Measuring Component Power

You can determine individual component power consumption by comparing the current drawn by the device when the component is in the desired state (on, active, scanning, etc.) and when the component is off. Measure the average instantaneous current drawn on the device at a nominal voltage using an external power monitor, such as a bench power supply or specialized battery-monitoring tools (such as Monsoon Solution Inc. Power Monitor and Power Tool software).

Manufacturers often supply information about the current consumed by an individual component. Use this information if it accurately represents the current drawn from the device battery in practice. However, validate manufacturer-provided values before using those values in your device power profile.

Controlling power consumption

When measuring, ensure the device does not have a connection to an external charge source, such as a USB connection to a development host used when running Android Debug Bridge (adb). The device under test might draw current from the host, thus lowering measurements at the battery. Avoid USB On-The-Go (OTG) connections, as the OTG device might draw current from the device under test.

Excluding the component being measured, the system should run at a constant level of power consumption to avoid inaccurate measurements caused by changes in other components. System activities that can introduce unwanted changes to power measurements include:

- **Cellular, Wi-Fi, and Bluetooth receive, transmit, or scanning activity**. When not measuring cell radio power, set the device to airplane mode and enable Wi-Fi or Bluetooth as appropriate.
- **Screen on/off**. Colors displayed while the screen is on can affect power draw on some screen technologies. Turn the screen off when measuring values for non-screen components.
- **System suspend/resume**. A screen off state can trigger a system suspension, placing parts of the device in a low-power or off state. This can affect power consumption of the component being measured and introduce large variances in power readings as the system periodically resumes to send alarms, etc. For details, see Controlling system suspend (#control-suspend).
- **CPUs changing speed and entering/exiting low-power scheduler idle state**. During normal operation, the system makes frequent adjustments to CPU speeds, the number of online CPU cores, and other system core states such as memory bus speed and voltages of power rails associated with CPUs and memory. During testing, these adjustments affect power measurements:
  - CPU speed scaling operations can reduce the amount of clock and voltage scaling of memory buses and other system core components.
  - Scheduling activity can affect the percentage of the time CPUs spend in low-power idle states. For details on preventing these adjustments from occurring during testing, see Controlling CPU speeds (#control-cpu).

For example, Joe Droid wants to compute the `screen.on` value for a device. He enables airplane mode on the device, runs the device at a stable current state, holds the CPU speed constant, and uses a partial wakelock to prevent system suspend. Joe then turns the device screen off and takes a measurement (200mA). Next, Joe turns the device screen on at minimum brightness and takes another measurement (300mA). The `screen.on` value is 100mA (300 - 200).

Note: For components that don't have a flat waveform of current consumption when active (such as cellular radio or Wi-Fi), measure the average current over time using a power monitoring tool.

When using an external power source in place of the device battery, the system might experience problems due to an unconnected battery thermistor or integrated fuel gauge pins (i.e. an invalid reading for battery temperature or remaining battery capacity could shut down the kernel or Android system). Fake batteries can provide signals on thermistor or fuel gauge pins that mimic temperature and state of charge readings for a normal system, and may also provide convenient leads for connecting to external power supplies. Alternatively, you can modify the system to ignore the invalid data from the missing battery.

Controlling system suspend

This section describes how to avoid system suspend state when you don't want it to interfere with other measurements, and how to measure the power draw of system suspend state when you do want to measure it.

Preventing system suspend

System suspend can introduce unwanted variance in power measurements and place system components in low-power states inappropriate for measuring active power use. To prevent the system from suspending while the screen is off, use a temporary partial wakelock. Using a USB cable, connect the device to a development host, then issue the following command:

```
$ adb shell "echo temporary > /sys/power/wake_lock"
```

While in **`wake_lock`**, the screen off state does not trigger a system suspend. (Remember to disconnect the USB cable from the device before measuring power consumption.)

To remove the wakelock:

```
$ adb shell "echo temporary > /sys/power/wake_unlock"
```

Measuring system suspend

To measure the power draw during the system suspend state, measure the value of **`cpu.idle`** in the power profile. Before measuring:

- Remove existing wakelocks (as described above).
- Place the device in airplane mode to avoid concurrent activity by the cellular radio, which might run on a processor separate from the SoC portions controlled by the system suspend.
- Ensure the system is in suspend state by:
  - Confirming current readings settle to a steady value. Readings should be within the expected range for the power consumption of the SoC suspend state plus the power consumption of system components that remain powered (such as the USB PHY).
  - Checking the system console output.
  - Watching for external indications of system status (such as an LED turning off when not in suspend).

Controlling CPU speeds

Active CPUs can be brought online or put offline, have their clock speeds and associated voltages changed (possibly also affecting memory bus speeds and other system core power states), and can enter lower power idle states while in the kernel idle loop. When measuring different CPU power states for the power profile, avoid the power draw variance when measuring other parameters. The power profile assumes all CPUs have the same available speeds and power characteristics.

