Intel® SoC Watch 0.3 for Android* on SoFIA

User Guide

Legal Information

Revision History

Revision Number	Description	Revision Date
0.2	Initial release.	October 2014
0.3	Bug fix release	October 2014

Contents

§

1 About this Document

This document describes the Intel® SoC Watch for Android tool for the SoFIA platform. Note that SoC Watch is also a component within the Intel Energy Profiler tool.

1.1 Intended Audience

You should read this document if you are a 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.

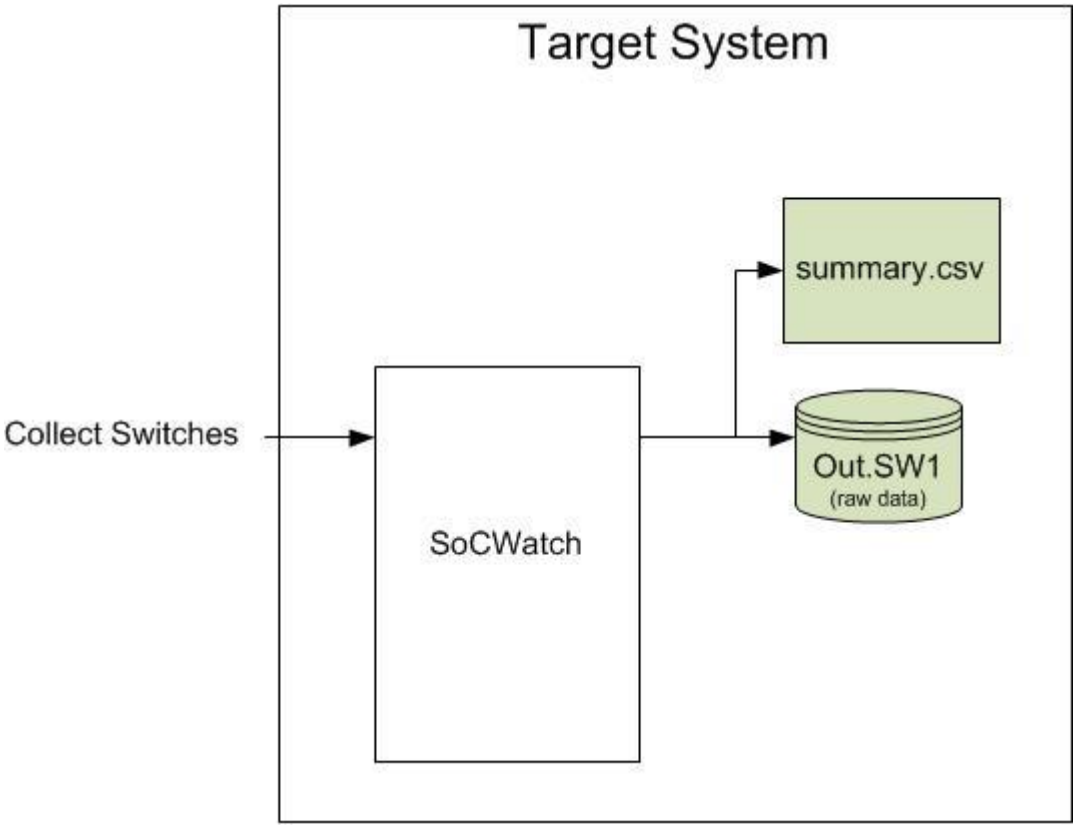
2 Overview

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 (SW1) file are produced by default on the target system (which is the system being analyzed).

Figure 1: Collecting SoC Watch Data and Generating Output Files



2.1 Requirements

Intel® SoC Watch for Android on SoFIA works on systems based on the following platforms:

- SoFIA

See the socwatch.intel.com wiki for detailed system requirements.

2.2 Building the Kernel Module

If the SoC Watch device drivers are not preinstalled in the OS image, you will need to build them.

