

地平线 PHY 调试手册

Rev 0.1

2020 年 11 月 10 日

Horizon Robotics
版权所有 禁止转载

免责声明

本文档信息仅用于帮助系统和软件使用人员使用地平线产品。本文档信息未以明示或暗示方式授权他人基于本文档信息设计或制造任何集成电路。

本文档中的信息如有更改，恕不另行通知。尽管本文档已尽可能确保内容的准确性，本文档中的所有声明、信息和建议均不构成任何明示或暗示的保证、陈述或担保。

本文档中的所有信息均按“原样”提供。地平线不就其产品在任何特定用途的适销性、适用性以及不侵犯任何第三方知识产权方面做出任何明示或暗示保证、陈述或担保。地平线不承担产品使用所引起的任何责任，包括但不限于直接或间接损失赔偿。

买方和正在基于地平线产品进行开发的其他方（以下统称为“用户”）理解并同意，用户在设计产品应用时应承担独立分析、评估和判断的责任。用户应对其应用（以及用于其应用的所有地平线产品）的安全性承担全部责任，并保证符合所有适用法规、法律和其它规定的要求。地平线产品简介和产品规格中提供的“典型”参数在不同应用下可能会不同，实际性能也可能随时间而变化。所有工作参数，包括“典型”参数，都必须由用户自己针对每项用户应用进行验证。

用户同意如因用户未经授权使用地平线产品或因不遵守本说明中的条款，造成任何索赔、损害、成本、损失和（或）责任，用户将为地平线及其代表提供全额赔偿。

© 2018 版权所有

北京地平线信息技术有限公司

<https://www.horizon.ai>

修订记录

修订记录列出了各文档版本间发生的主要更改。下表列出了每次文档更新的技术内容。

版本	作者	发布日期	变更说明
V0.1	朱宪坤	2020/11/10	新建

目录

免责声明.....	2
修订记录.....	3
1. PHY 与 MAC	5
1.1. 什么是 PHY.....	5
1.2. 什么是 MAC.....	5
1.3. PHY 与 MAC 的关系.....	6
2. uboot 下 PHY 的配置.....	7
2.1. 检查 MAC 是否使能.....	8
2.1.1. 检查 MAC config 是否配置.....	8
2.1.2. 检查 Kconfig 中是否有配置.....	8
2.1.3. 检查 Makefile 中是否加入源文件编译.....	8
2.2. PHY 相关配置与编译.....	8
2.2.1. PHY driver	8
2.2.2. 使能 phy config.....	11
2.2.3. Kconfig 中配置 config	11
2.2.4. Makefile 中加入 PHY driver 编译.....	12
2.2.5. 设备树中配置 phy addr.....	12
2.3. uboot 下打开 mii cmd	13
2.3.1. 通过 MDIO 方式访问 PHY 内部寄存器	13
3. kernel 下 phy 调试.....	18
3.1. 检查 MAC 是否使能.....	19
3.1.1. 检查 MAC config 是否配置.....	19
3.1.2. 检查 Kconfig 中是否有配置.....	19
3.1.3. 检查 Makefile 中是否加入源文件编译.....	20
3.2. PHY 相关配置与编译.....	20
3.2.1. PHY driver	20
3.2.2. 使能 PHY config.....	21
3.2.3. Kconfig 中配置 config	22
3.2.4. Makefile 中加入 PHY driver 源文件编译	22
3.2.5. 设备树中配置 PHY 设备节点.....	23
4. 常见问题汇总.....	25
4.1. 硬件接线错误.....	25
4.1.1. MDIO 与 MDC 接反.....	25
4.2. Horizon MAC 不支持延时.....	25

1. PHY 与 MAC

1.1. 什么是 PHY

PHY(Physical Layer, PHY)是 IEEE802.3 中定义的一个标准模块, STA(Station Management entity, 管理实体, 一般为 MAC 或 CPU)通过 SMI(Serial Manage Interface)对 PHY 的行为、状态进行管理和控制, 而具体管理和控制动作是通过读写 PHY 内部寄存器实现的。

PHY 是物理接口收发器, 它实现 OSI 模型的物理层。IEEE-802.3 标准定义了以太网 PHY。包括 MII/GMII(介质独立接口)子层、PCS(物理编码子层)、PMA(物理介质附加)子层、PMD(物理介质相关)子层、MDI 子层。它符合 IEEE-802.3k 中用于 10BaseT(第 14 条)和 100BaseTX(第 24 条和第 25 条)的规范。

一个 PHY 的基本结构如下:

