Motorcomm Inc

裕太以太网 Phy 驱动使用手册

The guide to use of the software

Ver-2 2023/08/22

motorcomm
2023/8/22

User Guide

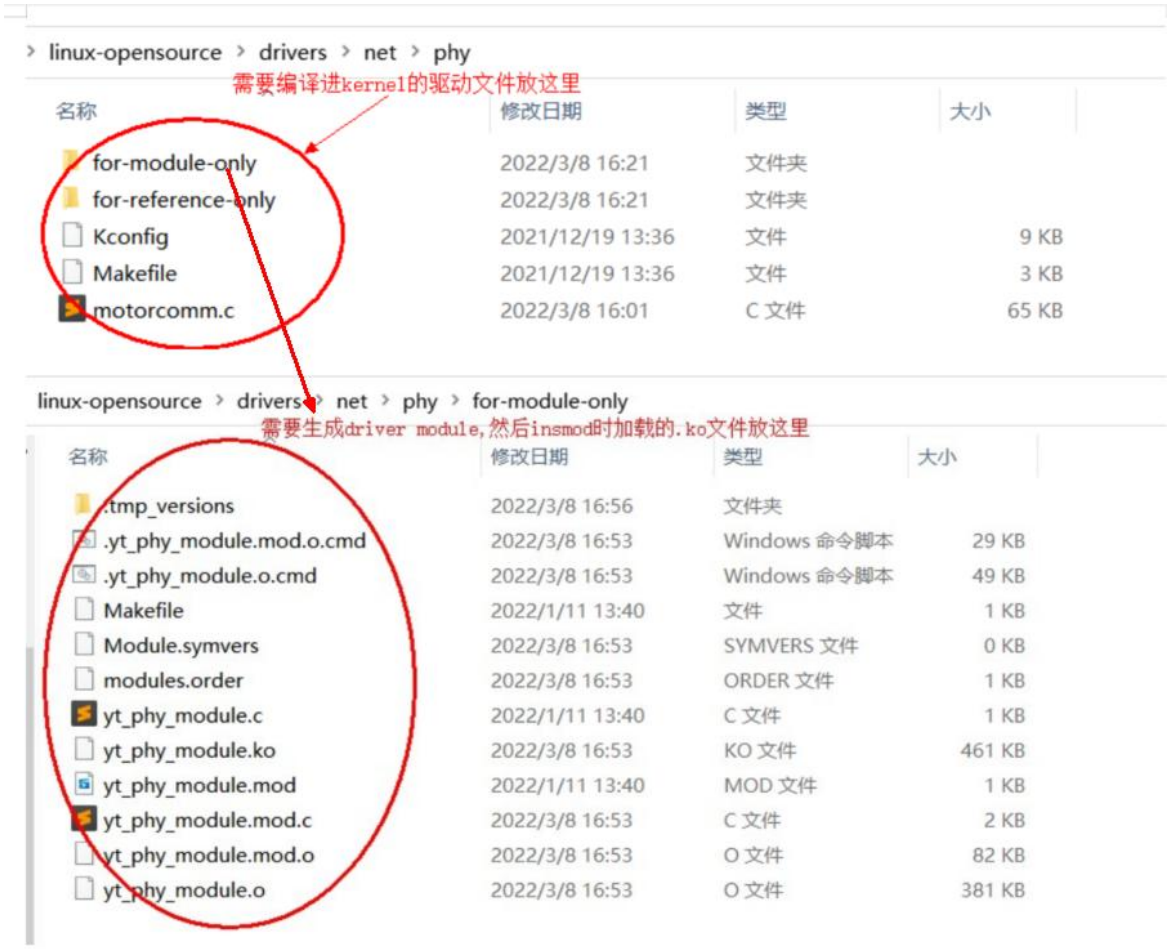
1. 内核空间驱动的编译加载说明

1.1 Linux 系统

1.1.1 解压驱动代码源文件

Motorcomm 的驱动 linux-opensource 如下：

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 copy 到如下的目录：

cp motorcomm.c ~/motocomm/kernel/linux-rt-4.19.94/drivers/net/phy/.

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 PHY, ,YT8521 PHY, YT8531 PHY, YT8614 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/linux-opensource .

   cd linux-opensource/drivers/net/phy/for-module-only/

   make clean && make

```
make -C /lib/modules/`uname -r`/build M=/home/kylin/yzhang/linux-opensource/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/linux-opensource/drivers/net/phy/for-module-only/.tmp_versions
  CLEAN   /home/kylin/yzhang/linux-opensource/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/linux-opensource/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/linux-opensource/drivers/net/phy/for-module-only/yt_phy_module.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/kylin/yzhang/linux-opensource/drivers/net/phy/for-module-only/yt_phy_module.mod.o
  LD [M]  /home/kylin/yzhang/linux-opensource/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@am335x-evm:~# ls /sys/bus/mdio_bus/drivers
总用量 0
drwxr-xr-x 2 root root 0 3月   9 22:36 'Generic Clause 45 PHY'
drwxr-xr-x 2 root root 0 3月   9 22:36 'Generic PHY'
drwxr-xr-x 2 root root 0 3月   9 14:36 'YT8010 Automotive Ethernet'
drwxr-xr-x 2 root root 0 3月   9 14:36 'YT8510 100!10Mb Ethernet'
drwxr-xr-x 2 root root 0 3月   9 14:36 'YT8511 Gigabit Ethernet'
drwxr-xr-x 2 root root 0 3月   9 14:36 'YT8512B Ethernet'
drwxr-xr-x 2 root root 0 3月   9 14:36 'YT8512 Ethernet'
drwxr-xr-x 2 root root 0 3月   9 14:36 'YT8521 Ethernet'
drwxr-xr-x 2 root root 0 3月   9 14:36 'YT8531 Gigabit Ethernet'
drwxr-xr-x 2 root root 0 3月   9 14:36 'YT8531S Ethernet'
drwxr-xr-x 2 root root 0 3月   9 14:36 'YT8614 Ethernet'
drwxr-xr-x 2 root root 0 3月   9 14:36 'YT8618 Ethernet'
drwxr-xr-x 2 root root 0 3月   9 14:36 'YT8821 2.5Gb Ethernet'
```

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
enaphyt4i0 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
```

1.2 uboot 系统

1.2.1 uboot 目录结构

```
u-boot-master+                  ; top directory of u-boot-master
             +---arch
             +---configs
             +---drivers        ; 驱动主目录
             |   +---net        ; 网络驱动目录
             |   |   +---phy    ; phy设备驱动目录
             +---dts
             +---include
```

1.2.2 合并 phy 驱动代码进 uboot

合并详细步骤如下：

. 解压 uboot 框架下 yt phy 驱动文件包，获取 motorcomm.c

. 拷贝 motorcomm.c 到目录 u-boot-master/drivers/net/phy/

**********如下步骤适用于 u-boot v2023.04(含)以下**********

. 编辑/include/phy.h

增加 int phy_yt_init(void);

. 编辑 u-boot-master/drivers/net/phy/phy.c, phy_init() function，增加如下所示红色代码

