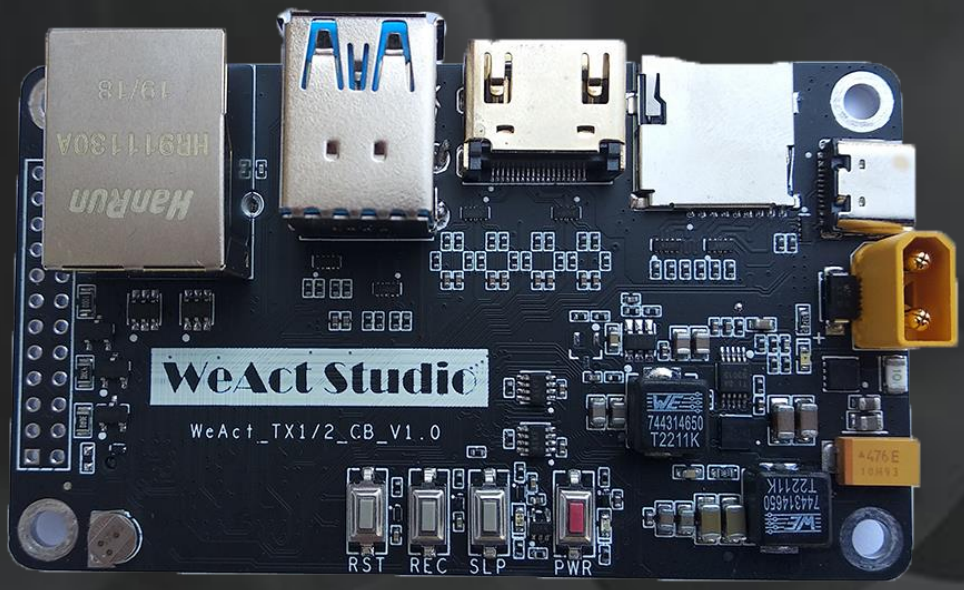


WeAct Studio

TX1/TX2 底板

使用教程

WeAct Studio

目录

REVISION HISTORY

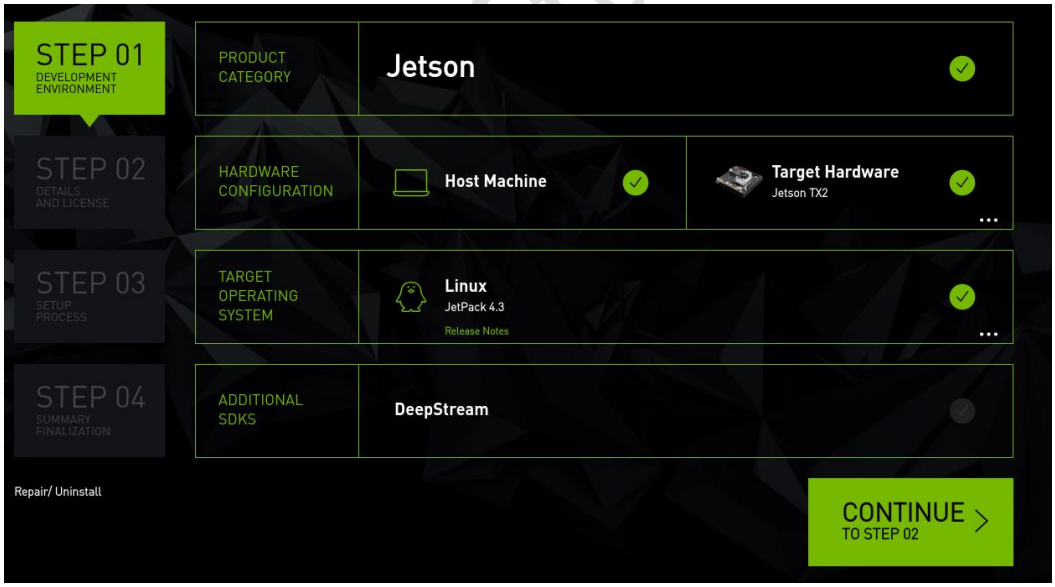
Draft Date	Revision	Description
2020.1.19	V1.0	1. 初始版本
2020.1.25	V1.0.1	1. 增加 SDK 安装教程 2. 增加镜像备份教程
2020.1.26	V1.0.2	1. 增加 CAN 通信教程
2020.10.5	V1.03	1. 增加 GPIO 的 Shell 使用教程

1. 搭建烧写环境

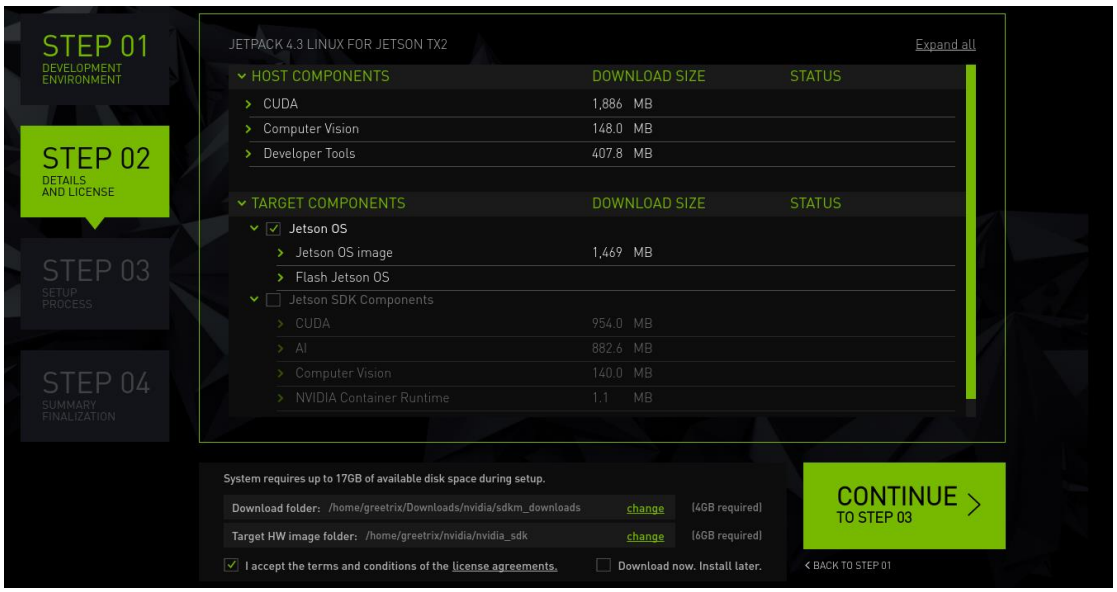
a) 首先，需要一台装有 **Ubuntu16.04** 以上的电脑作为 HOST 端给 JetsonTX2 烧写，或者可以在 Windows 上安装 VMware 来实现。
   - VMware 上如何安装 Ubuntu18.04：
     https://blog.csdn.net/u012556114/article/details/82751089

b) 在 NVIDIA 下载最新的 **SDK-Manager** 并在 ubuntu18.04 中安装（需要注册一个 NVIDIA 账号，后面也需要用到）
   - SDK-Manager 下载地址：https://developer.nvidia.com/nvidia-sdk-manager

