



Arm[®] Streamline

Version 8.5

Target Setup Guide for Linux

Non-Confidential

Copyright © 2021–2023 Arm Limited (or its affiliates).
All rights reserved.

Issue 00

101814_0805_00_en



Arm® Streamline Target Setup Guide for Linux

Copyright © 2021–2023 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
0708-00	20 August 2021	Non-Confidential	New document for v7.8
0709-00	18 November 2021	Non-Confidential	New document for v7.9
0800-00	18 February 2022	Non-Confidential	New document for v8.0
0801-00	20 May 2022	Non-Confidential	New document for v8.1
0802-00	19 August 2022	Non-Confidential	New document for v8.2
0803-00	17 November 2022	Non-Confidential	New document for v8.3
0804-00	14 February 2023	Non-Confidential	New document for v8.4
0805-00	20 April 2023	Non-Confidential	New document for v8.5

Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has

undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at <https://www.arm.com/company/policies/trademarks>.

Copyright © 2021–2023 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm® welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on <https://support.developer.arm.com>

To provide feedback on the document, fill the following survey: <https://developer.arm.com/documentation-feedback-survey>.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Introduction.....	6
1.1 Conventions.....	6
1.2 Useful resources.....	7
1.3 Other information.....	7
2. Target Setup.....	8
2.1 Introduction to using Streamline with a Linux device.....	8
2.2 Compile your application.....	8
2.3 Set up your target.....	10
2.4 System requirements for SPE support.....	11
2.5 System requirements for Mali Timeline support.....	11
2.6 gatord.....	12
2.6.1 gatord command-line options.....	13
2.7 Capture data locally on a target.....	17
2.8 Support for Arm CoreLink tools.....	17
2.9 Using Streamline with Arm NN.....	17
3. Customize your Streamline report.....	19
3.1 Add a CPU to Streamline.....	19
3.2 Add an uncore peripheral to Streamline.....	19
3.3 Contributing CPU and peripheral support.....	20
4. Advanced target setup information (Linux).....	21
4.1 Kernel configuration menu options.....	21
4.2 Profile the Linux kernel.....	22
4.3 Build gatord yourself.....	23
4.4 Add support to gatord for a new CPU or perf PMU.....	23

1. Introduction

Learn how to use Streamline to set up your Linux device for debugging and profiling.

1.1 Conventions

The following subsections describe conventions used in Arm documents.





Glossary



The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm® Glossary for more information: developer.arm.com/glossary.

Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
<i>italic</i>	Citations.
bold	Interface elements, such as menu names. Terms in descriptive lists, where appropriate.
monospace	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
monospace <u>underline</u>	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: <pre>MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2></pre>
SMALL CAPITALS	Terms that have specific technical meanings as defined in the <i>Arm® Glossary</i> . For example, IMPLEMENTATION DEFINED , IMPLEMENTATION SPECIFIC , UNKNOWN , and UNPREDICTABLE .
 Caution	Recommendations. Not following these recommendations might lead to system failure or damage.
 Warning	Requirements for the system. Not following these requirements might result in system failure or damage.
 Danger	Requirements for the system. Not following these requirements will result in system failure or damage.
 Note	An important piece of information that needs your attention.

Convention	Use
 Tip	A useful tip that might make it easier, better or faster to perform a task.
 Remember	A reminder of something important that relates to the information you are reading.

1.2 Useful resources

This document contains information that is specific to this product. See the following resources for other useful information.

Access to Arm documents depends on their confidentiality:

- Non-Confidential documents are available at developer.arm.com/documentation. Each document link in the following tables goes to the online version of the document.
- Confidential documents are available to licensees only through the product package.

Arm product resources	Document ID	Confidentiality
Arm Development Studio Getting Started Guide	101469	Non-Confidential
Arm Streamline User Guide	101816	Non-Confidential

1.3 Other information

See the Arm website for other relevant information.

- [Arm® Developer](#).
- [Arm® Documentation](#).
- [Technical Support](#).
- [Arm® Glossary](#).

2. Target Setup

This chapter explains how to set up your target and host devices ready to use Streamline for application or system profiling.

