SC20 Android Boot Speed Acceleration Guide

LTE Module Series

Rev. SC20_Android_Boot_Speed_Acceleration_Guide_V1.0

Date: 2016-09-26

www.quectel.com

Our aim is to provide customers with timely and comprehensive service. For any assistance, please contact our company headquarters:

Quectel Wireless Solutions Co., Ltd.
Office 501, Building 13, No.99, Tianzhou Road, Shanghai, China, 200233
Tel: +86 21 5108 6236
Email: info@quectel.com

Or our local office. For more information, please visit:
http://www.quectel.com/support/salesupport.aspx

For technical support, or to report documentation errors, please visit:
http://www.quectel.com/support/techsupport.aspx
Or email to: Support@quectel.com

GENERAL NOTES

QUECTEL OFFERS THE INFORMATION AS A SERVICE TO ITS CUSTOMERS. THE INFORMATION PROVIDED IS BASED UPON CUSTOMERS' REQUIREMENTS. QUECTEL MAKES EVERY EFFORT TO ENSURE THE QUALITY OF THE INFORMATION IT MAKES AVAILABLE. QUECTEL DOES NOT MAKE ANY WARRANTY AS TO THE INFORMATION CONTAINED HEREIN, AND DOES NOT ACCEPT ANY LIABILITY FOR ANY INJURY, LOSS OR DAMAGE OF ANY KIND INCURRED BY USE OF OR RELIANCE UPON THE INFORMATION. ALL INFORMATION SUPPLIED HEREIN IS SUBJECT TO CHANGE WITHOUT PRIOR NOTICE.

COPYRIGHT

About the Document

History

Revision	Date	Author	Description
1.0	2016-09-26	Hank HAN	Initial

Contents

Figure Index

1 Introduction

This document mainly introduces how to make Quectel SC20 module boot up faster through accelerating the boot speed of its Android system.

2 Android Boot Sequence

This chapter mainly introduces the boot sequence of Android.

