密级状态：绝密( ) 秘密( ) 内部( ) 公开(√)

Security Class: Top-Secret ( ) Secret ( ) Internal ( ) Public (√)

高可靠 OTA 使用说明文档

High_Reliable_OTA_Usage_Instruction

（技术部，第二系统产品部）

(Technical Department, R & D Dept. II)

文件状态： Status： [ ] 正在修改 [ ]Modifying [√] 正式发布 [√]Released	当前版本： Current Version:	V1.2
	作 者： Author:	纪大峣 Ji Dayao
	完成日期： Finish Date:	2019-10-15
	审 核： Auditor:	
	完成日期： Finish Date:	

福州瑞芯微电子股份有限公司

Fuzhou Rockchip Electronics Co., Ltd

版 本 历 史 Revision History

版本号 Version no.	作者 Author	修改日期 Revision Date	修改说明 Revision description	备注 Remark
V1.0	纪大峣 Ji Dayao	2018/5/25	初始版本 Initial version release	
V1.1	纪大峣 Ji Dayao	2019/10/15	1 重构文档组织结构 Refactoring document organization structure 2.新增步骤 2.2 Add step 2.2 3.完善步骤 2.1 和步骤 2.5 的描述 Improve the description of steps 2.1 and 2.5	
V1.2	纪大峣 Ji Dayao	2019/10/15	更新 2.5 节，增加 miniloader 的获取方式。 Update section 2.5 to add the way the miniloader is obtained.	

目 录 Contents

1 概述 Overview

本文档描述 RK 高可靠 OTA 方案使用说明，可以使用在 Rockchip RK3399 和 RK3288 的 Android 7.1 和 Android 8.1 SDK 平台上。

This document mainly describes the usage of RK high reliable OTA solution, which can be used on Android 7.1 and Android 8.1 SDK platforms on Rockchip RK3399 and RK3288.

该方案针对 U-BOOT 和 Trust 增加备份分区，确保设备正常出厂后其 Recovery 子系统总是可以正常引导。通过使用该方案，可以确保升级过程中的任意节点出现掉电意外，都不会使得设备变砖，总能够保证设备再次上电后能够再次进入 Recovery 进行继续 OTA 升级（完整包）或系统恢复。

The solution adds backup partition for U-BOOT and Trust, to ensure that Recovery sub system always can be loaded normally after the device is shipped out of the factory. Using this solution can prevent the device from becoming brick even if power down happens at any moment during upgrading and it can make the device enter Recovery again after reboot to continue OTA upgrading (complete package) or recover system.

2 使用步骤 Usage steps

Rockchip Android 7.1 和 Android 8.1 SDK 平台上，该升级方案默认关闭，可以通过如下步骤来使用该方案：

On Rockchip Android 7.1 and Android 8.1 SDK platforms, this upgrading solution is closed by default, and you can use it following below steps:

2.1 Config BoardConfig.mk

将 device/rockchip/common 中的 BoardConfig.mk 中开启

HIGH_RELIABLE_RECOVERY_OTA。如下：

Enable HIGH_RELIABLE_RECOVERY_OTA in the BoardConfig.mk under device /rockchip/common as below:

```
+HIGH_RELIABLE_RECOVERY_OTA := true
```

注意：如果你当前的 SDK 里，没有 HIGH_RELIABLE_RECOVERY_OTA 这个选项的默认配置，说明你当前的 SDK 还不支持本文档提到的高可靠 OTA 方案，请更新 RK 的对外服务器，确保拿到的 SDK 有这个选项的默认配置。

Note: if there is no default configuration of HIGH_RELIABLE_RECOVERY_OTA existing in your current SDK, it means your current SDK doesn't support the high reliable OTA solution mentioned in this document. Please sync with RK code server to get SDK with the default configuration of this option.

