Motorcomm Inc

裕太以太网 Phy 芯片软件开发说明

The guide to use of the YT PHY chips on target board

Ver 2.11.2025-15:37

Motorcomm
2021/2/24

Revision History

Revision	Release Date	Summary
1.0	2021.2.24	首版
1.1	2022.5.4	增加第 9 章：9.以太网数据通路问题调试思路
1.2	2022.5.10	第 9 章”以太网数据通路问题调试思路”中增加案例拓朴图，同时更新文中部分描述
1.3	2022.6.6	增加 5.1.2.1 网上 phy.c 的方法常见出错处理
1.4	2022.8.19	7.3.4 Iperf 测试问题与解决中增加 mac<->phy 数据接口 check
1.5	2025.2.7	Phy driver 中增加部分 debug 方法
1.6	2025.2.10	Phy driver support .set_tunable/.get_tunable callback

User Guide Index

1. 前言

本文档主要说明如何在 Linux 操作系统下开发使用裕太公司的 Phy 系列芯片，包括千兆系列 8511（电口）、8521（光口+电口）和百兆系列 8512 等。

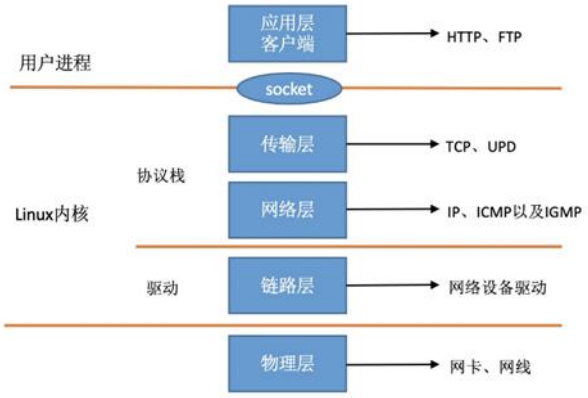
开发使用的参考文档包括：

- Datasheet
- Application notes
- 裕太驱动使用手册

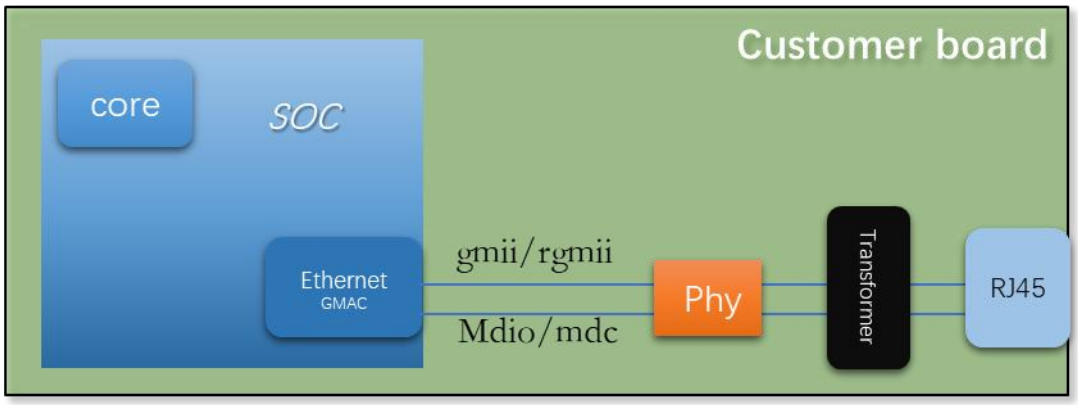
2. Linux 系统下的以太网架构

2.1 Linux 系统网络协议层架构

网络子系统是 linux 操作系统里很重要的一部分。关于这部分有很多的参考资料。这里主要说明一下 phy 芯片在整个子系统里的位置。从这个结构里看到，PHY 驱动的功能处于链路层。



2.2 以太网物理层与硬件连接



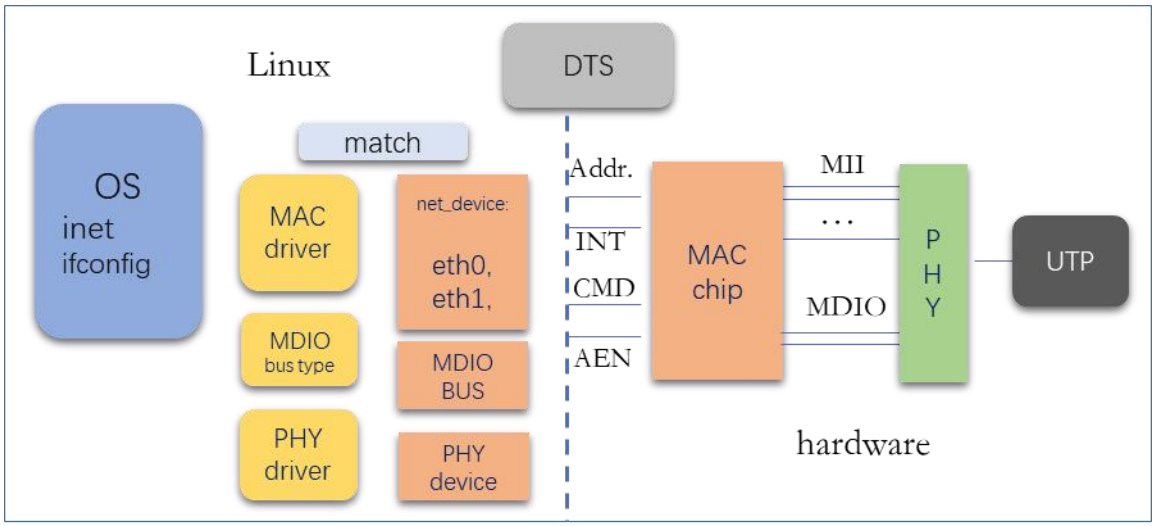
关于更多的 Phy 芯片对硬件连接的要求，请咨询硬件支持工程师。从软件角度，对 phy 芯片的控制主要包括二部分：

1） 与 MAC 设备的接口，即是 gmii 还是 rgmii。

2） Phy 芯片的地址正确配置，可以通过 mdio/mdc 正确访问到 phy 芯片的寄存器。

2.3 链路层与 Linux 网络设备管理

Linux 网络设备系统包括设备与驱动二大部分。网络设备驱动包括 MAC 层的驱动、MDIO 总结接口驱动与 phy 驱动。结合 linux 系统设备树定义以及设备管理系统，构成 phy 驱动在开发过程中涉及到的所有部分。示意如下图：



在 Linux 设备管理系统里，硬件设备对应于在/sys/devices 下有对应的节点。如 MAC 设备：

```
ls -l /sys/devices/platform/ | grep ethernet
drwxr-xr-x 5 root root       0 2011-01-01 13:00 ff290000.ethernet
```

同理，硬件的 mdio 总线和 phy 芯片也会对应有一个设备节点：

```
rk3288:/sys/devices/platform/ff290000.ethernet # ls -l | grep bus
lrwxrwxrwx 1 root root       0 2021-01-23 05:44 driver -> ../../../bus/platform/drivers/rk_gmac-dwmac
drwxr-xr-x 3 root root       0 2011-01-01 13:00 mdio_bus        ；mdio 设备节点
lrwxrwxrwx 1 root root       0 2021-01-23 10:37 subsystem -> ../../../bus/platform
```

```
rk3288: /sys/devices/platform/ff290000.ethernet/mdio_bus/stmmac-0 # ls -l | grep stmm
drwxr-xr-x 3 root root       0 2011-01-01 13:00 stmmac-0:04    ；地址为 4 的 phy 设备节点
```

设备节点总会有一个 driver 链接到对应的驱动程序：

```
rk3288:/sys/devices/platform/ff290000.ethernet/mdio_bus/stmmac-0/stmmac-0:04 # ls -l
total 0
lrwxrwxrwx 1 root root       0 2021-01-23 05:23 driver -> ../../../../../../bus/mdio_bus/drivers/YT8511 Gigabit Ethernet
```

2.4 Linux 以太网 phy 驱动基本开发流程

根据上图 linux 模块之间的关系，总结 Phy 驱动的开发流程如下：

1) 硬件设计。包括 PHY 芯片地址设定、与 MAC 接口模式（如 RGMII）、MAC 时钟的接入方式、PHY 芯片的复位管脚以及 PHY 芯片上电工作模式的设置。
2) 根据硬件连接，修改 linux 系统的设备定义树.dts 文件。通常是修改 GMAC 的定义以及 MII 管脚复用定义。具体需要看 SOC 芯片开发平台的定义。
3) 把 phy 驱动编译链接到到 Linux Kernel Image 里来。具体请参考文档《裕太驱动使用手册》。
4) 开机运行，检查目标机系统里有没有网络设备，如 eht0。如果有则进行下一步网络功能与性能的测试。如果没有，请检查：
   a) Phy 驱动有没被正确加载，参看文档《裕太驱动使用手册》。
   b) MAC 驱动有没有正确运行，参看开发平台提供的 MAC 驱动，重点检查(1)在 DTS 里配置的 compatible 字段与 MAC 驱动里的定义是否一致。(2)MAC 资源(io map)与 irq 是否加载正确。
   c) MDIO 是否访问正确。在 mdio 扫描代码里增加打印看 phy 寄存器的读写是否正确。
   d) Phy id 是否被正确识别。在 mdio 扫描代码里增加打印看读到的 phy id 与 phy 驱动的 phy id 是否一致。
   e) Phy 芯片是否能正常 linkup。正常的话 phy 应该可以 link up。如果不能请与硬件工程师确认 phy