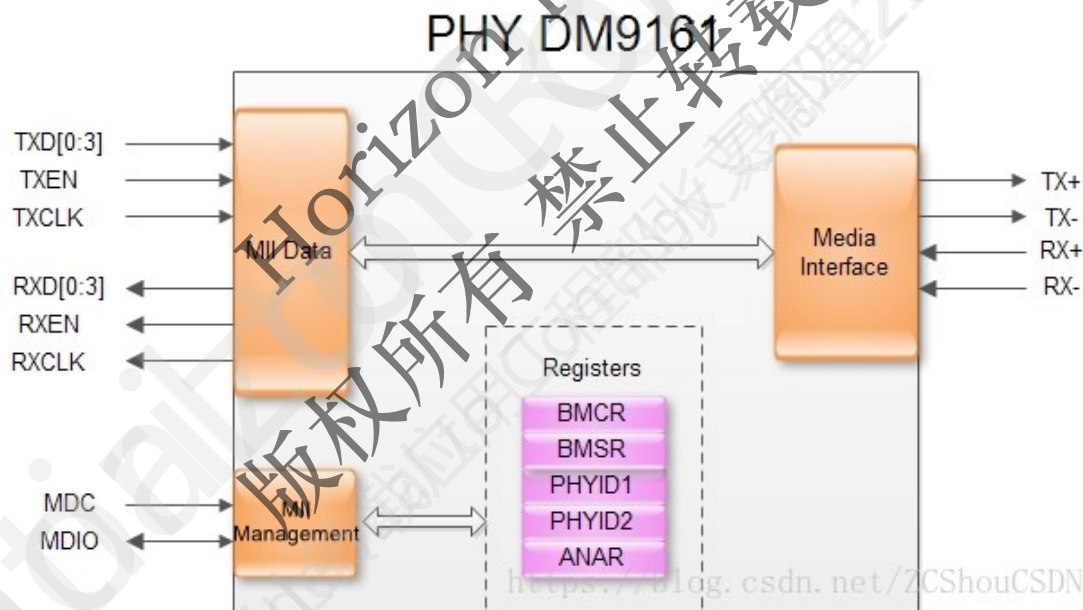


图 1-1 PHY 基本结构

1.2. 什么是 MAC

MAC (Media Access Control) 即媒体访问控制子层协议。该部分有两个概念: MAC 可以是一个硬件控制器及 MAC 通信以协议。该协议位于 OSI 七层协议中数据链路层的下半部分, 主要负责控制与连接物理层的物理介质。MAC 硬件大约就是下面的样子了:

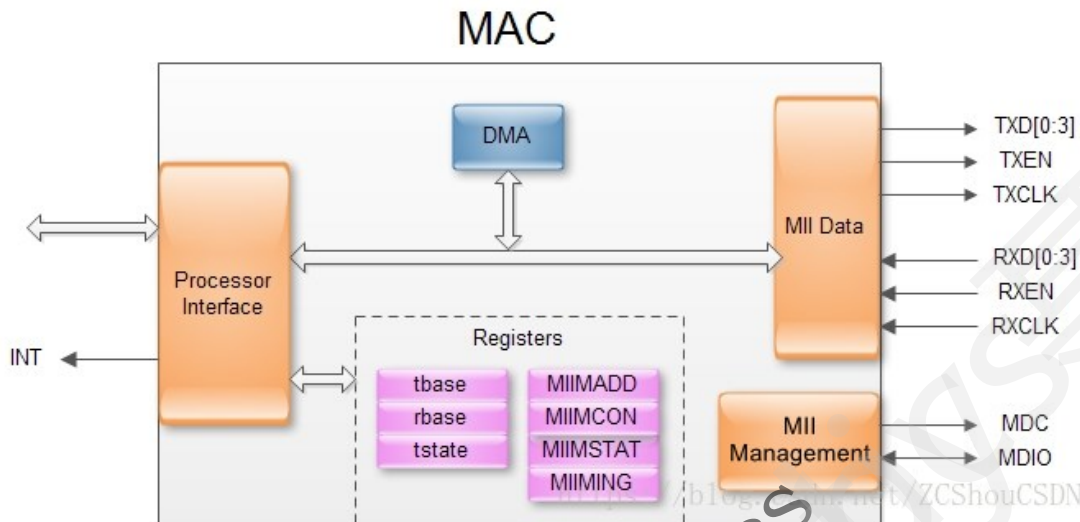


图 1-2 MAC 基本结构

在发送数据的时候，MAC 协议可以事先判断是否可以发送数据，如果可以发送将给数据加上一些控制信息，最终将数据以及控制信息以规定的格式发送到物理层；在接收数据的时候，MAC 协议首先判断输入的信息并是否发生传输错误，如果没有错误，则去掉控制信息发送至 LLC(逻辑链路控制)层。该层协议是以太网 MAC 由 IEEE-802.3 以太网标准定义。

1.3. PHY 与 MAC 的关系

从硬件的角度看，以太网接口电路主要由 MAC (Media Access Control) 控制器和物理层接口 PHY (Physical Layer, PHY) 两大部分构成。MAC 和 PHY 之间通过 IEEE 定义的标准的 MII/GMII(Media Independent Interface, 介质独立界面)通信。这个界面是 IEEE 定义的 MII 界面传递了网络的所有数据和数据的控制。ETHERNET 的接口实质是 MAC 通过 MII 总线控制 PHY 的过程。

2. uboot 下 PHY 的配置

uboot 下的 PHY 配置比较简单，主要完成以下几项内容：

1. 检查 MAC 配置是否打开
2. 检查 PHY driver 是否存在
3. 配置 PHY 对应的 config 开关
4. 添加 PHY 的设备树节点

uboot 下 PHY 涉及到的相关文件如下(本文 PHY 以 Realtek 公司的 RTL8821F 为例，其他 PHY 的使用与之类似)：

```
1. uboot
2.   |— arch
3.   |   |— arm
4.   |   |   |— dts
5.   |   |   |   |— hobot-x2-soc.dts
6.   |— drivers
7.   |   |— net
8.   |   |   |— phy
9.   |   |   |   |— Kconfig
10.  |   |   |   |— Makefile
11.  |   |   |   |— phy.c
12.  |   |   |   |   |— realtek.c
13.  |   |   |   |   |   |— x2_eth_gmac.c
14.  |— include
15.  |   |— configs
16.  |   |   |— x2.h
```

- 1) 其中 x2_eth_gmac.c 是 Horizon MAC driver 文件；
- 2) phy.c 是通用的 PHY 接口文件；
- 3) realtek.c 是 phy driver 文件(需要根据具体的硬件环境来确认 phy driver 源文件)；
- 4) hobot-x2-soc.dts 是设备树文件；
- 5) x2.h 是配置 PHY 使能的文件；
- 6) Makefile 和 Kconfig 用来选择编译 MAC & PHY driver。

2.1. 检查 MAC 是否使能

2.1.1. 检查 MAC config 是否配置

使能 MAC 的配置是在 uboot/configs/hr_x2_defconfig(需要根据具体使用的 deconfig 文件来配置)中, 需要确保如下配置正常使能:

```
CONFIG_X2_ETH_GMAC=y
```