```
int phy_init(void)
{
 ......
#ifdef CONFIG_MOTORCOMM_PHY
  phy_yt_init();
#endif
```

……

}

**********如下步骤适用于 u-boot v2023.04(含)以下**********

. 编辑 u-boot-master/drivers/net/phy/Makefile,

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

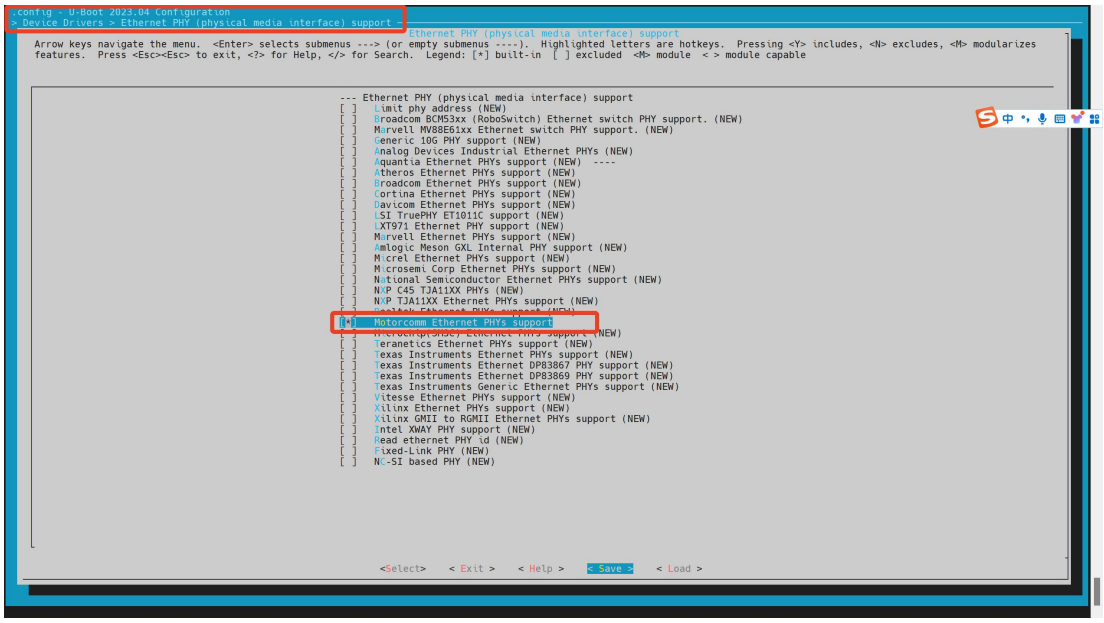
. 编辑 u-boot-master/drivers/net/phy/Kconfig

增加

config MOTORCOMM_PHY

bool "Motorcomm Ethernet PHYs support"

. u-boot-master 目录下 make menuconfig



. u-boot-master 目录下 make

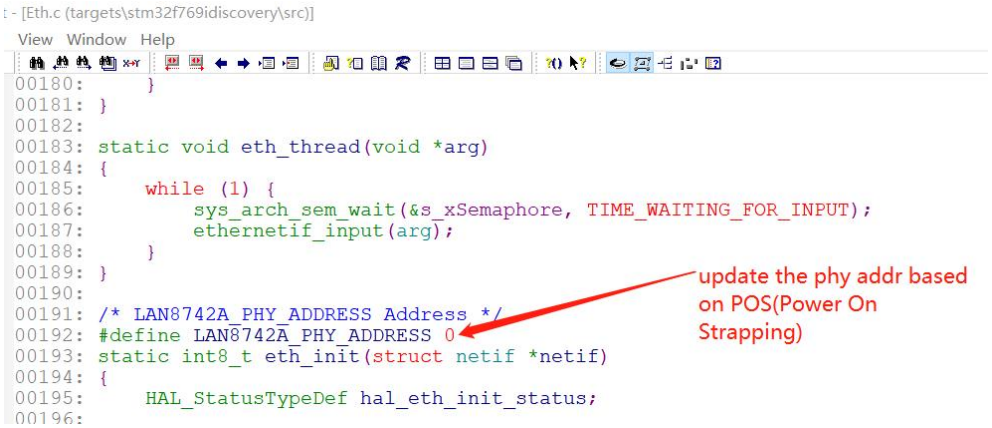
1.3 MCU phy driver porting

MCU stm32f769 + liteOS as sample in this guide will be introduced.

1.3.1 yt phy address definition

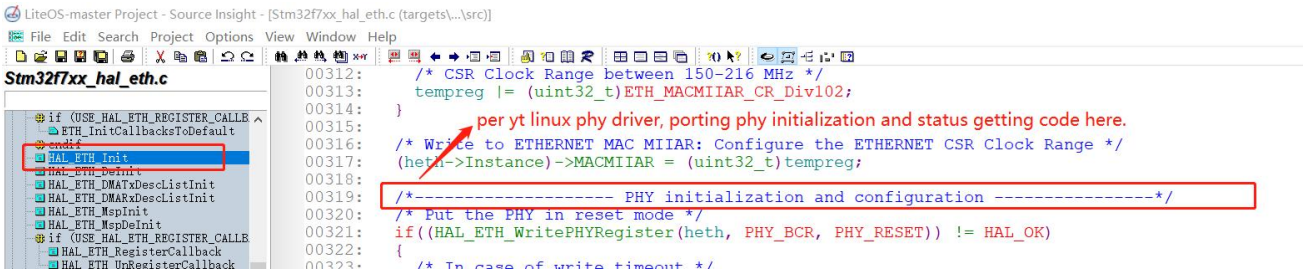
phy address based on the POS(Power On Strapping) will be updated in Eth.c (targets\stm32f769idiscovery\src).

#define LAN8742A_PHY_ADDRESS 0



1.3.2 yt phy initialization & phy status getting

Refer to yt linux phy driver, phy initializing & status getting code will be updated in HAL_ETH_Init() Stm32f7xx_hal_eth.c
(targets\bsp\drivers\stm32f7xx_hal_driver\src).



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。

2.2 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;
}
```

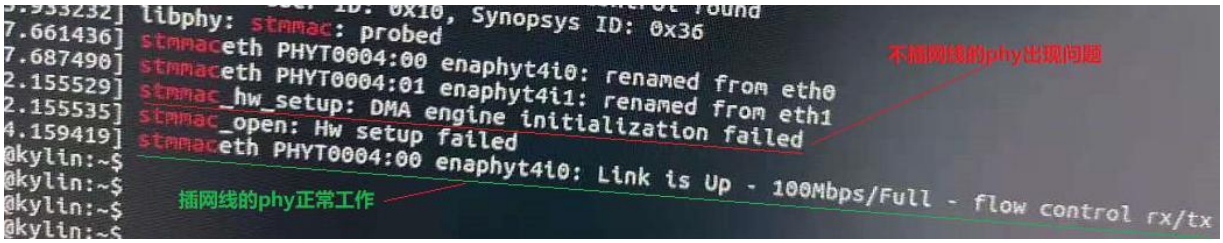
解决：

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：



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/YT8531/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。

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_WOL_FEATURE_ENABLE        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_WOL_FEATURE_ENABLE        1
```

如果宏#define YTPHY_WOL_FEATURE_ENABLE 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_WOL_FEATURE_ENABLE 定义为 1 时，这二个函数就被自动关联：

```
#if (YTPHY_WOL_FEATURE_ENABLE)
        .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_WOL_FEATURE_ENABLE 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_WOL_FEATURE_ENABLE)
        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。YT8614/YT8614Q/YT8618 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
    ├──yt8614                           ；yt8614 驱动目录
    |       yt8614-phy.c                ；yt8614 驱动.c 文件
    |       yt8614-phy.h                ；yt8614 驱动.h 文件
    |
    ├──yt8614Q                          ；yt8614Q 驱动目录
    |       yt8614Q-phy.c               ；yt8614Q 驱动.c 文件
    |       yt8614Q-phy.h               ；yt8614Q 驱动.h 文件
    |
    ├──yt8618                           ；yt8618 驱动目录
            yt8618-phy.c                ；yt8618 驱动.c 文件
            yt8618-phy.h                ；yt8618 驱动.h 文件
```

3.2 用户空间下驱动 API 与移植说明

3.2.1 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，以 YT8614 为例：

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

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
```

3.2.2 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，以 yt8614 为例，需要对以下二个函数重新实现：

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

功能：yt8614 寄存器读写

参数：

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。

3.2.2.1 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。

yt8614_phy_dev_info_t * yt8614_app_get_phy_info(u32 lport, yt8614_phy_dev_info_t *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。

3.2.3 yt8614 数据结构

3.2.3.1 yt8614_phy_dev_info

用于描述每个 YT8614 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 数目的定义见宏：MPHY_YT8614_CHIP_NUM。

```
#define MPHY_YT8614_CHIP_NUM        1

typedef struct yt8614_phy_dev_info {
    unsigned int lport;             //chip 内部端口 index
    unsigned int phy_addr;          //端口实际 phy address(base_phy_addr + lport)
    unsigned int base_phy_addr;     //chip 端口 phy 基地址
    s32 (*read)(struct yt8614_phy_dev_info *info, struct yt8614_phy_reg *param);//读寄存器接口
    s32 (*write)(struct yt8614_phy_dev_info *info, struct yt8614_phy_reg *param);//写寄存器接口
} yt8614_phy_dev_info_t;
```

读写寄存器接口说明：

1) 统一寄存器的访问。因为 YT8614 的寄存器种类多，通过上面这两个函数统一调用。
2) 保证读写寄存器的原子性以实现寄存器读写的同步。寄存器访问需要好几步骤才能完成，通过这两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。

3.2.3.2 yt8614_phy_reg

寄存器属性定义结构体。

```
struct yt8614_phy_reg {
    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
};
```

- 寄存器类型

因为 YT8614 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)寄存器和 EXT(扩展)寄存器，其中 UTP 还包括 MMD 寄存器和 LDS 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。
```

3.2.3.3 yt8614_polarity_dir_t

```
typedef enum {
```

```
  YT8614_SND,      //发送方向
  YT8614_RCV,      //接收方向
}yt8614_polarity_dir_t;
```

3.2.3.4 yt8614_utp_vct_t

```
typedef enum
{
  YT8614_VCT_INTER_BOARD,          //板间 vct 测试
  YT8614_VCT_NON_INTER_BOARD    //非板间 vct 测试
}yt8614_utp_vct_t;
```

3.2.3.5 yt8614_speed_t

```
typedef enum
{
  YT8614_10BT,      //10Mbps
  YT8614_100BT,     //100Mbps
  YT8614_1000BT,    //1000Mbps
}yt8614_speed_t;
```

3.2.3.6 yt8614_utp_test_mode_1000m_t

```
typedef enum {
  YT8614_1G_TRANS_WAVE_FORM_TEST = 1,      //Test Mode 1, Transmit waveform test
  YT8614_1G_TRANS_JITTER_MASTER_TEST,      //Test Mode 2, Transmit Jitter test (master mode)
  YT8614_1G_TRANS_JITTER_SLAVE_TEST,       //Test Mode 3, Transmit Jitter test (slave mode)
  YT8614_1G_TRANS_DISTORTION_TEST,         //Test Mode 4, Transmit distortion test
}yt8614_utp_test_mode_1000m_t;
```

3.2.3.7 yt8614_utp_test_mode_100m_t

```
typedef enum {
  YT8614_PAIR_0,                           //Test Mode 1, Pair0
  YT8614_PAIR_1,                           //Test Mode 2, Pair1
}yt8614_utp_test_mode_100m_t;
```

3.2.3.8 yt8614_utp_test_mode_10m_t

