nVidia Orin 开发板101

修改记录

修改日期	作者	改动内容
2022/3/9	乔永昌	初版

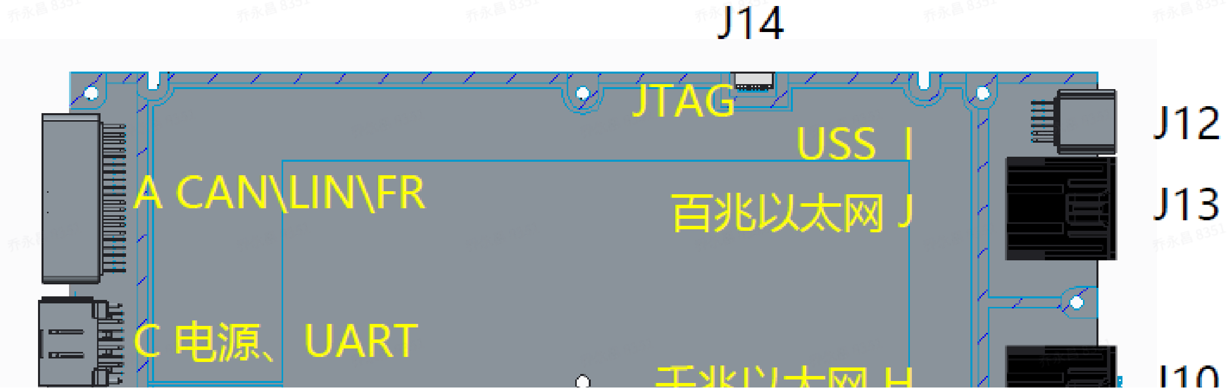
1. Orin环境搭建

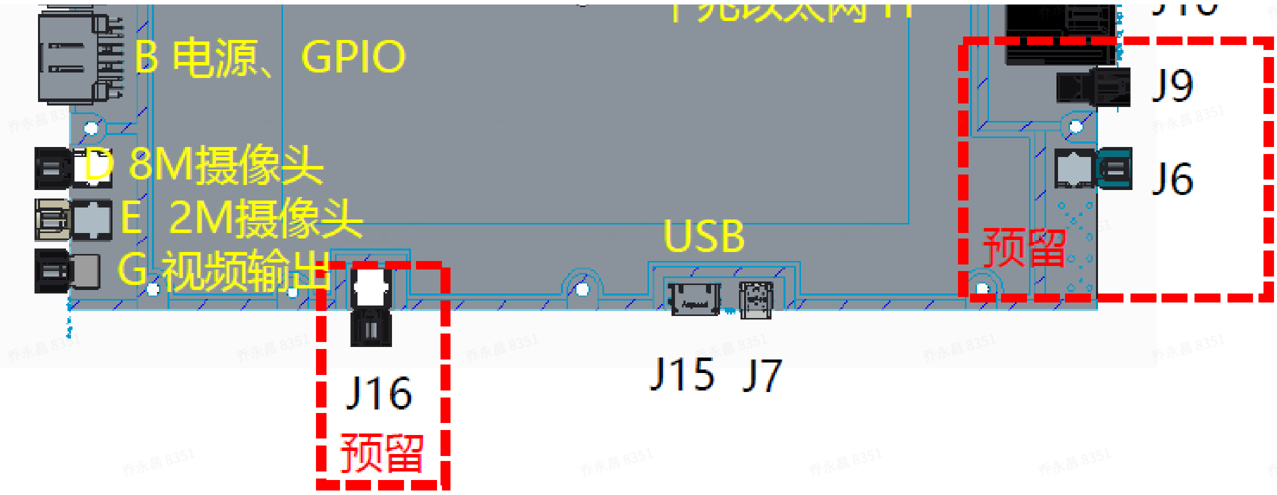
1.1 准备材料

- Orin开发板ADMC或ADMC+ADSC
- 直流电源（12V 40A以上）
- 电源线束
- Type-C数据线（烧写、扩展、USB转网卡）
- USB转RS232线束
- 水冷一套
- RJ45网线
- 千兆及百兆连接线
- 网络转换器