2.1.2. 检查 Kconfig 中是否有配置

在 uboot/drivers/net/Kconfig 中检查是否配置 X2_ETH_GMAC:

```
93 config X2_ETH_GMAC
94     bool "Hobot X2 Ethernet device support"
95     depends on DM_ETH
96     select PHYLIB
97     help
98     This driver supports the X2 Ethernet IP block.
99
```

图 2-1 MAC Kconfig 配置

2.1.3. 检查 Makefile 中是否加入源文件编译

检查 uboot/drivers/net/Makefile 中, 是否将 MAC 文件的编译加入进去:

```
75 obj-$(CONFIG_PHYLIB) += sh1_ave.o
76 obj-$(CONFIG_X2_ETH_GMAC) += x2_eth_gmac.o
```

图 2-2 MAC 源码编译

如果编译时候, 发现 MAC 源文件没有编译进去, 需要根据 Kconfig 来检查 X2_ETH_GMAC 依赖的 config 配置是否都打开了。

2.2. PHY 相关配置与编译

2.2.1. PHY driver

uboot 下 PHY driver 路径为 uboot/drivers/net/phy。在 PHY debug 时候, 需要确认该目录

下是否存在对应的 PHY driver。例如，realtek PHY 驱动文件为 realtek.c。如果 PHY driver 文件不存在，需要移植 PHY driver 或者实现 PHY driver。

2.2.1.1. 检查 PHY driver 中是否存在对应的 PHY UID

由于 uboot 下 PHY driver 的适配是根据 PHY UID 和 mask 值来确认的，因此在检查存在 PHY driver 文件后，需要根据 debug 的实际需求来确认 PHY driver 文件中是否存在硬件对应 PHY UID。

例如 RTL8821F PHY 的 UID 是 0x001cc916, 这个可以在 PHY datasheet 中的 **PHYID1** 和 **PHYID2** 寄存器查到，如下：

8.4.3. PHYID1 (PHY Identifier Register 1, Address 0x02)

Table 24. PHYID1 (PHY Identifier Register 1, Address 0x02)

Bit	Name	Type	Default	Description
2.15:0	OUI_MSB	RO	000000000001100	Organizationally Unique Identifier Bit 3:18. Always 000000000001100.

Note: Realtek OUI is 0x000732.

8.4.4. PHYID2 (PHY Identifier Register 2, Address 0x03)

Table 25. PHYID2 (PHY Identifier Register 2, Address 0x03)

Bit	Name	Type	Default	Description
3.15:10	OUI_LSB	RO	110010	Organizationally Unique Identifier Bit 19:24. Always 110010.
3.9:4	Model Number	RO	010001	Manufacture's Model Number
3.3:0	Revision Number	RO	0110	Revision Number

图 2-3 PHY UID 寄存器

因此，在确认过 PHY UID 后，需要在 PHY driver 文件中检查是否存在对应的 PHY driver 或者 PHY driver 有没有加入到 init 函数中。

```
1.  /* Support for RTL8211B PHY */
2.  static struct phy_driver RTL8211B_driver = {
3.      .name = "RealTek RTL8211B",
4.      .uid = 0x1cc912,
5.      .mask = 0xffffffff,
6.      .features = PHY_GBIT_FEATURES,
7.      .probe = &rtl8211b_probe,
8.      .config = &rtl8211x_config,
9.      .startup = &rtl8211x_startup,
10.     .shutdown = &genphy_shutdown,
11. };
12.
```

```
13. /* Support for RTL8211E-VB-CG, RTL8211E-VL-CG and RTL8211EG-VB-CG PHYs */
14. static struct phy_driver RTL8211E_driver = {
15.     .name = "RealTek RTL8211E",
16.     .uid = 0x1cc915,
17.     .mask = 0xffffffff,
18.     .features = PHY_GBIT_FEATURES,
19.     .probe = &rtl8211e_probe,
20.     .config = &rtl8211x_config,
21.     .startup = &rtl8211e_startup,
22.     .shutdown = &genphy_shutdown,
23. };
24.
25. /* Support for RTL8211DN PHY */
26. static struct phy_driver RTL8211DN_driver = {
27.     .name = "RealTek RTL8211DN",
28.     .uid = 0x1cc914,
29.     .mask = 0xffffffff,
30.     .features = PHY_GBIT_FEATURES,
31.     .config = &rtl8211x_config,
32.     .startup = &rtl8211x_startup,
33.     .shutdown = &genphy_shutdown,
34. };
35.
36. /* Support for RTL8211F PHY */
37. static struct phy_driver RTL8211F_driver = {
38.     .name = "RealTek RTL8211F",
39.     .uid = 0x1cc916,
40.     .mask = 0xffffffff,
41.     .features = PHY_GBIT_FEATURES,
42.     .config = &rtl8211f_config,
43.     .startup = &rtl8211f_startup,
44.     .shutdown = &genphy_shutdown,
45. };
46.
47. int phy_realtek_init(void)
48. {
49.     phy_register(&RTL8211B_driver);
50.     phy_register(&RTL8211E_driver);
51.     phy_register(&RTL8211F_driver);
52.     phy_register(&RTL8211DN_driver);
53.
54.     return 0;
55. }
```

如果 phy driver 文件中找不到对应的 phy uid, 就说明缺少该 phy 的 driver, 需要根据 phy datasheet 来实现具体的 driver, 然后在 phy_realtek_init 函数中加入对应的 phy driver。

2.2.2. 使能 phy config

使能 phy config 的配置是在 uboot/include/configs/x2.h 中, 可以在改文件中添加新的 phy config 或者关闭 phy config。

```
89 /*#define CONFIG_PHY_MARVELL*/  
90 #define CONFIG_PHY_REALTEK
```

