

IDO-EVB7606

Linux开发手册

深圳触觉智能科技有限公司

www.industio.cn

文档修订历史

版本	PCBA版本	修订内容	修订	审核	日期
V1.0	V1B	创建文档	LJH		2026/09/07

1. Linux开发

1.1. 获取 SDK

下载路径：百度网盘/IDO-EVB7606/4.软件资料/系统SDK/

1.2. 校验md5值

```Bash
$ md5sum IDO-EVB7606_Linux6.1_RKR5.tar.gz
```

1.3. 解压

首先准备一个空文件夹用于存放 SDK，建议在 home 目录下，本文以~/ido-evb7606 为例。

```Bash
$ mkdir ~/ido-evb7606
$ cd ~/ido-evb7606
$ tar -xvf IDO-EVB7606_Linux6.1_RKR5.tar.gz -C ./

$ cd RK3576_Linux6.1_RKR5
$ git checkout EVB7606-Buildroot2024-Debian12

#查看分支(默认分支为EVB7606-Buildroot2024-Debian12)
$ git branch
* EVB7606-Buildroot2024-Debian12
  EVB7606-Buildroot2024-Debian12-preempt-rt
  EVB7606-Buildroot2024-Debian12-xenomai-rt

#获取代码
$ git reset --hard
```

1.4. SDK目录介绍

```Bash
$ tree -L 1
.
├── app
├── buildroot
├── build.sh -> device/rockchip/common/scripts/build.sh
├── common -> device/rockchip/common
├── Copyright_Statement.md -> docs/licenses/LICENSE
├── debian
├── debian_rootfs
├── device
├── docs
├── external
├── kernel -> kernel-6.1
├── kernel-6.1
├── Makefile -> device/rockchip/common/Makefile
├── output
├── prebuilts
├── README.md -> device/rockchip/common/README.md
├── rkbin
├── rkflash.sh -> device/rockchip/common/scripts/rkflash.sh
├── rockdev -> output/firmware
├── tools
├── u-boot
└── yocto
```

```Bash
app：存放上层应用 APP，主要是一些应用Demo。
buildroot：基于 Buildroot（2024）开发的根文件系统。
device/rockchip：存放芯片板级配置以及SDK编译和打包固件的脚本和文件等。
docs：存放通用开发指导文档、芯片平台相关文档、Linux 系统开发相关文档、其他参考文档等。
external：存放第三方相关仓库，包括显示、音视频、摄像头、网络、安全等。
kernel：存放 Kernel 开发的代码。
output：存放每次生成的固件信息、编译信息、XML、主机环境等。
prebuilts：存放交叉编译工具链。
rkbin：存放 Rockchip 相关二进制和工具。
rockdev：存放编译输出固件,实际软链接到 output/firmware 。
tools：存放 Linux 和 Window 操作系统下常用工具。
u-boot：存放基于 v2017.09 版本进行开发的 U-Boot 代码。
yocto: 存放yocto相关编译脚本和代码。
```

注意：

1. SDK 采用交叉编译，所以要在 X86_64 电脑上使用 SDK，不要将 SDK 下载到板子上。

2. 编译环境请使用 Ubuntu22.04（真机或 虚拟机），如果使用其他版本可能导致编译出错。
3. 不要在虚拟机共享文件夹以及非英文目录存放、解压SDK。
4. 获取、编译 SDK 请全程使用普通用户，不允许也不需要使用 root 权限（除非需要 apt 安装软件）

1.5. 配置文件介绍

在 `device/rockchip/rk3576` 目录下，有不同板型的配置文件(xxxx_defconfig)，用于管理 SDK 每个环节的编译配置。rockchip_rk3576_ido_evb7606_v1*_defconfig 作为基础的配置。其他版型的配置文件会引用 rockchip_rk3576_ido_evb7606_v1*_defconfig 并且覆盖其中的一些变量配置。 以 rockchip_rk3576_ido_evb7606_v1_hdmi_defconfig 做介绍：