芯片是否工作正常(上电时序、时钟，供电电压及复位连接等)，正常的话 phy 应该可以 link up。

5) 网络设备建议与正常 Link up 之后，就可以进行网络功能测试。主要是 ping 通。
   Ping 通是很关键的一步。Ping 通与以下配置有关：
   a) 如果是 rgmii 接口，在设备树配置文件里的 tx_delay 和 rx_dealy 的配置很重要，参考后面关于 rgmii delay line 的配置。
   b) PHY 的状态(link up, speed, duplex)是否正确。这个需要在 phy state machine 里增加打印进行跟踪（通常在 phy.c 里）。
   c) 通过统计计数(MAC 层，phy 层)来确认是哪个方向（tx 或者 rx）的问题。
6) 如果数据通路不通，请参考后面章节“以太网数据通路问题调试思路”的讨论。
7) 网络性能测试。主要是 iperf 性能测试。测试在 1000m 和 100m 等各种情况下的网络性能。网络性能测试中的问题，请参考后面章节“Iperf 测试问题与解决”
8) 到这里整个 phy 驱动的功能就算完成了。

3. Linux 设备树里 MAC 定义

MAC 设备的硬件描述在不同的平台里有区别。比如对于全志（Allwinner）平台，使用的是.fex 文件。而大多数的系统平台都使用 linux 的.dts 文件。

3.1 全志平台 MAC 设备定义举例

3.1.1 RGMII，CQA64

sys_config.fex 文件：

注意：在全志平台下，.fex 文件的配置优先级大于.dts/.dtsi 文件。

```
;-----------------------------------------------------------------------------
;os life cycle para configuration
;-----------------------------------------------------------------------------

;-------------------------------------------------------------------------;
; 10/100/100Mbps Ethernet MAC Controller Configure                        ;
;-------------------------------------------------------------------------;
;    配置选项：                                                           ;
;    gmac_used   ---   1: gmac used, 0: not used                          ;
;-------------------------------------------------------------------------;
;            MII   GMII   RGMII          MII   GMII   RGMII          MII   GMII   RGMII      ;
;PA00~03   *      *       *      PA10          *      *      PA20    *     *       *         ;
;    PA04          *              PA11~14   *     *      *      PA21    *     *              ;
;    PA05          *              PA15            *             PA22    *     *              ;
;    PA06          *              PA16            *             PA23    *     *              ;
;    PA07          *              PA17            *             PA24    *     *              ;
;    PA08   *      *              PA18            *             PA25          *       *        ;
;    PA09   *      *       *      PA19    *       *      *      PA26~27 *     *       *        ;
;-------------------------------------------------------------------------;
[gmac_para]
gmac_used               = 1
compatible              = "allwinner,sunxi-gmac"
phy-mode                = "rgmii"
gmac_txd0               = port:PD18<4><default><3><default>
gmac_txd1               = port:PD17<4><default><3><default>
gmac_txd2               = port:PD16<4><default><3><default>
gmac_txd3               = port:PD15<4><default><3><default>
gmac_txclk              = port:PD19<4><default><3><default>
gmac_txen               = port:PD20<4><default><3><default>
gmac_rxd0               = port:PD11<4><default><3><default>
gmac_rxd1               = port:PD10<4><default><3><default>
gmac_rxd2               = port:PD09<4><default><3><default>
gmac_rxd3               = port:PD08<4><default><3><default>
gmac_rxdv               = port:PD13<4><default><3><default>
gmac_rxclk              = port:PD12<4><default><3><default>
gmac_clkin              = port:PD21<4><default><3><default>
gmac_mdc                = port:PD22<4><default><3><default>
gmac_mdio               = port:PD23<4><default><3><default>
gmac-power0             = "vcc-rgmii"
gmac-power1             = ""
gmac-power2             = ""
tx-delay                = 7
```

```
rx-delay                        = 0
```

sun50iw1p1.dtsi:

```
gmac0: eth@01c30000 {
    compatible = "allwinner,sunxi-gmac";
    reg = <0x0 0x01c30000 0x0 0x1054>,
          <0x0 0x01c00000 0x0 0x30>;
    pinctrl-names = "default";
    pinctrl-0          = <&gmac_pins_a>;
    interrupts           = <GIC_SPI 82 IRQ_TYPE_LEVEL_HIGH>;
    interrupt-names = "gmacirq";
    clocks      = <&clk_gmac>;
    clock-names = "gmac";
    phy-mode = "rgmii";
    tx-delay = <7>;
    rx-delay = <31>;
    phy-rst;
    gmac_power1 = "axp81x_dldo2:2500000";
    gmac_power2 = "axp81x_eldo2:1800000";
    gmac_power3 = "axp81x_fldo1:1200000";
    status = "okay";
};
```

3.2 瑞芯微平台 MAC 设备定义举例

3.2.1 RGMII，RK3399

cqrk3399-box-linux.dtsi：

```
&gmac {
    phy-supply = <&vcc_phy>;
    phy-mode = "rgmii";
    clock_in_out = "input";
    snps,reset-gpio = <&gpio3 15 GPIO_ACTIVE_LOW>;
    //snps,reset-gpio = <&gpio1 1 GPIO_ACTIVE_LOW>;
    snps,reset-active-low;
    snps,reset-delays-us = <0 10000 50000>;
    assigned-clocks = <&cru SCLK_RMII_SRC>;
    assigned-clock-parents = <&clkin_gmac>;
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&rgmii_pins>;
    pinctrl-1 = <&rgmii_sleep_pins>;
    tx_delay = <0x28>;
    rx_delay = <0x0>;
    status = "okay";
};
```

3.2.2 RGMII，RK3288

rk3288-evb-r86.dtsi：

```
&gmac {
    phy-supply = <&vcc_phy>;
    phy-mode = "rgmii";
    clock_in_out = "input";
    snps,reset-gpio = <&gpio4 8 0>;
    snps,reset-active-low;
    snps,reset-delays-us = <0 10000 50000>;
    assigned-clocks = <&cru SCLK_MAC>;
    assigned-clock-parents = <&ext_gmac>;
    pinctrl-names = "default";
    pinctrl-0 = <&rgmii_pins>;
    tx_delay = <0x0>;
    rx_delay = <0x0>;
    //max-speed = <100>;
    status = "okay";
};
```

3.3 飞腾（Phytium）平台 MAC 设备定义举例

3.3.1 RGMII，FT2004

arch/arm64/boot/dts/phytium/ft2004-generic-psci-soc.dtsi：

```
gmac0: eth@2820c000 {
            compatible = "snps,dwmac";
            reg = <0x0 0x2820c000 0x0 0x2000>;
            interrupts = <GIC_SPI 49 IRQ_TYPE_LEVEL_HIGH>;
            interrupt-names = "macirq";
```

```
            clocks = <&clk250mhz>;
            clock-names = "stmmaceth";
            status = "disabled";

            snps,pbl = <16>;
            snps,fixed-burst;
            snps,axi-config = <&phytium_axi_setup>;
            snps,force_sf_dma_mode;
            snps,multicast-filter-bins = <64>;
            snps,perfect-filter-entries = <128>;
            tx-fifo-depth = <4096>;
            rx-fifo-depth = <4096>;
            max-frame-size = <9000>;
    };
```

3.4 德州仪器（TI）平台 MAC 设备定义举例

3.4.1 RGMII，am355x

arch/arm/boot/dts/am335x-boneblack.dts：

```
mac: ethernet@4a100000 {
        compatible = "ti,am335x-cpsw","ti,cpsw"; //“cpsw”用于匹配驱动
        ti,hwmods = "cpgmac0";//首个 slave 的名称，要与驱动里的匹配
        clocks = <&cpsw_125mhz_gclk>, <&cpsw_cpts_rft_clk>;
        clock-names = "fck", "cpts";
        cpdma_channels = <8>;
        ale_entries = <1024>;
        bd_ram_size = <0x2000>;
        mac_control = <0x20>; //MAC 控制寄存器地址
        slaves = <2>; //MAC 数量：编号 0，1，…
        active_slave = <0>;//首选活动 MAC 号
        cpts_clock_mult = <0x80000000>;
        cpts_clock_shift = <29>;
        reg = <0x4a100000 0x800
                0x4a101200 0x100>;//寄存映射地址
        #address-cells = <1>;
        #size-cells = <1>;
        /*
         * c0_rx_thresh_pend
         * c0_rx_pend
         * c0_tx_pend
         * c0_misc_pend
         */
        interrupts = <40 41 42 43>;//中断号
        ranges;
        syscon = <&scm_conf>;
        status = "disabled";

        davinci_mdio: mdio@4a101000 {
                compatible = "ti,cpsw-mdio","ti,davinci_mdio";
                #address-cells = <1>;
                #size-cells = <0>;
                ti,hwmods = "davinci_mdio";
                bus_freq = <1000000>;
                reg = <0x4a101000 0x100>;
                status = "disabled";
        };

        cpsw_emac0: slave@4a100200 {
                /* Filled in by U-Boot */
                mac-address = [ 00 00 00 00 00 00 ];
        };//MAC 0

        cpsw_emac1: slave@4a100300 {
                /* Filled in by U-Boot */
                mac-address = [ 00 00 00 00 00 00 ];
        };//MAC 1

        phy_sel: cpsw-phy-sel@44e10650 {
                compatible = "ti,am3352-cpsw-phy-sel";
                reg= <0x44e10650 0x4>; //PHY 寄存器基址和大小
                reg-names = "gmii-sel"; //PHY 接口模式
        };
};
```

