



Android Quick Boot

Feature Specifications

Issue **00B01**

Release Date **2015-03-04**

Copyright © HiSilicon Technologies Co., Ltd. 2015. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of HiSilicon Technologies Co., Ltd.

Trademarks and Permissions



HISILICON, and other HiSilicon icons are trademarks of HiSilicon Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between HiSilicon and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

HiSilicon Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: <http://www.hisilicon.com>

Email: support@hisilicon.com



About This Document

Purpose

This document describes the principle, functions, configuration method, key test points, and problem locating method of the Android quick boot feature.

This document provides guidance on how to use and test the feature.

Related Versions

The following table lists the product versions related to this document:

Product Name	Version
HiAndroidSTB	V600R100C00SPC060 and later



Intended Audience

This document is intended for:

- Technical support engineers
- Software engineers
- Test engineers

Symbol Conventions

The symbols that may be found in this document are defined as follows.

Symbol	Description
 CAUTION	Alerts you to a potentially hazardous situation that could, if not avoided, result in equipment damage, data loss, performance deterioration, or unanticipated results.
 NOTE	Provides additional information to emphasize or supplement important points in the main text.



Change History

Changes between document issues are cumulative. Therefore, the latest document issue contains all changes made in previous issues.

Issue 00B01 (2015-03-04)

This issue is the first draft release.



Contents

About This Document.....	ii
1 Feature Description.....	1
1.1 Solution Description.....	1
1.1.1 Definition	1
1.1.2 Benefits	3
1.1.3 Availability	3
1.1.4 Impact	3
1.2 Application Scenarios.....	3
2 Quick Boot Feature Description.....	5
2.1 Feature Modification Description	5
2.1.1 Added Partitions.....	5
2.1.2 Added System Services.....	5
2.1.3 Data Reloading.....	6
2.2 Customization Description	6
2.2.1 Fastplay Support	6
2.2.2 Tips for Image Creation	6
2.2.3 Production Mode.....	7
2.2.4 Upgrade and Default Settings Restoration.....	7
2.2.5 Switchover Between Quick and Regular Boot Modes.....	7
2.2.6 Image Creation Time Customization	8
2.2.7 Debugging Means	8
2.2.8 System Service Customization.....	10
2.2.9 Security of Quick Boot	11
2.2.10 Description of Quick Boot Application Upgrade.....	12
2.2.11 Descriptions of Status Attributes in the Quick Boot.....	12
3 Configuration Description	13
3.1 Configuring the Quick Boot Feature	13
3.1.1 Enabling and Disabling.....	13
4 Maintenance and Test Guidelines.....	15
4.1 Test Environment Requirement.....	15
4.2 Test Procedure	15



4.3 Positioning Method	15
4.4 Log Analysis.....	15



Figures

Figure 1-1 Process for creating the image for the quick boot.....	1
Figure 1-2 Process for starting up the Android OS in quick boot mode.....	2
Figure 1-3 Data reloading.....	3
Figure 2-1 Functions of the QbAndroidManagerService	6
Figure 2-2 Adding a service to the SystemServer before image creation (1)	10
Figure 2-3 Adding a service to the SystemServer before image creation (2)	11



Tables

Table 2-1 Descriptions of status attributes in the quick boot feature	12
--	----



1 Feature Description

1.1 Solution Description

After the Android operating system (OS) is normally started up for the first time, the quick boot feature records and compresses the Android OS's memory status to an image and stores it in a preset partition. In the next startup, the Android OS decompresses the image and restores the memory status without going through the whole startup process, to achieve a quick boot.