图 2-4 使能 PHY config

这个 config 的使能, 是在 phy init 的时候生效的, phy init

```
1. int phy_init(void)  
2. {  
3.     #ifdef CONFIG_B53_SWITCH  
4.         phy_b53_init();  
5.     #endif  
6.     #ifdef CONFIG_MV88E61XX_SWITCH  
7.         phy_mv88e61xx_init();  
8.     #endif  
9.     .....  
10.    #ifdef CONFIG_PHY_REALTEK  
11.        phy_realtek_init();  
12.    #endif  
13.    .....  
14.    #ifdef CONFIG_PHY_FIXED  
15.        phy_fixed_init();  
16.    #endif  
17.    return 0;  
18. }
```

2.2.3. Kconfig 中配置 config

在 uboot/drivers/net/phy/Kconfig 中添加 PHY_REALTEK 配置项:

```
146  
147 config PHY_REALTEK  
148     bool "Realtek Ethernet PHYs support"  
149
```

图 2-5 PHY Kconfig 配置

2.2.4. Makefile 中加入 PHY driver 编译

在 uboot/drivers/net/phy/Makefile 中将 PHY driver 的编译添加进去。

1. `obj-$(CONFIG_PHY_REALTEK) += realtek.o`
- 2.

如果编译时候, 发现 PHY driver 源文件没有被编译, 需要检查一下一些依赖项是否没有正确打开。

2.2.5. 设备树中配置 phy addr

uboot 下 PHY 设备树配置比较简单, 只是需要根据实际的硬件连接来配置 PHY address, 然后在 MAC 设备节点下添加 PHY 子设备节点就可以了。

例如, Realtek 的 PHY address 硬件连接地址是 0x3, 如下图:

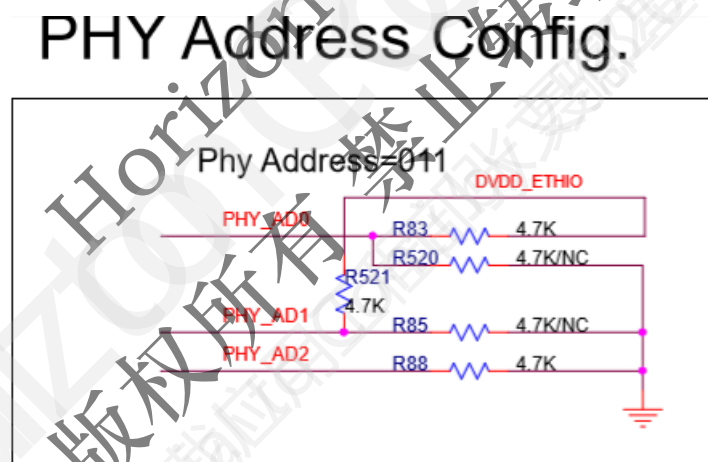


图 2-6 PHY Address 配置

那么 uboot dts 中 PHY 子设备配置如下:

```
1. &gmac {
2.     status = "okay";
3.     #address-cells = <0x1>;
4.     #size-cells = <0x0>;
5.     phyaddr = <3>;
6.
7.     phy0: ethernet-phy@0 {
8.         reg = <0>;
9.     };
10. };
```

2.3. uboot 下打开 mii cmd

在 uboot 下 debug PHY 的时候，可以在 uboot/configs/hr_x2_defconfig 中打开 mii cmd 功能，该命令可以通过 MDIO 方式来读取 PHY 内部寄存器的信息。

```
-- a/configs/hr_x2_defconfig
+++ b/configs/hr_x2_defconfig
@@ -510,7 +510,7 @@ CONFIG_CMD_TFTPPUT=y
CONFIG_CMD_TFTP_VARS=y
# CONFIG_CMD_RARP is not set
CONFIG_CMD_NFS=y
-# CONFIG_CMD_MII is not set
+CONFIG_CMD_MII=y
```

图 2-7 MII CMD 配置

2.3.1. 通过 MDIO 方式访问 PHY 内部寄存器

2.3.1.1. 确认 MDC 的频率

使用 MDIO 方式访问 PHY 寄存器的时候，MDC 的最高频率不能超过 12.5MHz，否则会导致通过 MDIO 方式访问 PHY 寄存器异常。读取 PHY 寄存器值是否正常，可通过 PHY 寄存器 0x2 和寄存器 0x3 来确认，这两个寄存器存储的是 PHY 的 UID，是 PHY 芯片出厂时已经固化好的。如果读出来的 UID 和 PHY datasheet 中的 UID 一致，就说明 MDIO 方式访问 PHY 寄存器正常，否则需要检查一下 MDC 频率是否符合要求。

配置 MDC 频率的位置是在 uboot/drivers/net/x2_eth_gmac.c 中，这个文件是 Horizon 的 MAC driver 文件。

```
1. static int eqos_mdio_write(struct mii_dev *bus, int mdio_addr, int mdio_devad, int mdio_reg, u16 mdio_val)
2. {
3.     struct eqos_priv *eqos = bus->priv;
4.     u32 val;
5.     int ret;
6.     printf("%s():\n", __func__);
7.
8.     debug("%s(dev=%p, addr=%x, reg=%d, val=%x):\n", __func__, eqos->dev,
9.         mdio_addr, mdio_reg, mdio_val);
10.
```

```

11. ret = eqos_mdio_wait_idle(eqos);
12. if (ret) {
13.     pr_err("MDIO not idle at entry");
14.     return ret;
15. }
16.
17. writel(mdio_val, &eqos->mac_regs->mdio_data);
18.
19. val = readl(&eqos->mac_regs->mdio_address);
20. val &= EQOS_MAC_MDIO_ADDRESS_SKAP |
21.     EQOS_MAC_MDIO_ADDRESS_C45E;
22. val |= (mdio_addr << EQOS_MAC_MDIO_ADDRESS_PA_SHIFT) |
23.     (mdio_reg << EQOS_MAC_MDIO_ADDRESS_RDA_SHIFT) |
24.     (EQOS_MAC_MDIO_ADDRESS_CR_35_60 <<
25.     EQOS_MAC_MDIO_ADDRESS_CR_SHIFT) |
26.     (EQOS_MAC_MDIO_ADDRESS_GOC_WRITE <<
27.     EQOS_MAC_MDIO_ADDRESS_GOC_SHIFT) |
28.     EQOS_MAC_MDIO_ADDRESS_GB;
29. writel(val, &eqos->mac_regs->mdio_address);
30.
31. udelay(10);
32.
33. ret = eqos_mdio_wait_idle(eqos);
34. if (ret) {
35.     pr_err("MDIO read didn't complete");
36.     return ret;
37. }
38.
39. return 0;
40. }

