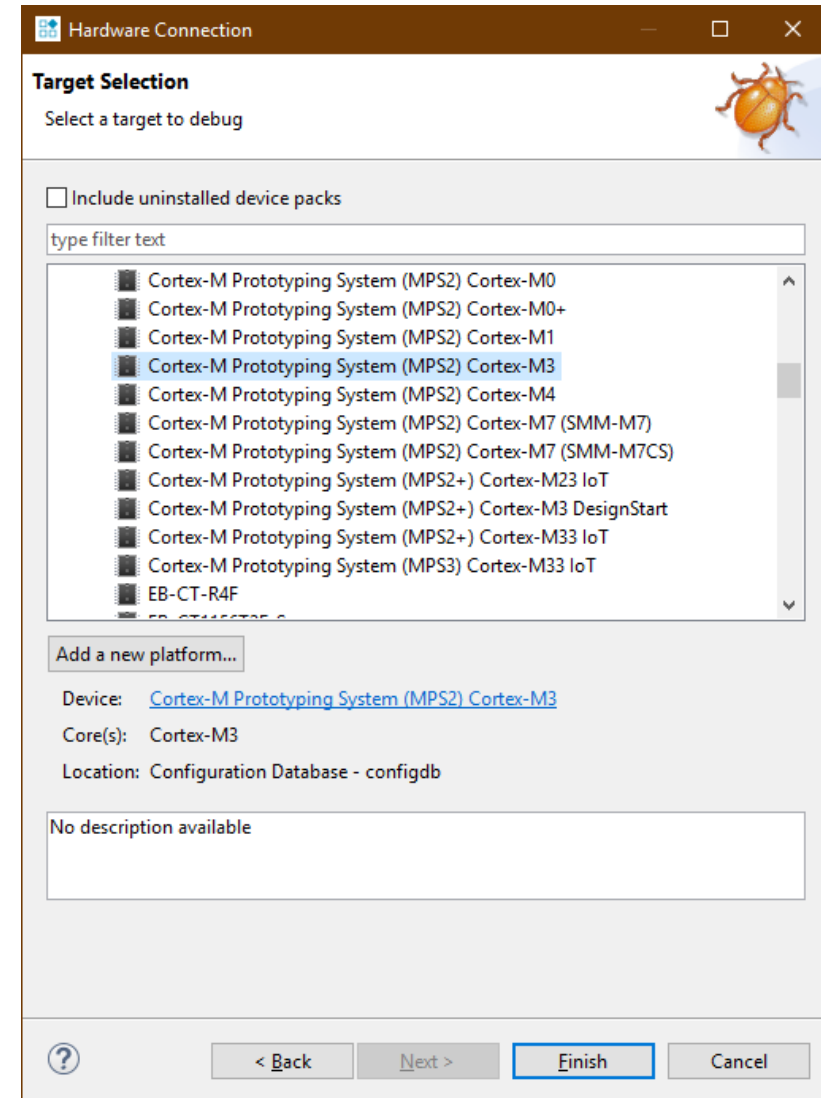
Big Snapshot connected

- swapper/1 #6 stopped (PID 0)
- Active Threads
 - swapper/1 #6 stopped (PID 0)
 - swapper/2 #7 stopped (PID 0)
 - swapper/0 #8 stopped (PID 0)
 - swapper/3 #9 stopped (PID 0)
 - sh #10 stopped (PID 1355)
 - swapper/5 #11 stopped (PID 0)
- All Threads
 - CSAL disconnected
 - Generic-Snapshot-Juno disconnected
 - Generic-Snapshot-Snowball disconnected

