密级状态：绝密（　　）　秘密（　　）　内部（ ）　公开（ √ ）

高可靠 OTA 使用说明文档

（技术部，第二系统产品部）

文件状态： [ ] 正在修改 [√] 正式发布	当前版本：	V1.0
	作　　者：	纪大峣
	完成日期：	2018-05-25
	审　　核：	
	完成日期：	

福州瑞芯微电子股份有限公司

Fuzhou Rockchip Electronics Co., Ltd

版 本 历 史

版本号	作者	修改日期	修改说明	备注
V1.0	纪大峣	2018/5/25	初始版本	

目 录

1 概述

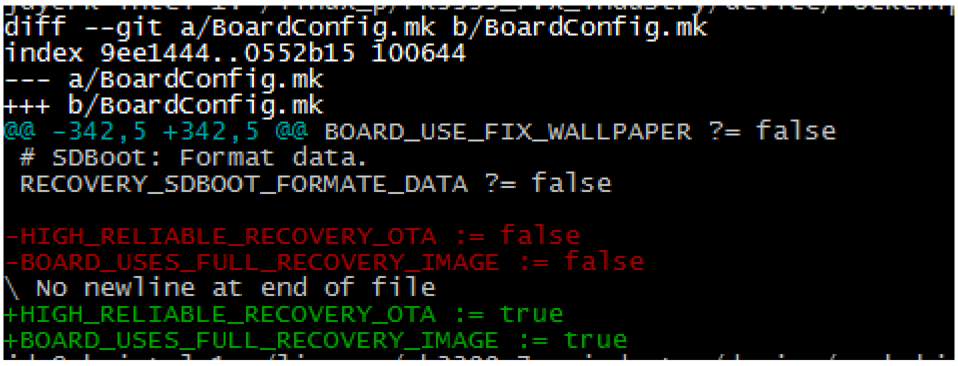
本文档描述 RK 高可靠 OTA 方案使用说明，可以使用在 Rockchip Android 7.1 和 Android 8.1 SDK 平台上。

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。

2 使用步骤

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：

1.将 device/rockchip/common 中的 BoardConfig.mk 中开启 HIGH_RELIABLE_RECOVERY_OTA 和 BOARD_USES_FULL_RECOVERY_IMAGE。如下：



```
diff --git a/BoardConfig.mk b/BoardConfig.mk
index 9ee1444..0552b15 100644
--- a/BoardConfig.mk
+++ b/BoardConfig.mk
@@ -342,5 +342,5 @@ BOARD_USE_FIX_WALLPAPER ?= false
 # SDBoot: Format data.
 RECOVERY_SDBOOT_FORMATE_DATA ?= false

-HIGH_RELIABLE_RECOVERY_OTA := false
-BOARD_USES_FULL_RECOVERY_IMAGE := false
\ No newline at end of file
+HIGH_RELIABLE_RECOVERY_OTA := true
+BOARD_USES_FULL_RECOVERY_IMAGE := true
```

注 意 ： 如 果 你 当 前 的 SDK 里 ， 没 有 HIGH_RELIABLE_RECOVERY_OTA 和 BOARD_USES_FULL_RECOVERY_IMAGE 这两个选项的默认配置，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 有这两个选项的默认配置。

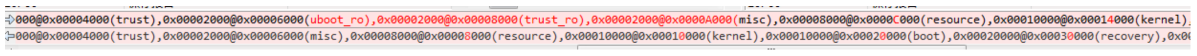
2.根据 device/rockchip 下使用的 parameter.txt 文件，手动生成 parameter_hrr.txt 文件。

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。

以 RK3399 Android **8.1**SDK 为例：

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



```
000@0x00004000(trust),0x00002000@0x00006000(uboot_ro),0x00002000@0x00008000(trust_ro),0x00002000@0x0000A000(misc),0x00008000@0x0000C000(resource),0x00010000@0x00014000(kernel),
000@0x00004000(trust),0x00002000@0x00006000(misc),0x00008000@0x00008000(resource),0x00010000@0x00010000(kernel),0x00010000@0x00020000(boot),0x00020000@0x00030000(recovery),0x0
```

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.

