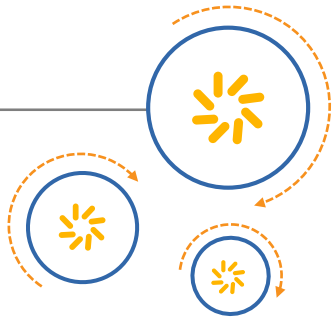
Qualcomm Technologies, Inc.



Camera Power Debugging Guide

80-NP961-1 D

April 18, 2016

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Revision history

Revision	Date	Description
A	July 2014	Initial release
B	November 2014	▪ Updated Section 2.1.3, 2.2, 2.3, and 2.4 ▪ Added Chapter 4, 5, and Appendix A
C	February 2015	▪ Added Section 4.8 ▪ Updated Chapters 4, 5, and Appendix A
D	April 2016	▪ Added Section 6

Contents

Figures

Tables

1 Introduction

1.1 Purpose

This document explains camera features and settings that affect power consumption and provides power debug tips for camera-specific use cases, including camera preview, snapshot, and video recording. Information in this document is generic to camera debugging across MSM8974, MSM8916, and newer chipsets.

1.2 Conventions

Function declarations, function names, type declarations, attributes, and code samples appear in a different font, for example, `#include`.

Code variables appear in angle brackets, for example, `<number>`.

Shading indicates content that has been added or changed in this revision of the document.

1.3 Technical assistance

For assistance or clarification on information in this document, submit a case to Qualcomm Technologies, Inc. (QTI) at https://support.cdmatech.com/.

If you do not have access to the CDMATech Support website, register for access or send email to support.cdmatech@qti.qualcomm.com.

2 Camera features

This chapter describes the camera features that affect power consumption during camera-specific use cases and explains how these features are controlled. The details provided are with reference to the QTI camera application, which is subject to vary on OEM targets based on the app selection.

In the following sections, the power consumption quantification is shared upon disabling and enabling the various features or with different settings. However, power saving by disabling any feature could come at the cost of the camera image quality or user experience. It is recommended that changes are reviewed by the customer camera IQ and the user experience team before implementation.

2.1 3A algorithm

- Auto White Balance (AWB) – Compensates for the influence of illuminants on a scene
- Auto Exposure Control (AEC) – Automatically adjusts the overall image brightness to a target
- Auto Focus (AF) – Helps find the right focus point for a given scene by adjusting lens position

2.1.1 AWB

AWB reproduces the colors of a scene, mimicking what is seen by the human eye. The basic AWB algorithm runs on an ARM processor and is a three-step process which:

1. Collects statistics from ISP hardware
2. Analyzes the statistics and compares the collected data to reference points
3. Applies AWB gains

2.1.2 AEC

AEC is used to set up the camera sensor to achieve a brightness value within a desired range. The basic AEC algorithm runs on an ARM processor and is a four-step process which:

1. Collects statistics for AEC from ISP hardware
2. Calculates brightness value
3. Determines exposure adjustments
4. Updates camera settings

2.1.3 AF

AF gets the scene in focus by adjusting the lens. The basic AF algorithm runs on an ARM processor and is a two-step process which:

1. Collects statistics for AF from ISP hardware
2. Captures a series of focus values to efficiently determine the optimal focus position

AF is on-demand by an application that calls the autofocus() API defined by the Android camera specs (see http://developer.android.com/reference/android/hardware/Camera.html#autoFocus(android.hardware.Camera.AutoFocusCallback) or run continuously). Continuously running AF or Continuous Auto Focus (CAF) gives improved user experience. Hence, it is used by the latest high-end devices in the market (specifically, devices using the ZSL mode camera operation), where the camera preview stays focused and looks sharp without much user intervention, thus reducing picture capture latency. However, it means a marginally higher power consumption, when compared to on-demand AF. CAF is enabled using the key KEY_QC_CONTINUOUS_AF, which is set via the API setContinuousAf() exposed in frameworks/base/core/java/android/hardware/Camera.java.

2.2 Face Detection (FD)

FD identifies facial regions of interest. The basic FD algorithm runs on an ARM processor and is a three-step process which:

1. Receives frames sent from ISP hardware
2. Outputs coordinates of where faces are located
3. Sends output to other algorithms for use or to the display to draw indications of the detected faces in the scene

Applications often perform special face-based processing or 3A emphasis on detected faces, hence not enabling this feature could impact user experience and image quality.

FD is enabled via the startFaceDetection() API, defined by Android camera specs (see http://developer.android.com/reference/android/hardware/Camera.html#startFaceDetection%28%29).

2.3 Zero Shutter Lag (ZSL)

Shutter lag is the delay between the moment the user takes a picture and the moment the captured snapshot frame is available. This delay often consists of user reaction time, UI latency to handle touch event, latency for the function call to traverse from the application until the QTI camera HAL layer and QTI camera code transitioning the camera state from preview to snapshot, and generating a snapshot frame.

In ZSL mode, camera is set to run in Full Size Output mode and pick a frame upon the QTI camera HAL receiving the takePicture() command, to compensate for the shutter lag.

In ZSL mode, since the camera operates in its full output size configuration, the power requirement is greater compared to non-ZSL, where the camera preview runs in the Subsampled mode. Also, based on the sensor capability, the MIPI lane configuration, and the maximum data rate capability of MSM, the maximum frame rate achievable in the ZSL mode varies, which is to

be confirmed by the users. Alternately, disabling ZSL causes noticeable impact to user experience.

2.4 Full or video size live snapshot

The Live Snapshot feature enables a still image capture during video recording.

The QTI software is configured to support either a video or full size live snapshot. For video size live snapshot, camera is configured at its default video resolution configuration. For full size live snapshot, camera is configured at Full Size Output mode. When a full size live snapshot is enabled, the power consumption and the maximum frame rate impacts are similar to those described in Section 2.3. These impacts need to be considered, since the camera sensor operates in full size output configuration.

2.5 Features that affect camera use cases power consumption

Table 2-1 Camera features that affect power consumption

Feature	Affects camera preview power	Default setting after boot up (camera preview)	Affects camcorder recording	Setting used for power dashboard measurement (camcorder recording)	Runs on
3A – AEC	Y	ON	Y	On	CPU
3A – AWB	Y	Auto	Y	Off (Incandescent)	CPU
3A – AF	Y	OFF	Y	On	CPU
FD	Y	ON	Y	Off	CPU
ZSL	Y	ON	NA	NA	Camera pipeline (CPU. ISP/CPP)

Note: Y – Applicable, NA – Not applicable

3 Camera power debugging

This chapter describes the power debugging for camera specific use cases.

3.1 Factors that affect camera power

- Hardware differences, such as camera sensor configuration and ISP, can affect power consumption
- Difference in fps results in power differences, with respect to QTI device measurement
- Differently configured camera software features result in power differences.
- Other factors listed in Table 2-1.

3.2 General camera power debug

1. Obtain the following OEM hardware specifications:
   - Display
     - Resolution
     - Display type
     - Video/command
   - DRR
     - Size
   - Camera
     - Size
     - Output width and height
2. Check if there are any software algorithm changes, for example, AF, AEC, and AWB, as detailed in Chapter 2. It is recommended to account for software and hardware differences, while debugging the power gap between the device and the QTI reference device.