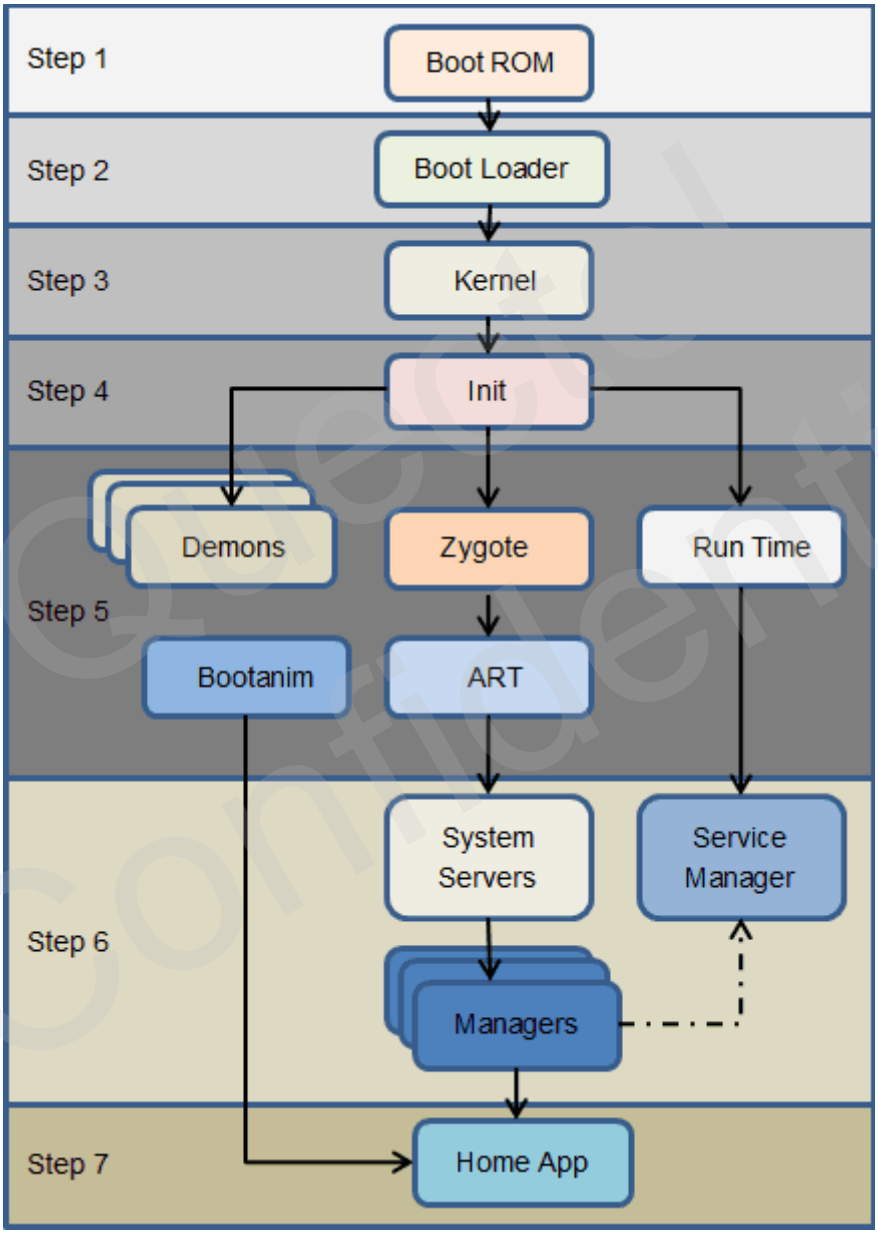

Figure 1: Android Boot Sequence

Step 1: Boot ROM

After the power key is pressed, the code at the IROM of CPU will be executed.

Step 2: Bootloader

Bootloader is used to initial some peripherals and to set the operating environment of Linux Kernel.

Step 3: Linux Kernel

The Linux Kernel is mainly used to set the cache, memory map and load drivers. Then Linux Kernel will search the init process of userspace.

Step 4: Init Process

Init process is the first process of userspace. It is mainly used to mount the file system like /sys, /dev and /proc, and to parse the *init.rc* file. Its source code is at <Rootdir>/system/core/init.c.

Step 5: Start Native Services

In this step, the Android native service like Zygote, SystemServer, MediaServer and SurfaceFlinger, etc. will be started.

Step 6: Start Java Services

In this step, Android will start some Java services and scan the Android packages.

Step 7: Start Home Screen

When native and Java services are running, Android will start the home screen. Then Android will check whether the drawing of wallpaper and keyguard is finished. If the drawing is finished, Android will set the property of service.bootanim.exit as 1.

Based on the information above, we can divide Android boot sequence into three phases: Bootloader phase (Step 1, 2), Kernel phase (Step 3) and Userspace phase (Step 4~7). The following chapters will discuss how to accelerate boot speed in each phase.