1.2 控制器接口说明及示意图

- 接口说明





开发板连线实物图

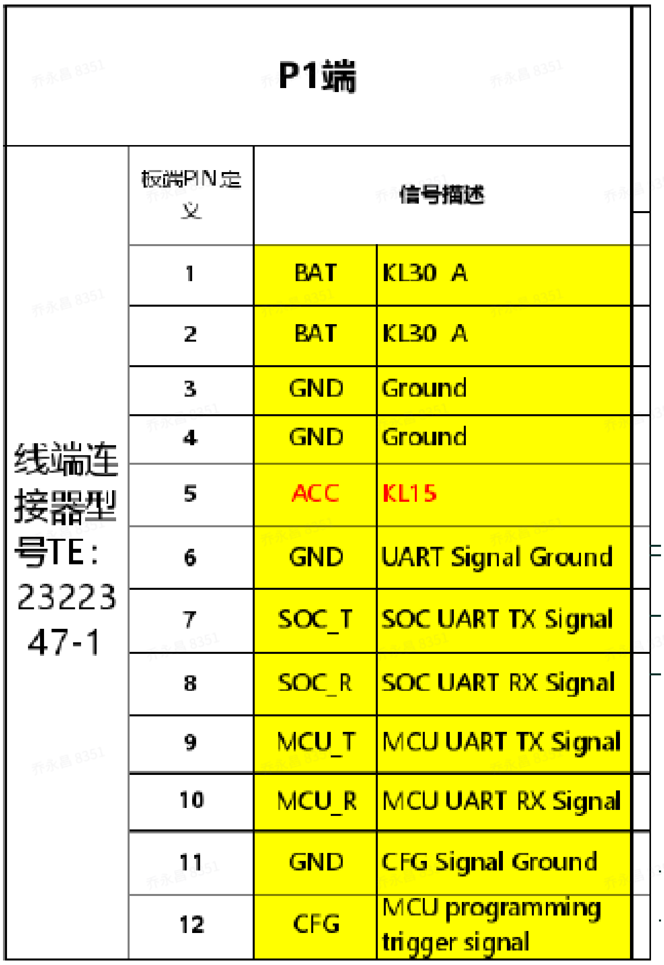
1.3 安装步骤

1.3.1 电源连接

> 电源要求12V 40A以上
>
> 控制器电压9V~16V， 不超过18V

每个控制器有两个12PIN电源线，两条电源的 BATT 及 GND 都需要连接。

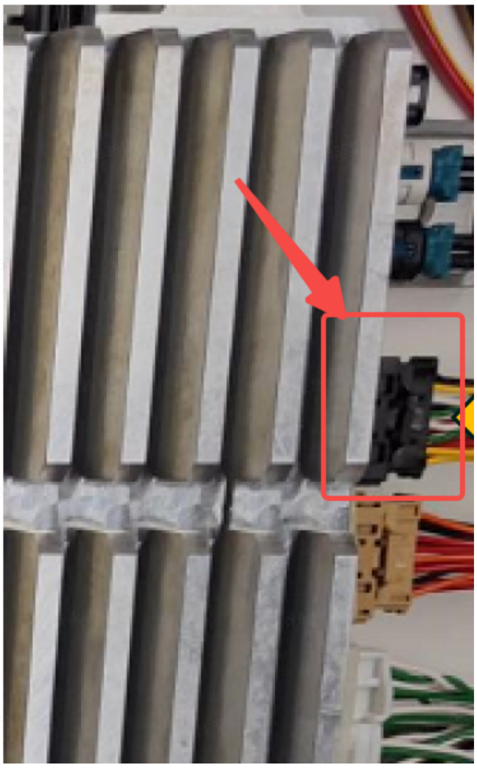
Pin定义如下图



P1端			
	板端PIN定义	信号描述	
线端连接器型号TE：2322347-1	1	BAT	KL30 A
	2	BAT	KL30 A
	3	GND	Ground
	4	GND	Ground
	5	ACC	KL15
	6	GND	UART Signal Ground
	7	SOC_T	SOC UART TX Signal
	8	SOC_R	SOC UART RX Signal
	9	MCU_T	MCU UART TX Signal
	10	MCU_R	MCU UART RX Signal
	11	GND	CFG Signal Ground
	12	CFG	MCU programming trigger signal