1.1.1 Definition

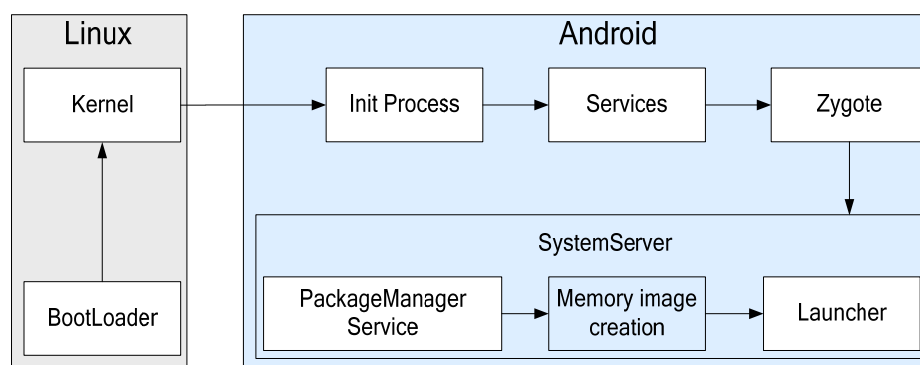
Image Creation

To enable the quick boot feature, store the Android OS's memory status to an image.

After the source code of the quick boot macro is opened, compiled, and put onto the board, the image for the quick boot is created when the Android OS is started up for the first time. If the image is successfully created, the Android OS can be properly started up until the launcher is accessed.

Figure 1-1 shows the process for creating the image for the quick boot.

Figure 1-1 Process for creating the image for the quick boot



During the preceding process, the memory status of the Android OS is recorded and stored after it is normally started up for the first time.



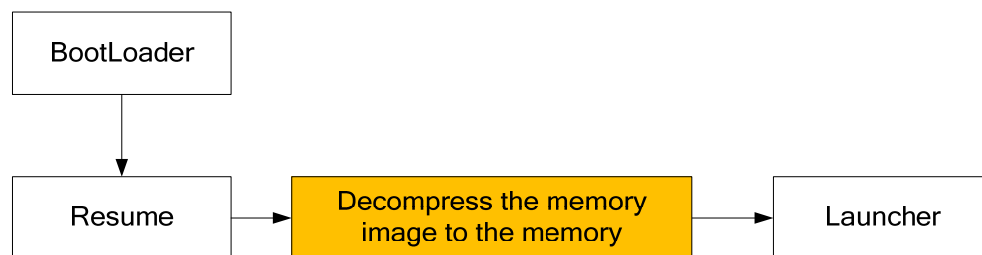
Startup of the Quick Boot Mode

After the Android OS completes the image creation process, the quick boot mode is used when the Android OS is started up later. The process for starting up the Android OS in quick boot mode is as follows:

- Step 1** After the boot loader program is executed, the Android OS determines whether the quick boot mode is used based on qbflag partition information.
- If the quick boot mode is used, decompress the image in the qbdata partition to the memory to restore the memory image.
 - If the quick boot mode is not used, start up the system based on the regular process.
- Step 2** The Android OS proceeds with the follow-up procedures after image restoration until the launcher is initiated, to complete the whole startup process.
- End

Figure 1-2 shows the process for starting up the Android OS in quick boot mode.

Figure 1-2 Process for starting up the Android OS in quick boot mode



Data Reloading

The Android OS restores the original memory status at the image creation time every time due to the restrictions of the quick boot feature. For this reason, although the user uses services established in the image, the Android OS still restores the memory at the image creation time in the next boot, leading to a scenario in which the user's data cannot be updated. In this case, the latest service data must be reloaded.



