

Motorcomm Inc

# 裕太以太网 Phy 驱动使用手册

The guide to use of the software

Ver 4.21.2022-09:34

# User Guide

User Guide.....	1
1. 内核空间驱动的编译加载说明.....	4
1.1 Linux 系统.....	4
1.1.1 解压驱动代码源文件.....	4
1.1.2 直接编译在 linux 内核中进行加载的操作步骤.....	4
1.1.2.1 重新 Make 目标板 Linux 系统.....	4
1.1.3 Driver module 方式加载操作说明.....	5
1.1.4 启动目标板 Linux 系统.....	5
2. 内核空间驱动的适配说明.....	6
2.1 通用说明.....	6
2.1.1 UBoot 下的适配说明.....	6
2.2 配置 YT8511 给 MAC 提供时钟输入.....	6
2.3 配置 YT8512 与 sunxi GMAC 适配, mii 接口。.....	7
2.4 配置 YT8521 上电输出 PHY Rx CLK.....	7
2.5 系统内多个 YT8521 的支持.....	8
2.6 关于多口 PHY YT8614.....	8
2.7 关于在 fiber 模式下几个问题。.....	8
2.8 关于对 WOL 功能支持的说明.....	8
2.8.1 wol 功能的简介与系统连接.....	8
2.8.2 支持 MAC 层设备实现 WOL.....	9
2.8.3 支持 PHY 设备实现 WOL.....	9
2.8.3.1 Linux 内核 PHY 层 WOL 相关回调函数.....	9
2.8.3.2 自定义实现 PHY 层 WOL 功能.....	10
3. 用户空间驱动的使用.....	11
3.1 用户空间下驱动包.....	11
3.2 用户空间下驱动 API 与移植说明.....	11
3.2.1 数据结构.....	11
3.2.1.1 phy_info_s.....	11
3.2.1.2 phy_data_s.....	12
3.2.2 API 返回值定义.....	12
3.2.3 多芯片支持与端口号(lport)的定义.....	12
3.2.4 寄存器访问函数的移植.....	13
3.2.4.1 局部函数 yt8614_app_get_phy_info().....	13
3.3 用户空间驱动的编译与开发.....	13
3.3.1 用户空间 Non-cli 版本的应用 demo.....	14
3.3.1.1 程序编译.....	15
3.4 用户空间下 CLI 版本的驱动应用 demo.....	15
3.4.1 Cygwin 环境下评估系统的硬件连接.....	15
3.4.2 Cygwin 环境安装.....	16
3.4.3 Cygwin 环境下运行 yt861x.exe.....	19

3.4.4	yt861x.exe 的功能简介.....	20
3.4.5	CLI 命令简介.....	20
3.4.5.1	打开/关闭 mdio-com 设备.....	20
3.4.5.2	访问 phy 寄存器.....	21
3.4.5.2.1	端口 phy 地址.....	21
3.4.5.2.2	Utp mii 寄存器.....	21
3.4.5.2.3	Utp ext 寄存器.....	22
3.4.5.2.4	Utp mmd 寄存器.....	22
3.4.5.2.5	Fiber/Sgmii mii 寄存器.....	22
3.4.5.2.6	Fiber/Sgmii ext 寄存器.....	22
3.4.5.2.7	Qsgmii mii 寄存器.....	22
3.4.5.2.8	Qsgmii ext 寄存器.....	22
3.4.5.3	调用驱动程序 API.....	22
3.4.5.4	重复执行一条命令.....	23
3.4.5.5	后台进程监视 phy 工作状态.....	23
3.4.5.6	执行命令脚本.....	24
4.	用户空间驱动 API 说明.....	24
4.1	YT8614 驱动 API 说明.....	24
4.1.1	寄存器读写 API.....	24
4.1.2	s32 yt8614_phy_soft_reset(u32 lport);.....	24
4.1.3	s32 yt8614_phy_init(u32 lport);.....	24
4.1.4	s32 yt8614_fiber_enable(u32 lport, BOOL enable);.....	24
4.1.5	s32 yt8614_otp_enable(u32 lport, BOOL enable);.....	25
4.1.6	s32 yt8614_fiber_unidirection_set(u32 lport, int speed, BOOL enable);.....	25
4.1.7	s32 yt8614_fiber_autosensing_set(u32 lport, BOOL enable);.....	25
4.1.8	s32 yt8614_fiber_speed_set(u32 lport, int fiber_speed);.....	25
4.1.9	s32 yt8614_qsgmii_autoneg_set(u32 lport, BOOL enable);.....	25
4.1.10	s32 yt8614_sgmii_autoneg_set(u32 lport, BOOL enable);.....	25
4.1.11	s32 yt8614_qsgmii_sgmii_link_status_get(u32 lport, BOOL *enable, BOOL if_qsgmii); 25	
4.1.12	int yt8614_combo_media_priority_set(u32 lport, int fiber);.....	25
4.1.13	int yt8614_combo_media_priority_get(u32 lport, int *fiber);.....	25
4.1.14	s32 yt8614_otp_autoneg_set(u32 lport, BOOL enable);.....	26
4.1.15	s32 yt8614_otp_autoneg_get(u32 lport, BOOL *enable);.....	26
4.1.16	s32 yt8614_otp_autoneg_ability_set(u32 lport, unsigned int cap_mask);.....	26
4.1.17	s32 yt8614_otp_autoneg_ability_get(u32 lport, unsigned int *cap_mask);.....	26
4.1.18	s32 yt8614_otp_force_duplex_set(u32 lport, BOOL full);.....	26
4.1.19	s32 yt8614_otp_force_duplex_get(u32 lport, BOOL *full);.....	26
4.1.20	s32 yt8614_otp_force_speed_set(u32 lport, unsigned int speed);.....	26
4.1.21	s32 yt8614_otp_force_speed_get(u32 lport, unsigned int *speed);.....	26
4.1.22	int yt8614_autoneg_done_get(u32 lport, int speed, int *aneg);.....	27
4.1.23	int yt8614_media_status_get(u32 lport, int *speed, int *duplex, int *ret_link, int *media); 27	



## 1. 内核空间驱动的编译加载说明

## 1.1 Linux 系统

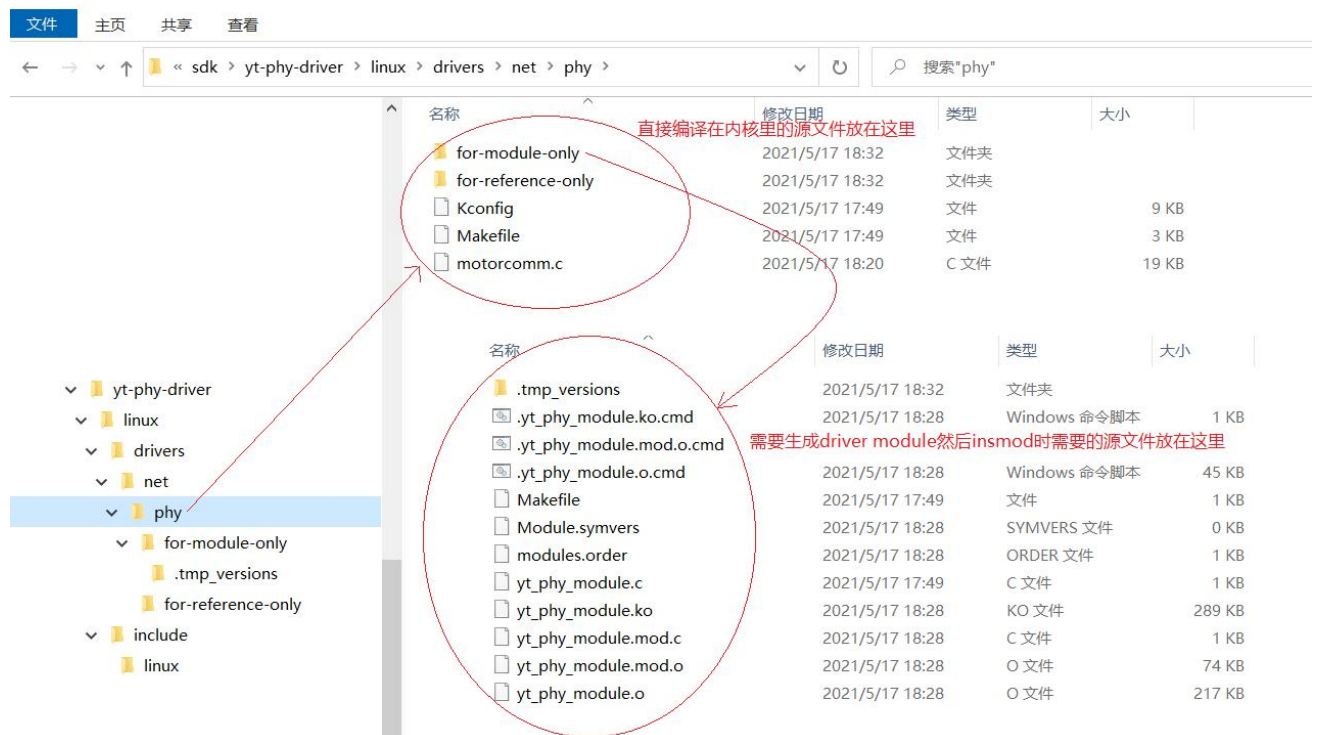
### 1.1.1 解压驱动代码源文件

Motorcomm 的驱动被打包成 rar 包:

yt-phy-driver	2021/5/17 18:32	文件夹	
yt-phy-driver.rar	2021/5/19 7:25	WinRAR 压缩文件	192 KB

驱动适用于 linux 多个内核版本 (linux3.x/4.x/5.x)。

文件的组织结构如下:



### 1.1.2 直接编译在 linux 内核中进行加载的操作步骤

解压后按照 Linux 的目录结构展开。用户需要将文件按相应的目录 copy 到目标板 Linux 系统开发环境中。比如我的目标板开发环境 Linux 是在如下目录：