1.3.2 UART连接

串口线在电源线束当中，线束位置如下图所示



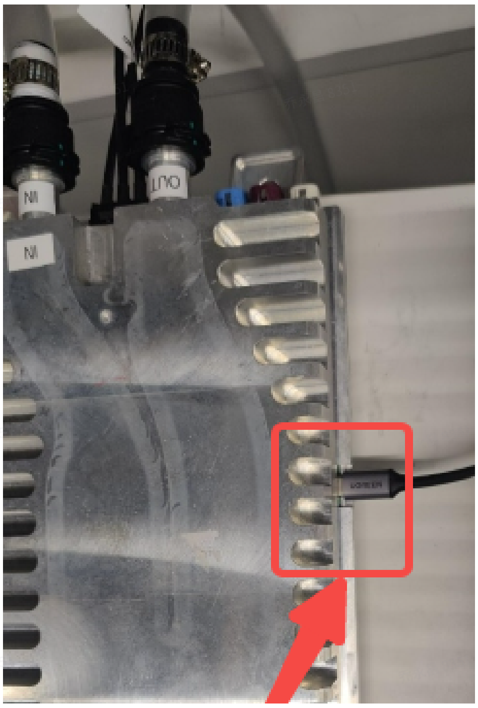
- SoC及MCU的UART连接

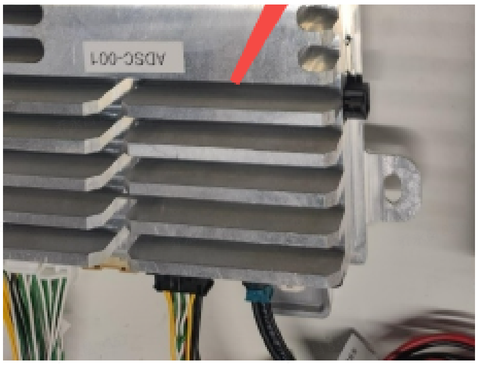
参考电源线的Pin定义，连接到RS232母头，并使用USB转RS232串口连接到PC