Status: connected OS Support: Enabled

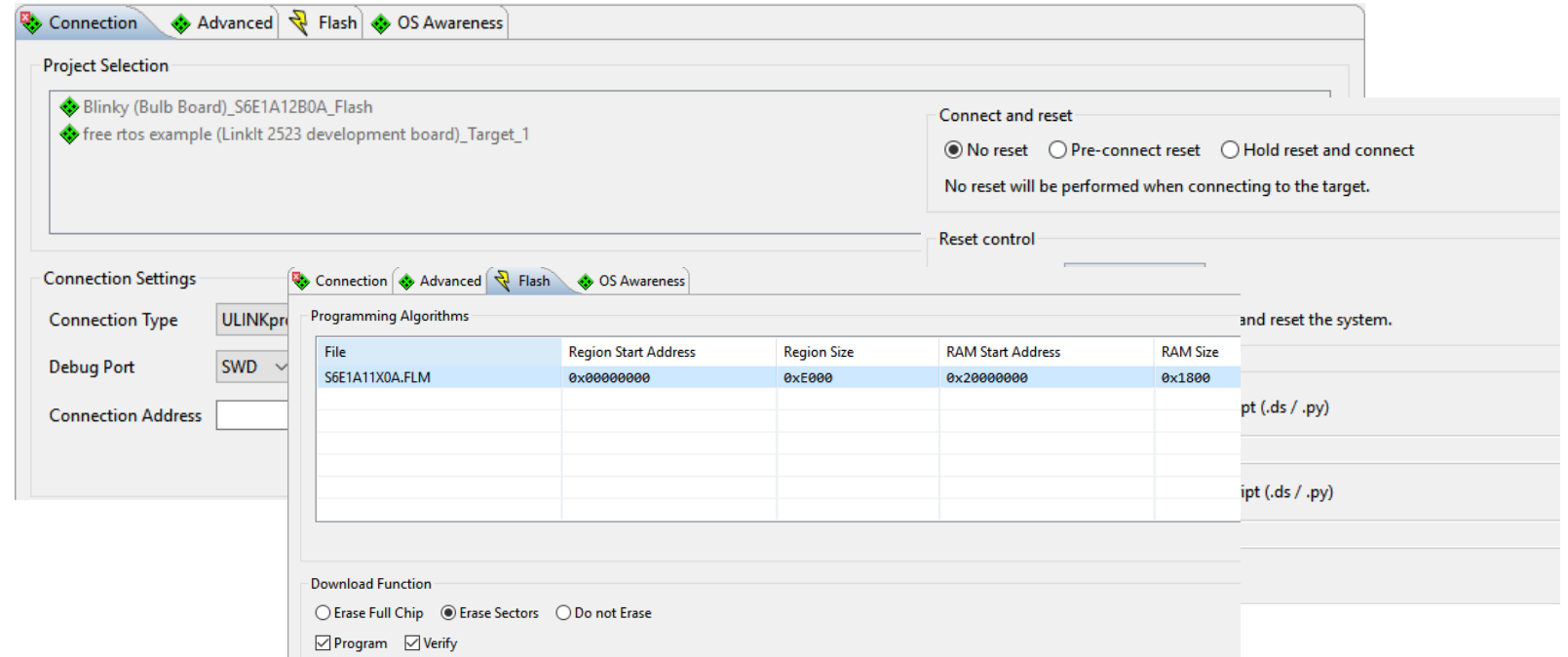
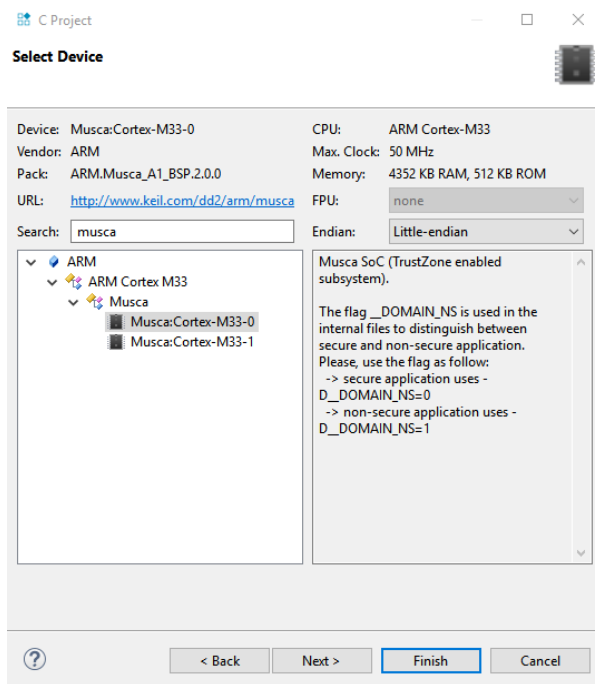
Target connection configuration

- Hardware and models connection wizard
- Select device, board or FVP easily
 - Recently used list
 - Add new platform if device not listed
 - Creates a new configuration with the platform configuration editor
- Configuration wizard generates an initial debugger configuration based on device information.



