NAMTSO

Android Development Guide

1. Build the compilation environment

Compiling Android requires a high level of machine configuration:

- 64-bit CPU
- 32GB physical memory + swap memory
- 250 GB free disk spaces

It is recommended to use Ubuntu 18.04 operating system or later, Ubuntu 16.04 or 18.04 package installation reference:

```
$ sudo apt-get update
```

```
$ sudo apt-get install git gnupg flex bison gperf libsdl1.2-dev make gawk \
libesd -java libwxgtk3.0-dev squashfs -tools build-essential zip curl minicom \
libncurses5-dev zlib1g-dev pngcrush schedtool libxml2 libxml2-utils exfat -fuse exfat -utils \
xsltproc lzop libc6-dev schedtool g++- multilib lib32z1-dev lib32ncurses5-dev \
lib32readline-dev gcc-multilib libswitch-perl libssl -dev unzip zip device-tree-compiler \
liblz4-tool python- pyelftools python3-pyelftools python- networkx openjdk-8-jdk -y
```

2. Download Android-13 SDK

a) To get the source code, install the git-lfs tool to pull large files:

```
$ sudo apt install git- lfs
```

b) Create an empty directory to hold the working files:

```
$ mkdir -p WORKING_DIRECTORY
$ cd WORKING_DIRECTORY
```

c) Run repo-init to download the inventory repository:

```
$ repo init -u https://github.com/namtso2207/android_manifest.git -b namtso-a10-android13
```

d) Run repo-sync to pull the Android source tree:

```
$ repo sync -j4
```

e) Go to the following directory and execute the git-lfs-pull command to pull a large file, otherwise the compile will report an error:

```
/users/a10-3588$ cd external/ camera_engine_rkaiq
/users/a10-3588/external/ camera_engine_rkaiq $ git lfs pull
Git LFS: (4 of 4 files) 208.97 MB / 208.97 MB
/users/a10-3588/external/ camera_engine_rkaiq $ cd -
/users/a10-3588
/users/a10-3588$ cd device/ namtso /rk3588
/users/a10-3588/device/ namtso /rk3588$ git lfs pull
Git LFS: (1 of 1 files) 211.07 MB / 211.07 MB
```

f) Start a new development branch

```
$ repo start <BRANCH_NAME> --all
```

3. Compile the source code

a) Complete compilation:

```
$ cd PATH_YOUR_PROJECT
$ source build/envsetup.sh
$ lunch a10_3588-userdebug
$ ./build.sh - AUCKu
```

b) Step by step compilation:

Compile U-Boot:

```
$ cd PATH_YOUR_PROJECT
$ source build/envsetup.sh
$ lunch a10_3588-userdebug
$ ./build.sh -U
```

Compile the Kernel:

```
$ cd PATH_YOUR_PROJECT
$ source build/envsetup.sh
$ lunch a10_3588-userdebug
$ ./build.sh -CK
```

Compile Android:

```
$ cd PATH_YOUR_PROJECT
$ source build/envsetup.sh
$ lunch a10_3588-userdebug
$ ./build.sh -A
```

You can change the file build. sh build _ JOBS = 16 to the desired number.

Package update.img:

```
$ cd PATH_YOUR_PROJECT
$ source build/envsetup.sh
$ lunch a10_3588-userdebug
$ ./build.sh -u
```

After compilation, the update.img needs to be reloaded when burning.

After a complete compilation, the following file will be generated:

```
rockdev /Image-a10_3588/
├──── boot- debug.img
├──── boot.img
├──── config.cfg
├──── dtbo.img
├──── MiniLoaderAll.bin
├──── misc.img
├──── parameter.txt
├──── pcba_small_misc.img
├──── pcba_whole_misc.img
├──── recovery.img
├──── resource.img
├──── super.img
├──── uboot.img
├──── update.img
└──── vbmeta.img
```

4. Firmware programming

4.1. Firmware acquisition

https://dl.namtso.com/products/a10/a10-3588/firmware/

4.2. Upgrades firmware using usb cable in Windows

a) Prepare the tools

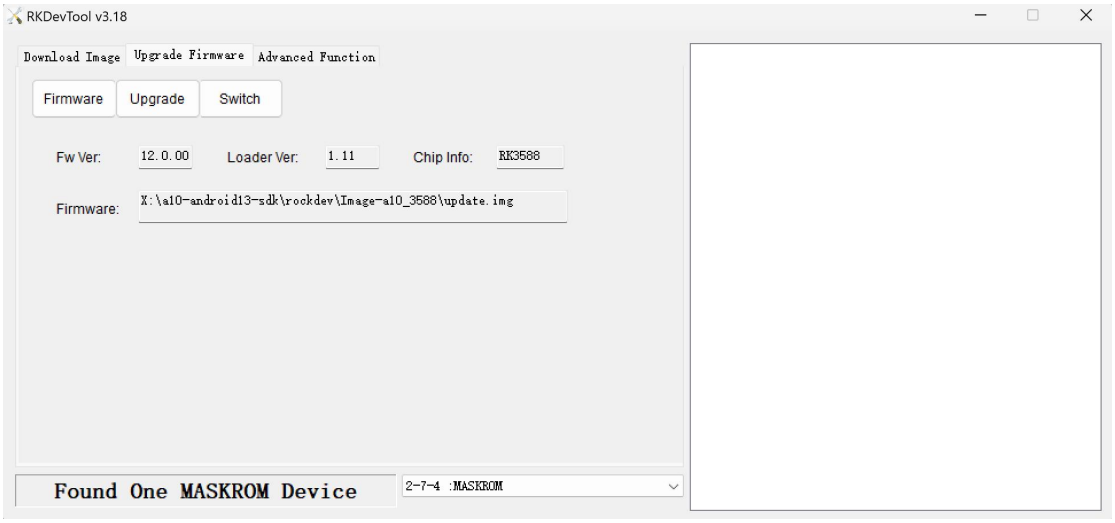
- ✓ A10-3588 Development Board
- ✓ The firmware
- ✓ Host
- ✓ Good Type-C cables

b) Install the programming tool

Install the RK USB driver; the Android 13 USB driver DriverAssitant needs to be updated to V5.1.1; download driver-assitant_v5.12.zip, decompress it, and then run the DriverInstall. Exe inside; In order for all devices to use the updated driver, select Driver Uninstall first, and then select Driver

Install.https://dl.namtso.com/products/a10/a10-3588/tools/windows-burn-tool/

Windows burning tool:



c) Enter upgrade mode

There are many ways to enter the upgrade mode, as follows:

- Serial port mode
- Switching mode of programming tool
- TST mode (recommended)

i. Serial port mode

1) Set the serial port.
2) Make sure the serial port is connected correctly.
3) When the system starts, press any key to enter the serial port command line mode.
4) Execute the 'run update' command to enter the loader mode. After entering the loader mode, the system led will light up.

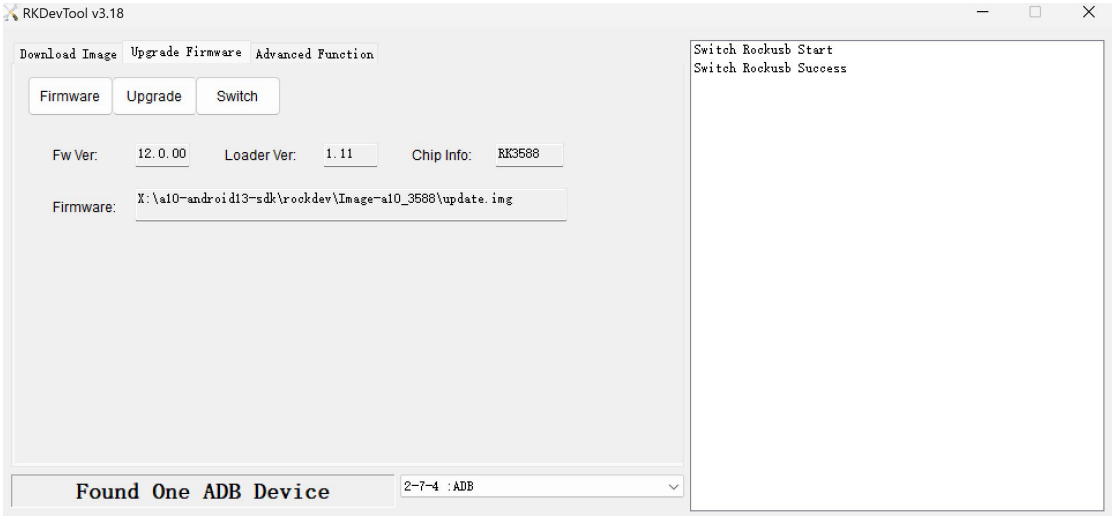
```
a10-3588# run update
```

5) Or execute the 'run maskrom' command to enter maskrom mode.

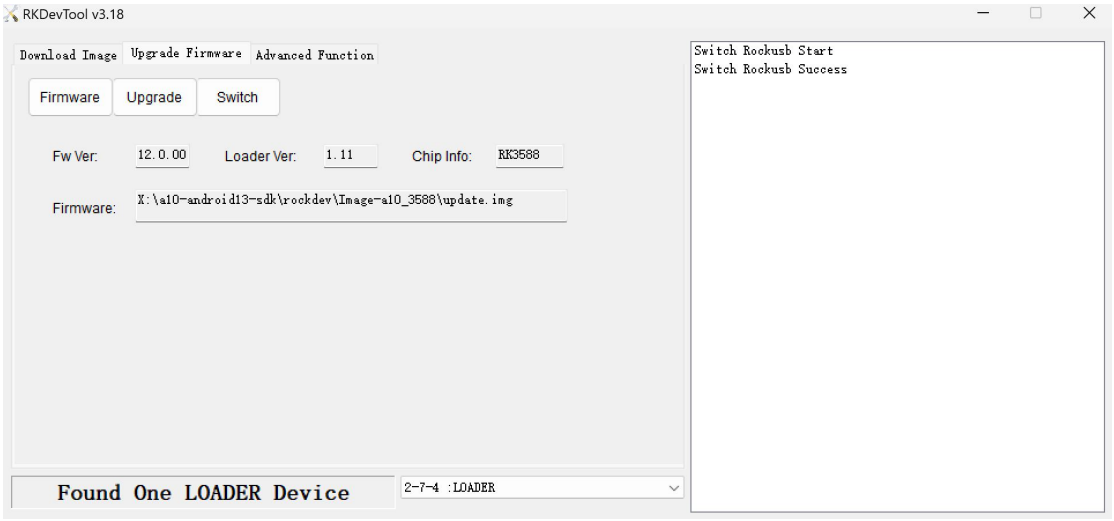
```
a10-3588# run maskrom
```

ii. Switching mode of programming tool

1) Connect the USB cable, start the system, and find the ADB mode, as shown in the following figure:



