Linux GMAC/EMAC

开发指南

版本号: 1.7

发布日期: 2023.11.24

版本历史

版本号	日期	制/修订人	内容描述
1.0	2020.6.20	AWA1637	建立初版
1.1	2021.5.10	AWA1637	添加 R528 说明
1.2	2021.8.16	AWA1730	添加 AC200 AC300 说明
1.3	2022.5.16	AWA1730	增加驱动适用范围
1.4	2022.10.28	XAA0250	1. 增加 BSP 独立仓库相关说明 2. 增加 emac 相关说明
1.41	2022.11.28	XAA0250	1. 适用产品列表新增 t113-i
1.5	2023.2.7	XAA0250	1. 新增常用调试工具章节
1.6	2023.10.24	XAA0250	1. 更新 jumbo 帧测试方法
1.7	2023.11.24	XAA0250	1. 添加 T113S3P&T113-S4&T113-S4P 相关信息

目　录

插　图

1 概述

1.1 编写目的

介绍以太网模块（GMAC/EMAC）配置及调试方法，为以太网模块开发提供参考。

1.2 适用范围

表 1-1: 适用产品列表

产品名称	内核版本	驱动文件
V853	Linux-4.9	sunxi-gmac.c
H616	Linux-4.9	sunxi-gmac.c
H618	Linux-5.4	sunxi-gmac.c
R528	Linux-5.4	sunxi-gmac.c
T507	Linux-5.10	sunxi-gmac.c
A40I	Linux-5.10	sunxi-gmac.c/sunxi-emac.c
T113	Linux-5.4	sunxi-gmac.c
T113S3P&T113-S4&T113-S4P	Linux-5.4	sunxi-gmac.c

1.3 相关人员

以太网模块开发/维护人员。

2 相关术语介绍

表 2-1: 以太网相关术语介绍

术语	解释说明
SUNXI	Allwinner 一系列 SOC 硬件平台
MAC	Media Access Control，媒体访问控制协议
GMAC	千兆以太网控制器
EMAC	以太网控制器
PHY	物理收发器
MII	Media Independent Interface，媒体独立接口，是 MAC 与 PHY 之间的接口
RMII	简化媒体独立接口
RGMII	简化千兆媒体独立接口

3 模块介绍

3.1 模块功能介绍

3.1.1 以太网简介

以太网是一种局域网通信技术，遵循 IEEE802.3 协议规范，包括 10M、100M、1000M 和 10G 等多种速率的以太网。以太网在 TCP/IP 协议族中的位置如下图所示。