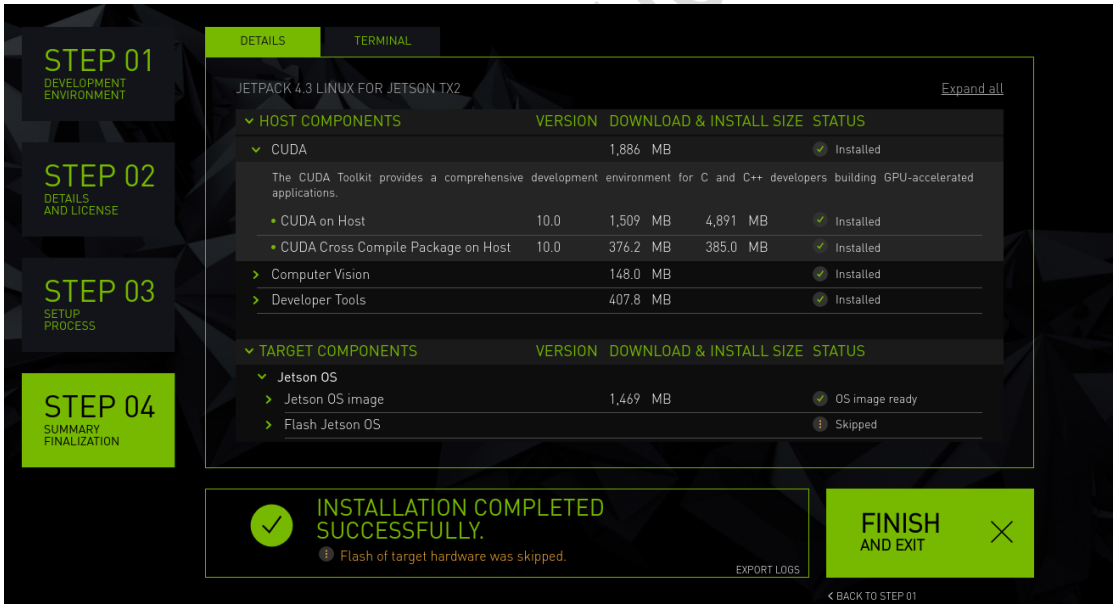
c) 选择需要的 **Target Hardware** 以及 **JetPack** 版本（JetPack3.x 为 Ubuntu16.04，JetPack4.x 为 Ubuntu18.04），这里选择 **JetsonTX2** 的 JetPack4.3 版本（Ubuntu18.04），点击 CONTINUE 进行下一步。



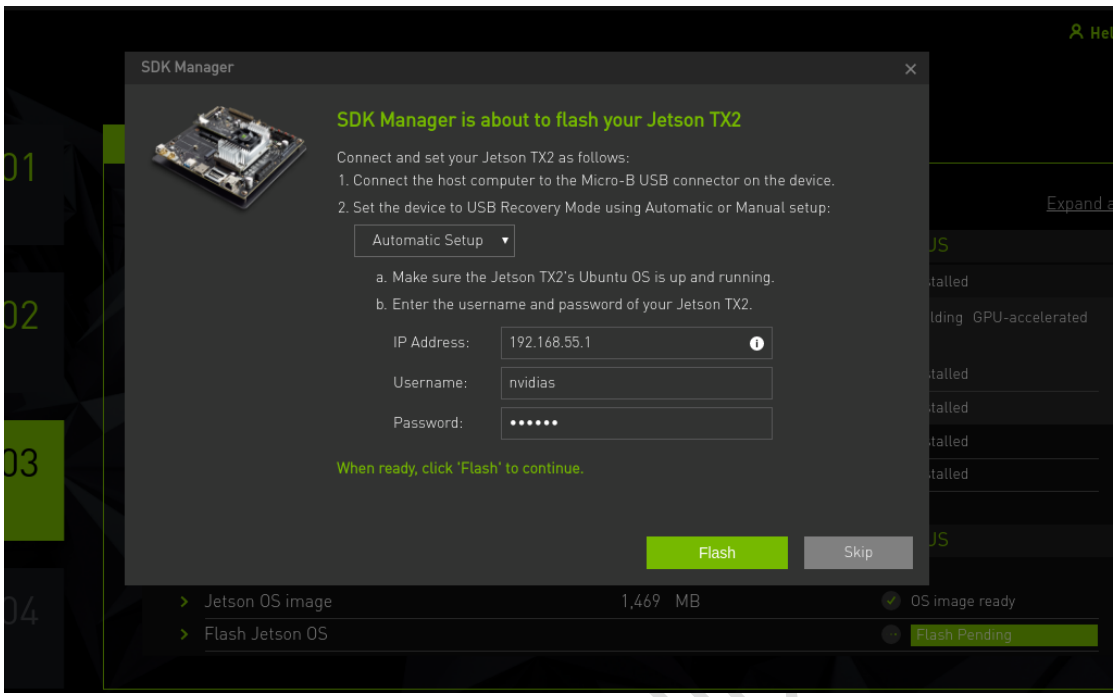
d) 这里需要勾选 **I accept the terms and conditions of the license agreements**，**取消勾选 Jetson SDK Components**（后面会有专门教程安装 Jetson SDK），点击 CONTINUE 进行下一步。



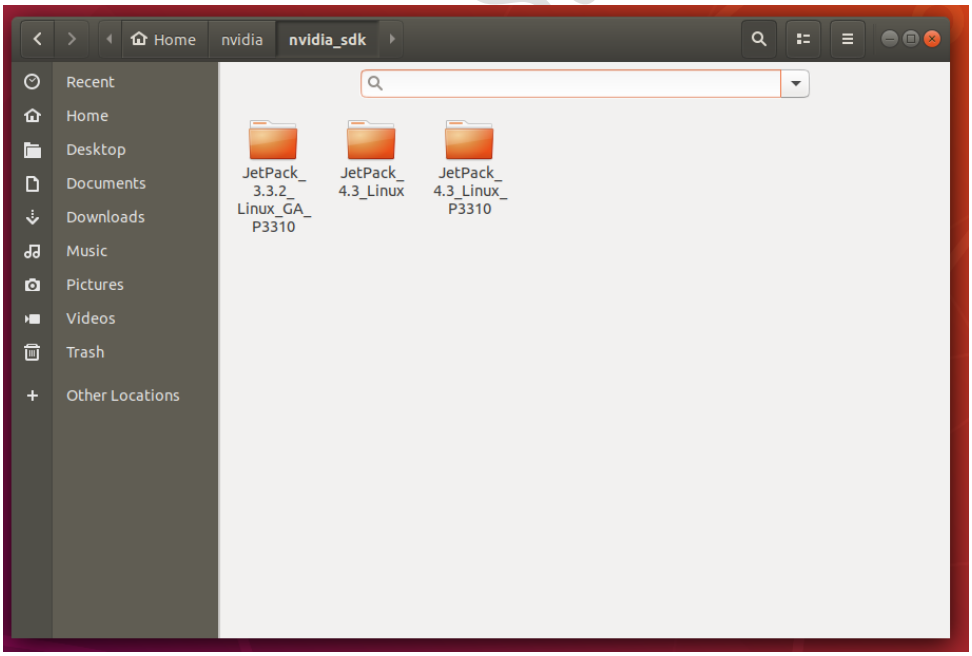
P.S：请在畅通的网络环境下进行下载以及安装，下载或安装失败时，可点击 Retry 继续，直至全部状态为 Installed 并且显示绿色。