2) Click the Switch button, and the system will restart to enter the upgrade mode (loader mode), as shown in the following figure:

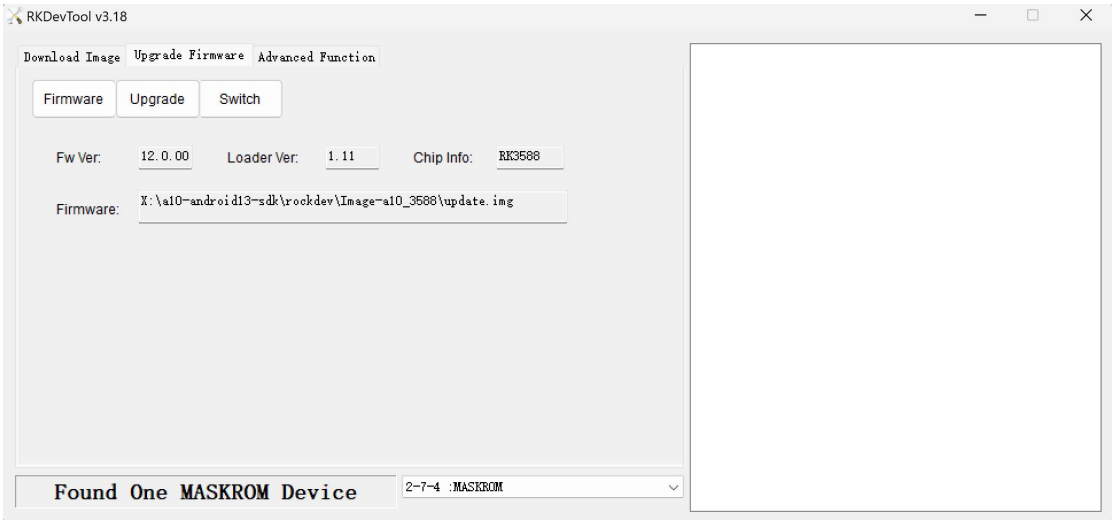


iii. TST mode (recommended)

1) Power on A10-3588.

2) Press the `Func 'key 3 times within 2 seconds and release.

3) You will see that the upgrade tool has entered upgrade mode (maskrom mode).



4.3. Upgrades firmware using usb cable in Ubuntu

a) Download the burning tool:

```
sudo apt-get install libusb -dev git parted
git clone https://github.com/khadas/utils
```

b) To install the burn tool:

```
cd /path/to/utils
git pull
sudo ./INSTALL
```

c) Check USB driver:

```
lsusb | grep Rockchip
Bus 003 Device 117: ID 2207:350b Fuzhou Rockchip Electronics Company
```

d) Burn a full image:

```
burn-tool -v rk - i /path/to/image
```

5. The interface uses

5.1.GPIO usage

GPIO, which stands for General-Purpose Input/Output, is a general-purpose pin that can be dynamically configured and controlled during software operation. The initial state of all the GPIOs after power-on is an input mode, which can be set as a pull-up pin or a pull-down pin through software, and can also be set as an interrupt pin. The driving strength is programmable, and the core of the method is to fill the method and parameters of the GPIO bank, and to call the gpiochip_add to register in the kernel.

a) GPIO pin calculation

RK3588 has 5 groups of GPIO banks: GPIO0 ~ GPIO4, and each group is numbered by A0 ~ A7, B0 ~ B7, C0 ~ C7, D0 ~ D7. The following formula is commonly used to calculate the pins:

- GPIO pin calculation formula: pin = bank * 32 + number
- GPIO group number calculation formula: number = group * 8 + X

The GPIO4 _ C3 pin calculation method is demonstrated as follows:

- bank=4；//GPIO4_C3=>4，bank∈[0,4]
- group=2；//GPIO4_C3=>2，group∈{(A=0),(B=1),(C=2),(D=3)}
- X=3；//GPIO4_C3=>3，X∈[0,7]
- number=group*8+X=2*8+3=19
- pin= bank*32+number=4*32+19=147

b) Input and output

i. User space uses GPIO

When the GPIO4 _ C3 pin is not multiplexed by other peripherals, we can export the pin to use:

```
echo 147 > /sys/class/gpio/export
```

The following example sets GPIO4 _ C3 as the output while outputting a high level:

```
echo out > /sys/class/gpio/gpio147/direction
echo 1 > /sys/class/gpio/gpio147/value
```

Output Low:

```
echo 0 > /sys/class/gpio/gpio147/value
```

ii. Using GPIO in the Kernel

Device Tree Add node: kernel/arch/arm64/boot/DTS/rockchip/rk3588-namtso-a10-3588.dts

```
normal  gpio :
/{
 ......
 dts  config:
   gpio_demo :  gpio_demo {
    compatible  =  "namtso,a10-rk3588-gpio";
      status  =  "okay";
      pinctrl -names  =  "default";
      pinctrl-0  =  <&pin147_gpio>;
     namtso-gpio  =  <&gpio4  RK_PC3  GPIO_ACTIVE_HIGH>;
   };
......
}

& pinctrl  {
   ......
   gpio {
     pin147_gpio:  pin147-gpio{
       rockchip,pins  =
       <4  RK_PC3  0  & pcfg_pull_none>;
     };
   };
   ......
};
```

```
   ......
   gpio {
     pin147_gpio:  pin147-gpio{
       rockchip,pins  =
       <4  RK_PC3  0  & pcfg_pull_none>;
     };
   };
   ......
};
```

GPIO_ ACTIVE_HIGH Indicates active high. If you want active low, change to: GPIO_ACTIVE_LOW. This property will be read by the driver.

Then, the resource added by DTS is parsed in the probe function, and the code is as follows:

```
static int namtso_gpio_demo_probe (struct platform_device * pdev )
{
 int ret = -1;
 int gpio = -1;
 enum of_gpio_flags flag;
 struct device_node * namtso_gpio_node = pdev -> dev.of_node ;
struct namtso_gpio * namtso_gpio_info = devm_kzalloc(&pdev->dev, sizeof(struct
namtso_gpio), GFP_KERNEL);
 if (! namtso_gpio_info ) {
      pr_err (" devm_kzalloc failed");
      return -ENOMEM;
 }

 gpio = of_get_named_gpio_flags(namtso_gpio_node, "namtso-gpio", 0, &flag);
 if (! gpio_is_valid ( gpio )) {
      pr_err (" gpio : %d is invalid\n", gpio );
       return -ENODEV;
 }

 if ( gpio_request ( gpio , " namtso-gpio")) {
    pr_err (" gpio %d request failed!\n", gpio );
     return -ENODEV;
}

 if ( gpio_request ( gpio , "namtso-gpio")) {
    pr_err (" gpio %d request failed!\n", gpio );
    return -ENODEV;
 }

 namtso_gpio_info -> gpio = gpio ;
```

```
 namtso_gpio_info->gpio_value = (flag == OF_GPIO_ACTIVE_LOW) ? 0:1;
 gpio_direction_output(namtso_gpio_info->gpio, namtso_gpio_info->gpio_value);

 platform_set_drvdata ( pdev , namtso_gpio_info );
       pr_info ("%s probe finished\n", __ func __);
      return 0;
}
```

The of _ get _ named _ gpio _ flags reads the namtso-gpio GPIO configuration number and flag from the device tree, the gpio _ is _ valid judges whether the GPIO number is valid, and the gpio _ request applies to occupy the GPIO. If the initialization process fails, the gpio _ free needs to be called to release the previously requested and successful

GPIO.You can set whether to output high or low level by calling the gpio _ direction _ output in the drive. The default output here is the effective level GPIO _ ACTIVE _ HIGH obtained from DTS, that is, high level. If the drive works normally, you can use a multimeter to measure that the corresponding pin should be high level. In practice, if you want to read out the GPIO, you need to set it to the input mode first, and then read the value:

```
int  val ;
gpio_direction_input ( your_gpio );
val  =  gpio_get_value ( your_gpio );

The following are commonly used  GPIO  API    Definition
#include  < linux / gpio.h >
#include  < linux / of_gpio.h >

int  of_get_named_gpio_flags(struct  device_node  *np,  const  char  *propname,
int  index,  enum  of_gpio_flags  *flags);
int  gpio_is_valid (int  gpio );
int  gpio_request (unsigned  gpio ,  const  char  *label);
void  gpio_free (unsigned  gpio );
int  gpio_direction_input (int  gpio );
int  gpio_direction_output (int  gpio ,  int  v);
```

c) Interrupt

The interrupt usage of the GPIO port is similar to the input and output of the GPIO:

Device Tree Add node: kernel/arch/arm64/boot/DTS/rockchip/rk3588-namtso-a10-3588.dts

```
gpio  {
    compatible  =  "namtso,rk3588-gpio";
    status  =  "okay";
    pinctrl -names  =  "default";
    pinctrl-0  =  <&pin147_gpio  >;
    namtso-irq-gpio  =  <&gpio4  RK_PD3  IRQ_TYPE_EDGE_RISING>;
};
```

The IRQ _ TYPE _ EDGE _ RISING indicates that the interrupt is triggered by a rising edge, and the interrupt function can be triggered when a rising edge signal is received on this pin. It can also be configured as follows:

```
IRQ_TYPE_NONE        // Default, no interrupt trigger type defined
IRQ_TYPE_EDGE_RISING   // Rising edge trigger
IRQ_TYPE_EDGE_FALLING   // Falling edge trigger
IRQ_TYPE_EDGE_BOTH     // Both rising and falling edge trigger
IRQ_TYPE_LEVEL_HIGH    // High level trigger
IRQ_TYPE_LEVEL_LOW      // Low level trigger
```