波特率115200

1.3.3 烧写口连接

- 接口位置

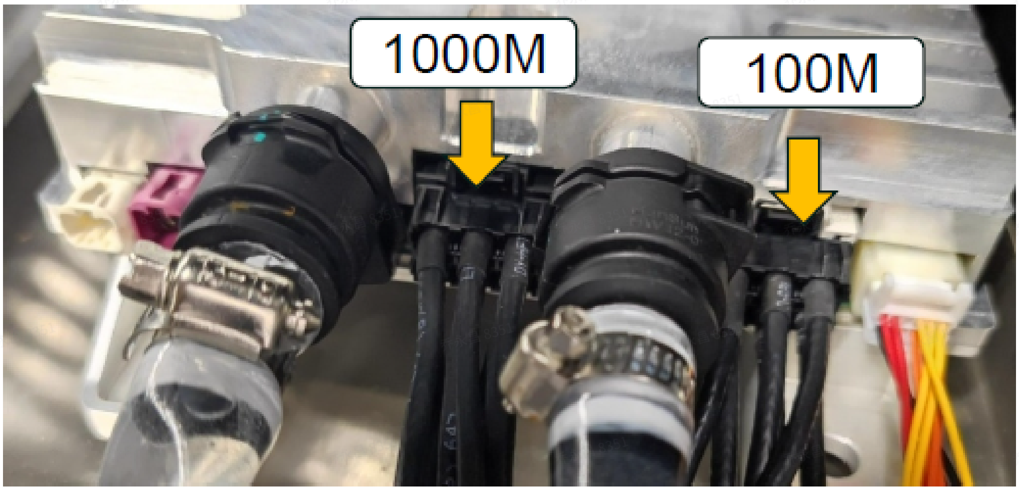




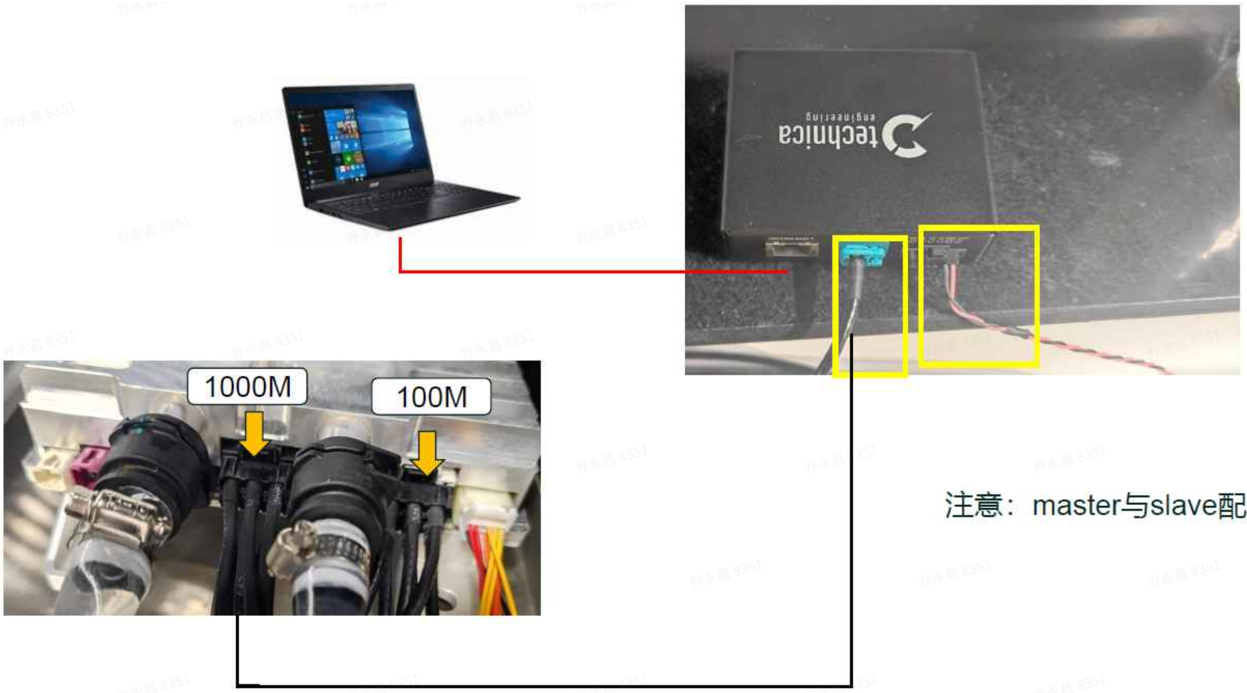
使用Type-C转USB-A数据线连接PC进行SoC烧录

1.3.4 网络连接

- 准备材料
    - 网线
    - 以太网转换器
- 接口位置

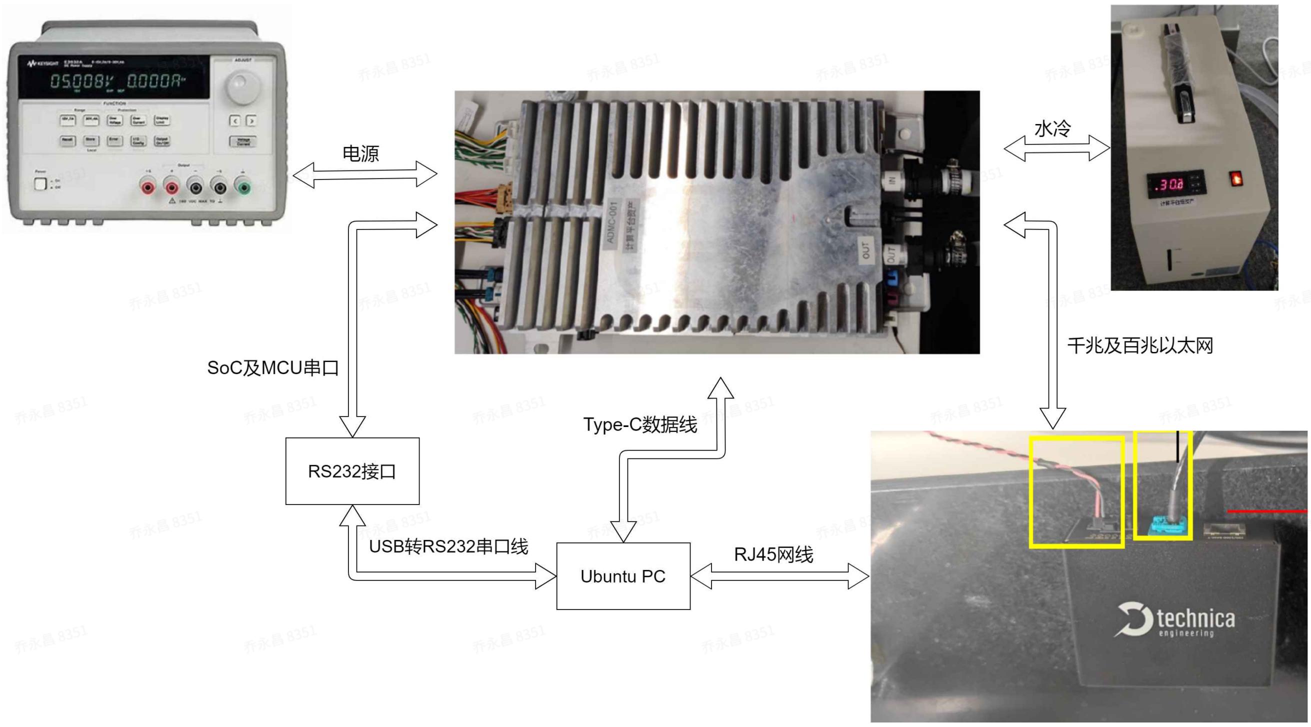


- 连接示意图



注意：master与slave配置。

1.4 连接好示意图



2. SDK获取与安装

2.1 获取

> 请联系相关同事

2.2 安装

> ！注意文件名的版本号可能有变化，请注意修改

Shell

```shell
# 1. remove previously installed version
sudo -E apt-get -y --purge remove nv-driveos*
sudo apt-get -y autoremove

# 2. install the local repo Debian packages
export NV_WORKSPACE=<SDK安装路径>
sudo dpkg -i ./nv-driveos-repo-sdk-linux-6.0.2.0-release-0008-29590679_6.0.2.0_amd64.deb
sudo dpkg -i ./nv-driveos-repo-pdk-linux-6.0.2.0-release-0008-29590679_6.0.2.0_amd64.deb

sudo apt-get update

sudo -E apt -f -y install nv-driveos-build-sdk-linux-6.0.2.0-29629403
sudo -E apt -f -y install nv-driveos-build-pdk-linux-6.0.2.0-29629403


# 3. remove cuda old package
sudo rm /var/lib/apt/lists/_var_cuda*
sudo apt --fix-broken install -y
sudo apt-get autoremove -y
sudo apt-get remove --purge -y "cuda*"
sudo apt-get remove --purge -y "*cublas*"


# 4. install cuda
sudo dpkg -i ./cuda-repo-ubuntu2004-11-4-local_11.4.14-470.88-1_amd64.deb
sudo dpkg -i ./cuda-repo-cross-aarch64-ubuntu2004-11-4-local_11.4.14-1_all.deb