1. Copy the `socwatch_sofia_android_XXX.zip` file to the host system used to build the target system's Android* kernel.
2. Extract the contents with the command:

   ```
   > unzip socwatch_sofia_android_XXX.zip
   ```

   The `socwatch_sofia_android_XXX` directory is created.
3. cd into the socwatch_android/socwatch_driver/lx_kernel directory.
4. Execute the `build_driver` script with the command:

   ```
   > sh ./build_driver -k <kernel-build-dir>
   ```

   where `<kernel-build-dir>` is replaced with the local Android* lib/modules directory produced while building the Android* kernel. The `-k` switch is optional and is not required if the `DEFAULT_KERNEL_BUILD_DIR` value is properly set in the `build_driver` script.

2.3 Installation

The Intel® SoC Watch device drivers are integrated into several Android* distributions based on Intel® Architecture. Note that the socwatch executable files are not integrated into the distribution and need to be installed with the socwatch_android_install script or BAT files. Before installing the socwatch executable files, make sure the appropriate drivers are installed and the socwatch driver version matches the version of the executable to be installed. For Intel® SoC Watch v0.3, the socwatch0_3.ko file should be installed. You can check the existence of the socwatch0_3.ko driver with the following commands;

- adb shell ls /lib/modules/socwatch0_3.ko
- adb shell ls /system//lib/modules/socwatch0_3.ko

If the driver is not available on the system, you will need to build it. See the *Building the Kernel Modules* section.

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

1. After unzipping 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 the `socwatch_android_install.bat` file 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 driver.

   ```
   > cd /system/lib/modules
   ```

4. Load the driver.

```
> insmod socwatch0_3.ko
```

5. Confirm the driver is loaded.

```
> lsmod
```

When necessary, type `rmmod rmmod socwatch0_3` to unload the driver.

2.4 Quick Start

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

Use the following steps, targeted for an Ubuntu*-based host, to quickly collect processor vm-switch and Android* wakeup data for 60 seconds 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 driver.

```
> insmod /system/lib/modules/socwatch0_3.ko
```

3. Confirm the driver is loaded.

```
> lsmod
```

Confirm the Intel® SoC Watch driver (`socwatch0_2`) is included in the list of installed modules.

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

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

5. Collect data and generate the `test.csv` and `test.sw1` files in the results directory. This step assumes the `/data/socwatch/result` directory exists.

```
>./socwatch –f vm-switch –f wakeup –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 /results
> adb pull /data/socwatch/results/test.sw1 /results
```

8. Review the summary. Open the `/results/test.csv` file.

Table 2: Key Files

`socwatch_android_install.sh`	The Linux install script. It creates a `/data/socwatch` directory on the target system and copies the necessary files there via the `adb push` shell command.
`socwatch_android_install.bat`	The Windows install script. It creates a `/data/socwatch` directory on the target system and copies the necessary files there via the `adb push` shell command.
`socwatch_driver/lx_kernel/build_driver`	The build script used to build the Intel® SoC Watch device driver for Android* systems.

`/lib/modules/socwatch0_3.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 Android* application. Use this file on Android* systems to collect data and generate additional results from a raw SW1 file.
`eula.txt`	End User License Agreement file.
`third-party-programs.txt`	Dual BSD/GPLv2 licenses

3 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.

NOTE: The first . character followed by a space must be specified in order to set the environment variables for the shell. Failure to include the initial character (period) will cause the Intel® SoC Watch execution step to fail.

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

To launch a collection, type

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

3.1 General Options

`-h, --help:`

Display usage information and exit.

`-v, --version:`

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

3.2 Collection Options

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

Specify the data to be collected. See the *Available Features* section below. 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 below.

`-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.sw1`. If the `-o` switch is not specified, data is saved to the `SoCWatchOutput.csv` and `SoCWatchOutput.sw1` files in the current directory. To create additional files, use the `-r` switch described below.

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

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

- the duration specified by the `-t` switch has elapsed

3.2.1 Available Features

The available data types that Intel® SoC Watch can collect and their possible collection methods are listed below. Detailed descriptions of the different kinds of data are provided in the *Data Type Descriptions* section below.

vm-switch

Collection Method(s): trace

Trace virtual machine context switch activity. Switches between the VMM, Android VM, Modem VM, and Secure VM are tracked. The residency of the VMM and the Guest OSes are provided. VM Exit causes are also described in the results.

wakeup

Collection Method(s): trace

Trace Android wakeups from VCPU sleep states. Android wakeups describe the VMM wakeups caused by the Android Guest OS.