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

FIRMWARE_VER:7.1

MACHINE_MODEL:RK3368

MACHINE_ID:007

MANUFACTURER: RK3368

MAGIC: 0x5041524B

ATAG: 0x00200800

MACHINE: 3368

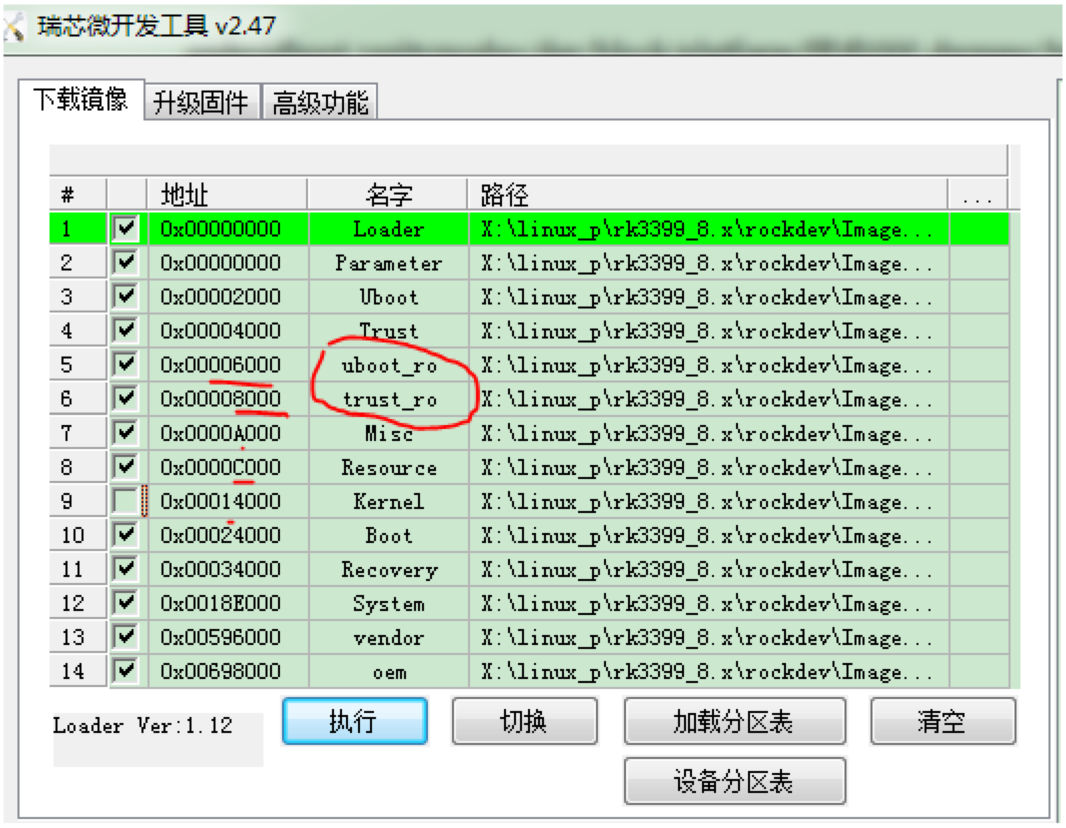
CHECK_MASK: 0x80

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(m etadata),0x00100000@0x00596000(vendor),0x00100000@0x00698000(oem),0x00000400@0x00798000(frp),-@0x00798400(userdata)