~/motocomm/kernel/linux-rt-4.19.94

就将 motorcomm.c 和头文件 motorcomm phy.h(新版本中不需要此文件了)copy 到各自的目录:

```
cp motorcomm.c ~/motocomm/kernel/linux-rt-4.19.94/drivers/net/phy/.
cp motorcomm_phy.h ~/motocomm/kernel/linux-rt-4.19.94/include/linux/.
```

注意 Kconfig 和 Makefile 不能直接 copy 过去而要与当前系统里的这二个文件做合并:

Kconfig 是合并下面的部分:

```
config MOTORCOMM_PHY
    tristate "Motorcomm PHYs"
    ---help---
    Supports the YT8010, YT8510, YT8511, YT8512 PHYs.
```

Makefile 合并下面的部分:

```
obj-$(CONFIG_MOTORCOMM_PHY) += motorcomm.o
```

### 1.1.2.1 重新 Make 目标板 Linux 系统

使用用户目标板 Linux 系统的 Make 命令生成目标板 Linux Image 文件。比如我的系统里是这样：

```
yzhang@ubuntu:~/motocomm/kernel/linux-rt-4.19.94$ make zImage
```

Make 完成后应该有如下文件：

```
./drivers/net/phy/motorcomm.o
```

如果没有请重新检查以上步骤。

### 1.1.3 Driver module 方式加载操作说明

使用 driver module 方式的优点是不需要更新内核 kernel。这对于没有内核源代码的情况下适用。但这种方法的前提是系统支持本机编译开发环境（比如可以直接执行 **gcc, make**）。这种方式适用于很多桌面的 linux 操作系统，比如麒麟系统，ubuntu，CentOS 等。

在做以下步骤时请确认上述的本机开发环境。

#### 1. 编译生成裕太 phy driver module

```
cd ~
cp -r /media/kylin/1CE843CDE843A43C/yt-phy-driver .
cd yt-phy-driver/linux/drivers/net/phy/for-module-only/
make clean && make
make -C /lib/modules/`uname -r`/build M=/home/kylin/yzhang/yt-phy-driver/linux/drivers/net/phy/for-module-only clean
make[1]: Entering directory '/usr/src/kylin-headers-4.4.131-20210514.kylin-test-generic'
CLEAN /home/kylin/yzhang/yt-phy-driver/linux/drivers/net/phy/for-module-only/.tmp_versions
CLEAN /home/kylin/yzhang/yt-phy-driver/linux/drivers/net/phy/for-module-only/Module.symvers
make[1]: Leaving directory '/usr/src/kylin-headers-4.4.131-20210514.kylin-test-generic'
make -C /lib/modules/`uname -r`/build M=/home/kylin/yzhang/yt-phy-driver/linux/drivers/net/phy/for-module-only modules
make[1]: Entering directory '/usr/src/kylin-headers-4.4.131-20210514.kylin-test-generic'
CC [M] /home/kylin/yzhang/yt-phy-driver/linux/drivers/net/phy/for-module-only/yt_phy_module.o
Building modules, stage 2.
MODPOST 1 modules
CC /home/kylin/yzhang/yt-phy-driver/linux/drivers/net/phy/for-module-only/yt_phy_module.mod.o
LD [M] /home/kylin/yzhang/yt-phy-driver/linux/drivers/net/phy/for-module-only/yt_phy_module.ko
make[1]: Leaving directory '/usr/src/kylin-headers-4.4.131-20210514.kylin-test-generic'
```

#### 2. 确认系统内核版本。这个版本号就是 copy 驱动 module 的目标目录。

```
uname -sr
Linux 4.4.131-20210514.kylin-test.kylin.desktop-generic
```

#### 3. 加载 phy driver module。

```
ls /lib/modules
4.4.131-20200710.kylin.desktop-generic
4.4.131-20210514.kylin.desktop-generic
4.4.131-20210514.kylin-test.kylin.desktop-generic
sudo cp yt_phy_module.ko
```

```
/lib/modules/4.4.131-20210514.kylin-test.kylin.desktop-generic/kernel/drivers/net/phy/
```

```
cd /lib/modules
ls
4.4.131-20200710.kylin.desktop-generic
4.4.131-20210514.kylin.desktop-generic
4.4.131-20210514.kylin-test.kylin.desktop-generic
cd /lib/modules/4.4.131-20210514.kylin-test.kylin.desktop-generic/
sudo depmod -a
```

### 1.1.4 启动目标板 Linux 系统

不管用上述哪种加载方式，目标板重新启动后应该看到下面的驱动加载：

```
root@am335x-evm:~# ls /sys/bus/mdio_bus/drivers
```

```

root@an335x-evm:~# ls /sys/bus/ndio_bus/drivers
Marvell 88E1110      Micrel KSZ8041KMLI      RTL8211DN Gigabit Ethernet  YT8010 Automotive Ethernet
Marvell 88E1121R      Micrel KSZ8051          RTL8211E Gigabit Ethernet  YT8510 100/10Mb Ethernet
Marvell 88E1145      Micrel KSZ8061          RTL8211F Gigabit Ethernet  YT8511 Gigabit Ethernet
Marvell 88E1149R      Micrel KSZ8081 or KSZ8091 RTL8366RB Gigabit Ethernet  YT8512 Ethernet
Marvell 88E1240      Micrel KSZ8795          RTL9010A Gigabit Ethernet  YT8512B Ethernet
Marvell 88E1318S      Micrel KSZ886X Switch   Rockchip integrated EPHY    YT8521 Ethernet
Marvell 88E1510      Micrel KSZ8873ML Switch  SMSC LAN8187

```

Phy 驱动会被 linux 自动加载。

然后在目标板 Linux 系统里的 ifconfig 应该可以看到这个网络设备。

比如：

```

kylin@kylin:~$ ifconfig
enaphyt410 Link encap:以太网 硬件地址 98:0e:24:6a:03:54
UP BROADCAST MULTICAST MTU:1500 跃点数:1
 接收数据包:0 错误:0 丢弃:0 过载:0 帧数:0
 发送数据包:0 错误:0 丢弃:0 过载:0 载波:0
 碰撞:0 发送队列长度:1000
 接收字节:0 (0.0 B) 发送字节:42 (42.0 B)
 中断:12

```

## 2. 内核空间驱动的适配说明

### 2.1 通用说明

在 linux 系统或者 Uboot 系统中缺省都提供了通用的 phy 驱动。在一般情况下（指只使用电口且没有下述的各种特殊需求），使用这些通用驱动也可以使 phy 正常工作。

本章后面列出一些情况必须使用 YT 驱动的情况。

#### 2.1.1 UBoot 下的适配说明

Uboot 下使用，前提是要对 uboot 整个 MAC、PHY 工作流程有一定的熟悉。

如前述，从一般 phy 功能来说，使用 uboot 本身的 Phy 配置流程就可以了。

下述的一些特殊情况，比如针对 8521 来说，要使用光口模式或者 combo 模式，就可把我们驱动里初始化 8521 和检查光/电口 link status 的流程加进来。具体就是：

static int yt8521\_config\_init(struct phy\_device \*phydev)和

static int yt8521\_read\_status(struct phy\_device \*phydev)

具体 8521 光口的配置以及其他 PHY 的特殊情况，请参考下面章节的说明。

### 2.2 配置 YT8511 给 MAC 提供时钟输入

首先与硬件确认系统的 GMAC 需要 phy 提供时钟输入。然后做如下改动：

1. motorcomm.c 文件中，修改：

```
#define GMAC_CLOCK_INPUT_NEEDED 1
```

2. 参考./for-reference-only/phy\_device.c 中下面 **红色斜粗**的代码配置 YT8511 输出相应的时钟：

```

/* yzhang added for YT8511 phy 125m clock out */
extern int yt8511_config_out_125m(struct mii_bus *bus, int phy_id);
extern int yt8511_config_dis_txdelay(struct mii_bus *bus, int phy_id);
...
struct phy_device *get_phy_device(struct mii_bus *bus, int addr, bool is_c45)
{
    struct phy_c45_device_ids c45_ids = {0};
    u32 phy_id = 0;
    int r;

    r = get_phy_id(bus, addr, &phy_id, is_c45, &c45_ids);
    if (r)
        return ERR_PTR(r);

    /* If the phy_id is mostly Fs, there is no device there */
    if ((phy_id & 0x1fffffff) == 0x1fffffff)

```

```

return NULL;

printk (KERN_INFO "yzhang..read phyaddr=%d, phyid=%08x\n",addr, phy_id);
if(0x10a == phy_id)
{
    #if 0
        r = yt8511_config_dis_txdelay(bus, addr);
        printk (KERN_INFO "yzhang..8511 dis txdelay, reg=%#04x\n",bus->read(bus,addr,0x1f)/*double check as delay*/);
        if (r<0)
        {
            printk (KERN_INFO "yzhang..failed to dis txdelay, ret=%d\n",r);
        }
    #endif

    #if 1
        printk (KERN_INFO "yzhang..get YT8511, abt to set 125m clk out, phyaddr=%d, phyid=%08x\n",addr, phy_id);
        r = yt8511_config_out_125m(bus, addr);
        printk (KERN_INFO "yzhang..8511 set 125m clk out, reg=%#04x\n",bus->read(bus,addr,0x1f)/*double check as delay*/);
        if (r<0)
        {
            printk (KERN_INFO "yzhang..failed to set 125m clk out, ret=%d\n",r);
        }
    #endif
}

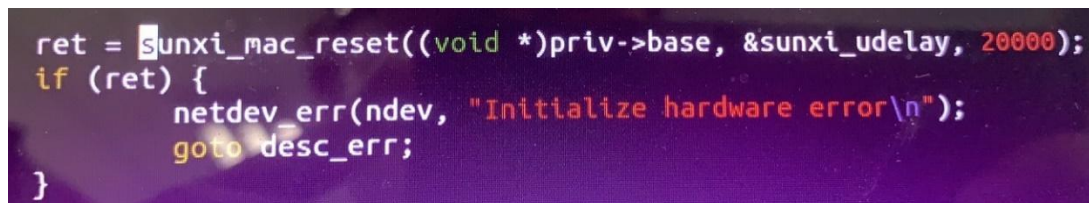
return phy_device_create(bus, addr, phy_id, is_c45, &c45_ids);
}