To check fps information:

1. Type this ADB command to check the preview fps type:

```
adb shell
    setprop persist.debug.sf.showfps 1
 adb logcat | find "PER_SEC"
```

2. Launch the camera to see the fps information on the command prompt window.
3. Check the video encode fps by playing the recorded clip on QuickTime player.

4. Check CPU utilization, processes, multimedia subsystem clocks, and rail levels as described in Section 3.3.
5. Pull out the recorded clip and play under QuickTime player to check the fps.

3.3 Analyze data

This section is divided into the following four parts:

- CPU analysis
- Clock-level analysis
- MSM bus request debugging
- Rail-level analysis

3.3.1 CPU analysis

3.3.1.1 PowerTOP

The PowerTOP tool identifies specific components of kernel and user space applications that frequently wake up the CPU. The power savings are higher, since the CPU is in Idle state for the maximum time. PowerTOP provides CPU profiling information of the system, as shown in Figure 3-1. It is advisable to simultaneously capture the PowerTOP data and the battery power measurements, to ensure that the PowerTOP data is aligned with the battery power measurements on the device.

Figure 3-1 PowerTOP log explanation

PowerTOP logs using the -d option capture information for 15 sec. To capture PowerTOP data for a specific duration, use:

```
powertop -d –t 60
```

where, -t indicates time in seconds.

If the -d option is supplied alone, the data is captured for 15 sec. All msmpmstats Idle state information is captured from this duration. For detailed information, msm_pm_stats logs are captured.

The CPU power difference depends on the duration the CPU was busy, the residency at each frequency level, and the duration for which the CPU was resident under each C0...C3 low power states.

Figure 3-2 demonstrates how to analyze PowerTOP data on good devices, that is, a QTI reference device; and bad devices, that is, a device with comparatively high power consumption, for a static image display use case.

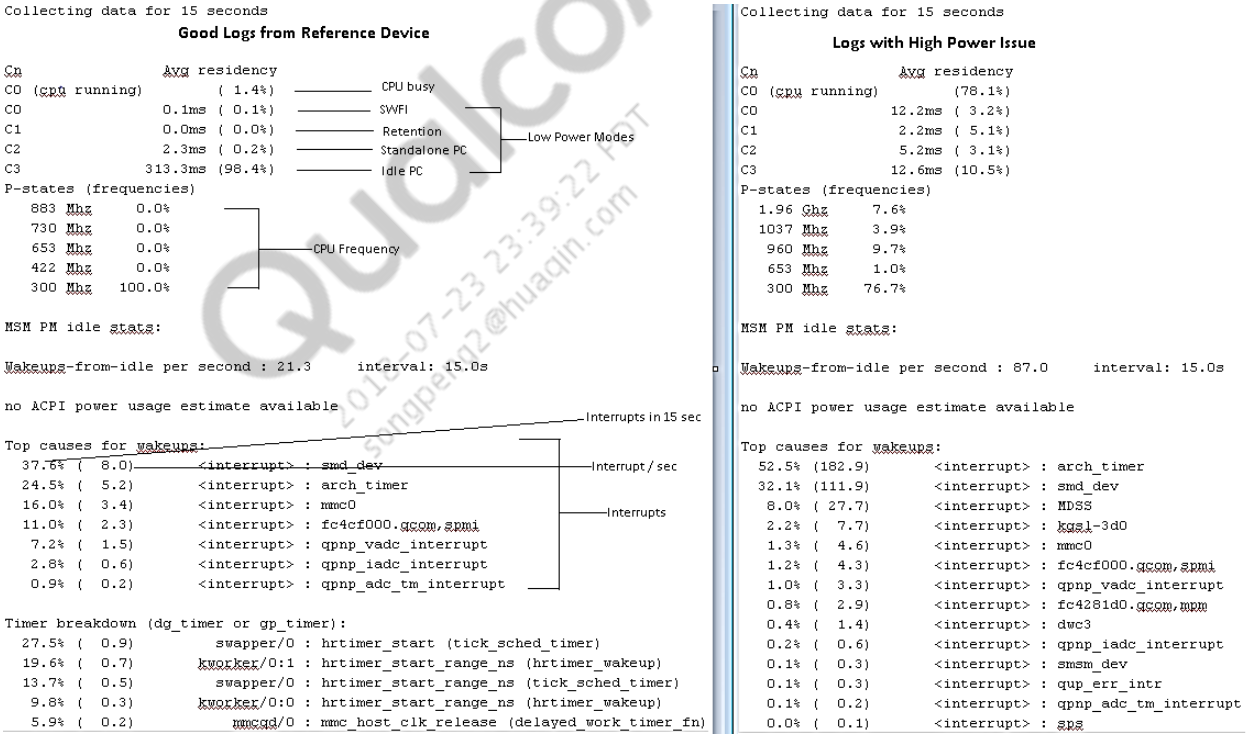


Figure 3-2 PowerTOP analysis for static image display use case

To analyze the PowerTOP data:

1. Compare the PowerTOP data between the QTI reference device and the customer device.
2. Check C0 (CPU running), which signifies the total percentage of duration that the CPU was busy.
3. Check the percentage of time the device was in different C states, that is, C0, C3, and so on, of Low Power modes.
4. Check the difference in the number of wakeups from idle per second; these wakeups prevent the device from entering into power collapse.

5. Check the top causes of wakeups, that is, wakeup interrupts, for the differences. The top cause of the wakeup is vital information, since it lists the interrupts that prevent the CPU from entering into power collapse. Optimizing these interrupts results in power savings.
6. Check the CPU frequency by observing the P-states.

   If the CPU frequency is higher on a device, it leads to high-power consumption PLLs/CX. This needs to be checked in conjunction with C0, that is, CPU running.
7. Check the wakeup from the idle field, which indicates the number of times the CPU wakes up from idle every second due to the interrupts as discussed above.

Limitations

PowerTOP logs provide CPU status comprising all CPU cores; as such, it is difficult to identify how many CPU cores were running.

3.3.1.2 Top

This utility captures the threads and process-level information. System percentage gives the overall CPU busy information and IOW is the input/output wait time for which the CPU is waiting. To analyze Top data:

1. Check the system and IOW percentages and match the percent differences with the reference device top data.
2. Check for the process consuming the majority of CPU cycles and also for any processes that are not supposed to run during a particular use case.