2.2 Config Build scripts

在 build/core 下打如下补丁：

Apply the following patch under build/core:

```
diff --git a/core/Makefile b/core/Makefile
old mode 100644
new mode 100755
index 2cc1588..6bdf3be
--- a/core/Makefile
+++ b/core/Makefile
@@ -1997,11 +1997,9 @@ ifeq ($(HIGH_RELIABLE_RECOVERY_OTA),true)
 endif
     $(call generate-userimage-prop-dictionary, $(zip_root)/META/misc_info.txt)
 ifneq ($(INSTALLED_RECOVERYIMAGE_TARGET),)
-ifneq ($(HIGH_RELIABLE_RECOVERY_OTA),true)
```

```
    $(hide)                                                    PATH=$(foreach
p,$(INTERNAL_USERIMAGES_BINARY_PATHS),$(p):)$$PATH
MKBOOTIMG=$(MKBOOTIMG) \
          ./build/tools/releasetools/make_recovery_patch $(zip_root) $(zip_root)
 endif
-endif
 ifeq ($(AB_OTA_UPDATER),true)
    @# When using the A/B updater, include the updater config files in the zip.
    $(hide)       $(ACP)       $(TOPDIR)system/update_engine/update_engine.conf
$(zip_root)/META/update_engine_config.txt
diff --git a/tools/releasetools/common.py b/tools/releasetools/common.py
index af4db33..133ab98 100755
--- a/tools/releasetools/common.py
+++ b/tools/releasetools/common.py
@@ -1654,6 +1654,7 @@ def MakeRecoveryPatch(input_dir, output_sink,
recovery_img, boot_img,

   full_recovery_image = info_dict.get("full_recovery_image", None) == "true"
   system_root_image = info_dict.get("system_root_image", None) == "true"
+   hrr_omit_recovery = info_dict.get("hrr_omit_recovery_image", None) == "true"

   if full_recovery_image:
     output_sink("etc/recovery.img", recovery_img.data)
@@ -1708,6 +1709,37 @@ fi
          'recovery_device': recovery_device,
          'bonus_args': bonus_args}
```

```
+
+   if hrr_omit_recovery:
+      if full_recovery_image:
+         sh_hrr = """#!/system/bin/sh
+if ! applypatch -c %(type)s:%(device)s:%(size)d:%(sha1)s; then
+   log -t recovery "Recovery image skip"
+else
+   log -t recovery "Recovery image already installed"
+fi
+""" % {'type': recovery_type,
+          'device': recovery_device,
+          'sha1': recovery_img.sha1,
+          'size': recovery_img.size}
+      else:
+         sh_hrr = """#!/system/bin/sh
+if                                    !                                    applypatch
-c %(recovery_type)s:%(recovery_device)s:%(recovery_size)d:%(recovery_sha1)s;
then
+   log -t recovery "Recovery image skip"
+else
+   log -t recovery "Recovery image already installed"
+fi
+""" % {'boot_size': boot_img.size,
+          'boot_sha1': boot_img.sha1,
+          'recovery_size': recovery_img.size,
```

```
+         'recovery_sha1': recovery_img.sha1,
+         'boot_type': boot_type,
+         'boot_device': boot_device,
+         'recovery_type': recovery_type,
+         'recovery_device': recovery_device,
+         'bonus_args': bonus_args}
+
+
    # The install script location moved from /system/etc to /system/bin
    # in the L release.  Parse init.*.rc files to find out where the
    # target-files expects it to be, and put it there.
@@ -1736,4 +1768,7 @@ fi

    print "putting script in", sh_location

-   output_sink(sh_location, sh)
+   if hrr_omit_recovery:
+     output_sink(sh_location, sh_hrr)
+   else:
+     output_sink(sh_location, sh)
diff --git a/tools/releasetools/ota_from_target_files.py
b/tools/releasetools/ota_from_target_files.py
index e88fa7d..b58a24d 100755
--- a/tools/releasetools/ota_from_target_files.py
+++ b/tools/releasetools/ota_from_target_files.py
@@ -834,6 +834,7 @@ def WriteBlockIncrementalOTAPackage(target_zip,
```

```
source_zip, output_zip):
    source_version = OPTIONS.source_info_dict["recovery_api_version"]
    target_version = OPTIONS.target_info_dict["recovery_api_version"]

+   hrr_omit_recovery = OPTIONS.info_dict.get("hrr_omit_recovery_image")
    if source_version == 0:
      print ("WARNING: generating edify script for a source that "
             "can't install it.")
@@ -1022,10 +1023,11 @@ else if get_stage("%(bcb_dev)s") != "3/3" then
      # patching on a device that's already on the target build will damage the
      # system. Because operations like move don't check the block state, they
      # always apply the changes unconditionally.
-     if blockimgdiff_version <= 2:
-       script.AssertSomeFingerprint(source_fp)
-     else:
-       script.AssertSomeFingerprint(source_fp, target_fp)
+     if not hrr_omit_recovery:
+       if blockimgdiff_version <= 2:
+         script.AssertSomeFingerprint(source_fp)
+       else:
+         script.AssertSomeFingerprint(source_fp, target_fp)
    else:
      if blockimgdiff_version <= 2:
        script.AssertSomeThumbprint(
@@ -1534,6 +1536,7 @@ def WriteIncrementalOTAPackage(target_zip, source_zip,
output_zip):
```

```
    source_version = OPTIONS.source_info_dict["recovery_api_version"]
    target_version = OPTIONS.target_info_dict["recovery_api_version"]
+   hrr_omit_recovery = OPTIONS.info_dict.get("hrr_omit_recovery_image")

    if source_version == 0:
      print ("WARNING: generating edify script for a source that "
@@ -1605,7 +1608,8 @@ def WriteIncrementalOTAPackage(target_zip, source_zip,
output_zip):
                                          OPTIONS.source_info_dict)

    if oem_props is None:
-     script.AssertSomeFingerprint(source_fp, target_fp)
+     if not hrr_omit_recovery:
+       script.AssertSomeFingerprint(source_fp, target_fp)
    else:
      script.AssertSomeThumbprint(
          GetBuildProp("ro.build.thumbprint", OPTIONS.target_info_dict),
```

2.3 Config parameter partition table

根据 device/rockchip 下使用的 parameter.txt 文件，手动生成 parameter_hrr.txt 文件。parameter_hrr.txt 文件在 parameter.txt 中的 trust 分区之后增加两个分区 uboot_ro 和 trust_ro，大小都是 0x00002000。同时下载工具 AndroidTools 增加 uboot_ro 和 trust_ro 下载分区。具体操作方法请参考《Android 增加一个分区配置指南 V1.00》文档说明。

Manually generate parameter_hrr.txt file according to the parameter.txt file used

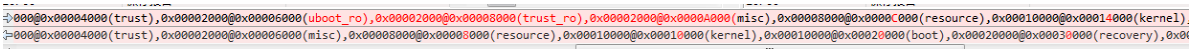
in device/rockchip. The parameter_hrr.txt file adds two partitions uboot_ro and trust_ro behind trust partition of parameter.txt, which size are both 0x00002000. Also add uboot_ro and trust_ro download partition for download tool AndroidTools. For the detailed operation method, please refer to the document of 《Android 增加一个分区配置指南 V1.00》.