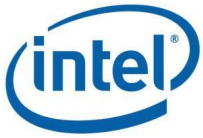
3.3 Examples

```
./socwatch -f vm-switch -t 10
```

Trace vm-switch activity on the system's physical cores. processor Generate the `SofiaSoCWatchOutput.csv` summary result file and the `SofiaSoCWatchOutput.sw1` file.

```
./socwatch -f wakeup -t 10
```

Collect Android OS VCPU wakeup information. Generate the `SofiaSoCWatchOutput.csv` summary result file and the `SofiaSoCWatchOutput.sw1` file.



4 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 SW1 file.

Table 3: Intel SoC Watch Output Files

Output File Type	Description
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.
Raw Data SW1 File	The raw data SW1 file captures the raw results of the collection in a binary format.

5 Data Type Descriptions

The following section describes the different data types that can be collected by Intel® SoC Watch. Each subsection's title corresponds to the `-f` switch used to collect a particular data type.

vm-switch

The `-f vm-switch` switch traces virtual machine context switch activity. Switches between the VMM, Android Guest VM, Modem Guest VM, and Secure Guest VM are tracked. The residency of the VMM and the Guest OSes are provided. VM Exit causes are also described in the results.

wakeup

The `-f wakeup` switch traces Android Guest OS wakeups from VCPU sleep states. Android wakeups describe the VMM wakeups caused by the Android Guest OS.

6 Interpreting Summary Results

6.1 vm-context Data

The following tables provide an example vm-switch result.

SOFIA VM residencies

	VMM	MODEM	ANDR	SECVM
Core 0	69.44	8.232	20.96	0
Core 1	81	11.83	5.92	0

SOFIA VM Exit causes

	EXT_INTERRUPT	INTERRUPT_WINDOW	HLT	VMCALL
MODEM	0	0	8991	3
ANDR	1491	1092	4456	11871
SECVM	0	0	0	0
Total	1491	1092	13447	11876

The *SOFIA VM residencies* table describes the *percentage* of total collection time the VMM and each guest OS operated during the collection. If cpu-cstates are not collected, the data provided by this table are not 100% accurate as time spent in cstates is not taken into account. When this is the case, the following warning will be printed;

```
The following VM residency data does not take into account C-state behavior!
```

In the above example, core 0 spent 69.44% of the total collection time executing the VMM, 8.232% of the total collection time executing the Modem guest OS, and 20.96% of the total collection time executing the Android guest OS.

The *SOFIA VM Exit causes* table describes the different reasons each guest OS exits to the VMM. A transition from Guest OS execution to VMM execution is called a VM Exit.

6.2 wakeup Data

The following tables provide an example wakeup result.

Android Wake-up Reasons

IRQ	TIMER	SCHEDULER	IPI	WORK QUEUE	ABORT	UNKNOWN
10284	377	0	0	9	0	0

Top 10 Android Wake-up Reasons

Category	IRQ Num	PID	TID	Count
IRQ	48			3981
IRQ	1			3739
IRQ	43			2412
TIMER		506	524	277
TIMER		106	133	81
IRQ	46			35
IRQ	39			35
IRQ	50			34
IRQ	289			29
IRQ	33			11

The *Android Wake-up Reasons* table provides an overall summary of Android VCPU wakeup behavior. In the above example, 10284 interrupts, 377 timers, and 9 workqueu executions caused VCPU wakeups.

The *Top 10 Android Wake-up Reasons* table breaks down the top ten Android VCPU wakeup reasons. Interrupts and timers are broken out by their IRQ numbers and Process & Thread IDs respectively. In the above example, IRQ 48 was the top cause of Android wakeups. It caused 3981 Android wakeups. Process ID 506, using Thread ID 524, caused 277 wakeups during the collection.