Intel® SoC Watch 2.3.1 for Linux* OS and Google Android* OS

User's Guide

Copyright © 2017 Intel Corporation
All Rights Reserved
Revision: 2.3.1

Legal Information

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel, the Intel logo, Intel Atom, Intel Core, Intel Inside, Intel Xeon Phi, and VTune are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others

© Intel Corporation

Revision History

Revision Number	Description	Revision Date
2.0.0	Initial release for 2.0 product	July 2015
2.1.0	Update for 2.1 product release	October 2015
2.1.1	Update for 2.1.1 product release	December 2015
2.2	Update for 2.2 product release and to combine Android* and Linux* User's Guides	June 2016
2.2.1	Update for 2.2.1 product release. Update from `socperf1_2.ko` to `socperf2_0.ko`. Bug fixes.	June 2016
2.3.0	Update for 2.3.0 product release.	October 2016
2.3.1	Update for 2.3.1 product release.	May 2017

Contents

1 About this Document ... 5
1.1 Intended Audience ... 5
1.2 Conventions and Symbols ... 5
2 Overview ... 6
3 Installation ... 7
3.1 Requirements ... 7
3.2 Extracting the Intel SoC Watch package ... 7
3.3 Build the Kernel Modules ... 8
3.3.1 Building Linux* Kernel Modules ... 8
3.3.2 Building Android* Kernel Modules ... 8
3.4 Install Intel SoC Watch ... 9
3.4.1 Intel SoC Watch for Linux* Installation ... 9
3.4.2 Intel SoC Watch for Android* Installation ... 10
3.5 Key Files ... 10
3.6 Remove the Intel SoC Watch Drivers ... 11
4 Quick Start ... 12
4.1 Collect On a Linux* based system ... 12
4.2 Collect on an Android* based system ... 12
4.3 View the Results in Intel VTune Amplifier for Systems ... 13
5 Using Intel SoC Watch ... 15
5.1 General Options ... 15
5.2 Collection Options ... 15
5.2.1 Available Features ... 18
5.2.2 Combination Features ... 21
5.3 Post-Processing Options ... 22
5.4 Examples ... 22
6 Intel SoC Watch Output Files ... 23
6.1 Summary CSV File ... 23
6.2 Summary HTML File ... 23
6.3 Raw Data SW2 File ... 23
6.4 Timed Trace CSV File ... 23
6.5 Intel VTune™ Amplifier for Systems PWR Import File ... 23
7 Feature Descriptions ... 24
8 Using Intel® VTune™ Amplifier for Systems to Analyze Intel SoC Watch Data ... 28
8.1 Importing Data ... 28
8.2 Using the Intel VTune Amplifier for Systems GUI ... 29
8.2.1 Filtering ... 29
8.2.2 Expanding Grid Columns ... 31
8.2.3 Changing Grid Groupings ... 32
8.2.4 Timeline Zooming ... 32

9 Intel SoC Watch Tips 33
9.1 IPI Wakeups 33
9.2 Submitting an issue/Getting Help 34
Appendix A: Acronyms and Definitions 35

§

1 About this Document

1.1 Intended Audience

You should read this document if you are an Intel SoC Watch user.

1.2 Conventions and Symbols

The following conventions are used in this document.

Table 1 Conventions and Symbols used in this Document

`This type style`	Indicates an element of syntax, reserved word, keyword, filename, computer output, or part of a program example. The text appears in lowercase unless uppercase is significant.
This type style	Indicates the exact characters you type as input. Also used to highlight the elements of a graphical user interface such as buttons and menu names.
`This type style`	Indicates a placeholder for an identifier, an expression, a string, a symbol, or a value. Substitute one of these items for the placeholder.
`[ items ]`	Indicates that the items enclosed in brackets are optional.
`{ item | item }`	Indicates to select only one of the items listed between braces. A vertical bar (|) separates the items.
... (ellipses)	Indicates that you can repeat the preceding item.

2 Overview

Intel SoC Watch is a command line tool for monitoring system behaviors related to power consumption on Intel® architecture-based platforms. It monitors power states, frequencies, bus activity, wakeups, and various other metrics that provide insight into the system's energy efficiency.

To collect and analyze power data:

1. Collect data. A summary (CSV) file and raw data (SW2) file are produced by default on the target system (which is the system being analyzed).
2. Optionally, generate a PWR file and import it into the Intel® Energy Profiler (via Intel® VTune™ Amplifier 2016 for Systems or later) tool on the host system (which is the system where Intel VTune Amplifier for Systems is installed) to correlate and visualize the system behavior over time. Utilize powerful filtering in Intel VTune Amplifier for Systems to analyze the results.
3. Optionally, collect a timed trace (CSV) file that can be opened by Microsoft* Excel* and graph the individual metrics overtime.
4. Optionally, collect an HTML summary result file that can be opened with any recent browser on the target or host system.

Figure 1: Intel SoC Watch inputs and ouputs

3 Installation

3.1 Requirements

Intel SoC Watch works on systems based on kernel 2.6.32 or later that is built with the following kernel configuration options enabled:

- CONFIG_TRACEPOINTS
- CONFIG_MODULES
- CONFIG_TIMER_STATS

Intel SoC Watch requires GLIBC_2.17 or later to be installed on Linux target systems.

To measure processor P-States:

- Enable the CONFIG_X86_SFI_CPUFREQ or CONFIG_X86_ACPI_CPUFREQ kernel configuration options by setting the value to 'y' or 'm'.
- Verify that one of the following P-State drivers is loaded: `sfi-cpufreq`, `acpi-cpufreq`, or `intel_pstate`. To determine which driver is loaded, check the sysfs `/sys/devices/system/cpu/cpu0/cpufreq/scaling_driver` file. If one of these P-State drivers is not loaded, the kernel needs to be reconfigured and recompiled.

To measure processor C-States:

- Enable the CONFIG_TIMER_STATS and CONFIG_INTEL_IDLE kernel configuration options by setting the value to 'y' or 'm'.
- The `intel_idle` kernel module has to support the target platform's processors. To determine if the `intel_idle` kernel module is loaded, check the sysfs `/sys/devices/system/cpu/cpuidle/current_driver` file. It must equal `intel_idle`. If it equals `acpi_idle`, only C0 and C1 will be used by the core.