```
typedef enum {
  YT8614_10M_HARMONIC_TEST = 1,          //Test Mode 1, packet with all ones, 10MHz sine wave, for harmonic test
  YT8614_10M_PSEUDO_RANDOM_TEST,         //Test Mode 2, pseudo random, for TP_idle/Jitter/Different voltage test
  YT8614_10M_NORMAL_LINK_PULSE_TEST,     //Test Mode 3, normal link pulse only
  YT8614_10M_5MHZ_SINE_WAVE_TEST,        //Test Mode 4, 5MHz sine wave
  YT8614_10M_NORMAL_TEST                 //Test Mode 5, Normal mode
}yt8614_utp_test_mode_10m_t;
```

3.2.3.9 yt8614_utp_crossover_config_t

```
typedef enum
```

```
{
  YT8614_CROSSOVER_MDI,        //Manual MDI configuration
  YT8614_CROSSOVER_MDIX,       //Manual MDIX configuration
  YT8614_CROSSOVER_AUTO        //Enable automatic crossover for all modes
}yt8614_utp_crossover_config_t;
```

3.2.3.10 yt8614_utp_optimization_type_t

```
typedef enum{
  YT8614_DIFF_OUTPUT_VOL,     //+/-Vout Differential Output Voltage
  YT8614_RAISE_FALL_TIME_INTER_POLATOR   //rise/fall time inter polator
}yt8614_utp_optimization_type_t;
```

3.2.3.11 yt8614_utp_optimization_data_t

```
typedef struct {
  yt8614_utp_optimization_type_t type;

  union {
       struct{
            yt8614_speed_t speed;
            BOOL is_fine;       //coarse tuning(0)/fine tuning(1)
            u8 data;
       }diff_output_vol_val;

       struct {
            yt8614_speed_t speed;
            u8 data[7];
       }interpolator_val;
  }u_data;
}yt8614_utp_optimization_data_t;
```

3.2.3.12 yt8614_loopback_t

```
typedef enum {
  YT8614_INTERNAL_LOOPBACK, //internal loopback
  YT8614_EXTERNAL_LOOPBACK, //external loopback
  YT8614_REMOTE_LOOPBACK     //remote loopback
}yt8614_loopback_t;
```

3.2.3.13 yt8614_utp_lds_status_t

```
typedef enum {
     YT8614_4pair_LRE_100M_FULL,
     YT8614_1000M_FULL,
     YT8614_1000M_HALF,
     YT8614_100M_FULL,
     YT8614_100M_HALF,
```

```
    YT8614_10M_FULL,
    YT8614_10M_HALF,
    YT8614_UNKNOWN
}yt8614_utp_lds_status_t;
```

3.2.3.14 yt8614_utp_vct_status_t

```
typedef enum {
    YT8614_VCT_OPEN,
    YT8614_VCT_SHORT,
    YT8614_VCT_LOAD,
}yt8614_utp_vct_status_t;
```

3.2.3.15 yt8614_qsgmii_main_emphasis_grade_t

```
/* QSGMII main cursor tuning grade definition */
typedef enum {
  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0000 +0mA=+0mA       */
  YT8614_QSGMII_MAIN_EMPHASIS_0_0_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0001 +0.625mA=+0.625mA */
  YT8614_QSGMII_MAIN_EMPHASIS_0_625_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0010 +1.25mA=+1.25mA   */
  YT8614_QSGMII_MAIN_EMPHASIS_1_25_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0011 +2.5mA=+2.5mA     */
  YT8614_QSGMII_MAIN_EMPHASIS_2_5_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0101 +0.625mA+2.5mA=+3.125mA */
  YT8614_QSGMII_MAIN_EMPHASIS_3_125_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0110 +1.25mA+2.5mA=+3.75mA   */
  YT8614_QSGMII_MAIN_EMPHASIS_3_75_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0111 +2.5mA+2.5mA=+5mA       */
  YT8614_QSGMII_MAIN_EMPHASIS_5_0_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b1101 +0.625mA+5mA=+5.625mA */
  YT8614_QSGMII_MAIN_EMPHASIS_5_625_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b1110 +1.25mA+5mA=+6.25mA   */
  YT8614_QSGMII_MAIN_EMPHASIS_6_25_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b1111 +2.5mA+5mA=+7.5mA     */
  YT8614_QSGMII_MAIN_EMPHASIS_7_5_MA_PLUS,
```

```
}yt8614_qsgmii_main_emphasis_grade_t;
```

3.2.3.16 yt8614_qsgmii_post_emphasis_grade_t

```
/* QSGMII post cursor tuning grade definition */
typedef enum {
  /* QSGMII ext 0xa1 bit[3:1] 3'b000   00:+0mA */
  YT8614_QSGMII_POST_EMPHASIS_0_0_MA_PLUS,

  /* QSGMII ext 0xa1 bit[3:1] 3'b001 +1.25mA */
  YT8614_QSGMII_POST_EMPHASIS_1_25_MA_PLUS,

  /* QSGMII ext 0xa1 bit[3:1] 3'b010 +1.875mA */
  YT8614_QSGMII_POST_EMPHASIS_1_875_MA_PLUS,

  /* QSGMII ext 0xa1 bit[3:1] 3'b011 +2.5mA */
  /* QSGMII ext 0xa1 bit[3:1] 3'b100 +0mA  +2.5mA=+2.5mA */
  YT8614_QSGMII_POST_EMPHASIS_2_5_MA_PLUS,

  /* QSGMII ext 0xa1 bit[3:1] 3'b101 +1.25mA+2.5mA=+3.75mA */
  YT8614_QSGMII_POST_EMPHASIS_3_75_MA_PLUS,

  /* QSGMII ext 0xa1 bit[3:1] 3'b110 +1.875mA+2.5mA=+4.375mA */
  YT8614_QSGMII_POST_EMPHASIS_4_375_MA_PLUS,

  /* QSGMII ext 0xa1 bit[3:1] 3'b111 +2.5mA+2.5mA=+5mA */
  YT8614_QSGMII_POST_EMPHASIS_5_0_MA_PLUS,
}yt8614_qsgmii_post_emphasis_grade_t;
```

3.2.3.17 yt8614_sgmii_main_emphasis_grade_t

```
/* SGMII main cursor tuning grade definition */
typedef enum {
  /* SGMII    ext_reg0xa1 bit[15:12] 4'b0000 +0mA=+0mA       */
  YT8614_SGMII_MAIN_EMPHASIS_0_0_MA_PLUS,

  /* SGMII    ext_reg0xa1 bit[15:12] 4'b0001 +0.625mA=+0.625mA */
  YT8614_SGMII_MAIN_EMPHASIS_0_625_MA_PLUS,

  /* SGMII    ext_reg0xa1 bit[15:12] 4'b0010 +1.25mA=+1.25mA   */
  YT8614_SGMII_MAIN_EMPHASIS_1_25_MA_PLUS,

  /* SGMII    ext_reg0xa1 bit[15:12] 4'b0011 +2.5mA=+2.5mA     */
  YT8614_SGMII_MAIN_EMPHASIS_2_5_MA_PLUS,

  /* SGMII    ext_reg0xa1 bit[15:12] 4'b0101 +0mA+2mA=+2mA       */
```

```
  YT8614_SGMII_MAIN_EMPHASIS_2_0_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b0110 +0.625mA+2mA=+2.625mA */
  YT8614_SGMII_MAIN_EMPHASIS_2_625_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b0111 +1.25mA+2mA=+3.25mA      */
  YT8614_SGMII_MAIN_EMPHASIS_3_25_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b1100 +2.5mA+2mA=+4.5mA    */
  YT8614_SGMII_MAIN_EMPHASIS_4_5_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b1101 +0mA+4mA=+4mA       */
  YT8614_SGMII_MAIN_EMPHASIS_4_0_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b1110 +0.625mA+4mA=+4.625mA */
  YT8614_SGMII_MAIN_EMPHASIS_4_625_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b1111 +1.25mA+4mA=+5.25mA   */
  YT8614_SGMII_MAIN_EMPHASIS_5_25_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b1111 +2.5mA+4mA=+6.5mA    */
  YT8614_SGMII_MAIN_EMPHASIS_6_5_MA_PLUS,
}yt8614_sgmii_main_emphasis_grade_t;
```

3.2.3.18 yt8614_sgmii_post_emphasis_grade_t

```
/* SGMII post cursor tuning grade definition */
typedef enum {
  /* SGMII ext_reg0xa1 bit[3:1] 3'b000 +0mA */
  YT8614_SGMII_POST_EMPHASIS_0_0_MA_PLUS,

  /* SGMII ext_reg0xa1 bit[3:1] 3'b001 +1.25mA */
  /* SGMII ext_reg0xa1 bit[3:1] 3'b100 +0mA+1.25mA=+1.25mA */
  YT8614_SGMII_POST_EMPHASIS_1_25_MA_PLUS,

  /* SGMII ext_reg0xa1 bit[3:1] 3'b010 +1.875mA */
  YT8614_SGMII_POST_EMPHASIS_1_875_MA_PLUS,

  /* SGMII ext_reg0xa1 bit[3:1] 3'b011 +2.5mA */
  /* SGMII ext_reg0xa1 bit[3:1] 3'b101 +1.25mA+1.25mA=+2.5mA */
  YT8614_SGMII_POST_EMPHASIS_2_5_MA_PLUS,

  /* SGMII ext_reg0xa1 bit[3:1] 3'b110 +1.875mA+1.25mA=+3.125mA */
  YT8614_SGMII_POST_EMPHASIS_3_125_MA_PLUS,
```