4. RGMII 接口 delay line 调整

对于 RGMII 接口，rx 和 tx delay line 调整的过程如下：

1） 如果有参考的板子，可以按照参考值尝试。如果不行就按下面的步骤再进行。

2） 在 DTS 里，把 tx_delay 和 rx_delay 都设置为 0。

3） 上电看能不能 ping 通。

4） 如果不能 ping 通或者 ping 时断时续，就按下面的步骤，修改 PHY 本身的 delay 配置。通常 rx_delay 不用调整。下面调整 tx_delay。

注：YT8511 Phy 的 tx_delay 是在扩展寄存器（Ext. Reg）0xc.bit[7:4]里配置。YT8511 Phy Ext. Reg 读写的规则是：先写扩展寄存器的地址到 mii reg 0x1e 里，再相应地读写 mii reg 0x1f（请参考 phy 应用手册）。Phy 寄存器读写方法参见下面的说明。

a) 设置 Ext. Reg 0xc.bit[7:4]为 0xf
b) 软件复位 phy（mii_reg_0.b15=1）
c) Link up 之后，看能不能 ping 通。
d) 如果不行，按二分法减少 delay 值，设置 Ext. Reg 0xc.bit[7:4]，再从 b)步骤尝试，直到 ping 通。如果尝试了所有值都不通，再修改 DTS 里 tx_delay 为 0xf 再从步骤 3）开始尝试。

5） 最后找到的值配置 dts 文件或者 Ext. Reg 0xc.bit[7:4]。建议优先设置 dts 文件。YT8511 Ext. Reg 0xc.bit[7:4]的缺省值是 5，建议使用缺省值，这样减少代码的配置。

5. 寄存器读写工具

5.1 PHY 寄存器读写

5.1.1 Linux MAC 设备 mdio 方法

通常 linux 系统的 mac 设备都有一个通用的方法对 phy 寄存器进行读写，不太好用，但不需要额外工作可以直接使用。方法是：

1） 找到 phy 设备，如：

```
ls -l /sys/devices/platform/ | grep eth
drwxr-xr-x 5 root root       0 2011-01-01 13:00 ff290000.ethernet
```

上面例子里，找到 mac 设备 ff290000.ethernet，继续找到 phy 设备，如下：

```
ls -l /sys/devices/platform/ff290000.ethernet/mdio_bus/
total 0
drwxr-xr-x 5 root root 0 2011-01-01 13:00 stmmac-0
ls -l /sys/devices/platform/ff290000.ethernet/mdio_bus/stmmac-0/
total 0
lrwxrwxrwx 1 root root       0 2021-01-23 13:32 device -> ../../../ff290000.ethernet
drwxr-xr-x 2 root root       0 2011-01-01 13:00 power
drwxr-xr-x 3 root root       0 2011-01-01 13:00 stmmac-0:00
drwxr-xr-x 3 root root       0 2011-01-01 13:00 stmmac-0:04
lrwxrwxrwx 1 root root       0 2021-01-23 13:32 subsystem -> ../../../../../class/mdio_bus
-rw-r--r-- 1 root root 4096 2011-01-01 13:00 uevent
```

找到 phy 设备 stmmac-0:04：

```
cd /sys/devices/platform/ff290000.ethernet/mdio_bus/stmmac-0/stmmac-0:04
ls -l
total 0
lrwxrwxrwx 1 root root       0 2021-01-23 05:23 driver -> ../../../../../bus/mdio_bus/drivers/YT8511 Gigabit Ethernet
-r--r--r-- 1 root root 4096 2021-01-23 05:23 phy_has_fixups
-r--r--r-- 1 root root 4096 2021-01-23 05:23 phy_id
-r--r--r-- 1 root root 4096 2021-01-23 05:23 phy_interface
-rw-r--r-- 1 root root 4096 2021-01-23 05:23 phy_registers
drwxr-xr-x 2 root root       0 2011-01-01 13:00 power
lrwxrwxrwx 1 root root       0 2021-01-23 05:23 subsystem -> ../../../../../bus/mdio_bus
-rw-r--r-- 1 root root 4096 2011-01-01 13:00 uevent
```

对 phy 寄存器的读写就是操作文件 *phy_registers*。

2） 读寄存器

```
cat phy_registers
```

3） 写 mii 寄存器

例：写 mii reg 0 为 0x9000:

echo 0x0 0x9000> phy_registers

4） 读 ext 寄存器

例：读 ext reg 0xc:

echo 0x1e 0xc> phy_registers && cat phy_registers

30: 0xc

31: 0x8052

返回值在 mii reg 0x1f（31d）里，例中为 0x8052。

5） 写 ext 寄存器

例：写 ext reg 0xc 值为 0x80f6:

echo 0x1e 0xc> phy_registers && echo 0x1f 0x80f6> phy_registers

5.1.2 网上 phy.c 的方法

在网上流行 phy.c 读取 phy 寄存器的方法，编译成目标板可执行 elf 文件，再在目标板上运行。

Phy.c 源文件：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <linux/mii.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <net/if.h>
#include <linux/sockios.h>
#include <linux/types.h>
#include <netinet/in.h>

#define reteck(ret)         \
            if(ret < 0){        \
                printf("%m! \"%s\" : line: %d\n", __func__, __LINE__);      \
                goto lab;       \
            }

#define help() \
    printf("mdio:\n");                              \
    printf("read operation: mdio reg_addr\n");              \
    printf("write operation: mdio reg_addr value\n");       \
    printf("For example:\n");                       \
    printf("mdio eth0 1\n");                        \
    printf("mdio eth0 0 0x12\n\n");             \
    exit(0);

int sockfd;

int main(int argc, char *argv[]){

    if(argc == 1 || !strcmp(argv[1], "-h")){
        help();
    }

    struct mii_ioctl_data *mii = NULL;
    struct ifreq ifr;
    int ret;

    memset(&ifr, 0, sizeof(ifr));
    strncpy(ifr.ifr_name, argv[1], IFNAMSIZ - 1);

    sockfd = socket(PF_LOCAL, SOCK_DGRAM, 0);
    reteck(sockfd);

    //get phy address in smi bus
    ret = ioctl(sockfd, SIOCGMIIPHY, &ifr);
    reteck(ret);

    mii = (struct mii_ioctl_data*)&ifr.ifr_data;

    if(argc == 3){

        mii->reg_num       = (uint16_t)strtoul(argv[2], NULL, 0);

        ret = ioctl(sockfd, SIOCGMIIREG, &ifr);
        reteck(ret);

        printf("read phy addr: 0x%x    reg: 0x%x      value : 0x%x\n", mii->phy_id, mii->reg_num, mii->val_out);
        }else if(argc == 4){

        mii->reg_num       = (uint16_t)strtoul(argv[2], NULL, 0);
        mii->val_in        = (uint16_t)strtoul(argv[3], NULL, 0);

        ret = ioctl(sockfd, SIOCSMIIREG, &ifr);
        reteck(ret);

        printf("write phy addr: 0x%x    reg: 0x%x    value : 0x%x\n", mii->phy_id, mii->reg_num, mii->val_in);
    }

lab:
    close(sockfd);
```

```
    return 0;
}
```

如，在目标板上直接编译源文件 phy.c 为 elf 文件 phyreg:

gcc -o phyreg phy.c

这样，系统起来后就有 phy 寄存读写工具软件 phyreg 了。

(1) Phy 寄存器读

./phyreg eth0 0x1e

(2) Phy 寄存器写

./phyreg eth0 0x1e 0xc

(3) 读 ext 寄存器

例：读 ext reg 0xc:

./phyreg eth0 0x1e 0xc && ./phyreg eth0 0x1f

(4) 写 ext 寄存器

例：写 ext reg 0xc 值为 0x80f6:

./phyreg eth0 0x1e 0xc && ./phyreg eth0 0x1f 0x80f6

5.1.2.1网上 phy.c 的方法常见出错处理

(1) Operation not supported! "main" : line: 50

原因：指定的 Phy 设备没有 power up。

解决方法：请硬件检查 phy 的供电。

(2) read phy addr: 0x1 reg: 0x3 value: 0xfffb

原因：在系统上电时，Phy 设备供电可能是正常的，但随后可能供电有问题了。

解决方法：请硬件检查 phy 的供电是否稳定，然后重启设备。

5.1.3 phy driver 中集成的 phy 寄存器读写

