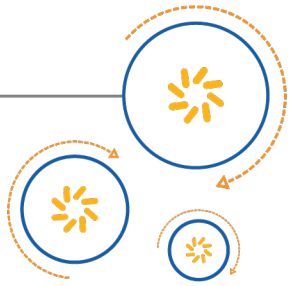
Qualcomm Technologies, Inc.



Display Postprocessing Service

User Guide

80-P2271-1 Rev. C

April 4, 2017

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

Revision history

Revision	Date	Description
A	September 2015	Initial release
B	January 2017	Updated to conform to QTI standards; no technical content was changed in this document revision
C	April 2017	Additional updates to conform to QTI standards; no technical content was changed in this document revision

Contents

1 Introduction

Qualcomm's Display Postprocessing Service (DPPS) refers to a tunable software block that manages postprocessing features in the display hardware. By relying on the device's mobile display processor (MDP) and ambient light sensor, DPPS manipulates display hardware to optimize power saving features such as Sunlight Visibility Improvement (SVI), Content Adaptive Backlight (CABL), and Assertive Display (AD).

1.1 Purpose

This document provides OEMs with a high-level overview of DPPS and describes its enablement, tuning, and debugging. The reader should have a general knowledge of the following:

- MDP postprocessing hardware design and architecture
- Hardware capabilities including histogram analysis, packet accelerator (PA), and lookup table (LUT) generation
- Linux display driver implementation and operation

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>`.

Commands to be entered appear in a different font, for example, `copy a:*.* b:`.

Button and key names appear in bold font, for example, click **Save** or press **Enter**.

1.3 Technical assistance

For assistance or clarification on information in this document, submit a case to Qualcomm Technologies, Inc. (QTI) at https://createpoint.qti.qualcomm.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 DPPS overview

Device hardware manipulation managed by DPPS produces instructions that specify feature-switching based on switch parameters. These parameters – which may include the user brightness setting or the ambient LUT value – allow one feature to take over completely while the other is paused.

2.1 Dependencies

DPPS depends on the MDP hardware block and the ambient light sensor in the device. The panel type, which may be either LCD or OLED, must be taken into account since each panel type has specific dependencies. The reader should also remember that DPPS is intended for device display exclusively; not for external display, HDMI, WFD or other virtual display modules.

2.2 OEM product requirements

The average RAM size used by DPPS is about 3200kB. In total, DPPS uses about 5000kB of shared library memory. Optimal performance of DPPS requires, at minimum, the following hardware products and software products with their associated dependencies.

- An ambient light sensor (ALS) fully tuned and calibrated on the target, specifically for SVI and AD features. (Consult the feature enablement guides when choosing an ALS.)
- A minimum of 8-bit granularity or 256 backlight levels for CABL
- An interface for screen refresh is required in state screen use cases to allow DPPS to call for the algorithm to converge when needed

2.3 Software design and implementation considerations

A socket-based interface to DPPS allows clients to send messages to turn on/off power saving or sunlight visibility features. OEMs must therefore implement a service to turn on/off DPPS features.

It is important to note that Partial Update (PU) and DPPS features cannot run concurrently. When DPPS features are enabled, PU will be disabled.

Performance and latency variants

Turning on DPPS takes between 2 and 4 frames since DPPS must turn off PU.

The turning off latency varies from feature to feature. The varying time is due to the convergence time required by the algorithms. Features will be turned off with in 256 frames once a command is received from the client.

Power dashboard details

Power dashboard details are provided in feature enablement guides for each algorithm.