Figure 3-3 is an example of the top statistics showing the process-level information for every second.

```
User 2%, System 18%, IOW 0%, IRQ 0%
User 9 + Nice 0 + Sys 79 + Idle 326 + IOW 4 + IRQ 1 + SIRQ 1 = 420

  PID   TID PR CPU% S     VSS     RSS PCY UID      Thread          Proc
 2084  2084  0   8% D      0K      0K     root     irq/341-synapti
15126 15126  1   5% R   1736K    980K     root     top             top
 1384  1402  0   1% S   6612K    872K     root     mpdecision      /system/bin/mpdecision
  865   974  0   1% S 579796K  55336K  fg system   UEventObserver  system_server
  865   960  0   0% S 579796K  55336K  fg system   ActivityManager system_server
  140   140  0   0% S      0K      0K     root     kworker/0:3
 1029  1029  0   0% S 491776K  48880K  fg u0_a60   ndroid.systemui com.android.systemui
    3     3  0   0% S      0K      0K     root     ksoftirqd/0
  395   677  0   0% S  69824K  10620K  fg system   EventThread
/system/bin/surfaceflinger
 2195  2195  0   0% S      0K      0K     root     kworker/0:0


User 0%, System 24%, IOW 0%, IRQ 1%
User 0 + Nice 0 + Sys 83 + Idle 250 + IOW 0 + IRQ 5 + SIRQ 0 = 338

  PID   TID PR CPU% S     VSS     RSS PCY UID      Thread          Proc
 2084  2084  0  22% D      0K      0K     root     irq/341-synapti
15126 15126  0   6% R   1744K    992K     root     top             top
 1384  1402  0   3% S   6612K    872K     root     mpdecision      /system/bin/mpdecision
    3     3  0   0% S      0K      0K     root     ksoftirqd/0
  140   140  0   0% S      0K      0K     root     kworker/0:3
   19    19  0   0% S      0K      0K     root     kworker/0:1H
  126   126  0   0% S      0K      0K     root     mmcqd/0
  865   974  0   0% S 579796K  55336K  fg system   UEventObserver  system_server
15094 15094  0   0% S      0K      0K     root     kworker/u:2
  413   413  0   0% S   1180K    624K     system   qrngd           /system/bin/qrngd
```

Figure 3-3 Top statistics

3.3.1.3 Ftrace/Pytime chart

Ftrace is a kernel function tracer that helps debug kernel function calls. The Pytime chart is a tool to visualize the collected Ftrace logs. PowerTOP details interrupt information, but do not provide the cause of the interrupt at a more granular level, in terms of what caused the interrupt. Ftrace allows further debugging of the cause of the interrupts.

Ftrace analysis is done in a graphical format, as shown in Figure 3-4 and Figure 3-5. The analysis shown is for a static display use case example, and is the same way this tool is used for camera-related use cases.

Figure 3-4 shows an Ftrace analysis for a current increase caused by KGSL interrupts and the mdss_fb_commit_wq_handler thread.

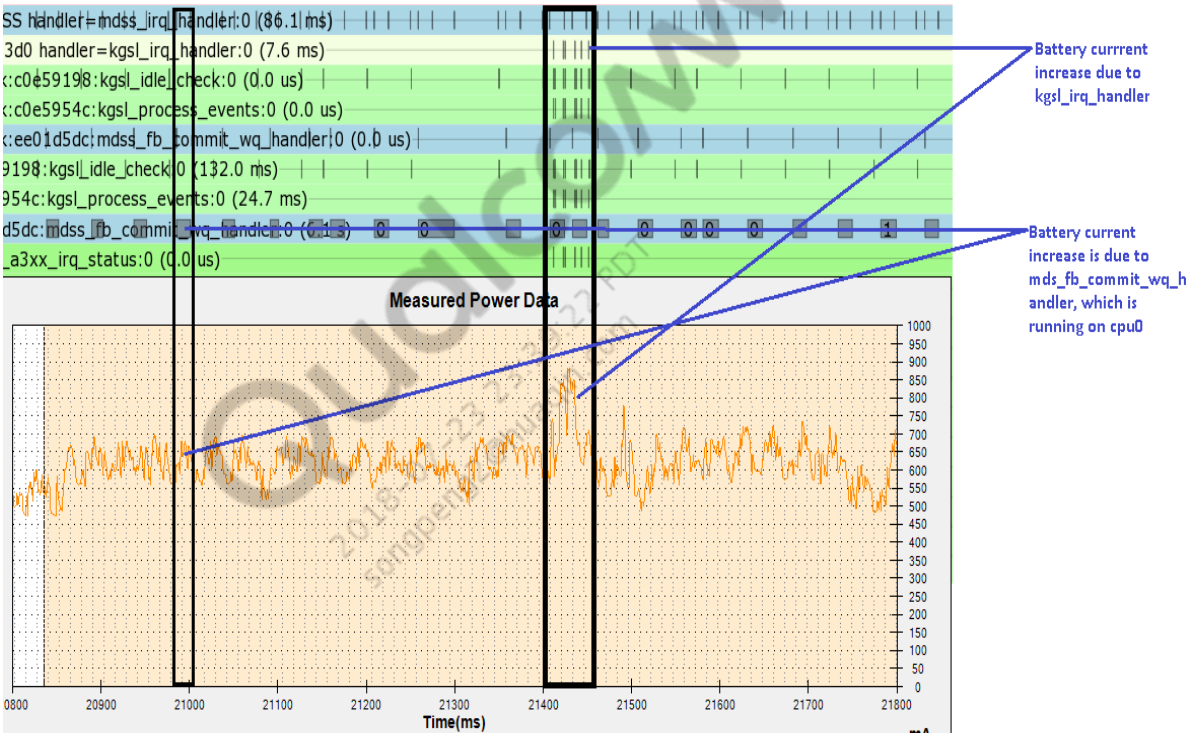


Figure 3-4 Ftrace log analysis – Static display use case with high current caused by KGSL interrupt

Figure 3-5 shows an Ftrace analysis of a static display use case with a current increase. This occurs when the device does not enter the Idle power collapse because of a touch driver interrupt.

