Rockchip UEFI 软件开发指南

文档标识：RK-KF-YF-935

发布版本：V1.4.0

日期：2022-08-22

文件密级：□绝密　□秘密　□内部资料　■公开

免责声明

本文档按"现状"提供，瑞芯微电子股份有限公司（"本公司"，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

商标声明

"Rockchip"、"瑞芯微"、"瑞芯"均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址：　福建省福州市铜盘路软件园A区18号

网址：　www.rock-chips.com

客户服务电话：+86-4007-700-590

客户服务传真：+86-591-83951833

客户服务邮箱：fae@rock-chips.com

前言

概述

文档作为 Rockchip Buildroot/Debian/Yocto Linux 系统软件开发指南，旨在帮助软件开发工程师、技术支持工程师更快上手 Rockchip Linux 平台的开发及调试。

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

日期	作者	版本	修改说明
2021-11-9	YiFeng Zhao	V1.0.0	初始版本
2022-01-12	Chris Zhong	V1.1.0	启动到Linux系统
2022-04-22	Chris Zhong	V1.2.0	增加GRUB启动方式
2022-05-18	Chris Zhong	V1.3.0	增加PCIE支持
2022-08-22	Chris Zhong	V1.4.0	增加硬件配置说明

目录

代码说明

RK3588 Linux SDK已经集成UEFI代码，请检查SDK中是否存在uefi目录。Rockchip相关目录如下：

1、edk2-platforms/Silicon/Rockchip

```
├── Applications  //测试Demo和Tool
├── Drivers   //通用驱动
├── Include   //通用头文件
├── Library   //通用库
├── RK3568    //RK3568专有IP驱动等
├── RK3588    //RK3588专有IP驱动等
├── Rockchip.dsc.inc   //通用配置
├── Rockchip.fdf.inc   //通用配置
└── RockchipPkg.dec    //通用配置
```

2、edk2-platforms/Platform/Rockchip

```
├── DeviceTree
|   └── rk3588.dtb   // 设备树文件，拷贝自
kernel/arch/arm64/boot/dts/rockchip/rk3588-evb1-lp4-v10-linux.dtb
├── RK3568
└── RK3588
```

```
    ├── AcpiTables
    ├── BootGraphicsResourceTableDxe
    ├── Include
    ├── Library
    ├── LogoDxe
    ├── RK3588.dec
    ├── RK3588.dsc
    ├── RK3588Dxe
    ├── RK3588.fdf
    └── RK3588GpioDxe
```

UEFI编译

RK3588 Linux SDK提供以下2种编译方式。

在uefi目录中执行脚本：

```
./make.sh rk3588 #默认编译rk3588
```

或使用SDK根目录中的build.sh脚本：

```
./build.sh uefi
```

脚本中已经配置了EDK2的环境变量，编译过程中会重编u-boot，编译完成后会在当前目录生成UEFI固件：uboot_uefi.img 和 RK3588_NOR_FLASH.img，将此文件烧录到uboot分区即可。EMMC启动的板子请选择uboot_uefi.img ， SPI Nor Flash启动的板子请选择 RK3588_NOR_FLASH.img。

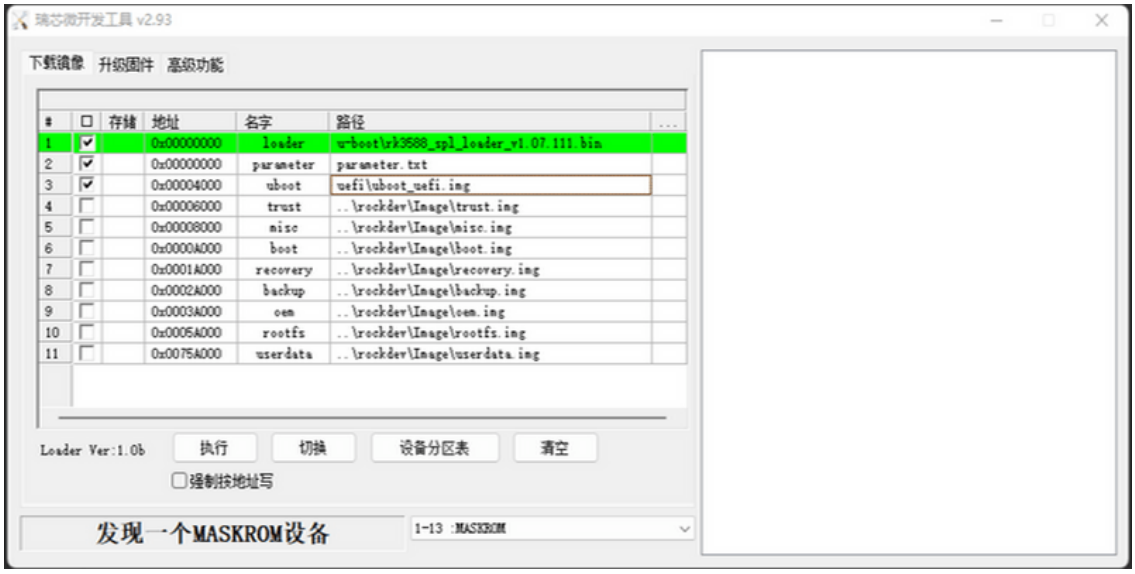
目前RK3588仅支持UEFI + DTB启动，还不支持UEFI + ACPI的启动方式。build.sh uefi编译时，会拷贝Kernel的DTB文件拷贝到UEFI目录中，因此编译UEFI前需要先完成Kernel编译。如果是用UEFI目录中的make.sh编译，需要手动拷贝dtb文件到uefi/edk2-platforms/Platform/Rockchip/DeviceTree/rk3588.dtb：

```
$cp kernel/arch/arm64/boot/dts/rockchip/rk3588-evb1-lp4-v10-linux.dtb uefi/edk2-platforms/Platform/Rockchip/DeviceTree/rk3588.dtb
```