Verification requirements

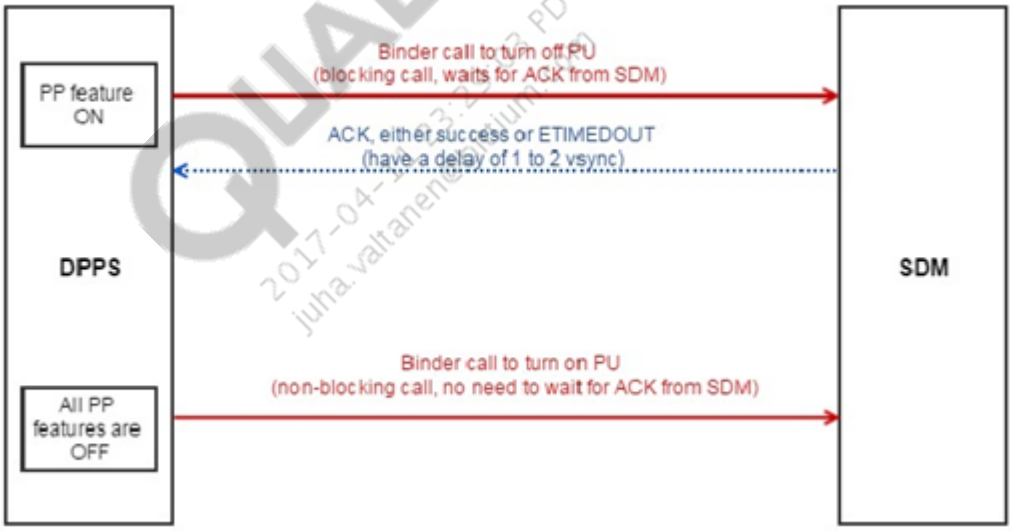
Verification requirements include:

- An LDC panel for CABL verification in both command and video modes
- An on-target light sensor for AD and SVI verification on LCD and OLED panels

2.4 PU design limitation

As noted, DPPS features are not compatible with the PU feature. PU would cause artifacts if enabled with CABL for example, because the LUT would only be applied on a partial image.

Assuming that DPPS and PU are available, one of them will always be enabled. Enabling a DPPS feature will disable default PU. Turning off all DPPS features will re-enable PU and these dynamics are shown below.



2.5 Licensing

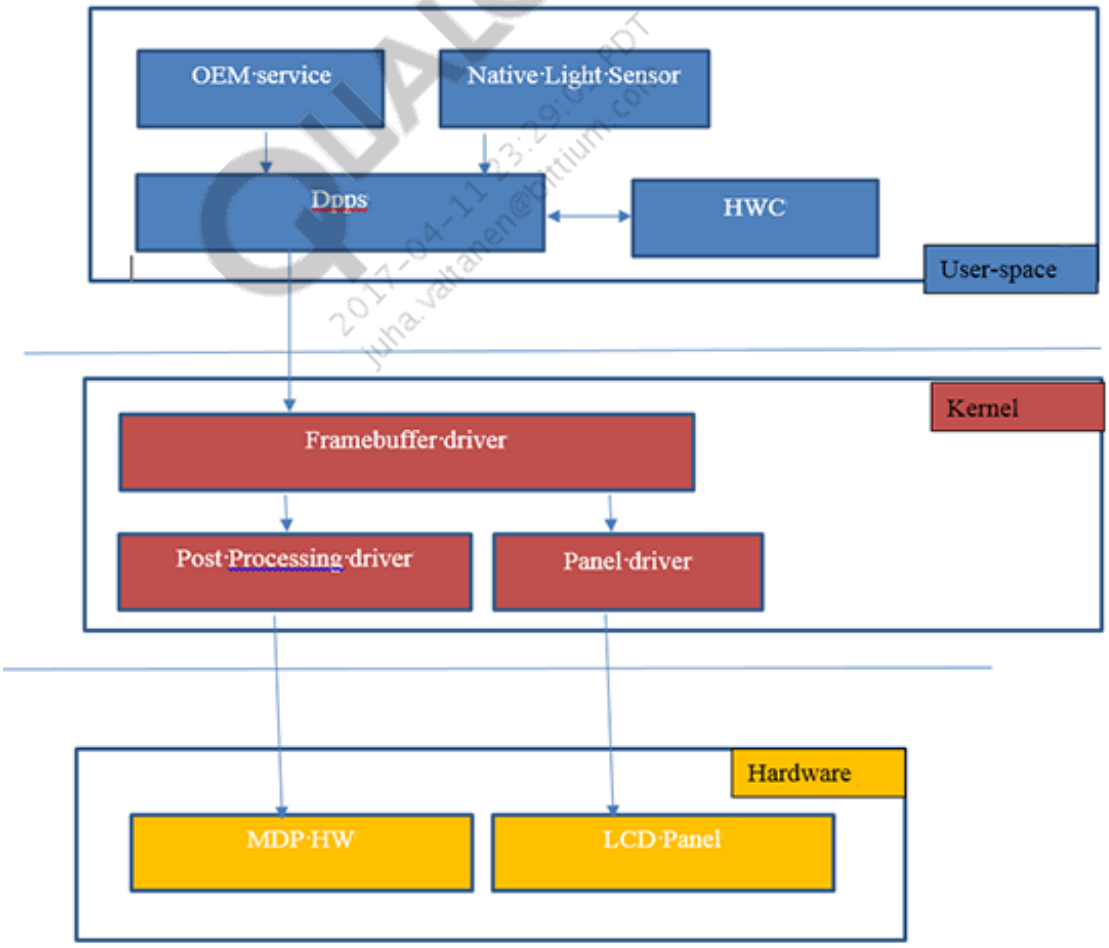
The appropriate licensing agreements are required for all DPPS features with the exception of CABL.