Bash

```
RK_BUILDROOT_BASE_CFG="rk3576"
RK_CHIP_FAMILY="rk3576"                                 #芯片类型
# RK_WIFIBT is not set
RK_UBOOT_SPL=y
RK_KERNEL_DTS_NAME="ido-evb7606-v1a-hdmi"               # 板型 dts 名称
RK_USE_FIT_IMG=y
RK_KERNEL_CFG="rockchip_linux_rk3576_defconfig"             # kernel 编译配置
```

1.6. 分区说明

parameter 文件中包含了固件的分区信息，以 parameter.txt 为例：

```Bash
FIRMWARE_VER: 1.0
MACHINE_MODEL: RK3576
MACHINE_ID: 007
MANUFACTURER: RK3576
MAGIC: 0x5041524B
ATAG: 0x00200800
MACHINE: 0xffffffff
CHECK_MASK: 0x80
PWR_HLD: 0,0,A,0,1
TYPE: GPT
GROW_ALIGN: 0
CMDLINE: mtdparts=:0x00002000@0x00004000(uboot),0x00002000@0x00006000(misc),0x00020000@0x00008000(boot),0x00040000@0x00028000(recovery),0x00010000@0x00068000(backup),0x00800000@0x00078000(userdata),0x00040000@0x00878000(oem),-@0x008B8000(rootfs:grow)
uuid:rootfs=614e0000-0000-4b53-8000-1d28000054a9
uuid:boot=7A3F0000-0000-446A-8000-702F00006273
```

CMDLINE 属性是我们关注的地方，以 uboot 为例， 0x00002000@0x00004000(uboot) 中 0x00002000 为uboot 分区的起始位置，0x00004000 为分区的大小，以此类推。

2. 编译

2.1. 选择项目

```Bash
xxx$ ./build.sh lunch
1. rockchip_rk3576_ido_evb7606_v1_hdmi_debian12_defconfig
2. rockchip_rk3576_ido_evb7606_v1_hdmi_defconfig
3. rockchip_rk3576_ido_evb7606_v1_hdmi_ubuntu2204_defconfig
4. rockchip_rk3576_ido_evb7606_v1_mipi1920x1200_debian12_defconfig
5. rockchip_rk3576_ido_evb7606_v1_mipi1920x1200_defconfig
6. rockchip_rk3576_ido_evb7606_v1_mipi1920x1200_ubuntu2204_defconfig
Which would you like? [1]:
...
```