烧录

EMMC启动板子和SPI Nor Flash启动板子的烧录有所不同：

- EMMC启动



EMMC启动的板子需要有一个parameter文件，因此至少需要烧录3个文件：

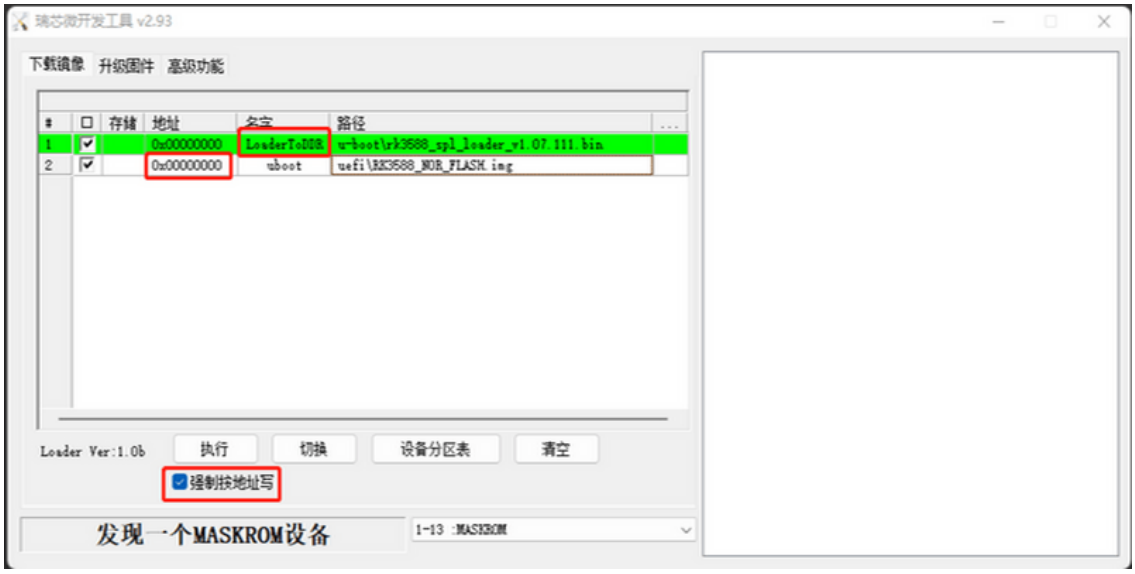
```
u-boot/rk3588_spl_loader_v1.07.111.bin
parameter.txt
uefi/uboot_uefi.img
```

其中uboot_uefi.img的地址由parameter.txt文件中uboot分区起始决定。

以下parameter.txt定义的UEFI固件烧录地址是0x4000，size不超过0x2000，单位是512 Byte。

```
FIRMWARE_VER: 1.0
MACHINE_MODEL: RK3588
MACHINE_ID: 007
MANUFACTURER: RK3588
MAGIC: 0x5041524B
ATAG: 0x00200800
MACHINE: 0xffffffff
CHECK_MASK: 0x80
PWR_HLD: 0,0,A,0,1
TYPE: GPT
CMDLINE:
mtdparts=rk29xxnand:0x00002000@0x00004000(uboot),0x00002000@0x00006000(misc)
,0x00020000@0x00008000(boot),0x00040000@0x00028000(recovery),0x00010000@0x00
068000(backup),0x01c00000@0x00078000(rootfs),0x00040000@0x01c78000(oem),-
@0x01d18000(userdata:grow)
uuid:rootfs=614e0000-0000-4b53-8000-1d28000054a9
uuid:boot=7A3F0000-0000-446A-8000-702F00006273
```