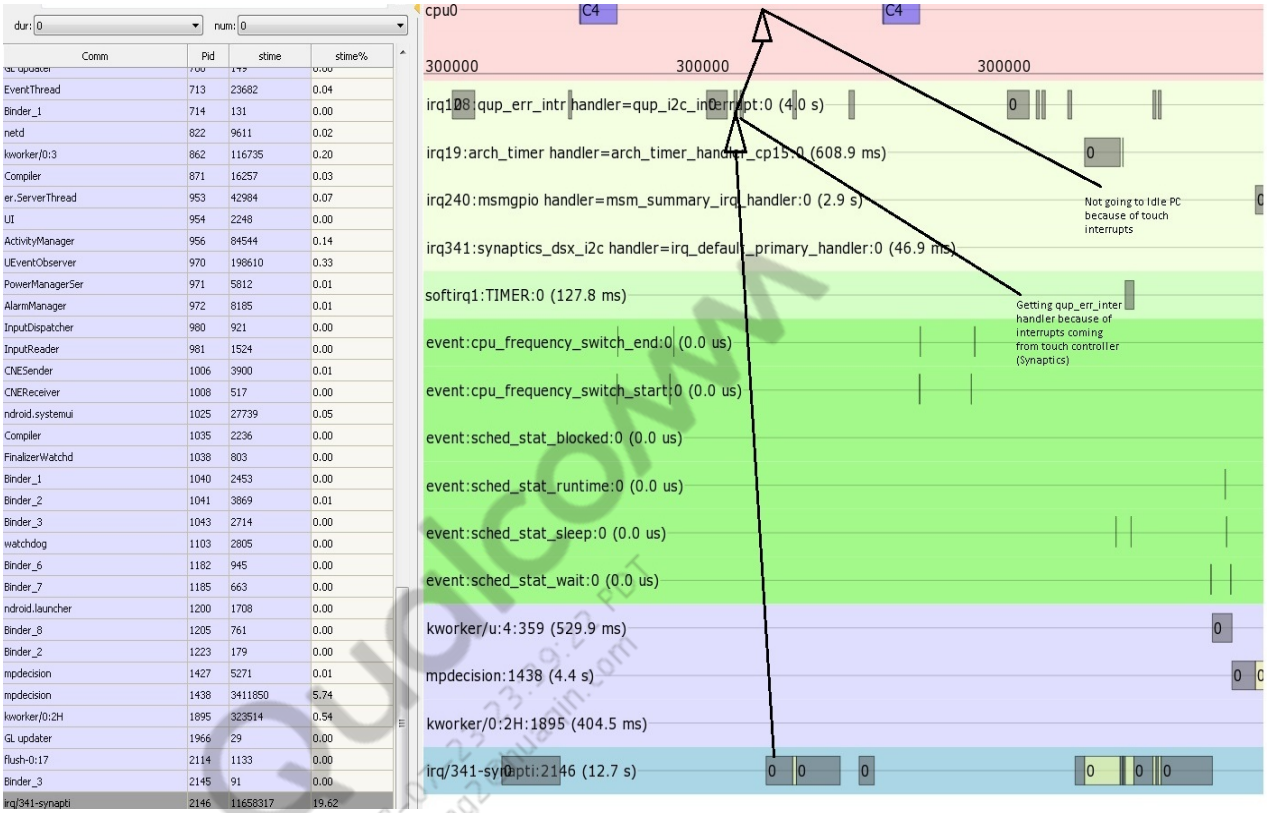


Figure 3-5 Ftrace logs analysis – Static display use case with high current caused by synaptic touch interrupt

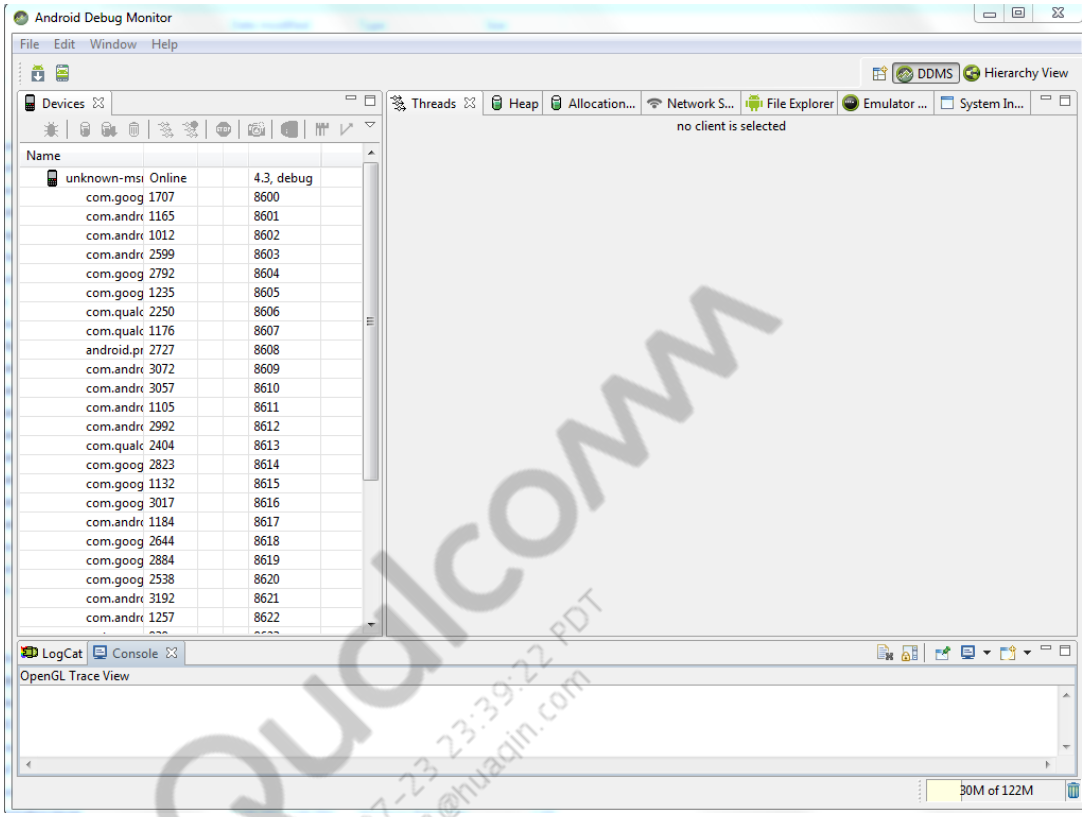
3.3.1.4 Systrace

The Systrace tool helps analyze the performance of an application by capturing and displaying the execution times of the application and other Android system processes. Systrace helps debug current increase issues related to the application or launcher. Systrace provides function-level detail, that is, causes based on which the function call thread runs.

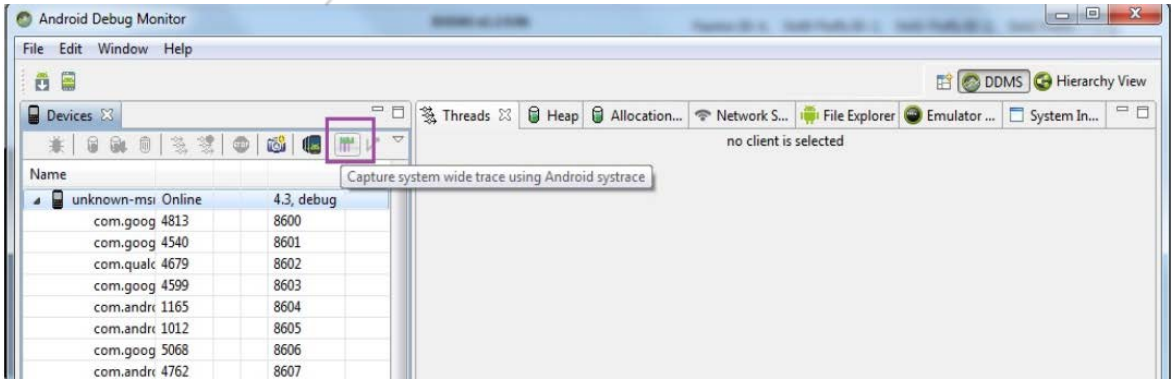
To run Systrace after prerequisite installation is done:

1. Power on the device and make sure it is rooted. If the device is not rooted, enter the ADB root command in the command prompt. The USB must be connected for Systrace capturing.
2. Go to adt-bundle-windows-x86_64-20130729\sdk\tools.