Then, the resource added by DTS is parsed in the probe function, and then the interrupt registration application is made. The code is as follows:

```
static int namtso_gpio_demo_probe (struct platform_device * pdev )
{
 int ret = -1;
 int gpio = -1;
 enum of_gpio_flags flag;
 struct device_node * namtso_gpio_node = pdev -> dev.of_node ;
struct namtso_gpio * namtso_gpio_info = devm_kzalloc(&pdev->dev, sizeof(struct
namtso_gpio), GFP_KERNEL);
 if (! namtso_gpio_info ) {
     pr_err (" devm_kzalloc failed");
     return -ENOMEM;
 }

 gpio = of_get_named_gpio_flags(namtso_gpio_node, "namtso-gpio", 0, &flag);
 if (! gpio_is_valid ( gpio )) {
    pr_err (" gpio : %d is invalid\n", gpio );
     return -ENODEV;
 }

 if ( gpio_request ( gpio , " namtso-gpio ")) {
     pr_err (" gpio %d request failed!\n", gpio );
     return -ENODEV;
 }

 namtso_gpio_info -> gpio = gpio ;
 namtso_gpio_info->gpio_value = (flag == OF_GPIO_ACTIVE_LOW) ? 0:1;
 //gpio_direction_output(namtso_gpio_info->gpio, namtso_gpio_info->gpio_value);

 namtso_gpio_info->gpio_irq = gpio_to_irq(namtso_gpio_info->gpio);
 if ( namtso_gpio_info -> gpio_irq ) {

 ret = request_irq(namtso_gpio_info->gpio_irq, namtso_gpio_irq_isr, IRQF_TRIGGER_RISING
 | IRQF_TRIGGER_FALLING, "namtso_gpio_irq", namtso_gpio_info);

36.     }
37.     platform_set_drvdata ( pdev , namtso_gpio_info );
38.     pr_info ("%s probe finished\n", __ func __);
39.     return 0;
40. }
```

```
ret = request_irq(namtso_gpio_info->gpio_irq, namtso_gpio_irq_isr, IRQF_TRIGGER_RISING
| IRQF_TRIGGER_FALLING, "namtso_gpio_irq", namtso_gpio_info);
 if (!ret)
{
    gpio_free ( namtso_gpio_info -> gpio );
     dev_err (& pdev ->dev, "Failed to request IRQ: %d\n", ret);
 }
 }
 platform_set_drvdata ( pdev , namtso_gpio_info );
pr_info ("%s probe finished\n", __ func __);
 return 0;
}
```

d) GPIO multiplexing

GPIO4 _ C3 can be multiplexed into the following 6 functions:

- I2C7_SCL_M1
- PWM4_M1
- SPI3_CS1_M0
- I2S2_SDO_M0
- GMAC0_MCLKINOUT
- GPIO4_C3

Our SDK releases GPIO4 _ C3 as a PWM pin by default.

Debugging method:

i. View gpio usage:

```
cat /sys/kernel/debug/ gpio
```

ii. View pinmux-pins:

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

5.2.PWM use

Here, pwm4 is taken as an example.

a) The PWM configuration

i. To enable pwm4, the following configuration is required, which is configured by default:

```
&pwm4 {
pinctrl-0 = <&pwm4m1_pins>;
status = "okay";
};
```

The pin corresponding to pwm4m1 is GPIO4 _ C3. After the above nodes are enabled, pwmchipx will be generated under/sys/class/PWM.

ii. How do I confirm which pwmchip?

```
cat /sys/class/ pwm / pwmchip */device/ uevent
```

iii. View the OF _ FULLNAME value.

The complete node corresponding to pwm4 is PWM @ febd0000, and the OF _ FULLNAME value in/sys/class/PWM/pwmchip0/device/uevent is also PWM @ febd0000, so the complete node corresponding to pwmchip0 is pwm4.

b) Process using PWM

In the pwmchipx pwmchipx record, write 0 to the export export file, that is, turn on the PWM timer 0, and a pwm0 record will be generated. The record contains the following files:

- Enable: write 1 to enable PWM and write 0 to close PWM;
- Polarity: There are two parameters, normal or inversed, and the level of the output pin of the meter is inverted;
- Duty _ cycle: in normal mode, the duration of the level in one cycle (unit: ns); in reversed mode, the duration of the low level in one cycle (unit: ns);
- Period: the period of the PWM wave (unit: nanosecond);
- Oneshot _ count: Table Number of PWM waveforms in one-shot mode, the value cannot exceed 255;

Set pwm0 as follows: output frequency 100K, duty cycle 50%, positive polarity, Continuous mode output:

```
cd  /sys/class/ pwm /pwmchip0/
echo  0  >  export
cd  pwm0
echo  10000  >  period
echo  5000  >  duty_cycle
echo  normal  >  polarity
echo  1  >  enable
```

c) Using PWM in Kernel

```
/{
  ……
    pwm_demo :   pwm_demo   {
       compatible   =   "namtso,rk3588-pwm-demo";
       status   =   "okay";
       pwms   =   <&pwm4   0   10000000   1>;
       duty_ns   =   <5000000>;
    };
    ……
};

12. &pwm4   {
13.     pinctrl-0   =   <&pwm4m1_pins>;
14.     status   =   "okay";
15. };
```

```
&pwm4   {
  pinctrl-0   =   <&pwm4m1_pins>;
  status   =   "okay";
};
```

In pwms = < & pwm4 0 10000000 1 >, & pwm4 is defined in rk3588s.dtsi, parameter 0, table index (per-chip index of the PWM to request), generally 0, Because Rockchip PWM has only one per chip; 10000000: one cycle time is 10000000 nanoseconds, and one second has 100 10000000 nanoseconds, so this PWM output cycle is 100 Hz. 1 for polarity, 0 for normal polarity, 1 for reverse polarity; duty _ ns for duty cycle activation time, also in nanoseconds.

d) Interface description and use

You can use the PWM nodes generated in the above steps in other driver files as follows:

i. The following header files are included in the driver:

```
#include < linux / pwm.h >
```

ii. Apply for PWM interface:

```
struct pwm_device * devm_pwm_get (struct device *dev, const char * con_id );
```

For example

```
struct pwm_device * pwm_demo = devm_pwm_get (& pdev ->dev, NULL);
```

iii. Configure the PWM interface:

```
int pwm_config (struct pwm_device * pwm , int duty_ns , int period_ns );
```

For example

```
pwm_config ( pwm_demo , 5000000, 10000000);
```

iv. Enable PWM function interface:

```
int pwm_enable (struct pwm_device * pwm );
```

For example

```
pwm_enable ( pwm_demo );
```

v. Disable PWM interface:

```
void pwm_disable (struct pwm_device * pwm );
```

For example

```
pwm_disable ( pwm_demo );
```

vi. Release the applied PWM interface:

```
void pwm_free (struct pwm_device * pwm );
```

For example

```
pwm_free ( pwm_demo );
```

vii. Debugging method:

```
pwm_free ( pwm_demo );
```

Find the corresponding node

```
ls /sys/kernel/debug/*. pwm
```

View debugging information

```
cat *. pwm
```

5.3.Ethernet use

a) Configuration of mainboard network card:

```
&gmac1 {
/* Use rgmii-rxid mode to disable rx delay inside Soc */
  phy -mode = " rgmii-rxid ";
  clock_in_out = "output";

/* snps,reset-gpio = <&gpio3 RK_PB7 GPIO_ACTIVE_LOW>;*/
/* snps,reset -active-low;*/
/* Reset time is 20ms, 100ms for rtl8211f */
/* snps,reset -delays-us = <0 20000 100000>;*/

  wolirq-gpio = <&gpio0 RK_PC2 GPIO_ACTIVE_LOW>;
  wolctrl-gpio = <&gpio0 RK_PC5 GPIO_ACTIVE_LOW>;

  pinctrl -names = "default";
pinctrl-0 = <&gmac1_miim
  &gmac1_tx_bus2
  &gmac1_rx_bus2
  &gmac1_rgmii_clk
  &gmac1_rgmii_bus
    & wol_gpio >;

  tx_delay = <0x3e>;
/* rx_delay = <0x35>;*/

  phy -handle = <& rgmii_phy >;
status = "okay";
```

```
};

&mdio1 {
  rgmii_phy : phy@1 {
  compatible = "ethernet-phy-ieee802.3-c22";
  reg = <0x1>;
};
};
```

b) How to use three Ethernet ports:

Eth0: corresponds to the network port of the motherboard

If the N-LINK board is connected:

Eth1: corresponding to 2.5G PCIE network port of N-LINK board

Eth2: corresponding to 2.5g USB network port of N-LINK board

c) Connectivity test

eth0： ping -I eth0 www.baidu.comhttp://www.baidu.com/

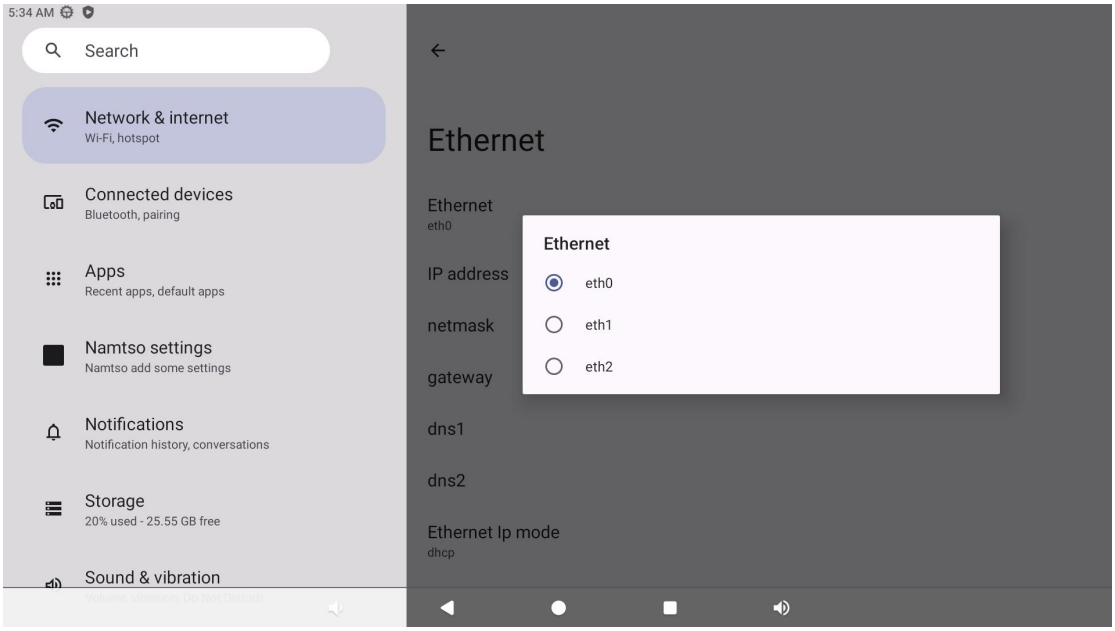
If the N-LINK board is connected:

eth1： ping -I eth1 www.baidu.comhttp://www.baidu.com/

eth2： ping -I eth2 www.baidu.comhttp://www.baidu.com/

d) Multiple network card settings

As shown in the figure, click Ethernet to select settings:



5.4.Display uses

The RK3588 has four Video output ports, each of which is bound with a fixed display controller. For example, Port0 can be used to connect with display controllers such as DP0, DP1, HDMI/eDP0 and HDMI/eDP1, and so on for other Portx.

