

Rockchip RK356X Linux NVR SDK 快速入门

文件标识: RK-JC-YF-542

发布版本: V1.6.0

日期: 2023-02-03

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。

版权所有 © 2023 瑞芯微电子股份有限公司

超越合理使用范畴, 非经本公司书面许可, 任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部, 并不得以任何形式传播。

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

Rockchip NVR SDK入门以及编译使用指南。

产品版本

芯片名称	内核版本
RK356X	Linux 4.19

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

版本号	作者	修改日期	修改说明
V0.1.0	XYP	2021-4-1	初始版本
V1.0.0	XYP	2021-8-5	发布版本，完善文档增加FAQ
V1.1.0	XYP	2021-9-14	完善FAQ
V1.2.0	XYP	2021-10-29	增加FAQ以及媒体FAQ文档索引
V1.3.0	XYP	2022-01-27	修改RKMPI目录以及编译介绍
V1.3.1	XYP	2022-04-06	修复uboot以及显示相关问题
V1.4.0	XYP	2022-06-11	修复PCIE3信号兼容性问题添加debugfs、优化显示直通和同步模式
V1.5.0	XYP	2022-07-28	增加拼接同步pll、clk调节接口功能、更新RKMPI V1.7.2
V1.5.1	XYP	2022-11-11	更新SDK下载地址
V1.6.0	XYP	2023-02-03	更新flash驱动、更新驱动修复错误片源解码问题 支持EDP热拔插功能、更新RKMPI V1.9.1

目录

Rockchip RK356X Linux NVR SDK 快速入门

1. 介绍
2. 开发包目录说明
3. 软件更新记录
4. SDK编译说明
 - 4.1 查看编译命令
 - 4.2 自动编译
 - 4.3 各模块编译及打包
 - 4.3.1 U-Boot编译
 - 4.3.2 Kernel编译
 - 4.3.3 Rootfs编译
 - 4.3.4 固件打包
 - 4.4 应用编译
5. RKMPI媒体包
6. 固件烧写
7. SecureBoot功能
8. FAQ

1. 介绍

基于 rk356x_linux_release_v1.1.3_20210805.xml RK356X 完整linux SDK裁剪而来，针对类NVR产品优化了多视频播放能力等。

SDK下载地址:

```
repo init --repo-url https://gerrit.rock-chips.com:8443/repo-release/tools/repo \
-u https://gerrit.rock-chips.com:8443/linux/rockchip/platform/manifests \
-b linux -m rk356x_nvr_linux_lite.xml

.repo/repo/repo sync -c --no-tags
```

如遇问题请参考 [Rockchip_User_Guide_SDK_Application_And_Synchronization_CN.pdf](#)

2. 开发包目录说明

SDK目录包含有 kernel、u-boot、tools、docs、rkbina 等目录。每个目录或其子目录会对应一个 git 工程，提交需要在各自的目录下进行。

```
- SDK
-- docs           //存放开发指导文件、平台支持列表、工具使用文档、Linux 开发指南等
-- kernel         //存放 Kernel 4.19 开发的代码。
-- rkbina         //存放 Rockchip 相关 Binary 和工具
-- tools          //存放 Linux 和 Window 操作系统下常用工具。
-- u-boot         //存放基于 v2017.09 版本进行开发的 U-Boot 代码。
-- IMAGE          //存放每次生成编译时间、XML、补丁和固件目录。
-- rockdev        //存放编译输出固件。
-- build          //存放编译脚本、rootfs以及toolchain编译工具链。
```

3. 软件更新记录

V1.6.0版本对比上一版本V1.5.1主要更新点描述:

序号	更新简要描述	详细描述	备注
1	支持EDP热拔插功能	<p>【问题描述】支持EDP热拔插功能</p> <p>【原因分析】不涉及</p> <p>【解决方法】不涉及</p> <p>【主要修改文件】</p> <p>kernel:</p> <p>drivers/gpu/drm/bridge/analogix/analogix_dp_core.c</p>	必须更新
2	更新flash驱动	<p>【问题描述】更新flash驱动，增加兼容性</p> <p>【原因分析】不涉及</p> <p>【解决方法】不涉及</p> <p>【主要修改文件】</p> <p>kernel: drivers/rkflash/ u-boot: drivers/spi/、drivers/mtd/ rkbin: bin/rk35/</p>	必须更新
3	多媒体MPI版本更新到V1.9.1	<p>【问题描述】多媒体MPI版本更新到V1.9.1</p> <p>【原因分析】不涉及</p> <p>【解决方法】不涉及</p> <p>【主要修改文件】</p> <p>build: app/RKMPI_Release/</p>	必须更新
4	更新驱动修复错误片源解码问题	<p>【问题描述】错误片源解码概率异常</p> <p>【原因分析】错误情况下，解码器和iommu访问硬件异常</p> <p>【解决方法】修改流程统一处理异常访问</p> <p>【主要修改文件】</p> <p>kernel: drivers/video/rockchip/mpp</p>	必须更新
5	修复eth频繁down和up异常	<p>【问题描述】频繁对eth做dump和up操作导致eth异常</p> <p>【原因分析】频繁操作导致没有free完成又去申请</p> <p>【解决方法】修改流程处理free状态</p> <p>【主要修改文件】</p> <p>kernel:</p> <p>drivers/net/ethernet/stmicro/stmmac/stmmac_main.c</p>	必须更新
6	更新crypto驱动	<p>【问题描述】更新crypto驱动</p> <p>【原因分析】不涉及</p> <p>【解决方法】不涉及</p> <p>【主要修改文件】</p> <p>kernel: crypto/ kernel: drivers/crypto/rockchip/</p>	可选更新
7	修复pll调节异常问题	<p>【问题描述】vp0、vp1都绑定到hpll的时候，vp1频率不对</p> <p>【原因分析】pll微调功能导致vp1频率不对</p> <p>【解决方法】修改clk文件pll微调只影响hpll</p> <p>【主要修改文件】</p> <p>kernel: drivers/clk/rockchip/clk-rk3568.c drivers/clk/rockchip/clk-pll.c</p>	必须更新

序号	更新简要描述	详细描述	备注
8	添加节点支持vp0和vp1和vp2开关crtc	<p>【问题描述】支持vp0和vp1和vp2开关crtc</p> <p>【原因分析】不涉及</p> <p>【解决方法】不涉及</p> <p>【主要修改文件】</p> <p>kernel:</p> <p>drivers/gpu/drm/rockchip/rockchip_drm_vop2.c</p>	必须更新

4. SDK编译说明

根目录下有两编译脚本 `build_emmc.sh` 以及 `build_spi_nand.sh` 分别用于编译emmc以及spi nand设备。

两个脚本编译指令都相同下面以 `build_emmc.sh` 为例。

4.1 查看编译命令

在根目录执行命令：`./build_emmc.sh -h|help`

```
rk356x$ ./build_emmc.sh -h
=====Start check sdk env =====
Running check_env succeeded.
processing option: --help
Usage: build.sh [OPTIONS]
uboot          -build uboot
kernel         -build kernel
rootfs         -build default rootfs, currently build buildroot as default
all            -build uboot, kernel, rootfs image
cleanall       -clean uboot, kernel, rootfs
update         -pack update image
env            -check sdk env

Default option is 'all'.
```

4.2 自动编译

进入工程根目录执行以下命令自动完成所有的编译：

```
./build_emmc.sh all # 编译模块代码(u-Boot, kernel, Rootfs), 并进行固件打包

./build_emmc.sh #同上
```