2.1 Introduction to using Streamline with a Linux device

Streamline supports system-wide profiling of applications running on Linux-based embedded devices. Analyze the behavior of the system hardware by selecting the required Arm CPU, Arm® Mali™ GPU, or Arm Immortalis™ GPU hardware performance counters for your scenario.



Streamline's Linux device features are license-managed. For more information, see [Licenses](#) in the [Arm Streamline User Guide](#).

This analysis can be supplemented by connecting power measurement probes, such as the Arm Energy Probe or National Instruments DAQ, to provide accurate measurement of system energy use. To provide more context to the analysis, you can use software annotations in Arm software libraries, such as the Mali GPU OpenCL device driver.

Use the information in this guide to prepare your Linux target, ready for profiling with Streamline. Then refer to the [Arm Streamline User Guide](#) for details about working with Streamline.

2.2 Compile your application

Before you can profile your executable with Streamline, you must compile your executable. This topic describes the compiler options to use when you compile your application.

Recommended compiler options

When compiling with GCC or Clang, use the following options:

-g

Turns on the debug symbols necessary for quality analysis reports.

-fno-inline

Disables inlining and substantially improves the call path quality.

-fno-omit-frame-pointer

Compiles your EABI images and libraries with frame pointers. This option enables Streamline to record the call stack with each sample taken.

-mno-omit-leaf-frame-pointer

Keeps the frame pointer in leaf functions.

-marm

When building for AArch32, if GCC was compiled with the `--with-mode=thumb` option enabled, this `-marm` is required. Using `--with-mode=thumb` without `-marm` breaks call stack unwinding in Streamline.

Optional compiler options for call stack unwinding

To enable call stack unwinding in Streamline, you need to compile your executable with some additional compiler options:

- For AArch64 applications:
 - Compiling with GCC, use: `-fno-omit-frame-pointer` and `-mno-omit-leaf-frame-pointer`



Arm recommends using `-mno-omit-leaf-frame-pointer` to prevent samples in leaf functions incorrectly listing their grand-parent function as their parent.

- Compiling with Clang, use: `-fno-omit-frame-pointer`



`-mno-omit-leaf-frame-pointer` is not supported on Clang.

- For AArch32 applications, compiling with either GCC or Clang, use: `-fno-omit-frame-pointer`, `-marm`, and `-mapcs-frame`.



Streamline supports call stack unwinding for code that has been generated by Arm® Compiler 6.

Streamline does not support call stack unwinding for T32 (Thumb®) code.

Source code annotations

To enable Streamline to provide extra context when profiling your executable, you can add annotations to your source code. Streamline supports two types of annotations:

- User space annotations, for annotating your application
- Kernel annotations, to profile system calls

You can read more information about annotating your code in the [Annotate your code](#) chapter of the *Arm Streamline User Guide*.

2.3 Set up your target

To profile your software using Streamline, you must have a suitably configured Linux kernel and `gator` running on an Arm®-based hardware target.

Before you begin

- Download Arm Development Studio. See Arm Development Studio [Downloads](#).
- Install Arm Development Studio. Follow the instructions in the [Arm Development Studio Getting Started Guide](#).
- If you intend to use the Statistical Profiling Extension (SPE) in Streamline, there are some additional system requirements. Refer to [System requirements for SPE support](#).

Procedure

1. Validate the Linux kernel configuration.
The kernel configuration must include the options that are described in [Kernel configuration menu options](#). You can usually find your kernel configuration in `/proc/config.gz`. If this file is not visible, depending how your system is configured, you can create it by running:

```
sudo modprobe configs
```

The following command confirms whether an option is enabled:

```
zcat /proc/config.gz | grep <OPTION>
```

For example:

```
zcat /proc/config.gz | grep CONFIG_PROFILING
```

2. Install `gator` on your target.
Two pre-built `gator` binaries are available:
 - For Armv7 targets, and Armv8 targets that support AArch32 execution state.
 - For Armv8 AArch64 targets.