- SPI Nor Flash启动



SPI Nor Flash 烧录和EMMC不同，不需要parameter文件，只需：

```
boot/rk3588_spl_loader_v1.07.111.bin
uefi/RK3588_NOR_FLASH.img
```

注意上图中红色框部分。

启动

启动到系统有2种方式：Android Boot 启动和Grub 启动，下面分别介绍这2种启动方式。

Android Boot 启动

Android Boot 启动方式与RK3588 通用Linux启动方式几乎一样，不同点：

- 使用uboot_uefi.img和boot_uef.img分别替换原uboot和boot，其他分区保持不变；
- parameter中加入boot分区的UUID。

  build.sh kernel即可直接编译生成boot_uefi.img，用于烧录到boot分区，boot_uefi.img 和boot.img的区别是second段打包dtb，而不是resource.img。

boot分区识别

UEFI启动boot_uefi.img需要使用GPT的GUID等信息来确认分区，因此在通用parameter文件外，需要添加一行信息，指定boot分区的GUID:

```
uuid:boot=7A3F0000-0000-446A-8000-702F00006273
```

除此之外，还需要确认boot分区的offset和size。具体的分区定义是uefi/edk2-platforms\Platform\Rockchip\RK3588\RK3588.dsc中的PcdAndroidBootDevicePath变量，请注意这里的UUID，offset，size都需要和parameter文件一一对应，否则UEFI会找不到此分区，导致Android Boot失败。

```
gEmbeddedTokenSpaceGuid.PcdAndroidBootDevicePath|L"VenHw(100C2CFA-B586-4198-9B4C-1683D195B1DA)/HD(3,GPT,7A3F0000-0000-446A-8000-702F00006273,0x8000,0x20000)"
```

在上面的PcdAndroidBootDevicePath定义中，boot分区的UUID是7A3F0000-0000-446A-8000-702F00006273, offset是0x8000，size是0x20000。

通过UEFI Shell的map命令，也可以看到整个分区表，其中就能找到boot分区是：

```
BLK3: Alias(s):
          VenHw(100C2CFA-B586-4198-9B4C-1683D195B1DA)/HD(3,GPT,7A3F0000-0000-446
A-8000-702F00006273,0x8000,0x20000)
```

注意：Android Boot会检查Boot分区是否有Android的头信息，Boot不能使用FIT格式打包，而应使用Android格式打包，否则会提示找不到Boot。使用kernel补丁中修改的脚本，会生成Android格式的boot_uefi.img文件。而默认Linux SDK的编译脚本会编译出FIT格式的boot.img

Grub启动

UEFI会找ESP分区的/efi/boot/grubaa64.efi文件，这个文件就是GRUB的应用。一般系统安装盘中会有这个文件，GRUB的配置文件是grub.cfg，此文件中配置了kernel和initrd等信息，此部分内容请参考GRUB官方文档和各个系统安装文档，如《Debian_Install_Guide》。

这里简单介绍如何编译RK3588的kernel：

```
#编译kernel deb包
cd kernel
export CROSS_COMPILE=../prebuilts/gcc/linux-x86/aarch64/gcc-arm-10.3-2021.07-
x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu-
export ARCH=arm64
export LOCALVERSION=
export KDEB_PKGVERSION=5.10.84-1
make rockchip_linux_defconfig
make bindeb-pkg -j16

#编译动态库，initrd中需要用到
make modules
make modules_install INSTALL_MOD_PATH=$(realpath $CUR_DIR)/tmp
```

在SDK根目录中会生成一下几个文件，可用于kernel安装：

```
linux-headers-5.10.66_5.10.84-1_arm64.deb
linux-image-5.10.66_5.10.84-1_arm64.deb
linux-libc-dev_5.10.84-1_arm64.deb
```

另外kernel目录中的kernel/arch/arm64/boot/Image 文件可用于系统安装盘的启动：

```
cp kernel/arch/arm64/boot/Image /udisk/install.a64/vmlinuz
```

cmdline传递

GRUB启动kernel时，kernel会直接使用GRUB传递的cmdline，而不是dts中带的cmdline，因此如果需要使用ttyS作为debug串口，需要修改grub.cfg：

```
linux    /install.a64/vmlinuz  --- quiet
改成
linux    /install.a64/vmlinuz earlycon=uart8250,mmio32,0xfeb50000
console=ttyS2,1500000n8 --- quiet
```