3 DPPS design

The general design of DPPS revolves around user-space, kernel, a hardware block, and software modules.

3.1 Software architecture

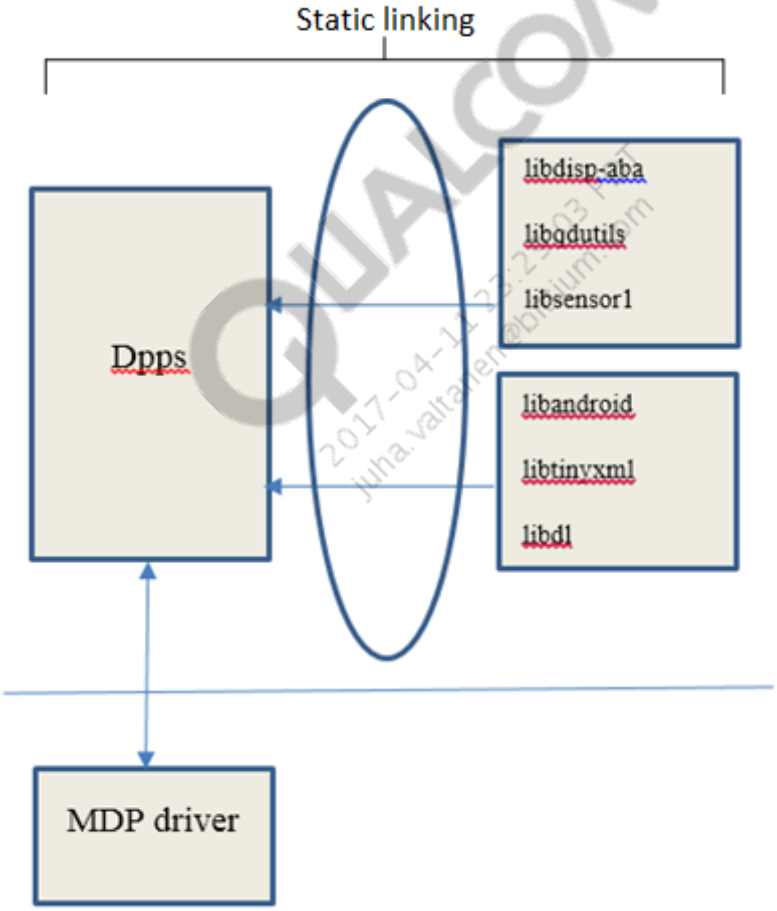
General DPPS software architecture – user-space, kernel, and hardware block – is represented in the figure below.



The following components are highlighted in the figure:

- OEM Service – Android Java service that sends enable commands to DPPS features once the device has completed its bootup sequence
- Native Light Sensor – An interface to light sensor hardware implemented through the Android native service, Qualcomm Sensor Core, or sensor simulation
- HWC – Hardware composer interface for forcing screen updates and turning on/off PU
- Framebuffer driver – Exposes the kernel interface exposed through the framebuffer ioctl
- Postprocessing driver – An extension of framebuffer driver, postprocessing ioctls come asynchronously and new configurations are cached. The new configurations are applied synchronously to the hardware during display commits through the framebuffer driver
- Panel driver – Panel backlight is driven through the backlight scale ioctl and framebuffer sysfs node

An interaction among these various software modules is represented in the following figure.