Debugger configuration with software packs

Device selection based on packs



Advanced target connection, reset options

Flash programming with support for multiple flash regions

Trace – how it works

- CPU instruction execution history via ETM
 - Data load and store (certain CPUs)
 - Instrumented trace (ITM, STM)
- Arm trace data collected in
 - On-chip CoreSight trace buffer (ETB, ETF, ETR)
 - Off-chip debug probe via physical trace port on the target
- Rotating trace buffer fills in the background during a debug session
 - When a breakpoint is hit the content of the trace buffer is displayed

Trace view in Arm Debugger

A non-intrusive 'flight recorder' for your system

The screenshot displays the Arm Debugger interface. On the left, the source code for `plot3` and `plot` functions is shown. The `plot` function is highlighted in blue, and the `s->age--;` line is highlighted in green. On the right, the Trace view shows a per-CPU timeline with a function profile. The profile lists functions and their execution percentages: `plot3` (45.98%), `moveSpark` (29.24%), `plot` (13.45%), `drawSparks` (5.39%), `rand` (2.72%), `spray1` (1.80%), `newSpark` (0.64%), `random` (0.64%), and `spray` (0.14%). Below the profile is a reconstructed instruction trace table.

Index	Address	Opcode	Detail
-6,310	S:0x8000052E	F1B07F16	CMP r0, #0x258000
-6,309	S:0x80000532	DA04	BGE moveSpark+48 ; 0x8000053E
-6,308	S:0x80000534	1401	ASRS r1, r0, #16
-6,307	S:0x80000536	1410	ASRS r0, r2, #16
-6,306	S:0x80000538	2200	MOVS r2, #0
-6,305	S:0x8000053A	F7FFFFFFE	BL plot ; 0x800004DA
-6,304	S:0x800004DA	B570	plot
-6,303	S:0x800004DC	4605	PUSH {r4-r6, lr}
-6,302	S:0x800004DE	460C	MOV r5, r0
-6,301	S:0x800004E0	4616	MOV r4, r1
-6,300	S:0x800004E2	2C00	MOV r6, r2
-6,299	S:0x800004E4	DD04	CMP r4, #0
-6,298	S:0x800004E6	1E61	BLE plot+22 ; 0x800004F0
-6,297	S:0x800004E8	4632	SUBS r1, r4, #1
-6,296	S:0x800004EA	4628	MOV r2, r6
-6,295	S:0x800004EC	F7FFFFFFD3	MOV r0, r5
-6,294	S:0x80000496	B292	
-6,293	S:0x80000498	EB0103C1	
-6,292	S:0x8000049C	2800	
-6,291	S:0x8000049E	EB031101	
-6,290	S:0x800004A2	EB001141	
-6,289	S:0x800004A6	DD06	
-6,288	S:0x800004A8	1E48	
-6,287	S:0x800004AA	0058	
-6,286	S:0x800004AC	F1A343C0	LSLS r3, r3, #1
-6,285	S:0x800004B0	F1A53FD	SUB r3, r3, #0x60000000
			SUB r3, r3, #0x1fa00000

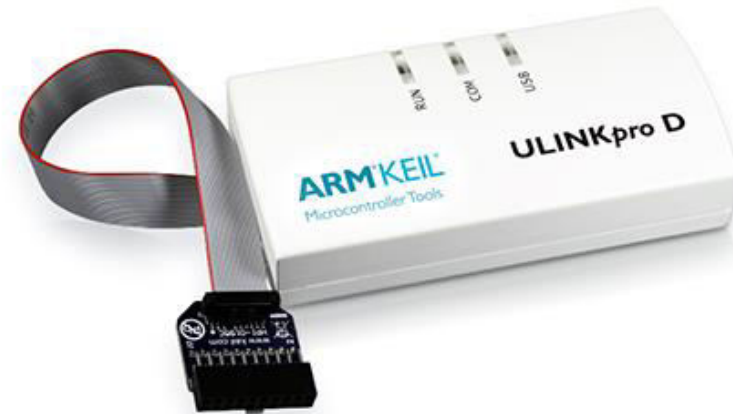
Source code highlighting, synchronized with Trace view

Per-CPU timeline shows order of execution and indicative function profile

Reconstructed instruction trace

Debug probes

Range of probes to match your requirements



DSTREAM-ST family

Second-generation debug and trace probes

- New design featuring patented technology to address the needs of modern SoC's

Capabilities	DSTREAM-ST	DSTREAM-PT	DSTREAM-HT
JTAG / SWD	180 / 125 MHz	180 / 125 MHz	180 / 125 MHz
Download speed	Up to 12MB/s	Up to 12MB/s	Up to 12MB/s
Maximum CoreSight components	1022	1022	1022
Parallel trace pins	1 - 4	1 - 32	1 - 4
High-speed serial trace			Yes
Maximum trace bandwidth	1.6Gb/s	19.2Gb/s	60Gb/s
Host connectivity	USB 3.0 Gigabit Ethernet	USB 3.0 Gigabit Ethernet	USB 3.0 Gigabit Ethernet
On-probe trace store	Stream to host only	8GB	8GB