以 RK3399 Android **8.1**SDK 为例：

Take RK3399 Android **8.1**SDK as an example:

在 device/rockchip/rk3399 下根据 parameter.txt 新增 parameter_hrr.txt，修改点如下：

The modification of adding parameter_hrr.txt in device/rockchip/rk3399 based on parameter.txt is as below:



注意：trust 分区后增加 uboot_ro 和 trust_ro 后，分区表后面的所有分区（misc, resource,kernel,boot,recovery 等等）偏移都要修改，即偏移地址都增加 0x00004000.

Note: after adding uboot_ro and trust_ro behind trust partition, all the back partitions in the partition table (such as misc, resource, kernel, boot, recovery and so on) should modify the offset, that is, the offset address adds 0x00004000.

一个完整的 **device/rockchip/rk3399/parameter_hrr.txt** 参考如下，

A complete device/rockchip/rk3399/parameter_hrr.txt reference is as below:

FIRMWARE_VER:7.1

MACHINE_MODEL:RK3368

MACHINE_ID:007

MANUFACTURER: RK3368

MAGIC: 0x5041524B

ATAG: 0x00200800

MACHINE: 3368

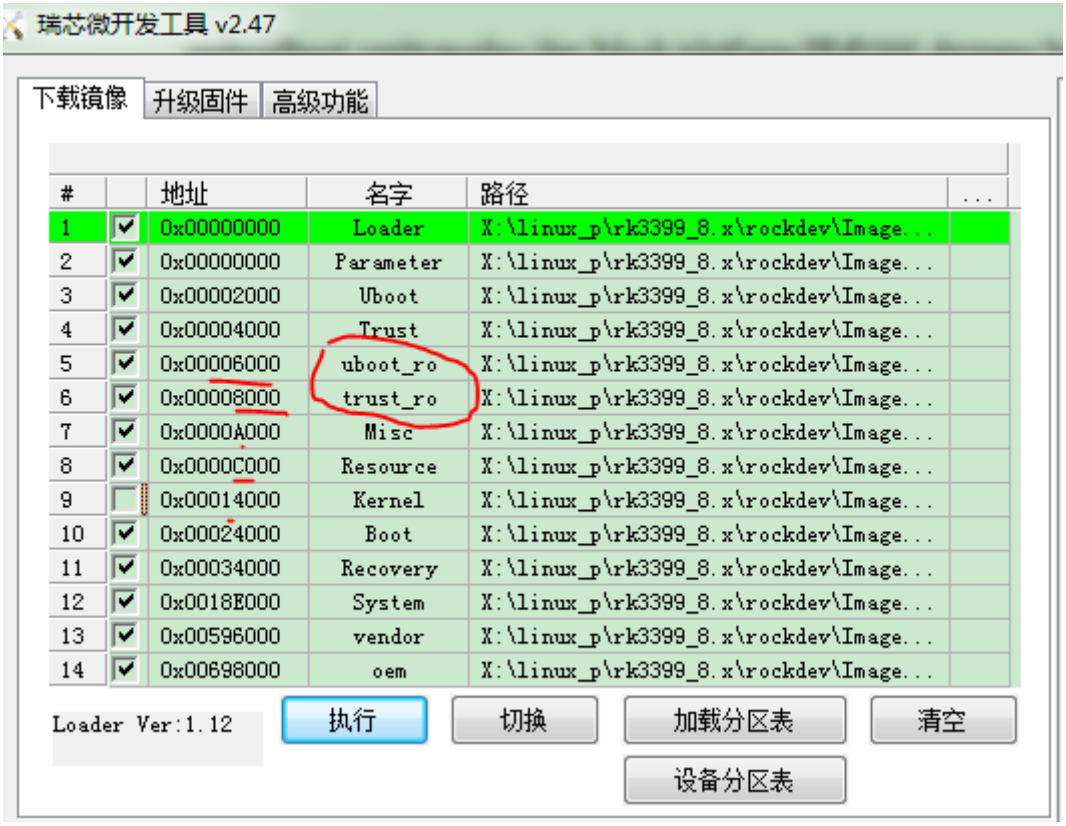
CHECK_MASK: 0x80

PWR_HLD: 0,0,A,0,1

CMDLINE: console=ttyFIQ0 androidboot.baseband=N/A androidboot.selinux=permissive androidboot.veritymode=/dev/block/platform/ff0f0000.dwmmc/by-name/metadata androidboot.hardware=rk30board androidboot.console=ttyFIQ0 init=/init initrd=0x62000000,0x00800000 mtdparts=rk29xxnand:0x00002000@0x00002000(uboot),0x00002000@0x00004000(trust),0x00002000@0x00006000(uboot_ro),0x00002000@0x00008000(trust_ro),0x00002000@0x0000A000(misc),0x00008000@0x0000C000(resource),0x00010000@0x00014000(kernel),0x00010000@0x00024000(boot),0x00020000@0x00034000(recovery),0x00038000@0x00054000(backup),0x00002000@0x0008C000(security),0x00100000@0x0008E000(cache),0x00400000@0x0018E000(system),0x00008000@0x0058E000(metadata),0x00100000@0x00596000(vendor),0x00100000@0x00698000(oem),0x00000400@0x00798000(frp),-@0x00798400(userdata)