3 Bootloader Phase

3.1. General Overview

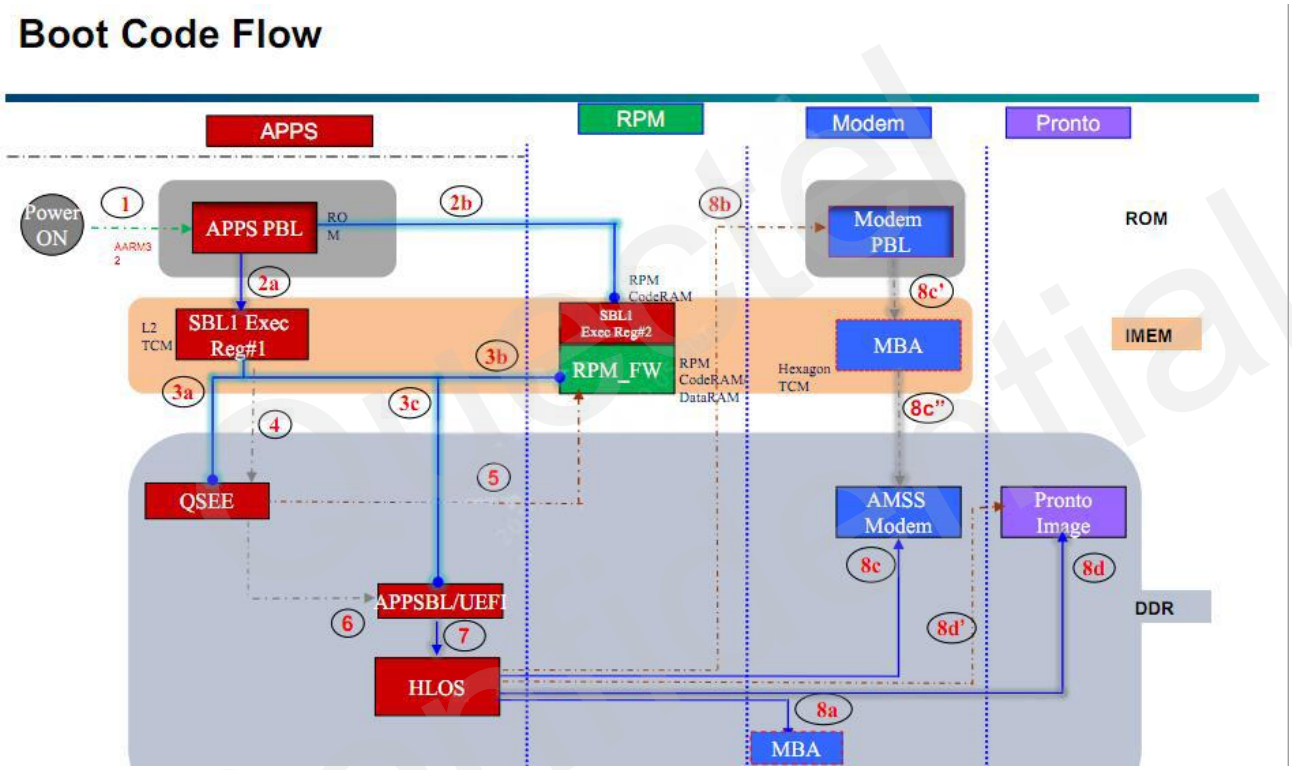

Figure 2: Boot Code Flow

According to the figure above, we can see that there are mainly three parts in Bootloader phase:

1. The execution of PBL in IROM of CPU;
2. The execution of SBL1 in L2 TCM;
3. The execution of LK in DDR.

3.2. Enable Timestamp

We can use the KPI of bootloader as timestamp to calculate the exact boot time spent in the phase.

The output time of KPI is as below:

```
<6>[0.215917] KPI: Bootloader start count = 32407 //The start time of LK is A.
<6>[0.215932] KPI: Bootloader end count = 72919 //The end time of LK is B.
<6>[0.215941] KPI: Bootloader display count = 64889
<6>[0.215950] KPI: Bootloader load kernel count = 2988
<6>[0.215960] KPI: Kernel MPM timestamp = 125192 //The time when bootloader is finished is C.
<6>[0.215969] KPI: Kernel MPM Clock frequency = 32768    //The Clock is D.
```

Then the boot time spent in the phase can be calculated as below:

Duration of PBL and SBL1: A/D = 32407/32768 = 0.99s
Duration of LK: (B-A)/D = (72919 - 32407)/32768= 1.24s
Duration of Bootloader: C/D –kmsg(C) = 125192/32768 – 0.215960 = 3.6s