4.3 各模块编译及打包

4.3.1 U-Boot编译

```
### U-Boot编译命令
./build_emmc.sh uboot

### U-Boot配置参数
emmc配置
# 默认的编译配置，不需要修改
export RK_UBOOT_DEFCONFIG=rk3568
# 默认即可不修改
export RK_UBOOT_FORMAT_TYPE=fit

spi nand配置
# 根据硬件是否带pmic选择配置：rk3568-spi-nand/rk3568-spi-nand-pmic
export RK_UBOOT_DEFCONFIG=rk3568-spi-nand
# 默认即可不修改
export RK_UBOOT_FORMAT_TYPE=no-fit
# 单个uboot大小，最终uboot.img大小 = 单个uboot * count
export RK_UBOOT_TOTAL_SIZE=1024
# uboot.img包含count个uboot，默认为2个用于备份。
export RK_UBOOT_BACKUP_COUNT=2
# 单个trust大小，最终trust.img大小 = 单个trust * count
export RK_TRUST_TOTAL_SIZE=1024
# trust.img包含count个trust，默认为2个用于备份。
export RK_TRUST_BACKUP_COUNT=2
```

4.3.2 Kernel编译

```
### Kernel编译命令
./build_emmc.sh kernel

### kernel配置参数
emmc配置
# kernel默认的配置
export RK_KERNEL_DEFCONFIG=rockchip_linux_defconfig
# kernel nvr产品形态配置
export RK_KERNEL_DEFCONFIG_FRAGMENT=rk3568_nvr.config
# kernel的dts根据具体版型修改：rk3568-nvr-demo-v10-linux rk3568-evb1-ddr4-v10-linux
rk3568-nvr-demo-v12-linux
export RK_KERNEL_DTS=rk3568-nvr-demo-v12-linux
# 默认即可不需要修改
export RK_BOOT_IMG=zboot.img

spi nand配置
# kernel默认的配置
export RK_KERNEL_DEFCONFIG=rockchip_linux_defconfig
# kernel nvr产品形态配置
export RK_KERNEL_DEFCONFIG_FRAGMENT=rk3568_nvr.config
# kernel的dts根据具体版型修改：rk3568-nvr-demo-v10-linux rk3568-evb1-ddr4-v10-
linux rk3568-nvr-demo-v12-linux
export RK_KERNEL_DTS=rk3568-nvr-demo-v12-linux-spi-nand
```

```
# 默认即可不需要修改
export RK_BOOT_IMG=zboot.img
```

4.3.3 Rootfs编译

执行后会把 `build/rootfs/` 目录打包成特定格式的img固件，格式为根目录下build.sh的配置

`RK_ROOTFS_TYPE`

```
### Rootfs编译命令
./build_emmc.sh rootfs
emmc配置
# parameter分区表，增删分区可以修改这个文件，build/parameter-nvr-emmc.txt
export RK_PARAMETER=parameter-nvr-emmc.txt
# rootfs格式，默认支持ext4 squashfs ubi
export RK_ROOTFS_TYPE=ext4
# 默认即可不需要修改
export RK_ROOTFS_IMG=rootfs.${RK_ROOTFS_TYPE}
# update打包文件，增删分区后需要修改这个问题以便打包正确的update.img。
tools/linux/Linux_Pack_Firmware/rockdevrk356x-package-file-nvr-emmc.txt
export RK_PACKAGE_FILE=rk356x-package-file-nvr-emmc

spi nand配置
# parameter分区表，增删分区可以修改这个文件，build/parameter-nvr-spinand.txt
export RK_PARAMETER=parameter-nvr-spinand.txt
# rootfs格式，默认支持ext4 squashfs ubi，spi nand只支持squashfs ubi
export RK_ROOTFS_TYPE=ubi
# 默认即可不需要修改
export RK_ROOTFS_IMG=rootfs.${RK_ROOTFS_TYPE}
# update打包文件，增删分区后需要修改这个问题以便打包正确的update.img。
tools/linux/Linux_Pack_Firmware/rockdevrk356x-package-file-nvr-spi-nand.txt
export RK_PACKAGE_FILE=rk356x-package-file-nvr-spi-nand
```

客户可以自行在 `build/rootfs/` 增删根文件系统内容。

4.3.4 固件打包

上面 Kernel/U-Boot/Rootfs 各个部分的编译后，进入工程目录根目录执行以下命令自动完成所有固件打包到 rockdev 目录下并且生成编译时间、XML、补丁和固件到IMAGE目录：

```
### 固件打包命令
./build_emmc.sh update
```

4.4 应用编译

目前的编译方式只支持CMake脚本，其它编译系统可以参考build/app/build/build.sh脚本配置编译工具链。

这里以我们发布的RKMPI为例

1. 使能SDK环境


```
在根目录执行 ./build_emmc.sh env
```

2. 编译

```
cd build/app/build
./build.sh ../RKMPI_Release/
```

编译生成的bin文件会在 `build/app/bin` 目录下。

编译出来的bin文件可以通过以下两种方式放到板端运行：

1. 可以放到 `build/rootfs/usr/bin` 目录下然后执行 `./build_emmc.sh rootfs` 重新生成 `rootfs.img` 然后烧写。
2. 板端挂载nfs设备 `mount -t nfs -o nolock 169.254.210.33:/opt/rootfs /mnt/nfs`

拉取最新代码编译报错可以执行 `./build.sh cleanall` 把之前的编译缓存清除再编译。

5. RKMPI媒体包

RKMPI是Rockchip 多媒体处理平台接口，在 `build/app/RKMPI_Release` 目录下有带一份发布的RKMPI包

相关使用文档参考： `build/app/RKMPI_Release/doc/`

媒体FAQ文档： `build/app/RKMPI_Release/doc/Rockchip_FAQ_MPI_CN.pdf`

6. 固件烧写

烧写文档请参考 `docs/RK356x/Rockchip_RK356X_Linux_SDK_*_V*_*_CN.pdf`

7. SecureBoot功能

安全启动功能文档请参考

`docs/Linux/Security/Rockchip_Developer_Guide_Linux_Secure_Boot_CN.pdf`

8. FAQ

Q: 把资源放到 `build/rootfs/usr/bin` 目录下编译rootfs后烧写提示rootfs分区太小。

A: 需要修改parameter分区表，文件存放在build目录下。

emmc: `parameter-nvr-emmc.txt`

spi-nand: `parameter-nvr-spinand.txt`

分区大小算法：比如rootfs要配置200M，则 $200M * 2048 = 0x64000$

0x00064000	0x0000a800	rootfs
分区大小	分区起始地址	分区名字

Q: rootfs设置为ext4格式下，烧写成功后根目录剩余空间太小。

A: 因为rootfs是根据build/rootfs目录大小打包的没有剩余太多空间，可以按照如下修改生成新的img大小，总大小不能超过parameter配置的分区大小。

```
+++ b/tools/build_emmc.sh
@@ -154,7 +154,7 @@ function build_rootfs() {
    SRC_DIR=rootfs/
    tar cf $TEMP $SRC_DIR &>/dev/null
    SIZE=$(du -m $TEMP|grep -o "[0-9]*")
-   let SIZE+=10 #fix out of space
+   let SIZE+=50 #fix out of space
    rm -rf $TEMP
    echo "Making $SRC_DIR (auto size:${SIZE}M)"
    rm -rf rootfs.img rootfs.ext4
```

Q: 如何查看NPU/GPU/CPU/DDR/VDEC频率使用率等。

A:

查看NPU频率：cat /sys/kernel/debug/clk/clk_scmi_npu/clk_rate

查看GPU频率：cat /sys/kernel/debug/clk/clk_scmi_gpu/clk_rate 或者
cat /sys/devices/platform/fde60000.gpu/devfreq/fde60000.gpu/cur_freq