e) 安装过程中会弹出联网烧写的信息，选择 **Skip**（需要搭配我们提供的设备树烧写才能正常使用载板，后面教程将会单独介绍烧录）



f) 安装成功后，会在**~/nvidia/nvidia_sdk/** 下有相应版本烧写所需的文件



g) 在终端通过 **sudo apt-get install python** 安装 python 支持以便后续烧写环境。

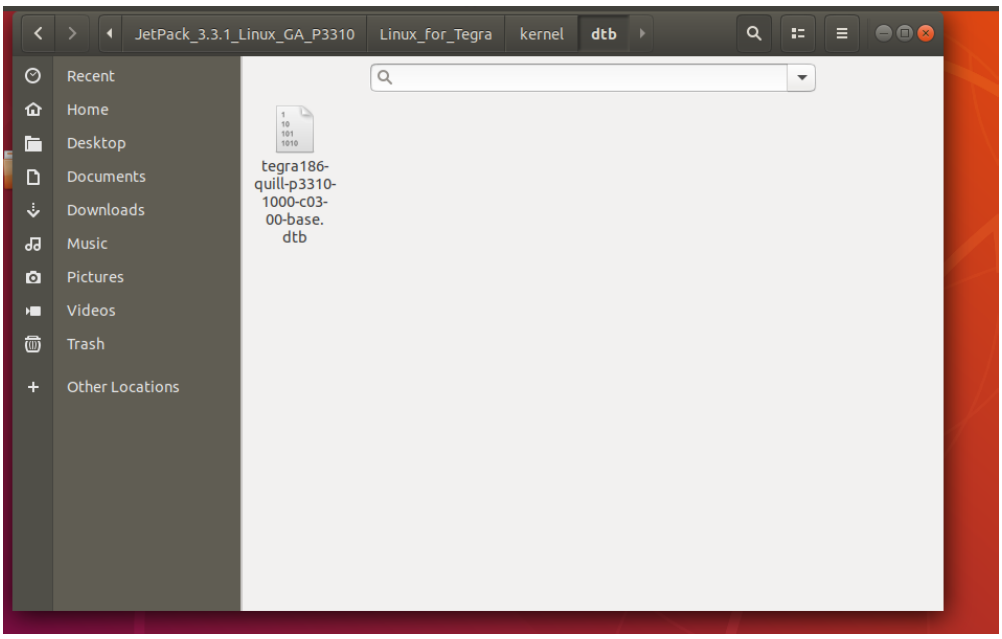
2. 为 TX1/TX2 烧写系统

a) 这里以 TX2 为例，在 WeAct Studio 的 github 或者码云上下载相应的设备树文件，对于 Jetpack3.x 版本使用 L4TR28，而 JetPack4.x 使用 L4TR32。
   - Github：https://github.com/WeActTC/WeAct-TX1_2-CB
   - 码云：https://gitee.com/WeAct-TC/WeAct-TX1_2-CB

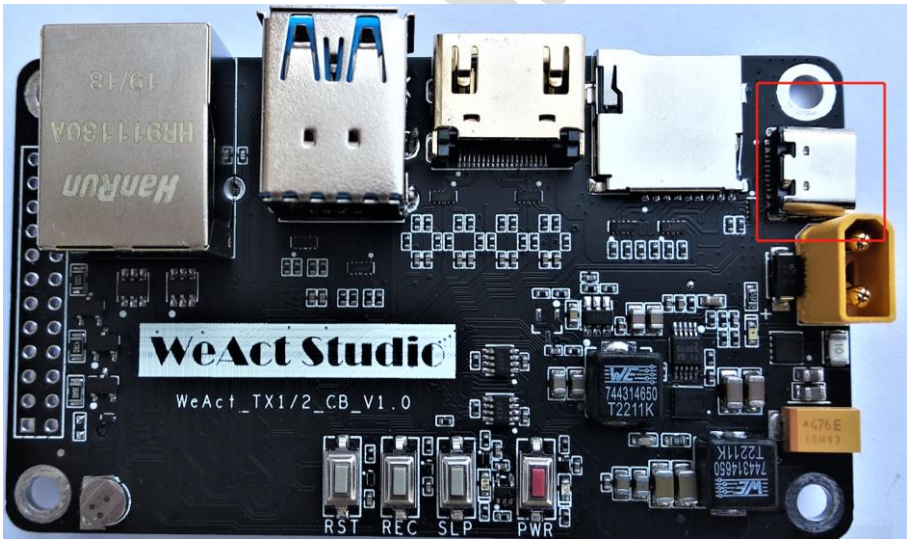
b) 以 JetPack3.1 为例，首先需要进入**~/nvidia/nvidia_sdk/JetPack_3.3.1_Linux_GA_P3310** 文件夹，打开这个文件夹下的 **p2771-0000.conf.common** 文件，将这个文件中的 ODMDATA 的值修改为 **0x7090000**，如下图所示，这个设置将 **USB 配置**为配置 4。

```
local bdv=${board_version^^};
local bid=${board_id^^};
local uboot_build=500;
local fromfab="-a00";
local tofab="-c03";             # default = C03
local pmicfab="-c00";           # default = C00
local bpfdtbfab="-c00";         # default = C00
local tbcdtbfab="-c03";         # default = C03
local kerndtbfab="-c03";        # default = C03
ODMDATA=0x7090000;              # default = C0X
```