```

## 2.3 配置 YT8512 与 sunxi GMAC 适配，mii 接口。

出错现象：

[ 0.727023] WARNING: Get ephy clock is failed



```

ret = sunxi_mac_reset((void *)priv->base, &sunxi_udelay, 20000);
if (ret) {
    netdev_err(ndev, "Initialize hardware error\n");
    goto desc_err;
}

```

解决：

Sunxi GMAC 不需要 8512 的时钟

```

static int yt8512_config_init(struct phy_device *phydev)
{
    ...
    /* 把这里注释掉试下 */
    ret = yt8512_clk_init(phydev);
    if (ret < 0)
        return ret;
    /*
    ...
}

```

## 2.4 配置 YT8521 上电输出 PHY Rx CLK

请与硬件确认系统 GMAC 在做 soft reset 时是否需要检测 phy rx clk 的存在。比如 Phytium stmmac 在初始化时进行 MAC soft reset，这个过程中要检查 phy 设备的 rx clk 是否正常。YT8521 在插网线时就关闭 rx clk 的输出，导致 MAC soft reset 不能完成从而 MAC 初始化的过程失败，网络设备建立出错。具体错误信息在 linux dmesg 中如下：



```

7.953232] libphy: stmmac ID: 0x10, Synopsys ID: 0x36
7.661436] stmmaceth PHYT0004:00 enaphyt4i0: renamed from eth0
7.687490] stmmaceth PHYT0004:01 enaphyt4i1: renamed from eth1
2.155529] stmmac_hw_setup: DMA engine initialization failed
2.155535] stmmac_open: Hw setup failed
4.159419] stmmaceth PHYT0004:00 enaphyt4i0: Link is Up - 100Mbps/Full - flow control rx/tx
@kylin:~$
@kylin:~$
@kylin:~$
@kylin:~$

```

在这种情况下，就需要把我们驱动里初始化 8521 的流程加进来。具体就是：

```
static int yt8521_config_init(struct phy_device *phydev)
```

## 2.5 系统内多个 YT8521 的支持

目前驱动已经实现 8521 配置模式的自动检测功能。也实现了一个板上支持多个 yt8521 芯片。目前驱动里最多支持 8 个 yt8521 phy 芯片。通过以下宏可以修改支持的数目：

```
#define YTPHY_BOARD_MAX_NUM_OF_CHIP_8521 8
```

## 2.6 关于多口 PHY YT8614

多口 phy 驱动支持二种方式：

Kernel 方式：这种方式同其他 phy 的工作方式，可以是 kernel 或者 Module 方式。

用户空间方式：完全在用户模式下编译运行。参见二个文件：yt8614-phy.c 和 yt8614-phy.h 以及后面章节的关于用户空间驱动的专门介绍。

目前驱动已经支持 yt8614 工作模式的自动检测功能。

多口 phy 在 kernel 模式下运行时，相当于多个单口 phy。目前驱动已经支持多个 yt8614。支持的数目在下面的宏里定义，用户可以根据需要修改支持的数目。

```
#define YTPHY_BOARD_MAX_NUM_OF_CHIP_8614 4
```

可见，缺省时驱动可以支持 4x4 共 16 个 8614 phy 端口。

## 2.7 关于在 fiber 模式下几个问题。

YT8521/YT853/YT8614 都支持 Fiber 模式。

1. Fiber 模式下，link 在 100M 时， auto negotiation done 位不置 1， 802.3 标准里是这样定的。但有时需要统一处理时，就要提供这个值。函数 yt8xxx\_driver\_aneg\_done（）相应地完成这个功能。
2. 同样，在 fiber 时，没有 10M 这个速率，所以在查询 phy 状态时，要修正。这个功能统一在函数 yt8521\_adjust\_status（）里完成。

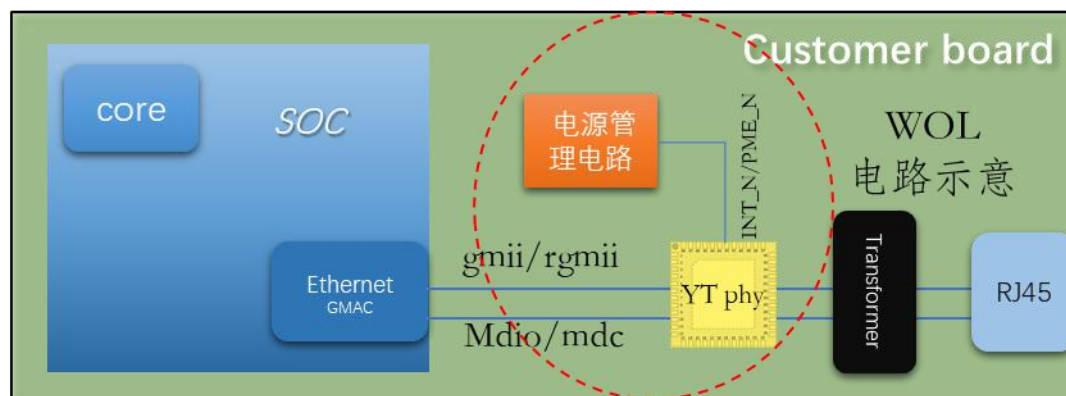
## 2.8 关于对 WOL 功能支持的说明

裕太 phy 芯片驱动支持网络唤醒（WOL）功能。

### 2.8.1 wol 功能的简介与系统连接

网络唤醒 WOL(Wake On LAN) 是指从局域网或者远程网络唤醒电脑。技术原理就是主机在睡眠状态下网络设备(MAC 层设备如网卡，PHY 层设备如以太网 phy 芯片)收到指定的报文后产生中断从而使休眠或关机状态转成开机状态。

WOL 功能需要硬件电路的支持，请与硬件工程师确认。下图示意了支持 WOL 功能的系统硬件连接：



现在 WOL 功能是很多网络设备的标准配置。所以就实现的方式来说，可以分为：

- MAC 层实现 WOL 功能 - 这种情况下，系统睡眠时 MAC 设备与 phy 芯片必须能收包。
- PHY 层实现 WOL 功能 - 这种情况下，系统睡眠时 phy 芯片必须能收包。

裕太 phy 驱动支持这二种方式。

### 2.8.2 支持 MAC 层设备实现 WOL

如下定义实现 MAC 层设备的 WOL：

```
#define SYS_WAKEUP_BASED_ON_ETH_PKT    1
#define YTPHY_ENABLE_WOL                0
```

### 2.8.3 支持 PHY 设备实现 WOL

如下定义实现 PHY 层设备的 WOL：

```
#define SYS_WAKEUP_BASED_ON_ETH_PKT    1
#define YTPHY_ENABLE_WOL                1
```

如果宏#define YTPHY\_ENABLE\_WOL 定义为 1，SYS\_WAKEUP\_BASED\_ON\_ETH\_PKT 也会自动定义为 1。

#### 2.8.3.1 Linux 内核 PHY 层 WOL 相关回调函数

Linux 内核定义了关于 phy 层支持 wol 功能的标准接口，参见 struct phy\_driver 的定义：

```
struct phy_driver {
    u32 phy_id;
    ...
    int (*set_wol)(struct phy_device *dev, struct ethtool_wolinfo *wol);
    void (*get_wol)(struct phy_device *dev, struct ethtool_wolinfo *wol);
    ...
};
```

回调函数 set\_wol()用于使能/禁用 phy WOL 功能。

回调函数 get\_wol()用于查询 phy WOL 当前的状态(是否使能)。

相应对，裕太 phy 驱动里对应的函数是：

```
static int ytpHY_set_wol(struct phy_device *dev, struct ethtool_wolinfo *wol);
static void ytpHY_get_wol(struct phy_device *dev, struct ethtool_wolinfo *wol);
```

宏#define YTPHY\_ENABLE\_WOL 定义为 1 时，这二个函数就被自动关联：

```

#if (YTPHY_ENABLE_WOL)
    .get_wol = &ytphy_get_wol,
    .set_wol = &ytphy_set_wol,
#endif

```

通过这种方式支持时，用户需要确认其系统在网络设备上使能/禁用 WOL 时能否直接回调到这些函数。我们提供的驱动缺省就是这种方式。

### 2.8.3.2 自定义实现 PHY 层 WOL 功能

但时，目前很多 linux 内核并未实现.set\_wol()的回调，如果用户想通过 phy 实现 WOL 功能时，需要自己修改代码来实现。步骤如下：

- 1) 定义 YTPHY\_ENABLE\_WOL 为 1。
- 2) 在 MAC 地址配置完成之后通过调用驱动的 ytphy\_set\_wol()函数使能 phy WOL 功能。这个可以在系统的 MAC 驱动里(请参考系统的 MAC 设备驱动)，例如以下代码：

以 stmmac MAC 代码为例：linux/drivers/net/ethernet/stmicro/stmmac/stmmac\_main.c

```

int stmmac_open(struct net_device *dev)
{
    struct stmmac_priv *priv = netdev_priv(dev);
    int mode = priv->plat->phy_interface;
    ...

    ret = init_dma_desc_rings(dev, GFP_KERNEL);
    if (ret < 0) {
        netdev_err(priv->dev, "%s: DMA descriptors initialization failed\n",
            __func__);
        goto init_error;
    }

    ret = stmmac_hw_setup(dev, true);
    if (ret < 0) {
        netdev_err(priv->dev, "%s: Hw setup failed\n", __func__);
        goto init_error;
    }

    //在这里，MAC 地址已经配置完成，可以调用驱动的 WOL 配置函数，如 ytphy_set_wol()

}

```

也可以在 PHY 设备的初始化回调函数里，以 yt8521 为例，可以放在 yt8521\_config\_init()里：

```

static int yt8521_config_init(struct phy_device *phydev)
{
    int ret;
    int val, hw_strap_mode;
    #if (YTPHY_ENABLE_WOL)
    struct ethtool_wolinfo wol;

```

```

    /* set phy wol enable */
    memset(&wol, 0x0, sizeof(struct ethtool_wolinfo));
    wol.wolopts |= WAKE_MAGIC;
    ytpHY_set_wol(phydev, &wol);
    #endif
    ...
}