ULINK family



Simplicity

Just select a supported platform from the database and connect

Consistent debugging experience across probes

Cost efficiency

Perfectly engineered to meet basic debug needs

Supported by both Keil and Arm Development Studio tools to protect your investment

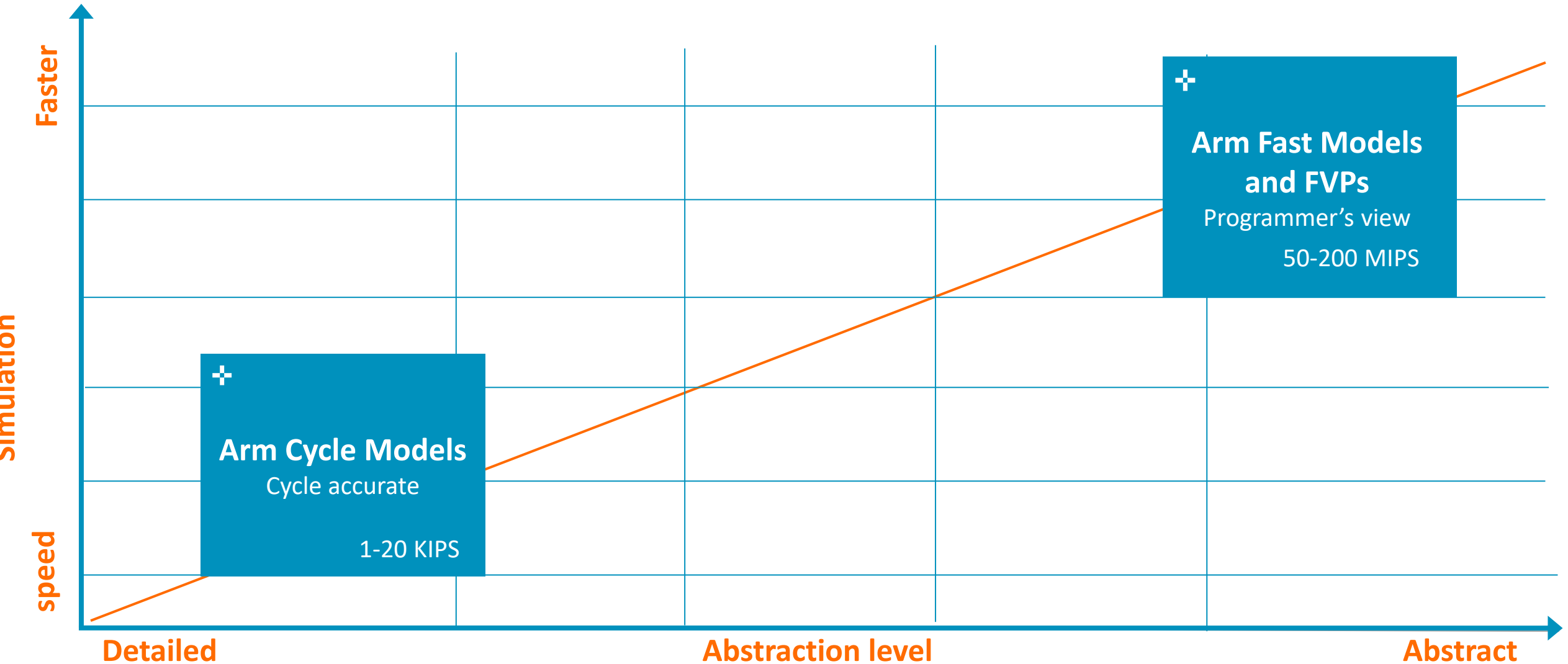
arm

Virtual platforms

Virtual prototyping solutions

- Fast Models
 - Instruction accurate, suitable for software development and test
- Cycle Models
 - Timing accurate models, derived from actual RTL
- Fully validated models of Arm IP
- Collection of Fixed Virtual Platforms (FVP) provided with Development Studio
 - Complete system models based on Fast Model technology

Virtual prototype solutions



arm

Arm Streamline
performance
analyzer



Speed Up Your Code

- Find out where the CPU is spending the most time
- Tune code for optimal cache usage