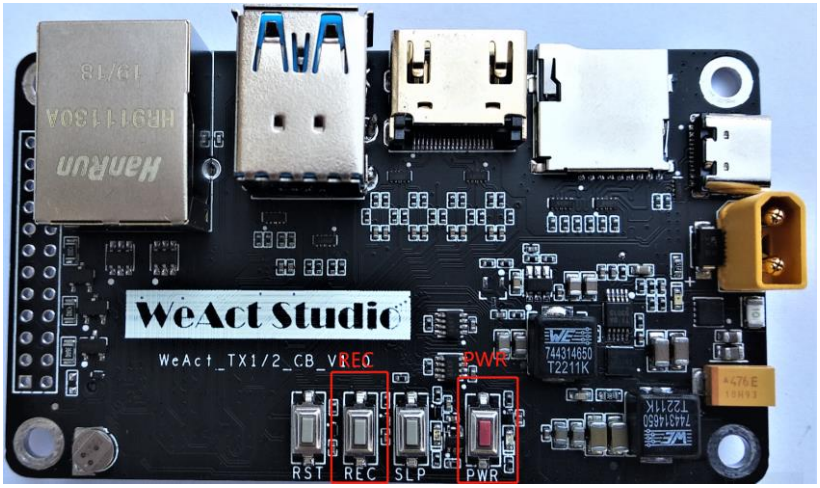
c) 找到相应版本的设备树（**JetsonTX2/L4TR28/tegra186-quill-p3310-1000-c03-00-base.dtb**），进入**~/nvidia/nvidia_sdk/JetPack_3.3.1_Linux_GA_P3310/Linux_for_Tegra/kernel** 并把所有.dtb 文件删除，复制提供的设备树 **tegra186-quill-p3310-1000-c03-00-base.dtb** 至该目录。



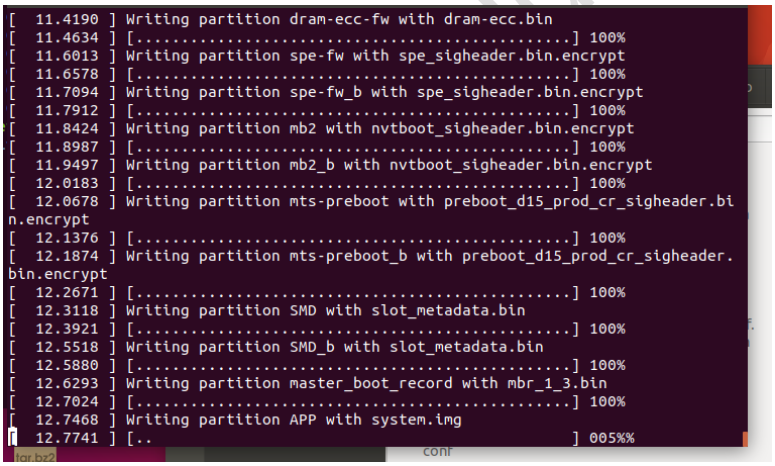
d) 使用 **USB Type-C** 线连接载板上的 **USB OTG** 接口。



e) 摁住 **REC** 键，再摁 **PWR** 键开机，松开 **REC** 键进入 Recovery 模式，此时 VMWare 右下角会出现 **NVIDIA 的 USB 驱动标志**，或者打开终端，输入 **lsusb** 命令，会发现 **Nvidia Corp**。



f) 进入**~/nvidia/nvidia_sdk/JetPack_3.3.1_Linux_GA_P3310/Linux_for_Tegra**，打开终端，运行命令 **sudo ./flash.sh jetson-tx2 mmcblk0p1**，等烧录成功就可以使用啦。【如果仅刷设备树，请使用：sudo ./flash.sh -r -k kernel-dtb jetson-tx2 mmcblk0p1】