```

### 3. 用户空间驱动的使用

通常多口 phy 驱动会在用户空间下进行开发。YT8616/YT8614 提供在用户空间下的驱动 APIs。

#### 3.1 用户空间下驱动包

```

.\
├──yt-phy-driver-api           ; 驱动包主目录
│   ├── lib_yt861x.so         ; cli 版本的驱动 cygwin/linux 动态库
│   ├── yt861x.exe            ; cli 版本的驱动 cygwin 可执行文件
│   └──
│       ├──for-yt8614-demo-user-app ; non-cli 版本的驱动目录
│       ├──demo_app_entry_yt8614.c ; non-cli 版本的驱动程序入口文件
│       ├──demo_app_yt8614.exe     ; non-cli 版本的驱动 cygwin 可执行文件
│       ├──Makefile               ; non-cli 版本的 makefile
│       └──yt8614                 ; yt8614 驱动目录
│           ├──yt8614-phy.c       ; yt8614 驱动.c 文件
│           ├──yt8614-phy.h       ; yt8614 驱动.h 文件
└──yt_phy_demo_cli_cygwin        ; cli 版本的驱动目录。只有二进制文件
    ├──build                     ; cli 版本的驱动 make 目录
    │   ├── lib_yt861x.so        ; 同上述 cli 版本的驱动 cygwin/linux 动态库
    │   ├── Makefile             ; cli 版本的 makefile
    └── yt861x.exe                ; 同上述 cli 版本的驱动 cygwin 可执行文件

```

#### 3.2 用户空间下驱动 API 与移植说明

##### 3.2.1 数据结构

###### 3.2.1.1 phy\_info\_s

用于描述每个 YT8614/8618 chip 的信息，如 chip phy 基本地址，读写寄存器接口。系统上每个 chip 对应一个结构体。系统中 chip 个的定义见宏：MPHY\_YT8614\_CHIP\_NUM。

```
#define MPHY_YT8614_CHIP_NUM    1
```

```

typedef struct phy_info_str {
    unsigned int lport;           //保留
    unsigned int bus_id;         //保留
    unsigned int phy_addr;       //保留
    unsigned int base_phy_addr;  //chip 端口 phy 基地址。

    s32 (*read)(struct phy_info_str *info, phy_data_s *param); //读寄存器接口

```

```
s32 (*write)(struct phy_info_str *info, phy_data_s *param);    //写寄存器接口
} phy_info_s;
```

读写寄存器接口说明：这二个函数的作用：

- 1) 统一寄存器的访问。因为 YT8614/8618 的寄存器种类多，通过这二个函数统一调用。
- 2) 保证读写寄存器的原子性以实现寄存器读写的同步。YT8614/8618 的 ext, mmd 寄存器访问需要好几步骤才能完成，通过这二个函数可以实现多进程的同步操作。
- 3) 其他的作用，如还可以实现同一系统内 chip 的不同物理访问接口。

### 3.2.1.2 phy\_data\_s

寄存器属性定义结构体。

```
typedef struct {
    u16 reg;        //寄存器 id。
    u16 val;        //寄存器值。读寄存器时的返回值，写寄存器时准备写入的值。
    u8 regType;     //寄存器类型。
    u8 mmd;        //MMD id
} phy_data_s;
```

- 寄存器类型

因为 YT8614/8618 支持电口(UTP)，QSGMII，SGMII，相应地有不同的寄存器地址空间，还有一套公共的地址空间。每种寄存器空间又包括以下类型：

MII(基本)寄存器  
EXT(扩展)寄存器  
MMD 寄存器

这样组合下来，共有 9 种寄存器类型如下：

#define YT8614_TYPE_COMMON	0x01 // 公共寄存器，不需要地址空间切换，随时访问。
#define YT8614_TYPE_UTP_MII	0x02 // 电口地址空间 MII 寄存器。
#define YT8614_TYPE_UTP_EXT	0x03 // 电口地址空间 EXT 寄存器。
#define YT8614_TYPE_LDS_MII	0x04 // 光口地址空间 MII 寄存器。
#define YT8614_TYPE_UTP_MMD	0x05 // 电口地址空间 MMD 寄存器。
#define YT8614_TYPE_SDS_QSGMII_MII	0x06 // QSGMII 地址空间 MII 寄存器。
#define YT8614_TYPE_SDS_SGMII_MII	0x07 // Fiber/SGMII 地址空间 MII 寄存器。
#define YT8614_TYPE_SDS_QSGMII_EXT	0x08 // QSGMII 地址空间 EXT 寄存器。
#define YT8614_TYPE_SDS_SGMII_EXT	0x09 // Fiber/SGMII 地址空间 EXT 寄存器。

### 3.2.2 API 返回值定义

```
enum ytpHY_8614_errno_e {
    SYS_E_NONE,
    SYS_E_PARAM,
    SYS_E_MAX
};
```

所有 API 正常返回值为 SYS\_E\_NONE(0)，非 0 为异常。

### 3.2.3 多芯片支持与端口号(lport)的定义

- 全局端口号与片内端口号

所有 API 的端口号 `lport` 统一定义为**全局端口号**（或者称为统一端口号），就是一个系统内所有 `chip` 的 `phy` 端口进行统一编号。但是，在访问每个 `chip` 的寄存器时，又需要相对于某个 `chip` 的片内端口号。对于单个 `chip` 的应用来说，全局端口号与片内端口号是一致的，如：

对于 YT8614 每个 `chip` 是 4 个端口，分别是 0,1,2,3;

对于 YT8618 每个 `chip` 是 8 个端口，分别是 0,1,2,3,4,5,6,7;

系统中 `chip` 数目定义见宏：MPHY\_YT8614\_CHIP\_NUM

```
#define MPHY_YT8614_CHIP_NUM      1
```

`Chip` 的端口数目定义见宏 YT8614\_MAX\_NUM\_OF\_PORT:

```
#define YT8614_MAX_NUM_OF_PORT    4
```

系统内所有端口数目定义：

```
#define YT8614_MAX_LPORT_ID      (MPHY_YT8614_CHIP_NUM * YT8614_MAX_NUM_OF_PORT - 1)
```

对于多个 `chip` 的应用，比如 2 个 `yt8614` `chip`，端口的总数是：2\*4=8 个端口，相应对，全局端口号 `lport` 的取值是 0~7;

反过来，已经知道全局端口号如 7，对于 `yt8614` 来说：

```
chip_id = 7 / 4 = 1;
```

```
local port id = 7 % 4 = 3;    //片内端口号
```

### 3.2.4 寄存器访问函数的移植

寄存器访问函数是所有 API 的基础。因为系统硬件实现的差别，需要对以下二个函数重新实现：

- `static s32 yt8614_mdio_read_reg(unsigned int bus_id, unsigned int phy_addr, unsigned int reg, u16* val)`
- `static s32 yt8614_mdio_write_reg(unsigned int bus_id, unsigned int phy_addr, unsigned int reg, const u16 val)`

功能：yt8614/8618 寄存器读写

参数：bus\_id-输入，用于区分 MAC MDIO 总线。当前未用。

phy\_addr-输入，PHY 地址

reg-输入，reg id。

\*val-输出，读寄存器时用于返回值。

val-输入，写寄存器时的值。

#### 3.2.4.1 局部函数 yt8614\_app\_get\_phy\_info()

这个函数用于每个 API 调用之前确定寄存器访问函数、基地址以及端口号的关系的。用户可以根据系统的实现进行修改。

- `phy_info_s * yt8614_app_get_phy_info(u32 lport, phy_info_s *in_phy_info)`

功能：统一端口号与芯片，基地址，读写寄存器函数关系的映射。

参数：lport-输入，统一端口号。

in\_phy\_info-输出，全局变量本地化，避免全局变量的耦合导致的寄存器空间切换与访问不同步。

返回值：通常是一个本地变量以避免寄存器访问的不同步。

## 3.3 用户空间驱动的编译与开发

在 linux 系统用户空间下，程序通常需要一个 `main()` 作为入口。对于 YT8616/YT8614 驱动，我们提供了二种驱动应用示例：

- **Non-cli 版本的应用：**方便用户快速导入集成到已经有系统中，比如已经带有 CLI 的系统。

- CLI 版本：带有 CLI 命令界面。

### 3.3.1 用户空间 Non-cli 版本的应用 demo

主入口文件：demo\_app\_entry\_yt8614.c。

主要功能：

- 主入口函数示例
- API 调用函数。
- phy 端口状态轮询示例。

主入口函数示例如下。用户可以根据自己需求进行修改。

```
int main(int argc, char **argv) {
    pid_t fpid;
    int count=0;

#ifdef UART_MMIF
    /* for register access */
    if(uart_init((uint8_t *)NULL))
        return 1;
#endif

    /* entry for yt8614 */
    yt8614_app_init();
    #if 0
    /* create status polling process */
    fpid = fork();
    if (fpid < 0)
        printf("Error in fork, cannot create status polling thread!\n");
    else if (fpid == 0) {
        /* child here. */
    } else {
        /* main process and sleep for child running... */
        sleep(10);
    }
#endif

    /* main loop here */
    printf("Hit any key to exit...\n");
    while(1) {
#ifdef CMD_CLI_DAEMON_SUPPORTED
        if(yt8614_app_daemon())
            break;
#else
        printf("nothing to do...\n");
#endif
        if(kbhit()) {
            break;
        }
        count++;
    }

#ifdef UART_MMIF
    uart_exit();
#endif
    return 0;
}
```

```
}
```

### 3.3.1.1 程序编译

驱动包里有 Makefile，可以直接进行 make:

```
cd for-yt8614-demo-user-app
make
```

Makefile:

```
demo_app_yt8614: demo_app_entry_yt8614.o
    gcc -o demo_app_yt8614 demo_app_entry_yt8614.o

demo_app_entry_yt8614.o:
# gcc -DCMD_CLI_DAEMON_SUPPORTED -c demo_app_entry_yt8614.c
gcc -DCMD_CLI_DAEMON_SUPPORTED -DUART_MMIF -c demo_app_entry_yt8614.c

clean:
    rm -f *.o ../yt8614/*.o demo_app_yt8614
```

Makefile 说明:

1. 宏定义 UART\_MMIF
  - 1: 用于包括 mdio-com 驱动以进行实际板子上 phy 寄存器的访问。
  - 0: API 中访问 phy 寄存器的函数就是空的。
2. 宏定义 CMD\_CLI\_DAEMON\_SUPPORTED
  - 1: 包括 phy 端口状态轮询示例函数 yt8614\_app\_daemon()。
  - 0: 不包括 phy 端口状态轮询示例函数。

## 3.4 用户空间下 CLI 版本的驱动应用 demo

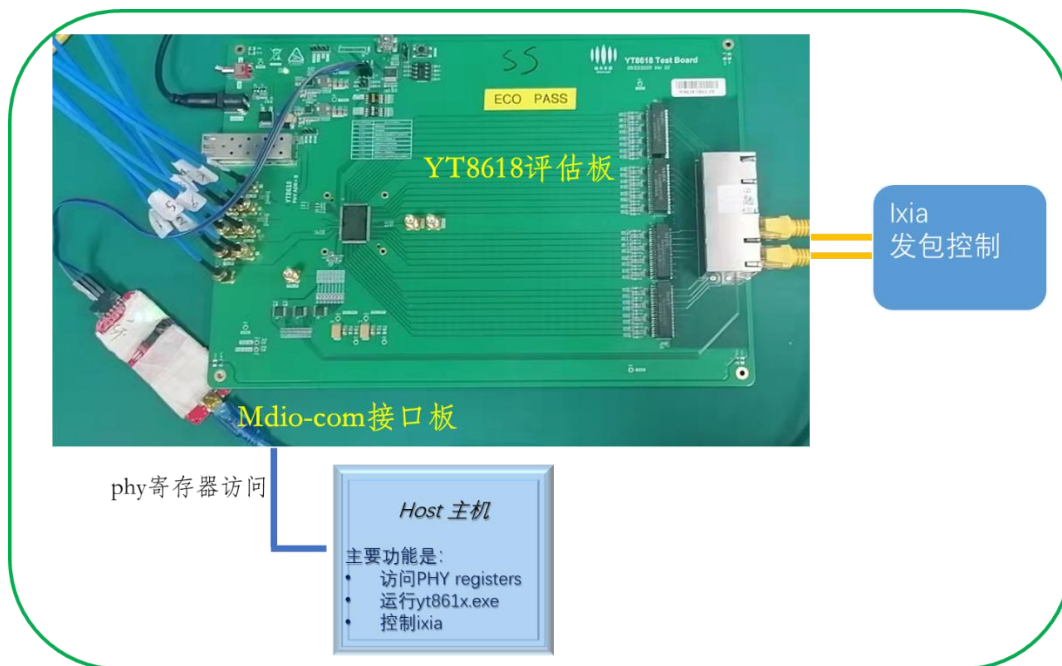
我们提供基于 Cygwin 环境的 CLI 版本的驱动应用 demo。

Cygwin 提供了在 Window 系统里模拟 linux 系统运行环境。裕太用户空间 phy 驱动可以在 cygwin 环境下编译运行以方便用户进行 API 的评估与测试调用。再结合我们公司相应的 demo 板就可以对 phy 进行性能评估与驱动 Api 调试。

### 3.4.1 Cygwin 环境下评估系统的硬件连接

硬件连接图如下:





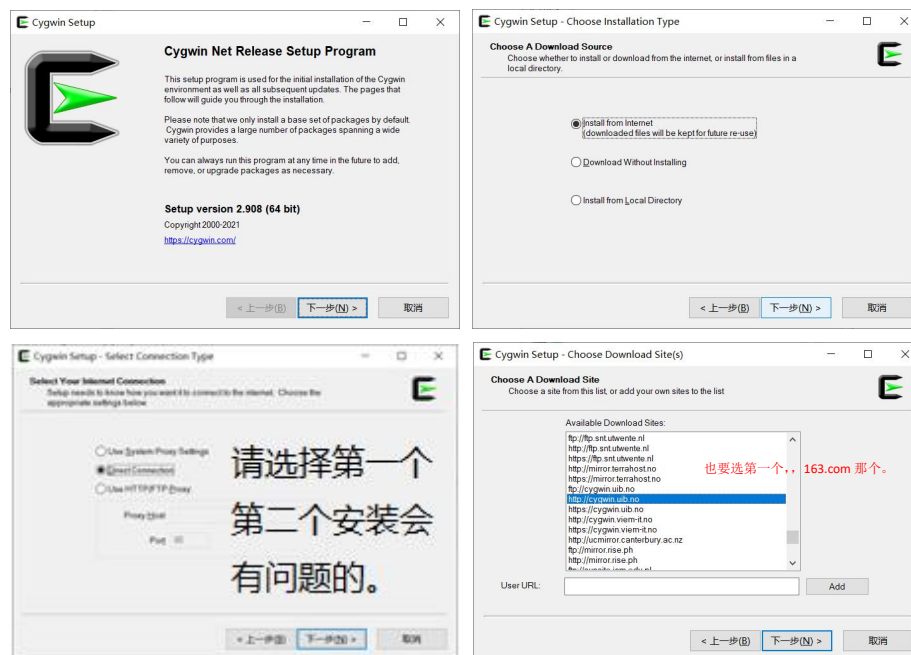
### 3.4.2 Cygwin 环境安装

步骤如下：

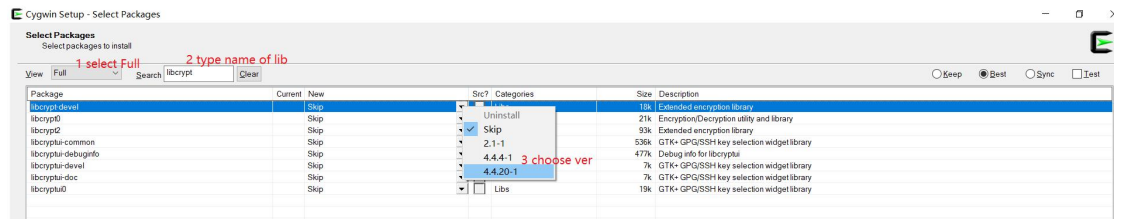
- a) 运行安装程序 setup-x86\_64.exe。

yt-phy-driver-api > cygwin >	修改日期
http%3a%2f%2fmirrors.163.com%2fc...	2021/11/25 19:06
setup.log	2021/11/4 16:13
setup.log.full	2021/11/4 16:13
setup-x86_64.exe	2021/5/18 14:40

setup-x86\_64.exe



接下来会让选 lib：除自动安装的库外，按图中样式一个一个选中特殊需要的库：



需要增加的模块 (绿色是编译环境需要的。如果不需要编译开发而只需运行我们提供的 yt861x.exe 文件则只安装红色部分的库即可：

```
Install binutils 2.37-2
Install gcc-core 11.2.0-1
Install gcc-g++ 11.2.0-1
Install gdb 10.2-1
Install libcrypt-devel 4.4.20-1
Install libncurses-devel 6.1-1.20190727
Install libreadline7 8.1-2
Install make 4.3-1
Install python36 3.6.13-2
Install python36-pytest 6.2.1-1
Install python36-tkinter 3.6.13-2
Install python36-devel 必选的，不然 cygwin 下编译通不过。
```