```

在 `eqos_mdio_write` 函数和 `eqos_mdio_read` 函数中，`EQOS_MAC_MDIO_ADDRESS_CR_35_60` 这个宏所在的位是用来控制 MDC 频率的(或者根据实际情况选择, 见下面 MDC 频率配置表, 如 `EQOS_MAC_MDIO_ADDRESS_CR_20_35`)，如果 MDIO 方式获取 PHY 寄存器数据异常，请测量 MDC 频率，然后按照实际的需求来配置，具体的配置可以参考如下：

Horizon MAC 通过配置 CSR clock range 调整 MDC 的输出频率：

bit[11-8]	description	bit[11-8]	description
0000	CSR= 60-100MHz MDC clock = CSR clock/42	1000	MDC clock = CSR clock/4

0001	CSR= 100-150MHz MDC clock = CSR clock/62	1001	MDC clock = CSR clock/6
0010	CSR= 20-35MHz MDC clock = CSR clock/16	1010	MDC clock = CSR clock/8
0011	CSR= 35-60MHz MDC clock = CSR clock/26	1011	MDC clock = CSR clock/10
0100	CSR= 150-250MHz MDC clock = CSR clock/102	1100	MDC clock = CSR clock/12
0101	CSR= 250-300MHz MDC clock = CSR clock/124	1101	MDC clock = CSR clock/14
0110	CSR= 300-500MHz MDC clock = CSR clock/204	1110	MDC clock = CSR clock/16
0111	CSR= 500-800MHz MDC clock = CSR clock/324	1111	MDC clock = CSR clock/18

bit[11]=0 的时候可以限制 MDC clock 在 1.0MHz 到 2.5MHz 之间，bit[11]=1 可以提供更高的 MDC clock。

2.3.1.2.MDIO 读取 PHY 寄存器

MDIO 方式可以直接读写 PHY 内部寄存器，其读取 PHY 寄存器的命令格式如下：

```
mii read phyaddr regaddr
```

可以先通过读取 PHY 寄存器 0x2 和 0x3 来确认读取的寄存器结果是否正确，这两个寄存器的值就是 PHY UID，是跟具体的 PHY 硬件型号绑定的。例如 Realtek8821F PHY 的 UID 是 0x001cc916，而读取的 PHY 寄存器 0x2 和 0x3 的值组合起来就是这个值。


```

5945 20200820_14:35:28:248:Hobot>mii read 3 0
5946 20200820_14:35:28:248:1040
5947 20200820_14:35:30:978:Hobot>mii read 3 1
5948 20200820_14:35:30:979:79A9
5949 20200820_14:35:32:924:Hobot>mii read 3 2
5950 20200820_14:35:32:924:001C
5951 20200820_14:35:35:878:Hobot>mii read 3 3
5952 20200820_14:35:35:879:C916
5953 20200820_14:35:37:767:Hobot>mii read 3 4
5954 20200820_14:35:37:768:01E1
5955 20200820_14:35:40:160:Hobot>mii read 3 5
5956 20200820_14:35:40:160:C5E1
5957 20200820_14:35:42:619:Hobot>mii read 3 6
5958 20200820_14:35:42:619:006F
5959 20200820_14:35:45:305:Hobot>mii read 3 7
5960 20200820_14:35:45:305:2001

```

图 2-8 MDIO 方式读取 PHY 内部寄存器

需要注意的是：一些 PHY 的寄存器 bit 需要连着读取两次才能获取到正确的值，使用 mii cmd 的时候，可以连着读几次试验一下，以确保正确性。

Bit	Name	Type	Default	Description
1.2	Link Status	RO	0	Link Status. 1: Linked 0: Not Linked This register indicates whether the link was lost since the last read. For the current link status, either read this register twice or read register bit 17,10 Link Real Time.

图 2-9 某些 PHY 寄存器需要读取两次

2.3.1.3.MDIO 写入 phy 寄存器

MDIO 写入 phy 寄存器的命令格式如下：

```
mii write phyaddr regaddr value
```

通过 MDIO 方式向 phy 寄存器中写入值的时候，需要根据 phy datasheet 来确认寄存器是否可以写入。

2.3.1.4.mii dump 方式查看寄存器的值

mii dump 方式也可以读取 phy 寄存器的值，而且显示的结果更加友好，其命令格式如下：

```
mii dump phyaddr regaddr
```

