Arm® Streamline

Version 7.7

Target Setup Guide for Linux

arm

101814_0707_00_en

Arm® Streamline

Target Setup Guide for Linux

Release Information

Document History

Issue	Date	Confidentiality	Change
0701-00	25 September 2019	Non-Confidential	New document for v7.1.
0701-01	30 October 2019	Non-Confidential	Updated document for v7.1.
0702-00	14 February 2020	Non-Confidential	New document for v7.2.
0703-00	29 May 2020	Non-Confidential	New document for v7.3.
0704-00	21 August 2020	Non-Confidential	New document for v7.4.
0705-00	04 December 2020	Non-Confidential	New document for v7.5.
0706-00	05 March 2021	Non-Confidential	New document for v7.6.
0707-00	14 May 2021	Non-Confidential	New document for v7.7.

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

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.

Web Address

developer.arm.com

Progressive terminology commitment

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

We believe that this document contains no offensive terms. If you find offensive terms in this document, please contact *terms@arm.com*.

Contents
Arm® Streamline Target Setup Guide for Linux

Appendix A Advanced target setup information

Preface

This preface introduces the *Arm® Streamline Target Setup Guide for Linux*.

It contains the following:

About this book

This document describes how to set up a Linux device for profiling with Arm® Streamline.

Using this book

This book is organized into the following chapters:

Chapter 1 Target Setup

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

Chapter 2 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.

Chapter 3 Troubleshooting Known Issues

Troubleshoot known Streamline issues.

Appendix A Advanced target setup information

This appendix provides extra configuration information beyond the standard setup.

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.

Typographic conventions

italic

Introduces special terminology, denotes cross-references, and citations.

bold

Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

`monospace`

Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

`monospace`

Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

`monospace italic`

Denotes arguments to monospace text where the argument is to be replaced by a specific value.

`monospace bold`

Denotes language keywords when used outside example code.

`<and>`

Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example:

```
MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>
```

SMALL CAPITALS

Used in body text for a few terms that have specific technical meanings, that are defined in the *Arm® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Feedback

Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

If you have comments on content then send an e-mail to *errata@arm.com*. Give:

- The title *Arm Streamline Target Setup Guide for Linux*.
- The number 101814_0707_00_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

Arm tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

Other information

- *Arm® Developer*.
- *Arm® Documentation*.
- *Technical Support*.
- *Arm® Glossary*.

Chapter 1
Target Setup

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

It contains the following sections:

1.1 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 or Mali™ GPU hardware performance counters for your scenario.

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.

1.2 Compiling your application

When building executables for profiling with Streamline, it is best practice to use these compiler options when using GCC or Clang.

`-g`
Turn on the debug symbols necessary for quality analysis reports.

`-fno-inline`
Disable inlining and substantially improve the call path quality.

`-fno-omit-frame-pointer`
Compile 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`
Keep the frame pointer in leaf functions.

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

Call stack unwinding

You must provide extra compiler flags for call stack unwinding to work.

For AArch64 applications, the flag `-fno-omit-frame-pointer` is required. `-mno-omit-leaf-frame-pointer` must also be set on GCC. `-mno-omit-leaf-frame-pointer` is not supported on Clang, therefore the caller for samples in leaf functions is missing from the stack trace.

For AArch32 applications, the flags `-fno-omit-frame-pointer`, `-marm`, and `-mapcs-frame` are required.

——— **Note** ———

Streamline does not support call stack unwinding for T32 (Thumb®) code. It also does not support call stack unwinding for code that Arm Compiler version 5 and earlier (`armcc`) generates.

———————————

1.3 Set up your target

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

Prerequisites

- *Download* and *install* Arm Development Studio.
- If you intend to use the Statistical Profiling Extension (SPE) in Streamline, there are some additional system requirements. Refer to *1.4 System requirements for SPE support* on page 1-14.

Procedure

1. Configure the Linux kernel.

   The kernel configuration must include the options that are described in *A.1 Kernel configuration menu options* on page Appx-A-31. 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 `gatord` on your target.

   Note

   Kernel space gator (`gator.ko`) is no longer supported and has been removed from distribution.

   Two pre-built `gatord` binaries are available:

   - For Armv7 targets, and Armv8 targets that support AArch32 execution state.
   - For Armv8 AArch64 targets.

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

   - `<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 `gatord` running on the target must be compatible with the version of Streamline you are using. Use the same version of `gatord` as the version of Streamline.

   For more information, see *1.5 gatord* on page 1-15.