3.3. Method to Accelerate Boot Time in Bootloader Phase

In Bootloader phase, the boot speed can be accelerated by reducing the size of boot.img BIN. We can modify the *Makefile* file in directory <Rootdir>/kernel/arch/arm/boot/dts/qcom/ to select the right .dts file that we need to reduce the BIN size.

```
-rwxrwx--- 1 root vboxsf  7234855      1 14:32 boot_modify.img
-rwxrwx--- 1 root vboxsf 13636904     23 13:21 boot_origin.img
```

Figure 3: Size of Boot.img BIN

Then the output time of KPI is as below:

```
<6>[0.177297] KPI: Bootloader start count = 32296
<6>[0.177311] KPI: Bootloader end count = 71796
<6>[0.177320] KPI: Bootloader display count = 64892
<6>[0.177328] KPI: Bootloader load kernel count = 1646
<6>[0.177337] KPI: Kernel MPM timestamp = 119545
<6>[0.177345] KPI: Kernel MPM Clock frequency = 32768
```

The boot time spent in the phase after modification can be calculated as below:

Duration of PBL and SBL1: A/D = 32296/32768 = 0.99s
Duration of LK: (B-A)/D = (71796 - 32296)/32768 = 1.20s
Duration of Bootloader: C/D –kmsg(C) = 119545/32768 – 0.177737 = 3.47s

3.4. Conclusion

Based on the information above, we can see that there is no much acceleration during Bootloader phase. And we can conclude that since the code of bootloader phase is based on the hardware, there is no too much work we can do to accelerate the boot speed.

4 Kernel Phase

4.1. Kernel Config file

The kernel config files are different when the Android source code is compiled as different versions. Customer can see the details in the below code.

In <Rootdir>/device/qcom/msm8909/AndroidBoard.mk:

```
ifeq ($(KERNEL_DEFCONFIG),)
    ifeq ($(TARGET_BUILD_VARIANT),user)
      KERNEL_DEFCONFIG := msm8909-1gb-perf_defconfig
    else
      KERNEL_DEFCONFIG := msm8909-1gb_defconfig
    endif
endif
```

The file named *msm8909-1gb_defconfig* is used in development stage and enables some debug functions. The other file *msm8909-1gb-perf_defconfig* is used in commercial mass production stage, and closes debug function to ensure the system's performance and power consumption.

4.2. Add Timestamp

When the kernel boot is finished, the init process will be started. Since there is no indication that kernel boot is finished in kernel log, so we can add a log as below into <Rootdir>/system/core/init.c file to judge whether the timestamp of the init process is started or not.

```
diff --git a/init/init.c b/init/init.c
index 9574fb4..b8f61dc 100644
--- a/init/init.c
+++ b/init/init.c
@@ -1010,7 +1010,7 @@ int main(int argc, char **argv)
     int keychord_fd_init = 0;
     bool is_charger = false;
     bool is_ffbm = false;
-
+    ERROR("the  init  process start! \n");
     if (!strcmp(basename(argv[0]), "ueventd"))
         return ueventd_main(argc, argv);
```

After adding the log, the boot time will be:

```
<3>[    7.025769] msm8x16-asoc-wcd sound.70: ASoC: CODEC (null) not registered
<3>[    7.025811] msm8x16-asoc-wcd sound.70: snd_soc_register_card failed (-517)
<6>[    7.025945] platform sound.70: Driver msm8x16-asoc-wcd requests probe deferral
<11>[    7.093762] init: the  init  process start!
<7>[    7.127503] SELinux: 2048 avtab hash slots, 8193 rules.
<7>[    7.135023] SELinux: 2048 avtab hash slots, 8193 rules.
```