```
  /* SGMII ext_reg0xa1 bit[3:1] 3'b111 +2.5mA+1.25mA=+3.75mA */
  YT8614_SGMII_POST_EMPHASIS_3_75_MA_PLUS,
}yt8614_sgmii_post_emphasis_grade_t;
```

3.2.3.19 yt8614_pkg_gen_checker_dir_t

```
typedef enum {
  YT8614_DOWN_STREAM,    //down stream direction
  YT8614_UP_STREAM,      //up stream direction
}yt8614_pkg_gen_checker_dir_t;
```

3.2.3.20 yt8614_pkg_gen_t

```
typedef struct {
  yt8614_pkg_gen_checker_dir_t dir; //direction
  u16 pkg_len;                      //package length
  u16 pkg_cnt;                      //package count
  u8   ipg;                         //inter package gap
  BOOL da_sa;                       //DA/SA enable or disable
  u8 da;                            //DA - destination address
  u8 sa;                            //SA - source address
}yt8614_pkg_gen_t;
```

3.2.3.21 yt8614 API 返回值定义

```
enum ytphy_8614_errno_e {
    YTSYS_ERR_NONE,
    YTSYS_ERR_PARAM,
    YTSYS_ERR_UNKNOWN,
    YTSYS_ERR_MAX
};
```

所有 API 正常返回值为 YTSYS_ERR_NONE(0)，非 0 为异常。

3.2.4 yt8614Q 数据结构

3.2.4.1 yt8614Q_phy_dev_info

用于描述每个 YT8614Q 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 数目的定义见宏：MPHY_YT8614Q_CHIP_NUM。

```
#define MPHY_YT8614Q_CHIP_NUM      1

typedef struct yt8614Q_phy_dev_info {
    unsigned int lport;             //chip 内部端口 index
    unsigned int phy_addr;          //端口实际 phy address(base_phy_addr + lport)
    unsigned int base_phy_addr;     //chip 端口 phy 基地址
    s32 (*read)(struct yt8614Q_phy_dev_info *info, struct yt8614Q_phy_reg *param);//读寄存器接口
    s32 (*write)(struct yt8614Q_phy_dev_info *info, struct yt8614Q_phy_reg *param);//写寄存器接口
} yt8614Q_phy_dev_info_t;
```

读写寄存器接口说明：这两个函数的作用：

1) 统一寄存器的访问。因为 YT8614Q 的寄存器种类多，通过这上面两个函数统一调用。
2) 保证读写寄存器的原子性以实现寄存器读写的同步。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。

3.2.4.2 yt8614Q_phy_reg

寄存器属性定义结构体。

```
struct yt8614Q_phy_reg {
    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(8614Q no mmd register)
};
```

- 寄存器类型

因为 YT8614Q 支持 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

这样组合下来，共有 5 种寄存器类型如下：

```
#define YT8614Q_TYPE_COMMON          0x01 //公共寄存器，不需要地址空间切换，随时访问。
#define YT8614Q_TYPE_SDS_QSGMII_MII 0x06 //QSGMII 地址空间 MII 寄存器。
#define YT8614Q_TYPE_SDS_SGMII_MII   0x07 //Fiber/SGMII 地址空间 MII 寄存器。
#define YT8614Q_TYPE_SDS_QSGMII_EXT0x08 //QSGMII 地址空间 EXT 寄存器。
#define YT8614Q_TYPE_SDS_SGMII_EXT  0x09 //Fiber/SGMII 地址空间 EXT 寄存器。
```

3.2.4.3 yt8614Q_polarity_dir_t

```
typedef enum {
  YT8614Q_SND,
  YT8614Q_RCV
}yt8614Q_polarity_dir_t;
```

3.2.4.4 yt8614Q_speed_t

```
typedef enum
{
  YT8614Q_10BT,    //10Mbps
  YT8614Q_100BT,  //100Mbps
  YT8614Q_1000BT, //1000Mbps
}yt8614Q_speed_t;
```

3.2.4.5 yt8614Q_loopback_t

```
typedef enum {
  YT8614Q_INTERNAL_LOOPBACK,     //internal loopback
  YT8614Q_EXTERNAL_LOOPBACK,     //external loopback
```

```
  YT8614Q_REMOTE_LOOPBACK        //remote loopback
}yt8614Q_loopback_t;
```

3.2.4.6 yt8614Q_qsgmii_main_emphasis_grade_t

```
/* QSGMII main cursor tuning grade definition */
typedef enum {
  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0000 +0mA=+0mA       */
  YT8614Q_QSGMII_MAIN_EMPHASIS_0_0_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0001 +0.625mA=+0.625mA */
  YT8614Q_QSGMII_MAIN_EMPHASIS_0_625_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0010 +1.25mA=+1.25mA   */
  YT8614Q_QSGMII_MAIN_EMPHASIS_1_25_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0011 +2.5mA=+2.5mA     */
  YT8614Q_QSGMII_MAIN_EMPHASIS_2_5_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0101 +0.625mA+2.5mA=+3.125mA */
  YT8614Q_QSGMII_MAIN_EMPHASIS_3_125_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0110 +1.25mA+2.5mA=+3.75mA   */
  YT8614Q_QSGMII_MAIN_EMPHASIS_3_75_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0111 +2.5mA+2.5mA=+5mA       */
  YT8614Q_QSGMII_MAIN_EMPHASIS_5_0_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b1101 +0.625mA+5mA=+5.625mA */
  YT8614Q_QSGMII_MAIN_EMPHASIS_5_625_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b1110 +1.25mA+5mA=+6.25mA   */
  YT8614Q_QSGMII_MAIN_EMPHASIS_6_25_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b1111 +2.5mA+5mA=+7.5mA     */
  YT8614Q_QSGMII_MAIN_EMPHASIS_7_5_MA_PLUS,
}yt8614Q_qsgmii_main_emphasis_grade_t;
```

3.2.4.7 yt8614Q_qsgmii_post_emphasis_grade_t

```
/* QSGMII post cursor tuning grade definition */
typedef enum {
  /* QSGMII ext 0xa1 bit[3:1] 3'b000   00:+0mA */
  YT8614Q_QSGMII_POST_EMPHASIS_0_0_MA_PLUS,

  /* QSGMII ext 0xa1 bit[3:1] 3'b001 +1.25mA */
```

```
  YT8614Q_QSGMII_POST_EMPHASIS_1_25_MA_PLUS,

  /* QSGMII ext 0xa1 bit[3:1] 3'b010 +1.875mA */
  YT8614Q_QSGMII_POST_EMPHASIS_1_875_MA_PLUS,

  /* QSGMII ext 0xa1 bit[3:1] 3'b011 +2.5mA */
  /* QSGMII ext 0xa1 bit[3:1] 3'b100 +0mA+2.5mA=+2.5mA */
  YT8614Q_QSGMII_POST_EMPHASIS_2_5_MA_PLUS,

  /* QSGMII ext 0xa1 bit[3:1] 3'b101 +1.25mA+2.5mA=+3.75mA */
  YT8614Q_QSGMII_POST_EMPHASIS_3_75_MA_PLUS,

  /* QSGMII ext 0xa1 bit[3:1] 3'b110 +1.875mA+2.5mA=+4.375mA */
  YT8614Q_QSGMII_POST_EMPHASIS_4_375_MA_PLUS,

  /* QSGMII ext 0xa1 bit[3:1] 3'b111 +2.5mA+2.5mA=+5mA */
  YT8614Q_QSGMII_POST_EMPHASIS_5_0_MA_PLUS,
}yt8614Q_qsgmii_post_emphasis_grade_t;
```

3.2.4.8 yt8614Q_sgmii_main_emphasis_grade_t

```
/* SGMII main cursor tuning grade definition */
typedef enum {
  /* SGMII   ext_reg0xa1 bit[15:12] 4'b0000 +0mA=+0mA       */
  YT8614Q_SGMII_MAIN_EMPHASIS_0_0_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b0001 +0.625mA=+0.625mA */
  YT8614Q_SGMII_MAIN_EMPHASIS_0_625_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b0010 +1.25mA=+1.25mA   */
  YT8614Q_SGMII_MAIN_EMPHASIS_1_25_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b0011 +2.5mA=+2.5mA     */
  YT8614Q_SGMII_MAIN_EMPHASIS_2_5_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b0101 +0mA+2mA=+2mA       */
  YT8614Q_SGMII_MAIN_EMPHASIS_2_0_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b0110 +0.625mA+2mA=+2.625mA */
  YT8614Q_SGMII_MAIN_EMPHASIS_2_625_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b0111 +1.25mA+2mA=+3.25mA       */
  YT8614Q_SGMII_MAIN_EMPHASIS_3_25_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b1100 +2.5mA+2mA=+4.5mA     */
```

```
  YT8614Q_SGMII_MAIN_EMPHASIS_4_5_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b1101 +0mA+4mA=+4mA       */
  YT8614Q_SGMII_MAIN_EMPHASIS_4_0_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b1110 +0.625mA+4mA=+4.625mA */
  YT8614Q_SGMII_MAIN_EMPHASIS_4_625_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b1111 +1.25mA+4mA=+5.25mA   */
  YT8614Q_SGMII_MAIN_EMPHASIS_5_25_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b1111 +2.5mA+4mA=+6.5mA     */
  YT8614Q_SGMII_MAIN_EMPHASIS_6_5_MA_PLUS,
}yt8614Q_sgmii_main_emphasis_grade_t;
```

3.2.4.9 yt8614Q_sgmii_post_emphasis_grade_t