3.2 Extracting the Intel SoC Watch package

Users will need to extract the `system_studio_target.tgz` package to the system containing the target device's kernel (this may be the host system if the target device is running an Android kernel, or the target system if the device is running a Linux kernel). By default the package can be found in the following locations:

- On Linux systems:
  `/opt/intel/system_studio_<version>/targets/system_studio_target.tgz`

- On Windows* systems: `C:\Program Files (x86)\IntelSWTools\system_studio_<version>\Targets\system_studio_target.tgz`

Use the `find . -name Module.symvers` command on the device containing the target system's kernel to determine the kernel build directory.

3.3 Build the Kernel Modules

If the Intel SoC Watch kernel modules (i.e. device drivers) are not present in the target system's OS image, you will need to build and possibly sign them. Building and signing device drivers requires access to the kernel build directory for the OS image running on your target device. A kernel build directory is generated while building the target system's OS image.

When building the kernel modules, the Intel SoC Watch package (i.e. ZIP file) should not be opened (i.e. unzipped) on a Windows* based system and then copied to a Linux system. The package should be unzipped on the Linux build system using the `unzip` command to make sure the build scripts and make files are unmodified.

If a kernel is built with the CONFIG_MODULE_SIG kernel config enabled, any device driver loaded into that kernel must be signed with the same keys used to build the kernel. In general, drivers built for Linux targets do not need to be signed and the following description assumes the drivers do not need to be signed. But, if an end user tries to load an unsigned driver into a kernel that requires signed drivers, the `insmod` command will fail with the error *Required key not available*. If a signed driver is loaded into a kernel that does not require signed drivers, the load will succeed.

The build_drivers script is provided to simplify building all of the drivers. The script supports multiple switches including;

-n // do not build the socperf driver; used for Intel® Core™ based systems

-l // build the kernel for a Linux target

-s <full path to sign-file> // signs the drivers; the path is normally .../kernel/*/scripts/sign-file

3.3.1 Building Linux* Kernel Modules

Linux kernel modules may only be built after the Intel SoC Watch package and kernel headers are copied to and installed on the target. See the section Intel SoC Watch Linux Installation below for instructions on how to build the kernel modules for a target device running Linux.

3.3.2 Building Android* Kernel Modules

1. Extract the Intel SoC Watch package to the host system used to build the target system's Android kernel. For more information, see Extracting the Intel SoC Watch package. The `<extract_dir>/system_studio_target/socwatch_android_v<version>` directory will be created.
2. Use one of the following commands:

   If the target system is based on a SoFIA SoC, use the following command to build both the `socwatch2_3.ko` and `socwatchmv1_2.ko` files.

   ```
   > sh ./build_drivers.sh -k <kernel-build-dir> --sofia -s <full path to sign-file>
   ```

   If the target system is based on an Intel Core processor, use the following command to build the `socwatch2_3.ko` file.

   ```
   > sh ./build_drivers.sh -k <kernel-build-dir> -n -s <full path to sign-file>
   ```

   If the target system is based on an Intel® Atom™ processor, use the following command to build the `socwatch2_3.ko` and `socperf2_0.ko` files.

   ```
   > sh ./build_drivers.sh -k <kernel-build-dir> -s <full path to sign-file>
   ```

NOTE: If the build fails because the `asm/intel_mid_pcihelpers.h` file is missing, remove the following line from the `soc_perf_driver/src/Makefile`

```
EXTRA_CFLAGS += -DCPI_HELPERS_API
```

3.4 Install Intel SoC Watch

Host: laptop, desktop, or server used to communicate with target device.

Target: device to be analyzed with Intel SoC Watch.

3.4.1 Intel SoC Watch for Linux* Installation

If Intel SoC Watch was previously installed on the target, delete the `socwatch _linux_*` directory before installing a new version. Then, perform the following steps on the target device.

1. Login to the target as root:

   ```
   > ssh root@<your_target_IP>
   ```

2. Extract the Intel SoC Watch package to the target Linux system. For more information, see Extracting the Intel SoC Watch package. The `<extract_dir>/system_studio_target/socwatch_linux_v<version>_x<architecture>` directory will be created.
3. Copy the Intel SoC Watch package to a working directory:

   ```
   > mkdir -p /home/socwatch
   > cp <extract_dir>/system_studio_target/socwatch_linux_v<version>_x<architecture> /home/socwatch/.
   ```

4. Navigate to the Intel SoC Watch directory:

```
> cd /home/socwatch/socwatch_linux_v<version>_x<architecture>
```

5. Use one of the following commands:
   a. If the target system has an Intel® Atom™ processor, use the following command to build the `socwatch2_3.ko` and `socperf2_0.ko` files.

      ```
      > sh ./buid_drivers.sh –l -k <kernel-build-dir> -s <full-path-to-sign-file>
      ```

   b. If the target system has an Intel® Core™ processor, use the following command to build the `socwatch2_3.ko` file.

      ```
      > sh ./build_drivers.sh –l –k <kernel-build-dir> –n –s <full-path-to-sign-file>
      ```

3.4.2 Intel SoC Watch for Android* Installation

Make sure the host is connected to the target via adb before running the install script.

1. After extracting the Intel SoC Watch package on your host system, run the `socwatch_android_install.sh` script on a Linux host or from a Cygwin window on a Windows host. Run `socwatch_android_install.bat` from a Windows host.

   ```
   > socwatch_android_install.sh or socwatch_android_install.bat
   ```

   The script installs the Intel SoC Watch executables to the /data/socwatch directory on the target by default. Use the –d option to select a different install directory and the –s option to define a specific Android* device if multiple devices are connected to the host via adb.

2. Using the adb command, start a shell with root privileges.

   ```
   > adb root
   > adb shell
   ```

3. Navigate to the directory containing the drivers on the target system. If the drivers are preinstalled, they are located in the `/lib/modules` or `/system/lib/modules` directory. If the drivers are built on the host system, they should be copiedto a directory on the target system.
4. Load the drivers. Note that the `socperf` driver must be loaded before the `socwatch` driver on a system with an Intel® Atom processor.

   ```
   > insmod socperf2_0.ko
   > insmod socwatch2_3.ko
   ```

NOTE: Make sure to use the latest version of the `socperf` driver.