Arm Streamline Performance Analyzer

OpenCL™ Visualizer

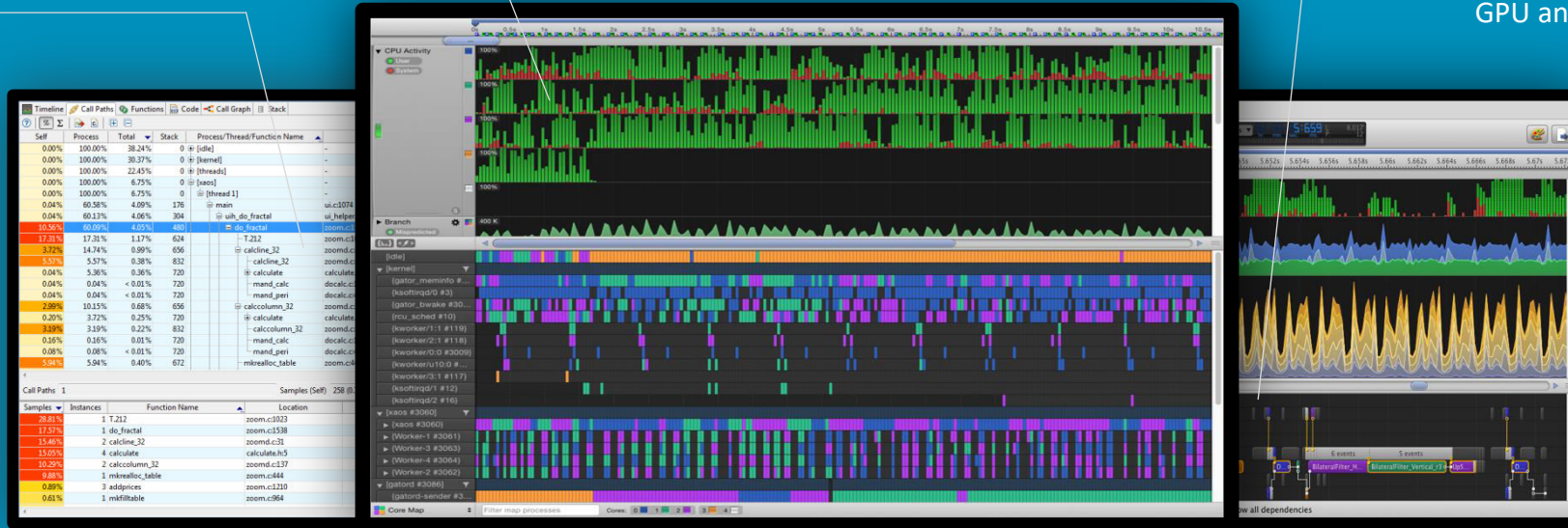


Visualization of OpenCL dependencies, helping you to balance resources between GPU and CPU better than ever