Each Portx has its own maximum output resolution:

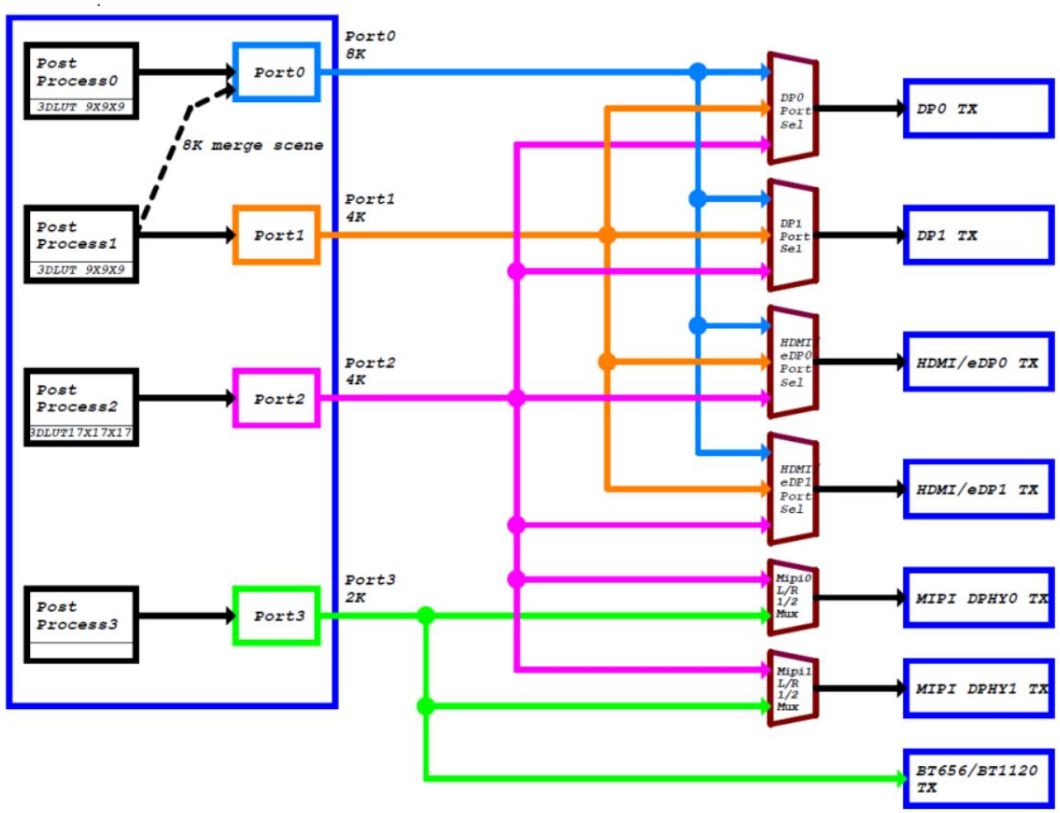
- Port 0 can output 7680x4320 @ 60Hz at most
- Port1 can output 4096x2304 @ 60Hz at most
- Port2 can output 4096x2304 @ 60Hz at most
- Port3 can output up to 1920x1080 @ 60Hz

In terms of software, if the HDMI0/1 display controller is connected to Port0 and the hardware phy supports HDMI2.1, then HDMI0/1 supports 8K @ 60Hz output.

If a separate display controller is assigned to each Portx, it can support simultaneous display (different display) of four screens at the same time.

Caution:

- When the RK3588 performs 8K output, the resources of Port0 and Port1 are used inside the chip at the same time, and only Port0 is used for output. Therefore, when the display controller connected to Port0 is connected to the external 8K display for output, the display controller connected to Port1 will work abnormally.
- Currently, one Portx can only output one resolution format at the same time, so if the software is configured with multiple display interfaces with different resolutions connected to the same Portx, only one of them can be used at the same time.



5.4.1 A10-3588 Display Interface Configuration

The A10-3588 motherboard is provided with four display output interfaces of HDMI2.1, DP0, MIPI1 and MIPI2, the Android/Linux system can realize multi-screen different display, and the A10-3588 N-LINK board is provided with two display output interfaces of DP1 and EDP.

a) HDMI configuration

Up to 8K

```
&hdmi1 {
enable- gpios = <&gpio4 RK_PB1 GPIO_ACTIVE_HIGH>;
  cec -enable;
status = "okay";
};

&hdmi1_in_vp0 {
status = "okay";
};

&hdmi1_sound {
status = "okay";
};

&hdptxphy_hdmi1 {
status = "okay";
};

&hdptxphy_hdmi_clk1 {
status = "okay";
};

&route_hdmi1 {
status = "okay";
connect = <&vp0_out_hdmi1>;
};

/* hdmi1_sound */
&i2s6_8ch {
status = "okay";
};
```

By default, HDMI1 is connected to Port0, that is, to support 8K output.

b) Display Port configuration

Up to 4K in DP0 configuration

```
&dp0 {
status = "okay";
};
```

```
&dp0_in_vp2 {
status = "okay";
};

&dp0_sound{
status = "okay";
};

/* dp0_sound */
&spdif_tx2 {
status = "okay";
};
```

DP0 depends on USBC configuration. Refer to relevant configuration of usbc0: fusb302 @ 22.

DP0 is connected to VP2, not to VP1, that is, to support 8K output for HDMI, and also to support HDMI + DP dual-screen display.

c) DSI0 configuration

The DSI0 configuration is compatible with our TS050 and TS101 screens. The DSI0 is connected to the VP2 and cannot be used with other peripherals connected to the VP2.

```
&dsi0 {
status = "okay";
reset-delay- ms = <20>;
reset- gpios = <&gpio3 RK_PC1 GPIO_ACTIVE_HIGH>;
  pinctrl -names = "default";
pinctrl-0 = <&lcd_rst1_gpio>;
};

&dsi0_in_vp2 {
status = "okay";
};

&dsi0_in_vp3 {
status = "disabled";
};

&route_dsi0 {
status = "okay";
connect = <&vp2_out_dsi0>;
};

&mipi_dcphy0 {
status = "okay";
};
```

```
  &dsi0_panel {
        status = "okay";
power-supply = <&vcc3v3_lcd1_en>;
};
```

d) DSI1 configuration

The DSI1 configuration is compatible with our TS050 and TS101 screens. The DSI1 is connected to the VP3.

```
&dsi1 {
status = "okay";
reset-delay- ms = <20>;
reset- gpios = <&gpio3 RK_PD4 GPIO_ACTIVE_HIGH>;
  pinctrl -names = "default";
pinctrl-0 = <&lcd_rst2_gpio>;
};

&dsi1_in_vp2 {
status = "disabled";
};

&dsi1_in_vp3 {
status = "okay";
};

&route_dsi1 {
status = "okay";
connect = <&vp3_out_dsi1>;
};

&mipi_dcphy1 {
status = "okay";
};

&dsi1_panel {
status = "okay";
power-supply = <&vcc3v3_lcd2_en>;
};
```

e) DP1 configuration

The N-LINK DP1 configuration is connected to the VP2 and cannot be used with other peripherals connected to the VP2 at the same time. It supports up to 4K.

```
&dp1 {
    pinctrl -names = "default";
  pinctrl-0 = <&dp1_hpd>;
    hpd-gpios = <&gpio3 RK_PD5 GPIO_ACTIVE_HIGH>;
  status = "okay";
};
.
&dp1_in_vp2 {
```

```
  status = "okay";
};

&dp1_sound{
  status = "okay";
};

/* dp1_sound */
&spdif_tx5 {
  status = "okay";
};
```

f) EDP0 configuration

The N-LINK EDP0 configuration is connected to VP2 and cannot be used with other peripherals connected to VP2.

```
&edp0 {
status = "okay";
  pinctrl -names = "default";
pinctrl-0 = <&edp0_hpd>;
  hpd-gpios = <&gpio1 RK_PA5 GPIO_ACTIVE_HIGH>;

ports {
  port@1 {
    reg = <1>;

    edp0_out_panel: endpoint {
      remote-endpoint = <&panel_in_edp0>;
    };
  };
};
};

&edp0_in_vp2 {
status = "okay";
};
```

```
&route_edp0 {
        connect = <&vp2_out_edp0>;
status = "okay";
};
        &hdptxphy0 {
        status = "okay";
};
```

5.4.2 Debugging means

To get information about the Video Portx in use on the system (and the connected display controller):

```
cat /sys/kernel/debug/ dri /0/summary
```

5.4.3 Android System Version 1 Firmware Compatible with Various Screen Logic

In uboot, different mipi screens are distinguished by reading the TP ID of the mipi screen, and then different screen timings are loaded. If the Namtso _ mipi _ ID is 0, it means there is no mipi screen and EDP screen; if 1, it means old TS050 screen; if 2, it means TS101 screen; if 3, it means new TS050 screen; if 4, it means EDP screen. The Namtso _ mipi _ ID corresponds to the LCD port 1, and the Namtso _ mipi _ id2 corresponds to the LCD port 2.