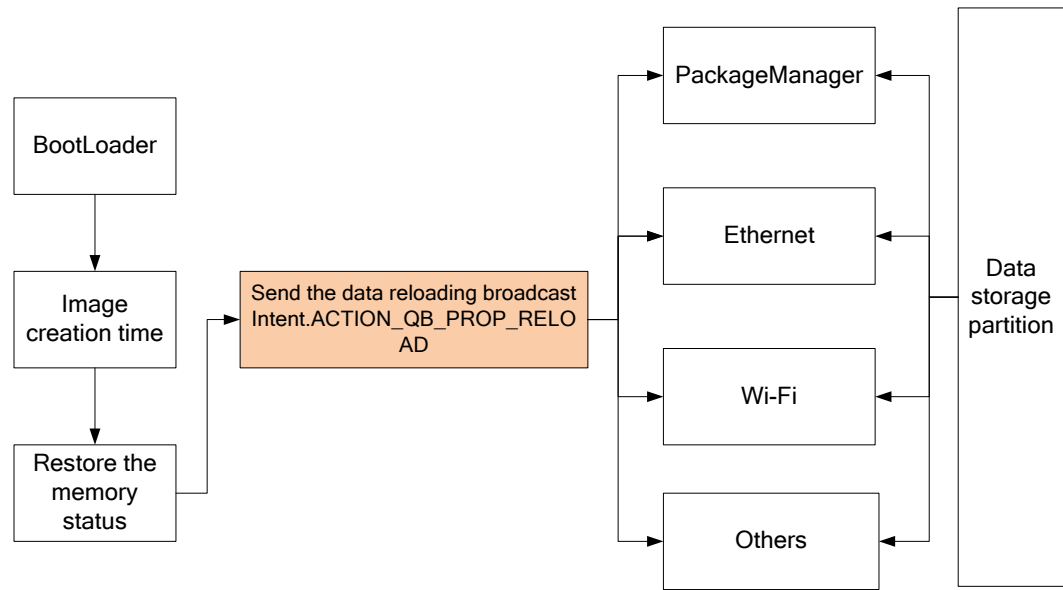
CAUTION

Data must be reloaded for services used before image creation. For services used after image creation, the data reloading process is not required.

Figure 1-3 shows the process for reloading service data in quick boot mode.



Figure 1-3 Data reloading



1.1.2 Benefits

Using this quick boot feature, the time required to start up the Android OS is significantly shortened, achieving quick human-computer interaction for customers and improving user experience.

1.1.3 Availability

This feature is supported by the Hi3798M V100 and Hi3796M V100 chips in HiAndroidSTBV600R100C00SPC060 and later versions.

1.1.4 Impact

As the feature is implemented based on the mirroring technology, an image must be created for each board when the Android OS is properly started up, to enter the quick boot mode.

1.2 Application Scenarios

When the customer needs to accelerate the speed for starting up the set-top-box (STB)-based Android OS, open the configuration file and change the corresponding attribute value to compile and program the quick boot source code.



For HiSilicon's HiAndroidSTBV600R100C00SPC060, it takes 20s to start up the Android OS on the demo board of Hi3798M V100. After the quick boot mode is applied, the default launcher is used, and the startup duration can be shortened to 10s. The startup duration stays around 8s in quick boot mode on the current version.



CAUTION

Under actual application environment, the read-write speed of the embedded multi media card (EMMC) on the board and customized launcher may affect the speed of the quick boot. Currently, only the EMMC supports the quick boot feature.



2 Quick Boot Feature Description

2.1 Feature Modification Description

2.1.1 Added Partitions

The following partitions are added as required by the quick boot feature:

- **hibdrv**: a partition for storing the **hibdrv.bin** file, which is used to create and decompress the image.
- **userapi**: a partition for supporting either of the following functions based on different storage devices, such as different kinds of EMMCs:
 - Function of writing the compressed quick boot image to the partition for storing the image
 - Function of reading the created image information
- **qbflag**: It is the flag for the quick boot. The Android OS reads the information in this partition when it is started up, to determine whether the quick boot mode should be used:
 - If the quick boot mode is used, read and decompress the created image from the storage device.
 - Otherwise, start up the Android OS based on the regular process.
- **qbdata**: a partition for storing the memory image of the quick boot. In quick boot mode, the Android OS reads the image from this partition.

2.1.2 Added System Services

As required by the quick boot feature, a key service is added to the SystemServer:

`QbAndroidManagerService`

The specific source code of the service is stored in the following directory:

`device/HiSilicon/bigfish/frameworks/qb`

Start up various services in sequence in the **SystemServer.java** file, and the following codes are added:

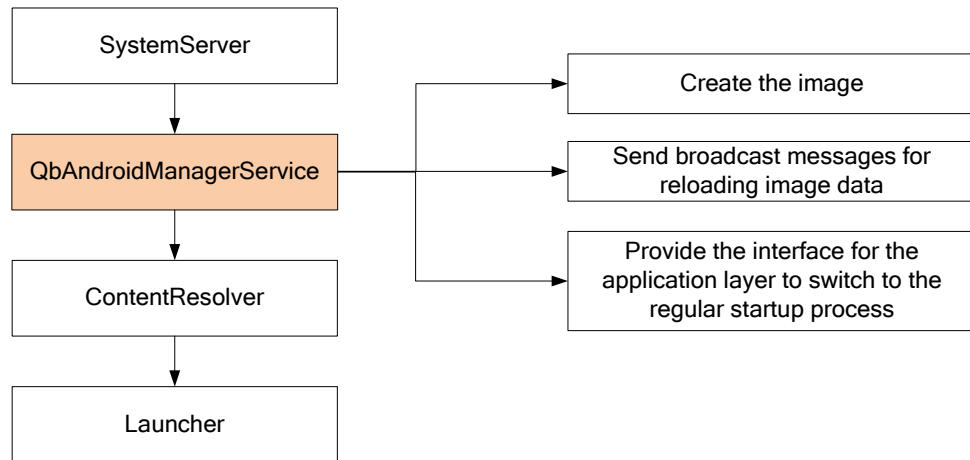
```
startQbService(context)
```

This function is mainly used to start up the **QbAndroidManagerService**, create images, send broadcast messages for system data reloading, and notify each module to reload data, based on the current property status of the system.



Figure 2-1 shows the functions of the **QbAndroidManagerService**.

Figure 2-1 Functions of the QbAndroidManagerService



2.1.3 Data Reloading

When the board runs in quick boot mode, the memory mirroring technology is used. Therefore, data must be reloaded, to update the data to the status when the Android OS is shut down last time.

Spaces must be reserved to store the settings of some modules, such as the Ethernet and Wi-Fi modules, which may be modified.



CAUTION

Data reloading for the Android OS is provided in HiSilicon's basic versions. Customers need to determine whether the system services they added require data reloading.

2.2 Customization Description

2.2.1 Fastplay Support

The Kernel startup process is not completely implemented in the current quick boot version, so the Fastplay is not supported.

2.2.2 Tips for Image Creation

Problem Description

When creating the image, the system suspends all drives at the Kernel layer. Therefore, the system cannot output video information at this time. From the user's point of view, the TV has no video outputs during the image creation process (for about 5s).



Solution

An Android animation is displayed before the image is created, so a prompt like "Creating the image..." should be provided in the animation.

HiSilicon has made related solutions with a demo. Customers can perform the following operations based on the demo:

Step 1 Create a **bootanimation.zip** file based on the method for producing a standard Android boot animation.

For the method of creating a standard Android boot animation, take the following steps:

Step 2 Rename the created **bootanimation.zip** as **quickboot.zip**.

Step 3 Store the **quickboot.zip** in the **system/media/** directory.

HiSilicon's boot animation, **quickboot.zip**, has been preset in the HiSilicon version. The file is stored in the following directory based on the process for creating a standard Android boot animation:

```
device/Hisilicon/bigfish/prebuilts
```

Users can customize the boot animation based on the package and the preceding link.

----End

2.2.3 Production Mode

Some devices (such as the EMMC) on each board are different with globally unique IDs, which are used in the kernel. Therefore, the boot image for the current quick boot on a board cannot be used on other boards. Instead, the process for creating the image must be implemented completely on each board, to properly use the quick boot feature.

Therefore, the process for starting up the Android OS, creating the image, and restarting the Android OS must be implemented locally in production, to complete the quick boot making.



CAUTION

The process of production and programming is similar to that of the regular startup process. The image creation must be completed for the quick boot feature.

2.2.4 Upgrade and Default Settings Restoration

When the Android OS is upgraded or restored to default settings, the image must be created again for the quick boot feature.

2.2.5 Switchover Between Quick and Regular Boot Modes

The current quick boot feature also supports the switchover between the quick and regular boot modes in a dynamic manner.

The switchover can be achieved by invoking the **setQbEnable(boolean flag)** interface in the **QbAndroidManagerService**.



- When the **setQbEnable** is true, switch to the quick boot mode.
- When the **setQbEnable** is false, switch to the regular boot mode.

The related demo application package (APK) is provided in the current version. It is stored in the following directory:

```
device/HiSilicon/bigfish/development/apps/HiFastboot
```

When the macro for the quick boot feature is opened, the APK is compiled into the Android OS.

2.2.6 Image Creation Time Customization

The time of image creation directly affects various key factors, such as the size of the image, system status when the image is restored, data reloading process, and whether the system needs to relieve the dependence between the database and data partitions. Therefore, HiSilicon has optimized the time for image creation on the current feature. The quick boot feature does not support the customization of the image creation time.