NOTE: Previous versions of the `socwatch2_x.ko` driver (e.g. `socwatch2_0.ko`) will work, but new collector support and/or bug fixes may be missing in the older drivers. If an older `socwatch2_x.ko` driver is used, some metrics may not be collected.

5. Confirm the drivers are loaded.

   ```
   > lsmod
   ```

3.5 Key Files

The following table describes the key files.

File	Description
`build_drivers.sh`	The build script used to build all of the device drivers utilized by Intel SoC Watch.
`socperf2_0.ko`	The socperf kernel module used to measure bandwidth and DRAM self refresh on systems with an Intel Atom processor.
`socwatch2_x.ko`	The Intel SoC Watch kernel module used to collect both hardware and kernel data at runtime.
`setup_socwatch_env.sh`	The script used to setup the Intel SoC Watch runtime environment.
`socwatch`	The Intel SoC Watch executable built as a native application. Use this file to collect data and generate additional results from a raw SW2 file.
`SOCWatchConfig.txt`	The Intel SoC Watch configuration file. The configuration file is read by Intel SoC Watch immediately before each collection. It contains hardware addresses utilized by the device driver during the collection.
`EULA.txt`	End User License Agreement file.
`third-party-programs.txt`	List of third party programs included in the package.
`plugins/libSWCore.so`	A library providing Intel SoC Watch functionality.

3.6 Remove the Intel SoC Watch Drivers

After using Intel SoC Watch, remove the drivers using the `rmmod` command (e.g. `rmmod socwatch2_3`). The `socwatch` driver must be unloaded before the `socperf` driver is unloaded.

4 Quick Start

The following steps assume the Intel SoC Watch drivers and executables are installed. See the Installation section above for instructions on how to install Intel SoC Watch.

Use the following steps to quickly collect processor C-state and P-state data for 60 seconds and import it into Intel VTune™ Amplifier 2016 for Systems or later for analysis. To graph the metrics in Microsoft* Excel* see the Graphing Timed Trace Data with Microsoft* Excel* section below.

4.1 Collect On a Linux* based system

To collect directly on a Linux target, login as 'root'. To collect remotely on a target device running Linux, use ssh to login to your target device as root.

1. Load the device drivers. The socperf2_0.ko driver must be loaded before the Intel SoC Watch driver. The socperf2_0.ko driver should only be loaded on systems with an Intel Atom processor.
```
> insmod /home/socwatch/socwatch
_linux_v2.3/soc_perf_driver/src/socperf2_0.ko
> insmod /home/socwatch/socwatch
_linux_v2.3/socwatch_driver/socwatch2_3.ko
```
2. Navigate to the socwatch directory: `> cd /home/socwatch/socwatch_linux_v2.3`
3. Setup the collection environment:
```
> source ./setup_socwatch_env.sh
```
4. Create a results directory:
```
> mkdir results
```
5. Collect data. The following command generates the test.csv, test.sw2 and test.pwr files in the results directory.
```
>./socwatch -r vtune -f cpu-cstate -f cpu-pstate -t 60 -o ./results/test
```
6. View the summary results:
```
> cat ./results/test.csv
```
7. To view the results in Intel VTune Amplifier for Systems on your host system, copy the test.pwr file from the target to the host using scp.The following steps assume a Windows host.
```
> scp root@<your_target_IP>:/home/socwatch/<path>/results/test.pwr
c:\results\.
```

4.2 Collect on an Android* based system

1. On the host system, establish a root adb shell on the target:
```
> adb root
> adb shell
```
2. Load the device drivers. Note that the socperf driver must be loaded before the socwatch driver on systems with an Intel Atom processor.

```
> insmod <path_to_drivers>/socperf2_0.ko
```

NOTE: Note: Make sure to use the latest version of the socperf driver

```
> insmod <path_to_drivers>/socwatch2_3.ko
```

3. Confirm the drivers are loaded:

   ```
   > lsmod
   ```

   Confirm the socperf and socwatch drivers are included in the list of installed modules.

4. Setup the collection environment. This step assumes the default install directory was used.

   ```
   > cd /data/socwatch
   > source ./setup_socwatch_env.sh
   ```

5. Collect data and generate the test.csv, test.sw2, and test.pwr files in the results directory. This step assumes the /data/socwatch/results directory exists.

   ```
   >./socwatch -r vtune -f cpu-cstate -f cpu-pstate -t 60 -o ./results/test
   ```

6. Exit the adb shell:

   ```
   > exit
   ```

7. Use adb to pull the result files to the host:

   ```
   > adb pull /data/socwatch/results/test.csv c:\results
   > adb pull /data/socwatch/results/test.sw2 c:\results
   > adb pull /data/socwatch/results/test.pwr c:\results
   ```

4.3 View the Results in Intel VTune Amplifier for Systems

If you are going to analyze results using Intel VTune Amplifier for Systems, make sure that you have Intel VTune Amplifier 2016 for Systems or later.

1. Navigate to the Intel VTune Amplifier for Systems bin32 or bin64 directory and start the Intel VTune Amplifier for Systems GUI.

```
amplxe-gui.exe
```

2. Open the result by selecting: Import Result.

3. Navigate to the PWR file. Click **Import**.

4. To collect additional data, see the Using Intel SoC Watchsection below.
5. To graph the data collected in Microsoft* Excel*, see the Graphing Timed Trace Data with Microsoft* Excel* section below.

5 Using Intel SoC Watch

NOTE: Intel SoC Watch does not collect any data by default. You must specify what data you wish to collect with the –f switch.

Before starting a collection, source the `setup_socwatch_env.sh` script to set the necessary environment variables.

```
source ./setup_socwatch_env.sh
```

Use the following basic command to launch a collection:

```
./socwatch <general options> <collection options> <post-processing options>
```

5.1 General Options

```
-c, --config-file <config_file_name>
```

Specify the configuration file. The `<config_file_name>` argument can optionally include an absolute or relative path. Intel SoC Watch loads architecture information from a configuration file, for example, MSR address information. If this switch is not specified, Intel SoC Watch loads the default configuration file `SoCWatchConfig.txt`.

```
-d [level], --debug [level]
```

Generate debug output from Intel SoC Watch. Used to debug Intel SoC Watch collection and processing issues. The default level is set to 1 if the level parameter is not specified.