4.3. Methods to Accelerate Boot Time in Kernel Phase

4.3.1. Disable Kernel Debug Macro

Remove the kernel debug macros shown below in *msm8909-1gb_defconfig*:

```
CONFIG_MODULE_SIG=y
CONFIG_MODULE_SIG_FORCE=y
CONFIG_MODULE_SIG_SHA512=y
CONFIG_REGULATOR_TPS65132=y
CONFIG_USB_BAM=y
CONFIG_CORESIGHT=y
CONFIG_CORESIGHT_EVENT=y
CONFIG_CORESIGHT_FUSE=y
CONFIG_CORESIGHT_CTI=y
CONFIG_CORESIGHT_TMC=y
CONFIG_CORESIGHT_TPIU=y
CONFIG_CORESIGHT_FUNNEL=y
CONFIG_CORESIGHT_REPLICATOR=y
CONFIG_CORESIGHT_STM=y
CONFIG_CORESIGHT_HWEVENT=y
CONFIG_CORESIGHT_ETM=y
CONFIG_CORESIGHT_MODEM_ETM=y
CONFIG_CORESIGHT_WCN_ETM=y
CONFIG_CORESIGHT_RPM_ETM=y
CONFIG_MSM_SMD_DEBUG=y
CONFIG_LOCKUP_DETECTOR=y
CONFIG_BOOTPARAM_SOFTLOCKUP_PANIC=y
CONFIG_DEBUG_KMEMLEAK=y
CONFIG_DEBUG_KMEMLEAK_DEFAULT_OFF=y
CONFIG_DEBUG_SPINLOCK=y
CONFIG_DEBUG_MUTEXES=y
```

```
CONFIG_DEBUG_ATOMIC_SLEEP=y
CONFIG_DEBUG_STACK_USAGE=y
CONFIG_DEBUG_MEMORY_INIT=y
CONFIG_DEBUG_LIST=y
CONFIG_FAULT_INJECTION=y
CONFIG_FAIL_PAGE_ALLOC=y
CONFIG_FAULT_INJECTION_DEBUG_FS=y
CONFIG_FAULT_INJECTION_STACKTRACE_FILTER=y
CONFIG_MSM_RTB=y
CONFIG_MSM_RTB_SEPARATE_CPUS=y
CONFIG_DYNAMIC_DEBUG=y
CONFIG_PANIC_ON_DATA_CORRUPTION=y
```

After disabling some kernel debug macros, the boot time will be:

```
<3>[    6.313535] msm8x16-asoc-wcd sound.70: ASoC: CODEC (null) not registered
<3>[    6.313575] msm8x16-asoc-wcd sound.70: snd_soc_register_card failed (-517)
<6>[    6.313704] platform sound.70: Driver msm8x16-asoc-wcd requests probe deferral
<11>[    6.345993] init: the  init  process start!
<7>[    6.376415] SELinux: 2048 avtab hash slots, 8193 rules.
<7>[    6.383129] SELinux: 2048 avtab hash slots, 8193 rules.
```

4.3.2. Disable Serial Console

Remove the config items in *msm8909-1gb_defconfig* to disable serial console as below:

```
CONFIG_SERIAL_MSM_HSL=y
CONFIG_SERIAL_MSM_HSL_CONSOLE=y
```

After disabling serial console, the boot time will be:

```
<6>[    2.604040] Freeing unused kernel memory: 60K (c0f00000 - c0f0f000)
<11>[    2.605067] init: the  init  process start!
<7>[    2.614883] SELinux: 2048 avtab hash slots, 8193 rules.
<7>[    2.621601] SELinux: 2048 avtab hash slots, 8193 rules.
```

4.4. Identify and Eliminate the Boot Time Problem

If the boot time in kernel phase is abnormal, debug the time of loading drivers can identify and eliminate the problem. There are many macros like module_init() in the kernel and customer can modify code as below to debug the time of loading drivers.

```
diff --git a/init/main.c b/init/main.c
index dd60587..ebb5ba3 100644
--- a/init/main.c
+++ b/init/main.c
@@ -684,7 +684,8 @@ int __init_or_module do_one_initcall(initcall_t fn)
        int count = preempt_count();
        int ret;

-       if (initcall_debug)
+       //if (initcall_debug)
+          if (1)
                ret = do_one_initcall_debug(fn);
        else
                ret = fn();
```