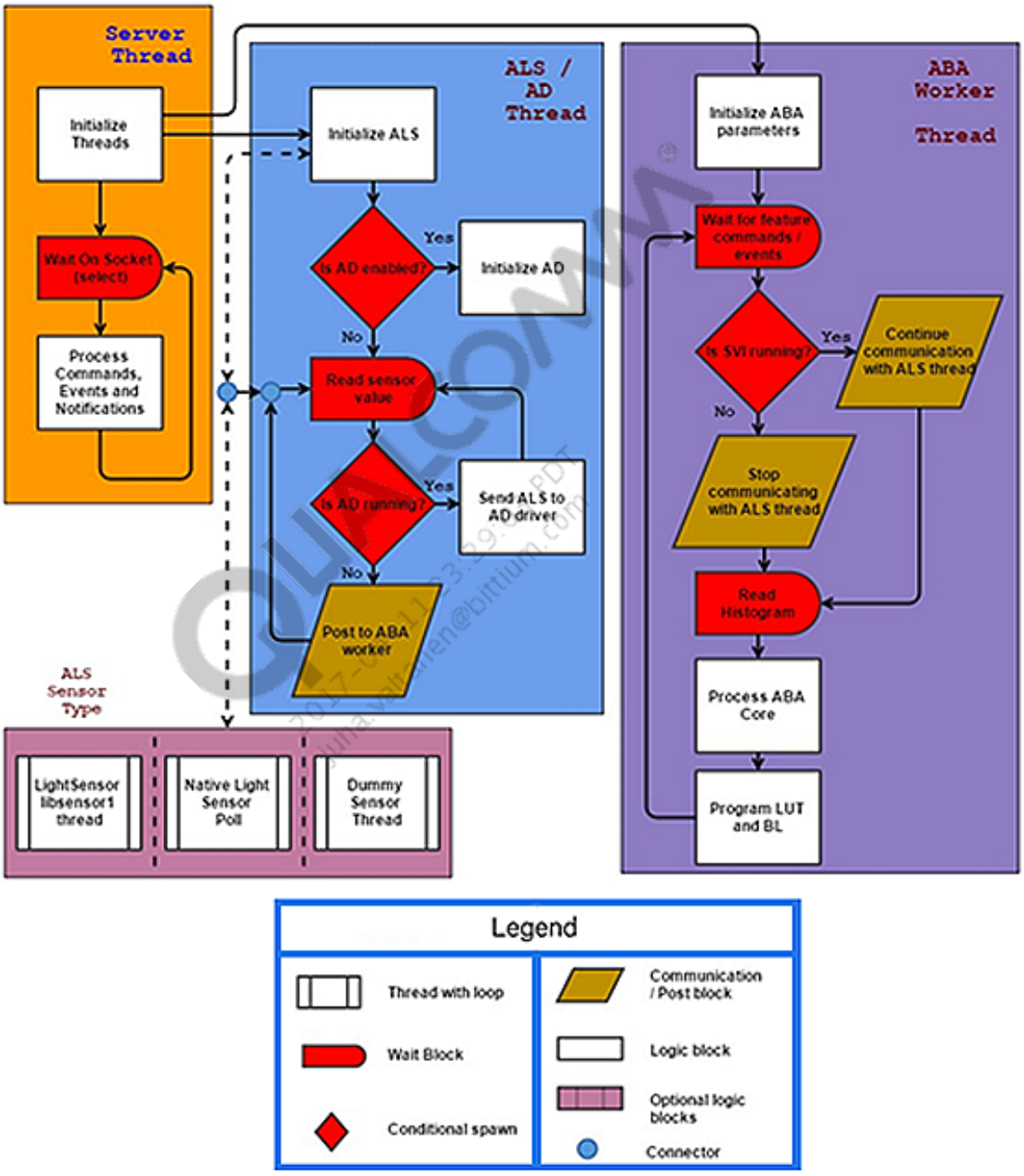


3.2 Memory analysis

The DPPS algorithm works on histogram analysis and LUT updates. Heap memory requirements are negligible.

3.3 Threading

Threading among DPPS software modules is represented in the following figure.



Three threads are shown, each displaying a specific interaction with another.

Server thread

The server thread initializes DPPS threads and resources. The server thread polls on client connections and handles incoming commands by passing commands to the ABA or ALS threads or by

replying directly when applicable. The server thread also handles shutdown events and resource cleanup for the DPPS process.

ABA thread

The ABA thread operates on a queue of incoming commands/events to be processed. The ABA thread is asleep when the queue is empty. The ABA thread polls on ABA events such as histogram ready, backlight event, and suspend/resume event and runs ABA loop iterations when the ABA algorithm has not converged. The ABA thread handles incoming ABA server commands such as on/off and set quality level, which are pushed to the ABA queue from the server thread.