sudo apt-key add /var/cuda-repo-ubuntu2004-11-4-local/7fa2af80.pub
sudo apt update
sudo apt -y install cuda-toolkit-11-4
sudo apt -y install cuda-cross-aarch64-11-4


# 5. install cudnn
sudo apt install ./cudnn-local-repo-ubuntu2004-8.2.6.28_1.0-1_amd64.deb
sudo apt update

sudo apt install libcudnn8
sudo apt install libcudnn8-dev
sudo apt install libcudnn8-samples


sudo dpkg -i ./cudnn-prune-87-repo-cross-aarch64-ubuntu2004-8-2-local_1.0-1_all.deb
sudo apt update
sudo apt install libcudnn8-cross-aarch64

sudo dpkg -i nv-tensorrt-repo-ubuntu2004-cuda11.4-trt8.3.0.10-x86-host-ga-20220116_1-1_amd64.deb
sudo dpkg -i nv-tensorrt-repo-ubuntu2004-cuda11.4-trt8.3.0.10-d6l-cross-ga-20220116_1-1_amd64.deb
sudo apt-key add /var/nv-tensorrt-repo-ubuntu2004-cuda11.4-trt8.3.0.10-d6l-cross-ga-20220116/7fa2af80.pub
sudo apt install tensorrt


# 6. install additional lib for aarch64
#refer to
 https://docs.nvidia.com/deeplearning/tensorrt/sample-support-guide/index.html#c

sudo apt install libnvinfer-dev-cross-aarch64
sudo apt install libnvinfer8-cross-aarch64
sudo apt install libnvinfer-plugin-dev-cross-aarch64
sudo apt install libnvinfer-plugin8-cross-aarch64
sudo apt install libnvparsers-dev-cross-aarch64
sudo apt install libnvparsers8-cross-aarch64
sudo apt install libnvonnxparsers-dev-cross-aarch64