```
/* SGMII post cursor tuning grade definition */
typedef enum {
  /* SGMII ext_reg0xa1 bit[3:1] 3'b000 +0mA */
  YT8614Q_SGMII_POST_EMPHASIS_0_0_MA_PLUS,

  /* SGMII ext_reg0xa1 bit[3:1] 3'b001 +1.25mA */
  /* SGMII ext_reg0xa1 bit[3:1] 3'b100 +0mA+1.25mA=+1.25mA */
  YT8614Q_SGMII_POST_EMPHASIS_1_25_MA_PLUS,

  /* SGMII ext_reg0xa1 bit[3:1] 3'b010 +1.875mA */
  YT8614Q_SGMII_POST_EMPHASIS_1_875_MA_PLUS,

  /* SGMII ext_reg0xa1 bit[3:1] 3'b011 +2.5mA */
  /* SGMII ext_reg0xa1 bit[3:1] 3'b101 +1.25mA+1.25mA=+2.5mA */
  YT8614Q_SGMII_POST_EMPHASIS_2_5_MA_PLUS,

  /* SGMII ext_reg0xa1 bit[3:1] 3'b110 +1.875mA+1.25mA=+3.125mA */
  YT8614Q_SGMII_POST_EMPHASIS_3_125_MA_PLUS,

  /* SGMII ext_reg0xa1 bit[3:1] 3'b111 +2.5mA+1.25mA=+3.75mA */
  YT8614Q_SGMII_POST_EMPHASIS_3_75_MA_PLUS,
}yt8614Q_sgmii_post_emphasis_grade_t;
```

3.2.4.10 yt8614Q_pkg_gen_checker_dir_t

```
typedef enum {
  YT8614Q_DOWN_STREAM,      //down stream direction
  YT8614Q_UP_STREAM,        //up stream direction
}yt8614Q_pkg_gen_checker_dir_t;
```

3.2.4.11 yt8614Q_pkg_gen_t

```
typedef struct {
  yt8614Q_pkg_gen_checker_dir_t dir;     //direction
  u16 pkg_len;                           //package length
  u16 pkg_cnt;                           //package count
  u8   ipg;                              //inter package gap
  BOOL da_sa;                            //DA/SA enable or disable
  u8 da;                                 //DA - destination address
  u8 sa;                                 //SA - source address
}yt8614Q_pkg_gen_t;
```

3.2.4.12 yt8614Q API 返回值定义

```
 enum ytphy_8614Q_errno_e {
        YTSYS_ERR_NONE,
        YTSYS_ERR_PARAM,
        YTSYS_ERR_UNKNOWN,
        YTSYS_ERR_MAX
 };
```

所有 API 正常返回值为 YTSYS_ERR_NONE(0)，非 0 为异常。

3.2.5 yt8618 数据结构

3.2.5.1 yt8618_phy_dev_info

用于描述每个 YT8618 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 数目的定义见宏：MPHY_YT8618_CHIP_NUM。

```
#define MPHY_YT8618_CHIP_NUM        1

typedef struct yt8618_phy_dev_info {
      unsigned int lport;              //chip  内部端口 index
      unsigned int phy_addr;           //端口实际 phy address(base_phy_addr + lport)
      unsigned int base_phy_addr;      //chip 端口 phy 基地址
      s32 (*read)(struct yt8618_phy_dev_info *info, struct yt8618_phy_reg *param);//读寄存器接口
      s32 (*write)(struct yt8618_phy_dev_info *info, struct yt8618_phy_reg *param);//写寄存器接口
} yt8618_phy_dev_info_t;
```

读写寄存器接口说明：

1) 统一寄存器的访问。因为 YT8618 的寄存器种类多，通过上面这两个函数统一调用。
2) 保证读写寄存器的原子性以实现寄存器读写的同步。寄存器访问需要好几步骤才能完成，通过这两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。

3.2.5.2 yt8618_phy_reg

寄存器属性定义结构体。

```
struct yt8618_phy_reg {
      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
};
```

- 寄存器类型

因为 YT8618 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)寄存器和 EXT(扩展)寄存器，其中 UTP 还包括 MMD 寄存器和 LDS 寄存器。

这样组合下来，共有 9 种寄存器类型如下：

```
#define YT8618_TYPE_COMMON          0x01 //公共寄存器，不需要地址空间切换，随时访问。
#define YT8618_TYPE_UTP_MII         0x02 //电口地址空间 MII 寄存器。
#define YT8618_TYPE_UTP_EXT         0x03 //电口地址空间 EXT 寄存器。
#define YT8618_TYPE_LDS_MII         0x04 //光口地址空间 MII 寄存器。
#define YT8618_TYPE_UTP_MMD         0x05 //电口地址空间 MMD 寄存器。
#define YT8618_TYPE_SDS_QSGMII_MII  0x06 //QSGMII 地址空间 MII 寄存器。
#define YT8618_TYPE_SDS_QSGMII_EXT  0x08 //QSGMII 地址空间 EXT 寄存器。
```

3.2.5.3 yt8618_polarity_dir_t

```
typedef enum {
    YT8618_SND,      //发送方向
    YT8618_RCV,      //接收方向
}yt8618_polarity_dir_t;
```

3.2.5.4 yt8618_utp_vct_t

```
typedef enum
{
    YT8618_VCT_INTER_BOARD,         //板间 vct 测试
    YT8618_VCT_NON_INTER_BOARD      //非板间 vct 测试
}yt8618_utp_vct_t;
```

3.2.5.5 yt8618_speed_t

```
typedef enum
{
    YT8618_10BT,      //10Mbps
    YT8618_100BT,     //100Mbps
    YT8618_1000BT,    //1000Mbps
}yt8618_speed_t;
```

3.2.5.6 yt8618_utp_test_mode_1000m_t

```
typedef enum {
    YT8618_1G_TRANS_WAVE_FORM_TEST = 1,     //Test Mode 1, Transmit waveform test
    YT8618_1G_TRANS_JITTER_MASTER_TEST,     //Test Mode 2, Transmit Jitter test (master mode)
    YT8618_1G_TRANS_JITTER_SLAVE_TEST,      //Test Mode 3, Transmit Jitter test (slave mode)
```

```
  YT8618_1G_TRANS_DISTORTION_TEST,          //Test Mode 4, Transmit distortion test
}yt8618_utp_test_mode_1000m_t;
```

3.2.5.7 yt8618_utp_test_mode_100m_t

```
typedef enum {
  YT8618_PAIR_0,                            //Test Mode 1, Pair0
  YT8618_PAIR_1,                            //Test Mode 2, Pair1
}yt8618_utp_test_mode_100m_t;
```

3.2.5.8 yt8618_utp_test_mode_10m_t

```
typedef enum {
  YT8618_10M_HARMONIC_TEST = 1,         //Test Mode 1, packet with all ones, 10MHz sine wave, for harmonic test
  YT8618_10M_PSEUDO_RANDOM_TEST,        //Test Mode 2, pseudo random, for TP_idle/Jitter/Different voltage test
  YT8618_10M_NORMAL_LINK_PULSE_TEST,    //Test Mode 3, normal link pulse only
  YT8618_10M_5MHZ_SINE_WAVE_TEST,       //Test Mode 4, 5MHz sine wave
  YT8618_10M_NORMAL_TEST                //Test Mode 5, Normal mode
}yt8618_utp_test_mode_10m_t;
```

3.2.5.9 yt8618_utp_crossover_config_t

```
typedef enum
{
  YT8618_CROSSOVER_MDI,       //Manual MDI configuration
  YT8618_CROSSOVER_MDIX,      //Manual MDIX configuration
  YT8618_CROSSOVER_AUTO       //Enable automatic crossover for all modes
}yt8618_utp_crossover_config_t;
```

3.2.5.10 yt8618_utp_optimization_type_t

```
typedef enum{
  YT8618_DIFF_OUTPUT_VOL,     //+/-Vout Differential Output Voltage
  YT8618_RAISE_FALL_TIME_INTER_POLATOR   //rise/fall time inter polator
}yt8618_utp_optimization_type_t;
```

3.2.5.11 yt8618_utp_optimization_data_t

```
typedef struct {
  yt8618_utp_optimization_type_t type;

  union {
      struct{
          yt8618_speed_t speed;
          BOOL is_fine;       //coarse tuning(0)/fine tuning(1)
          u8 data;
      }diff_output_vol_val;

      struct {
```

```
            yt8618_speed_t speed;
            u8 data[7];
        }interpolator_val;
  }u_data;
}yt8618_utp_optimization_data_t;
```

3.2.5.12 yt8618_loopback_t

```
typedef enum {
  YT8618_INTERNAL_LOOPBACK, //internal loopback
  YT8618_EXTERNAL_LOOPBACK,//external loopback
  YT8618_REMOTE_LOOPBACK    //remote loopback
}yt8618_loopback_t;
```

3.2.5.13 yt8618_utp_lds_status_t

```
typedef enum {
    YT8618_4pair_LRE_100M_FULL,
    YT8618_1000M_FULL,
    YT8618_1000M_HALF,
    YT8618_100M_FULL,
    YT8618_100M_HALF,
    YT8618_10M_FULL,
    YT8618_10M_HALF,
    YT8618_UNKNOWN
}yt8618_utp_lds_status_t;
```

3.2.5.14 yt8618_utp_vct_status_t

```
typedef enum {
    YT8618_VCT_OPEN,
    YT8618_VCT_SHORT,
    YT8618_VCT_LOAD,
}yt8618_utp_vct_status_t;
```

3.2.5.15 yt8618_qsgmii_main_emphasis_grade_t