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
   ```

Next Steps

- Capture a profile of your application as described in the *Arm Streamline User Guide*.
- To stop `gatord` at the end of a capture, press Ctrl+C.

1.4 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*.

1.5 gatord

To be able 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.

Note

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.

1.5.1 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
To enable this mode, specify an output directory with the `--output` flag. `gatord` writes the capture to a file in this directory then exits.

Options available to all modes	Description
`-h, --help`	Lists all the available `gatord` command-line options.
`-c, --config-xml <config_xml>`	Specify the path and filename of the configuration XML 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.
`-e, --events-xml <events_xml>`	Specify the path and filename of the events XML file to use. This file defines all the counters that Streamline collects during the capture session.
`-E, --append-events-xml <events_xml>`	Specify the path and filename of the events XML file to append.
`-P, --pmus-xml <pmu_xml>`	Specify the path and filename of the PMU XML file to append.
`-v, --version`	Print version information.
`-d, --debug`	Enable debug messages.

(continued)

Options available to all modes	Description
`-A`, `--app <cmd> <args...>`	Specify the command to execute when the capture starts. This argument must be the last argument that is passed to `gatord`. All subsequent arguments are passed to the launched application.
`-S`, `--system-wide <yes|no>`	Specify whether to capture the whole system. In daemon mode, `no` is only applicable when `--allow-command` is specified. In this mode, you must enter a command in the **Start** view. Defaults to `yes`, unless `--app`, `--pid`, or `--wait-process` is specified.
`-u, --call-stack-unwinding <yes|no>`	Enable or disable call stack unwinding. Defaults to `yes`.
`-r, --sample-rate <none|low|normal|high>`	Specify the sample rate for the capture. The frequencies for each sample rate are: • `high`=10kHz • `normal`=1kHz • `low`=100Hz • `none`=the lowest possible rate Defaults to `normal`.
`-t, --max-duration <s>`	Specify the maximum duration that the capture can run for in seconds. Defaults to `0`, which means unlimited.
`-f, --use-efficient-ftrace <yes|no>`	Enable efficient ftrace data collection mode. Defaults to `yes`.
`-w, --app-cwd <path>`	Specify the working directory for the application that `gatord` launches. Defaults to the current directory.
`-x, --stop-on-exit <yes|no>`	Stop the capture when the launched application exits. Defaults to `no`, unless `--app`, `--pid`, or `--wait-process` is specified.
`-Q, --wait-process <command>`	Wait for a process that matches the specified command to launch before starting the capture. Attach to the specified process and profile it.
`-Z`, `--mmap-pages <n>`	The maximum number of pages to map per mmaped perf buffer is equal to `<n+1>`. `n` must be a power of two.
`-O, --disable-cpu-onlining <yes|no>`	Disable turning CPUs temporarily online 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 `no`.

Options available to daemon mode only	Description
`-p, --port <port_number>|uds`	Set the port number that `gatord` uses to communicate with the host. The default is 8080. If you use the argument `uds`, the TCP socket is disabled and an abstract Unix domain socket is created. This socket is named `streamline-data`. Alternatively, you can connect to `localhost:<local_port>` in Streamline using: `adb forward tcp:<local_port> localabstract:streamline-data`
`-a, --allow-command`	Allows you to run a command on the target during profiling. The command is specified in the **Start** view. **Caution** If you use this option, an unauthenticated user could run arbitrary commands on the target using Streamline.

Options available to local capture mode only	Description
`-s`, `--session-xml <session_xml>`	Specify the session XML file that the configuration is taken from. Any additional arguments override values that are specified in this file.
`-o, --output <apc_dir>`	Specifies the path and filename of the output file (`.apc`) for a local capture.
`-i, --pid <pids...>`	A comma-separated list of process IDs to profile
`-C, --counters <counters>`	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: `--counters <device>_cnt<s>:<e>` Where: • `<device>` is the prefix that identifies the device type. • `<s>` is the slot number. • `<e>` is the event code.
`-X`, `--spe <id>[:events=<indexes>][:ops=<types>][:min_latency=<lat>]`	Enable the Statistical Profiling Extension (SPE). Where: • `<id>` is the name of the SPE properties that are specified in the events XML file or the PMU XML file. It uniquely identifies the available events and counters for the SPE hardware. • `<indexes>` is a comma-separated list of event indexes to filter the sampling by. A sample is only recorded if all events are present. • `<types>` is a comma-separated list of operation types to filter the sampling by. If a sample is any of the types in <types>, it is recorded. Valid types are `LD` for load, `ST` for store and `B` for branch. • `<lat>` 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).
`-F`, `--spe-sample-rate <n>`	Specify the SPE periodic sampling rate. The rate, <n>, 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.

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" class="absolute" 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
  ```

1.6 Capture data locally on a target

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

Start `gatord` specifying:

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

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

`gatord` 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

1.7 Support for Arm® CoreLink™ tools

Streamline supports the CoreLink™ Level 2 Cache Controller (L2C-310) and the CoreLink Cache Coherent Network (CCN).

Profile the CCN using perf. The perf driver for CCN is merged into Linux 3.17, but can be backported to previous versions. See *https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/diff/?id=a33b0daab73a0e08cc04459dd44b0121a8e8f81b* and later bug fixes.