sudo apt install libnvonnxparsers8-cross-aarch6
```

3. 编译

3.1 环境变量

Shell

```shell
SDK_ROOT="<SDK_6.0_ROOT>"
cd ${SDK_ROOT}

export _NV_INSTALL_LICENSE_BYPASS_="Destination Tegra Dominance"
export PDK_TOP=${PWD}
export DRIVE_LINUX=${PDK_TOP}/drive-linux
export DRIVE_FOUNDATION=${PDK_TOP}/drive-foundation
export DRIVE_LINUX_SRC=${PDK_TOP}/drive-linux_src
export NV_TOOLCHAIN=${PDK_TOP}/toolchains
export RFS_ORIGIN=${PDK_TOP}/filesystem/source
export RFS_COMMON=${PDK_TOP}/filesystem/common

# Define bootburn path
export BOARD_NAME="p3663-a01"
export BOARD_PCT="linux"
export CREATE_PYTHON=${DRIVE_FOUNDATION}/tools/flashtools/bootburn/create_bsp_images.py
export FLASH_PYTHON=${DRIVE_FOUNDATION}/tools/flashtools/bootburn/flash_bsp_images.py
export BOOTBURN_PYTHON=${DRIVE_FOUNDATION}/tools/flashtools/bootburn/bootburn.py
export BSP_INDEX="642-63663-0001-001_TS2"
export CUSTOMER_JSON=${DRIVE_FOUNDATION}/tools/flashtools/bootburn/customer_data_orin.json

# Define kernel path
export ARCH=arm64
export CROSS_COMPILE=${NV_TOOLCHAIN}/aarch64--glibc--stable-2020.08-1/bin/aarch64-buildroot-linux-gnu-
export LOCALVERSION="-tegra"
export KERNEL_SOURCE_PATH=${DRIVE_LINUX_SRC}/kernel/drive-linux/kernel/source/oss_src
export KERNEL_OUTPUT=${KERNEL_SOURCE_PATH}/output
export INSTALL_MOD_PATH=${KERNEL_OUTPUT}
```