After debugging the time of loading driver, the boot time will be:

```
<7>[    0.722097] initcall evdev_init+0x0/0xc returned 0 after 0 usecs
<7>[    0.722108] calling  atkbd_init+0x0/0x18 @ 1
<7>[    0.722228] initcall atkbd_init+0x0/0x18 returned 0 after 102 usecs
<7>[    0.722241] calling  xpad_driver_init+0x0/0x18 @ 1
<6>[    0.722340] usbcore: registered new interface driver xpad
<7>[    0.722354] initcall xpad_driver_init+0x0/0x18 returned 0 after 98 usecs
<7>[    0.722366] calling  vkeys_driver_init+0x0/0xc @ 1
<7>[    0.723035] initcall vkeys_driver_init+0x0/0xc returned 0 after 637 usecs
<7>[    0.723050] calling  ft5x06_ts_init+0x0/0x10 @ 1
<6>[    0.723489] input: ft5x06_ts as /devices/soc.0/78b9000.i2c/i2c-5/5-0038/input/input0
<3>[    0.963202] i2c-msm-v2 78b9000.i2c: msm_bus_scale_register_client(mstr-id:86):0x7 (ok)
<3>[    0.963749] i2c-msm-v2 78b9000.i2c: ARB_LOST: msgs(n:2 cur:0 tx) bc(rx:1 tx:1) mode:FIFO slv_addr:0x38
<3>[    0.963797] ft5x06_ts 5-0038: ft5x06_i2c_read: i2c read error.
<3>[    0.963807] ft5x06_ts 5-0038: version read failed
<4>[    1.242972] ft5x06_ts: probe of 5-0038 failed with error -2
<7>[    1.243038] initcall ft5x06_ts_init+0x0/0x10 returned 0 after 507784 usecs
<7>[    1.243051] calling  goodix_ts_init+0x0/0x8c @ 1
<4>[    1.243460] goodix_parse_dt    ############    934
<4>[    1.243508] ##############   934
<6>[    1.243541] Goodix-TS 5-005d: Regulator get failed avdd ret=-19
<6>[    1.243554] Goodix-TS 5-005d: Regulator get failed vdd ret=-19
<6>[    1.243566] Goodix-TS 5-005d: Regulator get failed vcc_i2c ret=-19
<6>[    1.364738] Goodix-TS 5-005d: Goodix Product ID = 9147
<6>[    1.366955] Goodix-TS 5-005d: Sensor ID selected: 0
<7>[    1.368081] cma: dma_alloc_from_contiguous(cma ee2d2200, count 1, align 0)
<7>[    1.368513] cma: dma_alloc_from_contiguous(): returned af068
<3>[    1.368596] sps: BAM device 0x07884000 is not registered yet.
<6>[    1.368630] sps:BAM 0x07884000 is registered.
```

4.5. Conclusion

Based on the information above, we can see that the boot time in kernel phase is approx. 7s. If customer wants to accelerate the boot time in this phase, just disable serial console.

5 Userspace Phase

5.1. Analysis of Userspace Log

Customer can use ADB commands as below to get userspace logs when Android is booting.

```
adb logcat -v threadtime -b events > logcat_envents.txt
adb logcat -v threadtime > logcat.txt
```