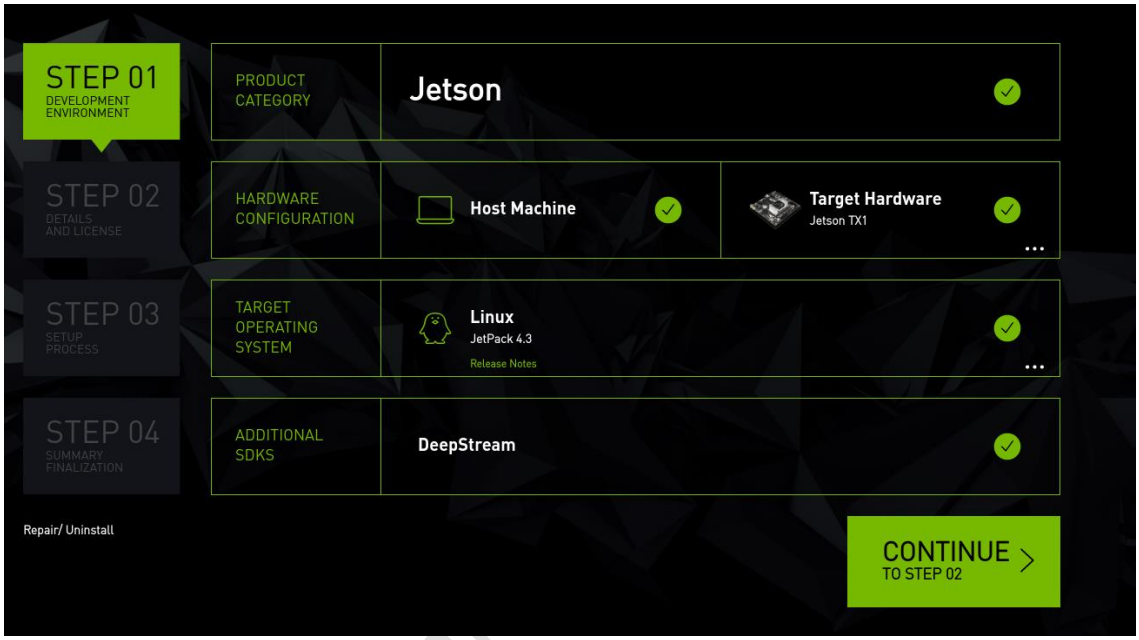


烧录成功后，会有 Successful!显示，如下图所示。

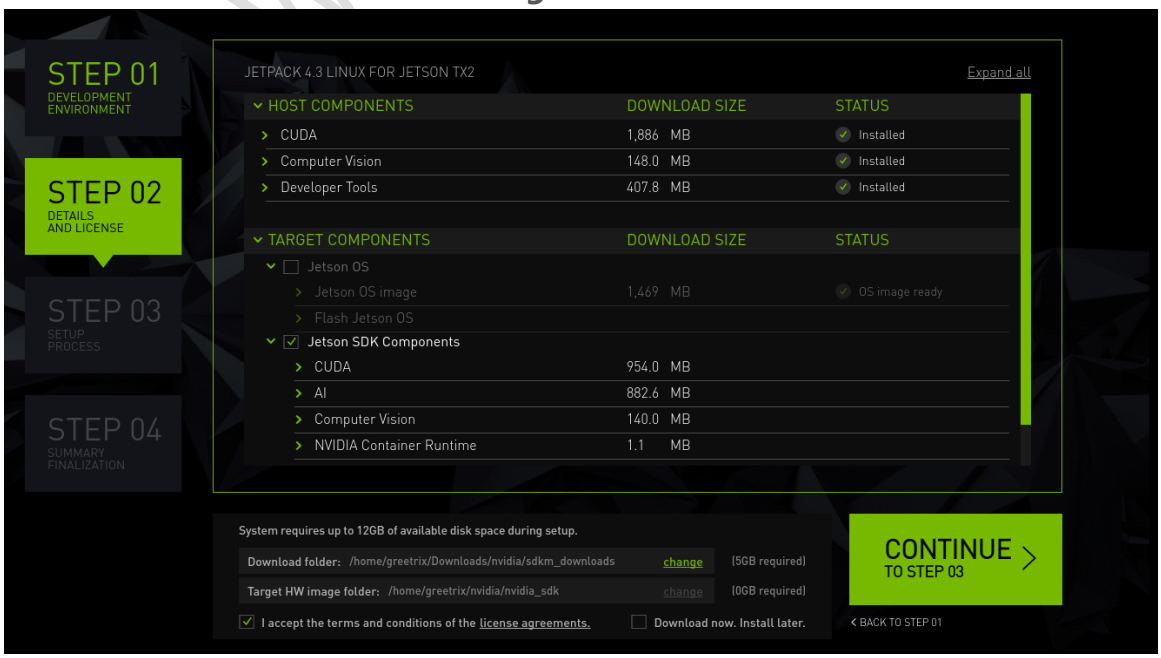
```
t.encrypt
[ 518.3405 ] Bootloader version 01.00.0000
[ 518.4021 ] Writing partition MB1_BCT with mb1_cold_boot_bct_MB1_sigheader.bct
encrypt
[ 518.4033 ] [.............................................] 100%
[ 518.5669 ]
[ 518.5699 ] tegradevflash_v2 --write MB1_BCT_b mb1_cold_boot_bct_MB1_sigheader
bct.encrypt
[ 518.5720 ] Bootloader version 01.00.0000
[ 518.6347 ] Writing partition MB1_BCT_b with mb1_cold_boot_bct_MB1_sigheader.b
t.encrypt
[ 518.6353 ] [.............................................] 100%
[ 518.7580 ]
[ 518.7581 ] Flashing completed

[ 518.7582 ] Coldbooting the device
[ 518.7598 ] tegradevflash_v2 --reboot coldboot
[ 518.7613 ] Bootloader version 01.00.0000
[ 518.8545 ]
*** The target t186ref has been flashed successfully. ***
Reset the board to boot from internal eMMC.
```

3. 安装系统组件(SDK)

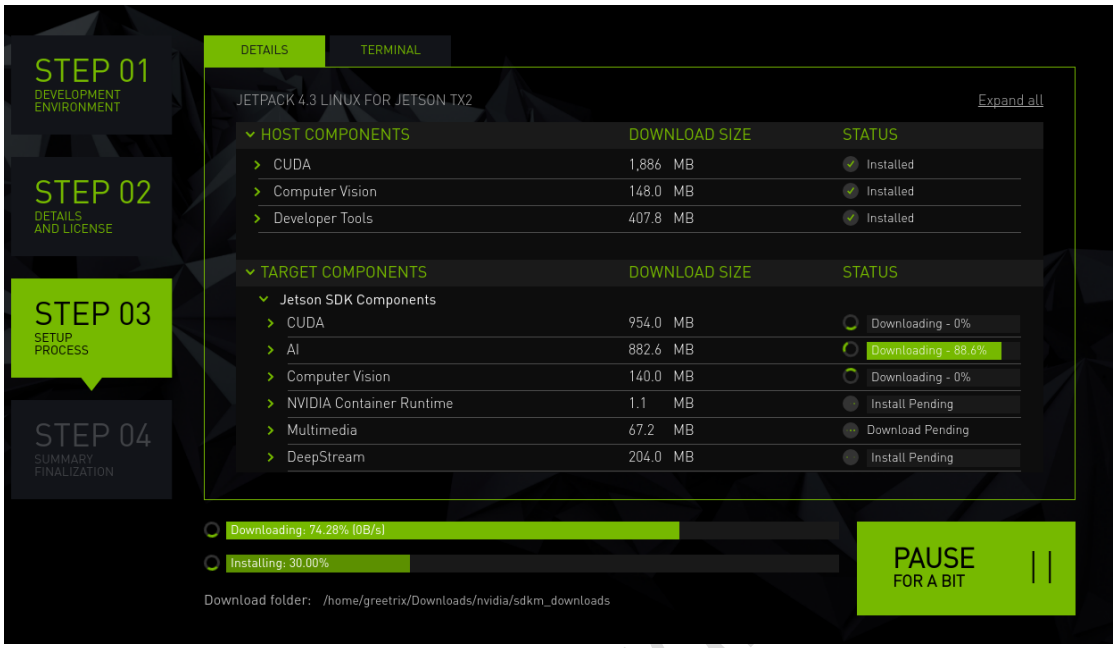
a) 这里已**JetPack4.3**为例（支持最新的DeepSteam组件），这里我们勾选**DeepStream**。



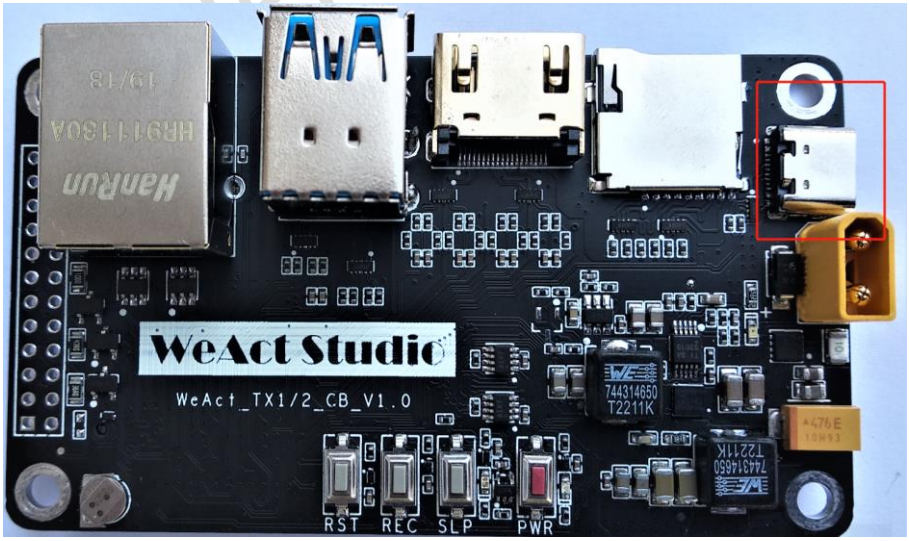
b) 这里需要勾选**Jetson SDK Components**，并取消勾选**Jetson OS**，勾选**I accept the terms and conditions of the license agreements。**