例如，使用 `mii dump` 方式读取 `phy` 寄存器 `0x1` 的结果如下：

```
20200813_17:48:01:079:Hobot>mii_dump 3 1
20200813_17:48:01:079:1. (7989) -- PHY status register --
20200813_17:48:01:080: (8000:0000) 1.15 = 0 100BASE-T4 able
20200813_17:48:01:080: (4000:4000) 1.14 = 1 100BASE-X full duplex able
20200813_17:48:01:080: (2000:2000) 1.13 = 1 100BASE-X half duplex able
20200813_17:48:01:080: (1000:1000) 1.12 = 1 10 Mbps full duplex able
20200813_17:48:01:080: (0800:0800) 1.11 = 1 10 Mbps half duplex able
20200813_17:48:01:080: (0400:0000) 1.10 = 0 100BASE-T2 full duplex able
20200813_17:48:01:080: (0200:0000) 1.9 = 0 100BASE-T2 half duplex able
20200813_17:48:01:116: (0100:0100) 1.8 = 1 extended status
20200813_17:48:01:116: (0080:0080) 1.7 = 1 (reserved)
20200813_17:48:01:116: (0040:0000) 1.6 = 0 MF preamble suppression
20200813_17:48:01:118: (0020:0000) 1.5 = 0 A/N complete
20200813_17:48:01:118: (0010:0000) 1.4 = 0 remote fault
20200813_17:48:01:118: (0008:0008) 1.3 = 1 A/N able
20200813_17:48:01:118: (0004:0000) 1.2 = 0 link status
20200813_17:48:01:119: (0002:0000) 1.1 = 0 jabber detect
20200813_17:48:01:125: (0001:0001) 1.0 = 1 extended capabilities
```

图 2-10 MII dump 方式读取 PHY 寄存器

注意：Horizon J2 中网络仅支持 `clause 22` 的标准读取 `phy` 寄存器相关信息，如果 `phy` 默认使用的是 `clause 45` 标准，需要根据 `phy datasheet` 说明来正确使用 `mii` 命令读写 `phy` 寄存器。

3. kernel 下 phy 调试

Kernel 下 PHY 调试的流程跟 uboot 下是类似的，也是要完成以下几件事情：

1. 检查 MAC 配置是否打开
2. 检查 PHY driver 是否存在
3. 配置 PHY 对应的 config 开关
4. 添加 PHY 的设备树节点

kernel 下 PHY debug 相关的文件分布如下：

```
1. kernel
2.   |— arch
3.   |   |— arm64
4.   |   |   |— boot
5.   |   |   |   |— dts
6.   |   |   |   |   |— hobot
7.   |   |   |   |   |   |— hobot-j2-sam.dts
8.   |   |   |   |   |   |— configs
9.   |   |   |   |   |   |   |— x2_perf_defconfig
10.  |— drivers
11.  |   |— net
12.  |   |   |— ethernet
13.  |   |   |   |— hobot
14.  |   |   |   |   |— Kconfig
15.  |   |   |   |   |— Makefile
16.  |   |   |   |   |   |— x2_eth.c
17.  |   |   |   |   |   |— phy
18.  |   |   |   |   |   |   |— Kconfig
19.  |   |   |   |   |   |   |— Makefile
20.  |   |   |   |   |   |   |   |— mdio_bus.c
21.  |   |   |   |   |   |   |   |— mdio_device.c
22.  |   |   |   |   |   |   |   |— phy.c
23.  |   |   |   |   |   |   |   |— phy-core.c
24.  |   |   |   |   |   |   |   |— realtek.c
```

3.1. 检查 MAC 是否使能

Kernel 下 MAC 的源文件所在路径为：

kernel/drivers/net/ethernet/hobot/ x2_eth.c

3.1.1. 检查 MAC config 是否配置

使能 MAC config 所在的路径为：

kernel/arch/arm64/configs/ x2_perf_defconfig

注意：具体要根据项目情况选择正确的 deconfig 文件

```
1075 CONFIG_VENDOR_HOBOT=y
1076 CONFIG_X2_ETH=y
```

图 3-1 Kernel 下 MAC config 配置

Kernel 下 MAC config 配置项为：CONFIG_X2_ETH，该项需要配置为使能。另外，CONFIG_X2_ETH 配置项依赖 CONFIG_VENDOR_HOBOT 配置项，该配置项也需要打开。

3.1.2. 检查 Kconfig 中是否有配置

Kernel 下 MAC 的 Kconfig 文件路径为：

kernel/drivers/net/ethernet/hobot/Kconfig

```
1 #
2 # Horizon network device configuration
3 #
4
5 config VENDOR_HOBOT
6     bool "Hobot x2 ethernet devices"
7     default y
8     ---help---
9     If you have a network (Ethernet) device belonging to this class, say Y.
10
11 if VENDOR_HOBOT
12
13     config X2_ETH
14         tristate "Hobot X2 Ethernet Device support"
15         select PHYLIB
16         select CRC32
17         select MII
18         depends on OF && HAS_DMA
19         ---help---
20         This driver supports the X2 Ethernet device from Hobot
21
```

图 3-2 Kernel 下 MAC Kconfig 配置

需要确认有 `VENDOR_HOBOT` 和 `X2_ETH` 配置项。另外需要根据依赖关系，检查其依赖项是否正确配置。

3.1.3. 检查 Makefile 中是否加入源文件编译

在 `kernel/drivers/net/ethernet/hobot/Makefile` 中检查是否加入 MAC 源文件编译。

```
1 #
2 # Makefile for the Hobot X2network device drivers.
3 #
4
5 obj-$(CONFIG_X2_ETH) += x2_eth.o
```

图 3-3 Kernel 下 MAC 源文件编译

3.2. PHY 相关配置与编译

3.2.1. PHY driver

在 `kernel/drivers/net/phy` 路径下，检查是否有 PHY driver 文件，在 PHY debug 时候，需要确认该目录下是否存在对应的 PHY driver。例如，realtek PHY 驱动文件文件为 `realtek.c`。如果 PHY driver 文件不存在，需要移植 PHY driver 或者实现 PHY driver。

```
qsemi.c      stel0Xp.c
realtek.c    swphy.c
realtek.o    swphy.h
```