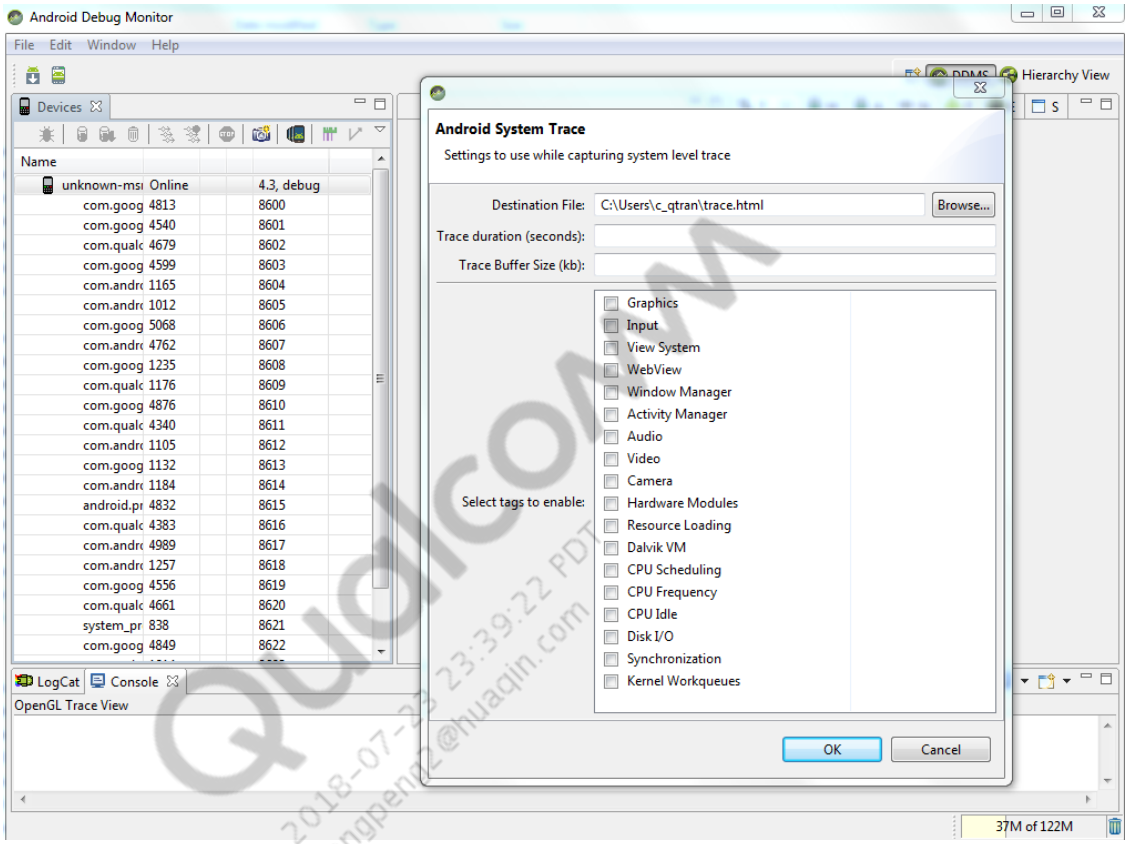
3. Click **Monitor.** The following window appears.



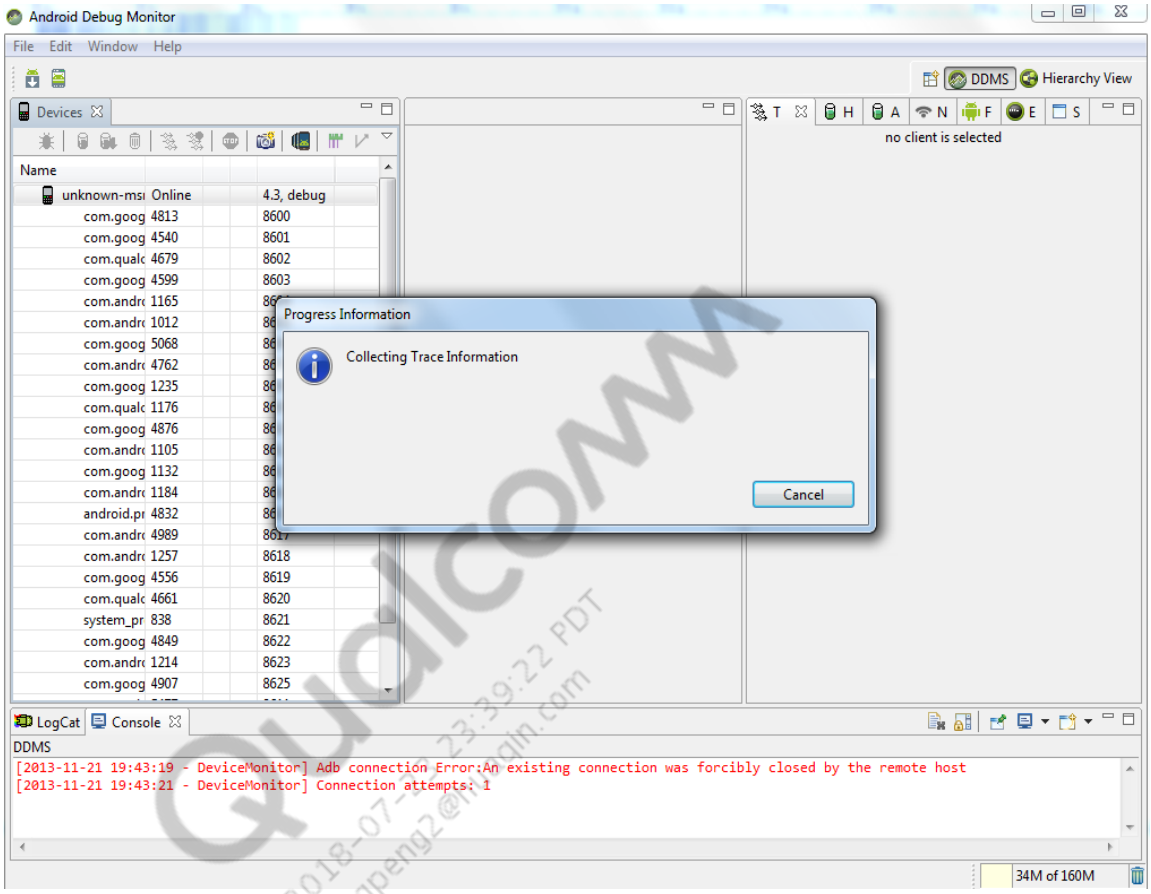
4. Click the highlighted button shown in the following figure.



