Purple Pi R1系统使用说明

Purple Pi R1

系统使用说明

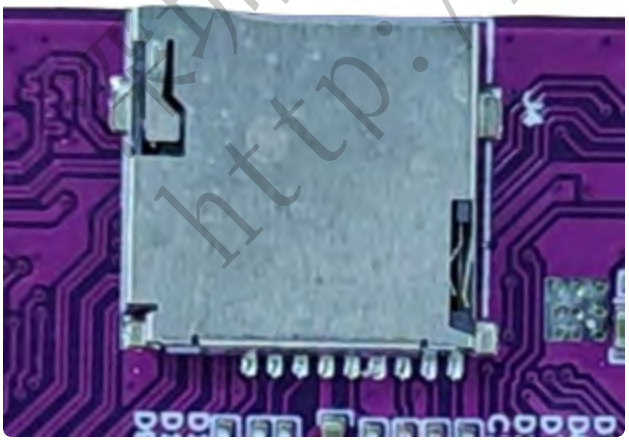
深圳触觉智能科技有限公司

www.industio.cn

接口说明

SD Card

开发板已经默认加载了SD卡驱动，插上SD卡后，在系统中会出现节点/dev/mmcblk1p1（如有多个分区，则会出现多个/dev/mmcblk1p*），开发板的SD卡对应接口位于J4。



插入SD卡后，系统会默认把SD卡，挂载到/sdcard目录下。

将SD卡插入卡槽中，系统会提示以下信息：

```
mmc1: new high speed SDHC card at address aaaa
mmcblk1: mmc1:aaaa SC16G 14.8 GiB
 mmcblk1: p1
FAT-fs (mmcblk1p1): utf8 is not a recommended IO charset for FAT filesystems, filesystem will be case sensitive!
FAT-fs (mmcblk1p1): Volume was not properly unmounted. Some data may be corrupt. Please run fsck.
```

弹出SD卡，系统有如下提示：

```
mmc1: card aaaa removed
[Padmux]reset Pad_51(reg 0x101e08; mask0x300) to GPIO(org: SDIO_MODE_1)
```

输入 df –h ，可查看SD卡挂载路径和挂载信息，默认挂载路径为“/sdcard”。

```
# df -h
Filesystem                Size      Used Available Use% Mounted on
ubi:rootfs               85.6M     15.6M     70.0M  18% /
devtmpfs                 51.1M         0     51.1M   0% /dev
df: /dev/shm: No such file or directory
tmpfs                    51.1M     56.0K     51.0M   0% /tmp
tmpfs                    51.1M     28.0K     51.0M   0% /run
mdev                     51.1M         0     51.1M   0% /dev
ubi0:miservice            7.5M      6.1M      1.4M  81% /config
ubi0:customer            88.1M    500.0K     87.7M   1% /customer
ubi0:appconfigs           2.0M     24.0K      2.0M   1% /appconfigs
/dev/mmcblk1p1           14.8G     32.5M     14.8G   0% /sdcard
#
```

测试SD卡的读写速度

```
1  # time dd if=/dev/zero of=/sdcard/test bs=1k count=10240 conv=fsync
2  # time dd if=/sdcard/test of=/dev/null bs=1k count=10240
```

USB

开发板USB对外接口为J5，如下图所示：

插入U盘后，系统会默认把U盘，挂载到/udisk目录下

插入U盘，系统会提示以下信息：

```
==20180309==> hub_port_init 1 #0
Plug in USB Port2
usb 1-1: new high-speed USB device number 3 using Sstar-ehci-2
usb 1-1: New USB device found, idVendor=090c, idProduct=3281
usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=0
usb 1-1: Product: U268
usb 1-1: Manufacturer: aigo
usb-storage 1-1:1.0: USB Mass Storage device detected
scsi host1: usb-storage 1-1:1.0
scsi 1:0:0:0: Direct-Access     aigo     U268            1100 PQ: 0 ANSI: 4
sd 1:0:0:0: [sdb] 61736960 512-byte logical blocks: (31.6 GB/29.4 GiB)
sd 1:0:0:0: [sdb] Write Protect is off
sd 1:0:0:0: [sdb] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
 sdb: sdb1
sd 1:0:0:0: [sdb] Attached SCSI removable disk
FAT-fs (sdb1): utf8 is not a recommended IO charset for FAT filesystems, filesystem will be case sensitive!
FAT-fs (sdb1): Volume was not properly unmounted. Some data may be corrupt. Please run fsck.
```

拔出U盘，系统有如下提示：

```
# usb 1-1: USB disconnect, device number 3
sd 1:0:0:0: [sdb] Synchronizing SCSI cache
sd 1:0:0:0: [sdb] Synchronize Cache(10) failed: Result: hostbyte=0x01 driverbyte=0x00
root hub reinitial [usbdis]
```

输入df –h，可查看U盘挂载路径和挂载信息，默认挂载路径为“/udisk”。

```
/ # df -h
Filesystem                Size      Used Available Use% Mounted on
ubi:rootfs              180.4M     26.4M    154.0M  15% /
devtmpfs                 14.4M         0     14.4M   0% /dev
tmpfs                    14.4M         0     14.4M   0% /tmp
var                      14.4M         0     14.4M   0% /var
mdev                     14.4M         0     14.4M   0% /dev
ubi0:miservice            7.5M      5.6M      1.9M  75% /config
ubi0:customer             2.9M    160.0K      2.8M   5% /customer
ubi0:appconfigs           2.9M     24.0K      2.9M   1% /appconfigs
/dev/sda1                14.4G      2.8G     11.6G  20% /udisk
/ #
```

测试U盘的读写速度

Plain Text | 复制代码

```
# time dd if=/dev/zero of=/udisk/test bs=1k count=10240 conv=fsync
# time dd if=/udisk/test of=/dev/null bs=1k count=10240
```

MIC

模块用的是AMIC，我们可用测试demo：audio_all_test_case进行测试，（此程序在发布包的sdk\verify\mi_demo\geonosis\audio_all_test_case）

prog_audio_all_test_case使用说明:

```
-t: 程序的运行时间（秒数），不指定则会一直运行
-I: 使能AI
-o: AI录音的输出路径
-d: AI的设备ID(Amic[0] Dmic[1] I2S RX[2] Linein[3])
-c: AI通道数
-v: AI音量参数(Amic 0~21, Dmic 0~4, Linein 0~7)
-s: AI采样率8000/16000/32000/48000
-q: 是否使用AI queue mode
-h: 使能AI Hpf
-g: 使能AI Agc
-e: 使能AI Eq
-n: 使能AI NR
-r: AI 重采样采样率8000/16000/32000/48000
-a: AI 编码类型g711a/g711u/g726_16/g726_24/g726_32/g726_40
-A: 使能AED
-b: 使能AEC
-O: 使能AO
-i: AO播放的输入文件路径
-D: AO设备ID(Lineout[0] I2S TX[1] HDMI[2])
-V: AO音量参数(-60~30)
-h: 使能AO Hpf
-g: 使能AO Agc
-e: 使能AO Eq
-n: 使能AO NR
-r: AO 重采样采样率8000/16000/32000/48000
```

录音指令如下：

```
#prog_audio_all_test_case  -t 20 -I -o /tmp -d 0  -c 2 -v 15 -s 8000
```

Amic 单声道 采样率8K，录音30秒，保存路径为/tmp，音量参数为15，采样率为8000。

播放指令如下：

```
 #prog_audio_all_test_case -t 10 -O -i /tmp/Chn0_Amic_8K_16bit_MONO.wav  -D 0 -V 3
```

耳机

Lineout播放测试demo同MIC一样：audio_all_test_case

Lineout播放/media/pizzicato.wav

note：Only support wav file.

```
#prog_audio_all_test_case -t 10 -O -i /media/pizzicato.wav -D 0 -V 3
```

TP

我们已经配置好tslib，可以直接使用它生成的工具。

```
# ls /usr/bin/ts*
/usr/bin/ts              /usr/bin/ts_harvest    /usr/bin/ts_test
/usr/bin/ts_calibrate    /usr/bin/ts_print      /usr/bin/ts_test_mt
/usr/bin/ts_conf         /usr/bin/ts_print_mt   /usr/bin/ts_uinput
/usr/bin/ts_finddev      /usr/bin/ts_print_raw  /usr/bin/ts_verify
#
```

TP测试方法如下：

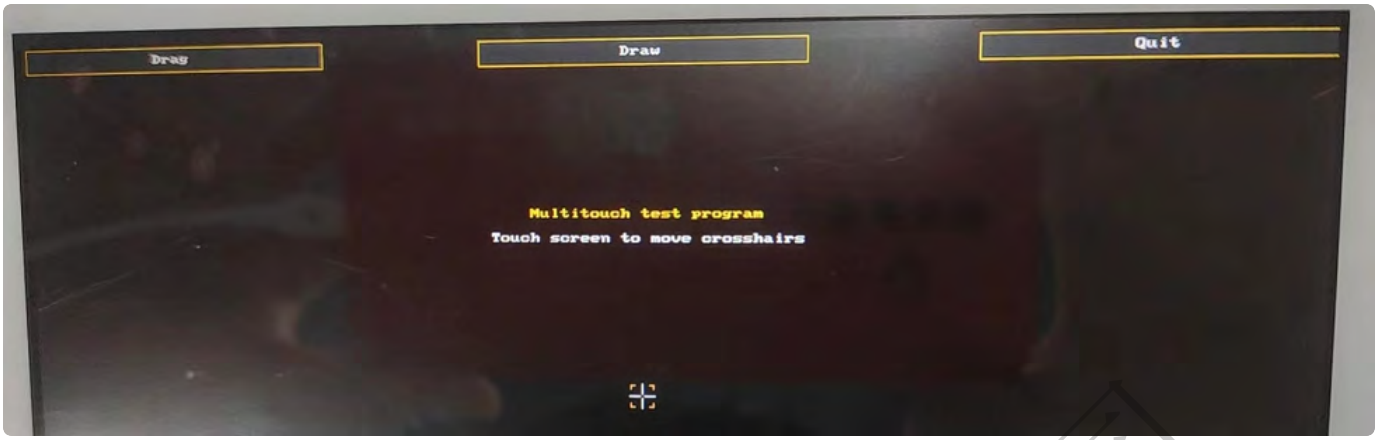
初始化屏幕：

```
# cd /customer
# ./disp_init &
```