```
root@bpi-r3:/sys/kernel/debug/yt# echo > phyreg
[ 1320.908137] usage:
[ 1320.910168]   read mii reg:          echo ethx mii read reg > /sys/kernel/debug/yt/phyreg
[ 1320.917631]   write mii reg val:     echo ethx mii write reg val > /sys/kernel/debug/yt/phyreg
[ 1320.925899]   read ext reg:          echo ethx ext read reg > /sys/kernel/debug/yt/phyreg
[ 1320.933380]   write ext reg val:     echo ethx ext write reg val > /sys/kernel/debug/yt/phyreg
[ 1320.941640]   read mmd reg:          echo ethx mmd read device reg > /sys/kernel/debug/yt/phyreg
[ 1320.949724]   write mmd reg val:     echo ethx mmd write device reg val > /sys/kernel/debug/yt/phyreg
root@bpi-r3:/sys/kernel/debug/yt# echo ethx mii read 0x2 > phyreg
[ 1354.578145] read mii reg = 0x2, val = 0x4f51
root@bpi-r3:/sys/kernel/debug/yt# echo ethx mii read 0x3 > phyreg
[ 1358.748154] read mii reg = 0x3, val = 0xea19
```

5.2 GMAC 寄存器读写

这个依赖于不同的平台。以下以瑞芯微 rk3399 开发平台为例。

5.2.1 内存读写工具: io

注意 GMAC 的基地址是 0xff730000。GMAC 基地址可以在 DTS 文件中查到。

1） 读 GMAC 寄存器（32 位寄存器，从偏移为 0 开始的 4 个寄存器）

```
[root@cqrk3399:/]# io -4 -l 16 0xff730000        ;显示当前 gpio1 的 in/out，及值配置
ff730000:    00000018 01024018 00000000 00000000 ;gpio1 a[1]=input, val=0
```

2） 写 GMAC 寄存器（32 位寄存器，偏移地址为 4）

```
[root@cqrk3399:/]# io -4 -w 0xff730004 0x0102401a; GPIO DDR 寄存器，gpio1 a[1]=output
```

3） 写 GMAC 寄存器（32 位寄存器，偏移地址为 0）

```
[root@cqrk3399:/]# io -4 -w 0xff730000 0x1a            ; GPIO DR 寄存器，gpio1 a[1]=output 1
```

6. 收发包统计计数查看

在定位 phy 收发包问题时，重要的一步是分清是发送问题还是接收问题。所以查看收发包的计数可以帮忙定位问题。

6.1 MAC 层收发包计数

6.1.1 ifconfig

ifconfig eth0

```
eth0      Link encap:Ethernet  HWaddr 5e:8a:8a:5b:9f:74  Driver rk_gmac-dwmac
          inet addr:192.168.1.202  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::d6ea:73da:63fa:8827/64 Scope: Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1993 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1046 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:142969 TX bytes:128206
          Interrupt:41
```

这里的 RX/TX packets 表示 MAC 成功接收和发出的包数。通过多次读取并对比可以确定哪个方向有问题。

6.1.2 /proc/net/dev

Linux proc 文件系统也提供了网络设备 MAC 层的包统计信息：

cat /proc/net/dev

```
Inter-|   Receive                                                |  Transmit
 face |bytes    packets errs drop fifo frame compressed multicast|bytes    packets errs drop fifo colls carrier compressed
  sit0:       0       0    0    0    0     0          0         0        0       0    0    0    0     0       0          0
    lo:     968      11    0    0    0     0          0         0      968      11    0    0    0     0       0          0
  eth0:  179476    2479    0    0    0     0          0         0   157092    1393    0    0    0     0       0          0
```

6.2 PHY 层收发包计数

Phy 层的计数功能取决于 phy 芯片。请参考 phy 芯片的 data sheet。下面的例子是 YT8511 提供的功能。

6.2.1 YT8511 的计数

1） 初始化 YT8511 计数功能。注意只需要调用一次。

init_8511_cnt.sh：

```
#!/bin/sh

./phyreg eth0 0x1e 0xa0 && ./phyreg eth0 0x1f 0xe8c0 && ./phyreg eth0 0x1f
```

2） 读取当前计数。注意计数是读清的。

cnt_8511.sh：

```
#!/bin/sh

#YT8511 PHY counters
echo "8511 MII counters:good 64-1518B"
./phyreg eth0 0x1e 0xad > null && ./phyreg eth0 0x1f
./phyreg eth0 0x1e 0xae > null && ./phyreg eth0 0x1f

echo "8511 MII counters:good >1518B"
./phyreg eth0 0x1e 0xaf > null && ./phyreg eth0 0x1f
./phyreg eth0 0x1e 0xb0 > null && ./phyreg eth0 0x1f

echo "8511 MII counters:good <64B"
./phyreg eth0 0x1e 0xb1 > null && ./phyreg eth0 0x1f
./phyreg eth0 0x1e 0xb2 > null &&./phyreg eth0 0x1f

echo "8511 MII counters:err 64-1518B"
```

```
./phyreg eth0 0x1e 0xb3 > null && ./phyreg eth0 0x1f

echo "8511 MII counters:err >1518B"
./phyreg eth0 0x1e 0xb4 > null && ./phyreg eth0 0x1f

echo "8511 MII counters:err <64B"
./phyreg eth0 0x1e 0xb5 > null && ./phyreg eth0 0x1f

echo "8511 MII counters:err no SFD"
./phyreg eth0 0x1e 0xb5 > null && ./phyreg eth0 0x1f

echo "8511 UTP counters:good 64-1518B"
./phyreg eth0 0x1e 0xa3 > null && ./phyreg eth0 0x1f
./phyreg eth0 0x1e 0xa4 > null && ./phyreg eth0 0x1f

echo "8511 UTP counters:good >1518B"
./phyreg eth0 0x1e 0xa5 > null && ./phyreg eth0 0x1f
./phyreg eth0 0x1e 0xa6 > null && ./phyreg eth0 0x1f

echo "8511 UTP counters:good <64B"
./phyreg eth0 0x1e 0xa7 > null && ./phyreg eth0 0x1f
./phyreg eth0 0x1e 0xa8 > null && ./phyreg eth0 0x1f

echo "8511 UTP counters:err 64-1518B"
./phyreg eth0 0x1e 0xA9 > null && ./phyreg eth0 0x1f

echo "8511 UTP counters:err >1518B"
./phyreg eth0 0x1e 0xAA > null && ./phyreg eth0 0x1f

echo "8511 UTP counters:err <64B"
./phyreg eth0 0x1e 0xAB > null && ./phyreg eth0 0x1f

echo "8511 UTP counters:err no SFD"
./phyreg eth0 0x1e 0xAC > null && ./phyreg eth0 0x1f
```

注： 1） MII counter，指 phy 从 GMII/RGMII 接收到的包数。

2） UTP counter，指 phy 从 UTP 网线侧接收到的包数。

6.2.2 phy driver 中集成的 phy 收发包统计

6.2.2.1 checker tool

```
root@bpi-r3:/sys/kernel/debug/yt# echo > checker
[ 1915.738101] usage:
[ 1915.740131]    checker: echo ethx utp(1)|sgmii(2)> /sys/kernel/debug/yt/checker
[ 1915.747421]    eg, echo ethx 1 > /sys/kernel/debug/yt/checker
root@bpi-r3:/sys/kernel/debug/yt# echo ethx 1 > checker
[ 1944.688410] YT8821 utp ib valid(64-1518Byte): 0
[ 1944.693070] YT8821 utp ib valid(>1518Byte): 0
[ 1944.697526] YT8821 utp ib valid(<64Byte): 0
[ 1944.701787] YT8821 utp ib crc err(64-1518Byte): 0
[ 1944.706537] YT8821 utp ib crc err(>1518Byte): 0
[ 1944.711134] YT8821 utp ib crc err(<64Byte): 0
[ 1944.715536] YT8821 utp ib no sfd: 0
[ 1944.719144] YT8821 utp ob valid(64-1518Byte): 0
[ 1944.723775] YT8821 utp ob valid(>1518Byte): 0
[ 1944.728255] YT8821 utp ob valid(<64Byte): 0
[ 1944.732484] YT8821 utp ob crc err(64-1518Byte): 0
```

[1944.737229] YT8821 utp ob crc err(>1518Byte): 0
[1944.741815] YT8821 utp ob crc err(<64Byte): 0
[1944.746218] YT8821 utp ob no sfd: 0
root@bpi-r3:/sys/kernel/debug/yt# echo ethx 2 > checker
[1977.998457] YT8821 serdes ib valid(64-1518Byte): 0
[1978.003378] YT8821 serdes ib valid(>1518Byte): 0
[1978.008117] YT8821 serdes ib valid(<64Byte): 0
[1978.012607] YT8821 serdes ib crc err(64-1518Byte): 0
[1978.017613] YT8821 serdes ib crc err(>1518Byte): 0
[1978.022459] YT8821 serdes ib crc err(<64Byte): 0
[1978.027120] YT8821 serdes ib no sfd: 0
[1978.030980] YT8821 serdes ob valid(64-1518Byte): 0
[1978.035872] YT8821 serdes ob valid(>1518Byte): 0
[1978.040602] YT8821 serdes ob valid(<64Byte): 0
[1978.045092] YT8821 serdes ob crc err(64-1518Byte): 0
[1978.050116] YT8821 serdes ob crc err(>1518Byte): 0
[1978.054951] YT8821 serdes ob crc err(<64Byte): 0
[1978.059618] YT8821 serdes ob no sfd: 0
root@bpi-r3:/sys/kernel/debug/yt#

6.2.2.2 ethool tool

root@bpi-r3:/sys/kernel/debug/yt# ethtool --phy-statistics ethx
PHY statistics:
    UTP ib valid(64B~1518B): 0
    UTP ib valid(>1518B): 0
    UTP ib valid(<64B): 0
    UTP ib crc err(64B~1518B): 0
    UTP ib crc err(>1518B): 0
    UTP ib crc err(<64B): 0
    UTP ib no sfd: 0
    UTP ob valid(64B~1518B): 0
    UTP ob valid(>1518B): 0
    UTP ob valid(<64B): 0
    UTP ob crc err(64B~1518B): 0
    UTP ob crc err(>1518B): 0
    UTP ob crc err(<64B): 0
    UTP ob no sfd: 0
    SERDES ib valid(64B~1518B): 0
    SERDES ib valid(>1518B): 0
    SERDES ib valid(<64B): 0
    SERDES ib crc err(64B~1518B): 0
    SERDES ib crc err(>1518B): 0
    SERDES ib crc err(<64B): 0
    SERDES ib no sfd: 0
    SERDES ob valid(64B~1518B): 0

SERDES ob valid(>1518B): 0
SERDES ob valid(<64B): 0
SERDES ob crc err(64B~1518B): 0
SERDES ob crc err(>1518B): 0
SERDES ob crc err(<64B): 0
SERDES ob no sfd: 0
root@bpi-r3:/sys/kernel/debug/yt#

7. 以太网功能与性能测试

7.1 测试系统连接

配置目标板 ip，如: 192.168.1.202
主机（Window10 机）ip，如: 192.168.1.112

7.2 功能测试

7.2.1 Ping 通

ping -h