如果没有在GRUB中指定：console=ttyS2,1500000n8，kernel会使用默认的ttyFIQ0作为debug口。

修改配置

UEFI硬件相关配置主要存在于以下文件：

```
make.sh //编译开关
edk2-platforms/Platform/Rockchip/RK3588/RK3588.dsc  //变量定义
edk2-platforms/Platform/Rockchip/RK3588/RK3588.fdf  //生成固件Image配置
edk2-
platforms/Platform/Rockchip/RK3588/Library/RockchipPlatfromLib/RockchipPlatfromL
ib.c //硬件相关信息
```

UEFI目前没有统一的GPIO驱动，因此主要修改方式为直接写寄存器。一个GPIO通常需要修改IOMUX和GPIO两个寄存器

SPI Nor 配置

根据原理图配置SPI Nor Flash使用的IO管脚，SDK默认配置为FSPI_M1，如有不同，请修改edk2-platforms/Platform/Rockchip/RK3588/Library/RockchipPlatfromLib/RockchipPlatfromLib.c中的NorFspiIomux函数：

```
void
EFIAPI
NorFspiIomux(void)
{
  /* io mux */
  MmioWrite32(NS_CRU_BASE + CRU_CLKSEL_CON78,
              (((0x3 << 12) | (0x3f << 6)) << 16) | (0x0 << 12) | (0x3f << 6));
#define FSPI_M1
#if defined(FSPI_M0)
   /*FSPI M0*/
  BUS_IOC->GPIO2A_IOMUX_SEL_L = ((0xF << 0) << 16) | (2 << 0); //FSPI_CLK_M0
  BUS_IOC->GPIO2D_IOMUX_SEL_L = (0xFFFFUL << 16) | (0x2222);
//FSPI_D0_M0,FSPI_D1_M0,FSPI_D2_M0,FSPI_D3_M0
  BUS_IOC->GPIO2D_IOMUX_SEL_H = ((0xF << 8) << 16) | (0x2 << 8); //FSPI_CS0N_M0
#elif defined(FSPI_M1)
  /*FSPI M1*/
  BUS_IOC->GPIO2A_IOMUX_SEL_H = (0xFF00UL << 16) | (0x3300);
//FSPI_D0_M1,FSPI_D1_M1
  BUS_IOC->GPIO2B_IOMUX_SEL_L = (0xF0FFUL << 16) | (0x3033);
//FSPI_D2_M1,FSPI_D3_M1,FSPI_CLK_M1
  BUS_IOC->GPIO2B_IOMUX_SEL_H = (0xF << 16) | (0x3); //FSPI_CS0N_M1
#else
  /*FSPI M2*/
  BUS_IOC->GPIO3A_IOMUX_SEL_L = (0xFFFFUL << 16) | (0x5555); //[FSPI_D0_M2-
FSPI_D3_M2]
  BUS_IOC->GPIO3A_IOMUX_SEL_H = (0xF0UL << 16) | (0x50); //FSPI_CLK_M2
  BUS_IOC->GPIO3C_IOMUX_SEL_H = (0xF << 16) | (0x2); //FSPI_CS0_M2
#endif
}
```

如需在SPI Nor中存储参数，需要关闭"emulated non-volatile variable mode"

```
diff --git a/edk2-platforms/Platform/Rockchip/RK3588/RK3588.dsc b/edk2-
platforms/Platform/Rockchip/RK3588/RK3588.dsc
index 1cf0ca85ff..6301a05a83 100644
--- a/edk2-platforms/Platform/Rockchip/RK3588/RK3588.dsc
+++ b/edk2-platforms/Platform/Rockchip/RK3588/RK3588.dsc
@@ -241,7 +241,7 @@
   #
   # Make VariableRuntimeDxe work at emulated non-volatile variable mode.
   #
-  gEfiMdeModulePkgTokenSpaceGuid.PcdEmuVariableNvModeEnable|TRUE
+  gEfiMdeModulePkgTokenSpaceGuid.PcdEmuVariableNvModeEnable|FALSE
```