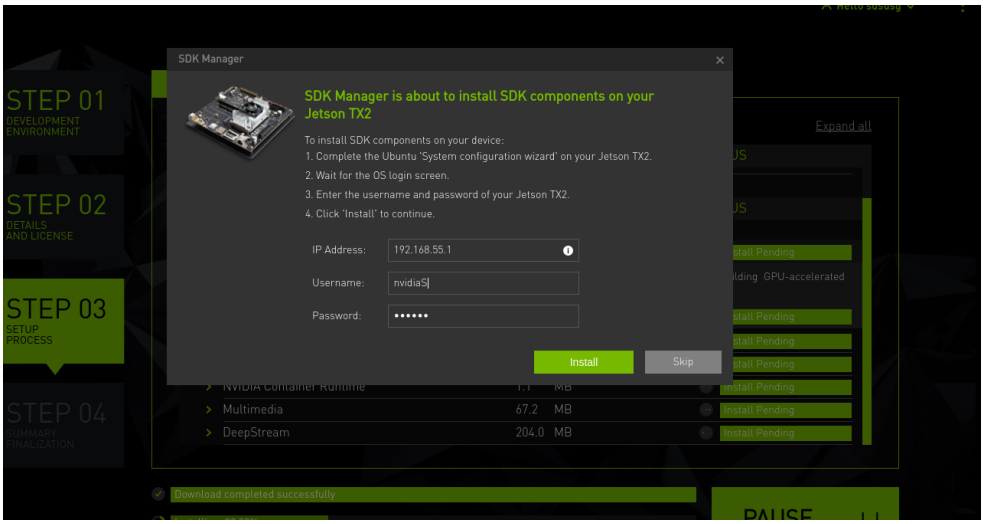
c) 等待下载完成。



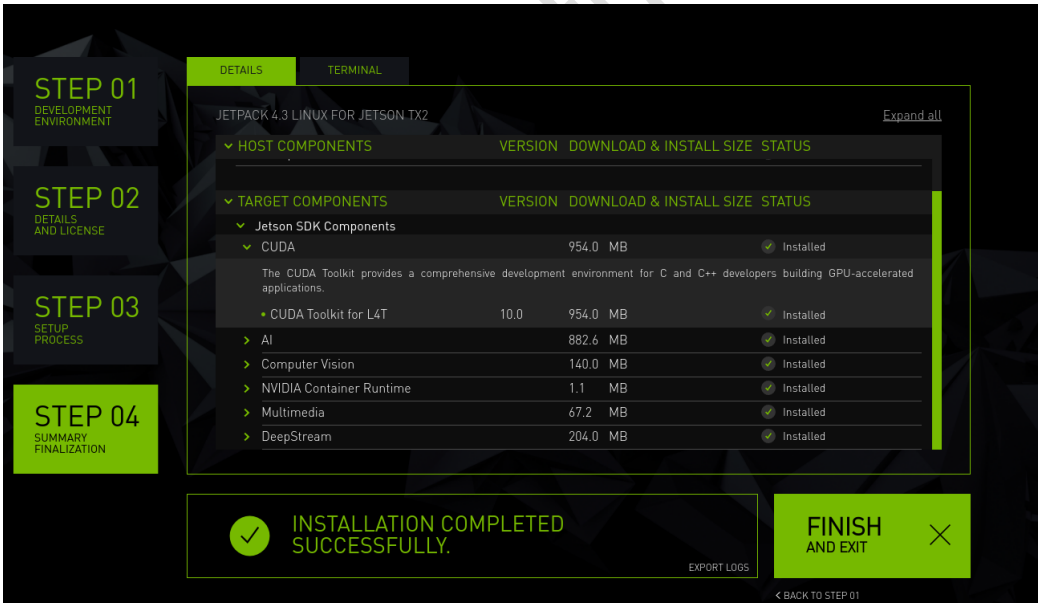
g) 使用 **USB Type-C** 线连接载板上的 **USB OTG** 接口，摁 **PWR** 正常开机，设置账号和密码，登入系统。这里使用 TX2 自带的 USB 网卡进行烧录。



d) 填入 TX2 的 Username 和 Password。点击 Install 进行 SDK 的安装。



e) 等待安装完成，此时所有勾选的 SDK 已经安装在你的设备上了。



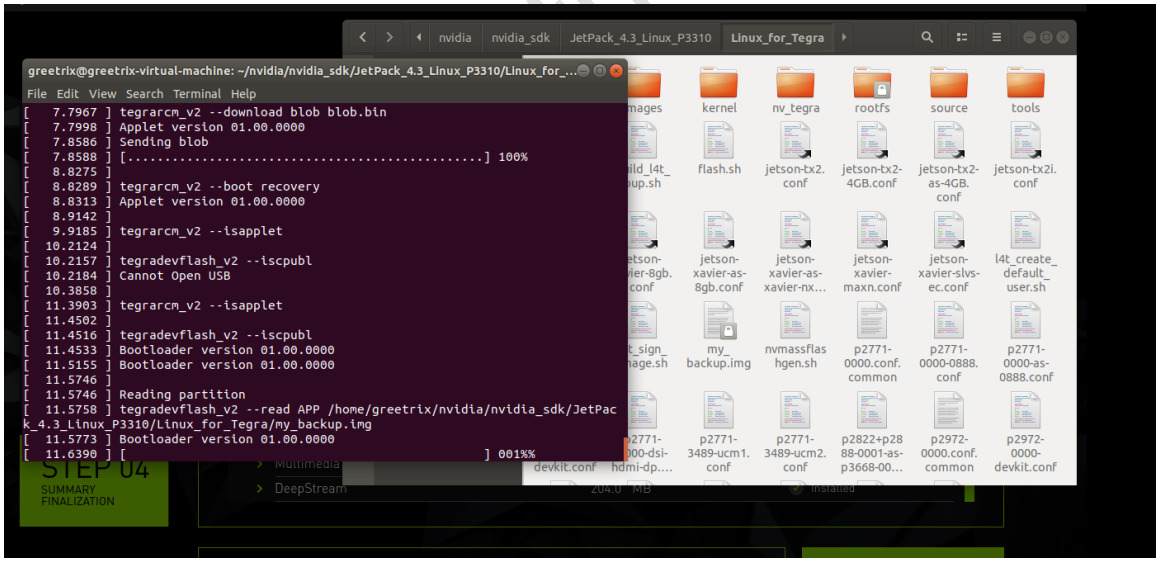
4. 镜像备份与恢复

➢ 镜像备份

a) 在每次刷机前最好先备份镜像，防止刷机时不小心覆盖了镜像。首先，需要搭建烧写环境，参考**教程 1。**

b) 参考**教程 2，使 Jetson TX2 进入 Recovery 模式。**

c) 进入 **~/nvidia/nvidia_sdk/JetPack_3.3.1_Linux_GA_P3310/Linux_for_Tegra**，打开终端，运行命令 **sudo ./flash.sh -r -k APP -G my_backup.img jetson-tx2 mmcblk0p1** 后进行备份。备份完成后会在当前目录下生成的备份文件 my_backup.img。