ALS thread

The ALS thread operates on a queue of incoming commands/events to be processed. The ALS thread is asleep when the queue is empty. The ASL thread polls on ALS events such as new sensor events and suspend/resume and runs AD input iterations when new ambient light or backlight data is available. It handles incoming AD commands such as on/off, set assertiveness level, and AD calibration commands, which are pushed to the ALS queue from the server thread.

3.4 Tuning

All features may require tuning depending on the panel and ambient light sensor components used. Refer to the appropriate Design Guides for tuning information:

RELATED INFORMATION

"Related documents" on page 20

3.5 Concurrency

As the table of concurrent features shows, sunlight visibility and power saving content adaptive features can run concurrently. If PU is enabled, however, all active postprocessing algorithms must be disabled as multiple sunlight visibility features cannot be enabled concurrently.

AD	SVI	CABL	PU
✓	✗	✓	✗
✗	✓	✓	✗
✓	✗	✗	✗
✗	✓	✗	✗
✗	✗	✓	✗
✗	✗	✗	✓

3.6 Software directory structure/source code references

Code location is proprietary and no-ship, mm-pp-dpps will be provided to OEMs. Compile binaries are located in the out directory:

- DPPS binary – system/bin/mm-pp-dpps
- ABA library – system/vendor/lib64/libdisp-aba.so

4 Integration

Integration is a key process of successful DPPS deployment and a process the OEM should be particularly attentive to. The following section provides an overview of basic concerns as well as implementation instructions.

4.1 Integration steps

Integration is initiated with the following steps:

1. Add mm-pp-dpps as a service in init.target.rc

   The DPPS provides services in form of socket message based command and response mechanisms. At boot time, DPPS is started as service. The service declaration for DPPS is:

   ```
   service ppd /system/bin/mm-pp-dpps
       class late_start
       user system
       group system graphics
       socket pps stream 0660 system system
       disabled
   ```

   As already noted, any native or Java service must have system graphics permission to connect to the postprocessing daemon socket. A native or Java service trying to communicate with postprocessing daemon without the required permissions will yield a permissions error.

   If Android OEM applications need to communicate to the postprocessing daemon using the socket, permissions can be provided through the Android application manifest file:

   ```
   <manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.android.qualcomm"
      android:sharedUserId="android.uid.system">
   ```

   Notice that for system user privileges, the application must be built with the platform as user privileges cannot be provided to non-OEM Android applications.

2. Update the SE Linux policy file to assign the required permissions for socket programming, binder calls with HWC.

3. License files from Qualcomm to enable features.

4. Incorporate a service layer which enables/disables features.

   a. Set system build.prop properties

   Features are enabled by adding their respective build properties to the system build.prop. These properties will not turn the feature on at bootup, but the build property will allow the feature to be turned on. If the property is not there, the feature will be prevented.

   The following is an example of adding the CABL property to build.prop.

```
# Previous sytem property
bt.max.blah=1
# System property for cabl
ro.qualcomm.cabl=2
#
# System props for telephony
telephony.blah=1
```

a. Update the build.prop as shown below.

```
$ adb root
$ adb remount
$ adb pull /system/build.prop .
# Modify build.prop
$ adb push build.prop /system/build.prop
$ adb shell chmod 644 /system/build.prop
$ adb reboot
```

4.2 Reference code – OEM service layer

On Linux Android, features like CABL need an initial enable message when the device finishes booting in order to start the feature. An Android Java service can be used to send a socket message to the DPPS once the bootup sequence has completed. Below is example code showing how a socket message can be sent to the DPPS using Java.

```
int MAX_CMD_LEN = 64;
byte[] buf = new byte[MAX_CMD_LEN];
LocalSocket s = new LocalSocket();
s.connect(LocalSocketAddress("pps")); //attach to socket 'pps'
OutputStream output = s.getOutputStream();
/* write example command to enable CABL */
output.write("cabl:on");
output.flush();
InputStream input = s.getInputStream();
/* read for response from socket to determine if
 * command was successful */
input.read(buf, 0, MAX_CMD_LEN);
/* write example command to set CABL quality level to High */
output.write("cabl:set High");
output.flush();
input.read(buf, 0, MAX_CMD_LEN);
/* write the command to disable CABL */
output.write("cabl:off");
output.flush();
input.read(buf, 0, MAX_CMD_LEN);
```

5 Feature-level debug

Historically, DPPS issues depend on log capture in order to debug. For enable/disable and internal stability issues, logs will generally be the best tool for debugging. DPPS is a user space process and will therefore output debug logs to logcat. Logs are formatted to include function name and line number of the log location.