```
Usage: ping [-aAbBdDfhLnOqrRUvV] [-c count] [-i interval] [-I interface]
            [-m mark] [-M pmtudisc_option] [-l preload] [-p pattern] [-Q tos]
            [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
            [-w deadline] [-W timeout] [hop1 ...] destination
```

ifconfig eth0 192.168.1.202 up

ping -c 4 192.168.1.101

```
PING 192.168.1.101 (192.168.1.101) 56(84) bytes of data.
64 bytes from 192.168.1.101: icmp_seq=1 ttl=128 time=2.37 ms
64 bytes from 192.168.1.101: icmp_seq=2 ttl=128 time=2.66 ms
64 bytes from 192.168.1.101: icmp_seq=3 ttl=128 time=2.88 ms
64 bytes from 192.168.1.101: icmp_seq=4 ttl=128 time=3.11 ms

--- 192.168.1.101 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 2.378/2.760/3.112/0.274 ms
```

7.2.2 查看 phy Link 状态

```
[ 2768.258718] rk_gmac-dwmac ff290000.ethernet eth0: Link is Down
[ 2770.265468] rk_gmac-dwmac ff290000.ethernet eth0: Link is Up - 1Gbps/Full - flow control rx/tx
```

7.2.3 查看 phy Speed

可以识别速率 1000m，100m，10m。

```
[ 2770.265468] rk_gmac-dwmac ff290000.ethernet eth0: Link is Up - 1Gbps/Full - flow control rx/tx
[ 2854.545443] rk_gmac-dwmac ff290000.ethernet eth0: Link is Up - 100Mbps/Full - flow control rx/tx
```

7.2.4 查看网络状态的工具程序

7.2.4.1ethtool

Ethtool 工具程序可以查看当前网络的各种参数，但不一定目标板系统上都有此命令。关于此工具的使用请参考网上资料，此处略。

ethtool eth0

```
Settings for eth0:
```

```
        Supported ports: [ TP MII ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Supported pause frame use: Symmetric Receive-only
        Supports auto-negotiation: Yes
...
```

7.2.4.2ip

Linux 工具程序 ip 也可以查看当前网卡的状态：

ip --help

```
Usage: ip [ OPTIONS ] OBJECT { COMMAND | help }
       ip [ -force ] -batch filename
where  OBJECT := { link | address | addrlabel | route | rule | neighbor | ntable |
                   tunnel | tuntap | maddress | mroute | mrule | monitor | xfrm |
                   netns | l2tp | fou | tcp_metrics | token | netconf }
       OPTIONS := { -V[ersion] | -s[tatistics] | -d[etails] | -r[esolve] |
                    -h[uman-readable] | -iec |
                    -f[amily] { inet | inet6 | ipx | dnet | mpls | bridge | link } |
                    -4 | -6 | -I | -D | -B | -0 |
                    -l[oops] { maximum-addr-flush-attempts } | -br[ief] |
                    -o[neline] | -t[imestamp] | -ts[hort] | -b[atch] [filename] |
                    -rc[vbuf] [size] | -n[etns] name | -a[ll] | 聽-c[olor]}
```

ip link

```
3: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN mode DEFAULT group default qlen 1000
    link/ether 5e:8a:8a:5b:9f:74 brd ff:ff:ff:ff:ff:ff
```

NO-CARRIER 表示 link down。

ip link

```
3: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
link/ether 5e:8a:8a:5b:9f:74 brd ff:ff:ff:ff:ff:ff
```

link up 时的状态。

7.2.4.3netstat

netstat 也可以查看网口状态。不过依赖于此程序的具体实现。如：

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        0      0      0 0             0      0      0      0 BMU
eth1       1500        0      0      0 0             0      0      0      0 BMU
lo        65536     3472      0      0 0          3472      0      0      0 LRU
```

但也有不支持此先项的：

netstat -i

```
See netstat --help
netstat: Unknown option i
```

7.3 性能测试

主要是通常的 iperf 测试。如果获取 iperf 可执行程序，请参考官方下载地址：

https://iperf.fr/iperf-download.php

7.3.1 iperf3 server 命令

```
iperf3 -s -i 1 -p 55555 -D
```

7.3.2 Iperf3 client 命令

例如，可执行文件 iperf3.2 放在根目录下：

```
cd /
./iperf3.2 -c 192.168.1.112 -i 10 -p 55555 -b 1G -t 60
```

7.3.3 测试结果举例

(1) 千兆 Tcp

```
[root@cqrk3399:/]# ./iperf3.2 -c 192.168.1.101 -i 10 -p 55555 -b 1G -t 10
./iperf3.2 -c 192.168.1.100 -i 10 -p 55555 -b 100M -t 10
```

```
Connecting to host 192.168.1.112, port 55555
[  5] local 192.168.1.202 port 47078 connected to 192.168.1.112 port 55555
init usec: 1358499186, usec: 639818
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-10.00  sec  1.05 GBytes   902 Mbits/sec  113    212 KBytes
[  5]  10.00-20.00  sec  1.10 GBytes   948 Mbits/sec    0    212 KBytes
[  5]  20.00-30.00  sec  1.10 GBytes   949 Mbits/sec    0    212 KBytes
[  5]  30.00-40.00  sec  1.10 GBytes   949 Mbits/sec    0    212 KBytes
[  5]  40.00-50.00  sec  1.10 GBytes   949 Mbits/sec    0    212 KBytes
[  5]  50.00-60.00  sec  1.10 GBytes   949 Mbits/sec    0    212 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-60.00  sec  6.57 GBytes   941 Mbits/sec  113             sender
[  5]   0.00-60.00  sec  6.57 GBytes   941 Mbits/sec                  receiver
iperf Done.
```

(2) 百兆 tcp

```
[root@cqrk3399:/]# ./iperf3.2 -c 192.168.1.100 -i 10 -p 55555 -b 100M -t 60
Connecting to host 192.168.1.100, port 55555
[  5] local 192.168.1.202 port 40960 connected to 192.168.1.100 port 55555
init usec: 1358467333, usec: 933092
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-10.00  sec   113 MBytes  95.2 Mbits/sec    0    141 KBytes
[  5]  10.00-20.00  sec   113 MBytes  94.9 Mbits/sec    0    141 KBytes
[  5]  20.00-30.00  sec   113 MBytes  94.9 Mbits/sec    0    141 KBytes
[  5]  30.00-40.00  sec   113 MBytes  95.0 Mbits/sec    0    141 KBytes
[  5]  40.00-50.00  sec   113 MBytes  94.9 Mbits/sec    0    141 KBytes
[  5]  50.00-60.00  sec   113 MBytes  94.9 Mbits/sec    0    141 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-60.00  sec   679 MBytes  95.0 Mbits/sec    0             sender
[  5]   0.00-60.00  sec   679 MBytes  94.9 Mbits/sec                  receiver

iperf Done.
```