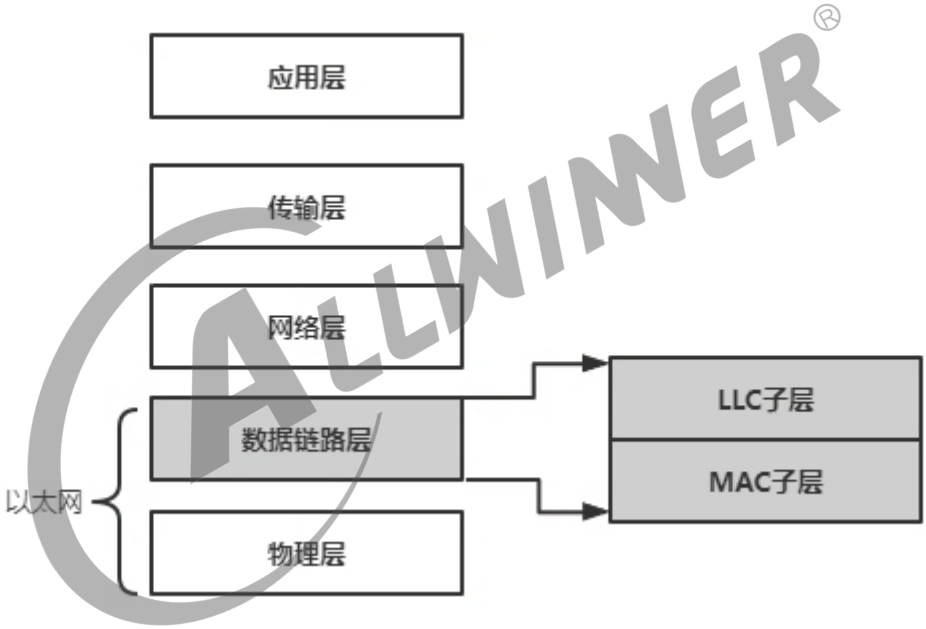

图 3-1: 以太网在 TCP/IP 协议族中的位置

以太网与 TCP/IP 协议族的物理层（L1）和数据链路层（L2）相关，其中数据链路层包括逻辑链路控制（LLC）和媒体访问控制（MAC）子层。

3.1.2 网络设备框架

Linux 内核中网络设备框架如下图所示。

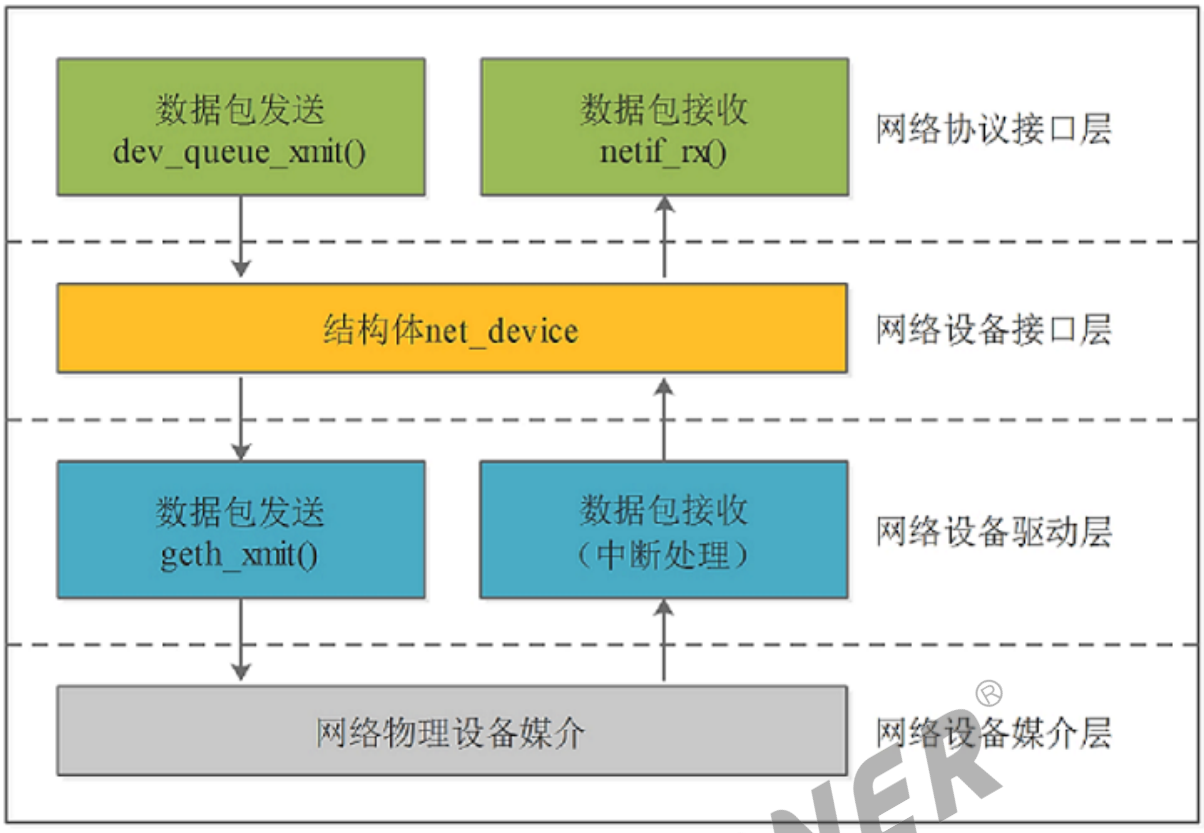

图 3-2: 网络设备框架

从上至下分为 4 层：

（1）网络协议接口层：向网络协议层提供统一的数据包收发接口，通过 dev_queue_xmit() 发送数据，并通过 netif_rx() 接收数据。

（2）网络设备接口层：向协议接口层提供统一的用于描述网络设备属性和操作的结构体 net_device，该结构体是设备驱动层中各函数的容器。

（3）网络设备驱动层：实现 net_device 中定义的操作函数指针（通常不是全部），驱动硬件完成相应动作。

（4）网络设备媒介层：完成数据包发送和接收的物理实体，包括网络适配器和具体的传输媒介。

3.2 模块配置介绍

3.2.1 menuconfig 配置说明

对于 longan，可以直接在 longan 的根目录执行./build.sh menuconfig；

对于 Tina 的环境，可以在根目录执行 make kernel_menuconfig 进入 menuconfig 配置界面。

（1）配置网络协议栈，如下图所示。

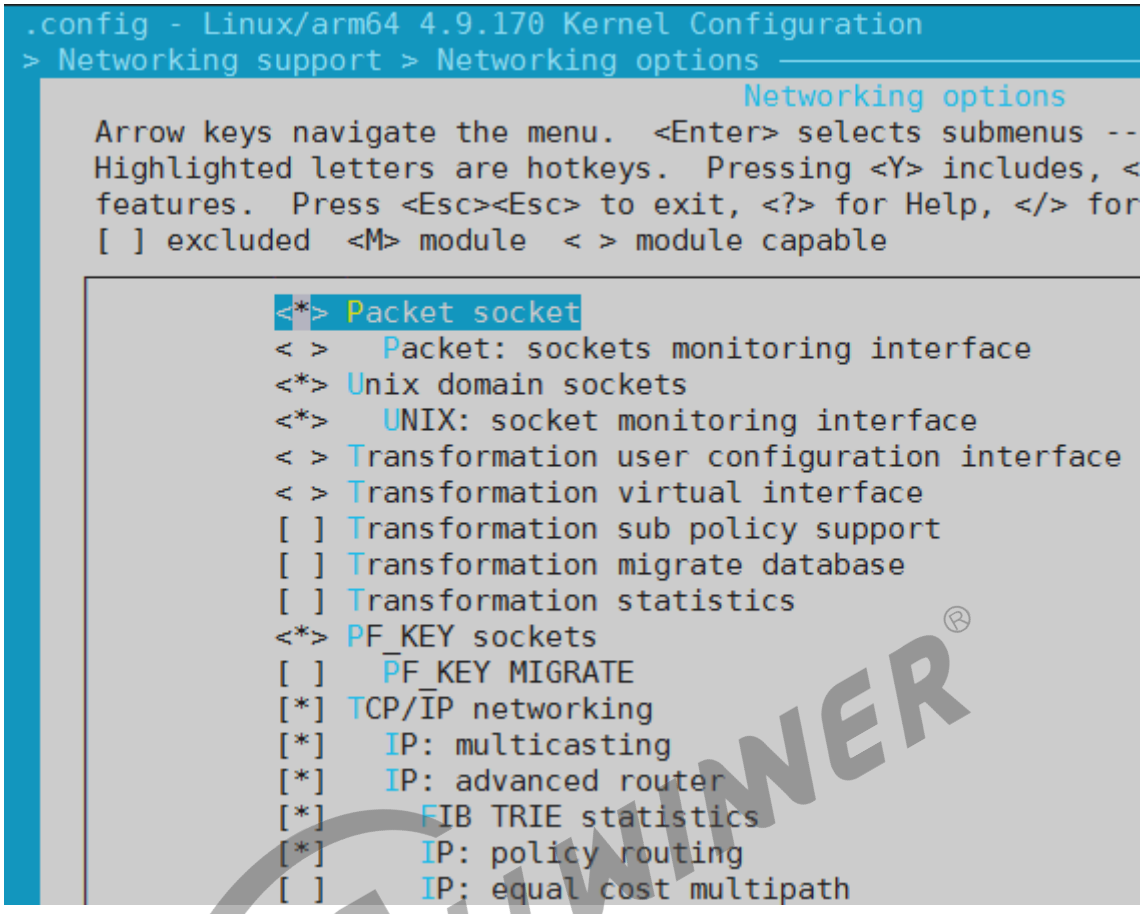

图 3-3: 网络协议栈配置

3.2.1.1 非 T113S3P&T113-S4&T113-S4P 方案

非 T113S3P&T113-S4&T113-S4P 方案请按照如下图片配置 menuconfig

（2）勾选 GMAC 驱动，如下图所示。

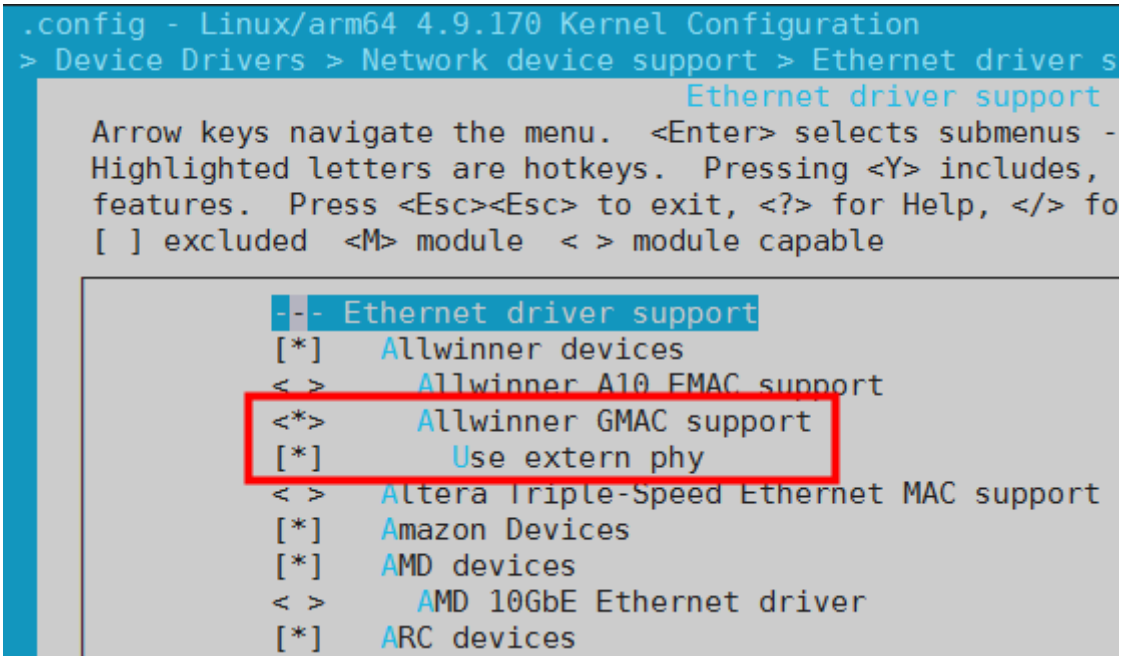

图 3-4: GMAC 驱动配置

说明

如果使用 SOC 内置 PHY，则需完成步骤 3 和步骤 4 配置，目前只有 TV 系列、部分 H 系列平台有使用内置 EPHY，如 TV303、H3、H6、H313、H616，其它平台可直接跳过。