下载工具 AndroidTool 修改后见如下截图：

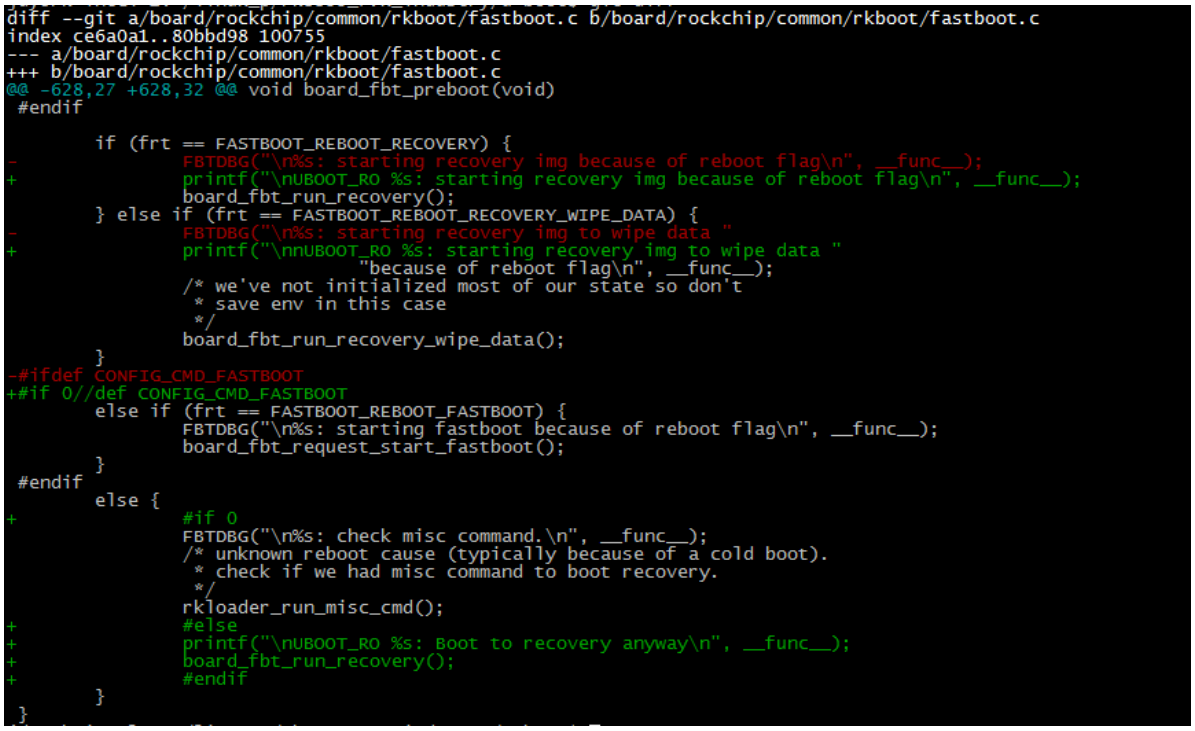
The download tool AndroidTool after modification is as below:



2.4 Generating uboot_ro.img

在 u-boot 打如下补丁，然后编译，编译成功后将 uboot.img 修改为 uboot_ro.img.

Apply the following patch in u-boot, compile, and then change uboot.img to uboot_ro.img after compiling successfully.



```
diff --git a/board/rockchip/common/rkboot/fastboot.c b/board/rockchip/common/rkboot/fastboot.c
index ce6a0a1..80bbd98 100755
--- a/board/rockchip/common/rkboot/fastboot.c
+++ b/board/rockchip/common/rkboot/fastboot.c
@@ -628,27 +628,32 @@ void board_fbt_preboot(void)
 #endif

        if (frt == FASTBOOT_REBOOT_RECOVERY) {
-               FBTDBG("\n%s: starting recovery img because of reboot flag\n", __func__);
+               printf("\nUBOOT_RO %s: starting recovery img because of reboot flag\n", __func__);
                board_fbt_run_recovery();
        } else if (frt == FASTBOOT_REBOOT_RECOVERY_WIPE_DATA) {
-               FBTDBG("\n%s: starting recovery img to wipe data "
+               printf("\nnUBOOT_RO %s: starting recovery img to wipe data "
                                "because of reboot flag\n", __func__);
                /* we've not initialized most of our state so don't
                 * save env in this case
                 */
                board_fbt_run_recovery_wipe_data();
        }
-#ifdef CONFIG_CMD_FASTBOOT
+#if 0//def CONFIG_CMD_FASTBOOT
        else if (frt == FASTBOOT_REBOOT_FASTBOOT) {
                FBTDBG("\n%s: starting fastboot because of reboot flag\n", __func__);
                board_fbt_request_start_fastboot();
        }
 #endif
        else {
+               #if 0
                FBTDBG("\n%s: check misc command.\n", __func__);
                /* unknown reboot cause (typically because of a cold boot).
                 * check if we had misc command to boot recovery.
                 */
                rkloader_run_misc_cmd();
+               #else
+               printf("\nUBOOT_RO %s: Boot to recovery anyway\n", __func__);
+               board_fbt_run_recovery();
+               #endif
        }
 }
```

diff --git a/board/rockchip/common/rkboot/fastboot.c b/board/rockchip/common/rkboot/fastboot.c

index ce6a0a1..80bbd98 100755

--- a/board/rockchip/common/rkboot/fastboot.c

+++ b/board/rockchip/common/rkboot/fastboot.c

@@ -628,27 +628,32 @@ void board_fbt_preboot(void)