u-boot/drivers/video/drm/rockchip_panel.c

```
if( namtso_mipi_id !=4){
  if( first_flag ){
     tp_id = kbi_i2c_read(6,0xA8,TP05_CHIP_ADDR);
  }
  else
     tp_id = kbi_i2c_read(0,0xA8,TP05_CHIP_ADDR);

   printf ("TP05 id=0x%x\n", tp_id );
  if( tp_id == 0x51){//old TS050
   if(! first_flag )
      namtso_mipi_id = 1;
   else
     namtso_mipi_id2 = 1;
     printf ("old TS050 to parse panel init sequence\n");
   data = dev_read_prop (dev, "panel- init -sequence", & len );
  }else if( tp_id == 0x79){//new TS050
   if(! first_flag )
      namtso_mipi_id = 3;
   else
      namtso_mipi_id2 = 3;
     printf ("new TS050 to parse panel init sequence2\n");
   data = dev_read_prop (dev, "panel-init-sequence2", & len );
  }else{
   if( first_flag )
      tp_id = kbi_i2c_read(6,0x9e,TP10_CHIP_ADDR);
```

```
    else{
        tp_id = kbi_i2c_read(0,0x9e,TP10_CHIP_ADDR);
    }



                    printf ("TP10 id=0x%x\n", tp_id );
    if( tp_id == 0x00){//TS101
      if(! first_flag )
         namtso_mipi_id = 2;
      else
        namtso_mipi_id2 = 2;
}else {
      if(! first_flag )
         namtso_mipi_id = 0;
      else
        namtso_mipi_id2 = 0;
    }
      printf ("old TS050 to parse panel init sequence\n");
    data = dev_read_prop (dev, "panel- init -sequence", & len );
  }
  printf (" namtso_mipi_id =%d namtso_mipi_id2=%d\n", namtso_mipi_id , namtso_mipi_id2);

       }
else{
  data = dev_read_prop (dev, "panel- init -sequence", & len );
}
  first_flag = ! first_flag ;
```

The EDP screen obtains the HPD pin through uboot to determine whether the EDP screen exists. If yes, the dsi0 DTS configuration will be disabled. Otherwise, the edp0 DTS configuration will be disabled.

u-boot/drivers/video/drm/ rockchip_connector.c

```
int rockchip_connector_enable (struct display_state *state)
{
struct rockchip_connector *conn;
#ifdef CONFIG_CHECKEDP
char *p;
char *mode;
extern int namtso_mipi_id ;
p = env_get (" checkedp ");
if (p != NULL) {
   run_command_list (p, -1, 0);
 // run_command (" gpio set 133", 0); //HDMI/EDP_SW GPIO4_A5 1  :    hdmi
   run_command (" gpio clear 133", 0); //HDMI/EDP_SW GPIO4_A5 0: edp
 mode = env_get (" namtso_mipi_id ");
   printf ("%s mode=%s\n", __ func __,mode);
 if( strcmp (mode,"4") == 0)
    namtso_mipi_id = 4;
   printf ("%s namtso_mipi_id =%d\n", __ func __, namtso_mipi_id );
 if( namtso_mipi_id == 4){
    run_command (" fdt    addr 0x08300000", 0);
    run_command (" fdt set /dsi@fde20000 status disable", 0);
    run_command (" fdt set /dsi@fde20000/panel@0 status disable", 0);
    run_command (" fdt    set /dsi@fde20000/ports/port@0/endpoint@0 status disable", 0);
```

```
    run_command (" fdt set /display-subsystem/route/route-dsi0 status disable", 0);
    printf ("dsi0 disable\n");
 }
 else{
    run_command (" fdt    addr 0x08300000", 0);
    run_command (" fdt set /edp@fdec0000 status disable", 0);
    run_command (" fdt set /edp@fdec0000/ports/port@0/endpoint@2 status disable", 0);
    run_command (" fdt set /display-subsystem/route/route-edp0 status disable", 0);
    run_command (" fdt set /phy@fed60000 status disable", 0);
   // run_command (" fdt set /backlight-edp0 status disable", 0);
   // run_command (" fdt set /pwm@febd0020 status disable", 0);
    printf ("edp0 disable\n");
 }
}
#endif
```

The kernel layer will identify the u-boot to the screen ID through the kernel pass parameter method and then load different screen timings.

kernel-5.10/drivers/gpu/drm/panel/panel-simple.c

```
static char lcd_propname [1] = "0";
static int __ init    namtso_mipi_id_para_setup (char *str)
{
if (str != NULL){
  sprintf ( lcd_propname , "%s", str);
  if(! strcmp ( lcd_propname , "4"))
      namtso_mipi_id = 4;
else if(! strcmp ( lcd_propname , "3"))
  namtso_mipi_id = 3;
  else if(! strcmp ( lcd_propname , "2"))
      namtso_mipi_id = 2;
  else if(! strcmp ( lcd_propname , "1"))
      namtso_mipi_id = 1;
  else
      namtso_mipi_id = 0;
}
// printk (" lcd_propname : %s namtso_mipi_id : %d\n", lcd_propname , namtso_mipi_id );
return 0;
}
__setup(" namtso_mipi_id =", namtso_mipi_id_para_setup );

static char lcd_propname2[1] = "0";
static int __ init namtso_mipi_id_para_setup2(char *str)
{
```

```
if (str != NULL){
  sprintf (lcd_propname2, "%s", str);
  if(! strcmp (lcd_propname2, "4"))
    namtso_mipi_id2 = 4;
else if(! strcmp (lcd_propname2, "3"))
namtso_mipi_id2 = 3;
  else if(! strcmp (lcd_propname2, "2"))
    namtso_mipi_id2 = 2;
  else if(! strcmp (lcd_propname2, "1"))
    namtso_mipi_id2 = 1;
  else
    namtso_mipi_id2 = 0;
}
// printk ("lcd_propname2: %s namtso_mipi_id2: %d\n", lcd_propname2, namtso_mipi_id2);
return 0;
}
__setup("namtso_mipi_id2=", namtso_mipi_id_para_setup2);
```

kernel-5.10/ drivers/video/of_display_timing.c

```
if( first_flag || (0 == namtso_mipi_id && 0 ! = namtso_mipi_id2)){
  if(2 == namtso_mipi_id2){
       timings_np = of_get_child_by_name (np, "display-timings1");
  }else if(namtso_mipi_id2 == 3){//new TS050
     timings_np = of_get_child_by_name (np, "display-timings2");
  }else{//old TS050
       timings_np = of_get_child_by_name (np, "display-timings");
  }
}else{
  if(2 == namtso_mipi_id ){
     timings_np = of_get_child_by_name (np, "display-timings1");
  }else if( namtso_mipi_id == 3){//new TS050
     timings_np = of_get_child_by_name (np, "display-timings2");
  }else{//old TS050
     timings_np = of_get_child_by_name (np, "display-timings");
  }
}
  first_flag = ! first_flag ;
```

The Android layer reads the kernel command line parameters to obtain the screen ID, and then sets the corresponding primary and secondary screens, rotation angle, and UI size.

system/core/property_service.cpp

```
static void export_lcd_status () {
int fd ;
char buf [2048];
```

```
LOG(INFO) << " export_lcd_status !";
if (( fd = open("/proc/ cmdline ", O_RDONLY)) < 0) {
LOG(FATAL) << "Failed to export lcd status!";
  InitPropertySet ("sys.lcd.id", "0");
return;
}
read( fd , buf , sizeof ( buf ) - 1);
if( strstr (buf,"namtso_mipi_id2=2") != NULL) {//TS101
    InitPropertySet ("persist.sys.rotation.einit-2", "0");
}else if( strstr (buf,"namtso_mipi_id2=1") != NULL || strstr (buf,"namtso_mipi_id2=3") != NULL)
{//old or new TS050
    InitPropertySet ("persist.sys.rotation.einit-2", "3");
}else {
    InitPropertySet ("persist.sys.rotation.einit-2", "0");
}

if( strstr ( buf ," namtso_mipi_id =2") != NULL) {//TS101
  InitPropertySet ("sys.lcd.id", "2");
  InitPropertySet (" vendor.hwc.device.primary ", "HDMI-A,DP");
    InitPropertySet (" vendor.hwc.device.extend ", "DSI");
    InitPropertySet ("persist.sys.rotation.einit-0", "0");
    InitPropertySet ("persist.sys.rotation.einit-1", "0");
    InitPropertySet (" persist.vendor.framebuffer.main ", "1920x1200@60");
LOG(INFO) << "switch TS101!";
}else if( strstr ( buf ," namtso_mipi_id =4") != NULL) {// edp
  InitPropertySet ("sys.lcd.id", "2");
  InitPropertySet (" vendor.hwc.device.primary ", "HDMI-A,DP");

    InitPropertySet (" vendor.hwc.device.extend ", " Edp ");
    InitPropertySet ("persist.sys.rotation.einit-0", "0");
    InitPropertySet ("persist.sys.rotation.einit-1", "0");
    InitPropertySet (" persist.vendor.framebuffer.main ", "2560x1600@60");
LOG(INFO) << "switch Edp !";
}else if( strstr ( buf ," namtso_mipi_id =1") != NULL || strstr ( buf ," namtso_mipi_id =3") != NULL)
{//old or new TS050
  InitPropertySet ("sys.lcd.id", "1");

  InitPropertySet (" vendor.hwc.device.primary ", "HDMI-A,DP");
    InitPropertySet (" vendor.hwc.device.extend ", "DSI");
    InitPropertySet ("persist.sys.rotation.einit-0", "0");
    InitPropertySet ("persist.sys.rotation.einit-1", "3");

    InitPropertySet (" persist.vendor.framebuffer.main ", "1920x1080@60");
LOG(INFO) << "switch TS050!";
```

```
}else if( strstr ( buf ," namtso_mipi_id =0") != NULL && strstr (buf,"namtso_mipi_id2=0") != NULL)
{//no lcd
  LOG(INFO) << "switch none!";
    InitPropertySet ("sys.lcd.id", "0");
    InitPropertySet (" vendor.hwc.device.primary ", "HDMI-A");
    InitPropertySet (" vendor.hwc.device.extend ", "DP");
    InitPropertySet ("persist.sys.rotation.einit-0", "0");
    InitPropertySet ("persist.sys.rotation.einit-1", "0");
  //std::string value = GetProperty (" persist.vendor.framebuffer.main ", "1920x1080@60");
  //LOG(ERROR) << "switch value=" + value;

}else {// only lcd2
  LOG(INFO) << "switch only lcd2!";
  InitPropertySet (" vendor.hwc.device.primary ", "HDMI-A,DP");
    InitPropertySet (" vendor.hwc.device.extend ", "DSI");
  if( strstr (buf,"namtso_mipi_id2=2") != NULL) {//TS101
      InitPropertySet ("sys.lcd.id", "2");
      InitPropertySet ("persist.sys.rotation.einit-0", "0");
      InitPropertySet ("persist.sys.rotation.einit-1", "0");
      InitPropertySet (" persist.vendor.framebuffer.main ", "1920x1200@60");
    LOG(INFO) << "switch TS101!";
  }else if( strstr (buf,"namtso_mipi_id2=1") != NULL || strstr (buf,"namtso_mipi_id2=3") != NULL)
{//old or new TS050
      InitPropertySet ("sys.lcd.id", "1");
      InitPropertySet (" persist.vendor.framebuffer.main ", "1920x1080@60");
    LOG(INFO) << "switch TS050!";
      InitPropertySet ("persist.sys.rotation.einit-0", "0");
      InitPropertySet ("persist.sys.rotation.einit-1", "3");
  }
}
close( fd );
}
```

5.5.ADC use

There are two types of AD interfaces on the A10-3588: Temperature Sensor and Successive Approximation ADC (Successive Approximation Register).