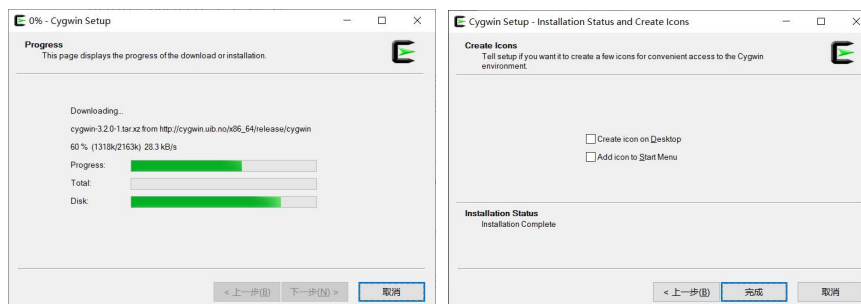
结果如下：红色是需要手动添加的模块

```
Install _autorebase 001007-1
Install alternatives 1.3.30c-10
Install base-cygwin 3.8-1
Install base-files 4.3-3
Install bash 4.4.12-3
Install binutils 2.37-2
Install bzip2 1.0.8-1
Install ca-certificates 2.50-4
Install coreutils 8.26-2
Install crypto-policies 20190218-1
Install cygutils 1.4.16-7
Install cygwin 3.2.0-1
Install cygwin-devel 3.2.0-1 (automatically added)
Install dash 0.5.11.5-1
Install dejavu-fonts 2.37-1 (automatically added)
Install diffutils 3.8-1
Install editrights 1.03-1
Install file 5.39-1
Install findutils 4.8.0-1
Install gawk 5.1.0-1
Install gcc-core 11.2.0-1
Install gcc-g++ 11.2.0-1
Install gdb 10.2-1
Install getent 2.18.90-4
Install grep 3.7-2
Install groff 1.22.4-1
Install gzip 1.11-1
Install hostname 3.13-1
Install info 6.8-2
Install ipc-utils 1.0-2
Install less 590-1
Install libX11_6 1.7.2-1 (automatically added)
Install libXau6 1.0.9-1 (automatically added)
Install libXdmc6 1.1.3-1 (automatically added)
Install libXext6 1.3.4-1 (automatically added)
Install libXft2 2.3.4-1 (automatically added)
Install libXrender1 0.9.10-1 (automatically added)
Install libXss1 1.2.3-1 (automatically added)
Install libargp 20110921-3
Install libatomic1 11.2.0-1 (automatically added)
Install libattr1 2.4.48-2
Install libblkid1 2.33.1-2
Install libboost_regex1.66 1.66.0-1 (automatically added)
Install libbrotlicommon1 1.0.9-2 (automatically added)
Install libbrotlidecl 1.0.9-2 (automatically added)
Install libbz2_1 1.0.8-1
Install libcrypt-devel 4.4.20-1
Install libcrypt2 4.4.20-1 (automatically added)
Install libexpat1 2.4.1-1 (automatically added)
Install libfdisk1 2.33.1-2
Install libffi6 3.2.1-2
Install libfontconfig-common 2.13.1-2 (automatically added)
Install libfontconfig1 2.13.1-2 (automatically added)
Install libfreetype6 2.11.0-2 (automatically added)
Install libgcl 8.0.6-1 (automatically added)
Install libgcc1 11.2.0-1
Install libgdbm6 1.18.1-1
Install libgdbm_compat4 1.18.1-1 (automatically added)
Install libgmp10 6.2.1-2
Install libgomp1 11.2.0-1 (automatically added)
Install libguile2.2_1 2.2.7-1 (automatically added)
Install libiconv 1.16-2 (automatically added)
Install libiconv2 1.16-2
Install libicu61 61.1-1 (automatically added)
Install libintl8 0.21-1
```

```

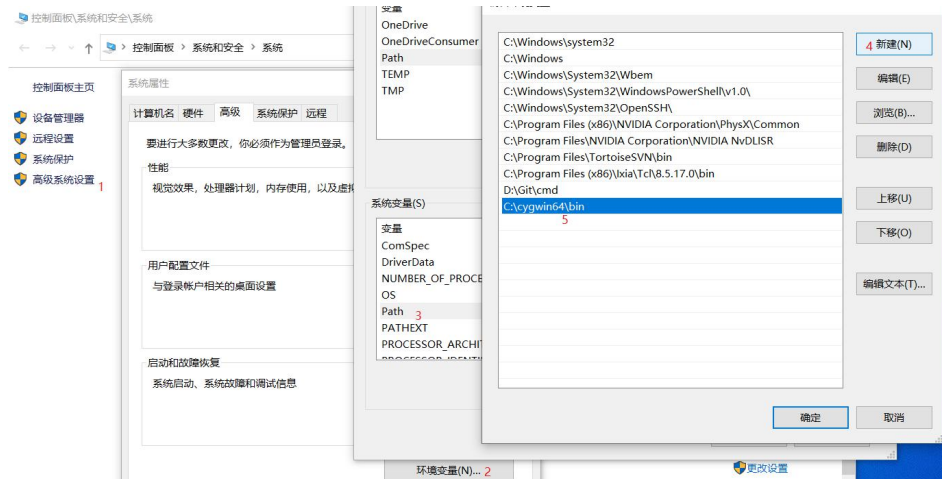
Install libisl23 0.24-2 (automatically added)
Install libltdl7 2.4.6-7 (automatically added)
Install liblz4_1 1.7.5-1
Install liblzma5 5.2.4-1
Install libmpc3 1.2.1-2 (automatically added)
Install libmpfr6 4.1.0-2
Install libncurses+*w10 6.1-1.20190727 (automatically added)
Install libncurses-devel 6.1-1.20190727
Install libncursesw10 6.1-1.20190727
Install libp11-kit0 0.23.20-1
Install libpcre1 8.45-1
Install libpcre2_8_0 10.38-1
Install libpipeline1 1.5.3-1
Install libpkgconf3 1.6.3-1 (automatically added)
Install libpng16 1.6.37-1 (automatically added)
Install libpopt-common 1.18-1
Install libpopt0 1.18-1
Install libquadmath0 11.2.0-1 (automatically added)
Install libreadline-devel 8.1-2
Install libreadline7 8.1-2
Install libsigsegv2 2.10-2
Install libsmartcols1 2.33.1-2
Install libsource-highlight-common 3.1.8-6 (automatically added)
Install libsource-highlight4 3.1.8-6 (automatically added)
Install libsqlite3_0 3.34.0-1 (automatically added)
Install libssl1.1 1.1.11-1
Install libstdc++6 11.2.0-1
Install libtasn1_6 4.14-1
Install libunistring2 0.9.10-1 (automatically added)
Install libuuid-devel 2.33.1-2 (automatically added)
Install libuuid1 2.33.1-2
Install libxcb1 1.14-1 (automatically added)
Install libzstd1 1.5.0-1 (automatically added)
Install login 1.13-1
Install make 4.3-1
Install man-db 2.9.4-1
Install mintty 3.5.1-1
Install ncurses 6.1-1.20190727
Install openssl 1.1.11-1
Install p11-kit 0.23.20-1
Install p11-kit-trust 0.23.20-1
Install pkg-config 1.6.3-1 (automatically added)
Install pkgconf 1.6.3-1 (automatically added)
Install python36 3.6.13-2
Install python36-attrs 20.3.0-1 (automatically added)
Install python36-pip 21.1.1-1 (automatically added)
Install python36-pytest 6.2.1-1
Install python36-setuptools 51.1.1-1 (automatically added)
Install python36-tkinter 3.6.13-2
Install python? -devel 必选的，不然 cygwin 下编译通不过。
Install python38 3.8.10-1 (automatically added)
Install python38-pip 21.1.1-1 (automatically added)
Install python38-setuptools 51.1.1-1 (automatically added)
Install rebase 4.5.0-1
Install run 1.3.4-2
Install sed 4.8-1
Install tar 1.34-1
Install tcl 8.6.11-1 (automatically added)
Install tcl-tix 8.4.3-3 (automatically added)
Install tcl-tk 8.6.11-1 (automatically added)
Install terminfo 6.1-1.20190727
Install terminfo-extra 6.1-1.20190727
Install tzcode 2021d-1
Install tzdata 2021d-1
Install util-linux 2.33.1-2
Install vim-minimal 8.2.0486-1
Install w32api-headers 9.0.0-1 (automatically added)
Install w32api-runtime 9.0.0-1 (automatically added)
Install which 2.20-2
Install windows-default-manifest 6.4-1 (automatically added)
Install xz 5.2.4-1
Install zlib0 1.2.11-1
Install zstd 1.5.0-1

```



## b) Cygwin 运行。

1. 请先修改 window 环境变量，增加一个路径：C:\cygwin64\bin

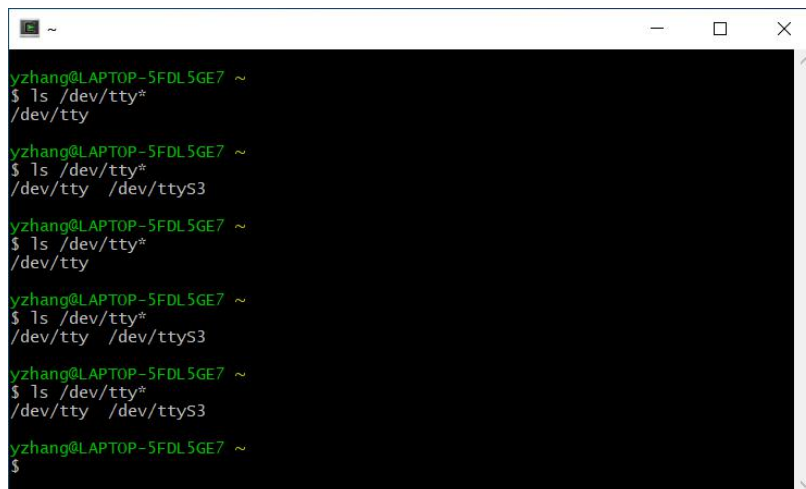


2. 连接 mdio-com USB 访问目标板的 registers.

连接上以后，通过下面的 cygwin mitty.exe 找到相应 com 端口的 tty 文件。或者运行：  
C:\cygwin64\bin\mintty.exe -i /Cygwin-Terminal.ico。（桌面上有。）



下图中 com 端口对应的 tty 设备是 ttyS3:



### 3.4.3 Cygwin 环境下运行 yt861x.exe

目前，CLI 版本的 phy 驱动应用 demo 只提供二进制文件 yt861x.exe 或者 lib\_yt861x.so。

yt861x.exe 使用步骤如下：

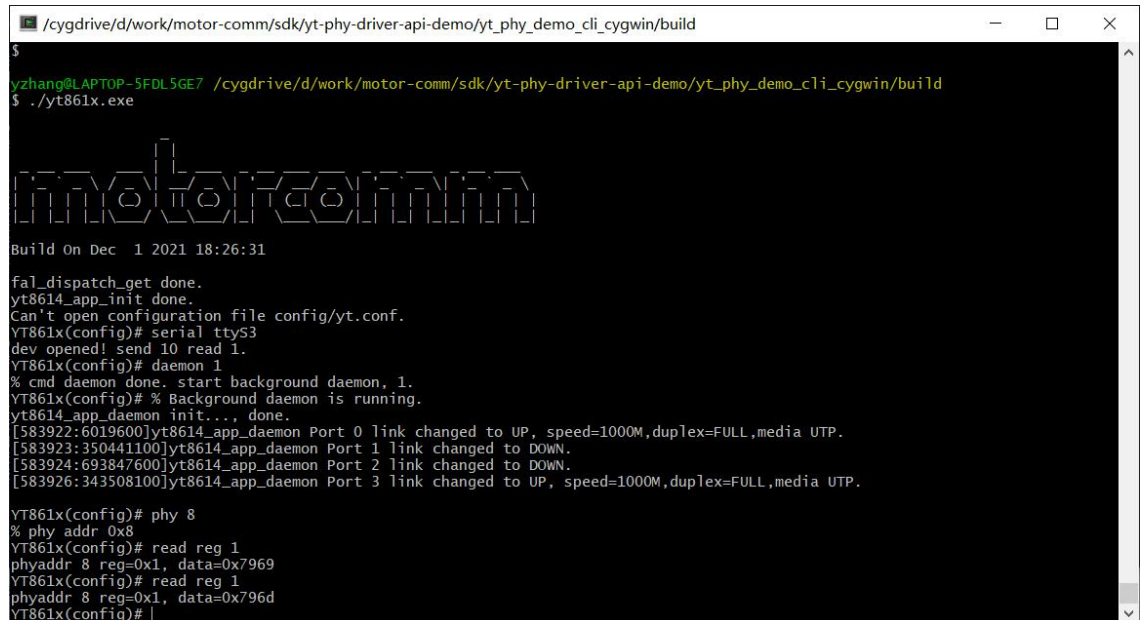
1) 运行：C:\cygwin64\bin\mintty.exe -i /Cygwin-Terminal.ico（桌面上有）。