选择对应的数字即可，两个配置区别如下：

rockchip_rk3576_ido_evb7606_v1_hdmi_xxx_defconfig：输出HDMI显示

rockchip_rk3576_ido_evb7606_v1_mipi_1920x1200_xxx_defconfig：输出MIPI 1920x1200显示

2.2. 指定编译文件系统

设置环境变量RK_ROOTFS_SYSTEM为buildroot/debian/ubuntu，即可指定编译的文件系统。

如需要编译buildroot文件系统：

```Bash
export RK_ROOTFS_SYSTEM=buildroot
```

如需要编译debian文件系统：

```Bash
export RK_ROOTFS_SYSTEM=debian
```

如需要编译ubuntu文件系统：

```Bash
export RK_ROOTFS_SYSTEM=ubuntu
```

2.3. 自动编译

```Bash
$ ./build.sh
```

固件编译后生成路径：rockdev/update.img

2.4. 分步编译

2.4.1. 编译uboot

```Bash
./build.sh uboot
```

固件编译后生成路径：rockdev/uboot.img

2.4.2. 编译kernel

```Bash
./build.sh kernel
```

固件编译后生成路径：rockdev/boot.img

2.4.3. 编译buildroot系统

```Bash
./build.sh buildroot
```

固件编译后生成路径：rockdev/rootfs.img

2.4.4. 编译ubuntu镜像

```Bash
./build.sh ubuntu
```

2.4.5. 编译debian镜像

```Bash
./build.sh debian
```

2.4.6. 编译recovery

```Bash
./build.sh recovery
```

固件编译后生成路径：rockdev/recovery.img

2.4.7. 固件打包

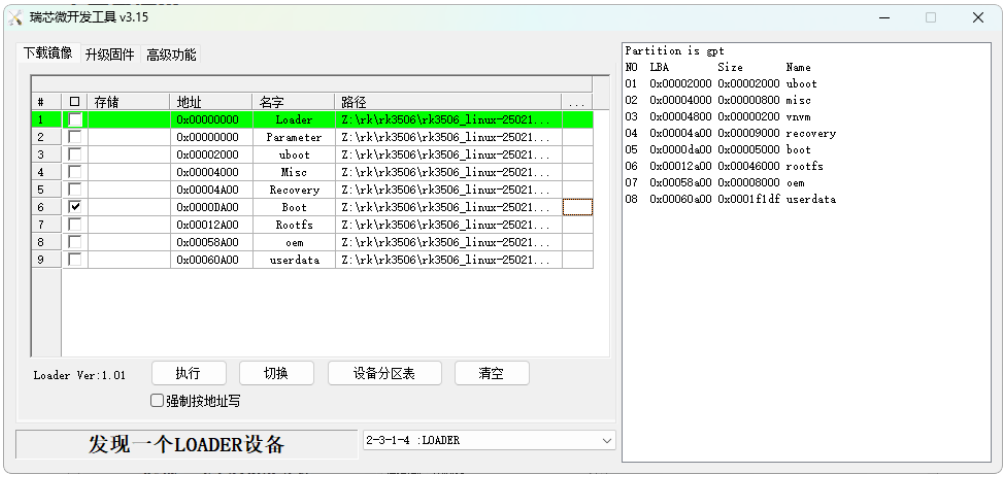
```bash
./build.sh updateimg
```

3. 烧录说明

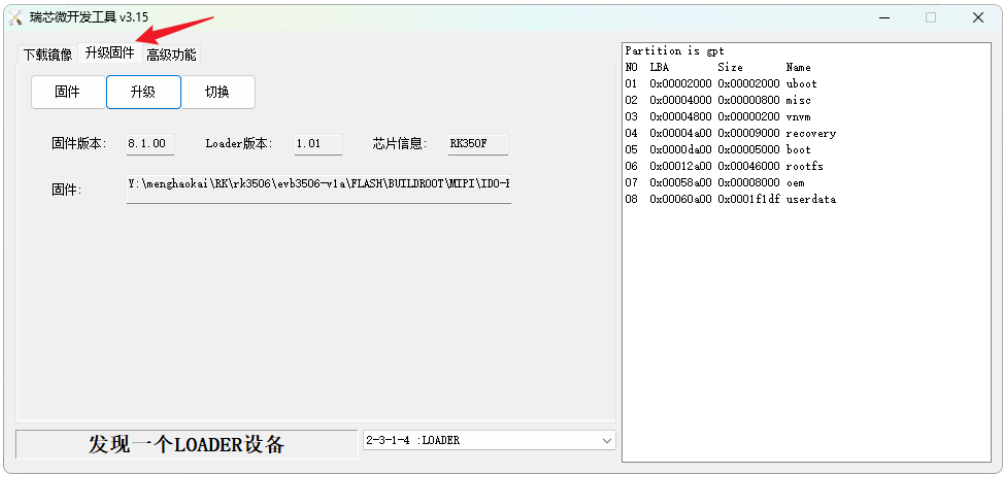
注意：板子需要进入到烧录模式，具体方法可以参考：《Purple Pi OH2 开发板固件烧录手册》