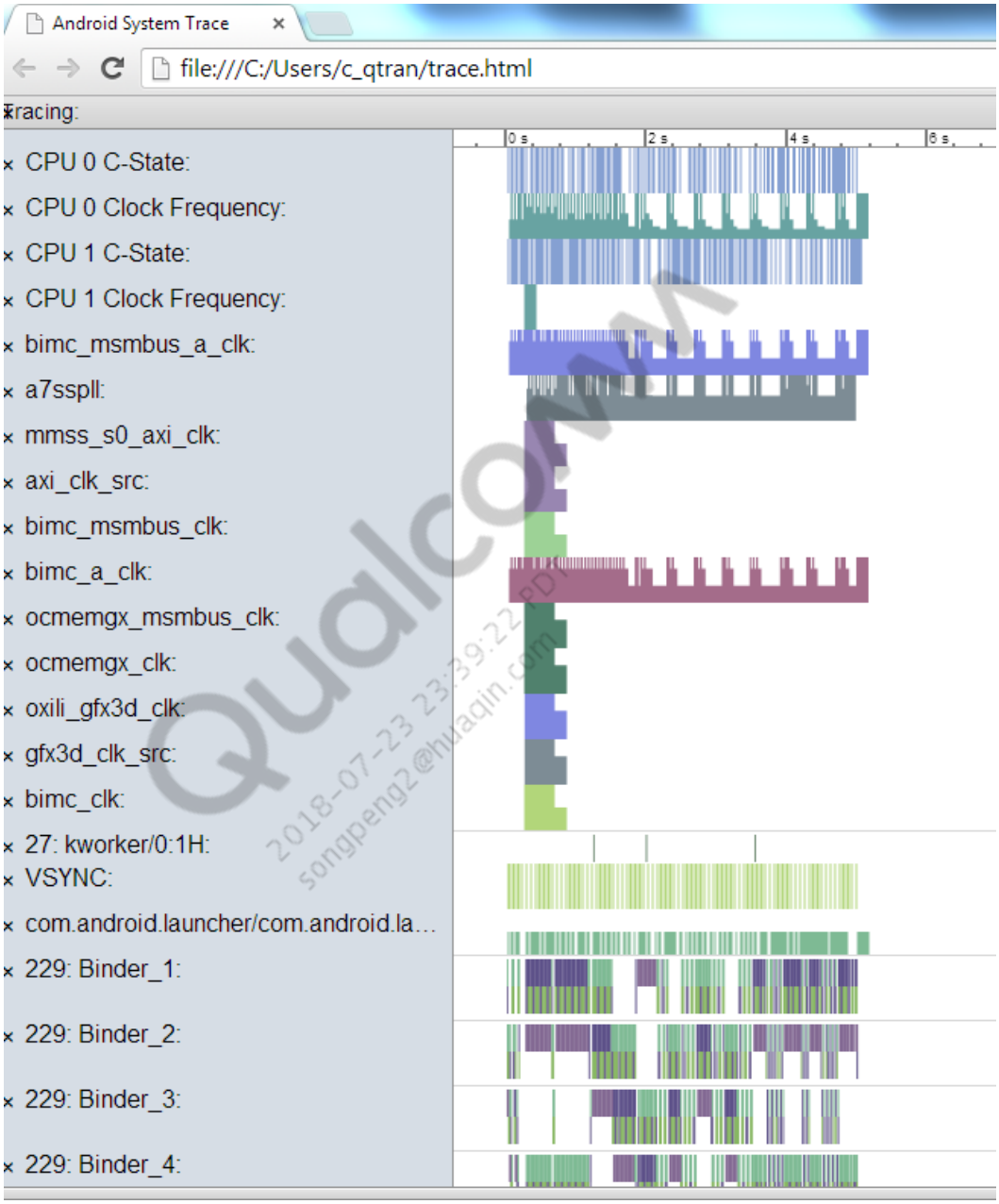
5. Select the area of interest, specify the destination location where data will be saved, and/or specify the trace duration in seconds; the default trace duration is 5 sec. When done, click **OK**.



6. On clicking OK, a pop-up window displaying progress information appears.



7. After data has been captured, open the trace.html file in Google Chrome to display the output.



8. Capture the same data on a reference device and compare for differences.

3.3.1.5 Governor parameter check

Check the interactive governor parameters, if the CPU frequency is not compatible with the QTI reference device. Modification in any of the parameters results in high current consumption.

The parameters located at /sys/devices/system/cpu/cpufreq/interactive are:

- above_hispeed_delay
- boost
- boostpulse
- boostpulse_duration
- go_hispeed_load
- hispeed_freq
- io_is_busy
- min_sample_time
- sampling_down_factor
- sync_freq
- target_loads
- timer_rate
- timer_slack
- up_threshold_any_cpu_freq
- up_threshold_any_cpu_load

The parameters located at /sys/devices/system/cpu/cpu0/cpufreq/ are:

- effected_cpus
- cpu_utilization
- cpuinfo_cur_freq
- cpuinfo_max_freq
- cpuinfo_min_freq
- cpuinfo_transition_latency
- related_cpus
- scaling_available_frequencies
- scaling_available_governors
- scaling_cur_freq
- scaling_driver
- scaling_governor
- scaling_max_freq
- scaling_min_freq
- scaling_setspeed

- stats
  - time_in_state
  - total_trans

The tuneable values for this governor are:

- target_loads – The CPU load value used to adjust speed to influence the current CPU load towards that value.
- min_sample_time – The minimum amount of time spent at the current frequency before ramping down.
- hispeed_freq – An intermediate high speed to initially ramp, when the CPU load hits the value specified in go_hispeed_load. If the load stays high for the amount of time specified in above_hispeed_delay, the speed may be increased.
- go_hispeed_load – The CPU load to ramp to hispeed_freq.
- above_hispeed_delay – When speed is at or above hispeed_freq, users need to wait for this amount of time before raising the speed in response to a continued high load.
- timer_rate – The sample rate to re-evaluate the CPU load when the CPU is not idle.
- timer_slack – The maximum additional time to defer handling the governor sampling timer beyond timer_rate, when running at speeds above the minimum.
- boost – If nonzero, immediately boost the speed of all CPUs to at least hispeed_freq, until 0 is written to this attribute. If 0, allow CPU speeds to drop below hispeed_freq according to the load as usual.
- boostpulse – On each write, immediately boost the speed of all CPUs to hispeed_freq for at least the period of time specified by boostpulse_duration, after which speeds are allowed to drop below hispeed_freq according to the load as usual.
- boostpulse_duration – The duration to hold the CPU speed at hispeed_freq on a write to boostpulse before allowing the speed to drop according to the load as usual.

3.3.1.6 sync_threshold comparison

Compare the sync_threshold of the device with the QTI reference device. This is checked by running the following command:

```
cat /sys/module/cpu_boost/parameters/sync_threshold
```

sync_threshold determines the frequency of the destination core, when a thread migrates from a source core to the destination core. If the source core frequency is higher than the sync_threshold, the destination core frequency ramps up to the sync_threshold frequency. If the source core frequency is lower than the sync_threshold, the destination core frequency matches the source core frequency. Configuring sync_threshold to a higher value results in better performance and higher power consumption.

3.3.2 Clock-level analysis

Clock dump debugging provides the entire list of clocks in the system with their respective frequencies. The clock dump information is used to ensure that device clocks are aligned to the reference target. When clocks are high, it results in extra use of PLLs, or pushes CX into higher

power modes, that is, Nominal, Turbo, and so on, than expected. The respective PLLs and CX voting by the clocks are found in the respective chip clock plan documents. For camera-related use cases, major clocks to refer are VFE, Video Core, MMSSNoC, BIMC, and CPU. However, other clocks must not be overlooked.

It is recommended to capture at least three different instances of clock dumps from RPM using JTAG. While comparing the clock dumps of two different devices, ensure that the CPU clocks are aligned to two out of three instances captured. If there are no matching instances, the closest CPU clocks are preferred.

For the expected clock plan for dashboard use cases, see the respective chipset clock plan document. Table 3-1 is an example of clock plan mapping based on MSM8974.

Table 3-1 MSM8974 clock mapping

Clock	
System Network on Chip (NoC)	gcc_sys_noc_axi_clk
Multimedia Subsystem (MMSS) NoC	mmss_mmssnoc_axi_clk
Configuration NoC	gcc_cfg_noc_ahb_clk
Peripheral NoC	gcc_periph_noc_ahb_clk
On-Chip Memory (OCMEM) NoC	ocmemnoc_clk
MMSS AHB clock	mmss_mmssnoc_ahb_clk
Bus Integrated Memory Controller (BMIC)	gcc_bimc_clk
Mobile Display Processor (MDP)	mdss_mdp_clk
Video core	venus0_vcodec0_clk
Graphics Processing Unit (GPU)	oxili_gfx3d_clk
Camera Post Processing (CPP)	camss_vfe_cpp_clk
Video Frontend (VFE)	camss_vfe_vfe0_clk

3.3.3 MSM bus request debugging

The MSM bus debug information is collected using the ADB interface. Capture the bandwidth request from each client and compare it with reference device data.