查看GPU负载：cat /sys/devices/platform/fde60000.gpu/utilisation

查看CPU频率：cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq

查看CPU可用的频率表：cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies

使能CPU性能模式，跑最高频：echo performance >

/sys/devices/system/cpu/cpufreq/policy0/scaling_governor

查看DDR频率：cat /sys/kernel/debug/clk/clk_scmi_ddr/clk_rate

查看VDEC频率：cat /sys/kernel/debug/clk/clk_rkvdec_core/clk_rate

Q: UBOOT会默认加载boot分区，如何在parameter中需要把boot分区改成其他名字

A: 修改步骤：

1. parameter中将boot分区修改成需要的名字
2. uboot下对应修改 u-boot/include/boot_rkimg.h

```
#define PART_BOOT "boot"
#define ANDROID_PARTITION_BOOT "boot"
```

Q: kernel中是如何识别加载跟文件系统的。

A: kernle会通过以下两种方式识别跟文件系统：

1. 在bootargs中指定跟文件系统。参考ubi的做法：

```
bootargs = "earlycon=uart8250,mmio32,0xfe660000 console=ttyFIQ0 ubi.mtd=4
root=ubi0:rootfs rootfstype=ubifs";
```

2. RK烧写（window，linux）工具通过识别parameter中的uuid:rootfs，会对rootfs分区打上uuid，kernel的bootargs中默认配置通过uuid来查找rootfs

```
bootargs = "earlycon=uart8250,mmio32,0xfe660000 console=ttyFIQ0
root=PARTUUID=614e0000-0000 rw rootwait";
```

Q: 如何修改rootfs为initramfs。

A: 步骤如下:

1. 在menuconfig中配置Initial RAM filesystem and RAM disk(initramfs/initrd) support

配置成根文件系统的文件夹或者cpio的路径。路径可以不用填写，只要将编译出来的initramfs.cpio.gz拷贝到kernel路径下usr目录下即可

```
make ARCH=arm64 menuconfig
General setup --->
[*] Initial RAM filesystem and RAM disk (initramfs/initrd) support
```

2. 修改 kernel/arch/arm64/boot/dts/rockchip/rk3568-linux.dtsi 中的 bootargs =

```
"earlycon=uart8250,mmio32,0xfe660000 swiotlb=1 console=ttyFIQ0
root=PARTUUID=614e0000-0000 rw rootwait";
```

把 root=PARTUUID=614e0000-0000 rw rootwait 修改为 root=/dev/ram rw rdinit=/linuxrc

3. 如果是采用压缩格式，那么默认解压后的boot不能超过56M，如果超过了会出现uboot解压失败。原因是解压地址超过限制了，把压缩地址覆盖了。

需要做如下修改，可以扩大到98M:

```
ubuntu@srv:/home1/rk3568_linux_sdk/u-boot$ git diff
diff --git a/include/configs/rk3568_common.h b/include/configs/rk3568_common.h
index c869b4e3ce..3c3e14ec09 100644
--- a/include/configs/rk3568_common.h
+++ b/include/configs/rk3568_common.h
@@ -77,8 +77,8 @@
     "fdt_addr_r=0x0a100000\0" \
     "kernel_addr_no_b132_r=0x00280000\0" \
     "kernel_addr_r=0x00a80000\0" \
-    "kernel_addr_c=0x04080000\0" \
+    "ramdisk_addr_r=0x0a200000\0"
+    "kernel_addr_c=0x0a280000\0" \
+    "ramdisk_addr_r=0x04000000\0"

#include <config_distro_bootcmd.h>
```

4. 注意initramfs不会去挂载devtmpfs，导致跟文件系统/dev下没有很多设备节点

需要手动执行一次 `/bin/mount -t devtmpfs devtmpfs /dev`
或者如下修改:

```
diff --git a/board/rockchip/common/base/etc/inittab
b/board/rockchip/common/base/etc/inittab
index 491145cdf0..0c236a9986 100644
--- a/board/rockchip/common/base/etc/inittab
+++ b/board/rockchip/common/base/etc/inittab
@@ -15,6 +15,7 @@

::sysinit:/bin/mount -t proc proc /proc
+::sysinit:/bin/mount -t devtmpfs dev /dev
::sysinit:/bin/mount -o remount,rw /
::sysinit:/bin/mkdir -p /dev/pts
::sysinit:/bin/mkdir -p /dev/shm
```

Q: 分区配置文件parameter中vnvm分区是什么

A: vnvm分区是预留用来保存ETH MAC地址，机器序列号等信息。

如果没有配置vmvm分区，uboot中会用来保存ETH的MAC地址等，需要另外保存。例如可以保存在ENV中：`setenv -f ethmac 00:11:22:33:44:55`

可以通过vendor_storage这个工具来读写，也可以通过PC工具 tools\windows\RKDevInfoWriteTool_1.2.6 来读写。

```
There are 16 types
. "VENDOR_SN_ID"
. "VENDOR_WIFI_MAC_ID"
. "VENDOR_LAN_MAC_ID"
. "VENDOR_BT_MAC_ID"
. "VENDOR_HDCP_14_HDMI_ID"
. "VENDOR_HDCP_14_DP_ID"
. "VENDOR_HDCP_2x_ID"
. "VENDOR_DRM_KEY_ID"
. "VENDOR_PLAYREADY_Cert_ID"
. "VENDOR_ATTENTION_KEY_ID"
. "VENDOR_PLAYREADY_ROOT_KEY_0_ID"
. "VENDOR_PLAYREADY_ROOT_KEY_1_ID"
. "VENDOR_SENSOR_CALIBRATION_ID"
. "VENODR_RESERVE_ID_14"
. "VENDOR_IMEI_ID"
. "VENDOR_CUSTOM_ID"
. And custom can define other id like
. VENDOR_CUSTOM_ID_1A (define ID = 26)
```

Q: 如何抓取火焰图分析

A: 抓火焰图方法：

1. 抓perf数据,机器中执行：

```
perf record -a -g -e cpu-cycles -p 643 -o data/perf.data
```

2. 转火焰图,机器中执行：

```
perf script --symfs=/ -i perf.data > perf.unfold
```

3. 把 perf.unfold 导出来放到PC上FlameGraph目录下,PC上执行：

```
./stackcollapse-perf.pl perf.unfold &> perf.folded
./flamegraph.pl perf.folded > perf.svg
```

Q: RK3568从机如何支持ramboot

A: 按如下步骤执行:

1. uboot的配置改成rk3568-ramboot.config，重新编译uboot，得到u-boot/uboot.img 和u-boot/rk356x_ramboot_loader_v1.09.108.bin，u-boot/trust.img

```
if use bl32 in rkbin/RKTRUST/RK3568TRUST.ini,should open config
CONFIG_OPTEE_CLIENT=y in rk3568-ramboot.config
./make.sh rk3568-ramboot --sz-uboot 2048 1 --sz-trust 1024 1

or if not use bl32 in rkbin/RKTRUST/RK3568TRUST.ini, use default rk3568-
ramboot.config
./make.sh rk3568-ramboot --sz-uboot 2048 1 --sz-trust 512 1
also should change [BL32_OPTION] SEC=1 to SEC=0 in
rkbin/RKTRUST/RK3568TRUST.ini, otherwise will occur fails when building
```

2. 主机接usbhost，从机接usb otg，从机进入loader模式，把从机的固件放到主机中然后运行如下命令:

```
./upgrade_tool/upgrade_tool_ramboot rd 3 //如果设备已经在maskrom模式下，则不需要
运行这条命令
./upgrade_tool/upgrade_tool_ramboot ul rk356x_ramboot_loader_v1.09.108.bin
./upgrade_tool/upgrade_tool_ramboot wl 0x2000 uboot.img
./upgrade_tool/upgrade_tool_ramboot wl 0x42000 trust.img
//./upgrade_tool/upgrade_tool_ramboot wl 0x80000 boot.img //如果boot.img不是从主
机下发，则不需要执行这条命令
./upgrade_tool/upgrade_tool_ramboot run 0x2000 0x42000 0x80000 uboot.img
trust.img boot.img
```