Level	Description
0	No error messages
1	Only error messages
2	Only error and warning messages
3	Error, warning and debug messages
4	Error, warning, debug and informational messages

```
-h, --help
```

Display usage information and exit.

```
-v, --version
```

Print Intel SoC Watch and device driver version information and exit.

5.2 Collection Options

```
-f, --feature <data_to_collect>
```

Specify the data to be collected. Each `data_to_collect` needs to be prefaced with a `-f` or `--feature` switch. For example, `-f cpu-cstate` will cause Intel SoC Watch to collect processor C-state data and `–f cpu-cstate –f cpu-pstate` will cause Intel SoC Watch to collect both processor C-

state and processor P-state data. The –m, `--max-detail` switch determines what collection mechanism is used. See the *Available Features* section below.

```
--low-overhead
```

Utilize OS context switch points to collect hardware data that is polled/sampled to minimize perturbation of residency and wakeup data. Use this switch to minimize the overhead of polled collections (see the "`-m, --max-detail`" section below for more information on the different collection methodologies).

```
-m, --max-detail
```

Specify the level of detail to collect. This switch applies to all data specified to be collected by the –f switch. Internally, this switch setting guides the tool's choice of a collection mechanism and also controls overhead. In general, if `-m` is specified, the tool collects the maximum detail possible and may also cause higher overhead. In general, if `-m` is not specified, the tool collects a minimum amount of detail and may also cause the minimum overhead.

Intel SoC Watch uses different methods to collect data to provide the maximum detail or minimum overhead. Data may be traced, polled, or snapshot to give you the most accurate result or an acceptable result with minimum system perturbation. Traced data is read when the metric itself changes providing the most accurate result. Polled data is read at regular intervals, unless the `--low-overhead` switch is specified, in which case the data is read only during OS context switch points. Note that polled data may not be 100% accurate as metric changes that take place between polling intervals may not be measured. Snapshot data is read at the beginning and end of the collection and the difference is provided in the result. Snapshot data provides the average result, minimum overhead, and the least amount of detail. The following algorithm used to determine the collection method for each data type is as follows:

If –m is specified;
  if the data can be traced, trace it;
  else if the data can be polled, poll it;
  else snapshot it.
If –m is **not** specified;
  if the data can be snapshot, snapshot it;
  else if the data can be traced, trace it;
  else poll it.

```
-n, --interval <milliseconds>
```

Specify the polling interval used when data is polled. The default polling interval is 100ms. The minimum polling interval is 1ms.

```
-o, --output <file_name>
```

Specify an output file name (optionally including path). Collected data (if any) is written to the file `file_name.csv` and `file_name.sw2`. If the `-o` switch is not specified, data is saved to the `SoCWatchOutput.csv` and `SoCWatchOutput.sw2` files in the current directory. To create additional files, use the `-r` switch described below.

```
-p, --program <application><parameters>
```

Specify the application (including path to the executable) and its parameters to be launched by Intel SoC Watch at the same time a collection is started.

NOTE: The `-p` switch MUST be the last Intel SoC Watch switch used on the command line.

```
-r, --result <result_type>
```

Specify the type of results to be generated. Three types are possible: `html`, `int`, and `vtune`. The `html` option generates a single summary HTML file that can be viewed with any recent browser. The `int` option generates a single timed trace CSV file that includes data for each metric collected (specified by the `-f` switch). The `vtune` option generates a single PWR file that includes data for each metric collected (specified by the `-f` switch) which can be imported into Intel VTune™ Amplifier for Systems for visualization. Each result file to be generated must be prefaced with the `-r` switch

```
-s, --startdelay <seconds>
```

Specify the collection delay in seconds. Data capture starts `seconds` after the Intel SoC Watch command is executed.

```
-t, --time <seconds>
```

Specify the duration of the collection, in seconds. Intel SoC Watch stops a collection when any of the following events occur:

- the duration specified by the `-t` switch has elapsed
- Intel SoC Watch receives a `SIGINT` (CTRL-C)
- the application launched by Intel SoC Watch with the `-p` switch exits

```
--start
--start-paused
--stop
--terminate
```

The --start, --start-paused, --stop, and --terminate switches allow Intel SoC Watch to be used in a non-blocking mode.

- When the `--start` switch is used, Intel SoC Watch checks if it is currently running in non-blocking mode. If so, it forwards the collection start command to the Intel SoC Watch daemon which executes the command. If Intel SoC Watch is not running in non-blocking mode, it forks a daemon task and forwards the command to the daemon to execute.
- When the `--start-paused` switch is used, Intel SoC Watch does the same check as the `--start` command except instead of actually starting a collection, it prepares Intel SoC Watch to

collect data and then waits for a `--start` command. Users should use the `--start-paused` command to eliminate any collection start up time.

- When the `--stop` switch is used, Intel SoC Watch ends a non-blocking mode collection. The Intel SoC Watch daemon continues to run, waiting for the next command.
- When the `--terminate` switch is used, Intel SoC Watch ends a non-blocking mode collection and kills the Intel SoC Watch daemon. The next Intel Soc Watch collection will run in a traditional blocking mode – i.e. the collection will not run as a background task.

5.2.1 Available Features

The available features that Intel SoC Watch can measure and their possible collection methods are listed below. Detailed descriptions of the different kinds of data are provided in the Feature Descriptions section below. Note that different platforms support different features – i.e. not all of the features listed below are available on every platform supported by Intel SoC Watch. Use the –h switch to determine which features are supported by a platform.

all-approx-bw

Collection Method(s): poll

Estimate multiple SoC bandwidths concurrently. SoC transactions are counted and multiplied by 64 to estimate multiple SoC bandwidths concurrently. The estimates are equal to or higher than the actual SoC bandwidths. If the transactions executed during a collection include partial or 32 byte transfers, the estimate is high.