To capture bandwidth requests from all clients, run:

```
adb root
adb remount
adb shell
cd /d/msm-bus-dbg/client-data
cat *
```

To capture a bandwidth request from a specific client, the client name is provided instead of *, that is, cat * becomes cat qcom,enc-ddr-ab-ib, in order to capture the bandwidth request from the VFE.

```
adb shell
cd d/msm-bus-dbg/client-data
ls
Acpuclock
grp3d
mdss_mdp
mdss_pp
msm-rng-noc
pil-venus
qcom,dec-ddr-ab-ib
qcom,enc-ddr-ab-ib
qseecom-noc
 scm_pas
 sdhc1
 sdhc2
 update-request
 usb
```

3.3.4 Rail-level comparison

The power grid/rail-level breakdown is found in the respective chipset power application note. Compare the major rails, such as VDD MX1, VDD CX1, CPU, DDR, and GPU.

The rail-level power differences from the QTI reference device is caused by extra hardware components, extra PLLs used, or CPU running with high frequency. High power numbers are observed on digital rails CX/MX/LDOs due to PLLs, since they are running in Turbo mode. The digital rails show a high power difference due to other cores, such as MDP, VFE, Venus and so on, running with higher frequency.

3.3.4.1 Camera features that affect CPU, digital core (CX), and MX power for camera use cases

Configuring the camera features in a different way will affect current consumption.

4 Camera power consumption optimization

The power consumption on the QTI Linux camera solution can be optimized by using the suggestions provided for non-ZSL camera preview, non-ZSL camera snapshot, and camcorder use cases. Along with different knobs to optimize the overall power consumption, a set property for each of the items is provided. Set the properties after the device boot up and before starting the camera application. The persist properties remain set across reboots unless they are reset or the device is loaded with a new build.

4.1 Use MDP for composition

When a camera application uses SurfaceTexture, the preview frames go through the GPU for YUV to ARGB conversion. The UI layer and preview frames blending along with the GPU consume more power than the MDP. Hence, OEMs are recommended to use MDP by using SurfaceView instead of SurfaceTexture. The QTI Snapdragon™ camera application uses SurfaceView by default and the following recommendations are based on the same.

To convert camera apps to use SurfaceView:

1. Replace TextureView with SurfaceView.
2. Replace SurfaceTexture with SurfaceHolder.
3. Modify TextureView.SurfaceTextureListener to SurfaceHolder.Callback.
4. Modify TextureView.setSurfaceTextureListener(TextureView.SurfaceTextureListener) to surfaceHolder = surfaceView.getHolder(); surfaceHolder.addCallback(SurfaceHolder.Callback).
5. Replace the setPreviewTexture(SurfaceTexture) of the camera object with setPreviewDisplay(SurfaceHolder).
6. For proper scaling to take effect, set the SurfaceView within a type of layout, which should not be a part of the root <merge> tag.
7. If the preview dimension does not exactly match the screen, in the SurfaceView path, manually rescale the preview with the correct aspect ratio when the surface changes as follows:

```
mSurfaceView.getLayoutParams().width = (int) correctWidth;
mSurfaceView.getLayoutParams().height = (int) correctHeight;
mSurfaceView.requestLayout();
```

NOTE: Some devices require unsetting the hardware acceleration flag within the <application> tag of AndroidManifest.xml:

```
<application android:hardwareAccelerated=”false” …>
```

For further information, see:

- Java Code Examples for android.view.SurfaceView – *http://www.programcreek.com/java-api-examples/index.php?api=android.view.SurfaceView*
- TextureView – *http://developer.android.com/reference/android/view/TextureView.html*
- Hardware Acceleration – *http://developer.android.com/guide/topics/graphics/hardware-accel.html*

4.2 Set the Sensor mode resolution to video resolution

Set the Sensor mode resolution to the same as video recording resolution instead of Full Resolution, to reduce the power consumption in the camcorder recording use case. For example, set a 13 MP Sensor mode to 1080p for 1080p video camcorder recording.

Similarly, set the Sensor mode to display size or lower size to reduce the power consumption in a non-ZSL preview use case.

It is recommended that the Sensor mode is set to a lower resolution in camcorder and camera preview use cases, if the device feature set permits it, or if tuning is available for the respective resolution size. Users can make changes in their respective sensor driver.

4.3 Reduce camera preview size

If the Sensor mode is set to higher resolution, reducing the preview size to an optimal size reduces the traffic from Image Signal Processor (ISP) to DDR bus, and from DDR bus to MDP. This reduces the power consumption for camera preview use cases.

To reduce camera preview size:

1. Use setprop “persist.camera.preview.size <value>”.
2. Use the different size options available in the QTI Snapdragon camera application:
   - Value: 0 – Default size as per snapshot aspect ratio
   - Value: 1 – 640 x 480
   - Value: 2 – 720 x 480
   - Value: 3 – 1280 x 720
   - Value: 4 – 1920 x 1080

Example:

```
adb shell setprop persist.camera.preview.size 1 // this will set the
preview to 640x480
```

To reset to default resolution:

```
adb shell setprop persist.camera.preview.size 0
```

4.4 Disable animation after snapshot capture

The current default QTI Snapdragon camera application shows a flying picture and thumbnail image on the top-right corner of display after each snapshot capture, which consumes both the CPU as well as GPU. Disabling this feature reduces power consumption in a snapshot use case.

```
"adb shell setprop persist.camera.capture.animate  0" // set it to 0 to
disable animation
```

To reset it:

```
"adb shell setprop persist.camera.capture.animate  1" // set it to 1 to
enable animation
```

4.5 Disable thumbnail encoding

Currently the default QTI camera stack encodes a thumbnail in addition to main image encode to create an ExIF file. This thumbnail encoding can be disabled to save power consumption. However, when a user opens a gallery, for the first-time thumbnail encoding will be done for a new image, and that may add a little delay. This helps in reducing power consumption in snapshot use cases.

```
"adb shell Setprop persist.camera.th.disable 1" // set it to 1 to disable
thumbnail encoding
```

To reset it:

```
"adb shell Setprop persist.camera.th.disable 0" // set it to 0 to enable
thumbnail encoding
```

4.6 Reduce camera preview FPS

The default QTI camera stack sets the camera preview FPS to 30. Reducing the FPS reduces the traffic from ISP to DDR and DDR to MDP, thus reducing power consumption in a non-ZSL camera preview use case. A reduction from 24 fps to 20 fps does not adversely impact the user experience.
To reduce the camera preview fps, set:

```
"adb shell Setprop persist.debug.set.fixedfps <FPS-value>"
```

4.7 Disable tintless

Disable tintless to reduce power consumption.
To disable tintless, set:

```
"adb setprop persist.camera.tintless=disable"
```

4.8 Single pass CPP processing in Camcorder use case

By default, the preview size and video size may be configured differently along with the color formats. If there is difference in the preview or video size, or in the color format, CPP would take two pass processing paths, which will result into more power consumption.