#endif

if (frt == FASTBOOT_REBOOT_RECOVERY) {

- FBTDBG("\n%s: starting recovery img because of reboot flag\n", __func__);

+ printf("\nUBOOT_RO %s: starting recovery img because of reboot flag\n", __func__);

board_fbt_run_recovery();

} else if (frt == FASTBOOT_REBOOT_RECOVERY_WIPE_DATA) {

- FBTDBG("\n%s: starting recovery img to wipe data "

```
+       printf("\nnUBOOT_RO %s: starting recovery img to wipe data "
                "because of reboot flag\n", __func__);
        /* we've not initialized most of our state so don't
         * save env in this case
         */
        board_fbt_run_recovery_wipe_data();
    }
-#ifdef CONFIG_CMD_FASTBOOT
+#if 0//def CONFIG_CMD_FASTBOOT
    else if (frt == FASTBOOT_REBOOT_FASTBOOT) {
        FBTDBG("\n%s: starting fastboot because of reboot flag\n", __func__);
        board_fbt_request_start_fastboot();
    }
 #endif
    else {
+       #if 0
        FBTDBG("\n%s: check misc command.\n", __func__);
        /* unknown reboot cause (typically because of a cold boot).
         * check if we had misc command to boot recovery.
         */
        rkloader_run_misc_cmd();
+       #else
+       printf("\nUBOOT_RO %s: Boot to recovery anyway\n", __func__);
+       board_fbt_run_recovery();
+       #endif
    }
```

}

2.5 Generating uboot.img

去掉前面生成 uboot_ro 的补丁修改，在 u-boot 下按如下方式补丁，编译生成 uboot.img.

Remove the patch used to generate uboot_ro in last step, apply the patch in u-boot according to below method, and then compile to generate uboot.img.

（1） 获取支持高可靠升级的 miniloader（通常位于 RKDocs/rkxxxx/patches/），针对 rk3399 的 miniloader 位于 RKDocs/rk3399/patches/ ，如 rk3399_miniloader_v1.15_load_back_uboot.bin。获取后将该文件放置在 u-boot/tools/rk_tools/bin/rk33/目录下。

Get a miniloader that supports highly reliable upgrades (Usually located at RKDocs/rkxxxx/patches/). The miniloader for rk3399 is located at RKDocs/rk3399/patches/, such as rk3399_miniloader_v1.15_load_back_uboot.bin. After getting the file, place it in the u-boot/tools/rk_tools/bin/rk33/ directory.

然后根据需要修改 rk_tools/RKBOOT/RK3399MINIALL.ini，假设获取到的 Miniloader 名字为 rk3399miniloaderall.bin，则对应 RK3399MINIALL.ini 修改如下：

Then modify rk_tools/RKBOOT/RK3399MINIALL.ini as needed. If the name of the obtained Miniloader is rk3399miniloaderall.bin, the corresponding RK3399MINIALL.ini is modified as follows:

```
--- a/tools/rk_tools/RKBOOT/RK3399MINIALL.ini
+++ b/tools/rk_tools/RKBOOT/RK3399MINIALL.ini
@@ -15,6 +15,6 @@ NUM=2
 LOADER1=FlashData
 LOADER2=FlashBoot
 FlashData=tools/rk_tools/bin/rk33/rk3399_ddr_800MHz_v1.10.bin
```

-FlashBoot=tools/rk_tools/bin/rk33/rk3399_miniloader_v1.12.bin

+FlashBoot=tools/rk_tools/bin/rk33/

rk3399_miniloader_v1.15_load_back_uboot.bin

[OUTPUT]