```
/* QSGMII main cursor tuning grade definition */
typedef enum {
  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0000 +0mA=+0mA       */
  YT8618_QSGMII_MAIN_EMPHASIS_0_0_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0001 +0.625mA=+0.625mA */
  YT8618_QSGMII_MAIN_EMPHASIS_0_625_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0010 +1.25mA=+1.25mA   */
  YT8618_QSGMII_MAIN_EMPHASIS_1_25_MA_PLUS,
```

```
  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0011 +2.5mA=+2.5mA     */
  YT8618_QSGMII_MAIN_EMPHASIS_2_5_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0101 +0.625mA+2.5mA=+3.125mA */
  YT8618_QSGMII_MAIN_EMPHASIS_3_125_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0110 +1.25mA+2.5mA=+3.75mA   */
  YT8618_QSGMII_MAIN_EMPHASIS_3_75_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b0111 +2.5mA+2.5mA=+5mA       */
  YT8618_QSGMII_MAIN_EMPHASIS_5_0_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b1101 +0.625mA+5mA=+5.625mA */
  YT8618_QSGMII_MAIN_EMPHASIS_5_625_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b1110 +1.25mA+5mA=+6.25mA   */
  YT8618_QSGMII_MAIN_EMPHASIS_6_25_MA_PLUS,

  /* QSGMII ext_reg0xa1 bit[15:12] 4'b1111 +2.5mA+5mA=+7.5mA     */
  YT8618_QSGMII_MAIN_EMPHASIS_7_5_MA_PLUS,
}yt8618_qsgmii_main_emphasis_grade_t;
```

3.2.5.16 yt8618_qsgmii_post_emphasis_grade_t

```
/* QSGMII post cursor tuning grade definition */
typedef enum {
  /* QSGMII ext 0xa1 bit[3:1] 3'b000   00:+0mA */
  YT8618_QSGMII_POST_EMPHASIS_0_0_MA_PLUS,

  /* QSGMII ext 0xa1 bit[3:1] 3'b001 +1.25mA */
  YT8618_QSGMII_POST_EMPHASIS_1_25_MA_PLUS,

  /* QSGMII ext 0xa1 bit[3:1] 3'b010 +1.875mA */
  YT8618_QSGMII_POST_EMPHASIS_1_875_MA_PLUS,

  /* QSGMII ext 0xa1 bit[3:1] 3'b011 +2.5mA */
  /* QSGMII ext 0xa1 bit[3:1] 3'b100 +0mA  +2.5mA=+2.5mA */
  YT8618_QSGMII_POST_EMPHASIS_2_5_MA_PLUS,

  /* QSGMII ext 0xa1 bit[3:1] 3'b101 +1.25mA+2.5mA=+3.75mA */
  YT8618_QSGMII_POST_EMPHASIS_3_75_MA_PLUS,

  /* QSGMII ext 0xa1 bit[3:1] 3'b110 +1.875mA+2.5mA=+4.375mA */
  YT8618_QSGMII_POST_EMPHASIS_4_375_MA_PLUS,
```

```
  /* QSGMII ext 0xa1 bit[3:1] 3'b111 +2.5mA+2.5mA=+5mA */
  YT8618_QSGMII_POST_EMPHASIS_5_0_MA_PLUS,
}yt8618_qsgmii_post_emphasis_grade_t;
```

3.2.5.17 yt8618_sgmii_main_emphasis_grade_t

```
/* SGMII main cursor tuning grade definition */
typedef enum {
  /* SGMII   ext_reg0xa1 bit[15:12] 4'b0000 +0mA=+0mA       */
  YT8618_SGMII_MAIN_EMPHASIS_0_0_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b0001 +0.625mA=+0.625mA */
  YT8618_SGMII_MAIN_EMPHASIS_0_625_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b0010 +1.25mA=+1.25mA   */
  YT8618_SGMII_MAIN_EMPHASIS_1_25_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b0011 +2.5mA=+2.5mA     */
  YT8618_SGMII_MAIN_EMPHASIS_2_5_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b0101 +0mA+2mA=+2mA       */
  YT8618_SGMII_MAIN_EMPHASIS_2_0_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b0110 +0.625mA+2mA=+2.625mA */
  YT8618_SGMII_MAIN_EMPHASIS_2_625_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b0111 +1.25mA+2mA=+3.25mA       */
  YT8618_SGMII_MAIN_EMPHASIS_3_25_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b1100 +2.5mA+2mA=+4.5mA     */
  YT8618_SGMII_MAIN_EMPHASIS_4_5_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b1101 +0mA+4mA=+4mA       */
  YT8618_SGMII_MAIN_EMPHASIS_4_0_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b1110 +0.625mA+4mA=+4.625mA */
  YT8618_SGMII_MAIN_EMPHASIS_4_625_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b1111 +1.25mA+4mA=+5.25mA   */
  YT8618_SGMII_MAIN_EMPHASIS_5_25_MA_PLUS,

  /* SGMII   ext_reg0xa1 bit[15:12] 4'b1111 +2.5mA+4mA=+6.5mA     */
  YT8618_SGMII_MAIN_EMPHASIS_6_5_MA_PLUS,
}yt8618_sgmii_main_emphasis_grade_t;
```

3.2.5.18 yt8618_sgmii_post_emphasis_grade_t

```
/* SGMII post cursor tuning grade definition */
typedef enum {
  /* SGMII ext_reg0xa1 bit[3:1] 3'b000 +0mA */
  YT8618_SGMII_POST_EMPHASIS_0_0_MA_PLUS,

  /* SGMII ext_reg0xa1 bit[3:1] 3'b001 +1.25mA */
  /* SGMII ext_reg0xa1 bit[3:1] 3'b100 +0mA+1.25mA=+1.25mA */
  YT8618_SGMII_POST_EMPHASIS_1_25_MA_PLUS,

  /* SGMII ext_reg0xa1 bit[3:1] 3'b010 +1.875mA */
  YT8618_SGMII_POST_EMPHASIS_1_875_MA_PLUS,

  /* SGMII ext_reg0xa1 bit[3:1] 3'b011 +2.5mA */
  /* SGMII ext_reg0xa1 bit[3:1] 3'b101 +1.25mA+1.25mA=+2.5mA */
  YT8618_SGMII_POST_EMPHASIS_2_5_MA_PLUS,

  /* SGMII ext_reg0xa1 bit[3:1] 3'b110 +1.875mA+1.25mA=+3.125mA */
  YT8618_SGMII_POST_EMPHASIS_3_125_MA_PLUS,

  /* SGMII ext_reg0xa1 bit[3:1] 3'b111 +2.5mA+1.25mA=+3.75mA */
  YT8618_SGMII_POST_EMPHASIS_3_75_MA_PLUS,
}yt8618_sgmii_post_emphasis_grade_t;
```

3.2.5.19 yt8618_pkg_gen_checker_dir_t

```
typedef enum {
  YT8618_DOWN_STREAM,    //down stream direction
  YT8618_UP_STREAM,      //up stream direction
}yt8618_pkg_gen_checker_dir_t;
```

3.2.5.20 yt8618_pkg_gen_t

```
typedef struct {
  yt8618_pkg_gen_checker_dir_t dir; //direction
  u16 pkg_len;                      //package length
  u16 pkg_cnt;                      //package count
  u8   ipg;                         //inter package gap
  BOOL da_sa;                       //DA/SA enable or disable
  u8 da;                            //DA - destination address
  u8 sa;                            //SA - source address
}yt8618_pkg_gen_t;
```

3.2.5.21 yt8618 API 返回值定义

```
enum ytphy_8618_errno_e {
  YTSYS_ERR_NONE,
```

```
        YTSYS_ERR_PARAM,
        YTSYS_ERR_UNKNOWN,
        YTSYS_ERR_MAX
    };
```

所有 API 正常返回值为 YTSYS_ERR_NONE(0)，非 0 为异常。

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 yt8614_phy_dev_info *info, struct yt8614_phy_reg *param);
s32 yt8614_write_reg(struct yt8614_phy_dev_info *info, struct yt8614_phy_reg *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, u8 type);

功能：软复位指定端口
参数：lport-输入：全局端口号
type-输入：SOFT_RESET_UTP(0)/SOFT_RESET_QSGMII(1)/SOFT_RESET_SGMII(2)

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(TRUE)
参数：lport-输入：全局端口号
enable: powerdown(FALSE)/正常(TRUE)

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(FALSE)/正常(TRUE)

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光纤单向使能/禁用
参数：lport-输入：全局端口号
speed-输入: SPEED_1000M(2)/SPEED_100M(1)
enable-输入：禁用(FALSE)/使能(TRUE)

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(FALSE)/使能(TRUE)

4.1.8 s32 yt8614_fiber_speed_set(u32 lport, int fiber_speed);

功能：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: SPEED_1000M（2）/SPEED_100M(1)

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(FALSE)/使能(TRUE)

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(FALSE)/使能(TRUE)

4.1.11 s32 yt8614_qsgmii_sgmii_link_status_get(u32 lport, BOOL if_qsgmii, BOOL *linkup);

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(TRUE), SGMII(FALSE)

linkup-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, BOOL 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(FALSE)/光口优先(TRUE)

4.1.13 int yt8614_combo_media_priority_get(u32 lport, BOOL *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(FALSE)/光口优先(TRUE)

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(FALSE)/使能(TRUE)

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(FALSE)/使能(TRUE)

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 CAP_10HALF      (1 << 0)  /* capability 10Mbps Half duplex */
#define CAP_10FULL      (1 << 1)  /* capability 10Mbps Full duplex */
#define CAP_100HALF     (1 << 2)  /* capability 100Mbps Half duplex */
#define CAP_100FULL     (1 << 3)  /* capability 100Mbps Full duplex */
#define CAP_1000HALF    (1 << 4)  /* capability 1000Mbps Half duplex */
#define CAP_1000FULL    (1 << 5)  /* capability 1000Mbps Full duplex */
#define CAP_PAUSE       (1 << 6)  /* capability pause */
```