Q: 机器没有烧写boot.img，uboot不能正常识别网卡

A: uboot中需要依赖boot.img的dtb，如果机器中没有boot.img,那么需要在uboot打包dtb。

1. 将kernel的dtb重命名成kern.dtb，放到uboot/dts目录下，然后重新编译uboot即可
编译后uboot.img里面将会打包dtb，不再需要依赖boot里面的dtb。

Q: 整机没有预留烧写按键，如何进入烧写模式

A: RK平台一共有两种烧写模式：Maskrom模式、Loader模式(U-Boot)。

1. 进入Loader烧写模式的方法:

- 1.1 开机时，机器长按音量+
- 1.2 开机时，pc串口中长按ctrl+d组合键
- 1.3 U-Boot命令行输入：download 或者 rockusb 0 \$devtype \$devnum

2. 进入Maskrom烧写模式的方法:

- 2.1 开机时，pc串口中长按ctrl+b组合键
- 2.2 U-Boot命令行输入：rbrom

Q: uboot中如何使把ENV保存到flash中

A: uboot中使能ENV分区，默认ENV是存放在内存（CONFIG_ENV_IS_NOWHERE）中的。

1. uboot的config中打开CONFIG_ENV_IS_IN_BLK_DEV 配置
2. parameter中新增ENV要存放的分区，例如 0x00000800@0x00001800(env)
3. 根据配置的env分区的大小，修改uboot配置文件中的CONFIG_ENV_OFFSET以及CONFIG_ENV_SIZE。例如(从3M开始，大小1M): CONFIG_ENV_OFFSET=0x300000, CONFIG_ENV_SIZE=0x100000
4. uboot中调用env_save()保存，或者用setenv -f xxx xxx 写入。
5. kernel中读取env分区的数据，可以使用u-boot/tools/env下的工具fw_printenv 来读取。

编译fw_printenv ./make.sh env

注意需要根据实际的env分区的位置和大小来配置fw_env.config中的MTD device name, Device offset, Env. size 这些参数。

u-boot/tools/env/fw_printenv // env读写工具

u-boot/tools/env/fw_env.config // env配置文件

u-boot/tools/env/README // env读写工具说明文档

Q: 如何修改串口波特率

A: SDK默认设置的波特率是1.5M，如果需要修改按照如下步骤,修改工具在rkbin/tools目录下:

1. 确认打包bin。打包的脚本是在uboot的config中指定的，如果没有指定默认是rkbin/RKBOOT/RK3568MINIALL.ini，看一下打包的是哪一个ddr bin。
2. 拷贝打包bin。将打包对应的ddr bin拷贝到rkbin/tools目录下
3. 修改参数。只修改需要修改的参数，其他参数不要去改。
如果修改串口波特率的，修改ddrbin_param.txt中的uart baudrate=115200
如果要修改ddr的频率，修改对应ddr类型的频率，例如ddr4: ddr4_freq= 1333
4. 执行命令修改bin。执行 ./ddrbin_tool ddrbin_param.txt DDR_BIN_NAME.bin (DDR_BIN_NAME就是第三步拷贝过来的bin)
5. 拷贝覆盖原来bin。然后将修改后的ddr bin拷贝会rkbin/bin/rk35目录下。
6. 重新编译。重新编译uboot，生成新的loader
7. kernel在dts中修改需要的波特率

参考:kernel/arch/arm64/boot/dts/rockchip/rk3568-linux.dtsi:18: rockchip,baudrate = <1500000>; /* Only 115200 and 1500000 */

Q: 如何修改gmac和linux下eth的对应关系

A: 在RK3568上默认gmac0 对应eth1，gmac1对应eth0。因为SDK代码默认需要过upstream的，要求dtsi中枚举顺序要根据寄存器地址排序

gmac1: ethernet@fe01000 对应eth0

gmac0: ethernet@fe2a0000对应eth1

如果需要改动顺序，需要修改rk3568.dtsi中gmac0和gmac1的定义顺序，即把gmac0放到gmac1前面。

Q: uboot下如何启动boot

A: uboot下可以通过如下方法加载系统

1. 识别u盘/emmc等后，加载其中的固件

fatload 加载u盘/emmc中的kernel后，再用bootm命令启动。

fatload [dev\[:part\]](#)

interface: 所用到接口, 如: MMC、USB

dev [:part]: 文件存放的设备 如: ide 0:1

addr: 装载到内存的开始地址。

filename: 装载的文件名称。

bytes: copy的字节数。

2. 从ftp下载boot, 具体可以参考

docs/Common/UBOOT/Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf:

```
dhcp 0x20000000 172.16.21.161:boot.img
bootm 0x20000000
```

Q: 如何查看HDMI输出参数

A: cat /sys/kernel/debug/dw-hdmi/status

Q: 为什么编译kernel的时候会弹出确认框

A: 因为目前的SDK需要确认dts中配置的io电源域是否以硬件板一致, 所以在第一次成功编译kernel之前会弹框确认。

另外, 为了避免客户直接include参考dtsi文件到实际项目, 引起实际项目硬件电压和软件dts配置电压域不匹配(主要是实际硬件高压软件配置低压的问题), SDK所有参考dtsi默认电压域都设置成3.3V。下载SDK之后, 如果需要编译RK原厂EVB固件, VCCIO4和VCCIO6需要配置成1V8, 其余的配置成3V3, 需改前需要和硬件工程师确认是否跟硬件实际电压相符合。

客户自己的板子, 请和硬件工程师确认, 需要根据实际的硬件配置, 电压域配置十分重要, 如果不匹配会产生严重后果。

RK NVR SDK板(V10、V12版本)需要做如下修改:

```
diff --git a/arch/arm64/boot/dts/rockchip/rk3568-nvr.dtsi
b/arch/arm64/boot/dts/rockchip/rk3568-nvr.dtsi
index 7d72e714a61d..58d2e531ca6a 100644
--- a/arch/arm64/boot/dts/rockchip/rk3568-nvr.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3568-nvr.dtsi
@@ -332,9 +332,9 @@
     pmuio2-supply = <&vcc_3v3>;
     vccio1-supply = <&vcc_3v3>;
     vccio3-supply = <&vcc_3v3>;
-    vccio4-supply = <&vcc_3v3>;
+    vccio4-supply = <&vcc_1v8>;
     vccio5-supply = <&vcc_3v3>;
-    vccio6-supply = <&vcc_3v3>;
+    vccio6-supply = <&vcc_1v8>;
     vccio7-supply = <&vcc_3v3>;
};
```

EVB SDK板需要做如下修改:

```
diff --git a/arch/arm64/boot/dts/rockchip/rk3568-evb.dtsi
b/arch/arm64/boot/dts/rockchip/rk3568-evb.dtsi
index 1bfaeb681069..21680d999834 100644
--- a/arch/arm64/boot/dts/rockchip/rk3568-evb.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3568-evb.dtsi
@@ -1544,9 +1544,9 @@
```

```

pmuio2-supply = <&vcc3v3_pmu>;
vccio1-supply = <&vccio_acodec>;
vccio3-supply = <&vccio_sd>;
-   vccio4-supply = <&vcc_3v3>;
+   vccio4-supply = <&vcc_1v8>;
vccio5-supply = <&vcc_3v3>;
-   vccio6-supply = <&vcc_3v3>;
+   vccio6-supply = <&vcc_1v8>;
vccio7-supply = <&vcc_3v3>;
};