2.2.7 Debugging Means

Using the quick boot feature, the memory image must be restored when the Android OS is started up. Therefore, when the Android OS is debugged, restore the OS to the regular boot mode (following the process described in section 2.2.5 "[Switchover Between Quick and Regular Boot Modes](#)"), add the debugging information on the basis of the regular boot mode, and recreate the image (following the process described in section 2.2.5 "[Switchover Between Quick and Regular Boot Modes](#)"). The modified file has been put into the image, and the log information can be captured by using the logcat, to debug the code.

Note that the **adb remount** command cannot be applied on the PC in quick boot mode. Therefore, if it is required to debug the quick boot feature and update the files in the **system/** directory, perform the following operations:

Step 1 Restore the Android OS to the regular boot mode based on the process described in section 2.2.5 "[Switchover Between Quick and Regular Boot Modes](#)."

Step 2 Connect the Android debug bridge (ADB) on the PC to that on the board.

Step 3 Put the files to be modified to the **/data/local/tmp/** directory that can be read and written through the ADB.

Step 4 Run the following command through the serial port on the board:

```
mount -o remount /system /system
```

Remount the **/system** directory.

Step 5 Copy the files under the **/data/local/tmp/** directory to specified directory.

Step 6 Run the following command through the serial port on the board:

```
sync
```

Step 7 Restart the board.



Step 8 After the Android OS is started up, switch it to the quick boot mode and debug it based on the process described in [2.2.5 "Switchover Between Quick and Regular Boot Modes."](#)

----End



CAUTION

The source code compiled for the quick boot is the user version which is different from the eng version. The jar file package compiled in the user version can be verified. For example, in the eng version, the **framework.jar** file is provided in the system directory **/system/framework/**, while in the user version, the **framework.jar** and **framework.odex** files are provided in the **/system/framework/** directory. Therefore, when adding debugging information, replace both the .jar file and the corresponding .odex file.

In the debugging process of the user version, if the debugging file generates the .jar and .odex files, replace both files at the same time. The verification function of the original Android OS must be shielded. Otherwise, after the image is recreated using the replaced .jar and .odex files, the Android OS cannot be started. Shield the verification of the original Android user version as follows:

Step 1 Open the **DexPrepare.cpp** file in the **dalvik/vm/analysis/** directory.

Step 2 Comment line 1306 out.

```
//const u1* signature = getSignature(cpe);
```

Step 3 Comment lines 1327 to 1332 out.

```
/*
if (memcmp(signature, ptr, kSHADigestLen) != 0) {
    ALOGI("DexOpt: mismatch dep signature for '%s'", cacheFileName);
    goto bail;
}
*/
```

Step 4 After the final version is debugged, change the previous codes back.

----End

The basic process for debugging is as follows:

Step 1 Comment the preceding lines out, and compile and program the quick boot source code.

Step 2 During the debugging, switch the Android OS to the regular boot mode based on the process described in section [2.2.5 "Switchover Between Quick and Regular Boot Modes."](#)

Step 3 Put the files to be replaced to the following directory:

```
/data/local/tmp/.
```

Step 4 To update files in the system directory, remount on the serial port of the board:

```
mount -o remount /system /system
```

Step 5 Copy the files to be replaced to the corresponding directory and run the **sync** command.



Step 6 Restart the Android OS. After the Android OS is properly started, recreate the image based on the process described in section 2.2.5 "Switchover Between Quick and Regular Boot Modes", and enable the quick boot feature.

Step 7 Check Android OS logs by using the logcat and debug the Android OS.

----End

2.2.8 System Service Customization

Because the image creation time cannot be customized in the quick boot feature, pay attention to the following items when adding system services:

- If the customer intends to add system services to the **SystemServer** before image creation, reload data after the Android OS is started.

For example, the user intends to add a **DVBService** before image creation (before the **startQbService** is invoked) in the **SystemServer**. The user and password data is available for the service. After the Android OS is restarted, these parameters are restored to their original status before image creation.