The pre-built `gator` binaries are available from the Arm Development Studio installation directory `<install_directory>/sw/streamline/bin/linux/`

The source code for `gator` is available from the following locations:

- The Arm Development Studio installation directory `<install_directory>/sw/streamline/gator/daemon/`
- <https://github.com/ARM-software/gator>. This site is the official distribution channel for all `gator` releases, and contains the latest source updates between Arm Development Studio releases.

The version of `gator` running on the target must be compatible with the version of Streamline that you are using. Use the same version of `gator` as the version of Streamline.

For more information, see [gatord](#).

3. To ensure `gatord` has execute permission, enter the following command:

```
chmod +x gatord
```

4. Ensure that you have root privileges, if necessary, then enter the following to execute `gatord`:

```
./gatord
```

By default, `gatord` uses port 8080 for communication with the host, but you can specify the port by launching `gatord` with the port number as a parameter. For example:

```
./gatord -p 5050
```

If the target runs a firewall that prevents Streamline from connecting to `gatord`, Arm recommends that you change the firewall settings to allow connections to `gatord`.

Next steps

- Capture a profile of your application as described in [Capture a Streamline profile](#).
- To stop `gatord` at the end of a capture, press Ctrl+C.

2.4 System requirements for SPE support

When using the Statistical Profiling Extension (SPE) support in Streamline, there are some more system requirements.

- Use Linux kernel 4.16 or later. To confirm that support is available, check for the path `/sys/bus/event_source/devices/arm_spe_XX`.
- Disable kernel page table isolation for the target. To ensure that kernel page table isolation is disabled, boot the device with the command-line argument `kpti=off`.
- For kernel versions 4.20 and later, apply the patch at <http://lkml.iu.edu/hypermail/linux/kernel/1903.3/06760.html>, or upgrade to a kernel version greater than or equal to 5.1-RC5.

For more information about SPE, see [Configure SPE counters](#) in the *Arm Streamline User Guide*.

2.5 System requirements for Mali Timeline support

Arm® Mali™ Timeline Events are supported on Linux from DDK version r43p0. This topic details some more system requirements to support Mali Timeline Events.

Enabling support for the Mali Timeline requires the following actions:

- Manually set the environment variable `MALI_GPU_RENDERSTAGES_ENABLE=1`

- Compile the driver with instrumentation and profiling support, and ensure the Perfetto `traced` daemon is running.
- To start profiling, configure the environment variable before starting the target application. As it starts, the driver checks the variable to see if it needs to start a profiling session and publish information to the Perfetto daemon.

For more information about Mali Timeline Events, see [Enable Mali Timeline Events](#) in the *Arm Streamline User Guide*.

2.6 gatord

To communicate with the target, Streamline requires a daemon, called `gatord`, to be installed and running on the target. `gatord` must be running before you can capture trace data.

`gatord` reads and processes data from applications that are running on the target. When running a headless capture, it then creates a directory, whose name ends in `.apc`, containing the capture data.

`gatord` uses the `perf` API to collect performance information. You can run `gatord` on a device with or without root permissions. Running `gatord` on a device with root permissions supports system-wide data collection, including profiling all applications and the kernel. When run as non-root user, `gatord` is limited to capturing processes that you have permission to profile. These processes are usually running as the same user.



System-wide data collection using non-root `gatord` falls back to `/proc` polling, which is deprecated. To perform system-wide data collection, use root `gatord`.

`gatord` has some restrictions, for example:

- It polls the following Linux counters every 100ms, instead of every 1ms or when they change because files in the `/proc` or `/sys` filesystem are read:
 - Memory
 - Disk I/O
 - Network

This rate is fixed and overrides the sample rate that is specified in the **Capture and Analysis Options** dialog box.

- When using `gatord`, the `Memory: Used` counter does not contain per-process information. As a result, memory statistics are not available in **Processes** mode.

2.6.1 gatord command-line options

`gatord` must be running before you can capture trace data. The command-line options configure how `gatord` captures events and how it communicates with Streamline running on your host.

`gatord` has two modes of operation:

Daemon mode (the default mode)

Sends captured events to a host running Streamline.

Local capture mode

Writes the capture to a file then exits.