3.1. 整包烧录

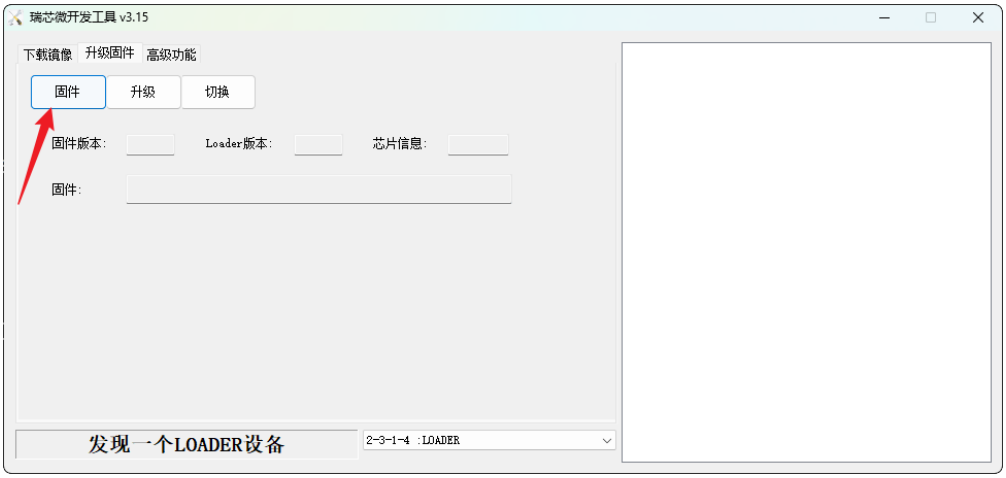
3.1.1. 运行RK开发工具



3.1.2. 点击升级固件

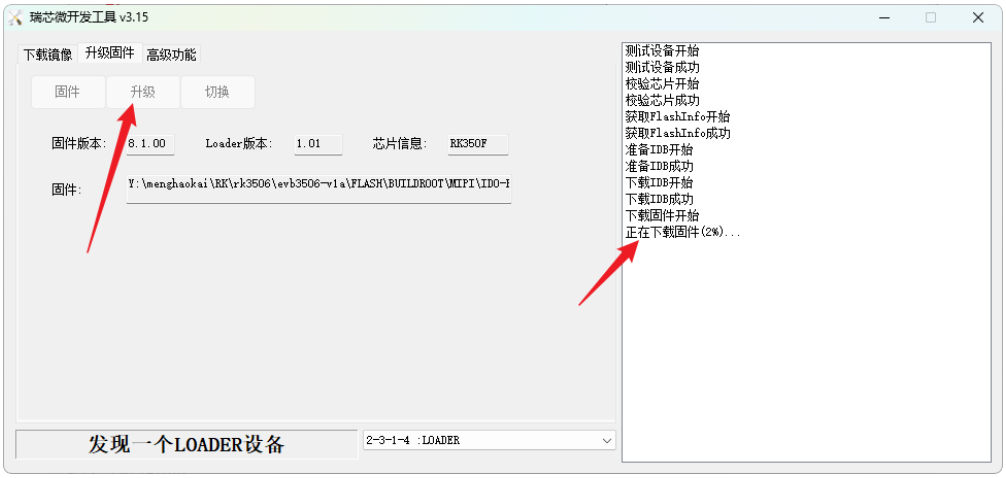


3.1.3. 点击固件



选择：rockdev/update.img

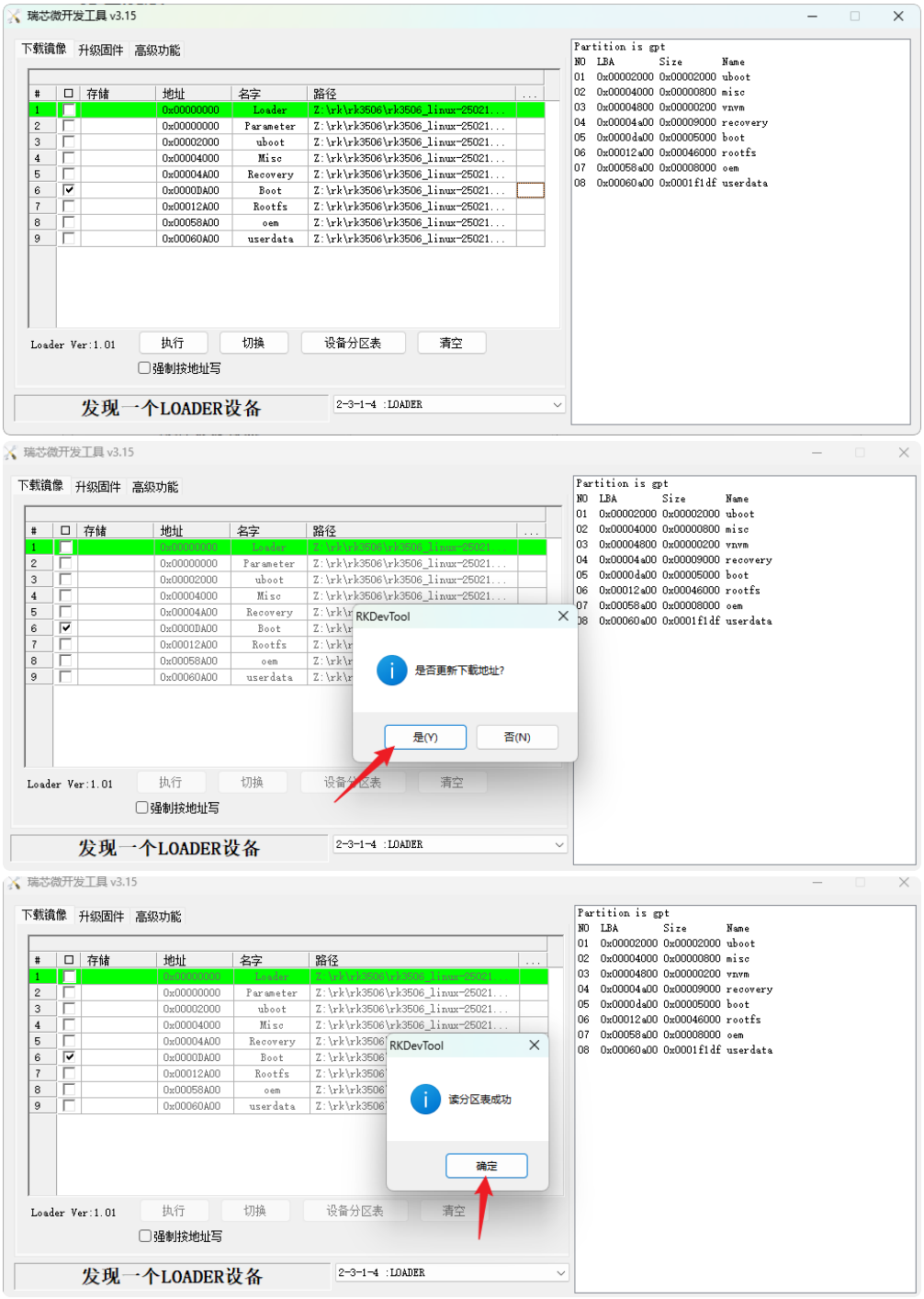
3.1.4. 点击升级



点击升级后，右侧会开始下载固件；下载完成后，板子会自动启动。

3.2. 分包烧录

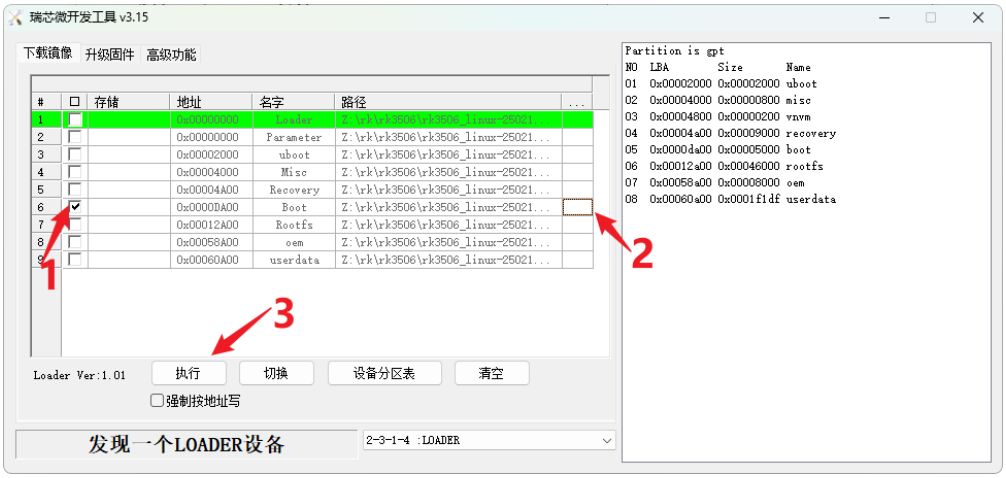
3.2.1. 更新下载地址



3.2.2. 烧录

1. 在对应的分区前选择对号
2. 然后点击右侧去选择对应的镜像

3. 点击执行即可烧录



4. 驱动开发

4.1. CAN

4.1.1. CAN简介

CAN(Controller Area Network)总线，即控制器局域网总线，是一种有效支持分布式控制或实时控制的串行通信网络。CAN总线是一种在汽车上广泛采用的总线协议，被设计作为汽车环境中的微控制器通讯。

4.1.2. DTS 节点配置

公共配置：kernel/arch/arm64/boot/dts/rk3576.dtsi

```
can0: can@2ac00000 {
  compatible = "rockchip,rk3576-canfd";
  reg = <0x0 0x2ac00000 0x0 0x1000>;