下载工具 AndroidTool 修改后见如下截图：

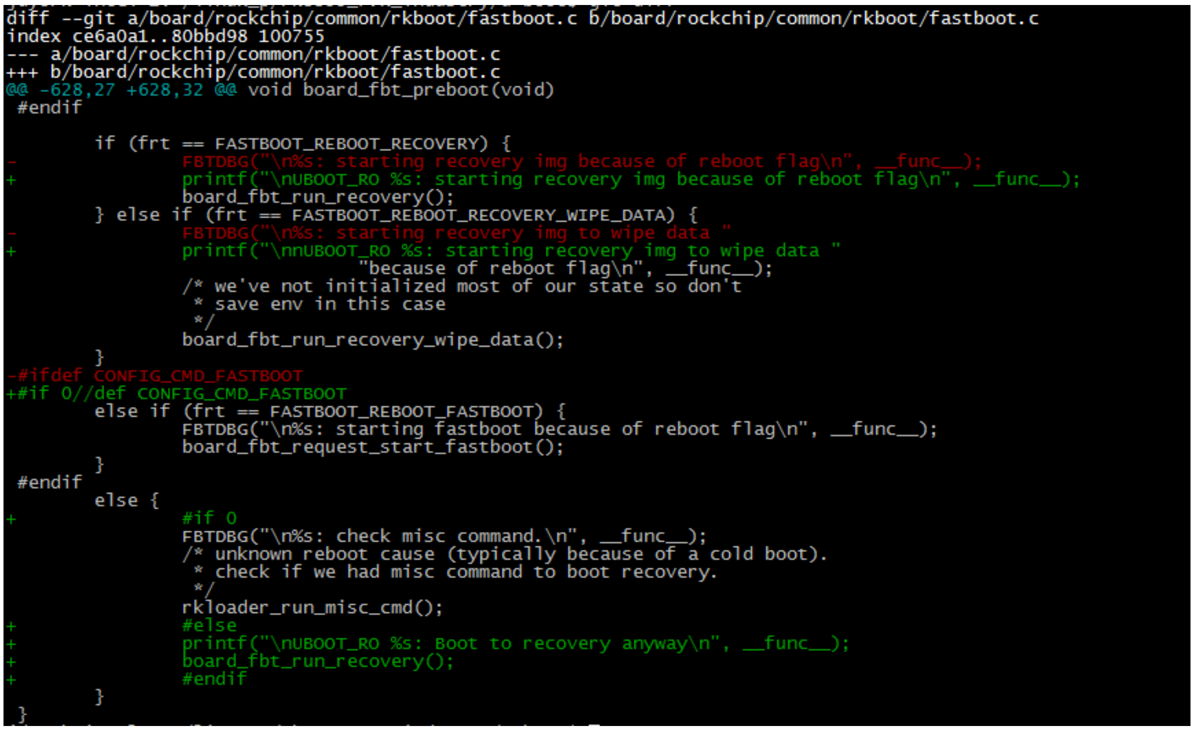


#		地址	名字	路径	...
1	✓	0x00000000	Loader	X:\linux_p\rk3399_8.x\rockdev\Image...	
2	✓	0x00000000	Parameter	X:\linux_p\rk3399_8.x\rockdev\Image...	
3	✓	0x00002000	Uboot	X:\linux_p\rk3399_8.x\rockdev\Image...	
4	✓	0x00004000	Trust	X:\linux_p\rk3399_8.x\rockdev\Image...	
5	✓	0x00006000	uboot_ro	X:\linux_p\rk3399_8.x\rockdev\Image...	
6	✓	0x00008000	trust_ro	X:\linux_p\rk3399_8.x\rockdev\Image...	
7	✓	0x0000A000	Misc	X:\linux_p\rk3399_8.x\rockdev\Image...	
8	✓	0x0000C000	Resource	X:\linux_p\rk3399_8.x\rockdev\Image...	
9		0x00014000	Kernel	X:\linux_p\rk3399_8.x\rockdev\Image...	
10	✓	0x00024000	Boot	X:\linux_p\rk3399_8.x\rockdev\Image...	
11	✓	0x00034000	Recovery	X:\linux_p\rk3399_8.x\rockdev\Image...	
12	✓	0x0018E000	System	X:\linux_p\rk3399_8.x\rockdev\Image...	
13	✓	0x00596000	vendor	X:\linux_p\rk3399_8.x\rockdev\Image...	
14	✓	0x00698000	oem	X:\linux_p\rk3399_8.x\rockdev\Image...	

Loader Ver:1.12

3.uboot_ro 的生成

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.



```
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
         }
  }
```

4.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.