Therefore, OEMs must configure the same format and size. Video supports NV12 format, so OEM/application needs to set NV12. It is advisable to do it for common resolutions like 1080p and 720p. For smaller resolutions, it may not help saving much.

This is applicable for the camera applications which do not expect preview buffer callback.

By default, the camera preview format in:

- Video mode:

```
yuv420sp
```

- Android code:

```
mParameters.set("preview-format", "yuv420sp");
```

NOTE: Both the above strings are the standard Google defined parameters and values can be found at: http://qwiki.qualcomm.com/public/OpenGrokAccess

To make single pass CPP processing, OEMs need to change their videomodule.java for the required resolutions as the following:

```
mParameters.set("preview-format", "nv12-venus");
```

NOTE: nv12-venus string is a QC added parameter and can be found in: hardware/qcom/camera/QCamera2/HAL/QCameraParameters.cpp

OEM needs to set “persist.camera.cpp.duplication” to 1 to use this feature.

And the corresponding sample application implementation is available at: https://www.codeaurora.org/cgit/quic/la/platform/packages/apps/SnapdragonCamera/commit/?id=e14d86362c467fd428bf0c951fbf3e7f35810dfa

4.9 Other settings

To reduce power consumption, disable the following as per the QTI Snapdragon camera application:

- Face detection
- Touch AF/AEC
- Flash mode
- Focus mode
- Wavelet Denoise (WNR)
- ZSL mode

To disable the settings, set:

```
adb shell setprop persist.camera.qcom.misc.disable 1.
```

It is recommended to set the BL brightness level to 50 (default is 102) through the UI option.

5 Sample power improvements

Table 5-1 shows the improvements in power consumption using different knobs defined in Chapter 4.

Table 5-1 Power improvements

Knobs	Details	Power improvements	Side-effects
Reduce preview size	Reduces the non-ZSL preview power consumption	▪ 35 mA on MSM8939 MTP 13 MP sensor ▪ 13 MP non-ZSL preview with 720p preview size ▪ 20 mA on MSM8939 MTP 13 MP sensor ▪ 13 MP non-ZSL preview with VGA preview size	Low preview size
Disable animation after snapshot capture	Disabling flying animation and thumbnail at the top-right display corner after image capture reduces the snapshot use case power consumption	▪ 10 mA on MSM8939 MTP 13 MP Sensor ▪ 5 MP non-ZSL capture	Impacts user experience. Only a shutter sound indicates capture completion.
Disable thumbnail encoding for each snapshot	Disabling thumbnail encoding and store in gallery for captured image reduces snapshot use case power consumption	Not measured on MTP	Extra latency when viewing a picture in the gallery for the first time. The EXIF file does not have the thumbnail when captured, whereby CTS may fail.
Reduce camera preview FPS	Helps in non-ZSL preview use case power consumption	60 mA on MSM8939 MTP 13 MP sensor @15 fps (13 MP sensor and 1080p display)	User experience depending on set FPS
Lower shutter volume	Reduces snapshot power consumption	Not measured on MTP	User experience

Sample properties to set for camera use cases power improvements:

- camera.capture.animate=0
- camera.tn.disable=1
- camera.preview.size=2

6 Power improvement in ZSL preview use case

NOTE: This section was added to this document revision.

- To improve the power performance when the display size is 1080p:
  - Set the preview stream size to 720p (set the resolution as the next lower resolution compared to the display size with the same aspect ratio, provided that the same is supported).
  - Maintain the aspect ratio with the maximum snapshot size.
- If the display size is higher than 1080p (QXGA or WQHD):
  - Set the preview stream size to 1080p (set the resolution as the next lower resolution compared to the display size with the same aspect ratio, provided that the same is supported).
  - Maintain the aspect ratio with the maximum snapshot size.
- For the implementation of the same:
  - Customer camera application needs to query the supported display size and the supported preview sizes.
  - Based on the above description, appropriate action (setting up the resolution) needs to be taken keeping the aspect ratio preserved for lower resolutions.

NOTE: QTI Snapdragon camera application takes care of this for the low-tier chipsets.

A TextureView to SurfaceView

The Snapdragon camera application uses SurfaceView by default. The code samples below show how to use SurfaceView and TextureView.

A.1 To use SurfaceView

Replace:

- TextureView→SurfaceView
- SurfaceTexture→SurfaceHolder
- TextureView.SurfaceTextureListener→SurfaceHolder.Callback
- TextureView.setSurfaceTextureListener(TextureView.SurfaceTextureListener)→surfaceHolder = surfaceView.getHolder()
- surfaceHolder.addCallback(SurfaceHolder.Callback);
  mCamera.setPreviewTexture(SurfaceTexture)→mCamera.setPreviewDisplay(SurfaceHolder)

The SurfaceView must be within some form of layout (for example, <LinearLayout>, <RelativeLayout>, <FrameLayout>, and so on). This is needed for any scaling or resizing to be applied properly.

Instead of using TextureView.setTransform(), resizing or scaling must be done via:

```
mSurfaceView.getLayoutParams().width = (int) correctWidth;
mSurfaceView.getLayoutParams().height = (int) correctHeight;
mSurfaceView.requestLayout();
```

Setting android:hardwareAccelerated=”false” in the <application> tag of AndroidManifest.xml ensures that GPU is not triggered even SurfaceView is used.

A.2 To use TextureView

```
public class LiveCameraActivity extends Activity implements
TextureView.SurfaceTextureListener {
   private Camera mCamera;
   private TextureView mTextureView;

   protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);

     mTextureView = new TextureView(this);
     mTextureView.setSurfaceTextureListener(this);

     setContentView(mTextureView);
```

```
    }

    public void onSurfaceTextureAvailable(SurfaceTexture surface, int width,
int height) {
      mCamera = Camera.open();

      try {
        mCamera.setPreviewTexture(surface);
        mCamera.startPreview();
      } catch (IOException ioe) {
        // Something bad happened
      }
    }

    public void onSurfaceTextureSizeChanged(SurfaceTexture surface, int
width, int height) {
      // Ignored, Camera does all the work for us
    }

    public boolean onSurfaceTextureDestroyed(SurfaceTexture surface) {
      mCamera.stopPreview();
      mCamera.release();
      return true;
    }

    public void onSurfaceTextureUpdated(SurfaceTexture surface) {
      // Invoked every time there's a new Camera preview frame
    }
  }
```

B References

B.1 Related documents

Resources	
Android camera specs	*http://developer.android.com/reference/android/hardware/Camera.html#autoFocus%28android.hardware.Camera.AutoFocusCallback%29*
Android camera specs	*http://developer.android.com/reference/android/hardware/Camera.html#startFaceDetection%28%29*
Java Code Examples for android.view.SurfaceView	*http://www.programcreek.com/java-api-examples/index.php?api=android.view.SurfaceView*
TextureView	*http://developer.android.com/reference/android/view/TextureView.html*
Hardware Acceleration	*http://developer.android.com/guide/topics/graphics/hardware-accel.html*

B.2 Acronyms and terms

Term	Definition
MDP	Mobile Display Processor
GPU	Graphics Processing Unit
ISP	Image Signal Processor
NoC	Network on Chip
MMSS	Multimedia Subsystem
OCMEM	On-Chip Memory
BMIC	Bus Integrated Memory Controller
CPP	Camera Post Processor
VFE	Video Front End