7.3.4 Iperf 测试问题与解决

如果在 iperf 性能测试中出现问题，比如吞吐量性能比较差、长时间测试中断之类的问题，建议如下的思路去查找问题。

先思考二个问题：

(1) 接收问题，还是发送问题？

(2) 应用层问题，还是 MAC 层问题，还是 PHY 层相关问题？

Iperf 在测试时，缺省 client 端会是数据发送方；如果带参数-R，那么 server 端是数据发送方。

Iperf 在测试时，缺省是 tcp 传输；使用参数-u 就用 udp 传输。

在用 tcp 测试时，数据其实是双向的，大量的发送数据与少量的接收数据（tcp 数据的 ack）。这种情况下，接收和发送方向有问题都会影响测试的性能与传输中断。

在用 udp 测试时，数据是单向的，不会有接收数据（电脑上其他应用产生的数据除外）。

✧ 注意：在使用 udp 测试时，通常缺省的参数性能不太好，要指定 buffer 长度，通常是 -l 65507，如下：

iperf3 -u -c 192.168.1.101 -p 5201 -i 1 -b 1000M -l 65507

那么如何来确认是接收还是发送问题呢？

a) 使用上述介绍的 iperf 各种测试。

b) 交换 iperf udp 测试的 client 和 server 端，对比测试结果来看是哪方的问题。

c) 结合 Mac 层的收发计数 counter 来看接收或者发送方向的计数是否有错包。

d) 通过收发端的抓包（比如通过 wireshark）来查找 tcp 或者 udp 协议的交互来判断。

e) 如果以上还不能确认，结合 phy 的计数（checker）功能来查看来自 rgmii 或者 utp/fiber 方向是否有错包。（参考前面关于 phy 层收发包计数的介绍）。

另外，确认 mac<->phy 间的数据接口和 phy 实际配置的数据接口是否一致？(eg. mac<->phy 间配置是 rgmii 接口，但是 phy 实际配置的接口是 mii 接口)

8. 关于 YT8521 电口/光口双模 phy 芯片

YT8521 支持电口与光口。在使用时的注意事项为：

1） 模式配置。YT8521 支持多种工作模式。通常是由硬件 power on strap pins 来配置。在软件调试阶段可以通过扩展寄存器 ext_reg_0xa001.b[2:0]来查看和配置工作模式。工作模式不对，会造成 phy 工作不正常。请参考 YT8521 datasheet。

例，配置 phy 工作模式为 combo-rgmii（即 fiber/utp<--->rgmii）：

```
# set 8521 mode utp/fiber<--->rgmii
# and fiber status check
# all values are of HEX format
set phy_addr 1
wr ext_reg 0xa001 8142      ;b[2:0]=2
wr mii_reg 0 9140           ;soft reset to phy for configuration taking effect
wr ext_reg 0xa000 2         ;switch to Fiber status
rd mii_reg 0x11             ;status register
```

2） Link 等状态的查看。当前是电口还是光口的 Link 状态，是由扩展寄存器 ext_reg_0xa000.b1 来决定的。这个一定要注意。如果光口 link up 但查的状态是电口的状态那结果就不对。反之也是。

例：读光口状态

```
wr ext_reg 0xa000 2
rd mii_reg 0x11
```

例：读电口状态

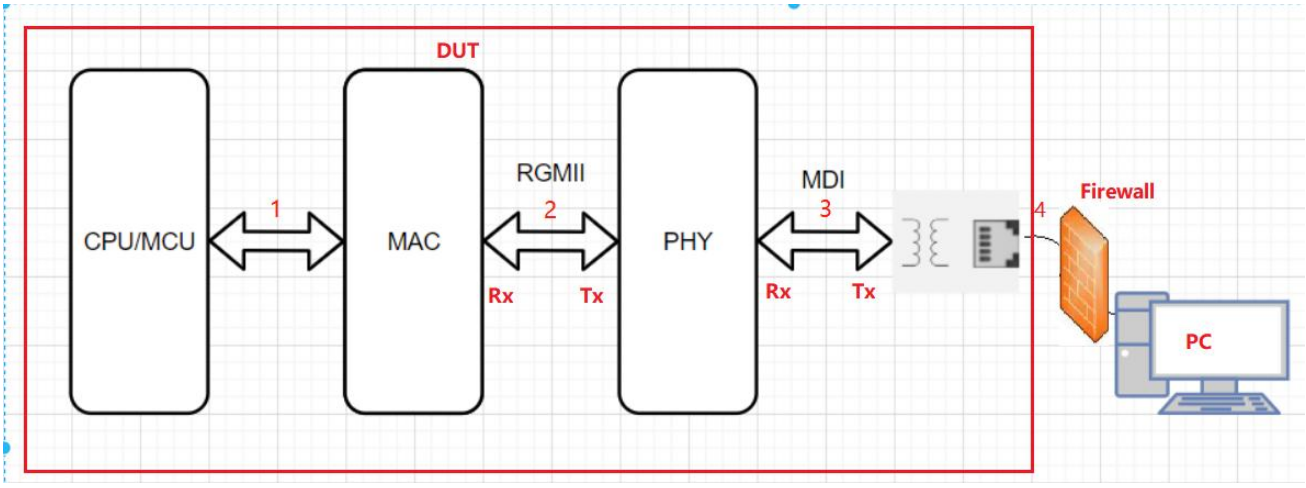
```
wr ext_reg 0xa000 0
rd mii_reg 0x11
```

9. 以太网数据通路问题调试思路

数据通路如果正常的话，就是 ping 是正常工作的，其他如 DHCP 等也都可以正常运行下去。

本章来讨论如果出现 ping 不通的情况，如何进行问题定位与排查。

本章以 yt8521S phy，chip mode 为 UTP<->RGMII 为例，拓扑图如下，说明排查问题思路。



9.1 单向不通还是双向不通

Ping 程序本身是双向的，但通常也会出现单向通的问题，比如设备和 PC 之间互 Ping。

1) 如果 PC ping->设备通（即拓扑图 4->3->2->1 数据通路通），反过来设备 Ping->PC 不通（即拓扑图 1->2->3->4 数据通路不通），可能的原因是：
   - PC 的防火墙（Firewall）是否打开？PC 打开防火墙时不会回应 ping。该状况下，关闭 PC 上防火墙（控制面板\系统和安全\Windows Defender 防火墙\自定义设置），如下图。

2) 如果设备 Ping->PC 通，反过来 PC ping->设备不通，可能的原因是：
   MAC 层的驱动在收发包机制上有问题，可能不能正确处理接收包，如收包中断没产生，不能收广播包等问题，需要检查 MAC 驱动。
3) 如果双向都 ping 不通，请按参看下面章节的专门讨论。

9.2 双向不通

如果双向不通，请检查：

1) PHY 是否能够正常 Link？如果不能正常 Link up，请检查下面关于不能正常 Link up 时检查的点。
2) PHY 能正常 link up，但 ping 还是不通时，就需要进行分段问题排查，就是说找到不通的点。可能的问题点包括：

   接收路径：
   - 对端 PC 到本设备 RJ45；
   - RJ45 到 PHY；
   - PHY 到 MAC（本例 rgmii 接口）；
   - MAC 设备到 CPU 系统（及上层应用）；

   发送路径：
   - CPU 系统（及上层应用）到 MAC 设备；
   - MAC 到 PHY（本例 rgmii 接口）；
   - PHY 到 RJ45；
   - RJ45 到对端 PC；

   下面分述不同路径下的检查点。

9.3 硬件排查思路

1) 检查电源: 输入 3.3v；输出 1.1v， RGMII 的电平配置是否如设计预期（1.8v/2.5v/3.3v），尤其 RGMII 电平配置为 3.3v 时的连接
2) clk、复位是否正常；尤其复位的时序、clk 频偏、Jitter 等会有影响

3) 检查 Rbias 电阻
4) 检查 MDIO 上拉电阻
5) 检查 Power Strap 配置：尤其模式配置，Rx/Tx delay 配置，phy 地址配置
6) 检查 LED、interrupt，wol 的配置
7) 检查 MDI 的连线，与参考设计做对比，确认连接正确；AC 电容耦合与变压器的连接，注意区别
8) 检查 RGMII 连接，尤其串阻大小，对地是否有并电容