（3）勾选 SUNXI-EPHY 驱动，如下图所示：首先，SOC 内部 ACX00 封装了 EPHY，因此需要先支持 ACX00 设备。

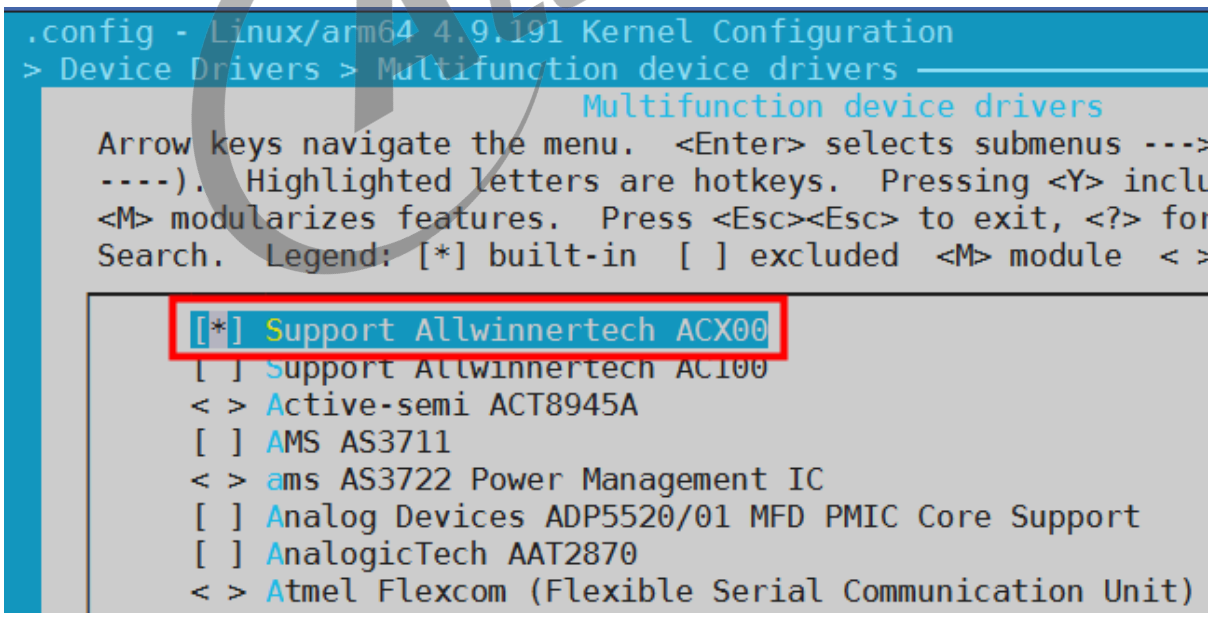

图 3-5: ACX00 驱动配置

然后，勾选 SUNXI-EPHY 驱动。

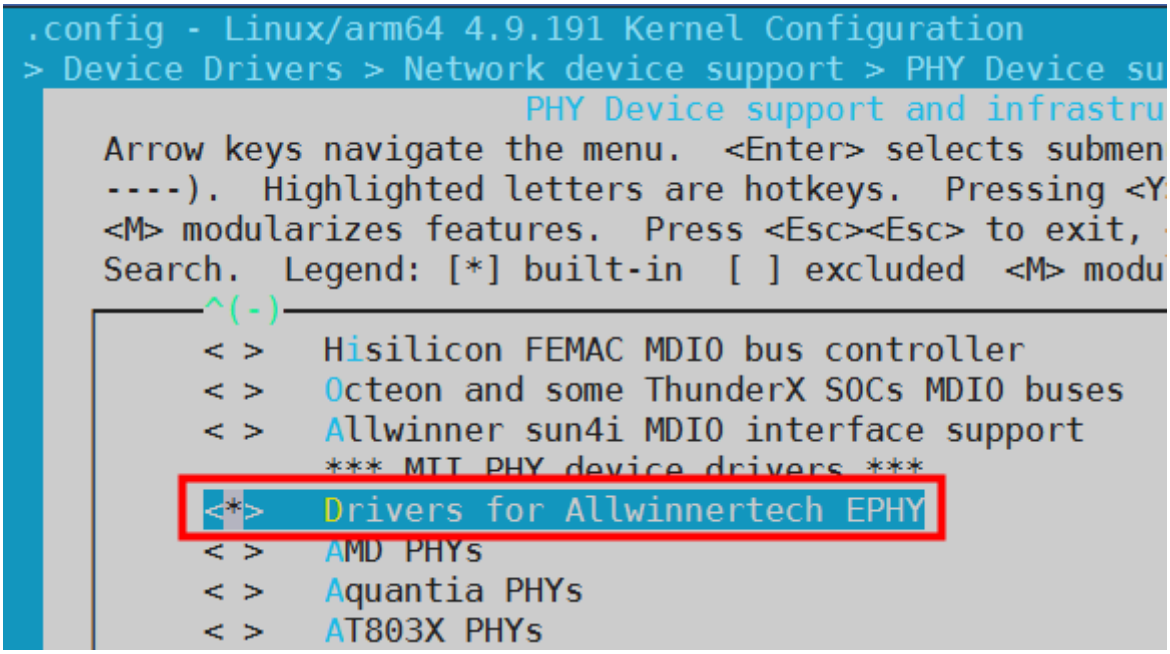

图 3-6: EPHY 驱动配置

3.2.1.2 T113S3P&T113-S4&T113-S4P 方案

（2）勾选 GMAC 驱动和 GMAC 对应的 MDIO 驱动，如下图所示。

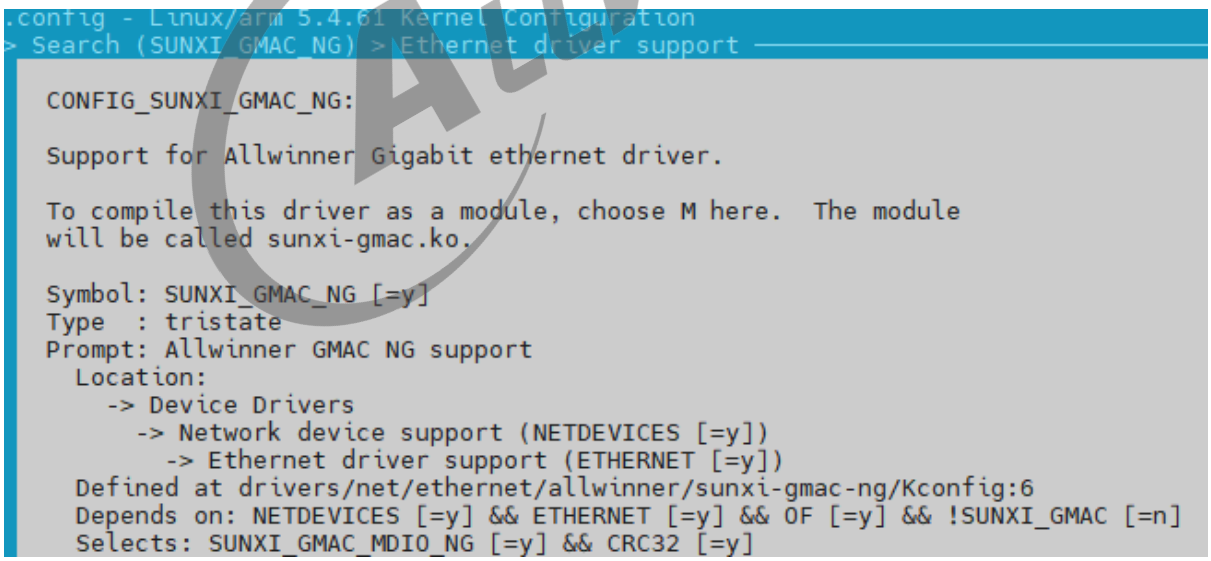

图 3-7: GMAC 配置

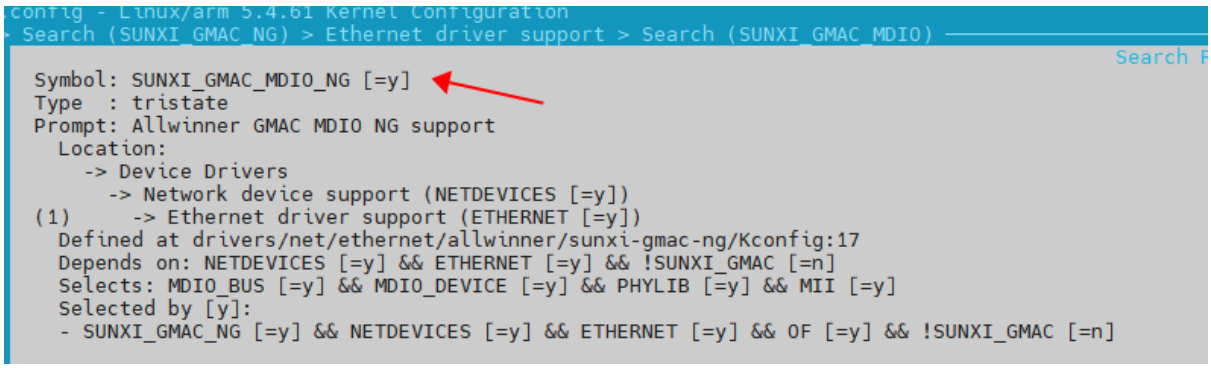

图 3-8: MDIO 配置

3.2.2 BSP 独立仓库 menuconfig 配置说明

对于 longan，可以直接在 longan 的根目录执行./build.sh menuconfig；

对于 Tina 的环境，可以在根目录执行 make kernel_menuconfig 进入 menuconfig 配置界面。

（1）勾选 GMAC 和 GMAC 对应的 MDIO 驱动

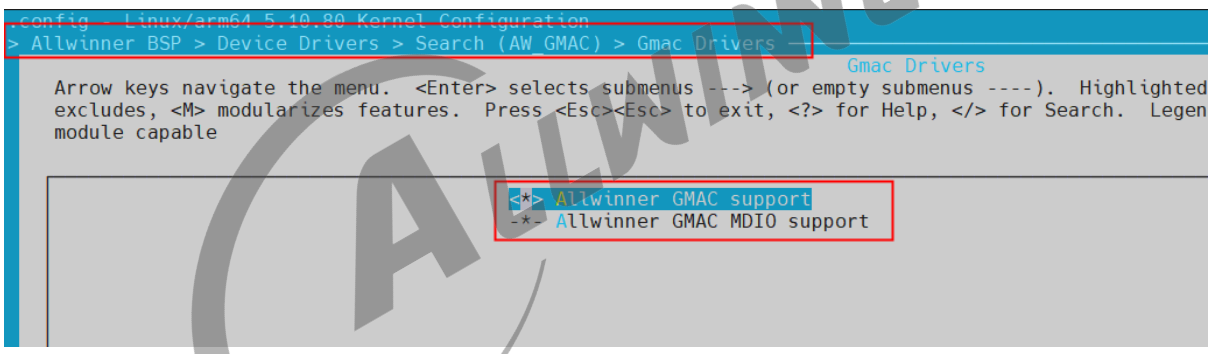

图 3-9: BSP 独立仓库 GMAC 配置

（2）若平台支持 EMAC 这个 IP，勾选 EMAC 和 EMAC 对应的 MDIO 驱动

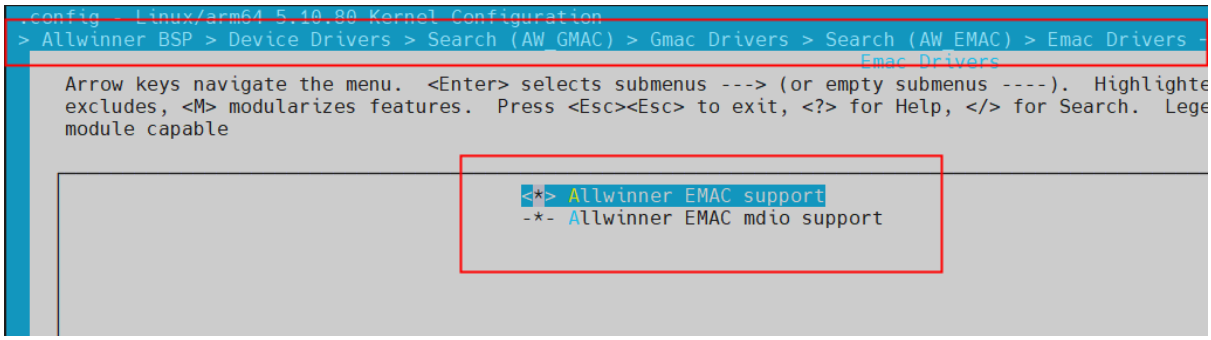

图 3-10: BSP 独立仓库 EMAC 配置

3.2.3 device tree 配置说明

3.2.3.1 非 T113S3P&T113-S4&T113-S4P 方案

linux-4.9 内核下的配置如下所示:

```
gmac0: eth@05020000 {
    compatible = "allwinner,sunxi-gmac";
    reg = <0x0 0x05020000 0x0 0x10000>,
          <0x0 0x03000030 0x0 0x4>;
    interrupts = <GIC_SPI 14 IRQ_TYPE_LEVEL_HIGH>;
    interrupt-names = "gmacirq";
    clocks = <&clk_gmac0>, <&clk_ephy_25m>;
    clock-names = "gmac", "ephy";
    device_type = "gmac0";
    pinctrl-0 = <&gmac_pins_a>;
    pinctrl-1 = <&gmac_pins_b>;
    pinctrl-names = "default", "sleep";
    phy-mode;
    tx-delay = <7>;
    rx-delay = <31>;
    phy-rst;
    gmac-power0;
    gmac-power1;
    gmac-power2;
    status = "disabled";
};
```

（1）”compatible” 表征具体的设备, 用于驱动和设备的绑定；

（2）”reg” 设备使用的地址；

（3）”interrupts” 设备使用的中断；

（4）”clocks” 设备使用的时钟；

（5）”pinctrl-0” 设备 active 状态下的 GPIO 配置；

（6）”pinctrl-1” 设备 suspend 状态下的 GPIO 配置；

（7）”phy-mode” GMAC 与 PHY 之间的物理接口，如 MII、RMII、RGMII 等；

（8）”tx-delay” tx 时钟延迟，tx-delay 取值 0-7，一档约 536ps（皮秒）；

（9）”rx-delay ” rx 时钟延迟，rx-delay 取值 0-31，一档约 186ps（皮秒）；

（10）”phy-rst” PHY 复位脚；

（11）”gmac-powerX” gmac 电源脚，根据实际情况配置；

（12）”status” 是否使能该设备节点。

在 linux-5.4 中，GMAC 的配置与 linux-4.9 内核配置有些不同，区别主要体现在 clock 的配置上：

```
gmac0: eth@4500000 {
    compatible = "allwinner,sunxi-gmac";
    reg = <0x0 0x04500000 0x0 0x10000>,
        <0x0 0x03000030 0x0 0x4>;
    interrupts-extended = <&plic0 62 IRQ_TYPE_LEVEL_HIGH>;
    interrupt-names = "gmacirq";
    clocks = <&ccu CLK_BUS_EMAC0>, <&ccu CLK_EMAC0_25M>;
    clock-names = "gmac", "ephy";
    resets = <&ccu RST_BUS_EMAC0>;
    device_type = "gmac0";
    pinctrl-0 = <&gmac_pins_a>;
    pinctrl-1 = <&gmac_pins_b>;
    pinctrl-names = "default", "sleep";