1.8 Enable ttrace annotations

Streamline supports ttrace annotations on targets that are running Tizen version 2.4 or later.

Procedure

1. Select **Window** > **Preferences** > **Streamline** > **External Tools**.
2. In the **ADB Path** field, specify the path to SDB.
3. In the **Start** view, select a target Tizen device with `gatord` running on it.
4. In the **Counter Configuration** dialog box, select the ttrace events to capture from the **Ttrace** section.

Streamline displays the ttrace tags as annotations in the **Timeline** view and the **Log** view.

1.9 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, `gatord` must first see which events are available in the running application. Streamline can then capture a profile as normal.

Prerequisites

- Build your application against Arm NN version 20.08 or later. See *https://github.com/ARM-software/armnn* for more information.
- When creating the runtime in your application code, you must set `options.m_ProfilingOptions.m_EnableProfiling = true` for the `options` passed to `IRuntime::Create`.
- To enable timeline profiling, also set `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 `gatord` in daemon mode as described in *1.3 Set up your target* on page 1-12.

   Note

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

2. Run your application.
3. Configure counters and *capture a profile of the application* as described in the *Arm Streamline User Guide*.

   Note

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

Chapter 2
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.

It contains the following sections:

2.1 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 `gatord`.

2.2 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 `gatord`.

2.3 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*.

Chapter 3
Troubleshooting Known Issues

Troubleshoot known Streamline issues.

It contains the following sections:

3.1 Troubleshooting target connection issues

Use these solutions for common target connection issues.

Problem	Solution

Error message generated:
`Unable to connect to the gator daemon at <target_address>.`
`Please verify that the target is reachable and that you are running gator daemon v17 or later. Installation instructions can be found in: streamline/gator/README.md.`
`If connecting over WiFi, please try again or use a wired connection.`
	Make sure `gatord` is running on your target. Enter the following command in the shell of your target:
	`ps ax | grep gatord`
	If this command returns no results, `gatord` is not active. Start it by navigating to the directory that contains `gatord` and entering the following command:
	`sudo ./gatord &`
	Try connecting to the target again.
	If `gatord` is active and you still receive this error message, disable any firewalls on your host machine that might interfere with communication between host and target.

Error message generated: `Unknown host`	Make sure that you have correctly entered the name or IP address of the target in the **Address** field. If you have entered a name, try entering an IP address instead.
When using event-based sampling, Streamline fails to find the PMU.	The PMU on your hardware might not be correctly configured to allow the processor interrupts necessary for Streamline to use event-based sampling. Test on alternate hardware or disable event-based sampling in the **Counter Configuration** dialog box.

The target is running a firewall, which prevents Streamline from connecting to `gatord`.	There are several possible ways to resolve this issue:
	• Update the firewall to allow connections to `gatord`, which defaults to using port 8080.
	• Use local captures.
	• If the target accepts SSH connections, you can establish an SSH tunnel by using the `ssh` command on the host. For example:
	`ssh <user>@<target> -L 8080:localhost:8080 -N`
	In this example, replace `<user>` with the username to log in as and `<target>` with the hostname of the target. On the target, use `localhost` as the hostname.
	Note
	An SSH tunnel requires extra processing on the target.
	• Reverse SSH tunnels are also possible by running `ssh` from the target to the host. For example:
	`ssh <user>@<host> -R 8080:localhost:8080 -N`

3.2 Troubleshooting gatord issues

Use these solutions for issues related to `gatord`.

Problem	Solution
Kernel version before 4.6 with `CONFIG_CPU_PM` enabled produces 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.	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*.
An Oops occurs when a processor is offlined.	The fix was merged into mainline in 3.14-rc5, see *https://git.kernel.org/pub/scm/linux/kernel/git/tip/tip.git/commit/?id=e3703f8cdfcf39c25c4338c3ad8e68891cca3731*, and has been backported to older kernels (3.4.83, 3.10.33, 3.12.14, and 3.13.6).
`dmesg` output: `CPU PMU: CPUx reading wrong counter -1`	Update to the latest Linux kernel.
Scheduler switch resolutions are on exact millisecond boundaries.	Update to the latest Linux kernel.

Appendix A
Advanced target setup information

This appendix provides extra configuration information beyond the standard setup.

It contains the following sections:

A.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 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. `gatord` 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*.

A.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 references

A.1 Kernel configuration menu options on page Appx-A-31

A.3 Build gatord yourself

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

Note

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 tasks

1.3 Set up your target on page 1-12

A.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.

Note

- 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 heterogenous CPU implementations, such as big.LITTLE™ or DynamIQ™.
2. Create an events XML file, named `events-<xxx>.xml` in the `gatord` 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 `gatord`, 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 `gatord` 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 `gatord` 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 `gatord` has built-in support for.

Related tasks

A.3 Build gatord yourself on page Appx-A-33

Related references

1.5.1 gatord command-line options on page 1-15