- **Module 0** and **Module 1**: the bandwidth from CPU Module 0 and Module 1 to the DDR
- **GFX**: the bandwidth from the graphics component to the DDR
- **Display**: the bandwidth from the display controller to the DDR
- **ISP**: the bandwidth from the camera image processor to the DDR
- **VED**: the bandwidth from the video encode and decode components to the DDR
- **IO**: the bandwidth between the north and south clusters

core-temp

Collection Method(s): poll

Collect Intel architecture core temperature data via MSR reads.

cpu-cstate

Collection Method(s): snapshot, trace

Collect Intel architecture core, module (when applicable), and package C-state data using hardware C-state residency MSR data and the wakeups that cause Intel architecture cores to exit a C-state. Wakeups are only collected if the –m or --max-detail switch is specified. The -m or --max-detail switch is required to ensure that cpu-cstate collections generate correct results on any Intel Core based systems.

cpu-ddr-mod0-bw, cpu-ddr-mod1-bw

Collection Method(s): poll

Measure the Intel architecture core to DDR bandwidth in MB/sec. The cpu-ddr-mod0-bw or cpu-ddr-mod1-bw features measure module 0 or module 1 Intel architecture core to DDR bandwidth respectively. Only one bandwidth or DRAM Self Refresh data type can be specified at a time.

cpu-pstate

Collection Method(s): trace

Collect Intel architecture core P-state (frequency) data using hardware MSR data.

ddr-bw

Collection Method(s): poll

Measure total DDR bandwidth in MB/sec using the system's VISA/SoCHaP functionality. Only one bandwidth or DRAM Self Refresh data type can be specified at a time.

disp-ddr-bw

Collection Method(s): poll

Measure the display controller to DDR bandwidth in MB/sec using the system's VISA/SoCHaP functionality. Only one bandwidth or DRAM Self Refresh data type can be specified at a time.

dram-bw

Collection Method(s): poll

Collect total DDR bandwidth with memory slice breakdown via MMIO reads.

dram-srr

Collection Method(s): poll

Measure the residency of DRAM while it is in self refresh mode using the system's VISA/SoCHaP functionality. Only one bandwidth or DRAM Self Refresh data type can be specified at a time. Do not specify a polling interval greater than 14 seconds when measuring DRAM Self Refresh on a SoC with an Intel® Atom™ processor. The polling interval is specified with the –n, --interval switches.

gfx-cstate

Collection Method(s): poll

Collect GPU C-state data via PCI reads. This data type can only be collected on 22nm Intel® Atom™ Processor-based SoC for Tablets and 2-in-1s (formerly code named Bay Trail).

NOTE: Some bandwidth measurements include ReadPartial or WritePartial traffic. The payload (how many bytes were transferred) of a partial transaction can range from 1 to 64 bytes. Therefore, the exact bandwidth cannot be accurately measured. In these cases, the Intel SoC Watch bandwidth results provide a Total Bandwidth Range. The Total Bandwidth Range describes the minimum and maximum bandwidth measured. The actual bandwidth falls within the Total Bandwidth Range.

gfx-ddr-bw

Collection Method(s): poll

Measure the graphics component to DDR bandwidth in MB/sec using the system's VISA/SoCHaP functionality. Only one bandwidth or DRAM Self Refresh data type can be specified at a time.

gfx-pstate

Collection Method(s): poll

Collect GPU P-state (frequency) data via PCI reads.

io-bw

Collection Method(s): poll

Measure the bandwidth between the North Cluster (NC) and South Cluster (SC) in MB/sec using the system's VISA/SoCHaP functionality. Only one bandwidth or DRAM Self Refresh data type can be specified at a time.

isp-ddr-bw

Collection Method(s): poll

Measure the ISP (camera image processor) to DDR bandwidth in MB/sec using the system's VISA/SoCHaP functionality. Only one bandwidth or DRAM Self Refresh data type can be specified at a time.

nc-dstate

Collection Method(s): poll

Collect North Complex component D0ix state data via PCI reads. The nc-dstate data represents the *state* of each device when the data was collected. It does not represent the residency of the device.

pkg-temp

Collection Method(s): poll

Collect temperature data from various thermal domains on the package via MMIO reads.

pmic-temp

Collection Method(s): poll

Collect MSIC/PMIC temperature data via sysfs reads.

sc-dstate

Collection Method(s): poll

Collect South Complex component D0ix state data via PCI reads. The sc-dstate data represents the *state* of each device when the data was collected. It does not represent the residency of the device.

sstate

Collection Method(s): snapshot (for S0ix) and trace (for ACPI S3)

If the sstate feature is specified, both S0ix and ACPI Suspend-To-RAM (S3) data are collected concurrently. S3 data is traced using kernel hooks to trace the system's S3 behavior. S0ix data is

snapshot at the start of the collection, at the end of the collection, and during S0->S3 and S3->S0 transitions.

skin-temp

Collection Method(s): poll

Collect device skin temperature data via sysfs reads.

soc-temp

Collection Method(s): poll

Collect SoC temperature data via PCI reads.

wakelock (Android only)

Collection Method(s): trace

Collect user and kernel wakelock data using kernel tracepoints and data gathered from the aplog.

5.2.2 Combination Features

The following features are combinations of the previously described features. These features can be used to simplify command lines used to collect multiple data types concurrently. For example, the `-f cpu` switch can replace the `-f cpu-cstate -f cpu-pstate` switches in a command line.

If a combination feature includes a feature not supported on the target platform, that feature will be ignored. For example, if the `device` switch is utilized on a platform with a 22nm Intel® Atom™ Processor-based SoC for Tablets and 2-in-1s (formerly code named Bay Trail), the `sc-dstate` feature, normally included in the `device` feature, is ignored.

cpu

A combination of the `cpu-cstate` and `cpu-pstate` features.

device

A combination of the `nc-dstate and sc-dstate` features.

gfx

A combination of the `gfx-cstate` and `gfx-pstate` features.

sys

A combination of the `cpu-cstate`, `cpu-pstate`, `sstate`, `nc-dstate`, `sc-dstate`, `core-temp`, `soc-temp`, `skin-temp`, `pmic-temp`, `gfx-cstate`, `gfx-pstate`, and `ddr-bw` features.

temp

A combination of the `core-temp`, `soc-temp`, `skin-temp`, and `pmic-temp` features.