执行测试demo：

```
#ts_test_mt
```

可看到以下界面，我们可以在demo中执行划线操作，验证tp是否触摸正常。



注意：使用ts_test_需mt要执行disp_init来初始化屏，用logo会出现花屏

我们也可以通过测试程序tp_test进行测试：

tp_test.c测试源码：

tp_test.c Plain Text 复制代码

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/input.h>

static int event0_fd = -1;
struct input_event ev0;

static int handle_event0()
{
        int rd;

        rd = read(event0_fd, &ev0, sizeof(struct input_event));
        if(rd < sizeof(struct input_event)){
                return 0;
        }

        if(EV_ABS == ev0.type){
                if (ev0.code == ABS_X){
                        printf("ABS_X:");
                }else if (ev0.code == ABS_Y){
                        printf("ABS_Y:");
                }else if (ev0.code == ABS_PRESSURE){
                        printf("ABS_PRESSURE:");
                }else{
                        printf("UNKNOWEN:");
                }
                printf("value:%d\n", ev0.value);
        }

        return 1;
}

int main(void)
{
        int done = 1;

        event0_fd = open("/dev/input/event0", O_RDONLY);
        if(event0_fd <0) {
                printf("open input device error\n");
                return -1;
        }
        while (done){
```

```
                done = handle_event0();
        }

        if(event0_fd > 0){
                close(event0_fd);
                event0_fd = -1;
        }
        return 0;
}
```

编译： arm-linux-gnueabihf-gcc tp_test.c -o tp_test

将TP测试源码拷贝到Ubuntu中，编译生成tp_test，然后通过U盘拷贝至开发板。

执行tp_test测试：/udisk/tp_test

```
# /udisk/tp_test
UNKNOWEN:value:60
UNKNOWEN:value:213
UNKNOWEN:value:239
UNKNOWEN:value:38
UNKNOWEN:value:38
ABS_X:value:213
ABS_Y:value:239
UNKNOWEN:value:-1
UNKNOWEN:value:61
UNKNOWEN:value:262
UNKNOWEN:value:304
UNKNOWEN:value:36
UNKNOWEN:value:36
ABS_X:value:262
ABS_Y:value:304
UNKNOWEN:value:-1
UNKNOWEN:value:62
UNKNOWEN:value:562
UNKNOWEN:value:265
UNKNOWEN:value:40
UNKNOWEN:value:40
ABS_X:value:562
ABS_Y:value:265
UNKNOWEN:value:-1
```

LCD

在Purple Pi上的LCD接口默认是RGB565的，需外接LCD显示屏的转接板,这里适配一块7寸1024x600分辨率的MIPI屏。

屏幕资料：

📎IDO-EXB2D06-V1-SCH.pdf

📎2D06-7寸屏资料.zip

我们可以通过logo来显示图片测试屏幕是否显示正常。

找一张1024x600分辨率JPG格式的图片，并重命名为logo.jpg。

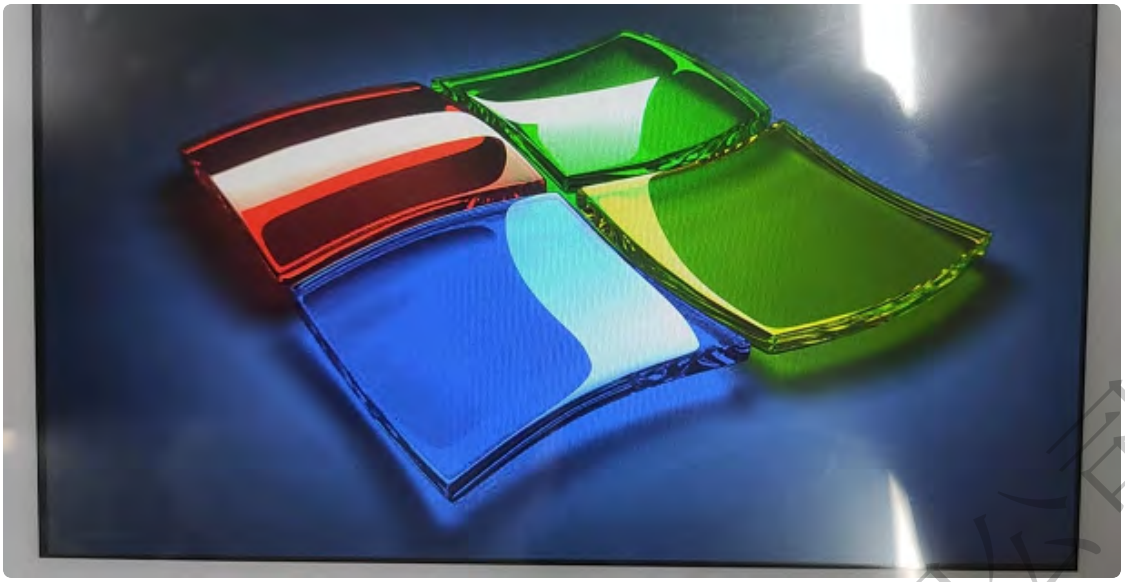
将logo.jpg 和 logo 放在同一目录下。

输入以下指令：

```
1   #./logo logo.jpg &
```

```
# ./logo logo.jpg
client [949] connected, module:sys
client [949] connected, module:disp
[MI_SYSCFG_GetPanelInfo 50] eTiming = 4, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 2, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 8, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 9, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 10, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 6, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 7, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 11, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 13, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 12, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 14, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 5, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 3, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 38, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 50, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 40, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 43, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 41, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 44, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 42, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 46, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 50, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 45, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 50, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 22, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 23, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 30, hdmiTx = 1  Not Fund!!!
[MI_SYSCFG_GetPanelInfo 50] eTiming = 32, hdmiTx = 1  Not Fund!!!
client [949] connected, module:panel
---------------- [DCLK:49]  [FPS:56] ----------
DISP width: 1024,height: 600
[MI_PANEL_Init][332]LCD environment is Invalid
main 166 286

sstar_FB_SetBlending 402 u8GOP=1,u8win=0 aType=1 u8constAlpha=255
xres: 1024,yres: 600
load_JPEG_file 32 600 1024 3072
load_JPEG_file 88 600 1024 3072
main 264 460
client [949] disconnected, module:panel
client [949] disconnected, module:disp
client [949] disconnected, module:sys
[MI ERR ]: MI_SYS_IMPL_Exit[3821]: gSysInitCount:1
#
```

我们可以看到图片正常显示了。



PWM

LCD屏使用GPIO4作为背光控制，默认配置为pwm0。
测试方法：

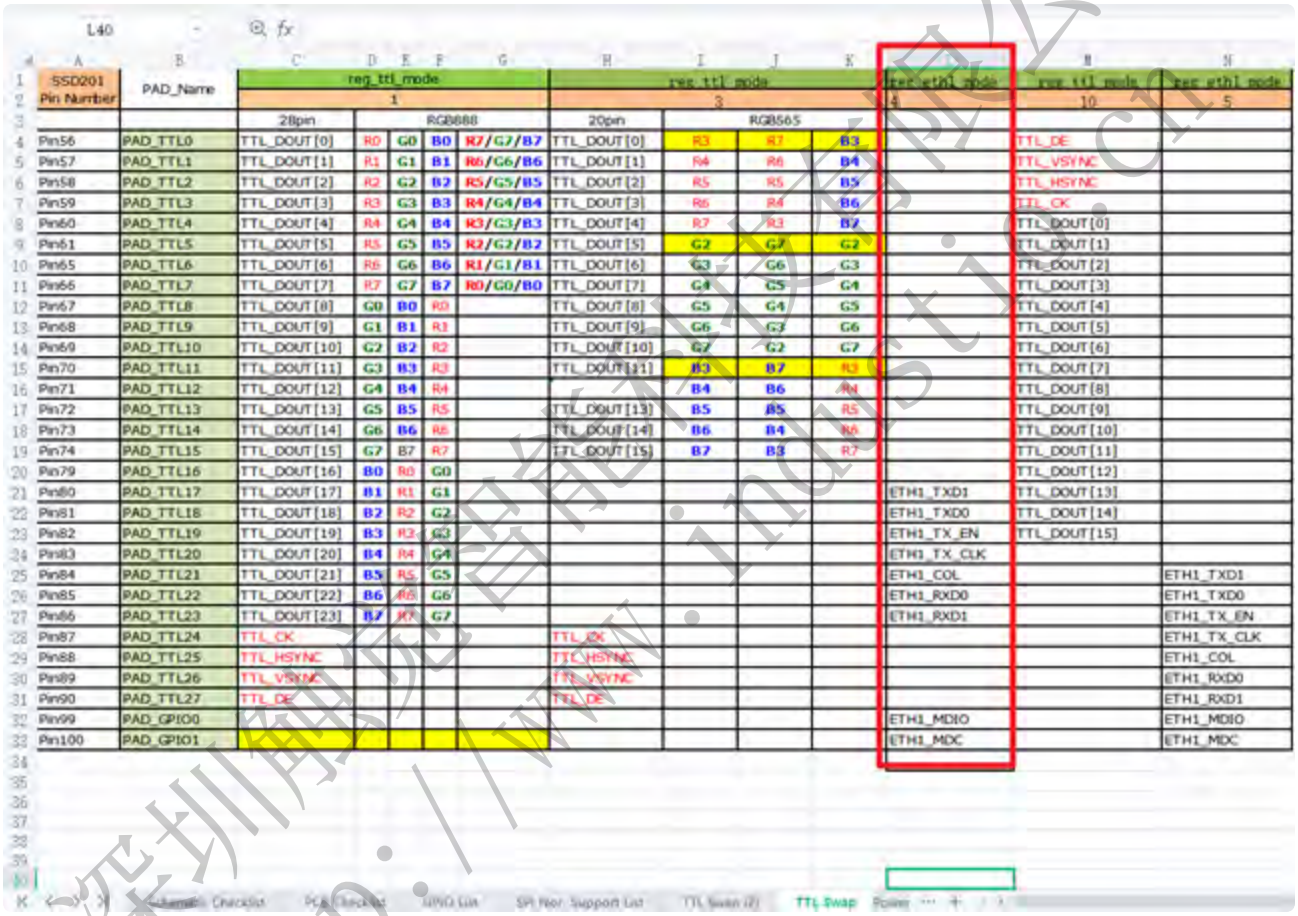
Plain Text | 复制代码