To enable this mode, specify an output directory with the `--output` flag.

Arguments available to all modes:

Table 2-1: gatord arguments available to all modes

Option	Description
<code>-h, --help</code>	Lists all the available <code>gatord</code> command-line options.
<code>-c, --config-xml <config_xml></code>	Specify the path and filename of the <code>configuration.xml</code> file that defines the capture options. In daemon mode, the list of counters is written to this file. In local capture mode, the list of counters is read from this file.
<code>-e, --events-xml <events_xml></code>	Specify the path and filename of the <code>events.xml</code> file. <code>events.xml</code> defines all the counters that Streamline collects during the capture session.
<code>-E, --append-events-xml <events_xml></code>	Specify the path and filename of the <code>events.xml</code> file to append.
<code>-P, --pmu-xml <pmu_xml></code>	Specify the path and filename of the <code>pmu.xml</code> file to append.
<code>-v, --version</code>	Print version information.
<code>-d, --debug</code>	Enable debug messages.
<code>-A, --app <cmd> <args...></code>	Specify the command to execute when the capture starts. This argument must be the last argument that is passed to <code>gatord</code> . All subsequent arguments are passed to the launched application.
<code>-k, --exclude-kernel <yes no></code>	Specify whether to filter out kernel events from the perf results.
<code>-S, --system-wide <yes no></code>	Specify whether to capture the whole system. In daemon mode, <code>no</code> is only applicable when <code>--allow-command</code> is specified. In this mode, you must enter a command in the Start view . Requires kernel events to be enabled in Capture and Analysis Options of Streamline , or by setting <code>--exclude-kernel no</code> . Defaults to <code>yes</code> , unless <code>--app</code> , <code>--pid</code> , or <code>--wait-process</code> is specified.
<code>-u, --call-stack-unwinding <yes no></code>	Enable or disable call stack unwinding. Defaults to <code>yes</code> .

Option	Description
<code>-r, --sample-rate <none low normal high></code>	Specify the sample rate for the capture. The frequencies for each sample rate are: <ul style="list-style-type: none"> <code>high</code> = 10kHz <code>normal</code> = 1kHz <code>low</code> = 100Hz <code>none</code> = the lowest possible rate Defaults to <code>normal</code> .
<code>-t, --max-duration <s></code>	Specify the maximum duration that the capture can run for in seconds. Defaults to 0, which means unlimited.
<code>-f, --use-efficient-ftrace <yes no></code>	Enable efficient ftrace data collection mode. Defaults to <code>yes</code> .
<code>-w, --app-cwd <path></code>	Specify the working directory for the application that <code>gator</code> launches. Defaults to the current directory.
<code>-x, --stop-on-exit <yes no></code>	Stop the capture when the launched application exits. Defaults to <code>no</code> , unless <code>--app</code> , <code>--pid</code> , or <code>--wait-process</code> is specified.
<code>-Q, --wait-process <command></code>	Wait for a process that matches the specified command to launch before starting the capture. Attach to the specified process and profile it.
<code>-Z, --mmap-pages <n></code>	The maximum number of pages to map per mmaped perf buffer is equal to <code><n+1></code> . <code>n</code> must be a power of two.
<code>-O, --disable-cpu-onlining <yes no></code>	To not switch on CPU cores that are offline to read their information. This option is useful for kernels that fail to handle this action correctly, for example they reboot the system. Defaults to <code>no</code> .
<code>-F, --spe-sample-rate <n></code>	Specify the SPE periodic sampling rate. The rate, <code><n></code> , is the number of operations between each sample, and must be a nonzero positive integer. The hardware specifies the minimum rate. Values below this threshold are ignored and the hardware minimum is used instead.
<code>--smmuv3-model <model_id> <iidr></code>	Specify the SMMUv3 model. You can specify the model ID string directly, such as <code>mmu-600</code> , or the hex value representation for the model's IIDR number either fully, such as <code>4832243b</code> , or partially, such as <code>483_43b</code> .