5.3 Post-Processing Options

```
-i, --input <file_name>
```

Specify an input SW2 file to post-process. Do not include the file extension, SW2, when specifying the filename. Use this switch in combination with the `-r` switches (`sum, html, int, or vtune`) to generate summary (CSV), summary (HTML), timed trace (CSV), or Intel VTune™ Amplifier for Systems import (PWR) file resepctively.

```
-r, --result <result_type>
```

Specify the type of results to be generated. Three types are possible: `html`, `int` and `vtune`. The `html` option generates a single summary HTML file that can be viewed with any recent browser. The `int` option generates a timed trace CSV file. The `vtune` option generates a PWR file that can be loaded into Intel VTune Amplifier 2016 for Systems. The `-o` switch can be used to define the output name of the file generated with the `-i -r html, int` or `vtune` switch combination. If the `-o` switch is not specified, the file will be named `SoCWatchOutput.html`, `SoCWatchOutput_trace.csv`, or `SoCWatchOutput.pwr` respectively and placed in the working directory.

```
-o, --output <file_name>
```

The `-o` switch can be used to define the output name of the file generated with the `-i -r` switch combination. If the `-o` switch is not specified, the file is named `SoCWatchOutput<_trace>.XXX` and placed in the working directory.

5.4 Examples

```
./socwatch -m -f cpu-cstate -t 10
```

Collect processor C-state information with maximum detail for 10 seconds. Generate the `SoCWatchOutput.csv` summary result file and the `SoCWatchOutput.sw2` file.

```
./socwatch -i SoCWatchOutput -r vtune
```

Use the `SoCWatchOutput.sw2` file collected earlier to generate the `SoCWatchOutput.pwr` that can be imported and visualized with Intel VTune Amplifier for Systems.

```
./socwatch --max-detail -f cpu-cstate -f cpu-pstate -f core-temp -t 120
```

Collect processor C-state, processor P-state, and core temperature information for 120 seconds. Polled data, `core-temp` data in this case, will be read at a 100ms (default) interval. Generate the `SoCWatchOutput.csv` summary result file and the `SoCWatchOutput.sw2` file.

```
./socwatch -m -f cpu-cstate -f core-temp -n 1000 -t 30 -o ./result/test1
```

Collect processor C-state and core temperature information with maximum detail for 30 seconds. Polled data will be read at a 1000ms (1 second) interval. Generate the `result/test1.csv` summary result file and the `result/test1.sw2` file.

6 Intel SoC Watch Output Files

Intel SoC Watch generates two files by default after every collection including a summary CSV file and a raw data SW2 file. Intel SoC Watch can optionally generate a summary HTML file, a timed trace CSV file that includes data for each collected metric, and a PWR file that can be imported and visualized with Intel VTune™ Amplifier for Systems.

6.1 Summary CSV File

The summary CSV file provides an overall summary of the data collected for each feature (data type) specified with the `-f` switch.

6.2 Summary HTML File

The summary HTML file provides an overall graphical summary of the data collected for each feature (data type) specified with the `-f` switch and can be viewed with any recent browser.

6.3 Raw Data SW2 File

The raw data SW2 file captures the raw results of the collection in a binary format.

6.4 Timed Trace CSV File

A timed trace CSV file can be optionally generated with the `-r int` switch. The resulting file can be imported into Microsoft* Excel* to be graphed.

6.5 Intel VTune™ Amplifier for Systems PWR Import File

A PWR file can be optionally generated with the `-r vtune` switch. The PWR file can be imported into Intel VTune Amplifier 2016 for Systems or later for visualization and analysis of the data. See the Quick Start section above.

7 Feature Descriptions

The following section describes the different features (i.e. metrics) that can be measured by Intel SoC Watch. Each subsection's title corresponds to the `-f` switch used to collect a particular data type.

all-approx-bw

The `-f all-approx-bw` switch measures multiple different SoC bandwidths concurrently. Each SoC transaction is counted and multiplied by 64 to ***estimate*** each bandwidth. The estimates areequal to or higher than the actual SoC bandwidths. If the transactions executed during a collection include partial or 32 byte transfers, the estimate will be high.

core-temp

The `-f core-temp` switch measures the temperature of each core during the collection. Understanding the core's temperature is important as the core will throttle when it gets too hot. See the Intel 64 and IA-32 Architectures Software Developer's Manual at http://download.intel.com/products/processor/manual/253669.pdf for more information.

cpu-cstate

The `-f cpu-cstate` switch measures Intel architecture core C-states. C-states are power states used to minimize the core, module, or package's power consumption when the core is not actively executing instructions. C-states range from C0 to Cn (for example, C6). C0 is the power state where the core is actively executing instructions. Deeper C-states (for example, C1, C2, and others) consume less power than C0 with Cn consuming the least amount of power. Each C-state deeper than C0 has a different enter and exit latency with the deeper C-states taking longer to enter and exit (return to C0). The -m or --max-detail switch is required to ensure that cpu-cstate collections generate correct results on any Intel Core based systems.

The operating system requests specific C-states using the `mwait` instruction. The processor may accept the request, select a different C-state based on its own status, or abort the C-state. See the Intel 64 and IA-32 Architectures Software Developer's Manual at http://download.intel.com/products/processor/manual/253669.pdf for more information.

cpu-ddr-mod0-bw, cpu-ddr-mod1-bw

The `-f cpu-ddr-mod0-bw` and `cpu-ddr-mod1-bw` switches measure module 0 or module 1 IA core to DDR bandwidth in MB/sec during the collection respectively. The CPU to DDR bandwidth includes all 32 byte and 64 byte read and write accesses to the memory from the CPU components in the north cluster. Only one bandwidth or DRAM Self Refresh data type can be specified at a time.

cpu-pstate

The `-f cpu-pstate` switch measures Intel architecture core P-states. P-states represent voltage-frequency control states defined as performance states in the industry standard Advanced Configuration and Power Interface (ACPI) specification. See http://www.acpi.info for more details. In voltage-frequency control, the voltage and clocks that drive circuits are increased or decreased in response to a workload. Also known as Dynamic Voltage and Scaling Frequency (DVFS), voltage-

frequency control leverages the equation $P=a \times C_{eff} \times V^2 \times f$ that describes the power consumption of a CMOS circuit. P0 represents turbo mode where the processor can operate at a frequency higher than its nominal frequency depending on the state of the other cores, the current thermal headroom, and so on. P0 consumes the most power. P1 is the processor's nominal frequency (advertised frequency), P2 is slower than P1, and so on. Pn consumes the least power.