Drill down to the Source Code

- Break performance down by function
- View it alongside the disassembly



Mali GPU Support

- Analyze and optimize Mali™ GPU utilization
- Monitor CPU and GPU cache usage



Optimize energy efficiency

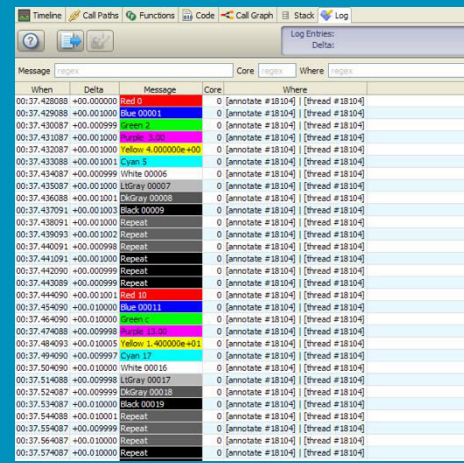
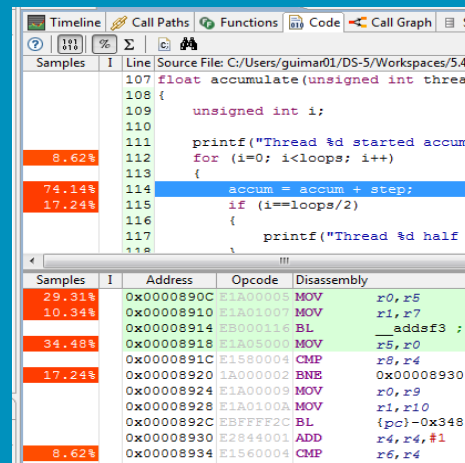
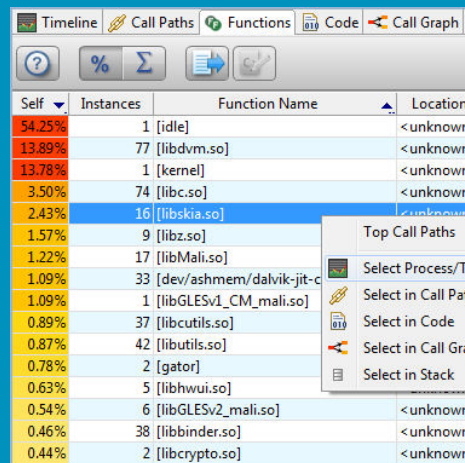
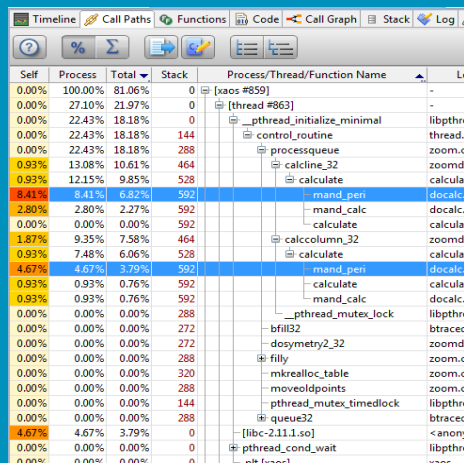
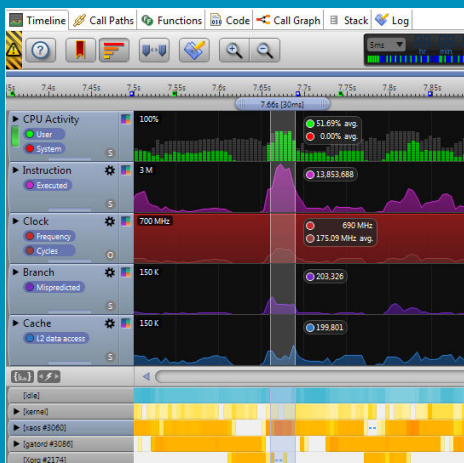
- Monitor actual power consumption with the Arm Energy Probe
- Correlate software execution to actual power consumption



Customize it for Your System

- Flexible architecture permits easy addition of new counters
- Open-source driver and daemon gives developers ultimate flexibility

Process to Success



Timeline

Visualization of system performance metrics, software profile and system events over time

Call Paths

Hierarchical profile table, aggregating samples per process, thread, and function call chain

Functions

Flat software profile table, listing shared libraries and function hotspots

Code

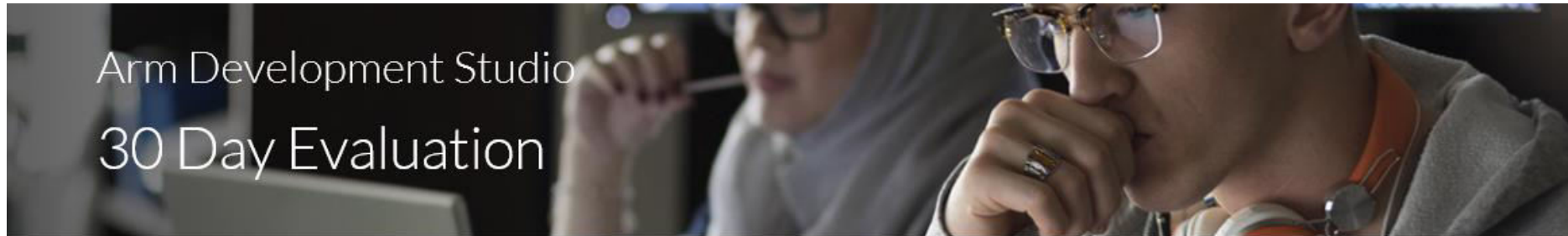
Source and instruction level profile. Color coded source code lines for easy identification

Log

Chronologic list of text and graphic user annotations sent to Streamline, offering flexible filtering

We invite you to try it out for yourself

For more information see arm.com/development-studio



You're One Step Away from Starting Your 30 Day Free Trial

Every feature, no credit card payment required

Try for yourself every component of Arm Development Studio Gold for free for 30 days – build, simulate, debug and optimize pre-packaged examples or your own code to experience hands-on how Arm's flagship toolchain helps you achieve your design goals more efficiently.

Get started today.

Evaluate for Windows

Evaluate for Linux