（1）向 RK 获取支持高可靠升级的 miniloader，比如针对 rk3399 的 rk3399miniloaderall.bin。获取

后将该文件放置在 u-boot/ tools/rk_tools/bin/rk33/目录下

（2）打如下补丁，然后编译生成 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);
```

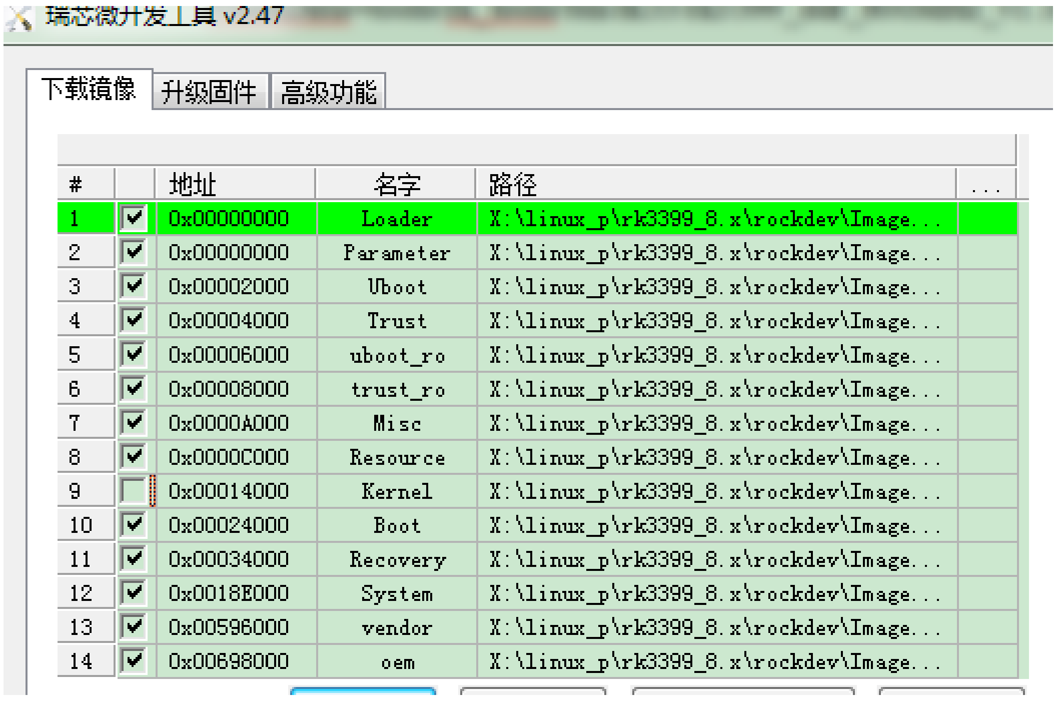
```
+#else
+        ALLOC_CACHE_ALIGN_BUFFER(u8,        buf,        DIV_ROUND_UP(sizeof(struct
bootloader_message),
+                        RK_BLK_SIZE) * RK_BLK_SIZE);
+#endif
+        const disk_partition_t *ptn = get_disk_partition(MISC_NAME);
+
+        if (!ptn) {
+            printf("misc partition 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
```

5.Android 系统 make clean 后重新编译。

通过 make otapackage 生成升级包。

6.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：



#		地址	名字	路径	...
1	✓	0x00000000	Loader	X:\linux_p\rk3399_8.x\rockdev\Image...	
2	✓	0x00000000	Parameter	X:\linux_p\rk3399_8.x\rockdev\Image...	
3	✓	0x00002000	Uboot	X:\linux_p\rk3399_8.x\rockdev\Image...	
4	✓	0x00004000	Trust	X:\linux_p\rk3399_8.x\rockdev\Image...	
5	✓	0x00006000	uboot_ro	X:\linux_p\rk3399_8.x\rockdev\Image...	
6	✓	0x00008000	trust_ro	X:\linux_p\rk3399_8.x\rockdev\Image...	
7	✓	0x0000A000	Misc	X:\linux_p\rk3399_8.x\rockdev\Image...	
8	✓	0x0000C000	Resource	X:\linux_p\rk3399_8.x\rockdev\Image...	
9		0x00014000	Kernel	X:\linux_p\rk3399_8.x\rockdev\Image...	
10	✓	0x00024000	Boot	X:\linux_p\rk3399_8.x\rockdev\Image...	
11	✓	0x00034000	Recovery	X:\linux_p\rk3399_8.x\rockdev\Image...	
12	✓	0x0018E000	System	X:\linux_p\rk3399_8.x\rockdev\Image...	
13	✓	0x00596000	vendor	X:\linux_p\rk3399_8.x\rockdev\Image...	
14	✓	0x00698000	oem	X:\linux_p\rk3399_8.x\rockdev\Image...	

7.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。