1. The content of *logcat_envents.txt* is as below.

```
01-02 03:28:38.157     275     275 I boot_progress_start: 10565
//Start time of userspace,10565 microsecond
01-02 03:28:38.994     275     275 I boot_progress_preload_start: 11402
//Start time of preloading zygote process
01-02 03:28:41.724     275     275 I boot_progress_preload_end: 14132
//End time of preloading zygote process
01-02 03:28:42.018     762     762 I boot_progress_system_run: 14427
//Start time of system server
01-02 03:28:42.410     762     762 I boot_progress_pms_start: 14818
//Start time of scanning packages
01-02 03:28:42.634     762     762 I boot_progress_pms_system_scan_start: 15043
//Start time of scanning the packages in the directory of system
01-02 03:28:46.490     762     762 I boot_progress_pms_data_scan_start: 18898
//Start time of scanning the packages in the directory of data
01-02 03:28:46.532     762     762 I boot_progress_pms_scan_end: 18940
//End time of scanning packages
01-02 03:28:46.612     762     762 I boot_progress_pms_ready: 19020
//Package manager is ready.
01-02 03:28:49.217     762     762 I boot_progress_ams_ready: 21625
//Activity manager is ready, then home activity will start.
01-02 03:28:53.737     762     785 I boot_progress_enable_screen: 26146
//Home activity is finished.
```

We can check whether the duration of each stage is too long:

- Duration of starting Zygote process:

"boot_progress_preload_end" – "boot_progress_preload_star"

If the duration is long, check whether it preloads some unnecessary resources.

- Duration of scanning packages

1) Duration of scanning packages in the directory of system:
"boot_progress_pms_data_scan_start" – "boot_progress_pms_system_scan_start"

2) Duration of scanning packages in the directory of data:

"boot_progress_pms_scan_end" – "boot_progress_pms_data_scan_start"

If the scanning time of the two items is too long, then there is a need to check whether the time is reasonable.

- Duration of starting home activity:

"boot_progress_enable_screen" – "boot_progress_ams_ready"

The duration is from home activity starting to being idle. If it is too long, then there is a need to check the behavior of home activity. For example, whether there are too many things dealt in the UI process.

2. The content of *logcat.txt* is as below:

```
01-02 03:28:52.976  3121  3121 D Launcher: Broadcasting Home Idle Screen Intent
```

This log means the boot animation and launcher are already completed, and this indicates the Android Boot is completed.

5.2. Methods to Accelerate Boot Speed in Userspace Phase

5.2.1. Simplify the Preloaded Java Classes and Resources

All the Android applications are forked from the Zygote process. To share some java classes and resources, Zygote will preload some frequently-used java classes and resources into its memory during initialization. In this way there is no need to load other Android applications again after they are forked from Zygote so as to accelerate the boot speed. The related files are as below:

<Rootdir>/frameworks/base/preloaded-classes
<Rootdir>/frameworks/base/core/res/res/values/arrays.xml

5.2.2. Simplify the Native Service and Java Service

The boot of Android is essentially to start the native services and the java services. The functions of Android are provided by these services indirectly. But not all of the services are needed, customer can simplify the services according to their demand. The related files are as below:

<Rootdir>/system/core/rootdir/init.rc
<Rootdir>/frameworks/base/services/java/com/android/server/SystemServer.java

5.2.3. Simplify the Android Packages

When Android is booting, the android packages in given directories will be scanned. The boot speed can be more quicker if there is fewer preloaded android packages in the system. So simplify the preloaded Android packages is one of the important methods to accelerate the boot speed. The related files are as below:

<Rootdir>/build/target/product/xxxx.mk
<Rootdir>/device/qcom/common/xxxx.mk
<Rootdir>/vendor/..../xxxx.mk

5.2.4. Reduce the Kernel Log Level

Too much kernel log level in the kernel will slow down the boot speed of Android. So reduce the kernel log level can also accelerate the boot speed.

6 Testing

We can compile the Android source code as the userdebug version and the user version. The userdebug version is used in the development stage and the user version is used in the commercialization stage. Compared with the userdebug version, there are fewer kernel debug macros, apk applications and services which are all used for debugging in user version, so the boot speed will be more quicker when using the user version.