- 镜像恢复

a) 进入**~/nvidia/nvidia_sdk/JetPack_3.3.1_Linux_GA_P3310/Linux_for_Tegra**，将 my_backup.img 复制一份，并且重命名为 system.img。

b) 进入 **~/nvidia/nvidia_sdk/JetPack_3.3.1_Linux_GA_P3310/Linux_for_Tegra/bootloader**，打开终端【**mv system.img system_bak.img.bak**】，先把原有的系统镜像备份。

c) 终端【mv ../system.img system.img】，复制生成的新镜像文件到 bootloader 目录。

d) 烧录步骤参考**教程 2，【重要】烧录命令：sudo ./flash.sh -r jetson-tx2 mmcblk0p1 多了-r 参数。**

5. 使用 CAN 进行通信

a) JetsonTX2 上集成了 2 个 CAN 控制器（CAN0/CAN1），另外 WeAct Studio 的载板上设计了 2 个 CAN 收发器，可直接挂载 CAN 物理总线使用。

b) TX2 自带 canbus 的驱动并集成到了镜像中，已经支持 canbus 无需多做处理。我们需要安装 canbus 模块。（在终端输入下面命令或者放入 rc.local 里面开启自启）

```
modprobe can        // 插入 can 总线子系统
modprobe can-raw    //插入 can 协议模块
modprobe can-bcm
modprobe can-gw
modprobe can_dev
modprobe mttcan     //真正的 can 口支持
```

c) 通过 **lsmod** 检查是否安装成功。

```
nvidia@localhost:~$ lsmod
Module                  Size  Used by
fuse                  103841  2
mttcan                 66251  0
can_dev                13306  1 mttcan
can_gw                 10919  0
can_bcm                16471  0
can_raw                10388  0
can                    46600  3 can_raw,can_bcm,can_gw
zram                   26166  6
overlay                48691  0
bcmdhd                934274  0
cfg80211              589351  1 bcmdhd
spidev                 13282  0
nvgpu                1575721  20
bluedroid_pm           13912  0
ip_tables              19441  0
x_tables               28951  1 ip_tables
```

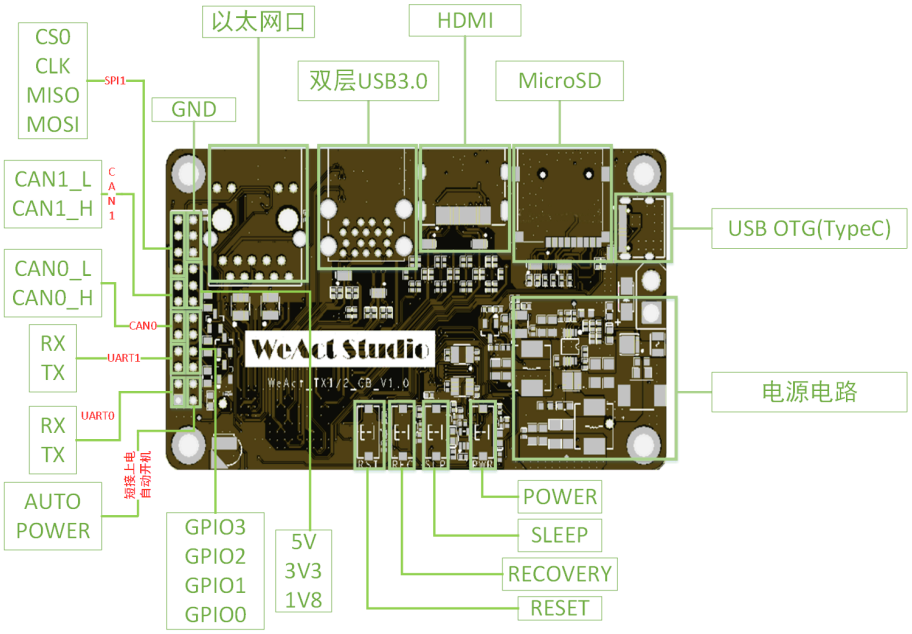
d) 配置 canbus 属性，和串口的波特率设置类似。

```
sudo ip link set can0 type can bitrate 500000
sudo ip link set up can0
sudo ip link set can1 type can bitrate 500000
sudo ip link set up can1
```

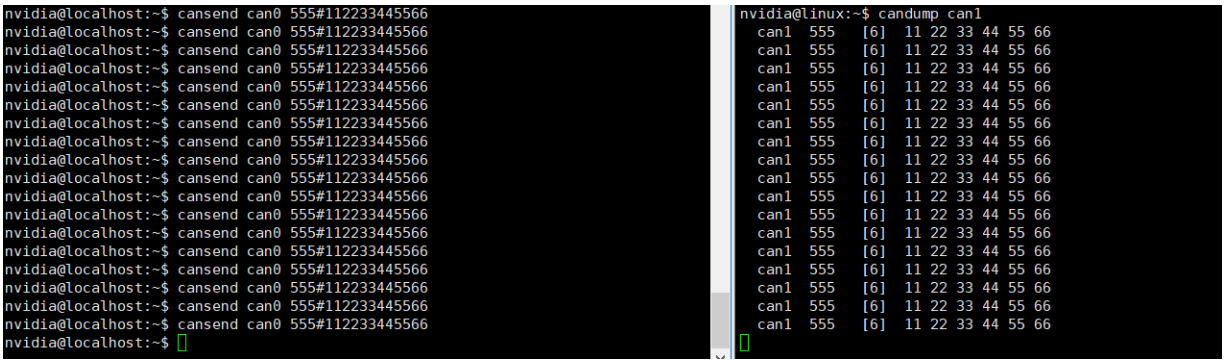
e) 通过 ifconfig 查看是否配置成功。

```
nvidia@localhost:~$ ifconfig
can0: flags=193<UP,RUNNING,NOARP>  mtu 16
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 10  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 131

can1: flags=193<UP,RUNNING,NOARP>  mtu 16
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 10  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 132
```

f) 由于载板上搭载了 CAN0/CAN1 的收发器，可以直接通过 CAN0(1)-H/L 互接进行通信（CAN0_H 接 CAN1_H，CAN1_L 接 CAN1_L）。