#define CAP_ASYM_PAUSE (1 << 7) /* capability Asymmetric pause */

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(FALSE)/full duplex(TRUE)

4.1.19 s32 yt8614_utp_force_duplex_get(u32 lport, unsigned int *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)/DUPLEX_NO_SENSE(0xffff 自协商使能)

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_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_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_chip_hard_reset(unsigned int chip_no);

功能：phy 芯片硬复位，硬复位后，芯片所有寄存器都恢复至 default 值

参数：chip_no-输入

4.1.24 4.1.24 int yt8614_utp_SNR_read(u32 lport, u8 *snr0, u8 *snr1, u8 *snr2, u8 *snr3);

功能：utp SNR(Signal Noise Ratio) read. 可以通过返回值判定当前信号质量

参数：lport-输入：全局端口号

snr0-输出：utp 第一对线 snr

snr1-输出：utp 第二对线 snr

snr2-输出：utp 第三对线 snr

snr3-输出：utp 第四对线 snr

说明：1000Mbps 下返回四对线 snr，100Mbps 下仅仅返回一对线的 snr

4.1.25 4.1.25 s32 yt8614_utp_fast_link_down_set(u32 lport, BOOL enable);

功能：utp fast link down 配置

参数：lport-输入：全局端口号

enable: 禁用(FALSE)/使能(TRUE)

4.1.26 s32 yt8614_utp_fast_link_down_get(u32 lport, BOOL *enable);

功能：获取当前 utp fast link down 状态

参数：lport-输入：全局端口号

enable-输出：禁用(FALSE)/使能(TRUE)

4.1.27 s32 yt8614_utp_smart_downgrade_set(u32 lport, BOOL enable);

功能：utp speed smart downgrade 配置

参数：lport-输入：全局端口号

enable：禁用(FALSE)/使能(TRUE)

4.1.28 s32 yt8614_utp_smart_downgrade_get(u32 lport, BOOL *enable);

功能：获取当前 utp speed smart downgrade 状态

参数：lport-输入：全局端口号

enable-输出：禁用(FALSE)/使能(TRUE)

4.1.29 s32 yt8614_utp_eee_set(u32 lport, BOOL enable);

功能：utp eee 配置

参数：lport-输入：全局端口号

enable：禁用(FALSE)/使能(TRUE)

4.1.30 s32 yt8614_utp_eee_get(u32 lport, BOOL *enable);

功能：获取当前 utp eee 配置状态

参数：lport-输入：全局端口号

enable-输出：禁用(FALSE-not in EEE)/使能(TRUE-in EEE)

4.1.31 s32 yt8614_utp_lds_set(u32 lport, BOOL enable);

功能：4 对线百兆长距离配置

参数：lport-输入：全局端口号

enable：禁用(FALSE)/使能(TRUE)

4.1.32 s32 yt8614_utp_lds_get_status(u32 lport, yt8614_utp_lds_status_t *status);

功能：4 对线百兆长距离状态获取

参数：lport-输入：全局端口号

status-输出：

YT8614_4pair_LRE_100M_FULL(0)/YT8614_1000M_FULL(1)/
YT8614_1000M_HALF(2)/YT8614_100M_FULL(3)/
YT8614_100M_HALF(4)/YT8614_10M_FULL(5)/YT8614_10M_HALF(6)/
YT8614_UNKNOWN(7)

4.1.33 s32 yt8614_utp_sleep_set(u32 lport, BOOL enable);

功能：utp sleep 配置

参数：lport-输入：全局端口号

enable：禁用(FALSE)/使能(TRUE)

4.1.34 s32 yt8614_utp_sleep_get(u32 lport, BOOL *enable);

功能：获取当前 utp sleep 配置状态

参数：lport-输入：全局端口号

enable-输出：禁用(FALSE)/使能(TRUE)

4.1.35 s32 yt8614_utp_vct(u32 lport, yt8614_utp_vct_t type, yt8614_utp_vct_status_t* status, u16 *channel0, u16 *channel1, u16 *channel2, u16 *channel3);

功能：utp vct 测试
参数：lport-输入：全局端口号
type: YT8614_VCT_INTER_BOARD(0)/YT8614_VCT_NON_INTER_BOARD(1)
status--输出：YT8614_VCT_OPEN(0)/YT8614_VCT_SHORT(1)/YT8614_VCT_LOAD(2)
channel0-输出：第一对线出问题的地点(单位 cm)
channel1-输出：第二对线出问题的地点(单位 cm)
channel2-输出：第三对线出问题的地点(单位 cm)
channel3-输出：第四对线出问题的地点(单位 cm)
说明：当 status 为 YT8614_VCT_LOAD，返回值 channel0~channel3 无意义

4.1.36 s32 yt8614_utp_crossover(u32 lport, yt8614_utp_crossover_config_t type);

功能：utp crossover 配置
参数：lport-输入：全局端口号
type: YT8614_CROSSOVER_MDI(0)/YT8614_CROSSOVER_MDIX(1)
/YT8614_CROSSOVER_AUTO(2)

4.1.37 s32 yt8614_utp_optimization(u32 lport, yt8614_utp_optimization_data_t odata);

功能：utp 电口指标优化
参数：lport-输入：全局端口号
odata-优化指标(详细参见 yt8614_optimization_data_t 说明)

4.1.38 s32 yt8614_utp_test_mode1000M(u32 lport, yt8614_utp_test_mode_1000m_t type);;

功能：utp template 测试
参数：lport-输入：全局端口号
type-测试类别(详细参见 yt8614_utp_test_mode_1000m_t 说明)

4.1.39 s32 yt8614_utp_test_mode100M(u32 lport, yt8614_utp_test_mode_100m_t type);

功能：utp template 测试
参数：lport-输入：全局端口号
type-测试类别(详细参见 yt8614_utp_test_mode_100m_t 说明)

4.1.40 s32 yt8614_utp_test_mode10M(u32 lport, yt8614_utp_test_mode_10m_t type);

功能：utp template 测试
参数：lport-输入：全局端口号
type-测试类别(详细参见 yt8614_utp_test_mode_10m_t 说明)

4.1.41 s32 yt8614_qsgmii_polarity_reverse(u32 lport, yt8614_polarity_dir_t dir);

功能：qsgmii 极性反转配置
参数：lport-输入：全局端口号
dir-收发方向指定，YT8614_SND(0)/YT8614_RCV(1)

4.1.42 s32 yt8614_sgmii_polarity_reverse(u32 lport, yt8614_polarity_dir_t dir)

功能：sgmii 极性反转配置
参数：lport-输入：全局端口号

dir-收发方向指定，YT8614_SND(0)/YT8614_RCV(1)

4.1.43 s32 yt8614_qsgmii_loopback(u32 lport, yt8614_loopback_t type, BOOL enable);

功能：qsgmii 环回配置

参数：lport-输入：全局端口号

type-环回类型指定: YT8614_INTERNAL_LOOPBACK(0)/YT8614_EXTERNAL_LOOPBACK(1)
/YT8614_REMOTE_LOOPBACK(2)

enable: 禁用(FALSE)/使能(TRUE)

4.1.44 s32 yt8614_sgmii_loopback(u32 lport, yt8614_loopback_t type, BOOL enable);

功能：sgmii 环回配置

参数：lport-输入：全局端口号

type-环回类型指定: YT8614_INTERNAL_LOOPBACK(0)/YT8614_EXTERNAL_LOOPBACK(1)
/YT8614_REMOTE_LOOPBACK(2)

enable: 禁用(FALSE)/使能(TRUE)

4.1.45 s32 yt8614_qsgmii_main_emphasis(u32 lport, yt8614_qsgmii_main_emphasis_grade_t grade);

功能：qsgmii main cursor 测试

参数：lport-输入：全局端口号

grade-加重等级(详细参见 yt8614_qsgmii_main_emphasis_grade_t 说明)

4.1.46 s32 yt8614_qsgmii_post_emphasis(u32 lport, yt8614_qsgmii_post_emphasis_grade_t grade);

功能：qsgmii post cursor 测试

参数：lport-输入：全局端口号

grade-加重等级(详细参见 yt8614_qsgmii_post_emphasis_grade_t 说明)

4.1.47 s32 yt8614_sgmii_main_emphasis(u32 lport, yt8614_sgmii_main_emphasis_grade_t grade);

功能：sgmii main cursor 测试

参数：lport-输入：全局端口号

grade-加重等级(详细参见 yt8614_sgmii_main_emphasis_grade_t 说明)

4.1.48 s32 yt8614_sgmii_post_emphasis(u32 lport, yt8614_sgmii_post_emphasis_grade_t grade);

功能：sgmii post cursor 测试

参数：lport-输入：全局端口号

grade-加重等级(详细参见 yt8614_sgmii_post_emphasis_grade_t 说明)

4.1.49 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(FALSE)/使能(TRUE)

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

4.1.50 s32 yt8614_app_set_chip_base_phy_addr(unsigned int chip_no, unsigned int base_phy_addr)