While measuring CPU power, or while holding CPU power constant to make other measurements, keep the number of CPUs brought online constant (such as having one CPU online and the rest offline/hotplugged out). Keeping all CPUs except one in scheduling idle may product acceptable results. Stopping the Android framework with **`adb shell stop`** can reduce system scheduling activity.

You must specify the available CPU speeds for your device in the power profile **`cpu.speeds`** entry. To get a list of available CPU speeds, run:

```
$ adb shell cat /sys/devices/system/cpu/cpu0/cpufreq/stats/time_in_state
```

These speeds match the corresponding power measurements in value **`cpu.active`**.

For platforms where number of cores brought online significantly affects power consumption, you might need to modify the cpufreq driver or governor for the platform. Most platforms support controlling CPU speed using the userspace cpufreq governor and using sysfs interfaces to set the speed. For example, to set speed for 200MHz on a system with only 1 CPU or all CPUs sharing a common cpufreq policy, use the system console or adb shell to run the following commands:

```
$ echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
$ echo 200000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
$ echo 200000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
$ echo 200000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
```

Note: The exact commands differ depending on the platform cpufreq implementation.

These commands ensure the new speed is not outside the allowed bounds, set the new speed, then print the speed at which the CPU is actually running (for verification). If the current minimum speed prior to execution is higher than 200000, you might need to reverse the order of the first two lines, or execute the first line again to drop the minimum speed prior to setting the maximum speed.

To measure current consumed by a CPU running at various speeds, use the system console to place the CPU in a CPU-bound loop using the command:

```
# while true; do true; done
```

Take the measurement while the loop executes.

Some devices can limit maximum CPU speed while performing thermal throttling due to a high temperature measurement (i.e. after running CPUs at high speeds for sustained periods). Watch for such limiting, either using the system console output when taking measurements or by checking the kernel log after measuring.

For the **`cpu.awake`** value, measure the power consumed when the system is not in suspend and not executing tasks. The CPU should be in a low-power scheduler *idle loop* , possibly executing an ARM Wait For Event instruction or in an SoC-specific low-power state with a fast-exit latency suitable for idle use.

For the **`cpu.active`** value, measure power when the system is not in suspend mode and not executing tasks. One CPU (usually the primary CPU) should run the task while all other CPUs should be in an idle state.

Measuring screen power

When measuring screen on power, ensure that other devices normally turned on when the screen is enabled are also on. For example, if the touchscreen and display backlight would normally be on when the screen is on, ensure these devices are on when you measure to get a realistic example of screen on power usage.

Some display technologies vary in power consumption according to the colors displayed, causing power measurements to vary considerably depending on what is displayed on the screen at the time of measurement. When measuring, ensure the screen is displaying something that has power characteristics of a realistic screen. Aim between the extremes of an all-black screen (which consumes the lowest power for some technologies) and an all-white screen. A common choice is a view of a schedule in the calendar app, which has a mix of white background and non-white elements.

Measure screen on power at *minimum* and *maximum* display/backlight brightness. To set minimum brightness:

- **Use the Android UI** (not recommended). Set the Settings > Display Brightness slider to the minimum display brightness. However, the Android UI allows setting brightness only to a minimum of 10-20% of the possible panel/backlight brightness, and does not allow setting brightness so low that the screen might not be visible without great effort.
- **Use a sysfs file** (recommended). If available, use a sysfs file to control panel brightness all the way down to the minimum brightness supported by the hardware.

Additionally, if the platform sysfs file enables turning the LCD panel, backlight, and touchscreen on and off, use the file to take measurements with the screen on and off. Otherwise, set a partial wakelock so the system does not suspend, then turn on and off the screen with the power button.

Measuring Wi-Fi power

Perform Wi-Fi measurements on a relatively quiet network. Avoid introducing additional work processing high volumes of broadcast traffic that is unrelated to the activity being measured.

The **`wifi.on`** value measures the power consumed when Wi-Fi is enabled but not actively transmitting or receiving. This is often measured as the delta between the current draw in system suspend (sleep) state with Wi-Fi enabled vs. disabled.

The **`wifi.scan`** value measures the power consumed during a Wi-Fi scan for access points. Applications can trigger Wi-Fi scans using the WifiManager class **`startScan()`** API (http://developer.android.com/reference/android/net/wifi/WifiManager.html). You can also open Settings > Wi-Fi, which performs access point scans every few seconds with an apparent jump in power consumption, but you must subtract screen power from these measurements.

Note: Use a controlled setup (such as iperf (http://en.wikipedia.org/wiki/Iperf)) to generate network receive and transmit traffic.