3.2 Foundation编译

Shell

```shell
cd ${SDK_ROOT}

# Build foundation bpmp dtsi
make -C ${DRIVE_FOUNDATION}/platform-config/bpmp_dtsi/t23x clean
make -C ${DRIVE_FOUNDATION}/platform-config/bpmp_dtsi/t23x

make -C ${DRIVE_FOUNDATION} -f make/Makefile.bind BOARD=${BOARD_NAME} PCT=${BOARD_PCT} clean
make -C ${DRIVE_FOUNDATION} -f make/Makefile.bind BOARD=${BOARD_NAME} PCT=${BOARD_PCT}
${DRIVE_FOUNDATION}/make/bind_partitions -b ${BOARD_NAME} ${BOARD_PCT} -p ufs_boot
```

3.3 Linux Kernel编译

Shell

```
SDK_ROOT="<SDK_6.0_ROOT>"

# change to kernel source path
cd ${SDK_ROOT}/drive-linux/kernel/source/oss_src

# make output path
mkdir -p out-linux

# export necessary environment
CPU_CORES=$(cat /proc/cpuinfo | grep -c "core id")
export ARCH=arm64
export CROSS_COMPILE=${SDK_ROOT}/toolchains/aarch64--glibc--stable-2020.08-1/bin/aarch64-buildroot-linux-gnu-
export LOCALVERSION="-tegra"
export INSTALL_MOD_PATH=${PWD}/out-linux

# apply Linux RT patch
bash kernel/scripts/rt-patch.sh apply-patches

make -C kernel O=${PWD}/out-linux clean
make -C kernel O=${PWD}/out-linux tegra_defconfig
make -j${CPU_CORES} -C kernel O=${PWD}/out-linux
make -C kernel O=${PWD}/out-linux modules_install

# install kernel image and dtb
cp ${PWD}/out-linux/arch/arm64/boot/Image ${SDK_ROOT}/drive-linux/kernel/preempt_rt/images
cp ${PWD}/out-linux/arch/arm64/boot/dts/nvidia/*.dtb ${SDK_ROOT}/drive-linux/kernel/preempt_rt

# install kernel module
sudo rm ${SDK_ROOT}/drive-linux/filesystem/targetfs/lib/modules/* -rf
sudo cp ${PWD}/out-linux/lib/modules/* ${SDK_ROOT}/drive-linux/filesystem/targetfs/lib/modules/ -arf
sudo chown root:root ${SDK_ROOT}/drive-linux/filesystem/targetfs/lib/modules/* -R
```

3.4 刷机包生成

Shell

```
cd ${SDK_ROOT}

make -C ${DRIVE_FOUNDATION} -f make/Makefile.bind BOARD=${BOARD_NAME} PCT=${BOARD_PCT} clean
make -C ${DRIVE_FOUNDATION} -f make/Makefile.bind BOARD=${BOARD_NAME} PCT=${BOARD_PCT}
${DRIVE_FOUNDATION}/make/bind_partitions -b ${BOARD_NAME} ${BOARD_PCT} -p ufs_boot
python3 ${CREATE_PYTHON} -b ${BOARD_NAME} -B qspi -r 1 -g ${PROJECT_OUT}
sudo cp ${DRIVE_FOUNDATION}/tools/flashtools/storage_configs/t23x/ufs-provision-p*.cfg ${PROJECT_OUT}
```