```
#echo 0 > /sys/class/pwm/pwmchip0/export
#echo 2000 > /sys/class/pwm/pwmchip0/pwm0/period
#echo 25 > /sys/class/pwm/pwmchip0/pwm0/duty_cycle
#echo 1 > /sys/class/pwm/pwmchip0/pwm0/enable
#echo 100 > /sys/class/pwm/pwmchip0/pwm0/duty_cycle
```

可以看到屏幕的背光根据参数的改变而变化。

以太网

使用infinity2m-spinand-ssc011a-s01a-rgb565-rmii.dts，根据原理图，ETH1使用PAD_TTL16-PAD_TTL23、PAD_GPIO0和PAD_GPIO1：



SSD201 Pin Number	PAD_Name	reg_ttl_mode 1					reg_ttl_mode 3				reg_ttl_mode 4	reg_ttl_mode 10	reg_ttl_mode 5
		28pin	RGB888				20pin	RGB565					
Pin56	PAD_TTL0	TTL_DOUT[0]	R0	G0	B0	R7/G7/B7	TTL_DOUT[0]	R3	R7	B3		TTL_DE	
Pin57	PAD_TTL1	TTL_DOUT[1]	R1	G1	B1	R6/G6/B6	TTL_DOUT[1]	R4	R6	B4		TTL_VSYNC	
Pin58	PAD_TTL2	TTL_DOUT[2]	R2	G2	B2	R5/G5/B5	TTL_DOUT[2]	R5	R5	B5		TTL_HSYNC	
Pin59	PAD_TTL3	TTL_DOUT[3]	R3	G3	B3	R4/G4/B4	TTL_DOUT[3]	R6	R4	B6		TTL_CK	
Pin60	PAD_TTL4	TTL_DOUT[4]	R4	G4	B4	R3/G3/B3	TTL_DOUT[4]	R7	R3	B7		TTL_DOUT[0]	
Pin61	PAD_TTL5	TTL_DOUT[5]	R5	G5	B5	R2/G2/B2	TTL_DOUT[5]	G2	G7	G2		TTL_DOUT[1]	
Pin65	PAD_TTL6	TTL_DOUT[6]	R6	G6	B6	R1/G1/B1	TTL_DOUT[6]	G3	G6	G3		TTL_DOUT[2]	
Pin66	PAD_TTL7	TTL_DOUT[7]	R7	G7	B7	R0/G0/B0	TTL_DOUT[7]	G4	G5	G4		TTL_DOUT[3]	
Pin67	PAD_TTL8	TTL_DOUT[8]	G0	B0	R0		TTL_DOUT[8]	G5	G4	G5		TTL_DOUT[4]	
Pin68	PAD_TTL9	TTL_DOUT[9]	G1	B1	R1		TTL_DOUT[9]	G6	G3	G6		TTL_DOUT[5]	
Pin69	PAD_TTL10	TTL_DOUT[10]	G2	B2	R2		TTL_DOUT[10]	G7	G2	G7		TTL_DOUT[6]	
Pin70	PAD_TTL11	TTL_DOUT[11]	G3	B3	R3		TTL_DOUT[11]	B3	B7	R3		TTL_DOUT[7]	
Pin71	PAD_TTL12	TTL_DOUT[12]	G4	B4	R4			B4	B6	R4		TTL_DOUT[8]	
Pin72	PAD_TTL13	TTL_DOUT[13]	G5	B5	R5		TTL_DOUT[13]	B5	B5	R5		TTL_DOUT[9]	
Pin73	PAD_TTL14	TTL_DOUT[14]	G6	B6	R6		TTL_DOUT[14]	B6	B4	R6		TTL_DOUT[10]	
Pin74	PAD_TTL15	TTL_DOUT[15]	G7	B7	R7		TTL_DOUT[15]	B7	B3	R7		TTL_DOUT[11]	
Pin79	PAD_TTL16	TTL_DOUT[16]	B0	R0	G0							TTL_DOUT[12]	
Pin80	PAD_TTL17	TTL_DOUT[17]	B1	R1	G1						ETH1_TXD1	TTL_DOUT[13]	
Pin81	PAD_TTL18	TTL_DOUT[18]	B2	R2	G2						ETH1_TXD0	TTL_DOUT[14]	
Pin82	PAD_TTL19	TTL_DOUT[19]	B3	R3	G3						ETH1_TX_EN	TTL_DOUT[15]	
Pin83	PAD_TTL20	TTL_DOUT[20]	B4	R4	G4						ETH1_TX_CLK		
Pin84	PAD_TTL21	TTL_DOUT[21]	B5	R5	G5						ETH1_COL		ETH1_TXD1
Pin85	PAD_TTL22	TTL_DOUT[22]	B6	R6	G6						ETH1_RXD0		ETH1_TXD0
Pin86	PAD_TTL23	TTL_DOUT[23]	B7	R7	G7						ETH1_RXD1		ETH1_TX_EN
Pin87	PAD_TTL24	TTL_CK					TTL_CK						ETH1_TX_CLK
Pin88	PAD_TTL25	TTL_HSYNC					TTL_HSYNC						ETH1_COL
Pin89	PAD_TTL26	TTL_VSYNC					TTL_VSYNC						ETH1_RXD0
Pin90	PAD_TTL27	TTL_DE					TTL_DE						ETH1_RXD1
Pin99	PAD_GPIO0										ETH1_MDIO		ETH1_MDIO
Pin100	PAD_GPIO1										ETH1_MDC		ETH1_MDC

结合SSD201 HW Checklist V6.xlsx，ETH1的MODE为4：

```
<PAD_GPIO0          PINMUX_FOR_ETH1_MODE_4      MDRV_FUSE_NA >,
<PAD_GPIO1          PINMUX_FOR_ETH1_MODE_4      MDRV_FUSE_NA >,
<PAD_TTL16          PINMUX_FOR_GPIO_MODE      MDRV_FUSE_ENAC1_PHY_RESET>,
<PAD_TTL17          PINMUX_FOR_ETH1_MODE_4      MDRV_FUSE_NA >,
<PAD_TTL18          PINMUX_FOR_ETH1_MODE_4      MDRV_FUSE_NA >,
<PAD_TTL19          PINMUX_FOR_ETH1_MODE_4      MDRV_FUSE_NA >,
<PAD_TTL20          PINMUX_FOR_ETH1_MODE_4      MDRV_FUSE_NA >,
<PAD_TTL21          PINMUX_FOR_ETH1_MODE_4      MDRV_FUSE_NA >,
<PAD_TTL22          PINMUX_FOR_ETH1_MODE_4      MDRV_FUSE_NA >,
<PAD_TTL23          PINMUX_FOR_ETH1_MODE_4      MDRV_FUSE_NA >,
```

其他默认配置即可。

测试

以eth0为例，在网口接口插上网线（另一端连接路由器），执行以下命令可对网口进行操作。

1、接上网线，dhcpcd自动获取IP地址。

```
# ps |grep dhcpcd
  736 root      /sbin/dhcpcd -f /etc/dhcpcd.conf
 1063 root      grep dhcpcd
```

```
# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:30:1B:BA:02:D1
          inet addr:192.168.31.70  Bcast:192.168.31.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7640 errors:0 dropped:0 overruns:0 frame:0
          TX packets:36 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:459359 (448.5 KiB)  TX bytes:2202 (2.1 KiB)
          Interrupt:34
```

尝试ping外网

Plain Text | 复制代码

```
1   # ping www.baidu.com
```

```
# ping www.baidu.com
PING www.baidu.com (14.215.177.38): 56 data bytes
64 bytes from 14.215.177.38: seq=0 ttl=55 time=7.566 ms
64 bytes from 14.215.177.38: seq=1 ttl=55 time=7.634 ms
64 bytes from 14.215.177.38: seq=2 ttl=55 time=7.434 ms
```

确认可以ping通

其它以太网的配置可以参考余下几点。

2、查看eth0 网卡

Plain Text | 复制代码

```
1   # ifconfig eth0
```

```
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:30:1B:BA:02:D1
          inet addr:192.168.31.70  Bcast:192.168.31.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7644 errors:0 dropped:0 overruns:0 frame:0
          TX packets:36 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:459599 (448.8 KiB)  TX bytes:2202 (2.1 KiB)
          Interrupt:34
```

3、关闭和开启网卡

Plain Text | 复制代码

```
1    #ifconfig eth0 up
2    #ifconfig eth0 down
```

4、设置静态IP地址

Plain Text | 复制代码

```
1   # ifconfig eth0 192.168.0.23
```

5、设置MAC地址

Plain Text | 复制代码

```
1   #ifconfig eth0 hw ether 36:72:C3:0A:FE:B3
```

6、设置子网掩码

Plain Text | 复制代码

```
1   # ifconfig eth0 netmask 255.255.255.0
```

7、设置广播地址

Plain Text | 复制代码

```
1   # ifconfig eth0 broadcast 192.168.0.255
```

8、网关添加和删除

```
# route add default gw 192.168.0.1
# route del default gw 192.168.0.1
```

9、设置DNS

添加和修改DNS，需要修改“/etc/resolv.conf”文件。

例：给开发板添加DNS “114.114.114.114”，操作方法如下所示

```
# vi /etc/resolv.conf
```

在文件最后添加一行nameserver 114.114.114.114

```
# Generated by dhcpcd from eth0.dhcp
# /etc/resolv.conf.head can replace this line
domain lan
nameserver 114.114.114.114

# /etc/resolv.conf.tail can replace this linea
```

10、手动动态获取IP地址

```
3 udhcpc -i eth0
```

11、配置静态IP

修改“/etc/network/interfaces”文件

```
3 vi /etc/network/interfaces
```

添加内容，设置eth0为静态IP，地址为192.168.0.19

```
#auto eth0
#iface eth0 inet static
#address 192.168.0.19
#netmask 255.255.255.0
#gateway 192.168.0.1
```

开启网卡服务