```

Q: 为什么把SDK板子的emmc换成了spi nand之后，烧写固件后一直进maskrom模式

A: 需要做如下排查：

1. 需要确认parameter中定义的分區，不能超过spi nand的容量。

例如128MB的 block size为128K的spi nand，尾部需要预留5个block做坏块处理等，容量最大不能超过128M-128K*5约为127M。

如果定义的分區超过了实际容量，会出现进maskrom的情况。

2. 如果最后一个带grow的分區，如果是ubifs格式的，那么需要用2.89以后的RKDevTool。之前的版本对这种情况支持有问题。

Q: 如何做到NVR和VGA常输出

A: 目前NVR的SDK，HDMI和VGA是强制输出方案，同时应用还是能检测到HDMI热拔插事件，可以获取到HDMI的状态。

1. 默认配置，uboot，kernel阶段HDMI & VGA强制输出，默认输出分辨率为1024x768。

详细配置在arch/arm64/boot/dts/rockchip/rk3568-nvr.dtsi的&route_hdmi 以及&route_edp 节点中。

2. 应用通过drm接口可以获取到HDMI接口的状态。上层应有可以通过RK_S32

RK_MPI_VO_RegCallbackFunc(RK_U32 enIntfType, RK_U32 u32Id, RK_VO_CALLBACK_FUNC_S *pstCallbackFunc)这个接口注册回调。在接口状态变化的时候，通知该回调函数。

3. 应用起来后，使能HDMI&VGA对应的VoDev之后对应的设备就是常输出的，建议（HDMI用RK356X_VO_DEV_HD0，VGA用RK356X_VO_DEV_HD1）。

Q: RK3568是否支持双屏同显以及双屏异显切换

A: RK3568支持同显也支持异显。

1. 同显的模式下，HDMI&VGA只能输出同一种分辨率。
2. 异显的模式下，建议HDMI最大支持4kP30，VGA最大支持1080P30。
3. 同显/异显切换，是通过应用层去操作VoDev来实现的。

同显：dts中把edp_in_vp0使能起来；应用使能RK356X_VO_DEV_HD0，VoPubAttr.enIntfType = VO_INTF_HDMI | VO_INTF_EDP；

异显：应用使能两个VoDev

Q: ubifs文件系统空间优化，以及制作方法

A: NVR SDK中有针对ubifs进行坏块管理等优化，可以放心使用。

1. spi nand这种裸存储介质，分區3M及以上的都推荐ubifs。

在build_spi_nand.sh中默认支持将rootfs以及data配置成ubifs，需要配置如下：


```
export RK_ROOTFS_TYPE=ubi
export RK_USERDATA_TYPE=ubi
```

其他分区支持成ubifs可以参考build/tools/mk-image.sh中mk_ubi_image() 或者参考文档

[docs/Linux/ApplicationNote/Rockchip_Developer_Guide_Linux_Nand_Flash_Open_Source_Solution_CN.pdf](#)

2. ubi block支持squashfs

参考文档

[docs/Linux/ApplicationNote/Rockchip_Developer_Guide_Linux_Nand_Flash_Open_Source_Solution_CN.pdf](#)

3. ubifs 空间优化

参考文档

[docs/Linux/ApplicationNote/Rockchip_Developer_Guide_Linux_Nand_Flash_Open_Source_Solution_CN.pdf](#)

Q: spi nand是否支持烧录器烧录固件

A: 可以支持烧录器烧录。

1. 参考文档

[docs/Linux/ApplicationNote/Rockchip_Developer_Guide_Linux_Nand_Flash_Open_Source_Solution_CN.pdf](#) 中烧录器烧录章节，

用 `./tools/linux/programmer_image_tool/programmer_image_tool` 工具转换出用于烧录器烧录的镜像。

2. 用烧录器提供的工具，将分立的镜像打包成一个烧录的镜像。例如：nsp7500系列

注意：镜像中默认都没有OOB，镜像中也没有ECC，烧录器配置的时候把ONDIE-ECC这个的选项打开。

放 阅读模式 回 放大 缩小 旋转文档 单页 双页 连续阅读 自动滚动 翻译 全文翻译 压缩 截图和对比 朗读 查找

Page0

V1 Adjust(hex):	10	V2 Adjust(hex):	10
Freq Set(hex):	10		
UBA start blk(hex):	0000	Size of UBA blks(hex):	03EC
Boot start blk(hex):	0000	Size of Boot blks(hex):	0400
bad block limit(%):	0002		
Option:	<input checked="" type="checkbox"/> EXCLUDE SPARE AREA 1 不适用范围		
ONDIE-ECC	<input checked="" type="radio"/> Enable 2 使能 ECC		
	<input type="radio"/> Disabled		
Boot Check Assert	<input type="radio"/> Enable		
	<input checked="" type="radio"/> Disable		
	Bad Block Handling Mode		
	<input checked="" type="radio"/> Skip Bad Block 3 跳过坏块		
	<input type="radio"/> Hard Copy		
	<input type="radio"/> Qualcomm Partition		
	<input type="radio"/> MST Partition		
	Mark BB when verification err		
	<input type="radio"/> Enable		
	<input checked="" type="radio"/> Disable		

Q: uboot如何升级boot等分区

A: uboot默认提供了tftpflash工具可以升级除了loader和GPT之外的所有分区。

具体使用方法参考 [docs/Common/UBOOT/Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf](#) 中 5.23.4 章节

也可以用tftp工具下载到ddr后，用mtd write接口写入。

Q: uboot如何升级loader和GPT分区表

A: 需要按照如下步骤升级

1. 把loader和parameter转成mtd可以操作的格式, 假定block size为128K

```
./tools/linux/programmer_image_tool/programmer_image_tool -i update.img -b 128 -p
```

2 -t spinand -o out 把parameter转成gpt.img, Miniloader转成idblock.img。转出来gpt.img大小为128K, idblock.img大小为128K。

注意: programmer_image_tool 不能单独转parameter, 要打包成update.img后一次性转, 单独转loader是可以的。

2. idblock 做双备份, `cat out/idblock.img >> idblock_mutli_copies.img`

3. 写入数据到flash, block的大小根据flash的型号来确认, 正常是128k或者256k。

- 写入gpt.img 到第0个block, 调用blk_dwrite(xx,0, sizeof(gtp.img)/512上取整,gpt数据)。注意: 这里的参数的单位都是512B
- 写入idblock_mutli_copies.img, 到第1-6个block的位置。
- 假设block的大小是128K, 调用blk_dwrite(xx,128k/512, sizeof(idblock_mutli_copies.img)/512上取整, xx数据)

注意: 写入的loader不能超过第七个block的位置。

4. 请注意: 写备份idblock的时候, 不能超过第7个block的地址, idblock仅存放在第1-6 block, 要求结束地址为block 7。 如果flash的block是256K, 需要注意parameter中分配的loader分区大小必需为2M。

Q: 用户态下如何升级loader和GPT分区表

A: 我们不建议用户态下升级loader和GPT, 风险较高。

从0地址开始, 创建一个loader分区, 覆盖之前的保留分区。上层通过写入这个mtd0分区, 来实现更新loader和GPT的功能

具体实现方法和步骤:

1. parameter中增加loader分区, 后面的分区可根据需要依次修改, spi flash的容量比较小, loader分区要控制在1M以内 (1M大小为假定block 128K, 如果block是256K, 控制在2M), parameter中分配loader分区如下 (分配1M大小, 完整文件内容可参阅文件夹下parameter.txt)

CMDLINE: mtdparts=rk29xxnand:0x00000800@0x00000000(loader),0x00000800@0x00000800(vnvm)