    phy-mode = "rgmii";
    use_ephy25m = <1>;
    tx-delay = <7>;
    rx-delay = <31>;
    phy-rst = <&pio PA 14 GPIO_ACTIVE_LOW>;
    gmac-power0;
    gmac-power1;
    gmac-power2;
    status = "disabled";
};
```

其中 gmac_pins_a, gmac_pins_a 为 GMAC 的引脚配置的配置节点。

lixnu4.9 中该配置的路径为 arch/arm64（32 位平台为 arm）/boot/dts/sunxi/xxxx-pinctrl.dtsi，具体配置如下所示：

```
    gmac_pins_a: gmac@0 {
        allwinner,pins = "PI0", "PI1", "PI2", "PI3",
                "PI4", "PI5", "PI6", "PI7",
                "PI8", "PI9", "PI10", "PI11",
                "PI12", "PI13", "PI14", "PI15",
                "PI16";
        allwinner,function = "gmac0";
        allwinner,muxsel = <2>;
        allwinner,drive = <3>;
        allwinner,pull = <0>;
    };

    gmac_pins_b: gmac@1 {
        allwinner,pins = "PI0", "PI1", "PI2", "PI3",
                "PI4", "PI5", "PI6", "PI7",
                "PI8", "PI9", "PI10", "PI11",
                "PI12", "PI13", "PI14", "PI15",
                "PI16";
        allwinner,function = "io_disabled";
        allwinner,muxsel = <7>;
        allwinner,drive = <3>;
        allwinner,pull = <0>;
    };
```

（1）” pins” 表示 xMII 使用的 GPIO 管脚；

（2）” function” pinctrl 用到的 function 名称；

（3）” muxsel” GPIO 管脚复用，需查看 Spec 来设定；

（4）”drive” GPIO 管脚驱动能力；

（5）”pull” 输出电平状态。

注：不同平台的 pin 配置不一样。

linux-5.4 中该配置的路径为 arch/arm64（32 位平台为 arm）/boot/dts/sunxi/xxxx.dtsi，具体如下所示：

```
gmac_pins_a: gmac@0 {
    pins = "PA0", "PA1", "PA2", "PA3",
           "PA4", "PA5", "PA6", "PA7",
           "PA8", "PA10", "PA11", "PA12",
           "PA13", "PA17", "PA18", "PA28",
           "PA29", "PA30", "PA31";
    function = "gmac0";
    drive-strength = <10>;
};

gmac_pins_b: gmac@1 {
    pins = "PA0", "PA1", "PA2", "PA3",
           "PA4", "PA5", "PA6", "PA7",
           "PA8", "PA10", "PA11", "PA12",
           "PA13", "PA17", "PA18", "PA28",
           "PA29", "PA30", "PA31";
    function = "gpio_in";
    drive-strength = <10>;
};
```

（1）”pins” 表示 xMII 使用的 GPIO 管脚；

（2）”function” pinctrl 用到的 function 名称；

（3）”drive-strength” GPIO 管脚的驱动能力，具体查看 GPIO 文档。

另外 clk_gmac0, clk_ephy_25m 为时钟的配置。

在 linux-4.9 中，路径为 arch/arm64（32 位平台为 arm）/boot/dts/sunxi/XXXX-clk.dtsi，具体配置如下所示：

```
clk_gmac0_25m: gmac0_25m {
    #clock-cells = <0>;
    compatible = "allwinner,periph-clock";
    clock-output-names = "gmac0_25m";
};
clk_gmac0: gmac0 {
    #clock-cells = <0>;
    compatible = "allwinner,periph-clock";
    clock-output-names = "gmac0";
};
```