Arguments available in daemon mode only:

Table 2-2: gatord arguments available in daemon mode only

Option	Description
<code>-p, --port <port_number> uds</code>	<p>Set the port number that <code>gatord</code> uses to communicate with the host. The default is 8080.</p> <p>If you use the argument <code>uds</code>, the TCP socket is disabled and an abstract Unix domain socket is created. This socket is named <code>streamline-data</code>. If you use Android, creating a Unix domain socket is useful because <code>gatord</code> is usually prevented from creating a TCP server socket.</p> <p>Alternatively, you can connect to <code>localhost:<local_port></code> in Streamline using:</p> <pre>adb forward tcp:<local_port> localabstract:streamline-data</pre>
<code>-a, --allow-command</code>	<p>Allows you to run a command on the target during profiling. The command is specified in the Start view.</p> <p>Caution: If you use this option, an unauthenticated user could run arbitrary commands on the target using Streamline.</p>

Arguments available to local capture mode only:

Table 2-3: gatord arguments available to local capture mode only

Option	Description
<code>-s, --session-xml <session_xml></code>	Specify the <code>session.xml</code> file that the configuration is taken from. Any additional arguments override values that are specified in this file.
<code>-o, --output <apc_dir></code>	Specify the path and filename of the output directory for a local capture. The directory path will be appended with the extension <code>.apc</code> if it is not already the case.
<code>-i, --pid <pids...></code>	A comma-separated list of process IDs to profile.
<code>-C, --counters <counters></code>	<p>A comma-separated list of counters to enable. You can specify this option multiple times. An event code and a slot identify most hardware counters. To specify the counter for a particular slot, pass:</p> <pre>--counters <device>_cnt<s>:<e></pre> <p>Where:</p> <ul style="list-style-type: none"> <code><device></code> is the prefix that identifies the device type. <code><s></code> is the slot number. <code><e></code> is the event code.

Option	Description
<code>-X, --spe <id>[:events=<indexes>][:ops=<types>][:min_latency=<lat>]</code>	<p>Enable the Statistical Profiling Extension (SPE).</p> <p>Where:</p> <ul style="list-style-type: none"> <code><id></code> is the name of the SPE properties that are specified in the <code>events.xml</code> or <code>pmus.xml</code> file. It uniquely identifies the available events and counters for the SPE hardware. <code><indexes></code> is a comma-separated list of event indexes to filter the sampling by. A sample is only recorded if all events are present. <code><types></code> is a comma-separated list of operation types to filter the sampling by. If a sample is any of the types in <code><types></code>, it is recorded. Valid types are <code>LD</code> for load, <code>ST</code> for store and <code>B</code> for branch. <code><lat></code> is the minimum latency. A sample is only recorded if its latency is greater than or equal to this value. The valid range is [0,4096).

Argument usage examples

- Use `--pmus-xml` and `--append-events-xml` to add support for a new PMU without having to rebuild `gatord`.

`--pmus-xml` specifies an XML file that defines a new PMU to add to the list of PMUs that `gatord` has built-in support for. The list of built-in PMUs is defined in `pmus.xml`, which is in the `gatord` source directory.

`--append-events-xml` specifies an XML file that defines one or more event counters to append to the `events.xml` file. This option allows you to add new events to `gatord` without having to rebuild `gatord` or to entirely replace `events.xml`.

The `events.xml` file must include the XML header and elements that are shown in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<events>
  <category name="Filesystem">
    <event counter="filesystem_loginuid" path="/proc/self/loginuid"
      title="loginuid" name="loginuid" description="loginuid"/>
  </category>
</events>
```

- The Instructions Executed counter is configured in slot 0 as:

```
--counters ARMv8_Cortex_A53_cnt0:0x08
```

To configure the cycle counter, specify `--counters <device>_ccnt`. For example:

```
--counters ARMv8_Cortex_A53_ccnt
```


Other counters do not have event codes and are identified only by name. For example:

```
--counters PERF_COUNT_SW_PAGE_FAULTS
```

2.7 Capture data locally on a target

You can run `gator` from the command line to configure the counters and capture data locally on your target.

Procedure

Start `gator` specifying:

- The location for the output file with `--output`.
- Optionally specify which counters to use with either `--config-xml` or `--counters`.