2) 改变到 **yt861x.exe** 文件所在的路径，如：

D:\work\motor-comm\sdk\yt-phy-driver-api-demo\yt-phy-driver-api\yt861x.exe，在 cygwin 环境下是：

```
cd /cygdrive/d/work/motor-comm/sdk/yt-phy-driver-api-demo/yt-phy-driver-api/
./yt861x.exe
```



```

/cygdrive/d/work/motor-comm/sdk/yt-phy-driver-api-demo/yt_phy_demo_cli_cygwin/build
$
yzhang@LAPTOP-5FDL5GE7 /cygdrive/d/work/motor-comm/sdk/yt-phy-driver-api-demo/yt_phy_demo_cli_cygwin/build
$ ./yt861x.exe

motorcomm

Build On Dec 1 2021 18:26:31

fal_dispatch_get done.
yt8614_app_init done.
Can't open configuration file config/yt.conf.
YT861x(config)# serial ttyS3
dev opened! send 10 read 1.
YT861x(config)# daemon 1
% cmd daemon done. start background daemon, 1.
YT861x(config)# % Background daemon is running.
yt8614_app_daemon init..., done.
[583922:6019600]yt8614_app_daemon Port 0 link changed to UP, speed=1000M,duplex=FULL,media UTP.
[583923:350441100]yt8614_app_daemon Port 1 link changed to DOWN.
[583924:693847600]yt8614_app_daemon Port 2 link changed to DOWN.
[583926:343508100]yt8614_app_daemon Port 3 link changed to UP, speed=1000M,duplex=FULL,media UTP.

YT861x(config)# phy 8
% phy addr 0x8
YT861x(config)# read reg 1
phyaddr 8 reg=0x1, data=0x7969
YT861x(config)# read reg 1
phyaddr 8 reg=0x1, data=0x796d
YT861x(config)#

```

通常的初始化命令序列如下：

**serial ttyS3** ;打开 mdio-com 设备以进行寄存器访问。

**phy 8** ;设置 yt8614 芯片的 phy 基地址。这个需要向硬件确认。

做完以上二步就可以正常使用，比如访问各种 phy 寄存器，call api 等。

### 3.4.4 yt861x.exe 的功能简介

如下：

- 提供 cli 命令界面
- 访问 phy 寄存器
- 调用驱动程序的 API
- 多进程环境，提供监视 phy 工作状态的的后台进程，模拟真实运行环境。
- 提供方便调试或者测试的各种功能，比如：
  - 1) 多次运行一个命令的功能
  - 2) 脚本执行命令功能
  - 3) 后台进程执行/停止控制功能

### 3.4.5 CLI 命令简介

如下：

#### 3.4.5.1 打开/关闭 mdio-com 设备

在 yt861x.exe 开始工作时，先要打开 mdio-com 设备：

**serial [tty 设备名]** ;必须在 /dev 下有相应的设备名。

```

/cygdrive/d/work/motor-comm/sdk/yt-phy-driver-api-demo/yt-phy-driver-api
YT861x(config)#
YT861x(config)#
YT861x(config)#
YT861x(config)#
YT861x(config)# serial ttyS3
dev opened! send 10 read 1.
YT861x(config)#

```

关闭 mdio-com 设备命令是：

`serial close`

### 3.4.5.2 访问 phy 寄存器

如前述，yt8614/8618 的寄存器包括多种类型，相应的命令如下各章节介绍。

可以通过帮助命令获取寄存器访问的所有命令：

`YT861x(config)# read reg [TAB 键或者?]`

```

/cygdrive/d/work/motor-comm/sdk/yt-phy-driver-api-demo/yt-phy-driver-api
qsgmii
YT861x(config)# read reg
REG      Reg name e.g. 0xa000
ext      Reg name e.g. 0xa
fiber
mmd
qsgmii
YT861x(config)# read reg
ext      fiber      mmd      qsgmii
YT861x(config)# read reg

```

注意，访问寄存器之前需要正确设置 phy 地址。

#### 3.4.5.2.1 端口 phy 地址

访问寄存器之前需要设置端口 phy 地址。

`phy [端口 phy 地址]`；设置端口 phy 地址。

如：命令 `phy 0` 表示切换到 phy 地址 0。端口 phy 地址=chip phy 基地址+端口号。

请与硬件确认 8614/8618 phy 基地址的配置。

Phy 基地 址	端口 0 phy 地址	端口 1 phy 地址	端口 2 phy 地址	端口 3 phy 地址	端口 4 phy 地址	端口 5 phy 地址	端口 6 phy 地址	端口 7 phy 地址
x	x+0	x+1	x+2	x+3	x+4	x+5	x+6	x+7

#### 3.4.5.2.2 Utp mii 寄存器

- 读 phy utp mii 寄存器：

```

/cygdrive/d/work/motor-comm/sdk/yt-phy-driver-api-demo/yt-phy-driver-api
YT861x(config)# phy 0
% phy addr 0x0
YT861x(config)# read reg 1
reg=0x1, data=0x796d
YT861x(config)#
YT861x(config)#
YT861x(config)#

```

设置好基地址后，就可以读取相应端口的寄存器了：

`read reg [reg id]`；读取端口 utp mii 寄存器。

- 写 phy utp mii 寄存器：

`write reg [reg id]`；写端口 utp mii 寄存器。



```

/cygdrive/d/work/motor-comm/sdk/yt-phy-driver-api-demo/yt-phy-driver-api
reg=0x1, data=0x796d
YT861x(config)#
YT861x(config)#
YT861x(config)# read reg 0
reg=0x0, data=0x1000
YT861x(config)# write reg 0 0x9000
YT861x(config)# |

```

#### 3.4.5.2.3 Utp ext 寄存器

- 读 phy utp ext 寄存器:  
read reg ext [reg id] ;读取端口 utp ext 寄存器。
- 写 phy utp ext 寄存器:  
write reg ext [reg id] ;写端口 utp ext 寄存器。

#### 3.4.5.2.4 Utp mmd 寄存器

- 读 phy utp mmd 寄存器:  
read reg mmd [reg id] ;读取端口 utp mmd 寄存器。
- 写 phy utp mmd 寄存器:  
write reg mmd [reg id] ;写端口 utp mmd 寄存器。

#### 3.4.5.2.5 Fiber/Sgmii mii 寄存器

- 读 phy fiber mii 寄存器:  
read reg fiber [reg id] ;读取端口 fiber mii 寄存器。
- 写 phy fiber mii 寄存器:  
write reg fiber [reg id] ;写端口 fiber mii 寄存器。

#### 3.4.5.2.6 Fiber/Sgmii ext 寄存器

- 读 phy fiber ext 寄存器:  
read reg fiber ext [reg id] ;读取端口 fiber ext 寄存器。
- 写 phy fiber ext 寄存器:  
write reg fiber ext [reg id] ;写端口 fiber ext 寄存器。

#### 3.4.5.2.7 Qsgmii mii 寄存器

- 读 phy Qsgmii mii 寄存器:  
read reg qsgmii [reg id] ;读取端口 qsgmii mii 寄存器。
- 写 phy Qsgmii mii 寄存器:  
write reg qsgmii [reg id] ;写端口 qsgmii mii 寄存器。

#### 3.4.5.2.8 Qsgmii ext 寄存器

- 读 phy Qsgmii ext 寄存器:  
read reg qsgmii ext [reg id] ;读取端口 qsgmii ext 寄存器。
- 写 phy Qsgmii ext 寄存器:  
write reg qsgmii ext [reg id] ;写端口 qsgmii ext 寄存器。

### 3.4.5.3 调用驱动程序 API

`call api [函数名] [参数 1/参数 2/参数 n...]` ;调用驱动 api

如：调用 API `s32 yt8614_phy_init(u32 lport)`:

```

/cygdrive/d/work/motor-comm/sdk/yt-phy-driver-api-demo/yt-phy-driver-api
YT861x(config)# read reg 0
reg=0x0, data=0x1000
YT861x(config)# write reg 0 0x9000
YT861x(config)# call api yt8614_phy_init 0
yt8614_phy_init called for port 0.
lport:0 reg 0x41 = 0x0033
lport:0 reg 0x42 = 0x0066
lport:0 reg 0x43 = 0x00aa
lport:0 reg 0x44 = 0x0d0d
lport:0 reg 0x0 = 0x9000
YT861x(config)#

```

注意多个参数是以/进行分隔，如：参数 0/参数 1/参数 2:

```

/cygdrive/d/work/motor-comm/sdk/yt-phy-driver-api-demo/yt-phy-driver-api
YT861x(config)# write reg 0 0x9000
YT861x(config)# call api yt8614_phy_init 0
yt8614_phy_init called for port 0.
lport:0 reg 0x41 = 0x0033
lport:0 reg 0x42 = 0x0066
lport:0 reg 0x43 = 0x00aa
lport:0 reg 0x44 = 0x0d0d
lport:0 reg 0x0 = 0x9000
YT861x(config)# call api yt8614_media_status_get 0/0/0/0/0/0
yt8614_media_status_get lport=0,speed=2,duplex=1,ret_link=1,media=1
YT861x(config)# |

```

#### 3.4.5.4 重复执行一条命令

`monitor [重复次数]` ;重复执行接下来的 cli 命令指定次数（0 表示无限）。回车键停止重复。

示例：

`monitor 0` ; 后面一条命令将被重复执行。

`read reg 1` ; 这条命令将被重复执行。按下回车键停止重复。

```

/cygdrive/d/work/motor-comm/sdk/yt-phy-driver-api-demo/yt_phy_demo_cli_cygwin/build
YT861x(config)#
YT861x(config)#
YT861x(config)# monitor 0
% cmd monitor done. And next cmd will run forever. Press Enter to stop.
YT861x(config)# read reg 1
phyaddr 8 reg=0x1, data=0x7969
phyaddr 8 reg=0x1, data=0x796d
phyaddr 8 reg=0x1, data=0x796d
phyaddr 8 reg=0x1, data=0x796d
phyaddr 8 reg=0x1, data=0x796d
phyaddr 8 reg=0x1, data=0x796d
phyaddr 8 reg=0x1, data=0x796d
phyaddr 8 reg=0x1, data=0x796d
phyaddr 8 reg=0x1, data=0x796d