The operating system requests specific P-states based on the current workload. The processor may accept or reject the request and set the P-state based on its own state. Turbo mode is completely controlled by the hardware.

NOTE: Operating the processor at the maximum P-state to finish the workload as quickly as possible and enter a deep C-state (known as *Race to Halt*) is often the most efficient power management strategy.

ddr-bw

The `-f ddr-bw` switch measures the system's total DDR bandwidth in MB/sec during the collection. The total DDR bandwidth includes all 32 byte and 64 byte read and write accesses to the memory from any component on the SoC including the cores, display, image processing, graphics, and south cluster components. Total DDR bandwidth activity indicates that the SoC is actively executing a workload and is a clue when trying to understand the system's behavior when diagnosing system power consumption issues. Only one bandwidth or DRAM Self Refresh data type can be specified at a time.

disp-ddr-bw

The `-f disp-ddr-bw` switch measures the system's display controller to DDR bandwidth in MB/sec during the collection. Only one bandwidth or DRAM Self Refresh data type can be specified at a time.

dram-bw

The -f dram-bw switch collects total DDR bandwidth during the collection. The output data includes information on memory slice breakdown. Data is retrieved via MMIO reads.

dram-srr

The `-f dram-srr` switch measures the system's DRAM self refresh residency during the collection. The system's DRAM enters a low power self refresh mode when it is not actively being utilized. DRAM self refresh residency should be high during idle workloads. Only one bandwidth or DRAM Self Refresh data type can be specified at a time.

gfx-cstate

The `-f gfx-cstate` switch measures the system's GPU power state residency during the collection. GPU power states range from Render C0 to C6 and Media C0 to C6. The C0 states indicate the GPU is fully powered on and operational while the C6 states indicate the GPU has been placed in the deepest power saving state available.

gfx-ddr-bw

The `-f gfx-ddr-bw` switch measures the system's graphics component to DDR bandwidth in MB/sec during the collection. Only one bandwidth or DRAM Self Refresh data type can be specified at a time.

gfx-pstate

The `-f gfx-pstate` switch polls the system's GPU frequency during the collection. The measurements are reads of the GPU frequency at that point in time – i.e. gfx-pstate measurements do not measure the gfx-pstate since the last measurement and therefore should not be interpreted as 100% accurate. The system can vary the GPU's P-state during the collection in response to the graphics workload in an effort to minimize power consumption.

io-bw

The `-f io-bw` switch measures the system's bandwidth between the north and south cluster in MB/sec during the collection. Only one bandwidth or DRAM Self Refresh data type can be specified at a time.

isp-ddr-bw

The `-f isp-ddr-bw` switch measures the system's image processor to DDR bandwidth in MB/sec during the collection. Only one bandwidth or DRAM Self Refresh data type can be specified at a time.

nc-dstate

The `-f nc-dstate` switch polls the SoC's component power states for the north cluster components *while the SoC is operating in the S0i0 state*. D0ix power states ranges from D0i0 to D0i3, where D0i0 is fully powered on and D0i3 is primarily powered off.

The SoC is organized into a north and south cluster where the compute intensive components (for example, video decode, image processing, and others) are located in the north cluster. The south cluster contains I/O, audio, system management, and other components. SoC components should be in the D0i3 state when not in use.

pkg-temp

The `-f pkg-temp` switch collects temperature data from the following thermal domains on the package: `IA cores, Graphics domain, Camera domain, System Agent domain` and `Package`. These temperatures are read via a combination of MSR and MMIO reads.

pmic-temp

The `-f pmic-temp` switch measures the temperature of the system's Power Management Integrated Circuit (PMIC) during the collection. The PMIC is also called the Mixed Signal Integrated Circuit (MSIC).

sc-dstate

The `-f sc-dstate` switch polls the SoC's component power states for the south cluster components *while the SoC is operating in the S0i0 state*. D0ix power states ranges from D0i0 to D0i3, where D0i0 is fully powered on and D0i3 is primarily powered off.

The SoC is organized into a north and south cluster where the compute intensive components (for example, video decode, image processing, and others) are located in the north cluster. The south cluster contains I/O, audio, system management, and other components. SoC components should be in the D0i3 state when not in use.

skin-temp

The `-f skin-temp` switch measures the system's skin temperature during the collection. The temperatures are read from the sysfs.

soc-temp

The `-f soc-temp` switch measures the system's SoC temperature during the collection.

sstate

The `-f sstate` switch measures the residency in the Intel SoC idle standby power states and the system's residency in the ACPI Suspend-To-RAM (S3) state. The S0ix states shut off parts of the SoC when they are not in use. The S0ix states are triggered when specific conditions within the SoC have been achieved – for example, certain SoC components are in low power states. See http://landley.net/kdocs/ols/2012/ols2012-mansoor.pdf for a full description of the S0ix power states. The SoC consumes the least amount of power in the deepest (for example, S0i3) state.

In the Suspend-To-RAM state, the Linux kernel powers down many of the systems' components while maintaining the system's state in its main memory. The system consumes the least amount of power possible while in the Suspend-To-RAM state. Any wakelock currently held prevents the system from entering the Suspend-To-RAM state. See the description of Suspend-To-RAM at http://www.linuxsymposium.org/archives/OLS/Reprints-2008/brown-reprint.pdf.

wakelock (Android only)

The `-f wakelock` switch traces each user mode and kernel mode wakelock used by the system's software to determine which wakelock or wakelocks are preventing the system from entering ACPI S3.

8 Using Intel® VTune™ Amplifier for Systems to Analyze Intel SoC Watch Data

You can analyze Intel SoC Watch data graphically using the Intel VTune Amplifier for Systems GUI. The GUI provides a dynamic timeline view for interacting with Intel SoC Watch data and provides powerful filtering of data for in-depth analysis of a platform's power management behavior. Intel VTune Amplifier 2016 for Systems Update 3 or later is recommended to analyze data collected with Intel SoC Watch v2.3.