3.2.3.2 T113S3P&T113-S4&T113-S4P 方案

3.2.3.2.1 gmac 配置说明

以 T113S3P 为例，设备树配置如下

```
mdio0: mdio0@4500048 {
   compatible = "allwinner,sunxi-mdio";
   #address-cells = <1>;
   #size-cells = <0>;
   reg = <0x0 0x04500048 0x0 0x8>;
   gmac0_phy0: ethernet-phy@1 {
      reg = <1>;
      max-speed = <100>;  /* Max speed capability */
      reset-gpios = <&pio PE 13 GPIO_ACTIVE_LOW>;
      /* PHY datasheet rst time */
      reset-assert-us = <10000>;
      reset-deassert-us = <15000>;
   };
};

gmac0: gmac0@4500000 {
   compatible = "allwinner,sunxi-gmac";
   reg = <0x0 0x04500000 0x0 0x10000>,
      <0x0 0x03000030 0x0 0x4>;
   interrupts = <GIC_SPI 46 IRQ_TYPE_LEVEL_HIGH>;
   interrupt-names = "gmacirq";
   clocks = <&ccu CLK_BUS_EMAC0>, <&ccu CLK_EMAC0_25M>;
   clock-names = "gmac", "phy25m";
   resets = <&ccu RST_BUS_EMAC0>;
   phy-handle = <&gmac0_phy0>;
   pinctrl-0 = <&gmac0_pins_a>;
   pinctrl-1 = <&gmac0_pins_b>;
   pinctrl-names = "default", "sleep";
   phy-mode = "rmii";
   sunxi,phy-clk-type = <0>;
   tx-delay = <7>;
   rx-delay = <31>;
   status = "okay";
};
```

（1）”compatible” 表征具体的设备, 用于驱动和设备的绑定；

（2）”reg” 设备使用的地址；

（3）”interrupts” 设备使用的中断；

（4）”clocks” 设备使用的时钟；

（5）”pinctrl-0” 设备 active 状态下的 GPIO 配置；

（6）”pinctrl-1” 设备 suspend 状态下的 GPIO 配置；

（7）”phy-mode” GMAC 与 PHY 之间的物理接口，如 MII、RMII、RGMII 等；

（8）”tx-delay” tx 时钟延迟，tx-delay 取值 0-7，一档约 536ps（皮秒）；

（9）”rx-delay ” rx 时钟延迟，rx-delay 取值 0-31，一档约 186ps（皮秒）；

（10）”gmac-powerX” gmac 电源脚，根据实际情况配置；

（11）”status” 是否使能该设备节点；

（13）“phy-handle”phy 器件句柄；

（14）phy 子节点配置：“reg” 表征 phy 地址，“max-speed” 表征 phy 的最大速率，“reset-gpios” 表征 phy 硬件复位的引脚，“reset-assert-us” 硬件复位拉低时间，“reset-deassert-us” 硬件复位拉高时间。

3.2.4 BSP 独立仓库 device tree 配置说明

3.2.4.1 gmac 配置说明

```
mdio0: mdio0@4500048 {
    compatible = "allwinner,sunxi-mdio";
    #address-cells = <1>;
    #size-cells = <0>;
    reg = <0x0 0x01c50048 0x0 0x8>;
    status = "okay";
    phy0: ethernet-phy@0 {
        /* RTL8211F (0x001cc916) */
        reg = <0>;
        max-speed = <1000>; /* 1000M */
        reset-gpios = <&pio PA 17 GPIO_ACTIVE_LOW>;
        /* PHY datasheet rst time */
        reset-assert-us = <10000>;
        reset-deassert-us = <150000>;
    };
};

gmac0: gmac0@01c50000 {
    compatible = "allwinner,sunxi-gmac";
    reg = <0x0 0x01c50000 0x0 0x1000>,
        <0x0 0x01c20164 0x0 0x04>;
    interrupts = <GIC_SPI 85 IRQ_TYPE_LEVEL_HIGH>;
    interrupt-names = "gmacirq";
    clocks = <&ccu CLK_BUS_GMAC>;
    clock-names = "gmac";
    resets = <&ccu RST_BUS_GMAC>;
    phy-handle = <&phy0>;
    phy-mode = "rgmii";
    pinctrl-names = "default";
    pinctrl-0 = <&gmac_pins_a>;
    tx-delay = <7>;
    rx-delay = <0>;
    sunxi,phy-clk-type = <1>;
    gmac-power0-supply = <&reg_aldo1>;
    gmac-power1-supply;
```

```
    gmac-power2-supply;
    gmac-power0-vol = <3000000>;
    gmac-power1-vol;
    gmac-power2-vol;
    status = "okay";
};
```

（1）”compatible” 表征具体的设备, 用于驱动和设备的绑定；

（2）”reg” 设备使用的地址；

（3）”interrupts” 设备使用的中断；

（4）”clocks” 设备使用的时钟；

（5）”pinctrl-0” 设备 active 状态下的 GPIO 配置；

（6）”pinctrl-1” 设备 suspend 状态下的 GPIO 配置；

（7）”phy-mode” GMAC 与 PHY 之间的物理接口，如 MII、RMII、RGMII 等；

（8）”tx-delay” tx 时钟延迟，tx-delay 取值 0-7，一档约 536ps（皮秒）；

（9）”rx-delay ” rx 时钟延迟，rx-delay 取值 0-31，一档约 186ps（皮秒）；

（10）”gmac-powerX” gmac 电源脚，根据实际情况配置；

（11）”status” 是否使能该设备节点；

（13）“phy-handle”phy 器件句柄；

（14）phy 子节点配置：“reg” 表征 phy 地址，“max-speed” 表征 phy 的最大速率，“reset-gpios” 表征 phy 硬件复位的引脚，“reset-assert-us” 硬件复位拉低时间，“reset-deassert-us” 硬件复位拉高时间。

3.2.4.2 emac 配置说明

```
mdio1: mdio1@1c0b080 {
    compatible = "allwinner,sun4i-mdio";
    #address-cells = <1>;
    #size-cells = <0>;
    reg = <0x0 0x01c0b080 0x0 0x14>;
    status = "okay";
    phy1: ethernet-phy@1 {
        reg = <1>;
    };
};

emac0: emac0@1c0b000 {
    compatible = "allwinner,sunxi-emac";
    reg = <0x0 0x01c0b000 0x0 0x0c000>;
    interrupts     = <GIC_SPI 55 IRQ_TYPE_LEVEL_HIGH>;
    interrupt-names = "emacirq";
```

```
    clocks  = <&ccu CLK_BUS_EMAC>;
    clock-names = "emac";
    resets = <&ccu RST_BUS_EMAC>;
    pinctrl-names = "default";
    pinctrl-0   = <&emac_pins_a>;
    phy-rst = <&pio PH 27 1 1 1 0>;
    status = "okay";
    phy-handle = <&phy1>;
    status = "disabled";
};
```

（1）”compatible” 表征具体的设备, 用于驱动和设备的绑定；

（2）”reg” 设备使用的地址；

（3）”interrupts” 设备使用的中断；

（4）”clocks” 设备使用的时钟；

（5）”pinctrl-0” 设备 active 状态下的 GPIO 配置；

（6）”phy-rst” PHY 复位脚；

（7）“phy-handle”phy 器件句柄；

（8）phy 子节点配置：“reg” 表征 phy 地址。

3.2.5 AW 定制 PHY

AW 部分 SOC 集成了 AC200 和 AC300，而 AC200 和 AC300 内部又集成了 EPHY

3.2.5.1 AC200

ARM 通过 TWI 与 AC200 进行通讯, 把 EPHY 初始化，然后 MAC 通过 MDIO 总线是访问 EPHY，PWM 模块提供一个内部 25M 时钟给 EPHY。