2. 把loader和parameter转成mtd可以操作的格式, 假定block size为128K

```
./tools/linux/programmer_image_tool/programmer_image_tool -i update.img -b 128 -p 2 -t spinand -o out
```

把parameter转成gpt.img, Miniloader转成idblock.img。转出来gpt.img大小为128K, idblock.img大小为128K。

注意: programmer_image_tool 不能单独转parameter, 要打包成update.img后一次性转, 单独转loader是可以的。

3. 将生成的gpt.img和idblock.img通过adb push到板子上, 利用mtd_debug工具擦除后再写即可完成对应内容更新;

以下命令均假定flash block为128k, gpt.img和idblock.img位于loader分区属于mtd0, gpt位于第0个block, idblock位于第1-6 block, 总共6个block大小,

所以两者偏移分别为0x0以及0x20000 (该偏移值是相对mtd0的偏移), 如果block大小为256k, 修改对应偏移及大小即可;

```
mtd_debug erase dev/mtd0 0x0 0x20000
mtd_debug write dev/mtd0 0x0 0x20000 userdata/gpt.img
mtd_debug erase dev/mtd0 0x20000 0x60000
mtd_debug write dev/mtd0 0x20000 0x60000 userdata/idblock.img
```

4. idblock这个分区，至少要做双备份，防止写入的时候断电导致没有loader，因此在写入第一份idblock成功后，往其后的地址再写入一份。

请注意：写备份idblock的时候，不能超过第7个block的地址，idblock仅存放在第1-6 block，要求结束地址为block 7。如果block是256K，需要注意parameter中分配的loader分区大小必需为2M。

命令同样假定block 128k情况，block 256k请自行修改对应偏移和大小：

```
mtd_debug erase dev/mtd0 0x80000 0x60000
mtd_debug write dev/mtd0 0x80000 0x60000 userdata/idblock.img
```

Q: 如何裁剪uboot，boot的大小

A: NVR SDK默认已经对uboot和boot的大小进行的优化，uboot单份大小1M，boot大小在5M左右。

1. uboot进一步裁剪思路：

建议裁剪掉不需要的CMD。如果uboot下不需要支持logo显示，可以去掉CONFIG_DRM_ROCKCHIP=y等配置。

2. boot进一步裁剪思路：

裁剪掉不必要的外设驱动，例如：wifi，屏，触摸屏等。

1.kernel hacking下面的一些调试的选项可以去掉能省比较多空间。需要特别注意：去掉这些选项之后，所有的ko都要重新编译，否则会出现开机ko不匹配引起的崩溃等问题。

如下列举的这些可以作为去掉的参考，结合实际项目取舍。

```
Kernel hacking --->
[*] Collect scheduler debugging info
[*] Collect scheduler statistics

Lock Debugging (spinlocks, mutexes, etc...) --->
[*] Spinlock and rw-lock debugging: basic checks

[*] Verbose BUG() reporting (adds 70K)

[*] Debug credential management

[*] Tracers --->

[*] Runtime Testing --->
```

2. 文件系统下的一些没用到的文件系统可以考虑去掉，文件系统占用空间也蛮大

```
File systems --->
```

3. 其他需要根据项目的实际去裁剪没用到的外设

Q: 如何查看pin管脚复用配置情况

A: `cat sys/kernel/debug/pinctrl/pinctrl-rockchip-pinctrl/pinmux-pins`

Q: USB3.0 OTG口如何切换device和host

A: USB3.0 默认是otg模式。

切换成host: `echo host > /sys/devices/platform/fe8a0000.usb2-phy/otg_mode`

切换成device: `echo peripheral > /sys/devices/platform/fe8a0000.usb2-phy/otg_mode`

Q: 如何挂载NFS

A: 依赖: `sbin/mount.nfs`, `sbin/mount.nfs4`, `sbin/umount.nfs`, `sbin/umount.nfs4`, `./usr/lib/libtirpc.so.3`

这些库在NVR SDK提供的rootfs中都可以找到; NVR SDK提供的kernel config配置默认支持NFS功能。

挂载命令:

```
Linux :
mount -t nfs -o nolock 10.12.201.5:/nfs /mnt/nfs 或者 mount -t nfs -o
nolock,nfsvers=3,vers=3 10.12.201.5:/nfs /mnt/nfs

Window:
mount \\10.12.201.15\nfs I:\
```

Q: DDR如何保持固定频率, 关闭DDR变频

A: NVR SDK 默认配置DDR保持固定频率

1. 命令行中设定频率:

```
# 查看DDR频率的可设置范围
cat /sys/class/devfreq/dmc/available_frequencies
# 设置DDR governors为userspace模式
echo userspace > /sys/class/devfreq/dmc/governor
# 设置DDR频率
echo 1560000000 > /sys/class/devfreq/dmc/userspace/set_freq
```

2. 代码中关闭变频: 在对应的dts中关闭dmc, 参考如下

```
--- a/arch/arm64/boot/dts/rockchip/rk3568-evb.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3568-evb.dtsi
@@ -402,7 +402,7 @@
    &dmc {
        center-supply = <&vdd_logic>;
-       status = "okay";
+       status = "disabled";
    };
```

Q: 如何进行GDB调试

A: GDB调试注意事项如下:

1. NVR SDK发布的rootfs中默认带有GDB的支持`build/rootfs/usr/bin/gdb`。

2. `gdb xxx` 或者 `gdb attach pid` 的方式启动gdb调试。具体命令可以参考网络资料。

特别提示抓所有线程堆栈的命令: `thread apply all bt full`

3. GDB忽略信号处理

`handle SIGPIPE nostop noprint`

`handle SIGUSR2 nostop noprint`

`handle SIG32 nostop noprint`

`handle SIG34 nostop noprint`

`set print pretty on`

Q: rootfs是只读的, 如何链接新增的库文件

A: 通过设置环境变量LD_LIBRARY_PATH指定的动态库搜索路径: (可用export LD_LIBRARY_PATH="NEWDIRS" 命令添加临时环境变量)

例如: export LD_LIBRARY_PATH='/usr/local/lib:/usr/lib:/usr/lib64:/usr/lib32'

注意: 通过LD_LIBRARY_PATH 配置的库的路径, 链接时候搜索优先级高于默认的rootfs下的/usr/lib, 所以可以用来临时验证某些库。

Q: I2C 接口i2c_master_send发送大数据 (例如24KB) 失败

A: i2c-rk3x.c里面的timeout时间1s改大到3s

Q: SATA速率配置

A: 修改内核驱动:

```
pipe_con0_for_sata      = { 0x0000, 15, 0, 0x00, 0x0000 }, 1.5G
pipe_con0_for_sata      = { 0x0000, 15, 0, 0x00, 0x1110 }, 3G
pipe_con0_for_sata      = { 0x0000, 15, 0, 0x00, 0x2220 }, 6G
```

```
diff --git a/drivers/phy/rockchip/phy-rockchip-naneng-combphy.c
b/drivers/phy/rockchip/phy-rockchip-naneng-combphy.c
index 08445c1890eb..3df0e0e05ab4
--- a/drivers/phy/rockchip/phy-rockchip-naneng-combphy.c
+++ b/drivers/phy/rockchip/phy-rockchip-naneng-combphy.c
@@ -604,7 +604,7 @@ static const struct rockchip_combphy_grfcfg
rk3568_combphy_grfcfgs = {
    .con2_for_sata      = { 0x0008, 15, 0, 0x00, 0x80c3 },
    .con3_for_sata      = { 0x000c, 15, 0, 0x00, 0x4407 },
    /* pipe-grf */
-   .pipe_con0_for_sata = { 0x0000, 15, 0, 0x00, 0x2220 },
+   .pipe_con0_for_sata = { 0x0000, 15, 0, 0x00, 0x0000 },
    .pipe_sgmiimac_sel  = { 0x0040, 1, 1, 0x00, 0x01 },
    .pipe_xpcs_phy_ready = { 0x0040, 2, 2, 0x00, 0x01 },
    .u3otg0_port_en     = { 0x0104, 15, 0, 0x0181, 0x1100 },
```