TS-ADC (Temperature Sensor): support seven channels.

SAR-ADC (Successive Approximation Register): Supports eight channels of single-ended, 12-bit SAR-ADC with a maximum conversion rate of 1 MSPS and 20 MHz a/D converter clock.

The core uses an industrial I/O subsystem to control the ADC, which is mainly designed for AD or DA conversion sensors. The following takes SAR-ADC as an example to introduce the use and configuration method of ADC.

a) Get all channel ADC values

```
cat /sys/bus/ iio /devices/ iio \:device0/ in_voltage *_raw
```

b) Using ADC in Kernel

i. To configure a DTS node:

```
adc_demo :  adc_demo {
   compatible  =  "namtso,rk3588-adc";
   status  =  "okay";
    io-channels  =  <& saradc  3>;
};
```

ii. Driver:

```
static  void  namtso_adc_iio_poll (struct  work_struct  *work)
{
 int  ret  =  -1;
 int  value  =  -1;
 if  ( g_ctrl -> chan )
 {
   ret  =  iio_read_channel_raw ( g_ctrl -> chan ,  &value);
    if  (ret  <  0)
    {
        pr_err (" iio_read_channel_raw \n");
     }
     mutex_lock (& g_ctrl -> mutex_rd );
    g_ctrl -> read_value  =  value;
    mutex_unlock (& g_ctrl -> mutex_rd );
    schedule_delayed_work(&g_ctrl->adc_poll_work,  NAMTSO_ADC_SAMPLE_JIFFIES);
 }
}

static  ssize_t  namtso_adc_write (struct  file  *file,
  const  char  __user  * buf ,  size_t  n,  loff_t  *  offset)
{
        return  0;
}

static  ssize_t  namtso_adc_read (struct  file  *file,
  char  __user  * buf ,  size_t  n,  loff_t  *offset)
{
 int  ret  =  -1;
mutex_lock (& g_ctrl -> mutex_rd );
 ret  =  copy_to_user ( buf ,  & g_ctrl -> read_value ,  sizeof (int));
 mutex_unlock (& g_ctrl -> mutex_rd );
```

```
return  sizeof (int);
}

static  loff_t  namtso_adc_lseek(struct  file  *  file,  loff_t  offset,  int  whence)
{
 return  0;
}

static  int  namtso_adc_open(struct  inode  *  inode,  struct  file  *  file)
{
 return  0;
}

static  int  namtso_adc_close(struct  inode  *  inode,  struct  file  *  file)
{
 return  0;
}

static  long  namtso_adc_ioctl(struct  file  *filp,  unsigned  int  cmd,  unsigned  long  arg)
{
    switch( cmd )
      {
      case  ADC_SET:
       if  ( arg )
      {
         schedule_delayed_work (& g_ctrl -> adc_poll_work ,  1000);
      }
       else
              {
          cancel_delayed_work_sync (& g_ctrl -> adc_poll_work );
      }
       break;
      default:
       printk (KERN_ERR  " cmd  is  error\n");
        break;
       }
    return  0;
}

static  const  struct  file_operations  namtso_adc_fops  =  {
 .write  =  namtso_adc_write ,
 .read  =  namtso_adc_read ,
 . llseek  =  namtso_adc_lseek ,
.open  =  namtso_adc_open ,
.release  =  namtso_adc_close ,
```

```
 . unlocked_ioctl  =  namtso_adc_ioctl ,
.owner  =  THIS_MODULE,
};

static  struct  miscdevice  namtso_adc_misc  =  {
 .minor  =  MISC_DYNAMIC_MINOR,
 .name  =  " namtso_adc ",
 .fops  =  & namtso_adc_fops ,
};


static  int  namtso_adc_iio_probe (struct  platform_device  * pdev )
{
 pr_info (" namtso_adc_iio_probe !\n");

 g_ctrl  =  devm_kzalloc(&pdev->dev,  sizeof(struct  adc_ctrl),  GFP_KERNEL);
 if  (! g_ctrl )
 {
    dev_err (& pdev ->dev,  " devm_kzalloc  fail\n");
     return  -ENOMEM;
 }
 g_ctrl -> chan  =  iio_channel_get (& pdev ->dev,  NULL);
if  (! g_ctrl -> chan )
 {
    dev_err (& pdev ->dev,  " iio_channel_get  fail\n");
     return  -EINVAL;
 }
 mutex_init (& g_ctrl -> mutex_rd );
INIT_DELAYED_WORK(& g_ctrl -> adc_poll_work ,  namtso_adc_iio_poll );

 misc_register (& namtso_adc_misc );
  return  0;
}

static  int  namtso_adc_iio_remove (struct  platform_device  * pdev )
{
  pr_info (" namtso_adc_iio_remove !\n");
   cancel_delayed_work_sync (& g_ctrl -> adc_poll_work );
  if  ( g_ctrl -> chan )
  {
     iio_channel_release ( g_ctrl -> chan );
     g_ctrl -> chan  =  NULL;
   }
   misc_deregister (& namtso_adc_misc );
  return  0;
}
```

iii. Description of drive:

Get iio channel:

```
struct iio_channel * chan ; # Define IIO channel structure
g_ctrl -> chan = iio_channel_get (& pdev ->dev, NULL); # Get IIO channel structure
```

Read the raw data collected by AD:

```
iio_read_channel_raw ( g_ctrl -> chan , &value);
```

Calculate the collected voltage:

The AD converted value is converted into a voltage value required by the user using the standard voltage. The calculation formula is as follows:

Vref / (2^n-1) = Vresult / raw

Formula description:

- Vref is the standard voltage
- N is the number of bits for AD conversion
- Vresult is the acquisition voltage required by the user
- Raw refers to the raw data collected by AD

For example, if the standard voltage is 1.8 V, the number of AD acquisition bits is 12, and the original data acquired by AD is 1000, then:

Vresult = (1800mv * 1000) / 4095;

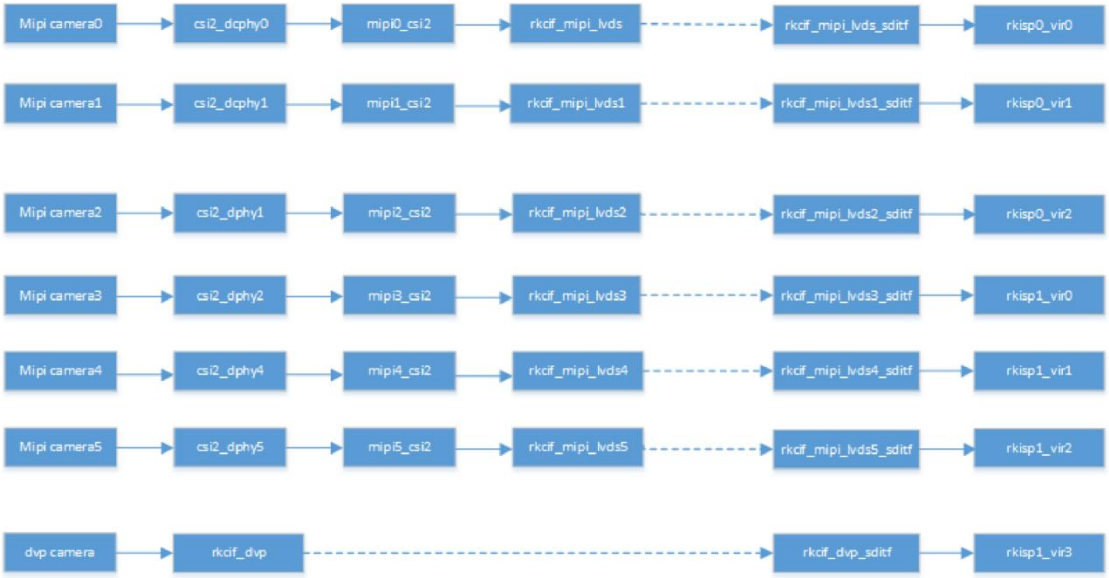
Release the channel obtained by the iio _ channel _ get function:

```
iio_channel_release ( g_ctrl -> chan );
```

5.6.Camera use

5.6.1 Introduction

The RK3588 hardware supports the acquisition of up to 7 sensors: 6mipi + 1dvp, and the software channels for multiple sensors are as follows:



The rk3588 supports two dcphys with node names csi2 _ dcphy0/csi2 _ dcphy1. Each dcphy hardware supports the simultaneous use of RX/TX, and RX is used for camera input. Support DPHY/CPHY protocol multiplexing. It should be noted that TX/RX of the same dcphy can only use DPHY or CPHY at the same time. Refer to the rk3588 data sheet for additional dcphy parameters.

The rk3588 supports two dphy hardwares, here we call them dphy0 _ HW/dphy1 _ HW. Both dphy hardwares can work in full mode and split mode:

a) dphy0_hw

- Full mode: The node name uses csi2 _ dphy0 and supports up to 4 lanes.
- Split mode: split into 2 Phys, csi2 _ dphy1 (using 0/1 lane) and csi2 _ dphy2 (using 2/3 lane). Each phy supports 2 lanes at most.

b) dphy1_hw

- Full mode: The node name uses csi2 _ dphy3 and supports up to 4lane.
- Split mode: split into 2 Phys, csi2 _ dphy4 (use 0/1 lane).
- Csi2 _ dphy5 (using 2/3 lanes), up to 2 lanes per phy.

To use the above mipi phy node, the corresponding physical node needs to be configured.