AC200 整体框图如下。

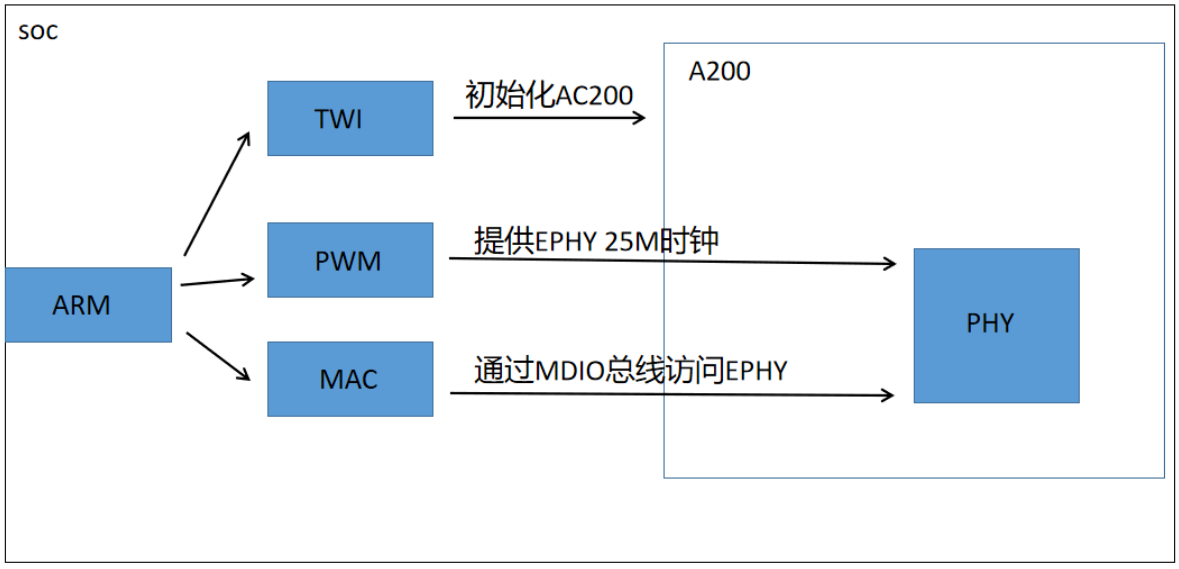

图 3-11: AC200 框图

3.2.5.2 AC300

ARM 通过 MDIO 总线与 AC300 进行通讯，把 EPHY 初始化，然后 MAC 通过 MDIO 总线是访问 EPHY，PWM 模块提供一个内部 25M 时钟给 EPHY。

AC300 整体框图如下。

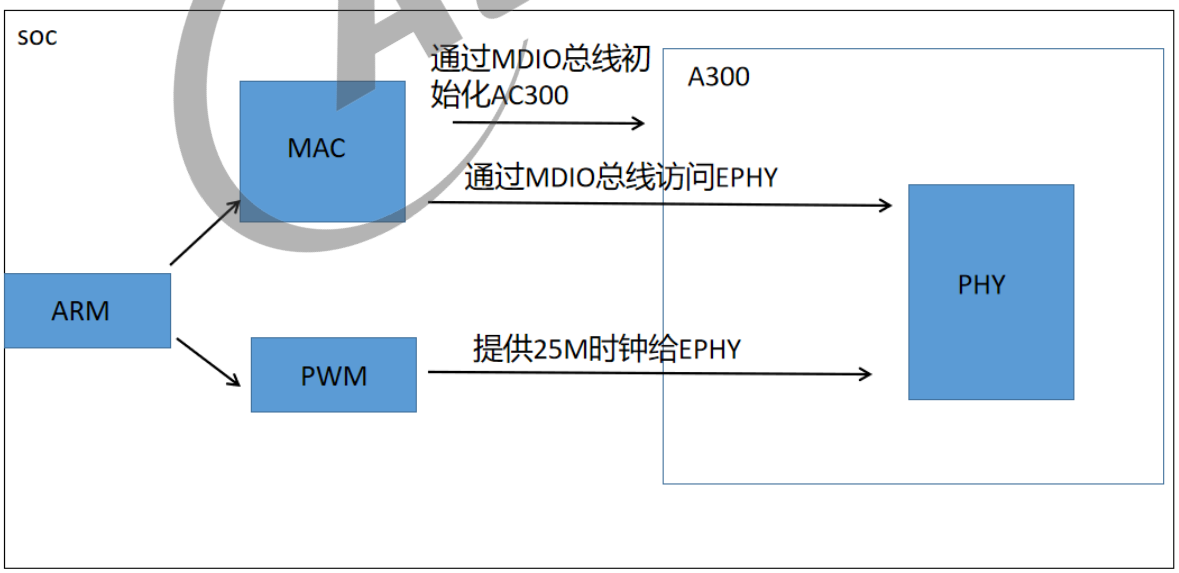

图 3-12: AC300 框图

3.2.6 board.dts 配置说明

3.2.6.1 RGMII 接口配置

对于 RGMII 接口，外挂 RTL8211F PHY 的 GMAC，使用 SOC 内部 EPHY_25M 时钟，支持 10Mbps/100Mbps/1000Mbps 速率。

board.dts 配置范例如下：

路径：longan/device/config/chips/{IC}/configs/{BOARD}/board.dts

```
gmac0: eth@05020000{
    phy-mode = "rgmii";
    use_ephy25m = <1>;
    tx-delay = <7>;
    rx-delay = <0>;
    status = "okay";
};
```

说明

use_ephy25m=1，代表 PHY 使用 SOC 内部 EPHY_25M 时钟；
use_ephy25m=0 或者不配置该参数，代表 PHY 不使用 SOC 内部 EPHY_25M 时钟，需外挂 25M 晶振为 PHY 提供时钟；
RGMII 接口对时钟和数据波形的相位要求比较严格，因此通常需要调整 tx-delay 和 rx-delay 参数保证数据传输的正确性。

3.2.6.2 RMII 接口配置

对于 RMII 接口，外挂 RTL8201F PHY 的 GMAC，使用外挂 25M 晶振，支持 10Mbps/100Mbps 速率。

board.dts 配置范例如下：

路径：longan/device/config/chips/{IC}/configs/{BOARD}/board.dts

```
gmac1: eth@05030000 {
    phy-mode = "rmii";
    status = "okay";
};
```

对于使用 SOC 内置 EPHY 的 GMAC，25M 时钟由 PWM 模块提供，支持 10Mbps/100Mbps 速率。

路径：longan/device/config/chips/{IC}/configs/{BOARD}/board.dts

```
gmac1: eth@05030000 {
    phy-mode = "rmii";
    status = "okay";
};

ac200: ac200 {
    tv_used = <1>;
    tv_twi_used = <1>;
    tv_twi_addr = <16>;
```

```
    tv_pwm_ch = <5>;
    status = "okay";
};
```

说明

有些 SOC 内部 AC200 封装了 EPHY，通过 TWI 与 AC200 进行通讯，把 EPHY 初始化，然后 MAC 通过 MDIO 总线是访问 EPHY。
有些 SOC 内部 AC300 封装了 EPHY，通过 MAC 控制器 MIDO 总线与 AC300 进行通信，把 EPHY 初始化，然后 MAC 同样利用 MDIO 总线去访问 EPHY

当然，关于设备树的配置，可以放在内核的设备树配置，或者是 board.dts。只不过 board.dts 的配置会覆盖内核的设备树配置。

3.3 GMAC 源码结构

3.3.1 非 T113S3P&T113-S4&T113-S4P 方案

GMAC 驱动的源代码位于内核 drivers/net/ethernet/allwinner 目录下:

```
drivers/net/ethernet/allwinner/

├──── sunxi-gmac.h // Sunxi平台GMAC驱动头文件，里面定义了一些宏、数据结构及内部接口
├──── sunxi-gmac.c // Sunxi平台GMAC驱动核心代码
├──── sunxi_gmac_ops.c // Sunxi平台GMAC驱动各个内部接口具体实现drivers/net/ethernet/allwinner/
```

3.3.2 T113S3P&T113-S4&T113-S4P 方案

GMAC 驱动的源代码位于内核 drivers/net/ethernet/allwinner/sunxi-gmac-ng 目录下:

```
├──── Kconfig
├──── Makefile
├──── sunxi-gmac.c    //GMAC驱动代码
├──── sunxi-gmac-trace.h  //GMAC的trace代码
└──── sunxi-mdio.c    //GMAC驱动对于的MDIO驱动代码
```

3.4 BSP 独立仓库源码结构

GMAC 驱动源码位于bsp/drivers/gmac目录下

```
├──── Kconfig
├──── Makefile
├──── sunxi-gmac.c    //GMAC驱动代码
└──── sunxi-mdio.c    //GMAC对应MDIO驱动代码
```

EMAC 驱动源码位于bsp/drivers/gmac目录下

```
├── Kconfig
├── Makefile
├── sun4i-emac.c   //EMAC驱动代码
└── sun4i-mdio.c   //EMAC对应MDIO驱动代码
```