  interrupts = <GIC_SPI 121 IRQ_TYPE_LEVEL_HIGH>;
  clocks = <&cru CLK_CAN0>, <&cru HCLK_CAN0>;
  clock-names = "baudclk", "apb_pclk";
  resets = <&cru SRST_CAN0>, <&cru SRST_H_CAN0>;
  reset-names = "can", "can-apb";
  dmas = <&dmac0 20>;
  dma-names = "rx";
  status = "disabled";
};

can1: can@2ac10000 {
  compatible = "rockchip,rk3576-canfd";
  reg = <0x0 0x2ac10000 0x0 0x1000>;
  interrupts = <GIC_SPI 122 IRQ_TYPE_LEVEL_HIGH>;
  clocks = <&cru CLK_CAN1>, <&cru HCLK_CAN1>;
  clock-names = "baudclk", "apb_pclk";
  resets = <&cru SRST_CAN1>, <&cru SRST_H_CAN1>;
  reset-names = "can", "can-apb";
  dmas = <&dmac1 21>;
  dma-names = "rx";
  status = "disabled";
};
```

4.1.3. 配置CAN

板级配置：ido-evb7609-v1a-extend.dtsi

```
&can0 {
  assigned-clocks = <&cru CLK_CAN0>;
  assigned-clock-rates = <200000000>;
  pinctrl-names = "default";
  pinctrl-0 = <&can0m2_pins>;
  status = "okay";
};

&can1 {
  assigned-clocks = <&cru CLK_CAN1>;
  assigned-clock-rates = <200000000>;
  pinctrl-names = "default";
  pinctrl-0 = <&can1m3_pins>;
  status = "okay";
};
```

4.1.4. 通信测试

```
#关闭can0设备
ip link set can0 down

#普通can协议（1.0）
ip link set can0 type can bitrate 1000000 dbitrate 1000000 fd on

#查看can信息，波特率等等
ip -details link show can0

#启动can0
ip link set can0 up

#执行candump，阻塞等待can0接收
candump can0

#can格式发送
cansend can0 123#DEADBEEF12345679
```