PATH=rk3399_loader_v1.10.112.bin

（2） 打如下补丁，然后编译生成 uboot.img

Apply the following patch, and then compile to generate uboot.img.

```
diff --git a/board/rockchip/common/rkboot/fastboot.c
b/board/rockchip/common/rkboot/fastboot.c
index ce6a0a1..d859c37 100755
--- a/board/rockchip/common/rkboot/fastboot.c
+++ b/board/rockchip/common/rkboot/fastboot.c
@@ -63,6 +63,8 @@ int exit_uboot_charge_level = 0;
 int exit_uboot_charge_voltage = 0;
 int uboot_brightness = 1;

+extern void board_fbt_run_recovery(void);
+
 #ifdef CONFIG_UBOOT_CHARGE
 /**
  * return 1 if is charging.
@@ -256,8 +258,12 @@ void board_fbt_boot_failed(const char* boot)
 #ifdef CONFIG_CMD_BOOTRK
     if (!memcmp(BOOT_NAME, boot, sizeof(BOOT_NAME))) {
         printf("try to start recovery\n");
+        #if 0
```

```
        char *const boot_cmd[] = {"bootrk", RECOVERY_NAME};
        do_bootrk(NULL, 0, ARRAY_SIZE(boot_cmd), boot_cmd);
+       #else
+       board_fbt_run_recovery();
+       #endif
    } else if (!memcmp(RECOVERY_NAME, boot, sizeof(RECOVERY_NAME))) {
        printf("try to start backup\n");
        char *const boot_cmd[] = {"bootrk", BACKUP_NAME};
@@ -326,13 +332,71 @@ const disk_partition_t* board_fbt_get_partition(const
char* name)
    return get_disk_partition(name);
 }

+void board_fbt_set_recovery_for_hrr_0(void)
+{
+   struct bootloader_message bmsg;
+
+   printf("board_fbt_set_recovery_for_hrr_0\n");
+
+
+   memset((char *)&bmsg, 0, sizeof(struct bootloader_message));
+   strcpy(bmsg.command, "boot-recovery");
+   bmsg.status[0] = 0;
+   rkloader_set_bootloader_msg_for_hrr(&bmsg);
+}
+void board_fbt_set_recovery_for_hrr_32(void)
```

```
+{
+    struct bootloader_message bmsg;
+
+    printf("board_fbt_set_recovery_for_hrr_32\n");
+
+
+    memset((char *)&bmsg, 0, sizeof(struct bootloader_message));
+    strcpy(bmsg.command, "boot-recovery");
+    bmsg.status[0] = 0;
+    if(is_bootloader_msg_has_content())
+    {
+        printf("board_fbt_set_recovery_for_hrr_32 bcb has content\n");
+    }
+    else
+    {
+        rkloader_set_bootloader_msg(&bmsg);
+    }
+}
+
+void board_fbt_set_recovery_for_hrr_reset(void)
+{
+    printf("board_fbt_set_recovery_for_hrr_reset reset to miniloader\n");

-static void board_fbt_run_recovery(void)
+#if 0
+#ifdef CONFIG_CMD_BOOTRK
```

```
+       char *const boot_recovery_cmd[] = {"bootrk", "recovery"};
+       do_bootrk(NULL,          0,          ARRAY_SIZE(boot_recovery_cmd),
boot_recovery_cmd);
+#endif
+#else
+   do_reset(NULL, 0, 0, NULL);
+#endif
+}
+
+
+void board_fbt_set_recovery_for_hrr(void)
 {
+   board_fbt_set_recovery_for_hrr_0();
+   check_misc_info_offset_0_and_32();
+   board_fbt_set_recovery_for_hrr_reset();
+}
+
+
+void board_fbt_run_recovery(void)
+{
+#if 0
 #ifdef CONFIG_CMD_BOOTRK
     char *const boot_recovery_cmd[] = {"bootrk", "recovery"};
     do_bootrk(NULL, 0, ARRAY_SIZE(boot_recovery_cmd), boot_recovery_cmd);
 #endif
+#else
```

```
+    board_fbt_set_recovery_for_hrr();
+#endif

     /* returns if recovery.img is bad */
     FBTERR("\nfastboot: Error: Invalid recovery img\n");
@@ -346,7 +410,7 @@ void board_fbt_run_recovery_wipe_data(void)
     FBTDBG("Rebooting into recovery to do wipe_data\n");

     if (!board_fbt_get_partition("misc")) {
-          FBTERR("not found misc partition, just run recovery.\n");
+          printf("not found misc partition, just run recovery.\n");
           board_fbt_run_recovery();
     }

@@ -359,7 +423,6 @@ void board_fbt_run_recovery_wipe_data(void)
     board_fbt_run_recovery();
 }

-
 #ifdef CONFIG_RK_POWER
 static void board_fbt_low_power_check(void)
 {
@@ -628,10 +691,13 @@ void board_fbt_preboot(void)
 #endif

     if (frt == FASTBOOT_REBOOT_RECOVERY) {
```

```
-       FBTDBG("\n%s: starting recovery img because of reboot flag\n", __func__);
+       printf("\n%s: starting recovery img because of reboot flag\n", __func__);
+       #if 1
+       board_fbt_set_recovery_for_hrr_32();
+       #endif
        board_fbt_run_recovery();
    } else if (frt == FASTBOOT_REBOOT_RECOVERY_WIPE_DATA) {
-       FBTDBG("\n%s: starting recovery img to wipe data "
+       printf("\n%s: starting recovery img to wipe data "
                "because of reboot flag\n", __func__);
        /* we've not initialized most of our state so don't
         * save env in this case
diff --git a/board/rockchip/common/rkloader/rkloader.c b/board/rockchip/common/rkloader/rkloader.c
index 3afe20c..99a2643 100755
--- a/board/rockchip/common/rkloader/rkloader.c
+++ b/board/rockchip/common/rkloader/rkloader.c
@@ -205,6 +205,8 @@ void rkloader_change_cmd_for_recovery(PBootInfo boot_info, char * rec_cmd)
 #define MISC_SIZE              (MISC_PAGES * PAGE_SIZE)//48K
 #define MISC_COMMAND_OFFSET (MISC_COMMAND_PAGE * PAGE_SIZE / RK_BLK_SIZE)//32

+extern void board_fbt_run_recovery(void);
+
 int rkloader_run_misc_cmd(void)
```

```
 {
     struct bootloader_message *bmsg = NULL;
@@ -234,8 +236,12 @@ int rkloader_run_misc_cmd(void)
 #endif
         printf("got recovery cmd from misc.\n");
 #ifdef CONFIG_CMD_BOOTRK
+        #if 0
         char *const boot_cmd[] = {"bootrk", "recovery"};
         do_bootrk(NULL, 0, ARRAY_SIZE(boot_cmd), boot_cmd);
+        #else
+        board_fbt_run_recovery();
+        #endif
 #endif
         return false;
     } else if (!strcmp(bmsg->command, "boot-factory")) {
@@ -259,6 +265,97 @@ int rkloader_run_misc_cmd(void)