9.4 软件排查思路

1) 可以通过文中“5.1.2 网上 phy.c 的方法”介绍的 phy 寄存器读写工具分别读 mii reg 0x0, 0x1, 0x4, 0x5, 0x9, 0xa, 0x11 值，确保其值符合期望，详细描述如下：
   - Reg 0x0 控制寄存器，确认其值是否为默认值 0x1140？bit12 1’b1 表示自协商 enable，当自协商位 enable 情况下，0x1140 其它 bit 位可以不关心。
     如果该寄存器读非 0x1140，则需要关注
     (1) bit11，该 bit 位为 1’b1 表示 power down，phy mii interface 将不工作，正常情况下该 bit 位为 1’b0。
     (2) bit10，该 bit 位为 1’b1 表示 isolate，phy mii interface 将不工作，正常情况下该 bit 位为 1’b0。
   - Reg 0x1 状态寄存器，主要关注
     (1) bit2 link status，该 bit 位为 1’b1 表示 link up，为 1’b0 表示 link down。需要注意该 bit 位是 Latching low，也即当该 bit 为 1’b0 时，需要再读一次该寄存器且以第二次读值的 bit2 来判断当前的 link 状态。
     (2) bit4 表示 remote fault，正常情况下默认值 1’b0，表示 link partner 状态正常，如果该 bit 位为 1’b1，表示 link partner 不正常，需要查明 link partner 出了什么问题？
     (3) bit5 表示自协商完成状态，当该 bit 位为 1’b0 则表示自协商未完成，为 1’b1 表示自协商完成。
   - Reg 0x4 是自协商时本端能力通告寄存器，主要关心 bit[8:5] 4 个 bit 位，默认 4b’1111，4 个 bit 位分别代表 100M full duplex，100M half duplex，10M full duplex，10M half duplex ability，1’b1 表示 support ability，1’b0 表示 not support ability
   - Reg 0x5 是自协商时 link partner 能力通告寄存器，同 reg 0x4，主要关心 bit[8:5] 4 个 bit 位，分别代表 100M full duplex，100M half duplex，10M full duplex，10M half duplex ability，1’b1 表示 support ability，1’b0 表示 not support ability
   - Reg 0x9 master/slave 控制寄存器，确认其是否为默认值 0x200？0x9 寄存器 bit9 1’b1 表示本端 phy 支持 1000M full duplex ability
   - Reg 0xa master/slave 状态寄存器，主要关注(1) bit13 Local Receiver Status，该 bit 位为 1’b1 表示 Local Receiver OK，该 bit 位为 1’b0 表示 Local Receiver not OK。(2) bit12 Remote Receiver，该 bit 位为 1’b1 表示 Remote Receiver OK，该 bit 位为 1’b0 表示 Local Receiver not OK
   - Reg 0x11 phy 特殊状态寄存器，该寄存器也记录了一份 phy 的 link status，speed，duplex 等状态信息。
     (1) bit10 表示 Link status real-time，该 bit 位为 1’b1 表示 link up，为 1’b0 表示 link down，只有当 bit10 1’b1，也即 phy link up 时，该寄存器其它 bit 位才有意义。
     (2) bit[15:14] 表示 speed，2’b10 表示 1000M speed, 2’b01 表示 100M speed, 2’b00 表示 10M speed。
     (3) bit13 表示 duplex，为 1’b1 表示 full duplex，为 1’b1 表示 half duplex。
2) 如果前述 1)中寄存器值都正常，且 reg 0x11 值显示是 link 在千兆，则可配置 mii reg 0x9=0x0，mii reg 0x0=0x9140，使 phy link 在百兆，再进行 DUT 和 PC 互 ping，如果 ping 正常无丢包，而 link 在 1000M

不正常，则可重点调整 Rx Tx delay 值。详细描述如下：

- Reg 0x9 写 0x0，其 bit9 1’b0 表示关闭 1000M full duplex ability（reg 0x9 默认值 0x200 bit9 1’b1 代表打开 1000M full duplex ability）
- Reg 0x0 写 0x9140，(1) bit12 自协商位，1’b1 开启自协商。(2) bit15 phy 软复位，1’b1 表示 phy 做软复位，软复位会将 reg 0x9 寄存器新配置的值生效，同时进行自协商。软复位完成后，bit15 会 self clearing，也即 bit15 会自动由 1’b1 变为 1’b0，理论上 phy 自协商完成后，reg 0x0 值为 0x1140
- 因默认打开 100M full duplex，10M full duplex ability，手动关闭了 1000M full duplex ability，phy 自协商结果理论上取双方最高 ability，也即 100M full duplex（可以读 reg 0x11，确认 bit15:14 2’b01）
- DUT 和 PC 互 ping，如果 100M ping 通，而 1000M ping 不通或有丢包，则重点调整 Tx Rx delay 值（根据 RGMII 标准，时钟的上升和下降沿采样，且时钟信号需要比数据信号 delay 1~2ns 来保证 setup/hold 时间）。此时再回到自协商 1000M 的场景下，在调整 Tx Rx delay 前，

  a. 先确认 Rx delay enable。确认 Rx delay enable 有两种途径：(1) 查看 phy 电路原理图，确认 pin 26 RXD0/RXDLY 上拉 (2) 读扩展寄存器 ext reg 0xa001，确认 bit8 Rxc_dly_en，bit8 1’b1 表示开启了 rx delay 功能，否则未开启 rx delay 功能，默认 bit8 1’b1 开启 rx delay 功能。

  b. 然后调整扩展寄存器 ext reg 0xa003，该寄存器 bit13:10 Rx_delay_sel 用作调整 rx delay，150ps per step。该寄存器 bit3:0 Tx_delay_sel 用作 1000M 场景下调整 tx dealy，该寄存器 bit7:4 Tx_delay_sel_fe 用作 100M 场景下调整 tx dealy。

3) 如果前述 1)中寄存器值 0x11 的值显示没正常 link，则重点排查 MDI 连线（参考本文 9.3 硬件排查思路 7）
4) 如果本端 mac 能够自发包，可以做 loopback 验证（参考应用说明 8.9 Loopback（回环）模式）。先做 internal loopback 确认 rgmii 收发是否正常（rgmii 收发是经常出问题的地方，需重点关注）？然后再做 external loopbak 以及 remote loopback 验证来定位数据通路问题出在哪里？
   - 如果 internal loop 不正常，则需确认是 rgmii tx 还是 rgmii rx 方向问题，此处可以结合 phy 自带 checker tool 一起来验证（checker 可以验证是否收到 rgmii tx 方向数据包），确认了是哪个方向问题，则重点检查这个方向上信号线连接，各条信号线信号，delay 等。
   - 如果 internal loopback 正常，基本能够说明 rgmii 上 rx，tx 通路正常。接着做 external loopback，如果 external loopback 不正常，则需要确认是 mdi tx 还是 mdi rx 方向问题，此处同 internal loop，在验证时结合 phy 自带 checker tool 一起来验证（checker 可以验证是否收到 mdi rx 方向数据包），确认了是哪个方向问题，则重点检查这个方向上信号线连接，各条信号线信号等。
5) 如果本端 mac 不能自发包，在设备和 PC 互 ping 的情况下，通过 phy 自带 checker tool(参考应用说明 6.7 包生成器和收发包统计)，可以验证 mdi rx 方向是否正常，rgmii rx 方向是否正常？
   - checker 脚本 init_8521_cnt.sh/cnt_8521.sh 制作参考本文 6.2.1 YT8511 的计数。由于 YT8521S 的 checker 相关的寄存器同 YT8511，所以 YT8511 的 checker 脚本适用于 YT8521S。
   - init_8521_cnt.sh 脚本命令行中运行一次即可。
   - cnt_8521.sh 脚本在 DUT 和 PC 互 ping 过程中可以运行多次（注意：该脚本中的所有寄存器都是读清的）。
   - cnt_8521.sh 脚本的输出信息中，一般会发现一些当前问题的线索，比如：

     a. mdi rx 方向长度为 64-1518Byte 的数据包为 0，表示 mdi rx 方向没有收到数据包，可以从 mdi 4 对线硬件电路上连接是否正确排查

     b. mii rx 方向长度为 64-1518Byte 的数据包为 0，表示 phy rgmii（对应于 rgmii tx 方向）上没有收到数据包，可以重点检查 rgmii 路径上信号线连接，各条信号线信号，delay 等

     c. mdi rx/mii rx 上有统计 crc error，或者有统计 no sfd error，该现象重点检查 clk

6) 建议做 4) 5)同时，rgmii 信号线接上示波器观察信号是否正常？

10. Phy driver 中集成的其它 debug 方法

```
root@bpi-r3:/sys/kernel/debug/yt# ls -lt
total 0
--w--w--w- 1 root root 0 Sep    9 19:19 checker
--w--w--w- 1 root root 0 Sep    9 19:09 phyreg
--w--w--w- 1 root root 0 Jan    1    1970 csd
--w--w--w- 1 root root 0 Jan    1    1970 drive_strength
--w--w--w- 1 root root 0 Jan    1    1970 dump
--w--w--w- 1 root root 0 Jan    1    1970 generator
--w--w--w- 1 root root 0 Jan    1    1970 loopback
-r--r--r-- 1 root root 0 Jan    1    1970 phydrv_ver
--w--w--w- 1 root root 0 Jan    1    1970 snr
--w--w--w- 1 root root 0 Jan    1    1970 template
root@bpi-r3:/sys/kernel/debug/yt#
```

10.1 phy driver version 查看

```
root@bpi-r3:/sys/kernel/debug/yt# cat phydrv_ver
Driver Version: 2.2.43619
root@bpi-r3:/sys/kernel/debug/yt#
```