（csi2_dcphy0_hw/csi2_dcphy1_hw/csi2_dphy0_hw/csi2_dphy1_hw）

Each mipi phy requires a csi2 module to parse the mipi protocol, and the node names are mipi0 _ csi2 ~ mipi5 _ csi2;

Rk3588 All camera data needs to go through VICAP and then link to ISP. The rk3588 only supports one VICAP hardware, and this VICAP supports the simultaneous input of six mipi phy channels and one DVP channel data, so we

differentiate the VICAP into seven sections such as rkcif _ mipi _ LVDS, rkcif _ mipi _, lvds5, and rkcif _ DVP. The link relationship between each VICAP node and the ISP is indicated by the corresponding virtual XXX _ sditf.

The rk3588 supports two ISP hardware, and each ISP device can virtualize multiple virtual nodes. The software reads the image data of each channel from the DDR in turn for ISP processing by means of readback. For a multi-shot scheme, it is recommended to divide the data stream equally between the two ISPs.

Pass-through and readback modes:

- Pass-through: means that the data is collected by the VICAP and sent directly to the ISP for processing instead of being stored in the DDR. It should be noted that in the case of HDR passthrough, only the short frame is a true passthrough, and the long frame needs to have DDR, and ISP reads from DDR
- Readback: the data is collected to DDR through VICAP. After the application obtains the data, it pushes the buffer address to ISP, and ISP obtains the image data from DDR

During DTS configuration, if only one virtual node is configured for an ISP hardware, the pass-through mode is used by default, and if multiple virtual nodes are configured, the read-back mode is used by default

The Namtso-A10 supports up to four 4Lane cameras (IMX415/OS08A10) using two dcphys, two dphys (full mode), Configure the path as shown in kernel/arch/arm64/boot/DTS/rockchip/rk3588-namtso-a10-3588-camera. Dtsi.

The following is an example of a camera mounted on the i2c2:

```
&i2c2 {
status = "okay";
  pinctrl -names = "default";
pinctrl-0 = <&i2c2m4_xfer>;

dw9714_cam1: dw9714_cam1@c {
  compatible = "dongwoon,dw9714";
  status = "okay";
  reg = <0x0c>;
    pinctrl -names = "focus_cam1_gpios";
  pinctrl-0 = <&focus_cam1_gpio>;
  focus- gpios = <&gpio1 RK_PB1 GPIO_ACTIVE_HIGH>;
    rockchip,vcm -start-current = <0>;
    rockchip,vcm -rated-current = <65>;
    rockchip,vcm -step-mode = <4>;
    rockchip,camera -module-index = <0>;
    rockchip,camera -module-facing = "back";
};

imx415_cam1: imx415_cam1@1a {
  compatible = "sony,imx415";
  status = "okay";
  reg = <0x1a>;
  clocks = <&cru CLK_MIPI_CAMARAOUT_M1>;
  clock-names = " xvclk ";
  power-domains = <&power RK3588_PD_VI>;
    pinctrl -names = "default", " cam_reset_gpios ";
  pinctrl-0 = <&mipim0_camera1_clk>, <&cam1_reset_gpio>;
    rockchip,grf = <& sys_grf >;
  reset- gpios = <&gpio1 RK_PB0 GPIO_ACTIVE_LOW>;
    rockchip,camera -module-index = <0>;
    rockchip,camera -module-facing = "back";
    rockchip,camera -module-name = "CMK-OT2022-PX1";
    rockchip,camera -module-lens-name = "IR0147-50IRC-8M-F20";
  lens-focus = <&dw9714_cam1>;
  port {
    imx415_cam1_out: endpoint {
      remote-endpoint = <&mipi_in_dcphy1>;
      data-lanes = <1 2 3 4>;
    };
  };
};
os08a10_cam1: os08a10_cam1@36 {
  compatible = "ovti,os08a10";
  status = "okay";
  reg = <0x36>;
```

```
  clocks = <&cru CLK_MIPI_CAMARAOUT_M1>;
  clock-names = " xvclk ";
  power-domains = <&power RK3588_PD_VI>;
   pinctrl -names = "default", " cam_reset_gpios ";
  pinctrl-0 = <&mipim0_camera1_clk>, <&focus_cam1_gpio>;
   rockchip,grf = <& sys_grf >;
  reset- gpios = <&gpio1 RK_PB1 GPIO_ACTIVE_HIGH>;
   rockchip,camera -module-index = <0>;
   rockchip,camera -module-facing = "back";
   rockchip,camera -module-name = "default";
   rockchip,camera -module-lens-name = "default";
  port {
   os08a10_cam1_out: endpoint {
    remote-endpoint = <&mipi_in_dcphy4>;
    data-lanes = <1 2 3 4>;
   };
  };
};
};

&csi2_dcphy1 {
status = "okay";

ports {
  #address-cells = <1>;
  #size-cells = <0>;
  port@0 {
   reg = <0>;
   #address-cells = <1>;
   #size-cells = <0>;

   mipi_in_dcphy1: endpoint@1 {
    reg = <1>;
    remote-endpoint = <&imx415_cam1_out>;
    data-lanes = <1 2 3 4>;
   };

   mipi_in_dcphy4: endpoint@2 {
    reg = <2>;
    remote-endpoint = <&os08a10_cam1_out>;
    data-lanes = <1 2 3 4>;
   };
  };
  port@1 {
   reg = <1>;
   #address-cells = <1>;
```

```
    #size-cells = <0>;

    csidcphy1_out: endpoint@0 {
      reg = <0>;
      remote-endpoint = <&mipi1_csi2_input>;
    };
  };
};
};

&mipi_dcphy1 {
status = "okay";
};

&mipi1_csi2 {
status = "okay";

ports {
  #address-cells = <1>;
  #size-cells = <0>;

  port@0 {
    reg = <0>;
    #address-cells = <1>;
    #size-cells = <0>;

    mipi1_csi2_input: endpoint@1 {
      reg = <1>;
      remote-endpoint = <&csidcphy1_out>;
    };
  };

  port@1 {
    reg = <1>;
    #address-cells = <1>;
    #size-cells = <0>;

    mipi1_csi2_output: endpoint@0 {
      reg = <0>;
      remote-endpoint = <&cif_mipi_in1>;
    };
  };
};
};

&rkcif_mipi_lvds1 {
```

```
status = "okay";

port {
  cif_mipi_in1: endpoint {
    remote-endpoint = <&mipi1_csi2_output>;
  };
};
};

&rkcif_mipi_lvds1_sditf {
status = "okay";

port {
  mipi1_lvds_sditf: endpoint {
    remote-endpoint = <&isp0_vir1>;
  };
};
};

&rkisp0_vir1 {
status = "okay";

port {
  #address-cells = <1>;
  #size-cells = <0>;

  isp0_vir1: endpoint@0 {
    reg = <0>;
    remote-endpoint = <&mipi1_lvds_sditf>;
  };
};
};
```

Path description: imx415 _ cam1/os08a10 _ cam1s-- > csi2 _ dcphy1-- > mipi1 _ csi2-- > rkcif _ mipi _ lvds1-- > rkcif _ mipi _ lvds1 _ sditfs-- > rkisp0_vir1

5.7.CAN use

5.7.1 Introduction to CAN

CAN (Controller Area Network) bus is a kind of serial communication network which effectively supports distributed control or real-time control. CAN-bus is a widely used bus protocol in automobiles, which is designed as a microcontroller communication in the automotive environment.

5.7.2 DTS Node Configuration

```
&can1 {
    status = "okay";
    assigned-clocks = <&cru CLK_CAN1>;
    assigned-clock-rates = <200000000>;
   pinctrl -names = "default";
    pinctrl-0 = <&can1m1_pins>;
};
```

Since the system creates only one CAN device based on the above DTS node, the first device created is CAN0s

5.7.3 CAN configuration test

```
# Set the bit rate to 200Kbps at the transceiver.
ip link set can0 type can bitrate 200000
# Open the can0 device at the transceiver end
ip link set can0 up
# Turn off the can0 device at the transceiver end
ip link set can0 down
```

5.8.LED usage

The A10-3588 has an onboard WHITE _ LED with an additional EXT _ LED port brought out:

- WHITE_LED: GPIO0_PC7
- EXT_LED: GPIO0_PD0

5.8.1 LED configuration

```
leds {
   compatible = " gpio-leds ";
   pinctrl -names = "default";
   pinctrl-0 = <& white_led_gpio & ext_led_gpio >;

   white_led {
     gpios = <&gpio0 RK_PC7 GPIO_ACTIVE_HIGH>;
     label = " white_led ";
      linux,default -trigger = "off";
      default-state = "on";
  };

   ext_led {
     gpios = <&gpio0 RK_PD0 GPIO_ACTIVE_HIGH>;
     label = " ext_led ";
      linux,default -trigger = "off";
      default-state = "on";
  };
};
```

5.8.2 LED control

a) WHITE_LED:

LED on: echo default-on >/sys/class/LEDs/white _ led/trigger

Turn off LEDs: echo off >/sys/class/LEDs/white _ led/trigger

Set the heartbeat effect: echo heart beat >/sys/class/LEDs/white _ led/trigger

Set Timer effect: echo timer >/sys/class/LEDs/white _ led/trigger

b) EXT_LED:

Turn on the LEDs: echo default-on >/sys/class/LEDs/ext _ led/trigger

Turn off LEDs: echo off >/sys/class/LEDs/ext _ led/trigger

Set the heartbeat effect: echo heartbeat >/sys/class/LEDs/ext _ led/trigger

Set Timer effect: echo timer >/sys/class/LEDs/ext _ led/trigger

5.9.PCIe usage

The corresponding relationship between available hardware resources of RK3588 PCIe and pcie controller nodes and PHY nodes in software is shown in the figure below

资源	模式	dts 节点	可用phy	内部 DMA
PCIe Gen3 x 4 lane	RC/EP	pcie3x4:pcie@fe150000	pcie30phy	是
PCIe Gen3 x 2 lane	RC only	pcie3x2:pcie@fe160000	pcie30phy	否
PCIe Gen3 x 1 lane	RC only	pcie2x1l0:pcie@fe170000	pcie30phy, combphy1_ps	否
PCIe Gen3 x 1 lane	RC only	pcie2x1l1: pcie@fe180000	pcie30phy, combphy2_psu	否
PCIe Gen3 x 1 lane	RC only	pcie2x1l2: pcie@fe190000	combphy0_ps	否

RK3588 has 5 PCIe controllers in total. The hardware IP is the same, but the configuration is different. One 4Lane DM mode can be used as EP, and the other 2Lane and 3 1Lane controllers can only be used as RC.