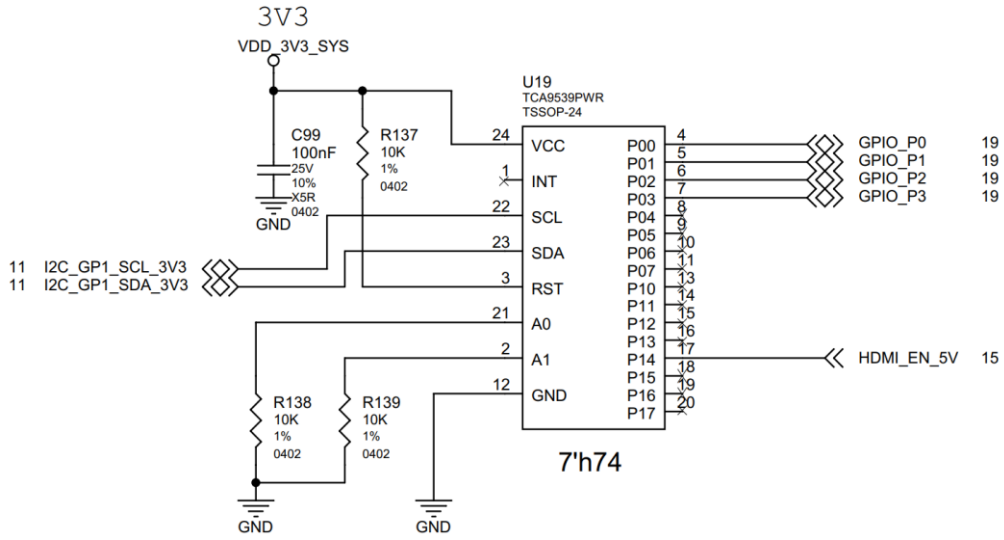


g) 在一个终端通过 cansend can0(can1) ×××命令来发送数据, 另一个终端通过 candump can1(can0)完成实际信号收发测试



6. GPIO 在 SHELL 中使用

a) TX2 载板上的 GPIO 是 I2C 总线 0 上的 IO 扩展器引出的，需要通过 I2C 协议去控制 GPO 的使用，原理图如下所示，器件型号为 TCA9539，**总线地址为 0，I2C 地址为 0x74。**



b) 我们无需关注 I2C 协议时序，可以直接通过 TX2 系统自带的 i2c-tools 进行 shell 操作去控制 IO。

c) 首先，需要关注一下 TCA9539 寄存器的操作，具体可以去看其 datasheet，这里我截取几个关键寄存器说明，载板上 4 个 IO 均挂在 P0 上，只需要关注**偶数寄存器**即可

Table 4. Registers 0 And 1 (Input Port Registers)

Bit	I0.7	I0.6	I0.5	I0.4	I0.3	I0.2	I0.1	I0.0
Default	X	X	X	X	X	X	X	X
Bit	I1.7	I1.6	I1.5	I1.4	I1.3	I1.2	I1.1	I1.0
Default	X	X	X	X	X	X	X	X

上图为输入寄存器（只需关注**寄存器 0**）GPIO 输入模式下，直接**读取**该寄存器能从 IO 输入电平信号。

Table 5. Registers 2 And 3 (Output Port Registers)

Bit	O0.7	O0.6	O0.5	O0.4	O0.3	O0.2	O0.1	O0.0
Default	1	1	1	1	1	1	1	1
Bit	O1.7	O1.6	O1.5	O1.4	O1.3	O1.2	O1.1	O1.0
Default	1	1	1	1	1	1	1	1

上图为输出寄存器（只需关注**寄存器 2**），在 GPIO 输出模式下，直接**写入** IO 电平即可控制 GPIO 的输出。

Table 7. Registers 6 And 7 (Configuration Registers)

Bit	C0.7	C0.6	C0.5	C0.4	C0.3	C0.2	C0.1	C0.0
Default	1	1	1	1	1	1	1	1
Bit	C1.7	C1.6	C1.5	C1.4	C1.3	C1.2	C1.1	C1.0
Default	1	1	1	1	1	1	1	1

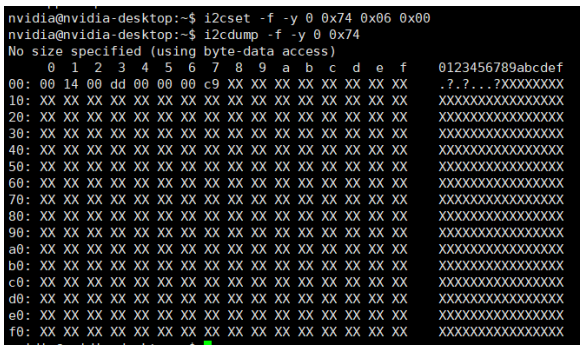
上图为配置寄存器（只需关注**寄存器 6**），设置 Bit 为 1 则配置该 IO 为输入模式，0 则为输出模式。

d) I2C-tools 的使用，由于 I2C-tools 命令众多，这里只挑选读取及写入进行说明，其他命令可以在 **http://www.lm-sensors.org/wiki/i2cToolsDocumentation 查阅。**

1） 控制 GPIO 输出。

i. 使用 i2cset 命令向 i2c 器件写入命令，首先把 GPIO 模式配置为输出模式，这里将全部配置为输出。器件 I2C 总线地址为 0，I2C 地址为 0x74，寄存器地址为 0x06，写入值为 0x00。

命令：i2cset -f -y **0 0x74 0x06 0x00**(写入) | i2cdump -f -y 0 0x74（读取 I2C 寄存器值验证）



令所有 GPIO 输出为高电平，器件 I2C 总线地址为 0，I2C 地址为 0x74，寄存器地址为 0x02，写入值为 0xff，器件电平为 3V3。

命令：i2cset -f -y **0 0x74 0x02 0xff**

```
nvidia@nvidia-desktop:~$ i2cset -f -y 0 0x74 0x02 0xff
```

2) 控制 GPIO 输入。

i. 使用 i2cset 命令向 i2c 器件写入命令，首先把 GPIO 模式配置为输入模式，这里将全部配置为输入。器件 I2C 总线地址为 0，I2C 地址为 0x74，寄存器地址为 0x06，写入值为 0xff。

命令：i2cset -f -y 0 0x74 0x06 0xff(写入) | i2cdump -f -y 0 0x74（读取 I2C 寄存器值验证)

```
nvidia@nvidia-desktop:~$ i2cset -f -y 0 0x74 0x06 0x00
nvidia@nvidia-desktop:~$ i2cdump -f -y 0 0x74
No size specified (using byte-data access)
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: 00 14 00 dd 00 00 00 c9 XX XX XX XX XX XX XX XX    .?.?...?XXXXXXXX
10: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
20: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
30: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
40: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
50: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
60: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
70: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
80: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
90: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
a0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
b0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
c0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
d0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
e0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
f0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
```

将 GPIO0 与 GND 相连，dump 出寄存器 0 的值（若 IO 悬空读出的值为 1），最低位为 0，所以值为 0xfe

```
nvidia@nvidia-desktop:~$ i2cdump -f -y 0 0x74
No size specified (using byte-data access)
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: fe 14 ff dd 00 00 ff c9 XX XX XX XX XX XX XX XX    ??.?...?XXXXXXXX
10: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
20: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
30: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
40: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
50: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
60: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
70: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
80: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
90: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
a0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
b0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
c0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
d0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
e0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
f0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
```

也可以通过 i2cget 进行读取，器件 I2C 总线地址为 0，I2C 地址为 0x74，寄存器地址为 0x00

命令：i2cget -f -y 0 0x74 0x00

```
nvidia@nvidia-desktop:~$ i2cget -f -y 0 0x74 0x00
0xfe
```

联系我们

- Github：https://github.com/WeActTC
- 码云：https://gitee.com/WeAct-TC
- 网站：https://www.weact-tc.cn/
- 淘宝：https://shop118454188.taobao.com

WeAct Studio
官方淘宝店