4.2. Ethernet

4.2.1. 公共配置

`kernel/arch/arm64/boot/dts/rk3576.dtsi`

```
gmac0: ethernet@2a220000 {
    compatible = "rockchip,rk3576-gmac", "snps,dwmac-4.20a";
    reg = <0x0 0x2a220000 0x0 0x10000>;
    interrupts = <GIC_SPI 293 IRQ_TYPE_LEVEL_HIGH>,
           <GIC_SPI 298 IRQ_TYPE_LEVEL_HIGH>;
    interrupt-names = "macirq", "eth_wake_irq";
    rockchip,grf = <&sdgmac_grf>;
    rockchip,php_grf = <&ioc_grf>;
    clocks = <&cru CLK_GMAC0_125M_SRC>, <&cru CLK_GMAC0_RMII_CRU>,
       <&cru PCLK_GMAC0>, <&cru ACLK_GMAC0>,
       <&cru CLK_GMAC0_PTP_REF>;
    clock-names = "stmmaceth", "clk_mac_ref",
           "pclk_mac", "aclk_mac",
           "ptp_ref";
    resets = <&cru SRST_A_GMAC0>;
    reset-names = "stmmaceth";
    power-domains = <&power RK3576_PD_SDGMAC>;

    dma-coherent;
    snps,mixed-burst;
    snps,tso;

    snps,axi-config = <&gmac0_stmmac_axi_setup>;
    snps,mtl-rx-config = <&gmac0_mtl_rx_setup>;
    snps,mtl-tx-config = <&gmac0_mtl_tx_setup>;
    status = "disabled";

    mdio0: mdio {
      compatible = "snps,dwmac-mdio";
      #address-cells = <0x1>;
      #size-cells = <0x0>;
    };

    gmac0_stmmac_axi_setup: stmmac-axi-config {
      snps,wr_osr_lmt = <4>;
      snps,rd_osr_lmt = <8>;
      snps,blen = <0 0 0 0 16 8 4>;
    };

    gmac0_mtl_rx_setup: rx-queues-config {
      snps,rx-queues-to-use = <1>;
      queue0 {};
    };

    gmac0_mtl_tx_setup: tx-queues-config {
```

```
        snps,tx-queues-to-use = <1>;
        queue0 {};
      };
    };
```

4.2.2. 板级的配置

```Bash
&gmac0 {
  /* Use rgmii-rxid mode to disable rx delay inside Soc */
  phy-mode = "rgmii";
  clock_in_out = "output";

  snps,reset-gpio = <&gpio0 RK_PC7 GPIO_ACTIVE_LOW>;
  snps,reset-active-low;
  /* Reset time is 20ms, 100ms for rtl8211f */
  snps,reset-delays-us = <0 20000 100000>;

  pinctrl-names = "default";
  pinctrl-0 = <&eth0m0_miim
         &eth0m0_tx_bus2
         &eth0m0_rx_bus2
         &eth0m0_rgmii_clk
         &eth0m0_rgmii_bus >;
         //&ethm0_clk0_25m_out>;

  tx_delay = <0x1a>;
  rx_delay = <0x26>;

  phy-handle = <&rgmii_phy0>;
  status = "okay";
};
```

4.3. RTC

Purple Pi OH2 采用 HYM8563 作为RTC(*Real Time Clock*)，需要接入 RTC 电池给 RTC 芯片供电才可以保证在短时间系统断电后 RTC 能正常运行。

DTS配置参考: `kernel/arch/arm64/boot/dts/ido-evb7609_v1b.dtsi`

驱动参考: `kernel/drivers/rtc/rtc-hym8563.c`

Linux下的rtc使用方法为:

```Bash
#同步网络时间
$ ntpdate cn.pool.ntp.org

#查看当前 RTC 的日期和时间:
$ cat /sys/class/rtc/rtc0/date
2025-05-27

$ cat /sys/class/rtc/rtc0/time
10:10:30

$ 查看系统时间
$ date
Tue May 27 18:10:41 CST 2025

#设置系统时间
$ date -s "2024-10-09 14:02:30"

#将rtc时间调整为与目前的系统时间一致
$ hwclock -w

#获取硬件rtc当前时间（断电重启读取时间没有太大偏差）
$ hwclock -r
2024-10-09 14:02:35.945604+00:00

#将rtc时间设置为系统时间
hwclock --systohc
```

4.4. UART

4.4.1. DTS配置

```Bash
&uart3 {
    status = "okay";
    pinctrl-names = "default";
    pinctrl-0 = <&uart3m0_xfer>;
};
```

配置好串口后，硬件接口对应软件上的节点为:

```
UART3:   /dev/ttyS3
```

4.4.2. 收发测试

使用microcom可以进行串口收发测试，命令如下：

```
root@rk3576:~# microcom -s 115200 -p /dev/ttyS3
```

注意：测试完成，按Ctrl+x退出。