The preceding problem can be resolved by: setting a monitor on the **DVBService** to receive the **Intent.ACTION_QB_PROP_RELOAD** broadcast message. After that, obtain the stored data and update the data to the current value.

The added **Intent.ACTION_QB_PROP_RELOAD** is stored in the following path:

frameworks/base/core/java/android/content/Intent.java

The **BroadcastReceiver** can be customized to monitor the following broadcast message:

Intent.ACTION_QB_PROP_RELOAD

After receiving the broadcast message, reload the related data.

Figure 2-2 shows the preceding process.

Figure 2-2 Adding a service to the SystemServer before image creation (1)

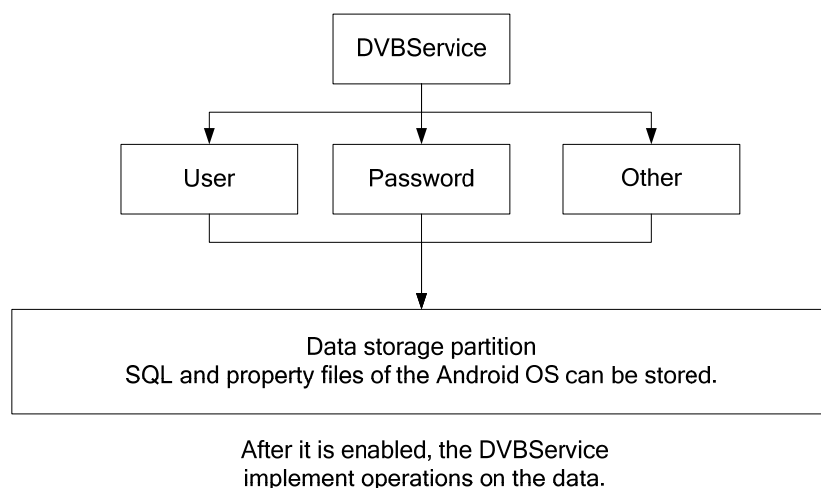
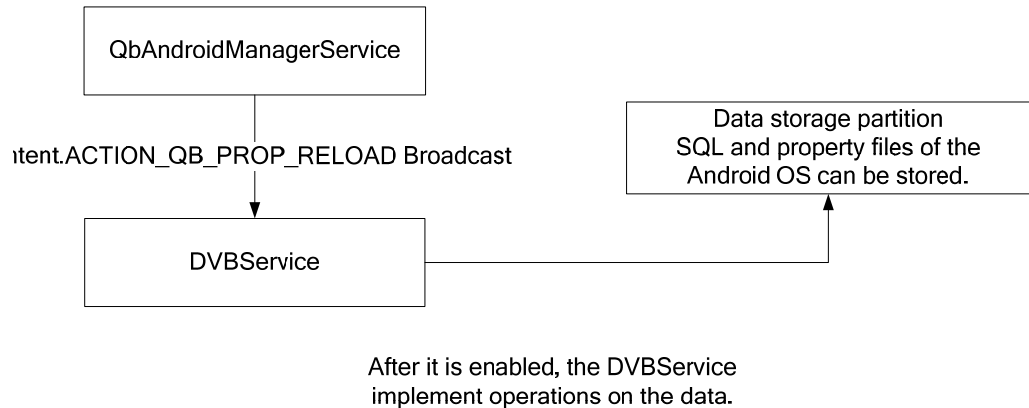




Figure 2-3 Adding a service to the SystemServer before image creation (2)



- If the service is added to the **SystemServer** after image creation, the preceding operations are not required.
- To add a service to the **init.rc**, pay attention to the following considerations:
Because the partition needs to be unmounted for image creation, ensure that there is no direct read and write dependence operations on the mounted partition when the image is created. To add a required task to the **init.rc**, ensure that there is no operation of reading and writing the data and cache partition in the task when the image is created. Otherwise, the image creation may fail.
Because the operations on the database are continuous after it is connected. Therefore, do not add any services with database operations to the **init.rc**.
Services with file reading and writing operations can be added to the **init.rc**. However, ensure that the file reading and writing are stopped at when the image is created. It is suggested not to implement file reading or writing operations or finish the operations in a short time.
- If the customer needs to customize and add a drive program, contact HiSilicon technical support engineers for help.
- The customer can add partitions as required without affecting the quick boot feature.