4 以太网特性使用

如下特性目前仅在drivers/net/ethernet/allwinner/sunxi-gmac-ng代码中支持

4.1 裸数据传输

原理：

通过应用 ioctl 的方式，封装了一套以太网协议，达成双方传输裸数据的目的

前提：

（1）收发双方必须都是 AW 支持裸数据传输的平台，比如两台 T113-S3P

（2）由于使用的非标准的协议，故网口必须打开混杂模式（ifconfig eth0 promisc），接收所有以太网报文

4.1.1 menuconfig 配置说明

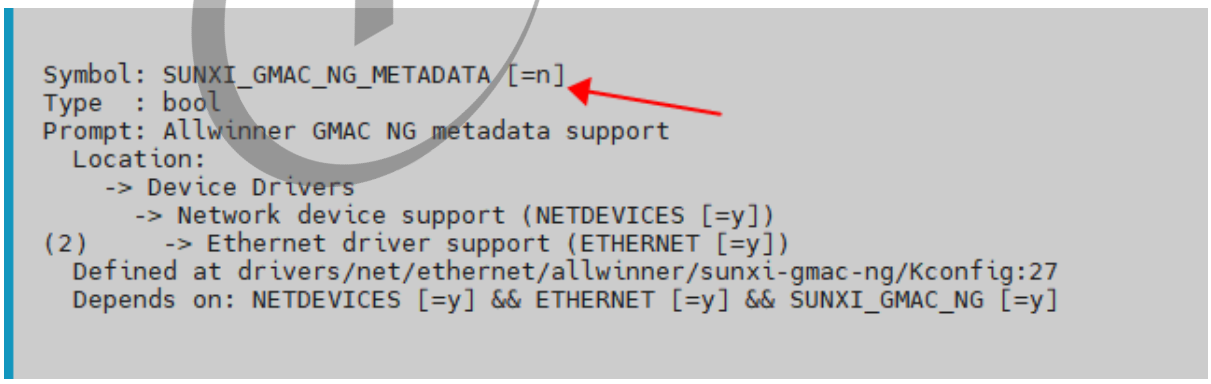

图 4-1: 裸数据传输内核配置

4.1.2 demo 示例

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdbool.h>
#include <getopt.h>
#include <errno.h>
#include <unistd.h>
#include <string.h>

#define GMAC_WRITE     _IOWR('X', 1, unsigned int)
#define GMAC_READ      _IOWR('X', 2, unsigned int)

static const char *device = "/dev/gmac";

bool read_flag = false, write_flag = false;

uint8_t buf_tx[100] = "gmac metadata test";
uint8_t *buf_rx;

bool gmac_read_flag = false, gmac_write_flag = false;

static int gmac_read(int fd)
{
   int ret;

   ret = ioctl(fd, GMAC_READ, buf_rx);
   if (ret < 0)  {
      printf("ioctl failed\n");
      return ret;
   }

   return memcmp(buf_rx, buf_tx, 18);
}

static int gmac_write(int fd)
{
   int ret;

   ret = ioctl(fd, GMAC_WRITE, buf_tx);
   if (ret < 0) {
      printf("ioctl failed\n");
      return ret;
   }

   return 0;
}

static void print_usage(const char *prog)
{
   printf("Usage: %s  [-rw]\n ", prog);
   puts(" -r --read    receive net packet\n"
      " -w --write   write net packet\n");
   exit(1);
```

```
}

static void parse_opts(int argc, char *argv[])
{
  static const struct option lopts[] = {
    { "read", 0, 0, 'r' },
    { "write", 0, 0, 'w' },
    { NULL, 0, 0, 0 },
  };
  int c;

  while (1) {
    c = getopt_long(argc, argv, "rw",
        lopts, NULL);

    if (c == -1)
      break;

    switch (c) {
    case 'r':
      gmac_read_flag = true;
      break;
    case 'w':
      gmac_write_flag = true;
      break;
    default:
      print_usage(argv[0]);
    }

  }

}

int main(int argc, char *argv[])
{
  const unsigned int mtu = 1500;
  int fd;

  parse_opts(argc, argv);

  fd = open(device, O_RDWR);
  if (fd < 0) {
    printf("can't open device: %s", device);
    exit(1);
  }

  buf_rx = malloc(mtu);
  if (!buf_rx) {
    printf("malloc rx buf failed\n");
    exit(1);
  }

  if (gmac_write_flag)
    gmac_write(fd);

  if (gmac_read_flag) {
    if (gmac_read(fd)) {
      printf("gmac read failed\n");
      free(buf_rx);
      close(fd);
```

```
            exit(1);
        }
    }

    free(buf_rx);

    close(fd);

    return 0;
}
```

该 demo 的使用方法：

（1）接收方：./gmac-metadata -r

（2）发送方：./gmac-metadata -w

（3）接收方未打印 “gmac read failed” 即为成功

4.2 jumbo 帧测试方法

前提：

准备两台均支持 jumbo 帧的板卡直连对测

步骤：

以千兆网卡为例

（1）板卡 A：ifconfig eth0 mtu 8100，板卡 B：ifconfig eth0 mtu 8100（设置 eth0 的 mtu 大小为 8100）

（2）板卡 A：ifconfig eth0 192.168.200.10，板卡 B：ifconfig eth0 192.168.200.9

（3）板卡 A：ping 192.168.200.9 -s 8000（传输 8000 长度的数据）

注：

（1）百兆网卡不存在带宽不足情况，一般不需要使用 jumbo 帧

（2）百兆网卡支持的 jumbo 帧范围为 1500 至 4000，千兆网卡支持的 jumbo 帧范围为 1500 至 8100

（3）测试时两端 mtu 无需设置成相同大小

（4）ping 通即表明测试通过

（5）ping 传输 8000 长度的原因：mtu 为整个以太网帧的长度，ping 工具指定的长度 8000 为纯数据长度，不包括 ICMP 及以太网报头的长度

5 以太网常用调试手段

5.1 以太网常用调试命令

（1）查看网络设备信息

```
查看网口状态：ifconfig eth0
查看收发包统计：cat /proc/net/dev
查看当前速率：cat /sys/class/net/eth0/speed
```

（2）打开/关闭网络设备

```
打开网络设备：ifconfig eth0 up
关闭网络设备：ifconfig eth0 down
```

（3）配置网络设备

```
配置静态IP地址：ifconfig eth0 192.168.1.100
配置MAC地址：ifconfig eth0 hw ether 00:11:22:aa:bb:cc
动态获取IP地址：udhcpc -i eth0
PHY强制模式：ethtool -s eth0 speed 100 duplex full autoneg on（设置100Mbps速率、全双工、开启自协商）
```

（4）常用测试命令

```
测试设备连通性：ping 192.168.1.100

TCP吞吐测试：
Server端：iperf -s -i 1
Client端：iperf -c 192.168.1.100 -i 1 -t 60 -P 4

UDP吞吐测试：
Server端：iperf -s -u -i 1
Client端：iperf -c 192.168.1.100 -u -b 100M -i 1 -t 60 -P 4
```

5.2 以太网通用排查手段

5.2.1 常用软件排查手段

（1）检查 phy mode 配置是否正确，如 rgmii、rmii 等；

（2）检查 clk 配置是否正确，如 gmac clk、ephy_25m clk；

（3）检查 GPIO 配置是否正确，如 IO 复用功能、驱动能力等；

（4）检查 phy reset 配置是否正确；

（5）通过 cat /proc/net/dev 命令查看 eth0 收发包统计情况。

5.2.2 常用硬件排查手段

（1）检查 phy 供电（vcc-ephy）是否正常；

（2）检查 phy 时钟波形是否正常。

5.3 以太网常见问题排查流程

5.3.1 ifconfig 命令无 eth0 节点

问题现象：

执行 ifconfig eth0 无相关 log 信息

问题分析：

以太网模块配置未打开或存在 GPIO 冲突

排查步骤：

（1）抓取内核启动 log，检查 gmac 驱动 probe 是否成功；

（2）如果无 gmac 相关打印，请参考 3.2 节确认以太网基本配置是否打开；

（3）如果 gmac 驱动 probe 失败，请参考 4.2.1 节并结合 log 定位具体原因，常见原因是 GPIO 冲突导致。

5.3.2 ifconfig eth0 up 失败

问题现象：

执行 ifconfig eth0 up，出现 “Initialize hardware error” 或 “No phy found” 异常 log

问题分析：

常见原因是供给 phy 使用的 25M 时钟异常

排查步骤：

（1）检查软件 phy_mode 配置与板级情况一致；

（2）检查 phy 供电是否正常；

（3）若步骤 1 和步骤 2 正常，需重点检查 phy 使用的 25M 时钟（ephy25M 或外部晶振）是否正常。

5.3.3 网络不通或网络丢包严

问题现象：

ping 不通对端设备、无法动态获取 ip 地址或有丢包现象

问题分析：

一般原因是 tx/rx 通路不通

排查步骤：

（1）检查 ifconfig eth0 up 是否正常；

（2）检查 eth0 能否动态获取 ip 地址；

（3）若步骤 1 正常，但步骤 2 异常，需首先确认 tx/rx 哪条通路不通；

（4）若无法动态获取 ip 地址，可配置静态 IP，和对端设备互相 ping；

（5）检查对端设备能否收到数据包，若能收到，则说明 tx 通路正常，否则 tx 通路异常；

（6）检查本地设备能否收到数据包，若能收到，则说明 rx 通路正常，否则 rx 通路异常；

（7）若 tx 通路异常，可调整 tx-delay 参数或对照原理图检查 tx 通路是否异常，如漏焊关键器件；

（8）若 rx 通路异常，可调整 rx-delay 参数或对照原理图检查 rx 通路是否异常，如漏焊关键器件；

（9）若经过上述排查步骤问题仍未解决，需检查 phy 供电与 GPIO 耐压是否匹配。

5.3.4 吞吐率异常

问题现象：

千兆网络吞吐率偏低，如小于 300Mbps

排查步骤：

（1）检查内核有无开启 CONFIG_SLUB_DEBUG_ON 宏，若有，则关闭此宏后再进行测试；

（2）如问题仍没有解决，请检查网络是否有丢包、错包现象，若有，参考 4.3.3 进行排查。

6 以太网常用调试工具

6.1 ifconfig

使用ifconfig -h查看支持的参数，如下所示