```
#/etc/init.d/S40network restart
```

```
/ # ifconfig
eth0      Link encap:Ethernet  HWaddr 00:30:1B:BA:02:DB
          inet addr:192.168.0.19  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:10258 errors:0 dropped:0 overruns:0 frame:0
          TX packets:88 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:790304 (771.7 KiB)  TX bytes:6762 (6.6 KiB)
          Interrupt:34

eth1      Link encap:Ethernet  HWaddr 00:30:1B:BA:02:DB
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:36

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

RTC

系统起来后，将看到/dev/rtc0设备节点

```
# ls /dev/rtc0
/dev/rtc0
#
#
```

验证

读取RTC时间：

```
1   # hwclock -r
```

```
# hwclock
Thu Jan  1 03:49:50 1970  0.000000 seconds
#
```

设置RTC时间：

```
1   # date -s "2021-03-03 00:00:00"
2   # hwclock -w
3   # hwclock -s
```

```
# date
Thu Jan  1 03:52:06 UTC 1970
# date
Thu Jan  1 03:52:10 UTC 1970
# date -s "2021-03-03 00:00:00"
Wed Mar  3 00:00:00 UTC 2021
# hwclock
Thu Jan  1 03:52:16 1970  0.000000 seconds
# hwclock  -w
# hwclock
Wed Mar  3 00:00:07 2021  0.000000 seconds
#
```

写入RTC时间

```
1   # hwclock -w
```

装上RTC电池，然后把开发板电源断开，并等待一段时间再接通电源，可以看到RTC在断电这段时间内是继续计时的。

```
1   # hwclock -r
```

```
xres: 1024,yres: 600
Freeing fb memory: 3072K
main 260 244

# date
Wed Mar  3 00:04:41 UTC 2021
#
#
#
# hwclock
Wed Mar  3 00:04:47 2021  0.000000 seconds
#
```

至此，内部RTC调试完成。

wifi

wifi是USB接口的（USB1）,系统启动后，通过lsusb可以看到1b20:8888的设备，它便是wifi模块。

```
# lsusb
Bus 002 Device 002: ID 1b20:8888
Bus 001 Device 001: ID 1d6b:0002
Bus 001 Device 002: ID 0930:6544
Bus 002 Device 001: ID 1d6b:0002
```

我们也可由此判断模块是否正常加载。

加载驱动

执行config/wifi/ssw01blnit_purple_pi.sh会自动加载驱动：

Plain Text | 复制代码

```
1  # /config/wifi/ssw01bInit_purple_pi.sh
```

```
# /config/wifi/ssw01bInit.sh
BANK:0xE 16bit-offset 0x30
0x0011
BANK:0x103C 16bit-offset 0x08
0x0001
BANK:0x103C 16bit-offset 0x08
0x0011
[Sstar_log]:Sstar_usb_module_init 0
[Sstar_log]:SVN_VER=1997,DPLL_CLOCK=2,BUILD_TIME=[===USB-ARES_B==
[Sstar_log]:Probe called -1 v1
```

驱动加载完后，我们便能看到wlan0网卡了：

```
# ifconfig wlan0
wlan0     Link encap:Ethernet  HWaddr 24:14:07:3C:9E:DB
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

STA模式

前面已经使用/config/wifi/ssw01blnit.sh加载了模块驱动，并且wlan0网卡已存在。现在我们通过wpa_supplicant工具（在/config/wifi目录下）来连接wifi热点。
修改/appconfigs/wpa_supplicant.conf，填入wifi热点信息：

```
Plain Text | 复制代码
1   # vi /appconfigs/wpa_supplicant.conf
```

```
# vi /appconfigs/wpa_supplicant.conf
# cat /appconfigs/wpa_supplicant.conf
ctrl_interface=/tmp/wifi/run/wpa_supplicant
update_config=1
network={
        scan_ssid=1
        ssid="HiWiFi_190B6A"
        psk="12345678"
}

#
```

当然，在连接之前先看看能否搜索到这个热点：

```
Plain Text | 复制代码
1   #  /config/wifi/iwlist wlan0 scan
```

```
Cell 05 - Address: D4:EE:07:19:0B:6A
          Channel:13
          Frequency:2.472 GHz (Channel 13)
          Quality=61/70  Signal level=-49 dBm
          Encryption key:on
          ESSID:"HiWiFi_190B6A"
          Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 9 Mb/s
                    18 Mb/s; 36 Mb/s; 54 Mb/s
          Bit Rates:6 Mb/s; 12 Mb/s; 24 Mb/s; 48 Mb/s
          Mode:Master
          Extra:tsf=7fffffffffffffff
          Extra: Last beacon: 560ms ago
          (Unknown Wireless Token 0x8C05)
```

可以看到已经搜索到这个热点，接下来尝试连接：

```
# /config/wifi/wpa_supplicant -D nl80211 -i wlan0 -c /appconfigs/wpa_supplicant.conf -B &
```

```
# /config/wifi/wpa_supplicant -D nl80211 -i wlan0 -c /appconfigs/wpa_supplicant.
conf -B &
# /config/wifi/wpa_supplicant: error while loading shared libraries: libnl.so.1: cannot open shared object file:

[1]+  Done(127)                 /config/wifi/wpa_supplicant -D nl80211 -i wlan0 -c /appconfigs/wpa_supplicant.c
```

提示没有相关库，这些库位于/config/wifi/目录下，我们配置一下LD_LIBRARY_PATH。

```
# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/config/wifi
 /config/wifi/wpa_supplicant -D nl80211 -i wlan0 -c /appconfigs/wpa_supplicant.conf -B &

```

```
# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/config/wifi
# /config/wifi/wpa_supplicant -D nl80211 -i wlan0 -c /appconfigs/wpa_supplicant.
conf -B &
# Successfully initialized wpa_supplicant
rfkill: Cannot open RFKILL control device
[Sstar_log]:ieee80211_mgd_auth:(d4:ee:07:19:0b:6a),ssid(HiWiFi_190B6A)
[Sstar_log]:Sstar_set_priv_queue_cap:[0],queue_cap[224]
[Sstar_log]:wlan0: authenticated
[Sstar_log]:ieee80211_start_connecting_work:bssid(d4:ee:07:19:0b:6a)
[Sstar_log]:wlan0: associated
[Sstar_log]:[d4:ee:07:19:0b:6a]:40M channel

[1]+  Done                    /config/wifi/wpa_supplicant -D nl80211 -i wlan0 -c /appconfigs/wpa_supplicant.conf -B
#
# [Sstar_log]:ieee80211_wk_connecting: time out
```

看起来是连接上了，但ifconfig发现没有分配到IP，这是因为没有dhcp服务：

```
# ifconfig wlan0
wlan0     Link encap:Ethernet  HWaddr 24:14:07:01:C4:ED
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:88 errors:0 dropped:5 overruns:0 frame:0
          TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:38354 (37.4 KiB)  TX bytes:284 (284.0 B)

#
```

这里我先手动给它设置一个IP，并测试是否能否和同一路由器下的设备通信：

```
# ifconfig wlan0 192.168.1.134
# ping 192.168.1.166
```

```
# ifconfig wlan0 192.168.1.134
[Sstar_log]:ieee80211_ifa_changed(wlan0):IPv4 enable,end_time(7197330)
# ifconfig wlan0
wlan0     Link encap:Ethernet  HWaddr 24:14:07:01:C4:ED
          inet addr:192.168.1.134  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:168 errors:0 dropped:8 overruns:0 frame:0
          TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:75715 (73.9 KiB)  TX bytes:284 (284.0 B)

# ping 192.168.1.166
PING 192.168.1.166 (192.168.1.166): 56 data bytes
64 bytes from 192.168.1.166: seq=0 ttl=128 time=57.124 ms
64 bytes from 192.168.1.166: seq=1 ttl=128 time=3.559 ms
64 bytes from 192.168.1.166: seq=2 ttl=128 time=10.598 ms
^C
--- 192.168.1.166 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 3.559/23.760/57.124 ms
#
```

可以看到能够正常通信，接下来给它设置DNS和网关，然后测试能否上网：

Plain Text | 复制代码

```
# route add default gw 192.168.1.1
# echo nameserver 114.114.114.114 > /etc/resolv.conf
# ping www.baidu.com
```

```
# route add default gw 192.168.1.1
# echo nameserver 114.114.114.114 > /etc/resolv.conf
# ping www.baidu.com
PING www.baidu.com (14.215.177.39): 56 data bytes
64 bytes from 14.215.177.39: seq=1 ttl=55 time=6.943 ms
64 bytes from 14.215.177.39: seq=2 ttl=55 time=7.827 ms
64 bytes from 14.215.177.39: seq=3 ttl=55 time=8.360 ms
```

好的，现在可以上网了。为了让它在连接热点时自动获取IP、DNS及网关，我们需要添加dhcp服务，当然，这个服务可以从buildroot获得：

Plain Text | 复制代码

```
industio@industio$: cd buildroot-2020.05/
industio@industio$: ARCH=arm make menuconfig
```

```
Target packages  --->
    Networking applications  --->
        [*] dhcpcd
```

这里我们默认配置好了，可以直接使用。系统启动后，可以看到dhcp服务已经开启了：

```
# ps | grep dhcp
  719 root      /sbin/dhcpcd -f /etc/dhcpcd.conf
  890 root      grep dhcp
#
```

接下来，开始wifi连接：

Plain Text 复制代码

```
# /config/wifi/ssw01bInit.sh
# ifconfig wlan0 up
# vi /appconfigs/wpa_supplicant.conf
# /config/wifi/wpa_supplicant -D nl80211 -i wlan0 -c /appconfigs/wpa_supplicant.conf -B &
# ifconfig wlan0
# route
# cat /etc/resolv.conf
# ping www.baidu.com
```

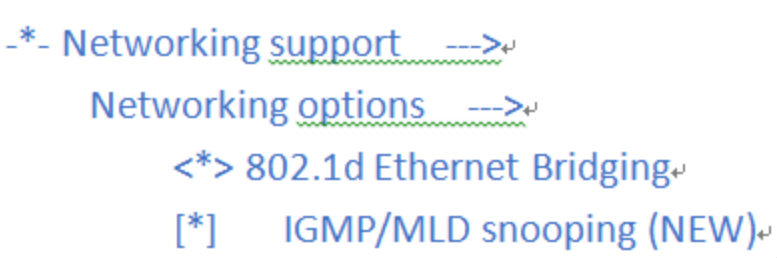
```
# ifconfig wlan0
wlan0     Link encap:Ethernet  HWaddr 24:14:07:01:C4:ED
          inet addr:192.168.1.230  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:78 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:24674 (24.0 KiB)  TX bytes:1641 (1.6 KiB)

# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         Hiwifi.lan      0.0.0.0         UG    304    0        0 wlan0
192.168.1.0     *               255.255.255.0   U     304    0        0 wlan0
# cat /etc/resolv.conf
# Generated by dhcpcd from wlan0.dhcp
# /etc/resolv.conf.head can replace this line
domain lan
nameserver 192.168.1.1
# /etc/resolv.conf.tail can replace this line
# ping www.baidu.com
PING www.baidu.com (14.215.177.39): 56 data bytes
64 bytes from 14.215.177.39: seq=0 ttl=55 time=224.612 ms
64 bytes from 14.215.177.39: seq=1 ttl=55 time=52.926 ms
64 bytes from 14.215.177.39: seq=2 ttl=55 time=19.693 ms
64 bytes from 14.215.177.39: seq=3 ttl=55 time=12.683 ms
64 bytes from 14.215.177.39: seq=4 ttl=55 time=7.048 ms
64 bytes from 14.215.177.39: seq=5 ttl=55 time=6.864 ms
```

可以看到dhcp服务已经正常工作了。

AP模式

首先需要配置kernel config：

```
industio@industio$:cd kernel
industio@industio$:ARCH=arm make menuconfig
```



每次修改配置后，更新一下defconfig：

```
industio@industio$: cp .config ./arch/arm/configs/infinity2m_spinand_ssc011a_s01a_minigui_double_net_defconfig -f
```

系统起来后，首先加载wifi驱动：

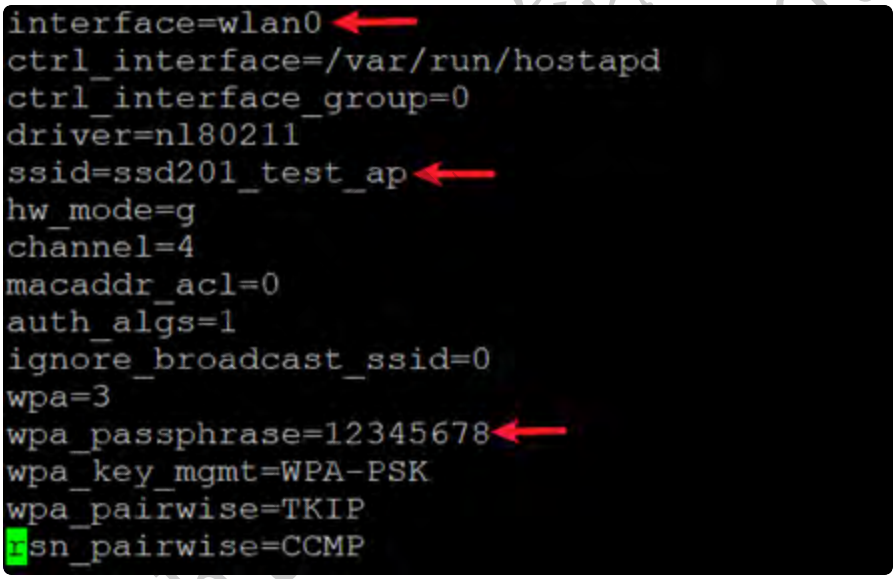
```
# /config/wifi/ssw01bInit.sh
# ifconfig wlan0 up
```

```
# ifconfig wlan0 up
# ifconfig p2p0 up
#
# ifconfig wlan0 0.0.0.0
# ifconfig p2p0 0.0.0.0
# brctl addbr br0
# brctl addif br0 wlan0
br0: port 1(wlan0) entered blocking state
br0: port 1(wlan0) entered disabled state
device wlan0 entered promiscuous mode
# brctl addif br0 p2p0
br0: port 2(p2p0) entered blocking state
br0: port 2(p2p0) entered disabled state
device p2p0 entered promiscuous mode
#
#
# ifconfig br0 up
```

当然，我们的WIFI模块作为AP热点，需要配置一下热点信息：

Plain Text | 复制代码

```
1  # vi /config/wifi/hostapd.conf
```

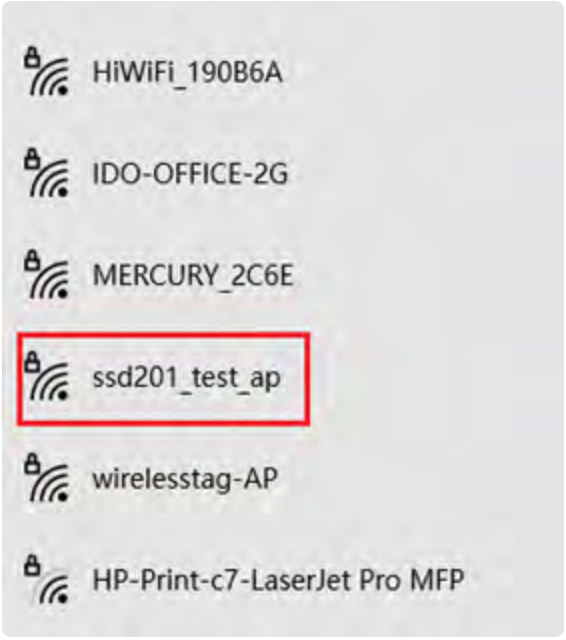


接下来，打开AP热点：

Plain Text | 复制代码

```
1  # /config/wifi/hostapd -B /config/wifi/hostapd.conf
```

此时，在手机/电脑上就可以搜索到我们的AP热点了：



尝试连接，发现一直在连接但并没有连接成功，这是因为没有开启DHCP服务，没有给连接设备分配到IP导致连接失败，所以我们还需要开启DHCP服务（使用dnsmasq工具）：

```
1   # vi /config/wifi/dnsmasq.conf
```

关注dhcp-range，它表示给配给设备的IP范围：

```
# repeat this for each network on which you want to supply DHCP
# service.
dhcp-range=192.168.0.101,192.168.0.200,12h

# This is an example of a DHCP range where the netmask is given. This
```

关注interface，这里把它设置为wlan0：

```
# interface (eg eth0) here.
# Repeat the line for more than one interface.
interface=wlan0
# Or you can specify which interface _not_ to listen on
#except-interface=
# Or which to listen on by address (remember to include 127.0.0.1 if
```

由于dhcp-range设置为192.168.1.x，因此wlan0的静态IP设置为192.168.0.1：

```
1   # ifconfig wlan0 192.168.0.1
```

此时，设备可以正常连接了，并且分配的IP位于dhcp-range范围内：

现在设备可以正常连接热点了，但此时我想连接设备能够上网，即把板子当作一个路由器，把eth0当作WAN，把wlan0当作LAN。

首先需要确认eth0是可以上网的：

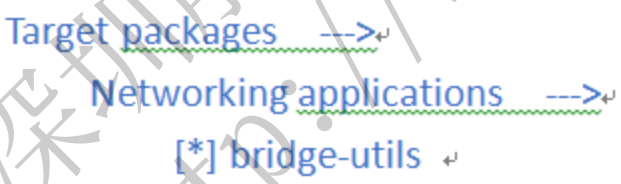
Plain Text | 复制代码

```
# ping www.baidu.com -I eth0
```

通过brctl桥接工具可以实现，此工具默认是没有安装的，和之前一样，从buildroot获取：

Plain Text | 复制代码

```
industio@industio$: cd buildroot-2020.05/
industio@industio$: ARCH=arm make menuconfig
```



```
industio@industio$: cp .config ./configs/ssd20x_defconfig -f
industio@industio$: make BR2_JLEVEL=4
industio@industio$: cd ../project/image/rootfs
industio@industio$: rm rootfs/* -rf
industio@industio$: cp ../../../buildroot-2020.05/output/images/rootfs.tar ./ -f
industio@industio$: tar -xvf rootfs.tar -C ./rootfs/
industio@industio$: tar -cvf rootfs.tar.gz ./rootfs
industio@industio$: cd ../../../
```

重新编译并更新固件：

```
industio@industio$: ./Release_to_customer.sh  -f nand -p ssd202  -m 256
```

更新固件后，前面的加载驱动、hostapd服务和dnsmasq服务需要重新执行，然后执行以下命令建立桥接：