2.2.9 Security of Quick Boot

The quick boot feature supports the forcible switchover to the regular boot through designated button on the remote control.

- The function is disabled by default. Enable the function by modifying the SDK configuration options, such as those in the file `hi3798mdmo_hi3798mv100_android_cfg.mak`, and change the **#CFG_HI_BUILD_WITH_IR** is not set to **CFG_HI_BUILD_WITH_IR=y**
- After the function is enabled, once the Android OS is switched to the regular boot mode, the Android OS is started in the regular boot mode afterwards.
- The button on the remote control for the function is the **"Return"** button by default. To change the button, perform the following operation:
Change **0x6f90ff00** to a specific value in the **Android_Qb_CheckRemote(void)**function in **the/device/Hisilicon/bigfish/sdk/source/boot/product/android/ customer_android.c** file.



2.2.10 Description of Quick Boot Application Upgrade

In the quick boot feature, the following applications need to be upgraded:

1. Applications that are installed by the user and started through user interactions during the running of the Android OS (such as the applications of some video clients). For these applications, HiSilicon has provided adaptations.
2. Applications of Android OS launchers. In the quick boot mode, the applications are scanned and upgraded after the launcher is accessed. Therefore, if the customer intends to upgrade the launcher or applications started together with the launcher, recreate the image after the application is upgraded, to include the upgraded applications into the image.

2.2.11 Descriptions of Status Attributes in the Quick Boot

In the quick boot feature, HiSilicon has added and adapted some system attributes, as described in the following table:

Table 2-1 Descriptions of status attributes in the quick boot feature

Status	Attribute Value	Status Meaning
Determining whether the quick boot mode is enabled.	<code>persist.sys.qb.enable</code> true	The quick boot mode is enabled.
	<code>persist.sys.qb.enable</code> false	The regular boot mode is enabled.
Image creation status for the quick boot mode.	<code>persist.sys.qb.enable</code> true	The image creation for the quick boot mode is prepared.
	<code>persist.sys.qb.flag</code> false	
	<code>persist.sys.firstboot.flag</code> false	
	<code>persist.sys.qb.enable</code> true <code>persist.sys.qb.flag</code> true <code>persist.sys.firstboot.flag</code> false	The image creation for the quick boot mode is completed.



3 Configuration Description

3.1 Configuring the Quick Boot Feature

3.1.1 Enabling and Disabling

Disabling of the Quick Boot Feature

The feature is disabled by default in HiAndroidSTB V600R100C00SPC060.

Enabling of the Quick Boot Feature

Step 1 Open the configuration options for the quick boot feature.

Access the following source code directory:

```
/device/HiSilicon/chip directory
```

The chip directory refers to the directory for chips, such as Hi3798M V100 and Hi3796M V100. For example, if the Hi3798M V100 chip is used, access the following directory:

```
/device/HiSilicon/Hi3798MV100
```

Open the **customer.mk** file.

Change the following configuration:

```
BOARD_QBSUPPORT := true
```

Step 2 Compile the quick boot source code.

After the configuration is completed, return to the source code root directory, to compile the quick boot source code. The current quick boot only supports the input of the following in the following user version source code directory:

```
source build/envsetup.sh
```

Then, input:

lunch chip-user

The chip refers to the chip mentioned in step 1. For example, if the Hi3798M V100 chip is used, input:

```
lunch Hi3798MV100-user
```



Step 3 Finally, program the generated mirror image into the board, to complete the configuration for the quick boot mode.

----End



4 Maintenance and Test Guidelines

4.1 Test Environment Requirement

The test environment of the quick boot mode is the same as that of the regular boot mode in the STB-based Android OS. For detailed requirements for the test environment, refer to that for the regular boot mode in the STB-based Android OS.

4.2 Test Procedure

Step 1 Compile and program the quick boot source code to the board.

Step 2 Test the quick boot feature based on the procedure for the regular boot mode of the STB-based Android OS.

----End

4.3 Positioning Method

Log analysis is the main method for locating problems. Logs are printed by using the logcat.

4.4 Log Analysis

Implement log analysis based on the printed logs. For details, see section [2.2.7 "Debugging Means."](#)