另外，因EMMC与SPI_M0复用管脚，所以如果启用了SPI Nor并且配置了FSPI_M0，会出现EMMC通讯出错问题，请关闭EMMC：

```
diff --git a/edk2-platforms/Platform/Rockchip/RK3588/RK3588.fdf b/edk2-
platforms/Platform/Rockchip/RK3588/RK3588.fdf
index 24c01a8463..f936c8ba34 100644
--- a/edk2-platforms/Platform/Rockchip/RK3588/RK3588.fdf
+++ b/edk2-platforms/Platform/Rockchip/RK3588/RK3588.fdf
@@ -279,7 +279,7 @@ READ_LOCK_STATUS   = TRUE
   #INF Silicon/Synopsys/DesignWare/Drivers/DwEmmcDxe/DwEmmcDxe.inf
   INF Silicon/Rockchip/Drivers/MmcDxe/MmcDxe.inf
   #INF Silicon/Rockchip/Drivers/DwEmmcDxe/DwEmmcDxe.inf
-  INF Silicon/Rockchip/Drivers/SdhciHostDxe/SdhciHostDxe.inf
+  #INF Silicon/Rockchip/Drivers/SdhciHostDxe/SdhciHostDxe.inf
```

PCIE 配置

检查make.sh中是否已是能编译开关ROCKCHIP_PCIE30，如未开启，需要手动加入以下修改：

```
diff --git a/make.sh b/make.sh
index f8f8d68cee..3e35971a9a 100755
--- a/make.sh
+++ b/make.sh
@@ -13,6 +13,7 @@ case "$1" in
         *)
                 CHIP=3588
+                FLAGS+=" -D ROCKCHIP_PCIE30"
                 ;;
 esac
 echo Start to build rk$CHIP UEFI
```

另外需要注意PCIE的电源和reset配置，配置函数在edk2-platforms/Platform/Rockchip/RK3588/Library/RockchipPlatfromLib/RockchipPlatfromLib.c

请根据原理图，对照TRM修改寄存器

```
void
EFIAPI
Pcie30IoInit(void)
{
    /* Set reset and power IO to gpio output mode */
```

```
    MmioWrite32(0xFD5F808C, 0xf << (8 + 16)); /* gpio4b6 to gpio mode -> reset
*/
    MmioWrite32(0xFEC50008, 0x40004000); /* output */

    MmioWrite32(0xFD5F8070, 0xf << (12 + 16)); /* gpio3c3 to gpio mode -> power
*/
    MmioWrite32(0xFEC4000c, 0x80008); /* output */
}

void
EFIAPI
Pcie30PowerEn(void)
{
    MmioWrite32(0xFEC40004, 0x80008); /* output high to enable power */
}

void
EFIAPI
Pcie30PeReset(BOOLEAN enable)
{
    if(enable)
        MmioWrite32(0xFEC50000, 0x40000000); /* output low */
    else
        MmioWrite32(0xFEC50000, 0x40004000); /* output high */
}
```

显示配置

默认关闭了显示功能，如需打开，请修改make.sh中的FLAGS

```
diff --git a/make.sh b/make.sh
index f8f8d68cee..3e35971a9a 100755
--- a/make.sh
+++ b/make.sh
@@ -13,6 +13,7 @@ case "$1" in
         *)
                 CHIP=3588
+                FLAGS+=" -D ROCKCHIP_VOPEN"
                 ;;
 esac
 echo Start to build rk$CHIP UEFI
```

USB配置

USB2.0 和Type-c 的host功能默认已开启，只需使能VBUS供电即可，如硬件与EVB不同，可修改edk2-platforms/Platform/Rockchip/RK3588/Library/RockchipPlatfromLib/RockchipPlatfromLib.c的UsbPortPowerEnable函数：

```
#define GPIO4_BASE          0xFEC50000
#define GPIO_SWPORT_DR_L    0x0000
#define GPIO_SWPORT_DR_H    0x0004
#define GPIO_SWPORT_DDR_L   0x0008
#define GPIO_SWPORT_DDR_H   0x000C
```

```
void
EFIAPI
UsbPortPowerEnable (void)
{
  /* enable usb host vbus supply */
  MmioWrite32(GPIO4_BASE + GPIO_SWPORT_DR_L, (0x0100UL << 16) | 0x0100); /*
GPIO4_B0 */
  MmioWrite32(GPIO4_BASE + GPIO_SWPORT_DDR_L, (0x0100UL << 16) | 0x0100);
  /* enable usb otg0 vbus supply */
  MmioWrite32(GPIO4_BASE + GPIO_SWPORT_DR_H, (0x0100UL << 16) | 0x0100); /*
GPIO4_D0 */
  MmioWrite32(GPIO4_BASE + GPIO_SWPORT_DDR_H, (0x0100UL << 16) | 0x0100);
}
```