     return false;
 }

+int is_bootloader_msg_has_content(void)
+{
+    struct bootloader_message *bmsg = NULL;
+#ifdef CONFIG_RK_NVME_BOOT_EN
+    ALLOC_ALIGN_BUFFER(u8, buf, DIV_ROUND_UP(sizeof(struct
bootloader_message),
+            RK_BLK_SIZE) * RK_BLK_SIZE, SZ_4K);
```

```
+#else
+   ALLOC_CACHE_ALIGN_BUFFER(u8,        buf,        DIV_ROUND_UP(sizeof(struct
bootloader_message),
+               RK_BLK_SIZE) * RK_BLK_SIZE);
+#endif
+   const disk_partition_t *ptn = get_disk_partition(MISC_NAME);
+
+   if (!ptn) {
+       printf("misc partition not found!\n");
+       return 1;
+   }
+
+   bmsg = (struct bootloader_message *)buf;
+   if      (StorageReadLba(ptn->start      +      MISC_COMMAND_OFFSET,      buf,
DIV_ROUND_UP(
+                       sizeof(struct bootloader_message), RK_BLK_SIZE)) != 0) {
+       printf("failed to read misc\n");
+       return 1;
+   }
+
+   if(strlen(bmsg->command) > 0)
+   {
+       printf("is_bootloader_msg_has_content                bmsg->command=%s
bmsg->recovery=%s\n", bmsg->command, bmsg->recovery);
+       return 1;
+   }
```

```
+   else
+   {
+       return 0;
+   }
+}
+
+void check_misc_info_offset_0_and_32(void)
+{
+       struct bootloader_message *bmsg = NULL;
+#ifdef CONFIG_RK_NVME_BOOT_EN
+       ALLOC_ALIGN_BUFFER(u8,          buf,          DIV_ROUND_UP(sizeof(struct
bootloader_message),
+                  RK_BLK_SIZE) * RK_BLK_SIZE, SZ_4K);
+#else
+       ALLOC_CACHE_ALIGN_BUFFER(u8,    buf,    DIV_ROUND_UP(sizeof(struct
bootloader_message),
+                  RK_BLK_SIZE) * RK_BLK_SIZE);
+#endif
+       const disk_partition_t *ptn = get_disk_partition(MISC_NAME);
+
+       if (!ptn) {
+           printf("misc partition not found!\n");
+           return;
+       }
+
+       memset(buf,   0x0,   DIV_ROUND_UP(sizeof(struct   bootloader_message),
```

```
RK_BLK_SIZE));
+       bmsg = (struct bootloader_message *)buf;
+       if (StorageReadLba(ptn->start, buf, DIV_ROUND_UP(
+                       sizeof(struct bootloader_message), RK_BLK_SIZE)) != 0) {
+           printf("failed to read misc\n");
+           return;
+       }
+
+       if(strlen(bmsg->command) > 0)
+       {
+           printf("check_misc_info_offset_0       bmsg->command=%s       \n",
bmsg->command);
+
+       }
+       else
+       {
+           printf("check_misc_info_offset_0 bmsg->command is NULL \n");
+       }
+
+       memset(buf,   0x0,   DIV_ROUND_UP(sizeof(struct   bootloader_message),
RK_BLK_SIZE));
+       bmsg = (struct bootloader_message *)buf;
+       if   (StorageReadLba(ptn->start   +   MISC_COMMAND_OFFSET,   buf,
DIV_ROUND_UP(
+                       sizeof(struct bootloader_message), RK_BLK_SIZE)) != 0) {
+           printf("failed to read misc\n");
```

```
+            return;
+        }
+
+        if(strlen(bmsg->command) > 0)
+        {
+            printf("check_misc_info_offset_32        bmsg->command=%s        \n",
bmsg->command);
+            return;
+        }
+        else
+        {
+            printf("check_misc_info_offset_32 bmsg->command is NULL \n");
+            return;
+        }
+
+}
+

 void rkloader_fixInitrd(PBootInfo pboot_info, int ramdisk_addr, int ramdisk_sz)
 {
@@      -317,4      +414,24      @@      int      rkloader_set_bootloader_msg(struct
bootloader_message* bmsg)
            DIV_ROUND_UP(sizeof(struct bootloader_message), RK_BLK_SIZE));
 }

+int rkloader_set_bootloader_msg_for_hrr(struct bootloader_message* bmsg)
```

```
+{
+#ifdef CONFIG_RK_NVME_BOOT_EN
+   ALLOC_ALIGN_BUFFER(u8,          buf,          DIV_ROUND_UP(sizeof(struct
bootloader_message),
+           RK_BLK_SIZE) * RK_BLK_SIZE, SZ_4K);
+#else
+   ALLOC_CACHE_ALIGN_BUFFER(u8,       buf,       DIV_ROUND_UP(sizeof(struct
bootloader_message),
+           RK_BLK_SIZE) * RK_BLK_SIZE);
+#endif
+   memcpy(buf, bmsg, sizeof(struct bootloader_message));
+   const disk_partition_t *ptn = get_disk_partition(MISC_NAME);
+   if (!ptn) {
+       printf("misc partition not found!\n");
+       return -1;
+   }
+
+   return rkloader_CopyMemory2Flash((uint32)(unsigned long)buf, ptn->start/* +
MISC_COMMAND_OFFSET*/,
+           DIV_ROUND_UP(sizeof(struct bootloader_message), RK_BLK_SIZE));
+}
+

diff          --git          a/board/rockchip/common/rkloader/rkloader.h
b/board/rockchip/common/rkloader/rkloader.h
index 202a4c8..5500ccd 100755
```

```
--- a/board/rockchip/common/rkloader/rkloader.h
+++ b/board/rockchip/common/rkloader/rkloader.h
@@ -22,5 +22,11 @@ void rkloader_change_cmd_for_recovery(PBootInfo boot_info,
char * rec_cmd);
 int rkloader_run_misc_cmd(void);
 void rkloader_fixInitrd(PBootInfo pboot_info, int ramdisk_addr, int ramdisk_sz);
 int rkloader_set_bootloader_msg(struct bootloader_message* bmsg);
+int rkloader_set_bootloader_msg_for_hrr(struct bootloader_message* bmsg);
+int is_bootloader_msg_has_content(void);
+void check_misc_info_offset_0_and_32(void);
+
+
+

 #endif /* __RK_LOADER_H__ */
diff --git a/tools/rk_tools/RKBOOT/RK3399MINIALL.ini
b/tools/rk_tools/RKBOOT/RK3399MINIALL.ini
index f2387e9..2773406 100755
--- a/tools/rk_tools/RKBOOT/RK3399MINIALL.ini
+++ b/tools/rk_tools/RKBOOT/RK3399MINIALL.ini
@@ -15,6 +15,6 @@ NUM=2
 LOADER1=FlashData
 LOADER2=FlashBoot
 FlashData=tools/rk_tools/bin/rk33/rk3399_ddr_800MHz_v1.10.bin
-FlashBoot=tools/rk_tools/bin/rk33/rk3399_miniloader_v1.12.bin
+FlashBoot=tools/rk_tools/bin/rk33/rk3399miniloaderall.bin
```