5.1 DPPS tag and log format

The logcat tag for DPPS is ‘Dpps’. The typical Dpps log format is:

```
<I/E>\Dpps: <Function name>():<Line number> Log
```

DPPS has the ability to dynamically change the log level of different features using socket-based messages. The log levels present for all features are

- ERROR
- INFO
- VERBOSE

When debugging DPPS issues, it might be necessary for OEMs to gather logs at a higher debug level that the default level of INFO. There are logs hidden underneath the level of VERBOSE that can be enabled through a DPPS socket message. The logs can also be enabled per feature.

5.2 Debug setup

Messages for changing the debug log level in the DPPS are shown in the table below.

Message	Description
`debug:aba;<error/info/verbose>`	Change log level of ABA based features.
`debug:al;<error/info/verbose>`	Change log level of ALS based features
`debug:aba;hist;<on/off>`	Turn on/off logging of ABA histograms

Message	Description
debug:aba;lut;<on/off>	Turn on/off logging of ABA LUTs
debug:dpps;<error/info/verbose>	Change log level of all DPPS logs

The following code may serve as an example of how to change the debug level using the ppd test app:

```
root@device:/ # ./data/display-tests/ppd 'debug:aba;verbose'
./data/display-tests/ppd 'debug:aba;verbose'
Testing for command: debug:aba;verbose
writing debug:aba;verbose to the socket
Daemon response: Success
```

5.3 DPPS stability log captures

Three log captures provide essential information about DPPS stability.

- For issues relating to ABA (CABL and SVI), capture the verbose logs in the ABA context.
- For issues relating to ALS-based features (SVI and AD), capture the verbose logs in the ambient light context.
- For issues relating to DPPS server communication, capture the verbose logs for all of DPPS.

5.4 DPPS quality log captures

For ABA quality issues (flickering/artifacts), capture the histogram and LUT logs.

5.5 Issue reporting template

Complete the following template when reporting issues to hasten the issue-solving process.

- Hardware:
  - Chipset – For example, MSM8974
  - Display panel information – LCD/OLED, command mode/video mode etc.
- Software release:
  - Metabuild
  - APPS – Apps AU or build path
- Criticality – TA/launch blocker, critical, high, etc.
- Issue description
- Steps to reproduce – Clear steps used to reproduce the issue. For certain features, such as AD, include all properties and calibration configuration files used to enable AD.
- Frequency of the issue – How often does the issue happen? Is it 100% reproducible or only a fraction of the attempts, etc.
- Reproducible on QTI platform?

- Answer affirmatively if attempted and the issue is reproducible. Mention the device used MTP/ CDP/ FLUID/LIQUID/QRD etc.
- Answer negatively if attempted and the issue is not reproducible.
- Answer ‘not attempted,’ if not attempted.

- Initial Analysis:
  - Describe the problem observed, for example high power consumption when AD is used with Android native sensor
  - Describe the expected behavior, for example power consumption is X ma when Qualcomm sensor configuration is used and switching to Android native sensor bumps the power up to Y ma
    Logs – Provide the logs relevant to the case/issue.

6 Verification

Verification occurs by means of a subset of logs that show the creation of the DPPS and associated worker threads.