The RK3588 has two types of PCIe PHYs, one of which is the pcie 3.0 PHY with two Ports and four lanes, and the other is the pcie 2.0 PHY with three lanes, each of which is a 2.0 1 Lane, combined with SATA and USB.

The 4Lane of pcie3.0 PHY can be split according to actual requirements. After splitting, the corresponding controller needs to be configured reasonably. All configurations are completed in DTS, and the driver needs to be modified

a) A10-3588 PCIE Configuration

```
&pcie2x1l0 {
reset- gpios = <&gpio1 RK_PB4 GPIO_ACTIVE_HIGH>;
vpcie3v3-supply = <&vcc3v3_pcie20>;
  pinctrl -names = "default";
pinctrl-0 = <&pcie2x1l0_reset_gpio>;
  rockchip,perst -inactive- ms = <500>;
status = "okay";
};

&pcie2x1l2 {
reset- gpios = <&gpio3 RK_PD1 GPIO_ACTIVE_HIGH>;
  pinctrl -names = "default";
pinctrl-0 = <&pcie2x1l2_reset_gpio>;
```

```
status = "okay";
};

&pcie3x4 {
reset- gpios = <&gpio4 RK_PB6 GPIO_ACTIVE_HIGH>;
vpcie3v3-supply = <&vcc3v3_pcie30>;
  pinctrl -names = "default";
pinctrl-0 = <&pcie3x4_reset_gpio>;
status = "okay";
};
```

The A10-3588 uses pcie2x1l0, pcie2x1l2, and pcie3x4 controllers.

- The pcie2x1l0 is connected to the PCIEX1 _ 0 Switch, and the WIFI/PCIE network card can be selected through the LINK/M2 PCIE _ SW.
- Pcie2x1l2 to SATA controller
- Pcie3x4 to M2 _ M-KEY

Node description:

- Reset-gpios: reset pin attribute
- Vpcie3v3-supply: power supply pin properties

b) WIFI/PCIE network card switching

The pcie2x1l0 is connected to wifi by default. To use the PCIE network card function, please connect our expansion board A9A10 or modify the following code to make it go through the else code.

u-boot/arch/arm/mach-rockchip/board.c

```
_weak int rk_board_init (void)
{
if(nbi_i2c_read(0x88)){//LINK_DET_L no input
    run_command (" gpio set 77", 0); //PCIEX1_0_SEL GPIO2_B5 WIFI
    printf ("%s switch wifi \n", __ func __);
    is_link_board_exit = 0;
    run_command ("i2c dev 1", 0);
    run_command ("i2c mw 0x18 0x27 0x0", 0);    //switch to UART of bt
} else {//input
    run_command (" gpio clear 77", 0); //PCIEX1_0_SEL GPIO2_B5 LINK
    printf ("%s switch link\n", __ func __);
    is_link_board_exit = 1;
    run_command ("i2c dev 1", 0);
    run_command ("i2c mw 0x18 0x27 0x1", 0);    //switch to spi
}
```

5.10.SATA

i. View the device

```
ls /dev/ sd *
```

ii. Mount:

```
mount /dev/ sd */ mnt /storage
```

5.11.M.2 SSD

i. View the device

```
ls /dev/ nvme *
```

5.12.RTC use

5.12.1 Introduction

A10-3588 development board uses PT7C4363 as RTC (Real Time Clock). It is a low-power CMOS real-time clock/calendar chip. It provides a programmable clock output, an interrupt output and a power-down detector. All addresses and data are transmitted serially through I2C bus interface. The maximum bus speed is 400Kbits/s, and the embedded word address register is automatically incremented each time data is read or written.

The characteristics are as follows:

- Timable seconds, minutes, hours, weeks, days, months and years based on a 32.768 kHz crystal
- Wide operating voltage range: 1.0 ~ 5.5V
- Low sleep current: typical 0.25 μa (VDD = 3.0 V, TA = 25 ° C)
- Internally integrated oscillating capacitor

- Open drain interrupt pin

Note: When using RTC, please connect the RTC battery.

5.12.2 RTC Configuration

```
pt7c4363: pt7c4363@51 {
  compatible = "haoyu,hym8563";
  reg = <0x51>;
  #clock-cells = <0>;
  clock-frequency = <32768>;
  clock-output-names = "pt7c4363";
  wakeup-source;
};
```

Linux provides four user-space call interfaces:

- SYSFS interface:/sys/class/RTC/rtc0/
- PROCFS interface:/proc/driver/RTC
- IOCTL interface:/dev/rtc0
- The hwclock command

a) SYS FS interface:

View the date and time of the current RTC:

```
# cat /sys/class/ rtc /rtc0/date
2024-04-10
# cat /sys/class/ rtc /rtc0/time
09:30:46
#
```

b) PROCFS interface:

Print RTC related information:

```
# cat /proc/driver/ rtc
rtc_time : 09:40:22
rtc_date : 2024-04-10
alrm_time : 09:38:00
alrm_date : 2024-04-11
alarm_IRQ : no
alrm_pending : no
update IRQ enabled : no
periodic IRQ enabled : no
periodic IRQ frequency : 1
max user IRQ frequency : 64
24hr : yes
```

c) IOCTL interface

You can use ioctl to control/dev/rtc0

d) The hwclock sets the time

```
date  092115372023.30      Set the system time
hwclock  -r                Check  rtc  Time
hwclock  -w                Sync system time is up rtc
```

5.13.SPI use

5.13.1 Introduction to SPI

SPI is a high-speed, full-duplex, synchronous serial communication interface for connecting microcontrollers, sensors, storage devices, etc.

5.13.2 SPI mode of operation

The SPI operates in a master-slave mode, which typically has one master and one or more slaves, requiring at least four wires:

- CS: Chip select signal
- SCLK: clock signal
- MOSI: master data output, slave data input
- MISO: master data input, slave data output

The Linux kernel uses a combination of CPOL and CPHA to represent the four modes of operation of the current SPI:

- CPOL = 0，CPHA = 0 SPI_MODE_0
- CPOL = 0，CPHA = 1 SPI_MODE_1
- CPOL = 1，CPHA = 0 SPI_MODE_2
- CPOL = 1，CPHA = 1 SPI_MODE_3

CPOL: indicates the state of the initial level of the clock signal, where 0 is low and 1 is high.

CPHA: Indicates on which clock edge to sample, 0 for the first clock edge and 1 for the second.

5.13.3 Drive development

Take mcp2515 as an example:

Configure the DTS node

Add the SPI driver node description in kernel-5.10/arch/arm64/boot/DTS/rockchip/rk3588-namtso-a10-3588.dts as follows:

```
&spi3 {
  pinctrl -names = "default";
  pinctrl-0 = <&spi3m0_cs0 &spi3m0_pins>;
  num-cs = <1>;
  status = "okay";

  spi_can@0 {
 compatible = "microchip,mcp2515";
  reg = <0>;
  clocks = <&mcp251x_clk>;
    spi -max-frequency = <20000000>;
  status = "okay";
   interrupt-parent = <&gpio0>;
  interrupts = <RK_PC6 IRQ_TYPE_EDGE_FALLING>;
  };
};
```

- Status: Set to okay if SPI is to be enabled, or disable if it is not to be enabled.
- SPI _ can @ 0: Since CS0 is used in this example, it is set to 00 here. If CS1 is used, it is set to 01.
- Compatible: The attribute here must be consistent with the member compatible in the struct: of _ device _ ID in the driver.
- Reg: This is consistent with the SPI _ can @ 0. In this example, it is set to: 0.
- Spi-max-frequency: This sets the maximum frequency used by the SPI. The A10-3588 supports up to 50000000.

5.13.4 The interface uses

Linux provides a limited SPI user interface. If you do not need to use IRQ or other kernel-driven interfaces, you can consider using the interface spidev to write user-level programs to control SPI devices. The corresponding path in the a10-3588 development board is:/dev/spidev3.0

Driver code corresponding to spidev: kernel-5.10/drivers/SPI/spidev. C.

The kernel config needs to have the SPI _ SPIDEV selected:

```
| Symbol: SPI_SPIDEV [=y]
| Type : tristate
| Prompt: User mode SPI device driver support
| Location:
| -> Device Drivers
| -> SPI support (SPI [=y])
| Defined at drivers/ spi /Kconfig:684
| Depends on: SPI [=y] && SPI_MASTER [=y]
```

DTS configuration is as follows:

```
&spi3  {
        pinctrl -names  =  "default";
    pinctrl-0  =  <&spi3m0_cs0  &spi3m0_pins>;
    num-cs  =  <1>;
    status  =  "okay";

    spidev3:  spidev@00{
       compatible  =  " rockchip,spidev ";
       status  =  "okay";
          reg  =  <0x0>;
      spi -max-frequency  =  <50000000>;
        };
};
```

Please refer to kernel-5.10/Documentation/SPI/spidev. Rst for detailed instructions on using spidev.

5.14.UART

a) Device node

The left serial port is UART0, and the right serial port is UART1:

```
/dev/ttyWCH0
/dev/ttyWCH1
```

b) Baud rate setting

Take UART0 as an example:

```
stty -F /dev/ttyWCH0 ispeed 115200 ospeed 115200 cs8
stty -F /dev/ttyWCH0 speed 115200 cs8 - parenb - cstopb -echo
```

c) Send data

```
echo “ aaa ” > /dev/ttyWCH0
```

d) Read the data

```
cat /dev/ttyWCH 0
```

5.15.Watchdog uses

This chapter mainly introduces the use of A10-3588 3588 development board hardware watchdog.

a) DTS configuration

The DTS node for watchdog of A10-3588 3588 is defined in the kernel-5.10/arch/arm64/boot/DTS/rockchip/rk3588-namtso-a10-3588.dtsi dtsi file as follows:

```
namtso_wdt {
    compatible = " linux,wdt-namtso ";
    status = "okay";
    hw_margin_ms = <500>;
    hw-gpios = <&gpio0 RK_PB0 GPIO_ACTIVE_HIGH>;
    pinctrl -names = "default";
    pinctrl-0 = <&wdt_gpio >;
};
```

The watchdog watchdog is turned on by default, and the kernel automatically feeds the dog every 0.55 seconds.

b) Use

The driver file for watchdog is kernel-5.10/drivers/watchdog/Namtso _ WDT. C.

Turn on the watchdog:

```
echo 1 > /sys/class/ namtso_watchdog /enable
```

Turn off the watchdog:

```
echo 0 > /sys/class/ namtso_watchdog /enable
```