8.1 Importing Data

To import data using Intel VTune Amplifier for Systems, perform the following steps:

1. Collect Intel SoC Watch data using the `-r vtune` option during collection or post-process the raw data (SW2 file) as you normally would on your target system. You may specify any output file name, but these instructions are written assuming you use the default output file name `SoCWatchOutput.pwr`.
2. Copy the raw data file back to the host development system:
3. Launch the Intel VTune Amplifier for Systems GUI:
   - Linux: `/path-to-vtune/bin32(bin64)/amplxe-gui`
   - Windows: `\path-to-vtune\bin32(bin64)\amplxe-gui`
4. Open the result by clicking **Import Result**.

5. Navigate to the PWR file. Click **Import**.

8.2 Using the Intel VTune Amplifier for Systems GUI

Intel VTune Amplifier for Systems has a tab for each type of Intel SoC Watch data collected. All of the tabs have the same basic layout consisting of a timeline, grid and filter area. These areas provide various mechanisms for manipulating and viewing the data.

8.2.1 Filtering

Each of the areas enables you to filter data to help focus the analysis on specific areas of the collection. Note that a filter from one part of the GUI affects the data shown in other parts of the GUI. This includes data not only within a single tab, but also includes data shown in other tabs.

The most common filtering is selecting a region of time to focus on. To do this, use the timeline by selecting a region of time and selecting a filter action. This restricts the data in the grid to only the data that was collected during the selected period of time.

A common choice is the `Zoom In and Filter In by Selection` filter. This filter zooms the timeline and adjusts the grid to include only the data shown in the timeline (for example, the grid only includes the C-state residency or wakeups shown in the zoomed timeline).

It is also possible to filter data using the grid by selecting rows of data, right-clicking to bring up the menu and selecting the filter action.

This filter then updates the timeline to show only the data associated with the selected rows in the grid. For example, this is useful when looking at data in the CPU Wake-up tab. By filtering on the selected rows, you can see the behavior of just those wakeup sources in the timeline.

Before filtering

After filtering

The filter area behaves similar to the grid filter in action and is useful for quickly doing the most common and simple filtering actions. The filter area also has the button used to remove all the filters.

8.2.2 Expanding Grid Columns

The grid has an ability to expand certain columns of data to obtain more information. If a column is expandable, you see a clickable >> symbol in the header of the column. Clicking on this symbol expands the column.

Grid data can also be shown as raw values (for example, time in a specific C-state), percentages, or other representations. Right click on a value in the grid and select **Show Data As** and then select the desired representation.

8.2.3 Changing Grid Groupings

Certain tabs have grids that can show the data using multiple row hierarchies. Use the **Grouping** drop-down box and select the desired grouping to change the row hierarchy.

8.2.4 Timeline Zooming

Using the zoom functionality in the timeline makes it possible to do finer grain filtering and to examine individual discrete events. For example, when zoomed out, this tooltip says that there are 96 wakeups at a given point. By zooming in, it is possible to select each of these individual wakeups.

9 Intel SoC Watch Tips

If the collection needs to be performed untethered (that is, not connected to a host via a USB cable), use the nohup command with a start delay when launching Intel SoC Watch and then disconnect the USB cable. The nohup command causes the terminal to ignore the hangup signal which would normally log the user out of the terminal when the USB cable is pulled. For example, use the following command to launch a 10 second collection after a start delay of 5 seconds. During the 5 second start delay, disconnect the USB cable from the device.

```
> nohup ./socwatch -s 5 -f cpu-cstate -t 10 &
```

Use the `-o console` switch to send the summary output results to the console in addition to the `console.csv` file.

If you see C-state wakeups caused by timers with either the thread ID or the process ID set to -1, then double-check your kernel parameters: this sometimes occurs if the kernel was not compiled with the `CONFIG_TIMER_STATS=y` option. In addition, under certain (rare) conditions, it is possible to have a wakeup caused by a timer for which Intel SoC Watch captures a thread ID but not a process ID. In this case, Intel SoC Watch assigns the thread ID to the process ID.

Users should consider graphing the timed trace results with Microsoft Excel. In order to graph results with Excel, open the `.csv` file in a text editor and copy the text into Excel to clear additional formatting. Removing this extra formatting will reduce readability in Excel, but allows you to create charts based on the data provided using Excel's charting functions, available from the **Insert** toolbar.

9.1 IPI Wakeups

Occasionally, interrupt, timer, workqueue execution, or sched_wakeup tracepoints do not occur before a core is placed again in a C-state. In other words, the C-state entry tracepoint occurs twice without an interrupt, timer, workqueue execution, or sched_wakeup tracepoint occurring in between. The associated wakeup may be caused by an IPI or Inter Processor Interrupt. IPIs are sent from one processor to another for different reasons[1]. Intel SoC Watch monitors some per-CPU kernel IRQ statistics[2] at the same time as the TSC, MPERF, and C-state residency MSRs are read. If the IPI counts increment during a C-state sample and an interrupt, timer, workqueue execution, or sched_wakeup

[1] See http://en.wikipedia.org/wiki/Inter-processor_interrupt.

[2] See the irq_stat struct in the struct_irq_cpustat_t struct in the arch/x86/include/asm/hardirq.h file.

tracepoint did not occur, the sample's break type is set to IPI. Experiments were done with patched kernels[3] to confirm the validity of this strategy.

9.2 Submitting an issue/Getting Help

To submit a Intel SoC Watch bug you have the following options.

If you have a license for the Intel® System Studio, and you did not register your product during installation, please do so at the Intel Registration Center at https://registrationcenter.intel.com/regcenter/register.aspx. Registration entitles you to free technical support, product updates and upgrades for the duration of the support term.

If you do not have a license for the Intel® System Studio, please submit your issue via the user forums at https://software.intel.com/en-us/forums/intel-system-studio .

[3] See the smp_reschedule_interrupt and smp_call_function_interrupt functions in the arch/x86/kernel/smp.c file and the local_apic_timer_interrupt function in the arch/x86/kernel/apic/apic.c file.

Appendix A: Acronyms and Definitions

ACPI: Advanced Configuration and Power Interface
DDR: Double Data Rate SDRAM
DVFS: Dynamic Voltage Frequency Scaling
MSIC: Mixed Signal Integrated Circuit
MSR: Model-Specific Register
PMIC: Power Management Intergrated Circuit
SoC: System on a Chip
TSC: Time Stamp Counter