```
/ # ifconfig -h
Usage:
  ifconfig [-a] [-v] [-s] <interface> [[<AF>] <address>]
  [add <address>[/<prefixlen>]]
  [del <address>[/<prefixlen>]]
  [[-]broadcast [<address>]]  [[-]pointopoint [<address>]]
  [netmask <address>]  [dstaddr <address>]  [tunnel <address>]
  [outfill <NN>] [keepalive <NN>]
  [hw <HW> <address>]  [mtu <NN>]
  [[-]trailers]  [[-]arp]  [[-]allmulti]
  [multicast]  [[-]promisc]
  [mem_start <NN>]  [io_addr <NN>]  [irq <NN>]  [media <type>]
  [txqueuelen <NN>]
  [[-]dynamic]
  [up|down] ...

  <HW>=Hardware Type.
  List of possible hardware types:
    loop (Local Loopback) slip (Serial Line IP) cslip (VJ Serial Line IP)
    slip6 (6-bit Serial Line IP) cslip6 (VJ 6-bit Serial Line IP) adaptive (Adaptive Serial Line IP)
    ash (Ash) ether (Ethernet) ax25 (AMPR AX.25)
    netrom (AMPR NET/ROM) rose (AMPR ROSE) tunnel (IPIP Tunnel)
    ppp (Point-to-Point Protocol) hdlc ((Cisco)-HDLC) lapb (LAPB)
    arcnet (ARCnet) dlci (Frame Relay DLCI) frad (Frame Relay Access Device)
    sit (IPv6-in-IPv4) fddi (Fiber Distributed Data Interface) hippi (HIPPI)
    irda (IrLAP) x25 (generic X.25) infiniband (InfiniBand)
    eui64 (Generic EUI-64)
  <AF>=Address family. Default: inet
  List of possible address families:
    unix (UNIX Domain) inet (DARPA Internet) inet6 (IPv6)
    ax25 (AMPR AX.25) netrom (AMPR NET/ROM) rose (AMPR ROSE)
    ipx (Novell IPX) ddp (Appletalk DDP) ash (Ash)
    x25 (CCITT X.25)
```

- -a：显示全部接口信息
- -s：显示摘要信息

```
/ # ifconfig -s
Iface      MTU    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0      1500       18      0      0 0           19      0      0      0 BMRU
lo       65536        0      0      0 0            0      0      0      0 LRU
/ # netstat -i
Kernel Interface table
Iface      MTU    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0      1500       18      0      0 0           19      0      0      0 BMRU
lo       65536        0      0      0 0            0      0      0      0 LRU
```

图 6-1: 显示摘要信息

- add/del：添加对应网口的 ipv6 地址

```
/ # ifconfig eth0 inet6 add 2001:250:250:250:250:250:250:222/64
/ # ifconfig -a
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 2001:250:250:250:250:250:250:222  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::8809:46ff:fe7f:229  prefixlen 64  scopeid 0x20<link>
        ether 8a:09:46:7f:02:29  txqueuelen 1000  (Ethernet)
        RX packets 30  bytes 2348 (2.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 13  bytes 1102 (1.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 60

eth1: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether 4a:e1:76:1e:98:83  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
```

图 6-2: 添加/删除 ipv6 地址

- netmask：设置子网掩码

```
/ # ifconfig eth0 netmask 255.255.255.254
/ # ifconfig -a
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.200.10  netmask 255.255.255.254  broadcast 192.168.200.255
        inet6 fe80::786a:2fff:fe06:915e  prefixlen 64  scopeid 0x20<link>
        ether 7a:6a:2f:06:91:5e  txqueuelen 1000  (Ethernet)
        RX packets 27  bytes 1894 (1.8 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 22  bytes 1732 (1.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 60
```

图 6-3: 设置子网掩码

- hw：设置 mac 地址

```
/ # ifconfig eth0 hw ether 12:34:56:78:90:00
/ # ifconfig -a
eth0: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether 12:34:56:78:90:00  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 60
```

图 6-4: 设置 mac 地址

- arp：打开或者关闭对应网口是否支持 arp 功能，如下图所示，关闭 arp 协议，png 不通，打开之后就可以 ping 通

```
/ #
/ # ifconfig eth0 -arp
/ # ifconfig eth0 192.168.200.10
[   65.624942] [sound  395][MACH simple_parse_of] simple_dai_link_of failed
[   65.788422] sunxi-gmac 5020000.gmac0 eth0: eth0: Type(8) PHY ID 001cc916 at 1 IRQ poll (5020048.mdio0-mii:01)
[   65.801404] sunxi-gmac 5020000.gmac0 eth0: Link is Up - 1Gbps/Full - flow control off
/ #
/ # ping 192.168.200.9
PING 192.168.200.9 (192.168.200.9): 56 data bytes
^C
--- 192.168.200.9 ping statistics ---
2 packets transmitted, 0 packets received, 100% packet loss
/ # ifconfig eth0 arp
/ # ping 192.168.200.9
PING 192.168.200.9 (192.168.200.9): 56 data bytes
64 bytes from 192.168.200.9: seq=0 ttl=64 time=0.867 ms
64 bytes from 192.168.200.9: seq=1 ttl=64 time=0.869 ms
^C
--- 192.168.200.9 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.867/0.868/0.869 ms
```

图 6-5: 打开/关闭 arp 功能

6.2 route

设置和查看路由表都可以用 route 命令，设置内核路由表的命令格式是：

```
# route  [add|del] [-net|-host] target [netmask Nm] [gw Gw] [[dev] If]
```

其中：

- add : 添加一条路由规则
- del : 删除一条路由规则
- -net : 目的地址是一个网络
- -host : 目的地址是一个主机
- target : 目的网络或主机
- netmask : 目的地址的网络掩码
- gw : 路由数据包通过的网关
- dev : 为路由指定的网络接口

示例：

```
/ # route add -host 192.168.200.9 dev eth0
/ # route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.200.0   0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.200.9   0.0.0.0         255.255.255.255 UH    0      0        0 eth0
/ # route add -host 192.168.200.9 gw 192.168.200.1
/ # route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.200.0   0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.200.9   192.168.200.1   255.255.255.255 UGH   0      0        0 eth0
192.168.200.9   0.0.0.0         255.255.255.255 UH    0      0        0 eth0
```

图 6-6: 添加路由表

```
route add -host 192.168.200.9 dev eth0 #添加eth0的目的地址为192.168.200.9
route add -host 192.168.200.9 gw 192.168.200.1 #添加到目的地址192.168.200.9经过网关192.168.200.1
```

6.3 mii_reg

mii_reg 工具是 GMAC 驱动中提供的调试节点, 其主要作用是对外部 PHY 寄存器地址进行读写操作

启动网卡

```
ifconfig eth0 up
```

进入操作目录

```
cd sys/devices/platform/soc@3000000/450000.eth
```

注: 不同的目录可能会因板卡不同有差异, 此处为大致路径

读取对应的 phy 寄存器

```
addr: PHY地址
reg: PHY寄存器
val: 数据
```

写操作:

```
echo addr reg val > mii_write; cat mii_write
eg:
echo 0x00 0x1f 0xa43 > mii_write; cat mii_write
```

读操作:

```
echo addr reg > mii_read; cat mii_read
eg:
echo 0x10 0x06 > mii_read; cat mii_read
```

一次读多位寄存器操作: 0x01~0x0a

```
cat mii_reg
```

著作权声明

商标声明

免责声明