Q: 多VP同步, 用于同一个CPU多个输出口的同步

A: 1. RK3568 NVR SDK需要升级到V1.5或以上

2. 多VP同步接口

1) 直接操作内核节点: `echo 1 2 > sys/kernel/debug/dri/0/video_port0/vp_sync` //vp1和vp2同步到vp0

2) MPI接口:

```
#define BIT(x) (1 << x)
RK_U32 timeout = 10;
RK_U32 Devs = BIT(0) | BIT(2); //vp0 vp2 sync
while (timeout--) {
    RK_U32 Ret = RK_MPI_VO_SyncDevs (Devs);
    if (Ret) {
        RK_LOGE("RK_MPI_VO_SyncDevs fail retry, timeout=%d", timeout);
        usleep(100000);
    }
    else {
        RK_LOGE("RK_MPI_VO_SyncDevs succeed");
        break;
    }
}
```

```
}  
}
```

注意事项:

1. 需要在所有需要同步的voDev都使能之后，设置同步模式。目前暂时只支持原生HDMI、DP/eDP、BT656/BT1120接口输出的同步；
2. 如果有设置 RK_MPI_VO_SetVcntTiming，需要在设置vcnt之后再设置同步模式。
如果有设置vcnt，单屏视频不能超过2路，否则可能无法保证同步性。如果单屏多路视频情况下要保证同步性，需要把vcnt配置成0才能保证同步；
3. 如果有设置HDMI属性RK_MPI_VO_SetHdmiParam()，需要在设置属性之后usleep(1000*1000llu);后等vp使能之后再调用设置同步模式接口。
4. 上述方法只保证了底层vp的同步性，应用层需要保证送给每个vo的数据是同步的。为了保证同步性，有以下几点建议：
 - vdec配置成预览模式RK_MPI_VDEC_SetDisplayMode(u32Ch, VIDEO_DISPLAY_MODE_PREVIEW)。
 - 上层建议使用vsync中断来触发送帧给vo，并且确认给vo送数据的时候是在vsync的前半部分（即vsync中断和给vo送数据的间隔，不能超过vsync/2）。
可以使用RK_MPI_VO_RegVsyncCallbackFunc注册vsync中断，也可以使用drm的wait vblank方式获取vsync中断回调。
注意：vsync回调中不能直接做送帧动作，不能做耗时操作。
 - 如果单屏只需要输出一个全屏画面，那么建议设置图层直通模式stLayerAttr.bBypassFrame=RK_TRUE，减少中间环节误差
并且VO CHN配置缓存数量为1，在enable_chn之前用 RK_MPI_VO_SetChnRecvThreshold()这个接口设置。
 - 直通模式下，如果送帧是30fps并且显示是60fps的场景下（即两个vsync送一帧数据），出现频繁不同步的情况下
可以尝试在RK_MPI_VO_SendFrame()之前加3ms的延时后再测试同步性。
 - 如果单屏多路视频拼接情况下要保证同步性，需要把vcnt配置成0才能保证同步。即不要调用RK_MPI_VO_SetVcntTiming()这个接口去配置VCNT，默认是0。
并且在enable_chn之前，调用 RK_MPI_VO_SetChnRecvThreshold()接口将VO CHN配置缓存数量为4。
用RK_MPI_VO_SetLayerDispBufLen()接口将layer的buffer设置成5个。
layer的帧率和解码的帧率也要配置成一样的，例如解码30fps，layer帧率也要配置成30fps。
如果在vo送帧已经同步的情况下出现频繁显示不同步的现象，可以尝试在RK_MPI_VO_SendFrame()之前加3ms的延时后再测试同步性。

同时要注意：送给vo的buffer不要频繁申请释放，建议直接使用vdec解码出来的buffer或者采用bufferpool方式循环使用buffer。

Q: 不同RK3568 cpu之间同步

A: RK3568 NVR SDK需要升级到V1.5或以上

1. 不同cpu之间需要同一时刻使能vo，保证初始化的时候是同步的。
如果多个cpu在同一个板子上，可以通过cpld(或其他硬件同步信号)发送中断给每个CPU，同时使能vo。
如果不在同一个板子上，则需要有同步多个cpu的机制，例如利用网络IEEE1588V2（linuxPTP）精确时钟同步协议等，RK3568支持硬件时间戳，根据我们自测使用硬件时间戳同步后误差可以控制在10us以内。

控制方法：具体实现在rockchip_drm_vop2.c中的vop2_crtc_enable(), 客户可以自行封装内核态下的同步接口

```
disable crtc: echo 0 > /sys/kernel/debug/dri/0/video_portN/enable //这个是阻塞的，直到standby生效了才返回；
enable crtc: echo 1 > /sys/kernel/debug/dri/0/video_portN/enable // 这个非阻塞的，取消standby后马上开始第一行的扫描
```

2. 多3568 cpu因为时钟不同源，跑一段时间后VSYNC可能出现相位差，需要微调vsync。

- 有关vsync同步检测

建议方法：多个cpu各自监控自己的vsync回调时间，计算单位时间内的vsync的时间总和，从机根据主机的vsync时间总和以及从机自身统计的单位时间vsync时间总和的差值，做相应的调整。

用户态获取vsync时间戳接口：通过Sample_VO_RegVsyncCallback 注册vsync回调，可以获取到vsync的时间。

内核态获取vsync时间戳方法：vsync中断处理流程是在void vop2_wb_handler(struct vop2_video_port *vp)

具体位置是在rockchip_drm_vop2.c的irqreturn_t vop2_isr(int irq, void *data)中断处理内，在调用vop2_wb_handler(vp)之前调用ktime_get_real_ts64(struct timespec64 *ts)可以获取到准确的vsync时间戳。

由于用户态获取vsync时间戳需要经过多次回调，容易受系统负载等影响引入误差，建议在内核态下获取vsync时间。

- RK3568调节使用的是系统pll（pll_hp11），使用rockchip_pll_clk_compensation接口去调dclk
cat sys/kernel/debug/clk/clk_summary 可以看到时钟树，查看需要调整的dclk在哪个PLL下面，找到对应的pll
调整完之后，clk_get_rate获取频率就会看到有变化的，或者设置后看时钟树：

```
cat sys/kernel/debug/clk/clk_summary
```

参考代码如下：

```
struct clk *clk = NULL;
int ret = 0, i = 0;

clk = __clk_lookup("hp11");
clk1 = __clk_lookup("pll_hp11");
if (clk == NULL)
    printk("---get hp11 clk fail---\n");
if (clk1 == NULL)
    printk("---get pll_hp11 clk fail---\n");

for (i = 1; i < 20; i++) {
    printk("clk name=%s,clk=%ld, clk1 name=%s,clk=%ld###\n",
        __clk_get_name(clk), clk_get_rate(clk), __clk_get_name(clk1),
        clk_get_rate(clk1));
    ret = rockchip_pll_clk_compensation(clk, i * 50);
    printk("clk name=%s,clk=%ld, clk1 name=%s,clk=%ld###,ret=%d\n",
        __clk_get_name(clk), clk_get_rate(clk), __clk_get_name(clk1),
        clk_get_rate(clk1), ret);
}
```



```

for (i = 1; i < 20; i++) {
    ret = rockchip_pll_clk_compensation(clk, -50 * i);
    printk("clk name=%s,clk=%ld, clk1 name=%s,clk=%ld###,ret=%d\n",
        __clk_get_name(clk), clk_get_rate(clk), __clk_get_name(clk1),
        clk_get_rate(clk1), ret);
}