```
./gator --output <path> --counters <counters>
```

Results

`gator` captures a profile of all the processes running on your system, and writes the data to the specified location.

Next steps

Copy the `.apc` directory into the Streamline capture directory. This directory is usually `<Home>/Documents/Streamline`.

Related information

[Creating a configuration.xml file](#)

2.8 Support for Arm CoreLink tools

Streamline supports various Arm® CoreLink™ interconnect parts, including: CCI-400, CCI-500, CCN, CMN-600, L2C-310, SMMUv3, and DSU.

Profiling these devices relies on support for the correct PMU driver being present and enabled in the Linux kernel. Ensure you are using a version that supports your hardware, and that you have the correct configuration options enabled in the kernel.

2.9 Using Streamline with Arm NN

Streamline supports capturing profiling information from Arm NN enabled applications. This feature enables you to collect performance counters from supported Arm® Ethos™ NPUs and performance

data from the Arm NN library. To profile an Arm NN application, `gator_d` must first see which events are available in the running application. Streamline can then capture a profile as normal.

Before you begin

- Build your application against Arm NN version 20.08 or later. See <https://github.com/ARM-software/armnn> for more information.
- When you create the runtime in your application code, you must set `options.m_ProfilingOptions.m_EnableProfiling = true` for the options passed to `IRuntime::Create`.
- Enable timeline profiling by setting `options.m_ProfilingOptions.m_TimelineEnabled = true`.

```
// Create Arm NN runtime

IRuntime::CreationOptions options; // default options
options.m_ProfilingOptions.m_EnableProfiling = true;
options.m_ProfilingOptions.m_TimelineEnabled = true;
IRuntimePtr run = IRuntime::Create(options);
```

Procedure

1. Start `gator_d` in daemon mode as described in [Set up your target](#).



Local capture mode is not available because `gator_d` does not know which events are available yet.

2. Run your application.
3. Configure counters and [Capture a Streamline profile](#) as described in the *Arm Streamline User Guide*.



To get the timeline data, you must start your application again after starting the capture.

3. Customize your Streamline report

This chapter explains how to add support for a new CPU, or an uncore peripheral outside of the CPU, to Streamline.

3.1 Add a CPU to Streamline

Customize your Streamline reports with support for custom CPU Performance Monitoring Units (PMUs).

Procedure

1. Create an `events-<processor_name>.xml` file and save it in `<DS_install_directory>/sw/streamline/gator/daemon/`.
2. Include the following in the file:

```
<counter_set name="<ID>_cnt" count="<NEVENTS>" />
<category name="<NAME>" counter_set="<ID>_cnt" per_cpu="no">
  <event .../ >
</category>
```

`<ID>` is a string that can contain letters, numbers, and underscores.

`<NEVENTS>` is the maximum number of counters in the hardware.

If the number of events is unlimited, the `<counter_set>` element and the `counter_set="<ID>_cnt"` attribute are not required.

3. Add a `<pmu>` element to `pmus.xml`.
See the comment at the top of the `pmus.xml` file for the required attributes.
4. Rebuild `gator`.

3.2 Add an uncore peripheral to Streamline

Customize your Streamline reports with support for custom uncore peripheral Performance Monitoring Units (PMUs).

Procedure

1. Create an `events-<peripheral_name>.xml` file and save it in `<DS_install_directory>/sw/streamline/gator/daemon/`.
2. Include the following in the file:

```
<counter_set name="<ID>_cnt" count="<NEVENTS>" />
<category name="<NAME>" counter_set="<ID>_cnt" per_cpu="no">
  <event .../ >
</category>
```

<ID> is a string that can contain letters, numbers, and underscores.

<NEVENTS> is the maximum number of counters in the hardware.

If the number of events is unlimited, the <counter_set> element and the counter_set="<ID>_cnt" attribute are not required.

3. Add an <uncore_pmu> element to pmus.xml.
See the comment at the top of the pmus.xml file for the required attributes.
4. Rebuild gator.

3.3 Contributing CPU and peripheral support

The gator daemon is an open-source component, licensed under the GPLv2 license, and published on GitHub. We are willing to accept pull requests from external developers to add support for custom Arm cores and supporting uncore peripherals.

The gator daemon is in this repository <https://github.com/ARM-software/gator/>, where you can also find the [contributing guidelines](#).

4. Advanced target setup information (Linux)

This appendix provides extra configuration information beyond the standard setup.

4.1 Kernel configuration menu options

You must enable certain kernel configuration options to run Streamline.

The following `menuconfig` menus have options that are required for Streamline:



Note

- If these options are not set correctly, you must change them and rebuild your kernel. If they are set correctly, you are ready to build and install the gator driver.
- The location of these options might change between releases. If so, use the search option in `menuconfig` to find them.
- Extra options are required to enable Arm® Mali™ GPU support.

General Setup

Enable the **Profiling Support** option `CONFIG_PROFILING`, and the **Kernel performance events and counters** option `CONFIG_PERF_EVENTS`. `CONFIG_PERF_EVENTS` is required for kernel versions 3.0 and later. Enable the **Timers subsystem > High Resolution Timer Support** option `CONFIG_HIGH_RES_TIMERS`.

Kernel Features

The **Enable hardware performance counter support for perf events** option `CONFIG_HW_PERF_EVENTS`. `CONFIG_HW_PERF_EVENTS` is required for kernel versions 3.0 and later. If you are using Symmetric MultiProcessing (SMP), enable the **Use local timer interrupts** option `CONFIG_LOCAL_TIMERS`. If you are running on Linux version 3.12 or later, the `CONFIG_LOCAL_TIMERS` option is not necessary.

CPU Power Management

Optionally enable the **CPU Frequency scaling** option `CONFIG_CPU_FREQ` to enable the CPU Freq **Timeline** view chart. `gator` requires kernel version 2.6.38 or greater to enable this chart.

Kernel hacking

If other trace configuration options are enabled, the **Trace process context switches and events** option `CONFIG_ENABLE_DEFAULT_TRACERS` might not be visible in `menuconfig` as an option. Enabling one of these other trace configurations, for example `CONFIG_GENERIC_TRACER`, `CONFIG_TRACING`, or `CONFIG_CONTEXT_SWITCH_TRACER`, is sufficient to enable tracing. Optionally enable the **Compile the kernel with debug info** option `CONFIG_DEBUG_INFO`. This option is only required for profiling the Linux kernel.



Caution

Kernel versions before 4.6, with `CONFIG_CPU_PM` enabled, produce invalid results. For example, counters not showing any data, large spikes, and non-sensible values for counters. This issue is due to the kernel PMU driver not saving state when the processor powers down, or not restoring state when it powers up. To avoid this issue, upgrade to the latest version of the kernel, or apply the patch found at <https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=da4e4f18afe0f3729d68f3785c5802f786d36e34>. This patch applies cleanly to version 4.4, and it might also be possible to back port it to other versions. If you apply the patch, you might also require the patch at <https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=cbcc72e037b8a3eb1fad3c1ae22021df21c97a51>.

4.2 Profile the Linux kernel

To ensure the statistics that the kernel generates align with source code in the Analysis Reports, include the kernel in the **Program Images** section in the **Capture and Analysis Options** dialog box. Before you can include the kernel in the **Program Images** section, you must build a version of `vmlinux` with kernel debug information enabled.

Procedure

1. Navigate to the kernel build directory.
2. Enter the following command to enable you to change `menuconfig` options:

```
make ARCH=arm CROSS_COMPILE=<cross_compiler_directory>/bin/arm-linux-gnueabihf-menuconfig
```

3. In the **Kernel Hacking** menu, select the **Compile the kernel with debug info** option. This option enables the `CONFIG_DEBUG_INFO` kernel option.
4. Enter the following command to build a new `vmlinux` image:

```
make -j5 ARCH=arm CROSS_COMPILE=<cross_compiler_directory>/bin/arm-linux-gnueabihf- uImage
```



Note

You can profile a driver by either statically linking it into the kernel image or by adding the module as an image in the **Capture and Analysis Options** dialog box.

5. Open the **Capture and Analysis Options** dialog box.
6. Click the **Add ELF image...** button in the **Program Images** section.
7. Navigate to your `vmlinux` file and select it.
8. Click **OK**.
9. Start a new capture session.

Related information

[Kernel configuration menu options](#) on page 21

4.3 Build gatord yourself

Build `gatord` yourself to apply patches for bug fixes or add support for new features.

About this task



It is not possible to build `gatord` on a Windows host.

Procedure

1. Either download the `gatord` source from the daemon directory in <https://github.com/ARM-software/gator>, or copy the source that is supplied in `<DS_install_directory>/sw/streamline/gator/daemon/`.
2. Follow the instructions in the `README.md` file in the `gator` directory.

Related information

[Set up your target](#) on page 9

4.4 Add support to gatord for a new CPU or perf PMU

There are two methods for adding support for a new CPU or perf PMU to `gatord`. One that requires rebuilding `gatord`, and one that does not.

- Perf support in Linux for the PMU is required because `gatord` uses perf to read the hardware counters.
- Check the perf PMUs supported by your kernel by running:



```
ls /sys/bus/event_source/devices/
```

If `ARMv7_Cortex_A<xx>`, `CCI_400`, or `cnn` are listed, then `A<xx>`, `CCI-400`, or `CNN` respectively are supported.

- Only XML changes are required, no code changes are necessary.
- Rebuilding `gatord` after the XML changes is recommended but not required because you can pass PMUs and events to `gatord` on the command line.

Make the following changes, then rebuild `gatord`:

1. Add a line to `<install_directory>/sw/streamline/gator/daemon/pmus.xml` describing the new PMU. For CPUs, the following information is required:
 - The CPU Implementer and Primary part number from the Main ID Register.

- The number of generic hardware counters that can be selected simultaneously.
 - Optionally, set the perf PMU name of the CPU to ensure correct operation in multi-PMU heterogeneous CPU implementations, such as big.LITTLE™ or DynamIQ™.
2. Create an events XML file, named `events-<xxx>.xml` in the `gator` source directory that defines the events that the new PMU generates. Exclude the XML header and `<events>` element. See the Cortex®-A15 events XML file, `<DS_install_directory>/sw/streamline/gator/daemon/events-Cortex-A15.xml` for an example.

Alternatively, to add support without having to rebuild `gator`, do the following:

1. Create an events XML file that defines the events that the new PMU generates. This file must include the XML header and `<events>` element, for example:

```
<?xml version="1.0" encoding="UTF-8"?>
<events>
  <counter_set name="ARMv7_Cortex_A9_cnt" count="6"/>
  <category name="Cortex-A9" counter_set="ARMv7_Cortex_A9_cnt" per_cpu="yes"
    supports_event_based_sampling="yes">
    <event counter="ARMv7_Cortex_A9_ccnt" event="0xff" title="Clock"
      name="Cycles" display="hertz" units="Hz" \
      average_selection="yes" average_cores="yes" description="The number of core
      clock cycles"/>
    <event event="0x00" title="Software" name="Increment"
      description="Incremented only on writes to the \
      Software Increment Register"/>
    <event event="0x01" title="Cache" name="Instruction refill"
      description="Instruction fetch that causes \
      a refill of at least the level of instruction or unified cache closest to
      the processor"/>
    <!-- ... -->
  </category>
</events>
```

2. Create an XML file that defines information about the new PMU. For the required format, see the `gator pmus.xml`. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<pmus>
  <pmu pmnc_name="ARMv7_Cortex_A9" cpuid="0x41c09" core_name="Cortex-A9"
    pmnc_counters="6"/>
</pmus>
```

3. Copy these files to the target and restart `gator` using the following flags:
 - `-E` to specify the location of the events XML file. This flag causes the events to be appended to the list of events that `gator` supports.
 - `-P` to specify the location of the PMU XML file. This option causes the new PMU to be added to the list of PMUs defined in `pmus.xml` that `gator` has built-in support for.

Related information

[Build gator yourself](#) on page 23

[gator command-line options](#) on page 12