10.2 phy registers dump

```
root@bpi-r3:/sys/kernel/debug/yt# echo > dump
[ 2546.518135] usage:
[ 2546.520165]    reg dump: echo ethx reg_space(utp|sgmii|qsgmii) > /sys/kernel/debug/yt/dump
root@bpi-r3:/sys/kernel/debug/yt# echo ethx utp > dump
[ 2562.438110] utp reg space:
[ 2562.441009] mii reg 0x0:        0x1040
[ 2562.444254] mii reg 0x1:        0x796d
[ 2562.447497] mii reg 0x2:        0x4f51
[ 2562.450768] mii reg 0x3:        0xea19
[ 2562.454016] mii reg 0x4:        0x11e1
[ 2562.457260] mii reg 0x5:        0xd1e1
[ 2562.460527] mii reg 0x6:        0x006f
[ 2562.463774] mii reg 0x7:        0x2001
[ 2562.467016] mii reg 0x8:        0x0000
[ 2562.470273] mii reg 0x9:        0x0200
[ 2562.473517] mii reg 0xa:        0x7800
[ 2562.476760] mii reg 0xb:        0x0000
[ 2562.480015] mii reg 0xc:        0x0000
[ 2562.483262] mii reg 0xd:        0x4007
[ 2562.486505] mii reg 0xe:        0x0000
[ 2562.489766] mii reg 0xf:        0x2000
[ 2562.493013] mii reg 0x10:       0x0062
[ 2562.496342] mii reg 0x11:       0x3e00
[ 2562.499687] mii reg 0x12:       0x0000
```

```
[ 2562.503018] mii reg 0x13:       0x7404
[ 2562.506347] mii reg 0x14:       0x082c
[ 2562.509690] mii reg 0x15:       0x0000
[ 2562.513021] mii reg 0x16:       0x0000
[ 2562.516351] mii reg 0x17:       0x0000
[ 2562.519691] mii reg 0x18:       0x0000
[ 2562.523021] mii reg 0x19:       0x0000
[ 2562.526351] mii reg 0x1a:       0x0000
[ 2562.529689] mii reg 0x1b:       0x0000
[ 2562.533019] mii reg 0x1c:       0x0000
[ 2562.536351] mii reg 0x1d:       0x0000
[ 2562.539695] mii reg 0x1e:       0xa000
[ 2562.543028] mii reg 0x1f:       0x0000
[ 2562.546387] ext reg 0xa001:    0x8000
[ 2562.549932] ext reg 0xa003:    0xa080
root@bpi-r3:/sys/kernel/debug/yt# echo ethx sgmii > dump
[ 2574.108100] sgmii reg space:
[ 2574.111228] mii reg 0x0:        0x0140
[ 2574.114474] mii reg 0x1:        0x41cd
[ 2574.117718] mii reg 0x2:        0x4f51
[ 2574.120983] mii reg 0x3:        0xea19
[ 2574.124230] mii reg 0x4:        0x0020
[ 2574.127473] mii reg 0x5:        0x0000
[ 2574.130731] mii reg 0x6:        0x0000
[ 2574.133977] mii reg 0x7:        0x0000
[ 2574.137222] mii reg 0x8:        0x0000
[ 2574.140481] mii reg 0xf:        0x8000
[ 2574.143727] mii reg 0x11:       0x2609
[ 2574.147057] mii reg 0x14:       0x7028
[ 2574.150401] mii reg 0x15:       0x0000
[ 2574.153732] mii reg 0x16:       0x0000
[ 2574.157063] mii reg 0x17:       0x0000
root@bpi-r3:/sys/kernel/debug/yt#
```

10.3 csd 检测

```
root@bpi-r3:/sys/kernel/debug/yt# echo > csd
[ 2980.458115] usage:
[ 2980.460145]    csd: echo ethx > /sys/kernel/debug/yt/csd
root@bpi-r3:/sys/kernel/debug/yt# echo ethx > csd
[ 2993.378223] channel 3 is normal
[ 2993.381381] channel 2 is normal
[ 2993.384510] channel 1 is normal
[ 2993.387638] channel 0 is normal
root@bpi-r3:/sys/kernel/debug/yt# echo ethx > csd
[ 3010.308154] The intra pair damage(open) location of channel 3: 0(cm).
```

[3010.314664] The intra pair damage(open) location of channel 2: 82(cm).
[3010.321253] The intra pair damage(open) location of channel 1: 0(cm).
[3010.327737] The intra pair damage(open) location of channel 0: 0(cm).
root@bpi-r3:/sys/kernel/debug/yt#

10.4 drive_strength 调整

root@bpi-r3:/sys/kernel/debug/yt# echo > drive_strength
[3095.088110] usage:
[3095.090142] rgmii drive strength: echo ethx > /sys/kernel/debug/yt/dump
root@bpi-r3:/sys/kernel/debug/yt#

10.5 Generator

root@bpi-r3:/sys/kernel/debug/yt# echo > generator
[3130.028105] usage:
[3130.030136] generator: echo ethx utp(1)|sgmii(2)> /sys/kernel/debug/yt/generator
root@bpi-r3:/sys/kernel/debug/yt#

10.6 Loopback

root@bpi-r3:/sys/kernel/debug/yt# echo > loopback
[3163.438106] usage:
[3163.440138] loopback: echo ethx utp(1)|sgmii(2) internal(1)|external(2)|remote(3) 10M(1)|100M(2)|1000M(3)|2500M(4) > /sys/kernel/debug/yt/loopback
root@bpi-r3:/sys/kernel/debug/yt#

10.7 Snr

root@bpi-r3:/sys/kernel/debug/yt# echo > snr
[3212.208097] usage:
[3212.210129] snr: echo ethx > /sys/kernel/debug/yt/snr
root@bpi-r3:/sys/kernel/debug/yt#

10.8 template

root@bpi-r3:/sys/kernel/debug/yt# echo > template
[3231.498100] usage:
[3231.498100] template(10M): test mode1(1) - packet with all ones, 10MHz sine wave, for harmonic test
[3231.498100] template(10M): test mode1(2) - pseudo random, for TP_idle/Jitter/Different voltage test
[3231.498100] template(10M): test mode1(3) - normal link pulse only
[3231.498100] template(10M): test mode1(4) - 5MHz sine wave
[3231.498100] template(10M): test mode1(5) - Normal mode
[3231.498100] template(10M): echo ethx 10M(1) test mode1(1)|test mode2(2)|test mode3(3)|test mode4(4)|test mode5(5) > /sys/kernel/debug/yt/template
[3231.548791] template(100M): echo ethx 100M(2) > /sys/kernel/debug/yt/template
[3231.555997] template(1000M): test mode1(1) - Transmit waveform test
[3231.555997] template(1000M): test mode1(2) - Transmit Jitter test (master mode)
[3231.555997] template(1000M): test mode1(3) - Transmit Jitter test (slave mode)
[3231.555997] template(1000M): test mode1(4) - Transmit distortion test
[3231.555997] template(1000M): echo ethx 1000M(3) test mode1(1)|test mode2(2)|test mode3(3)|test mode4(4) > /sys/kernel/debug/yt/template
[3231.595301] template(2500M): echo ethx 2500M(4) test mode1(1)|test mode2(2)|test mode3(3)|test mode4(4)|test mode5(5)|test mode6(6)|test mode7(7)

[3231.595301] [dual tone1(1)|dual tone2(2)|dual tone3(3)|dual tone4(4)|dual tone5(5)] > /sys/kernel/debug/yt/template
[3231.618834] echo ethx 10M(1)|100M(2)|1000M(3)|2500M(4) [test mode1(1)|test mode2(2)|test mode3(3)|test mode4(4)|test mode5(5)|test mode6(6)|test mode7(7)]
[3231.618834] [dual tone1(1)|dual tone2(2)|dual tone3(3)|dual tone4(4)|dual tone5(5)] > /sys/kernel/debug/yt/template
root@bpi-r3:/sys/kernel/debug/yt#

11. Ethtool interface

11.1 Phy downgrade config

root@bpi-r3:~# ethtool --get-phy-tunable ethx downshift
Downshift count: 5
root@bpi-r3:~# ethtool --set-phy-tunable ethx downshift off
root@bpi-r3:~# ethtool --get-phy-tunable ethx downshift
Downshift disabled
root@bpi-r3:~# ethtool --set-phy-tunable ethx downshift on count N

11.2 Phy fast link down config

root@bpi-r3:~# ethtool --get-phy-tunable ethx fast-link-down
Fast Link Down disabled
root@bpi-r3:~# ethtool --set-phy-tunable ethx fast-link-down off
root@bpi-r3:~# ethtool --set-phy-tunable ethx fast-link-down on msecs N

11.3 Phy 收发包统计

root@bpi-r3:~# ethtool --phy-statistics ethx
PHY statistics:
    UTP ib valid(64B~1518B): 0
    UTP ib valid(>1518B): 0
    UTP ib valid(<64B): 0
    UTP ib crc err(64B~1518B): 0
    UTP ib crc err(>1518B): 0
    UTP ib crc err(<64B): 0
    UTP ib no sfd: 0
    UTP ob valid(64B~1518B): 0
    UTP ob valid(>1518B): 0
    UTP ob valid(<64B): 0
    UTP ob crc err(64B~1518B): 0
    UTP ob crc err(>1518B): 0
    UTP ob crc err(<64B): 0
    UTP ob no sfd: 0
    SERDES ib valid(64B~1518B): 0
    SERDES ib valid(>1518B): 0
    SERDES ib valid(<64B): 0
    SERDES ib crc err(64B~1518B): 0
    SERDES ib crc err(>1518B): 0
    SERDES ib crc err(<64B): 0
    SERDES ib no sfd: 0

```
        SERDES ob valid(64B~1518B): 0
        SERDES ob valid(>1518B): 0
        SERDES ob valid(<64B): 0
        SERDES ob crc err(64B~1518B): 0
        SERDES ob crc err(>1518B): 0
        SERDES ob crc err(<64B): 0
        SERDES ob no sfd: 0
root@bpi-r3:~#
```