图 3-4 Kernel 下 PHY driver 源文件

如果 PHY driver 文件存在，需要确定源文件中是否包含对应 PHY 的 driver。这个可以通过 PHY UID 来确认。

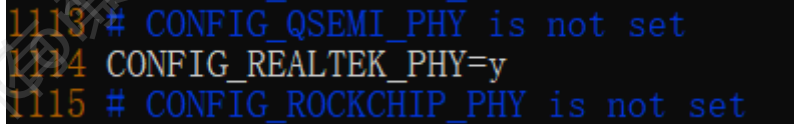
```
1. static struct phy_driver realtek_drvs[] = {
2.     .....
3.     .resume    = genphy_resume,
4.     }, {
5.     .phy_id    = 0x001cc916,
6.     .name      = "RTL8211F Gigabit Ethernet",
7.     .phy_id_mask = 0x001fffff,
8.     .features  = PHY_GBIT_FEATURES,
```

```
9.     .flags    = PHY_HAS_INTERRUPT,
10.    .config_aneg = &genphy_config_aneg,
11.    .config_init = &rtl8211f_config_init,
12.    .read_status = &genphy_read_status,
13.    .ack_interrupt = &rtl8211f_ack_interrupt,
14.    .config_intr = &rtl8211f_config_intr,
15.    .suspend = genphy_suspend,
16.    .resume = genphy_resume,
17. },
18. };
19.
20. module_phy_driver(realtek_drvs);
21.
22. static struct mdio_device_id __maybe_unused realtek_tbl[] = {
23.     { 0x001cc912, 0x001fffff },
24.     { 0x001cc914, 0x001fffff },
25.     { 0x001cc915, 0x001fffff },
26.     { 0x001cc916, 0x001fffff },
27.     {}
28. };
```

如果 PHY driver 源文件中没有包含对应的 PHY 的 driver，需要根据 PHY datasheet 来实现对应的 driver 并加入。

3.2.2. 使能 PHY config

如果 kernel/drivers/net/phy 路径下存在 PHY driver 源文件，需要在 kernel/arch/arm64/configs/x2_perf_defconfig(需要使用哪个 deconfig 文件需要根据具体项目来确定)使能 CONFIG_REALTEK_PHY 配置项。



```
1113 # CONFIG_QSEMI_PHY is not set
1114 CONFIG_REALTEK_PHY=y
1115 # CONFIG_ROCKCHIP_PHY is not set
```

图 3-5 Kernel 下 PHY config 配置

此外，还涉及到 CONFIG_MDIO_DEVICE、CONFIG_MDIO_BUS、CONFIG_PHYLIB 等几个配置。其中 CONFIG_MDIO_DEVICE、CONFIG_MDIO_BUS 是涉及 MDIO 功能的配置项，CONFIG_PHYLIB 是涉及 PHY 通用接口相关的配置项。如果有其他依赖项，请按照

config 的依赖关系正确配置，不在此一一描述。

```
1077 CONFIG_MDIO_DEVICE=y
1078 CONFIG_MDIO_BUS=y
1079 # CONFIG_MDIO_BCM_UNIMAC is not set
1080 # CONFIG_MDIO_BITBANG is not set
1081 # CONFIG_MDIO_BUS_MUX_GPIO is not set
1082 # CONFIG_MDIO_BUS_MUX_MMIOREG is not set
1083 # CONFIG_MDIO_HISI_FEMAC is not set
1084 # CONFIG_MDIO_OCTEON is not set
1085 CONFIG_PHYLIB=y
1086 CONFIG_SWPHY=y
```

图 3-6 Kernel 下 PHY 相关配置

3.2.3. Kconfig 中配置 config

在 x2_perf_defconfig 文件中使能 CONFIG_REALTEK_PHY 配置项后，需要在 kernel/drivers/net/phy/Kconfig 文件中配置 REALTEK_PHY 配置项。

```
364 config REALTEK_PHY
365     tristate "Realtek PHYs"
366     ---help---
367     Supports the Realtek 821x PHY.
368
```

图 3-7 Kernel 下 PHY Kconfig 配置

此外还需要检查 MDIO_DEVICE、MDIO_BUS、PHYLIB 等几个配置。

3.2.4. Makefile 中加入 PHY driver 源文件编译

配置好 PHY config 后，需要在 kernel/drivers/net/phy/Makefile 中加入 PHY driver 源文件的编译，如下：

```
75 obj-$(CONFIG_REALTEK_PHY) += realtek.o
```

图 3-8 Kernel 下 PHY 源文件编译

此外，还需要检查 mdio_bus.c, mdio_device.c, phy.c, phy-core.c 这几个源文件是否加入编译。如果源文件没有加入编译，请检查 deconfig, Kconfig, Makefile 中是否正确完成配置，或者说依赖项有没有正确打开。

3.2.5. 设备树中配置 PHY 设备节点

在 kernel/arch/arm64/boot/dts/hobot/hobot-j2-sam.dts(请根据具体项目确定使用哪个 dts 文件)的 ethernet 节点的 mdio 节点下添加 PHY 子设备节点。

```

244 &ethernet {
245     status = "okay";
246
247     pinctrl-names = "default";
248     pinctrl-0 = <&eth_func>;
249
250     /* local-mac-address = [ 00 11 22 33 44 55 ]; */
251     //fixed-link = <0 1 1000 0 0>;
252     clocks = <&eth0_clk>, <&eth0_ephy_2nddiv_clk>;
253     clock-names = "eth0_clk", "phy_ref_clk";
254     phy-mode = "rgmii";
255     phy-handle = <&phy1>;
256     mdio {
257         #address-cells = <0x1>;
258         #size-cells = <0x0>;
259         phy1: phy@3 {
260             compatible = "ethernet-phy-id001c.c916";
261             reg = <3>;
262         };
263     };
264 };
265

```

图 3-9 Kernel 下 dts 配置

PHY driver 是通过 compatible 属性来匹配的。compatible 属性是固定的格式：

1. ethernet-phy-idAAAA.BBBB" where
2. AAAA - The value of the 16 bit Phy Identifier 1 register as
3. 4 hex digits. This is the chip vendor OUI bits 3:18
4. BBBB - The value of the 16 bit Phy Identifier 2 register as
5. 4 hex digits. This is the chip vendor OUI bits 19:24,
6. followed by 10 bits of a vendor specific ID.