```

注意事项:

- rockchip_pll_clk_compensation() 的调整只能基于获取到的clk的基准频率进行调整，调整结果不能累加。

例如调用rockchip_pll_clk_compensation(clk, 200)后在调用rockchip_pll_clk_compensation(clk, 200)，结果调节200ppm，而不是基于第一次调整后的值在调整。

rk3568 vp1默认挂在vp1l下，vp1l不支持小数分频，不能进行调整。针对同步拼接的场景（vp0/vp1输出相同的分辨率），建议把vp1也挂载hpll下。

```

--- a/arch/arm64/boot/dts/rockchip/rk3568-nvr.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3568-nvr.dtsi
@@ -519,8 +519,8 @@
    &vop {
        status = "okay";
-       assigned-clocks = <&cru DCLK_VOPl>;
-       assigned-clock-parents = <&cru PLL_VPLL>;
+/*   assigned-clocks = <&cru DCLK_VOPl>;
+   assigned-clock-parents = <&cru PLL_VPLL>; */
        skip-ref-fb;
    };

```

- 调整接口rockchip_pll_clk_compensation()不能在中断中调用，因为其中有所可能会引起系统调度，建议在workqueue中调用。
 - 建议把调整的周期缩短一些（例如：一秒调节一次），每次调整的幅度小一些（例如：每次调整-2~2ppm）。
每个调整周期都需要判断是否需要调整pll参数（以上一个周期的调整参数作为基准做适当修改，如果不需要修改就保持上一次的pll调整参数）。
 - 目前调整dclk的方式暂时只验证了原生HDMI、DP/eDP、BT656/BT1120接口。
3. 上述方法只保证了底层vp的同步性，应用层需要保证送给每个vo的数据是同步的。为了保证同步性，有以下几点建议：
- vdec配置成预览模式RK_MPI_VDEC_SetDisplayMode(u32Ch, VIDEO_DISPLAY_MODE_PREVIEW)。
 - 上层建议使用vsync中断来触发发送帧给vo，并且确认给vo送数据的时候是在vsync的前半部分（即vsync中断和给vo送数据的间隔，不能超过vsync/2）。
可以使用RK_MPI_VO_RegVsyncCallbackFunc注册vsync中断，也可以直接使用drm的wait vblank方式获取vsync中断回调。
注意：vsync回调中不能直接做送帧动作，不能做耗时操作。
 - 如果单屏只需要输出一个全屏画面，那么建议设置图层直通模式stLayerAttr.bBypassFrame=RK_TRUE，减少中间环节误差
并且VO CHN配置缓存数量为1，在enable chn之前用 RK_MPI_VO_SetChnRecvThreshold ()这个接口设置。
 - 直通模式下，如果送帧是30fps并且显示是60fps的场景下（即两个vsync送一帧数据），出现频繁不同步的情况下
可以尝试在RK_MPI_VO_SendFrame()之前加3ms的延时后再测试同步性。

- 如果单屏多路视频拼接情况下要保证同步性，需要把vcnt配置成0才能保证同步。即不要调用RK_MPI_VO_SetVcntTiming()这个接口去配置VCNT，默认是0。
并且在enable_chn之前，调用RK_MPI_VO_SetChnRecvThreshold()接口将VO CHN配置缓存数量为4。
用RK_MPI_VO_SetLayerDispBufLen()接口将layer的buffer设置成5个。
layer的帧率和解码的帧率也要配置成一样的，例如解码30fps，layer帧率也要配置成30fps。
如果在vo送帧已经同步的情况下出现频繁显示不同步的现象，可以尝试在RK_MPI_VO_SendFrame()之前加3ms的延时后再测试同步性。
同时要注意：送给vo的buffer不要频繁申请释放，建议直接使用vdec解码出来的buffer或者采用bufferpool方式循环使用buffer。

Q: 调试VI-VENC-VDEC-VO 通路延时

A: RK3568 NVR SDK需要升级到V1.5或以上

1. 几个优化显示通路延时的建议：

- vdec配置成预览模式RK_MPI_VDEC_SetDisplayMode(u32Ch, VIDEO_DISPLAY_MODE_PREVIEW)并且VO CHN配置缓存数量为1，RK_MPI_VO_SetChnRecvThreshold()这个接口设置，在enable_chn之前配置。注意：此时应用需要保证送帧的均匀性。
- layer帧率配置成60，目的是减少VSYNC间隔，但是要注意GPU使用率。
- 如果输入源是VI，可以考虑VI源提高mipi传输频率，目的是缩短传输时间。
- 如果有编码，VENC 改成60FPS，目的是减少延时波动。
- CPU/GPU跑性能模式，要注意下功耗和散热。
- 使能直通模式，在enable_layer之前配置，stLayerAttr.bBypassFrame = RK_TRUE 当layer只有一个通道的时候可以直通显示，减少中间拼接流程。
注意：如果ui和视频是在同一个图层，也无法进入直通模式，需要把ui通道disable后才能进入直通模式。如果进入直通模式，GPU负载应该为很低，大部分情况为0。
- 在RK_MPI_VO_Enable之前，调用RK_MPI_VO_SetVcntTiming(VoDev, 900); //1080P建议是900，其他分辨率按显示的高度*0.8预估，需要调整测试到一个最优值。
注意：分辨率改变的时候，对应的VCNT也需要跟着修改，否则会出现无法显示的问题。
- VDEC解码配置为解码序，stVdecParam.stVdecVideoParam.enOutputOrder = VIDEO_OUTPUT_ORDER_DEC
- 编解码采用多slice机制，目前还未支持。

Q: 规范操作进入 maskrom 模式

A: 1). 有 maskrom 按键，先按住 reset 按键，再按住 maskrom 按键，松 reset（会看到进入工具显示 maskrom），再松 maskrom（次序很重要，避免异常）；

2). 无 maskrom，maskrom 按键用“flash 数据线 io0 短接地代替”

没有uboot或者uboot跑飞的情况下，不能进入loader模式，只能进入maskrom模式。

Q: kernel默认采用LZMA压缩格式，如果采用initram方式打包kernel的话，会出现uboot在bootm的时候失败

A: 需要改成LZ4压缩

```
diff --git a/arch/arm64/Makefile b/arch/arm64/Makefile
index a3a8e47..5b39088 100644
--- a/arch/arm64/Makefile
+++ b/arch/arm64/Makefile
```

```
@@ -189,7 +189,7 @@ define archhelp
    echo '                                install to $(INSTALL_PATH) and run lilo'
    endif

-kernel.img: Image.lzma
+kernel.img: Image.lz4
    $(Q)scripts/mkkrnlimg ${objtree}/arch/arm64/boot/Image
${objtree}/kernel.img >/dev/null
    @echo ' Image: kernel.img is ready'

ifdef CONFIG_MODULES
diff --git a/scripts/mkimg b/scripts/mkimg
index eac3447..2944a6d 100755
--- a/scripts/mkimg
+++ b/scripts/mkimg
@@ -60,8 +60,8 @@ if [ "${ARCH}" == "arm" ]; then
    ZIMAGE=zImage
else
    DTB_PATH=${objtree}/arch/arm64/boot/dts/rockchip/${DTB}
-    KERNEL_ZIMAGE_ARG="--kernel ${objtree}/arch/arm64/boot/Image.lzma"
-    ZIMAGE=Image.lzma
+    KERNEL_ZIMAGE_ARG="--kernel ${objtree}/arch/arm64/boot/Image.lz4"
+    ZIMAGE=Image.lz4
    fi
    if [ ! -f ${DTB_PATH} ]; then
echo "No dtb" >&2
```