```

#### 3.4.5.5 后台进程监视 phy 工作状态

`daemon [开关]` ;后台端口状态轮询进程开关。

启动：

`daemon 1`

停止：

`daemon 0`



```

/cygdrive/d/work/motor-comm/sdk/yt-phy-driver-api-demo/yt_phy_demo_cli_cygwin/build
YT861x(config)#
YT861x(config)# daemon 1[584184:281139900]yt8614_app_daemon Port 3 link changed to DOWN.
[584185:593199400]yt8614_app_daemon Port 0 link changed to DOWN.

% Command incomplete.
YT861x(config)# daemon 0
% cmd daemon done. stop background daemon, 0.
YT861x(config)# % Background daemon is cancelled.

YT861x(config)# daemon 1
% cmd daemon done. start background daemon, 1.
YT861x(config)# % Background daemon is running.
[584219:209127400]yt8614_app_daemon Port 0 link changed to UP, speed=1000M,duplex=FULL,media UTP.
[584223:530693100]yt8614_app_daemon Port 3 link changed to UP, speed=1000M,duplex=FULL,media UTP.
YT861x(config)#

```

### 3.4.5.6 执行命令脚本

load file [文件名] ;在当前目录的 config 下查找[文件名]并执行其包含的 cli 命令。

```

/cygdrive/d/work/motor-comm/sdk/yt-phy-driver-api-demo/yt_phy_demo_cli_cygwin/build

Build On Dec 7 2021 16:51:46

fal_dispatch_get done.
yt8614_app_init done.
Can't open configuration file config/yt.conf.
YT861x(config)# load file ./t.txt
Please specify string starting with alphabet
YT861x(config)# load file t.txt
% Invalid phy addr 255.
% Invalid phy addr 255.
YT861x(config)#

```

例中：t.txt 文件内容：

```
write reg 12 0x7555
```

```
read reg 12
```

注意：在脚本里不能包含 monitor 命令。

## 4. 用户空间驱动 API 说明

### 4.1 YT8614 驱动 API 说明

#### 4.1.1 寄存器读写 API

```
s32 yt8614_read_reg(struct phy_info_str *info, phy_data_s *param);
```

```
s32 yt8614_write_reg(struct phy_info_str *info, phy_data_s *param);
```

功能：读写寄存器。

参数：请参考数据结构里说明。

说明：建议实现访问互斥功能保证多进程情况下访问的正确性。

#### 4.1.2 s32 yt8614\_phy\_soft\_reset(u32 lport);

功能：软复位指定端口

参数：lport-输入，全局端口号

#### 4.1.3 s32 yt8614\_phy\_init(u32 lport);

功能：对指定端口进行初始化配置

参数：lport-输入，全局端口号

说明：端口初始化配置不是必须的。

#### 4.1.4 s32 yt8614\_fiber\_enable(u32 lport, BOOL enable);

功能：指定端口光口模式 powerdown/ 正常

参数: lport-输入, 全局端口号

enable: powerdown (0) / 正常 (1)

#### 4.1.5 s32 yt8614\_utp\_enable(u32 lport, BOOL enable);

功能: 指定端口电口模式 powerdown/ 正常

参数: lport-输入, 全局端口号

enable: powerdown (0) / 正常 (1)

#### 4.1.6 s32 yt8614\_fiber\_unidirection\_set(u32 lport, int speed, BOOL enable);

功能: 设置指定端口光口模式单光纤使能/禁用和速率

参数: lport-输入, 全局端口号

speed-输入: SPEED\_1000M(2)/SPEED\_100M(1)/SPEED\_10M(0)

enable-输入: 禁用 (0) / 使能 (1)

说明: 通常 fiber 模式下, 光纤是有收有发。本函数设置只收不发功能。

#### 4.1.7 s32 yt8614\_fiber\_autosensing\_set(u32 lport, BOOL enable);

功能: 指定端口光口模式自协商使能/禁用

参数: lport-输入, 全局端口号

enable: 禁用 (0) / 使能 (1)

#### 4.1.8 s32 yt8614\_fiber\_speed\_set(u32 lport, int fiber\_speed);

功能: combo 模式时, 设置指定端口光口模式的速率

参数: lport-输入, 全局端口号

fiber-speed-输入: YT8614\_COMBO\_UTP\_ONLY (2)/ YT8614\_COMBO\_FIBER\_100M (1)/ YT8614\_COMBO\_FIBER\_1000M (0)

#### 4.1.9 s32 yt8614\_qsgmii\_autoneg\_set(u32 lport, BOOL enable);

功能: 指定端口 QSGMII 接口自协商使能/禁用

参数: lport-输入, 全局端口号

enable: 禁用 (0) / 使能 (1)

#### 4.1.10 s32 yt8614\_sgmii\_autoneg\_set(u32 lport, BOOL enable);

功能: 指定端口 SGMII 接口自协商使能/禁用

参数: lport-输入, 全局端口号

enable: 禁用 (0) / 使能 (1)

#### 4.1.11 s32 yt8614\_qsgmii\_sgmii\_link\_status\_get(u32 lport, BOOL \*enable, BOOL if\_qsgmii);

功能: 获得指定端口 QSGMII/SGMII 的 link 状态

参数: lport-输入, 全局端口号

if\_qsgmii -输入, QSGMII(1), SGMII(0)

enable-输出: linkdown (0) / linkup (1)

说明: 对于 yt8614, 只有一个 QSGMII, 一个 SGMII, 所以端口号 0-3 的结果是一样的。

#### 4.1.12 int yt8614\_combo\_media\_priority\_set(u32 lport, int fiber);

功能: combo 模式时对指定端口设置电口/光口优先模式

参数: lport-输入, 全局端口号

fiber-输入: 电口优先 (0) / 光口优先 (1)

#### 4.1.13 int yt8614\_combo\_media\_priority\_get(u32 lport, int \*fiber);

功能: combo 模式下获得指定端口电口/光口优先状态

参数: lport-输入, 全局端口号

fiber-输出: 电口优先 (0) / 光口优先 (1)

**4.1.14 s32 yt8614\_utp\_autoneg\_set(u32 lport, BOOL enable);**

功能：指定端口电口模式自协商使能/禁用

参数：lport-输入，全局端口号

enable：禁用（0）/使能（1）

**4.1.15 s32 yt8614\_utp\_autoneg\_get(u32 lport, BOOL \*enable);**

功能：获取指定端口电口模式自协商状态

参数：lport-输入，全局端口号

Enable-输出：禁用（0）/使能（1）

**4.1.16 s32 yt8614\_utp\_autoneg\_ability\_set(u32 lport, unsigned int cap\_mask);**

功能：设置指定端口电口模式自协商的能力

参数：lport-输入，全局端口号

cap\_mask -输入：能力 mask 位图：

说明：位图定义如下：

```
#define ADVERTISE_1000HALF      0x1000  /* Advertise 1000BASE-T half duplex */
#define ADVERTISE_PAUSE_ASYM    0x0800  /* Try for asymmetric pause      */
#define ADVERTISE_PAUSE_CAP    0x0400  /* Try for pause                  */
#define ADVERTISE_1000FULL     0x0200  /* Advertise 1000BASE-T full duplex */
#define ADVERTISE_100FULL      0x0100  /* Try for 100mbps full-duplex */
#define ADVERTISE_100HALF      0x0080  /* Try for 100mbps half-duplex */
#define ADVERTISE_10FULL       0x0040  /* Try for 10mbps full-duplex  */
#define ADVERTISE_10HALF       0x0020  /* Try for 10mbps half-duplex  */
```

**4.1.17 s32 yt8614\_utp\_autoneg\_ability\_get(u32 lport, unsigned int \*cap\_mask);**

功能：获取指定端口电口模式自协商的能力

参数：lport-输入，全局端口号

cap\_mask -输入：能力 mask 位图。见上定义。

**4.1.18 s32 yt8614\_utp\_force\_duplex\_set(u32 lport, BOOL full);**

功能：设置指定端口电口模式强制双工

参数：lport-输入，全局端口号

full-输入：half duplex（0）/full duplex（1）

**4.1.19 s32 yt8614\_utp\_force\_duplex\_get(u32 lport, BOOL \*full);**

功能：获取指定端口电口模式强制双工状态

参数：lport-输入，全局端口号

full-输出：half duplex（0）/full duplex（1）

**4.1.20 s32 yt8614\_utp\_force\_speed\_set(u32 lport, unsigned int speed);**

功能：设置指定端口电口模式强制速率

参数：lport-输入，全局端口号

speed-输入：SPEED\_1000M(2)/SPEED\_100M(1)/SPEED\_10M(0)

**4.1.21 s32 yt8614\_utp\_force\_speed\_get(u32 lport, unsigned int \*speed);**

功能：获取指定端口电口模式强制速率状态

参数：lport-输入，全局端口号

speed-输出：SPEED\_1000M(2)/SPEED\_100M(1)/SPEED\_10M(0)/SPEED\_UNKNOWN(0xffff)

**4.1.22 int yt8614\_autoneg\_done\_get(u32 lport, int speed, int \*aneg);**

功能：获取指定端口自协商结果

参数：lport-输入，全局端口号

speed-输入，当前速率。

aneg-输出：未完成（0）/完成（1）

说明：在光口状态下，100M 速率时没有自协商结果。这种情况下直接返回为 1。

**4.1.23 int yt8614\_media\_status\_get(u32 lport, int \*speed, int \*duplex, int \*ret\_link, int \*media);**

功能：指定端口状态查询

参数：lport-输入，全局端口号

speed-输出，当前端口速率

duplex-输出，当前端口双工

ret\_link-输出，当前端口 link 状态：0 link down; 1 link up

media-输出，当前介质：

电口：YT8614\_SMI\_SEL\_PHY + 1

光口：YT8614\_SMI\_SEL\_SDS\_SGMII + 1

enable: 禁用（0）/使能（1）

说明：如果 link 状态为 0 link down，那么其他输出的值无效。