1. The boot speed using userdebug version is as below:

```
01-01 00:10:50.939   306   306 I boot_progress_start: 16235
01-01 00:10:52.092   306   306 I boot_progress_preload_start: 17389
01-01 00:10:55.112   306   306 I boot_progress_preload_end: 20409
01-01 00:10:55.401   789   789 I boot_progress_system_run: 20697
01-01 00:10:55.867   789   789 I boot_progress_pms_start: 21164
01-01 00:10:56.072   789   789 I boot_progress_pms_system_scan_start: 21369
01-01 00:11:00.429   789   789 I boot_progress_pms_data_scan_start: 25726
01-01 00:11:00.467   789   789 I boot_progress_pms_scan_end: 25764
01-01 00:11:00.541   789   789 I boot_progress_pms_ready: 25838
01-01 00:11:03.189   789   789 I boot_progress_ams_ready: 28485
01-01 00:11:07.834   789   811 I boot_progress_enable_screen: 33130
```

The boot speed is approx. 36.6s using a stopwatch.

2. The boot speed using user version is as below:

```
01-01 17:00:33.984   308   308 I boot_progress_start: 14711
01-01 17:00:34.657   308   308 I boot_progress_preload_start: 15385
01-01 17:00:37.093   308   308 I boot_progress_preload_end: 17820
01-01 17:00:37.323   760   760 I boot_progress_system_run: 18050
01-01 17:00:37.638   760   760 I boot_progress_pms_start: 18365
01-01 17:00:37.839   760   760 I boot_progress_pms_system_scan_start: 18566
01-01 17:00:40.980   760   760 I boot_progress_pms_data_scan_start: 21707
01-01 17:00:41.001   760   760 I boot_progress_pms_scan_end: 21728
01-01 17:00:41.068   760   760 I boot_progress_pms_ready: 21795
01-01 17:00:43.231   760   760 I boot_progress_ams_ready: 23958
01-01 17:00:46.156   760   781 I boot_progress_enable_screen: 26883
```

The boot speed is approx. 31.5s using a stopwatch.

3. Based on the user version we disabled the serial console, the boot speed is as below:

```
01-01 18:12:09.829   277   277 I boot_progress_start: 10484
01-01 18:12:10.549   277   277 I boot_progress_preload_start: 11204
01-01 18:12:13.025   277   277 I boot_progress_preload_end: 13679
01-01 18:12:13.314   763   763 I boot_progress_system_run: 13968
01-01 18:12:13.660   763   763 I boot_progress_pms_start: 14315
01-01 18:12:13.846   763   763 I boot_progress_pms_system_scan_start: 14500
01-01 18:12:17.096   763   763 I boot_progress_pms_data_scan_start: 17751
01-01 18:12:17.121   763   763 I boot_progress_pms_scan_end: 17776
01-01 18:12:17.191   763   763 I boot_progress_pms_ready: 17846
01-01 18:12:19.845   763   763 I boot_progress_ams_ready: 20500
01-01 18:12:22.998   763   784 I boot_progress_enable_screen: 23653
```

The boot speed is approx. 26.7s using a stopwatch.

4. After disabling the serial console, if simplify the preloaded java classes and resources, the android packages and services that customer does not need, and reduce the kernel log level, the boot time can be further reduced down to 26.7s or less.

7 Conclusion

In conclusion, the boot time using userdebug version is approx. 36s, and the time is approx. 31s while using user version. In bootloader phase, there is no too much work we can do to accelerate the boot time. In kernel phase, just disable the serial console can reduce the boot time a lot. After disabling the serial console, the boot time can be further reduced down to 26.7s or less by simplifying the preloaded java classes and resources, the android packages and services that customer does not need, and reducing the kernel log level.

8 Appendix A Reference

Table 1: Terms and Abbreviations

Abbreviation	Description
ADB	Android Debug Bridge
CPU	Central Processing Unit
DDR	Double Data Rate
KPI	Key Performance Indicator
LK	Little Kernel
PBL	Primary Boot Loader
ROM	Read-Only Memory
SBL	Secondary Boot Loader
TCM	Tightly-Coupled Memory