```
# brctl addbr br0
# brctl addif br0 wlan0
# brctl addif br0 eth0
# ifconfig br0 up
```

此时，连接设备就可以正常上网了：

双排针

双排针接口说明列表

序号	丝印	默认模式/功能	节点	序号	丝印	默认模式/功能	节点
1	3V3			2	VCC5V		
3	I2C0_SDA	IIC-0	/dev/i2c-0	4	VCC5V		
5	I2C0_SCL			6	GND		
7	GPIO13		/sys/class/gpio/gpio13（默认作为TP 中断）	8	NC		
9	GND			10	NC		
11	GPIO12		/sys/class/gpio/gpio12（默认作为TP 复位）	12	NC		
13	GPIO47		/sys/class/gpio/gpio47	14	GND		
15	GPIO48		/sys/class/gpio/gpio48	16	GPIO50		/sys/class/gpio/gpio50
17	3V3			18	GPIO49		/sys/class/gpio/gpio49
19	SPI0_DO	spi0	/dev/spidev0.0	20	GND		
21	SPI0_DI			22	NC		
23	SPI0_CK			24	SPI0_CZ	spi0	/dev/spidev0.0
25	GNG			26	GPIO5		/sys/class/gpio/gpio5

27	NC			28	NC		
29	GPIO16		/sys/class/gpio/gpio16	30	GND		
31	GPIO15		/sys/class/gpio/gpio15	32	GPIO73		/sys/class/gpio/gpio47
33	GPIO17		/sys/class/gpio/gpio17	34	GND		
35	GPIO18		/sys/class/gpio/gpio18	36	GPIO87		/sys/class/gpio/gpio87
37	GPIO59		/sys/class/gpio/gpio59	38	GPIO88		/sys/class/gpio/gpio88
39	GND			40	GPIO89		/sys/class/gpio/gpio89

SPI

接口定义列表：

GPIO8	SPI0_CZ
GPIO9	SPI0_CK
GPIO10	SPI0_DI
GPIO11	SPI0_DO

接口	节点
SPI0	/dev/spidev0.0

开发板默认配置好了SPI。我们可以通过回环测试，确定SPI是否能正常使用。

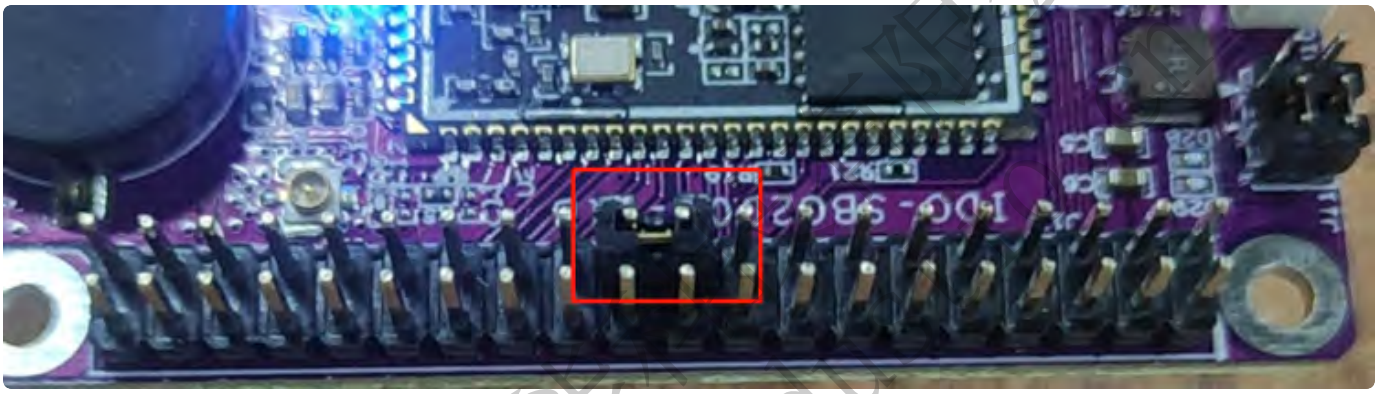
查看是否有spi节点生成：

```
# ls /dev/spi*
```

```
# ls /dev/spi*
/dev/spidev0.0
#
```

可以看到spi节点已经出来了。

使用跳线帽短接GPIO10—GPIO11（短接MISO和MOSI）进行回环测试。



测试源码：

spi_test.c

Plain Text | 复制代码

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdint.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <strings.h>
#include <string.h>
#include "../../project/kbuild/4.9.84/i2m/include/uapi/linux/spi/spidev.h"


static const char *device = "/dev/spidev0.0";
static uint8_t mode = 0; /* SPI 通信使用全双工，设置 CPOL=0，CPHA=0。 */
static uint8_t bits = 8; /* 8bits读写，MSB first。*/
static uint32_t speed = 12*1000*1000;/* 设置传输速度 */
static uint16_t delay = 0;
static int g_SPI_Fd = 0;

#define SPI_DEBUG 1

static void pabort(const char *s)
{
    perror(s);
    abort();
}

int SPI_Transfer(const uint8_t *TxBuf, uint8_t *RxBuf, int len)
{
    int ret;
    int fd = g_SPI_Fd;
    struct spi_ioc_transfer tr ={
        .tx_buf = (unsigned long) TxBuf,
        .rx_buf = (unsigned long) RxBuf,
        .len =len,
        .delay_usecs = delay,
    };
    ret = ioctl(fd, SPI_IOC_MESSAGE(1), &tr);
    if (ret < 1)
        perror("can't send spi message\n");
    else
    {
#if SPI_DEBUG
```

```
        int i;
        printf("nsend spi message Succeed\n");
        printf("nSPI Send [Len:%d]: \n", len);
        for (i = 0; i < len; i++)
        {
            if (i % 8 == 0)
                printf("nt\n");
            printf("0x%02X \n", TxBuf[i]);
        }
        printf("n");
        printf("SPI Receive [len:%d]:\n", len);
        for (i = 0; i < len; i++)
        {
            if (i % 8 == 0)
                printf("nt\n");
            printf("0x%02X \n", RxBuf[i]);
        }
#endif
    }
    return ret;
}

int SPI_Write(uint8_t *TxBuf, int len)
{
    int ret;
    int fd = g_SPI_Fd;
    ret = write(fd, TxBuf, len);
    if (ret < 0)
        perror("SPI Write error\n");
    else
    {
#if SPI_DEBUG
        int i;
        printf("SPI Write [Len:%d]: \n", len);
        for (i = 0; i < len; i++)
        {
            if (i % 8 == 0)
                printf("\n\t");
            printf("0x%02X \n", TxBuf[i]);
        }
        printf("\n");
#endif
    }
    return ret;
}

int SPI_Read(uint8_t *RxBuf, int len)
{
```

```
    int ret;
    int fd = g_SPI_Fd;
    ret = read(fd, RxBuf, len);
    if (ret < 0)
        printf("SPI Read error\n");
    else
    {
#if SPI_DEBUG
        int i;
        printf("SPI Read [len:%d]:\n", len);
        for (i = 0; i < len; i++)
        {
            if (i % 8 == 0)
                printf("\n\t");
            printf("0x%02X \n", RxBuf[i]);
        }
        printf("\n");
#endif
    }
    return ret;
}

int SPI_Open(void)
{
    int fd;
    int ret = 0;
    if (g_SPI_Fd != 0) /* 设备已打开 */
        return 0xF1;
    fd = open(device, O_RDWR);
    if (fd < 0)
        pabort("can't open device\n");
    else
        printf("SPI - Open Succeed. Start Init SPI...\n");
    g_SPI_Fd = fd;

    ret = ioctl(fd, SPI_IOC_WR_MODE, &mode);
    if (ret == -1)
        pabort("can't set spi mode\n");
    ret = ioctl(fd, SPI_IOC_RD_MODE, &mode);
    if (ret == -1)
        pabort("can't get spi mode\n");
    /*
     * bits per word
     */
    ret = ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits);
    if (ret == -1)
        pabort("can't set bits per word\n");
    ret = ioctl(fd, SPI_IOC_RD_BITS_PER_WORD, &bits);
```

```
    if (ret == -1)
        pabort("can't get bits per word\n");
    /*
     * max speed hz
     */
    ret = ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
    if (ret == -1)
        pabort("can't set max speed hz\n");
    ret = ioctl(fd, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
    if (ret == -1)
        pabort("can't get max speed hz\n");
    printf("spi mode: %d\n", mode);
    printf("bits per word: %d\n", bits);
    printf("max speed: %d KHz (%d MHz)\n", speed / 1000, speed / 1000 / 1
000);
    return ret;
}

int SPI_Close(void)
{
    int fd = g_SPI_Fd;
    if (fd == 0) /* SPI 是否已经打开*/
        return 0;
    close(fd);
    g_SPI_Fd = 0;
    return 0;
}

int SPI_LookBackTest(void)
{
    int ret, i;
    const int BufSize = 16;
    uint8_t tx[BufSize], rx[BufSize];
    bzero(rx, sizeof(rx));
    for (i = 0; i < BufSize; i++)
        tx[i] = i;
    printf("nSPI - LookBack Mode Test...\n");
    ret = SPI_Transfer(tx, rx, BufSize);
    if (ret > 1)
    {
        ret = memcmp(tx, rx, BufSize);
        if (ret != 0)
        {
            printf("tx:\n");
            for (i = 0; i < BufSize; i++)
            {
                printf("%d ", tx[i]);
            }
```

```
188             printf("\n");
189             printf("rx:\n");
190             for (i = 0; i < BufSize; i++)
191             {
192                 printf("%d ", rx[i]);
193             }
194             printf("\n");
195
```

将源码拷贝到Ubuntu虚拟机，下进行编译，生成执行文件spi_test。

Plain Text 复制代码

```
1   industio@industio$:arm-linux-gnueabihf-gcc spi_test.c -o spi_test -lpthread
```

将执行文件spi_test拷贝到开发板执行。

```
# ./spi_test
SPI - Open Succeed. Start Init SPI...
spi mode: 0
bits per word: 8
max speed: 12000 KHz (12 MHz)
************************* ret = 0************************
nSPI - LookBack Mode Test...
nsend spi message Succeed
nSPI Send [Len:16]:
nt
0x00
0x01
0x02
0x03
0x04
0x05
0x06
0x07
nt
0x08
0x09
0x0A
0x0B
0x0C
0x0D
0x0E
0x0F
nSPI Receive [len:16]:
nt
0x00
0x01
0x02
0x03
0x04
0x05
0x06
0x07
nt
0x08
0x09
0x0A
0x0B
0x0C
0x0D
0x0E
0x0F
SPI - LookBack Mode OK
```

I2C

I2C接口定义与节点列表：

接口	节点	备注
i2c0	/dev/i2c-0	gpio6、gpio7
i2c1	/dev/i2c-1	gpio2、gpio3

IO	配置	备注
GPIO6	I2C0_SCL	
GPIO7	I2C0_SDA	
GPIO2	I2C1_SCL	目前用于屏幕TP使用
GPIO3	I2C1_SDA	

I2C测试需要外接I2C设备，这里我们以TP为例，通过i2cdetect进行测试。
开发板中输入：

```
#i2cdetect -r -y 1 >> i2c.log
```

通过设备号确认识别i2c设备：

```
#cat i2c.log
```

```
# cat i2c.log
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- UU -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
#
```

可以看到识别到了TP，TP的设备地址：0x5d。

GPIO

可通过以下命令控制GPIO,例如GPIO14:

申请gpio：

```
#echo 14 > /sys/class/gpio/export
```

设置为输出：

```
#echo out > /sys/class/gpio/gpio14/direction
```

设置为输入：

```
#echo in > /sys/class/gpio/gpio14/direction
```

输出高电平：

```
#echo 1 > /sys/class/gpio/gpio14/value
```

输出低电平：

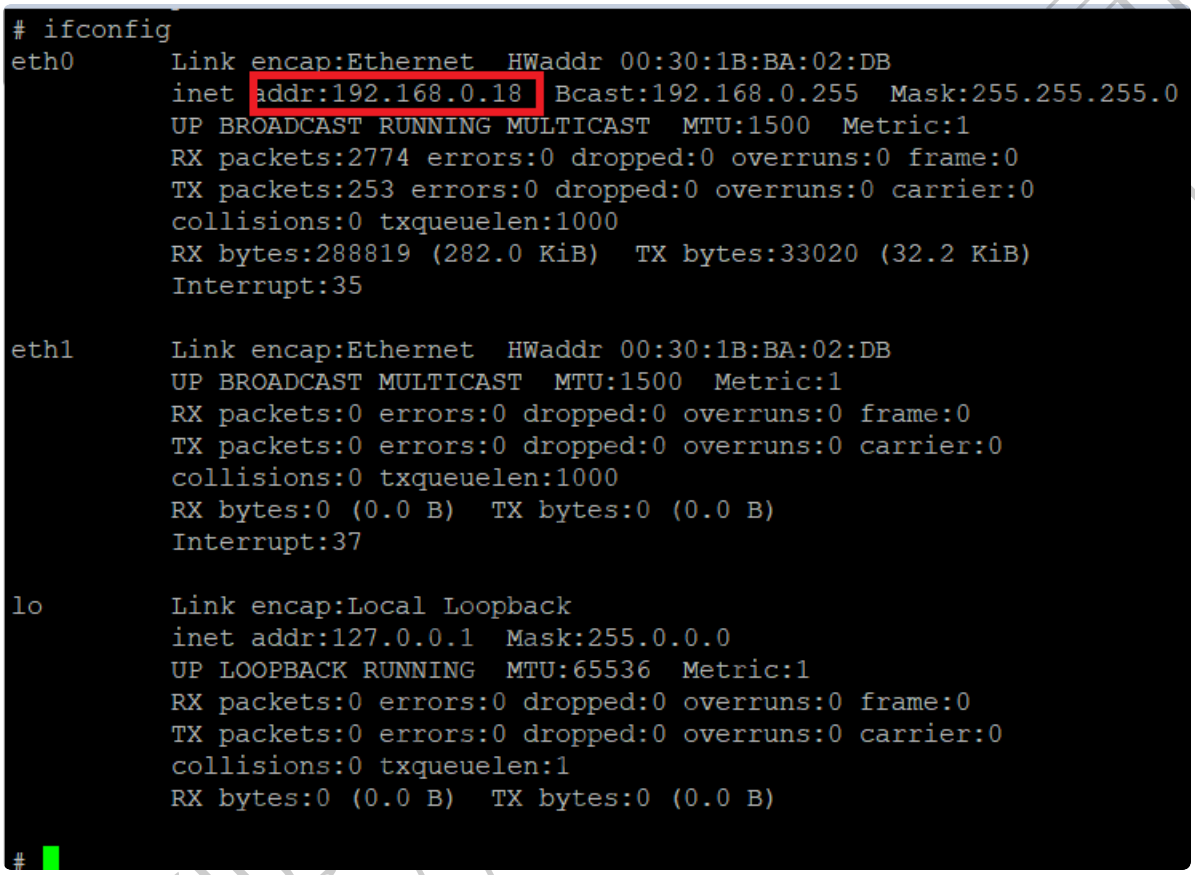
```
#echo 0 > /sys/class/gpio/gpio14/value
```

获取输入电平（0：低电平，1：高电平）

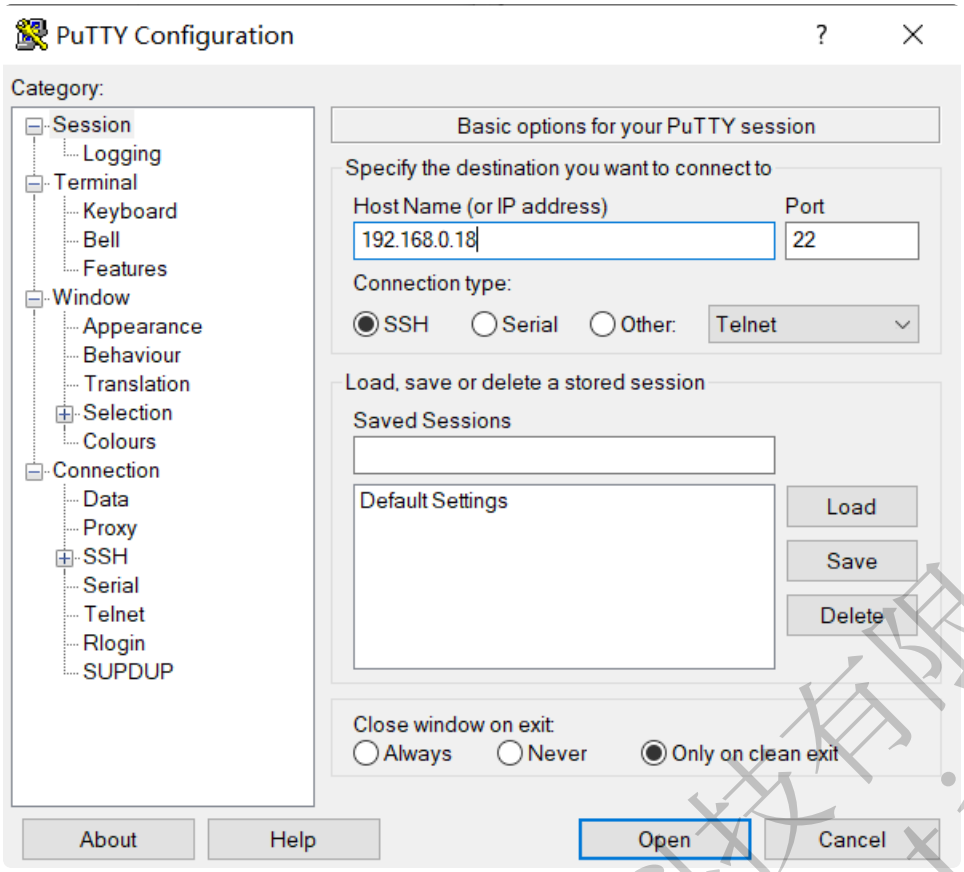
```
1   #cat /sys/class/gpio/gpio14/value
```

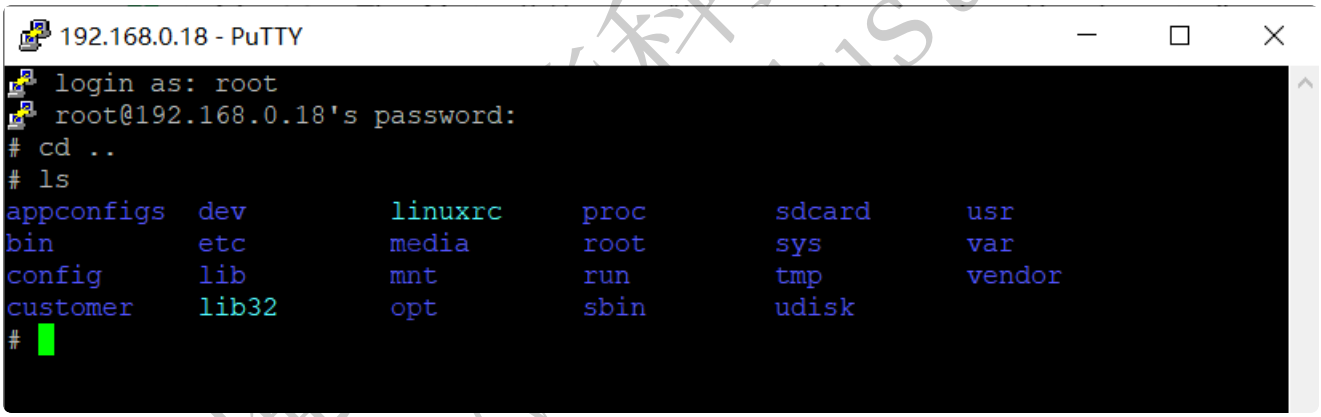
SSH

在buildroot中已经默认配置了SSH，不需要重新配置，可以直接使用，使用前需要知道开发板的IP地址



打开软件（putty,secureCRT，CMD等），账户名：root，密码：123456





NFS

在Ubuntu虚拟机下安装NFS服务

```
industio@industio$:sudo apt-get install nfs-kernel-server
```

Ubuntu虚拟机下准备NFS文件系统（如果不是使用提供的Ubuntu镜像，需要把源码复制到自己的Ubuntu虚拟机上）

```
industio@industio$:cd ~/ssd20x
industio@industio$:mkdir nfs
industio@industio$:cd  nfs
industio@industio$:cp ../project/image/output/rootfs/* ./ -rf
industio@industio$:cp ../project/image/output/customer/ ./ -rf
industio@industio$:cp ../project/image/output/appconfigs/ ./ -rf
industio@industio$:cp ../project/image/output/miservice/config/ ./ -rf
```

在Ubuntu虚拟机开启NFS服务

```
industio@industio$:sudo vi /etc/exports
//如果提示没有文档，请查看nfs服务是否安装。进入文件在后面添加以下内容
/home/industio/ssd20x/nfs *(rw,sync)

```

```
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#
/home/industio/ssd20x/nfs/ *(rw,sync)
```

如果上面的步骤没有问题，接下来重启nfs服务

```
industio@industio$:sudo /etc/init.d/nfs-kernel-server restart
```

NFS服务重启成功

```
root@industio:/home/industio/ssd20x/nfs# sudo /etc/init.d/nfs-kernel-server restart
[ ok ] Restarting nfs-kernel-server (via systemctl): nfs-kernel-server.service.
root@industio:/home/industio/ssd20x/nfs# _
```