详细说明请参考 kernel/Documentation/devicetree/bindings/net/phy.txt 文件。

PHY address 是通过 reg 来配置的，该值需要根据具体的硬件电路来配置。

1. static inline int of_mdio_parse_addr(struct device *dev,
2. const struct device_node *np)
3. {
4. u32 addr;



```
5.  int ret;
6.
7.  ret = of_property_read_u32(np, "reg", &addr);
8.  if (ret < 0) {
9.      dev_err(dev, "%s has invalid PHY address\n", np->full_name);
10.     return ret;
11. }
12.
13. /* A PHY must have a reg property in the range [0-31] */
14. if (addr >= PHY_MAX_ADDR) {
15.     dev_err(dev, "%s PHY address %i is too large\n",
16.             np->full_name, addr);
17.     return -EINVAL;
18. }
19.
20. return addr;
21. }
```

如果 dts 中没有配置 reg 属性，那么 mdio 驱动会找到一个没有注册的可用的 PHY address 来注册 PHY。

4. 常见问题汇总

4.1. 硬件接线错误

4.1.1. MDIO 与 MDC 接反

MDIO 与 MDC 接反会导致 MII 命令读取 PHY 内部寄存器错误，在原理图设计时候需要注意检查 MDIO 与 MDC 接线是否正确。

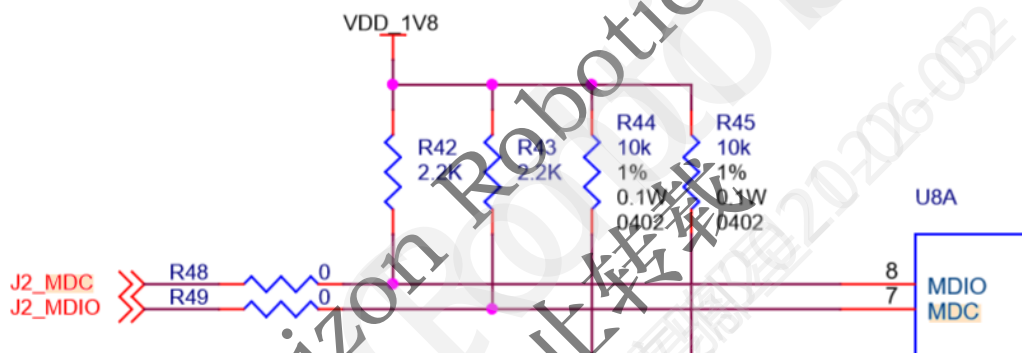


图 4-1 MDIO 与 MDC 接线错误

4.2. Horizon MAC 不支持延时

Horizon 的 MAC RGMII 接口时序如下：

2.4.9 EMAC Interface Timing

Figure 2-18 illustrates the RGMII (1000 Mbps max) RX timing diagram.

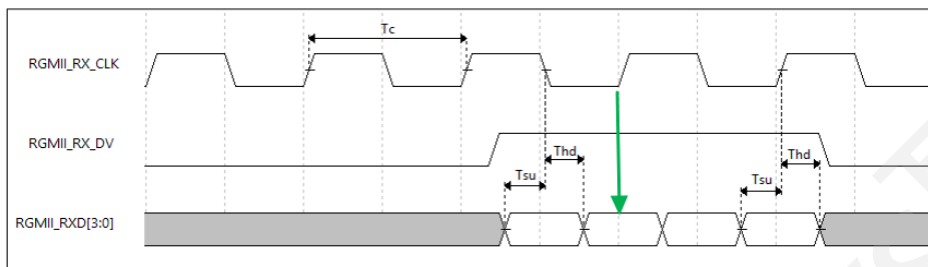


Figure 2-18 RGMII RX Timing Diagram

Figure 2-19 illustrates the RGMII (1000 Mbps max) TX timing diagram.

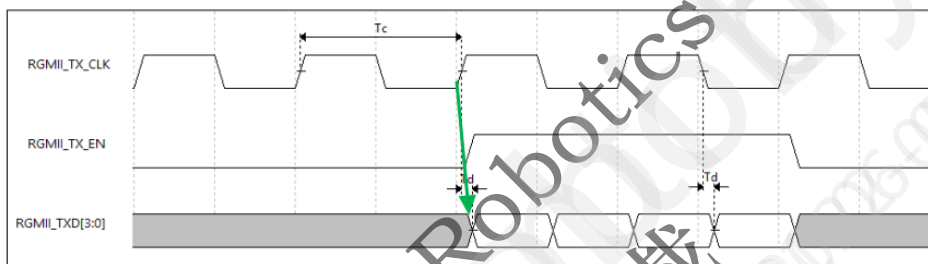


Figure 2-19 RGMII TX Timing Diagram

图 4-2 Horizon MAC RGMII 接口时序

TX 是 CLK 和 DATA 边沿对齐，需要对接芯片一侧把 CLK 移相 1/4 周期对齐 DATA 中心采样，RX 需要对接芯片一侧首先把 CLK 移相 1/4 周期直接对齐 DATA 中心，Horizon RX 逻辑直接用 CLK 采样 DATA，即 Horizon EMAC RGMII 接口不具备 CLK 移相功能，需要对接芯片完成 TX 和 RX 的 CLK 移相。无论是 EPHY 芯片，转换桥片，还是交换芯片，只要对方能够实现 TX 和 RX 的 CLK 移相，就可以和 Horizon 实现 RGMII-to-RGMII 对接，否则不行。