6.1 DPPS bootup verification

When booting, the following subset of logs will typically appear showing successful creation of the DPPS and worker threads running ABA or AD.

```
I/Dpps  ( 3690): static DppsServer* DppsServer::GetInstance(): DppsServer ref
count increased to 1
I/Dpps  ( 3690): AbaContext():24 ABA context instantiation
I/Dpps  ( 3690): AmbientLightContext():18 AL context instantiation
I/Dpps  ( 3690): InitFeatureContext():171 Initialize feature context
I/Dpps  ( 3690): InitContext():136 Context Initialization successful
I/Dpps  ( 3690): CreateAbaThread():61 Create ABA thread
I/Dpps  ( 3690): CreateAbaThread():84 Create ABA thread done ret 0
I/Dpps  ( 3690): InitLightSensor():446 sensor_type_ 2
I/Dpps  ( 3690): InitLightSensor():467 light sensor initialization ret 0
I/Dpps  ( 3690): CreateAmbientLightThread():141 create AL thread
I/Dpps  ( 3690): CreateAmbientLightThread():160 Create AL thread done ret 0
I/Dpps  ( 3690): InitFeatureContext():222 Initialize feature context done ret
0
I/Dpps  ( 3690): ProcessAbaThread():253 ABA thread execution started
I/Dpps  ( 3690): ProcessAmbientLightThread():98 AL thread execution started
I/Dpps  ( 3690): AddNewDppsClient():784 New DPPS client accepted, client fd =
15, poll idx 2
I/Dpps  ( 3690): HandleDppsClientEvent():1125 Incoming message on client fd
15 : 'cabl:on'
I/Dpps  ( 3690): ProcessAbaCommands():170 command recieved kCablOnCmd
```

All of the DPPS logcat logs during bootup should be of type Info (I\Dpps) with the line numbers subject to change based on new patches in the code.

Here the context is successfully created for ABA CABL and ambient light and the initial message for enabling CABL is processed. Assuming the thread resources are created without issue, the ABA and ambient light (AD) features will be able to start executing without errors.

6.2 Unit test plan

The ppd executable is located in /data/display-tests/. The ppd test app is used to send messages over socket to the DPPS. Each time the app is executed, it

- Opens a connection to the DPPS socket

- Sends the message provided in the command line argument
- Checks the DPPS response
- Closes the socket connection

Use the DPPS test app to test all DPPS commands and verify DPPS functionality. The message being sent to the DPPS through the command line must be enclosed in single quotes as shown in the example ppd text app execution below.

```
root@device:/ # ./data/display-tests/ppd 'cabl:status'
./data/display-tests/ppd 'cabl:status'
Testing for command: cabl:status
writing cabl:status to the socket
Daemon response: running
root@device:/ # ./data/display-tests/ppd 'cabl:set Medium'
./data/display-tests/ppd 'cabl:set Medium'
Testing for command: cabl:set Medium
writing cabl:set Medium to the socket
Daemon response: Success
```

6.3 Stability test recommendation

Stability testing can be done by writing scripts that execute many iterations of the ppd test app, send many messages to the DPPS in succession, and verify that the DPPS continues to function correctly.

A Supported Commands

The OEM will find it useful to have a description of supported commands, including the DPPS Msg Enum and command arguments, where applicable. Understanding these commands ensures optimal functionality of the DPPS.

A.1 Identification of supported commands

The following table lists and identifies supported commands.

Commands	Argument	DPPS Msg Enum	Description
cabl:on	-	kCablOnCmd	Enables CABL
cabl:off	-	kCablOffCmd	Disables CABL
cabl:set	<Low/Medium/High/Auto>	kCablSetQualityCmd	Sets CABL quality level
cabl:status	-	kCablStatusCmd	Gets current CABL running/ stopped status
cabl:get	-	kCablGetQualityCmd	Gets current CABL quality level
cabl:scale	-	kCablGetScaleCmd	Gets current CABL backlight scale
cabl:support	-	kCablSupportCmd	Checks if CABL is supported
debug:aba	:<error/info/verbose>	kAbaDebugCmd	Sets ABA debug log level
debug:aba:hist	:<on/off>	kAbaDebugHistCmd	Shows ABA histogram log
debug:aba:lut	:<on/off>	kAbaDebugLutCmd	Shows ABA LUT log
debug:al	:<error/info/verbose>	kAlDebugCmd	Sets ambient light log level
debug:dpps	:<error/info/verbose>	kDppsDebugCmd	Sets DPPS (all) log level

NOTE AD commands are provided by Apical in the Assertive Display Enablement Guide.

B References

Related documents and and useful acronyms and terms are provided to OEMs to facilitate context, comprehension, and successful application of tools.

B.1 Related documents

Title	Number
Qualcomm Technologies, Inc.	
CABL OEM Design Guide	80-NP952-1
SVI OEM Design Guide	80-NP906-2

B.2 Acronyms and terms

Acronym or term	Definition
AD	Assertive Display
CABL	Content Adaptive Backlight
DPPS	Display Postprocessing Service
LUT	Lookup Table
MDP	Mobile Display Processor
PU	Partial Update
SVI	Sunlight Visibility Improvement