本地验证

```
//使用ifconfig查看Ubuntu IP地址，Ubuntu IP地址为192.168.0.17
industio@industio$:sudo mount -t nfs -o nolock 192.168.0.17:/home/industio/ssd20x/nfs/ /mnt
//挂载成功查看目录是否与之前准备的文件系统一样
industio@industio$:ls /mnt
//使用umount命令卸载当前挂载的文件系统
industio@industio$:sudo umount /mnt
```

```
industio@industio:~/ssd20x$ ls /mnt
appconfigs  disp_init  logo        opt   sdcard       sys     vendor
bin         etc        logo.jpg    proc  snd.ko       tmp
config      lib        media       root  snd-pcm.ko   udisk
customer    lib32      mi_alsa.ko  run   snd-timer.ko usr
dev         linuxrc    mnt         sbin  soundcore.ko var
industio@industio:~/ssd20x$
```

开发板挂载NFS

确认开发板与Ubuntu虚拟机处于同一局域网内，并且能互相通信。

Ubuntu虚拟机的IP地址如下：

```
root@industio:/home/industio/ssd20x/nfs# ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.17  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::20c:29ff:fef8:c1ba  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:f8:c1:ba  txqueuelen 1000  (Ethernet)
        RX packets 33998  bytes 11812269 (11.8 MB)
        RX errors 0  dropped 53  overruns 0  frame 0
        TX packets 21762  bytes 11628272 (11.6 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 146  bytes 10912 (10.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 146  bytes 10912 (10.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@industio:/home/industio/ssd20x/nfs# _
```

开发板ping Ubuntu虚拟机的IP正常通信

```
# ping 192.168.0.17
PING 192.168.0.17 (192.168.0.17): 56 data bytes
64 bytes from 192.168.0.17: seq=0 ttl=64 time=1.050 ms
64 bytes from 192.168.0.17: seq=1 ttl=64 time=0.902 ms
64 bytes from 192.168.0.17: seq=2 ttl=64 time=0.846 ms
64 bytes from 192.168.0.17: seq=3 ttl=64 time=0.827 ms
```

```
//开发板挂载NFS文件系统
#mount -t nfs -o nolock 192.168.0.17:/home/industio/ssd20x/nfs/ /mnt
#ls /mnt
#umount /mnt
```

```
# mount -t nfs -o nolock 192.168.0.17:/home/industio/ssd20x/nfs/ /mnt
# ls /mnt
appconfigs  dev         linuxrc     proc        sdcard      usr
bin         etc         media       root        sys         var
config      lib         mnt         run         tmp         vendor
customer    lib32       opt         sbin        udisk
#
```

将NFS编译到kernel

```
industio@industio$:cd /kernel
industio@industio$:ARCH=arm make menuconfig
```

按照下面的步骤添加nfs

```
-*- Networking support  --->
        Networking options  --->
            -*- TCP/IP networking
            [*] IP: kernel level autoconfiguration
            [*]     IP: DHCP support
            [*]     IP: BOOTP support
File systems  --->
    [*] Network File Systems  --->
        <*>   NFS client support
        <*>       NFS client support for NFS version 2
        <*>       NFS client support for NFS version 3
        <*>       NFS client support for NFS version 4
        [*]   Root file system on NFS
```

保存退出之后需要把原来的_deconfig文件覆盖，查看Release_to_customer.sh可以知道Purple Pi是使用以下文件

```
if [ "${project}" = "2D06" ]; then
#       KERNEL_DEFCONFIG=infinity2m_spinand_ssc011a_s01a_display_for_mipi_defconfig
        KERNEL_DEFCONFIG=infinity2m_spinand_ssc011a_s01a_minigui_doublenet_defconfig
```

所以需要在kernel目录下使用命令覆盖原来文件：

```
industio@industio$:cp .config infinity2m_spinand_ssc011a_s01a_minigui_doublenet_defconfig
```

重新编译并更新kernel

```
industio@industio$:./Release_to_customer.sh -f nand -p ssd201 -o 2D06
```

编译完成需要重新更新kernel，步骤查看烧录流程中的更新uboot和kernel
更新kernel之后，板子上电回车进入uboot，重新设置bootargs，这里需要将bootargs设置根文件系统指向NFS，这里的192.168.0.17是Ubuntu虚拟机的IP，192.168.0.85为同一路由下的随机IP，192.168.0.1是路由的IP（由自身的IP而设置），在这里设置指定nfs版本为v3，详细内容如下：

```
#setenv bootargs console=ttyS0,115200 root=/dev/nfs rw nfsroot=192.168.0.17:/home/industio/ssd20x/nfs,v3,nolock ip=192.168.0.85:192.168.0.17:192.168.0.1:255.255.255.0::eth0:off init=/linuxrc rootwait=1 LX_MEM=0x3f00000 mma_heap=mma_heap_name0,miu=0,sz=0xa00000 mma_memblock_remove=1 highres=off mmap_reserved=fb,miu=0,sz=0x300000,max_start_off=0x3300000,max_end_off=0x3600000 mtdparts=nand0:384k@1280k(IPL0),384k(IPL1),384k(IPL_CUST0),384k(IPL_CUST1),768k(UBOOT0),768k(UBOOT1),256k(ENV),256k(ENV1),0x20000(KEY_CUST),0x60000(LOGO),0x500000(KERNEL),0x500000(RECOVERY),-(UBI)
//保存
#saveenv
//重新启动
#reset
```

重新启动后，当加载到以下信息，说明了nfs挂载rootfs成功

```
IP-Config: Complete:
     device=eth0, hwaddr=00:30:1b:ba:02:db, ipaddr=192.168.0.85, mask=255.255.255.0, gw=192.168.0.1
     host=192.168.0.85, domain=, nis-domain=(none)
     bootserver=192.168.0.17, rootserver=192.168.0.17, rootpath=
VFS: Mounted root (nfs filesystem) on device 0:13.
devtmpfs: mounted
This architecture does not have kernel memory protection.
random: fast init done
Starting syslogd: OK
Starting klogd: OK
Running sysctl: OK
Starting mdev... OK
modprobe: can't open 'modules.dep': No such file or directory
Saving random seed: SKIP (read-only file system detected)
Starting haveged: haveged: listening socket at 3
OK
```

```
# mount
192.168.0.17:/home/industio/ssd20x/nfs on / type nfs (rw,relatime,vers=3,rsize=4096,wsize=4096,namlen=255,hard,nolock,proto=udp,timeo=11,retrans=3,sec=sys,mountaddr=192.168.0.17,mountvers=3
,mountproto=udp,local_lock=all,addr=192.168.0.17)
devtmpfs on /dev type devtmpfs (rw,relatime,size=21472k,nr_inodes=5368,mode=755)
proc on /proc type proc (rw,relatime)
devpts on /dev/pts type devpts (rw,relatime,gid=5,mode=620,ptmxmode=666)
tmpfs on /dev/shm type tmpfs (rw,relatime,size=22496k,nr_inodes=5624,mode=777)
tmpfs on /tmp type tmpfs (rw,relatime,size=22496k,nr_inodes=5624)
tmpfs on /run type tmpfs (rw,nosuid,nodev,relatime,size=22496k,nr_inodes=5624,mode=755)
sysfs on /sys type sysfs (rw,relatime)
mdev on /dev type tmpfs (rw,relatime,size=22496k,nr_inodes=5624)
none on /sys/kernel/debug type debugfs (rw,relatime)
devpts on /dev/pts type devpts (rw,relatime,mode=600,ptmxmode=000)
```

开机自启

buildroot根文件系统已经在/etc/init.d/中添加了一个自启脚本，如需要添加开机自启，添加一个脚本命令必须为S开头

Plain Text | 复制代码

```
#!/bin/sh


# Start all init scripts in /etc/init.d
# executing them in numerical order.
#
for i in /etc/init.d/S??* ;do

     # Ignore dangling symlinks (if any).
     [ ! -f "$i" ] && continue

     case "$i" in
        *.sh)
            # Source shell script for speed.
            (
                trap - INT QUIT TSTP
                set start
                . $i
            )
            ;;
        *)
            # No sh extension, so fork subprocess.
            $i start
            ;;
    esac
done

export PATH=$PATH:/config
export TERMINFO=/config/terminfo
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/config/lib:/config/wifi
mkdir -p /dev/pts
mount -t sysfs none /sys
mount -t tmpfs mdev /dev
mount -t debugfs none /sys/kernel/debug/
mdev -s
mkdir -p /var/lock
mount -t ubifs ubi0:miservice /config
 mount -t ubifs ubi0:customer /customer
 mount -t ubifs ubi0:appconfigs /appconfigs

mkdir -p /dev/pts
mount -t devpts devpts /dev/pts
busybox telnetd&
echo 85 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio85/direction
```

```
echo 1 > /sys/class/gpio/gpio85/value
echo 86 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio86/direction
echo 1 > /sys/class/gpio/gpio86/value
if [ -e /etc/core.sh ]; then
    echo "|/etc/core.sh %p" > /proc/sys/kernel/core_pattern
chmod 777 /etc/core.sh
fi;
if [ -e /customer/demo.sh ]; then
    /customer/demo.sh
fi;
```

例如在同一目录下的命名方式

```
# ls
S01syslogd  S02sysctl   S20urandom  S40network  S50sshd     rcS
S02klogd    S10mdev     S21haveged  S41dhcpcd   rcK
```

添加完脚本后给脚本设置权限即可

Plain Text | 复制代码

```
//根据自己命名的实际情况
#chmod 777 Sxx
```

例如：使用nfs开机自动挂载文件系统到/mnt目录下

Plain Text | 复制代码

```
#touch S51nfs
#chmod 777 S51nfs
```

```
# ls
S01syslogd  S02sysctl   S20urandom  S40network  S50sshd     rcK
S02klogd    S10mdev     S21haveged  S41dhcpcd   S51nfs      rcS
```

Plain Text | 复制代码

```
#!/bin/sh
#
#mount nfs
#
mount -t nfs -o nolock 192.168.0.17:/home/industio/ssd20x/nfs/ /mnt
echo "mount ok!"
```

重新开机之后就会看到自动挂载的到/mnt目录下

```
# ls /mnt/
appconfigs  dev         linuxrc     proc        sdcard      usr
bin         etc         media       root        sys         var
config      lib         mnt         run         tmp         vendor
customer    lib32       opt         sbin        udisk
```