4. 烧录

4.1 烧写准备

- Ubuntu
- Type-C数据线连接PC及Ubuntu
- SoC及MCU串口连接
- 在MCU串口将SoC切换到烧写模式

Shell

```
tegrarecovery x1 on
tegrareset x1
```

4.2 整体烧录

4.2.1 刷写脚本准备

脚本内容

Shell

```
#!/bin/bash

IMAGE_TOP=${PWD}
BOARD_NAME="p3663-a01"
BSP_INDEX="642-63663-0001-001_TS2"
FLASH_PYTHON=${IMAGE_TOP}/tools/flashtools/bootburn/flash_bsp_images.py
MAC_ADDR_CONFIG_FILE=${IMAGE_TOP}/customer_data_orin.json
UFS_CFG_FILE=ufs-provision-p3710.cfg

if [ "$2" = "3663" ] && [ -f "ufs-provision-p3663.cfg" ] ;then
        UFS_CFG_FILE=ufs-provision-p3663.cfg
fi

function flash_ufs_cfg() {

        echo $UFS_CFG_FILE
        python3 ${FLASH_PYTHON} -b ${BOARD_NAME} -P ${IMAGE_TOP}/${BSP_INDEX} --customer-data ${MAC_ADDR_CONFIG_FILE} -U ${IMAGE_TOP}/$UFS_CFG_FILE
}

function flash_image() {

        python3 ${FLASH_PYTHON} -b ${BOARD_NAME} -P ${IMAGE_TOP}/${BSP_INDEX} --customer-data ${MAC_ADDR_CONFIG_FILE}
}

case $1 in
        ufs)
                flash_ufs_cfg
                ;;
        *)
                flash_image
                ;;
esac
```

保存为`flash.sh`，放置在`${SDK_ROOT}/drive-foundation/tools/flashtools/bootburn/images`路径下

Shell

```
$ cd ${SDK_ROOT}/drive-foundation/tools/flashtools/bootburn/images
$ ls -1
642-63663-0001-001_TS2
customer_data_orin.json
firmware
flash.sh
tools
ufs-provision-p3663.cfg
ufs-provision-p3710.cfg
```

4.2.2 UFS provison

`source flash.sh ufs`

4.2.3 烧录

`source flash.sh`

4.2.4 烧写结束

在MCU串口关闭recovery模式

Shell

```
tegrarecovery x1 off
tegrareset x1
```

4.3 单独烧录

4.3.1 完整编译单独烧写某个分区

0. 进入烧录模式
1. 查找分区

镜像名对应的分区名称在文件 `FileToFlash.txt` 中，内容如下

Plain Text

```
# LinuxPartitionName, PartitionName, FileName, Start, Size, BlockCount, Resize, sku_dependent, BchPartitionName, ImageHeaderType, MD5
/dev/block/3270000.spi bct A_bct_BR_zerosign.bct 0 524288 2 0 0 bct 8 38d9c155523e23171078087d80e289a0
....
/dev/block/2500000.ufshci:0 B_2_kernel-dtb B_2_2_tegra_dtb_zerosign.dtb 84878032896 262144 64 0 0 B_2_kernel-dtb 9 2559b7b90c3c162ac6e1d5e2a4427daa
/dev/block/2500000.ufshci:0 B_2_kernel B_2_3_kernel_zerosign.img 84878295040 24641536 6016 0 0 B_2_kernel 9 24a64ba762b5bb673e3e193efb7ef9c3
/dev/block/2500000.ufshci:0 pers-ota 12_pers-ota_null 111937585152 268435456 65536 0 0 pers-ota 4 d41d8cd98f00b204e9800998ecf8427e
```

例如，烧录`B_2_3_kernel_zerosign.img`，分区名为`B_2_kernel`

2. 烧录

`./flash_bsp_images.py -b p3663-a01 -P ${PWD}/642-63663-0001-001_TS2 --customer-data customer_data_orin.json -u <分区名>`

3. 退出烧录模式

4.3.2 单独编译单独烧写某个分区

0. 进入烧录模式
1. 编译及生成镜像文件
2. 替换需要烧写的文件到已有目录
3. 修改`FileToFlash.txt`中镜像对应的md5值

`$ md5sum xxxxx.img`

4. 烧录

`./flash_bsp_images.py -b p3663-a01 -P ${PWD}/642-63663-0001-001_TS2 --customer-data customer_data_orin.json -u <分区名>`

5. 退出烧录模式

5. 其他