[OUTPUT]

PATH=rk3399_loader_v1.10.112.bin

2.6 Rebuild Android system

Android 系统 make installclean 或者 make clean 后重新编译。

Re-compile after make installclean or make clean for Android system.

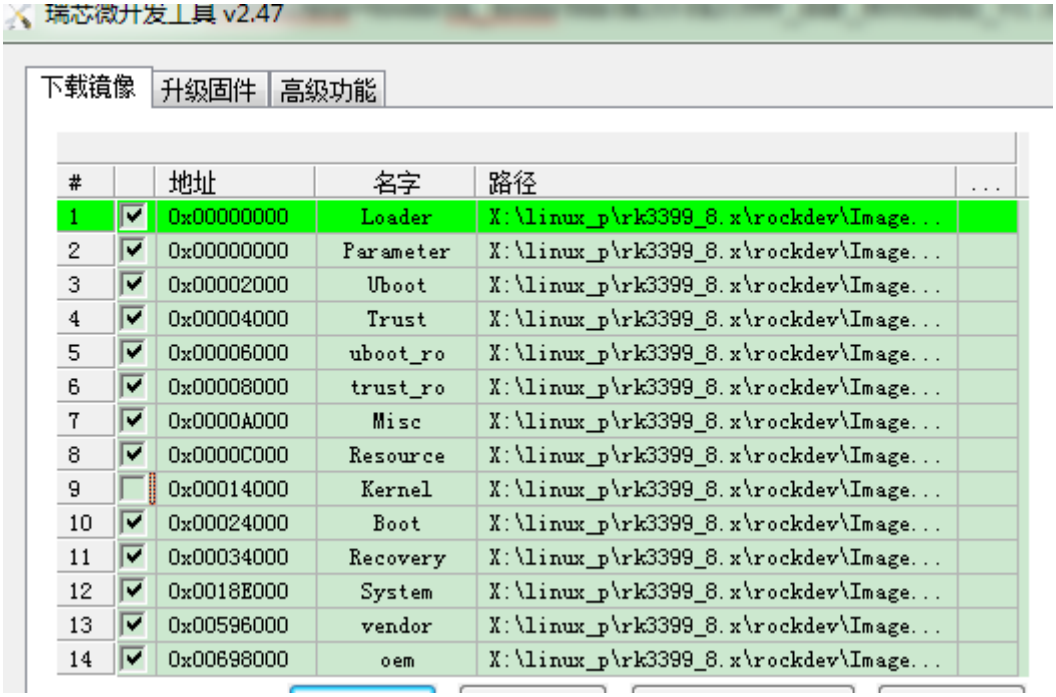
通过 make otapackage 生成升级包。

Generate the upgrading package through make otapackage.

2.7 Generate and flash images

编译完成后，通过./mkimage.sh ota 生成 images，将生成的 images 通过 AndroidTool 工具烧写到设备中。如：

After compilation, use ./mkimage.sh ota to generate images, and then flash the generated images into the device by AndroidTool. For example:



2.8 Generate OTA package and do update

对系统进行修改，然后通过 make otapackage 生成升级包，对系统进行升级。

Modify the system, and then generate the upgrading package through make otapackage, to upgrade the system.