功能：phy base address 配置

参数：**chip_no**-输入

4.1.51 unsigned int yt8614_app_get_chip_base_phy_addr(unsigned int chip_no)

功能：phy base address 读取

参数：**chip_no**-输入

返回值：当前 chip_no 对应的 phy base address

4.1.52 s32 yt8614_tx_dis_set(u32 lport, BOOL enable)

功能：tx dis config

参数：lport-输入：全局端口号

enable-输出: 禁用(FALSE)/使能(TRUE)

返回值：当前 chip_no 对应的 phy base address

4.1.53 s32 yt8614_tx_dis_get(u32 lport, BOOL *enable)

功能：获取当前 tx dis 配置状态

参数：lport-输入：全局端口号

enable-输出: 禁用(FALSE)/使能(TRUE)

4.1.54 s32 yt8614_utp_loopback(u32 lport, u8 speed, yt8614_loopback_t type, BOOL enable)

功能：utp loopback 配置

参数：lport-输入：全局端口号

speed-输入: SPEED_1000M(2)/SPEED_100M(1)/SPEED_10M(0)

type-输入:环回类型指定: YT8614_INTERNAL_LOOPBACK(0)/YT8614_EXTERNAL_LOOPBACK(1)/YT8614_REMOTE_LOOPBACK(2)

enable: 禁用(FALSE)/使能(TRUE)

4.1.55 s32 yt8614_fiber_loopback(u32 lport, yt8614_loopback_t type, BOOL enable)

功能：fiber loopback 配置

参数：lport-输入：全局端口号

type-输入:环回类型指定: YT8614_INTERNAL_LOOPBACK(0)/YT8614_EXTERNAL_LOOPBACK(1)/YT8614_REMOTE_LOOPBACK(2)

enable: 禁用(FALSE)/使能(TRUE)

4.1.56 s32 yt8614_pkg_generator_enable(u32 lport, BOOL enable, yt8614_pkg_gen_t para)

功能：包生成器配置

参数：lport-输入：全局端口号

para-输入：包生成器配置参数(详细参见 yt8614_pkg_gen_t 说明)

4.1.57 s32 yt8614_pkg_checker_enable(u32 lport, BOOL enable, yt8614_pkg_gen_checker_dir_t dir)

功能：收发包统计配置
参数：lport-输入：全局端口号
dir-输入：收发包统计的 data source 设定(详细参见 yt8614_pkg_gen_checker_dir_t 说明)

4.2 YT8614Q 驱动 API 说明

4.2.1 寄存器读写 API

s32 yt8614Q_read_reg(struct yt8614Q_phy_dev_info *info, struct yt8614Q_phy_reg *param);
s32 yt8614Q_write_reg(struct yt8614Q_phy_dev_info *info, struct yt8614Q_phy_reg *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。

4.2.2 s32 yt8614Q_phy_soft_reset(u32 lport, u8 type);

功能：软复位指定端口
参数：lport-输入：全局端口号
type-输入：SOFT_RESET_QSGMII(1)/SOFT_RESET_SGMII(2)

4.2.3 s32 yt8614Q_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。

4.2.4 s32 yt8614Q_fiber_enable(u32 lport, BOOL enable);

功能：指定端口光口模式 powerdown/正常(TRUE)
参数：lport-输入：全局端口号
enable-输入: powerdown(FALSE)/正常(TRUE)

4.2.5 s32 yt8614Q_fiber_unidirection_set(u32 lport, int speed, BOOL enable);

功能：设置指定端口光口模式光纤单向使能/禁用和速率
参数：lport-输入：全局端口号
speed-输入: SPEED_1000M(2)/SPEED_100M(1)
enable-输入: 禁用(FALSE)/使能(TRUE)

4.2.6 s32 yt8614Q_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-输入: 禁用(FALSE)/使能(TRUE)

4.2.7 s32 yt8614Q_fiber_speed_set(u32 lport, int fiber_speed);

功能：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: SPEED_1000M(2)/SPEED_100M(1)

4.2.8 s32 yt8614Q_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-输入: 禁用(FALSE)/使能(TRUE)

4.2.9 s32 yt8614Q_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-输入: 禁用(FALSE)/使能(TRUE)

4.2.10 s32 yt8614Q_qsgmii_sgmii_link_status_get(u32 lport, BOOL if_qsgmii, BOOL *linkup);

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(TRUE), SGMII(FALSE)

linkup-输出: linkdown(FALSE)/linkup(TRUE)

说明：对于 yt8614Q，只有一个 QSGMII，一个 SGMII，所以端口号 0-3 的结果是一样的。

4.2.11 int yt8614Q_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。

4.2.12 int yt8614Q_chip_hard_reset(unsigned int chip_no);

功能：phy 芯片硬复位，硬复位后，芯片所有寄存器都恢复至 default 值

参数：chip_no-输入

4.2.13 s32 yt8614Q_qsgmii_polarity_reverse(u32 lport, yt8614Q_polarity_dir_t dir);

功能：qsgmii 极性反转配置

参数：lport-输入：全局端口号

dir-收发方向指定，YT8614Q_SND(0)/YT8614Q_RCV(1)

4.2.14 s32 yt8614Q_sgmii_polarity_reverse(u32 lport, yt8614Q_polarity_dir_t dir)

功能：sgmii 极性反转配置

参数：lport-输入：全局端口号

dir-收发方向指定，YT8614Q_SND(0)/YT8614Q_RCV(1)

4.2.15 s32 yt8614Q_qsgmii_loopback(u32 lport, yt8614Q_loopback_t type, BOOL enable);

功能：qsgmii 环回配置

参数：lport-输入：全局端口号

type-环回类型指定: YT8614Q_INTERNAL_LOOPBACK(0)/YT8614Q_EXTERNAL_LOOPBACK(1)
/YT8614Q_REMOTE_LOOPBACK(2)

enable: 禁用(FALSE)/使能(TRUE)

4.2.16 s32 yt8614Q_sgmii_loopback(u32 lport, yt8614Q_loopback_t type, BOOL enable);

功能：sgmii 环回配置

参数：lport-输入：全局端口号

type-环回类型指定: YT8614Q_INTERNAL_LOOPBACK(0)/YT8614Q_EXTERNAL_LOOPBACK(1)
/YT8614Q_REMOTE_LOOPBACK(2)

enable: 禁用(FALSE)/使能(TRUE)

4.2.17 s32 yt8614Q_qsgmii_main_emphasis(u32 lport, yt8614Q_qsgmii_main_emphasis_grade_t grade);

功能：qsgmii main cursor 测试

参数：lport-输入：全局端口号

grade-加重等级(详细参见 yt8614Q_qsgmii_main_emphasis_grade_t 说明)

4.2.18 **s32 yt8614Q_qsgmii_post_emphasis(u32 lport, yt8614Q_qsgmii_post_emphasis_grade_t** grade)；

功能：qsgmii post cursor 测试

参数：lport-输入：全局端口号

grade-加重等级(详细参见 yt8614Q_qsgmii_post_emphasis_grade_t 说明)

4.2.19 **s32 yt8614Q_sgmii_main_emphasis(u32 lport, yt8614Q_sgmii_main_emphasis_grade_t** grade);

功能：sgmii main cursor 测试

参数：lport-输入：全局端口号

grade-加重等级(详细参见 yt8614Q_sgmii_main_emphasis_grade_t 说明)

4.2.20 **s32 yt8614Q_sgmii_post_emphasis(u32 lport, yt8614Q_sgmii_post_emphasis_grade_t** grade);

功能：sgmii post cursor 测试

参数：lport-输入：全局端口号

grade-加重等级(详细参见 yt8614Q_sgmii_post_emphasis_grade_t 说明)

4.2.21 int yt8614Q_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：

光口：YT8614Q_SMI_SEL_SDS_SGMII + 1

enable: 禁用(FALSE)/使能(TRUE)

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

4.2.22 s32 yt8614Q_app_set_chip_base_phy_addr(unsigned int chip_no, unsigned int base_phy_addr)

功能：phy base address 配置

参数：**chip_no**-输入

4.2.23 unsigned int yt8614Q_app_get_chip_base_phy_addr(unsigned int chip_no)

功能：phy base address 读取

参数：**chip_no**-输入

返回值：当前 chip_no 对应的 phy base address

4.2.24 s32 yt8614Q_tx_dis_set(u32 lport, BOOL enable)

功能：tx dis config

参数：lport-输入：全局端口号

enable-输入: 禁用(FALSE)/使能(TRUE)

返回值：当前 chip_no 对应的 phy base address

4.2.25 s32 yt8614Q_tx_dis_get(u32 lport, BOOL *enable)

功能：获取当前 tx dis 配置状态

参数：lport-输入：全局端口号

enable-输出：禁用(FALSE)/使能(TRUE)

4.2.26 s32 yt8614Q_fiber_loopback(u32 lport, yt8614Q_loopback_t type, BOOL enable)

功能：fiber 环回配置

参数：lport-输入：全局端口号

type-环回类型指定: YT8614Q_INTERNAL_LOOPBACK(0)/YT8614Q_EXTERNAL_LOOPBACK(1)
/YT8614Q_REMOTE_LOOPBACK(2)

enable: 禁用(FALSE)/使能(TRUE)

4.2.27 s32 yt8614Q_pkg_generator_enable(u32 lport, BOOL enable, yt8614Q_pkg_gen_t para)

功能：包生成器配置

参数：lport-输入：全局端口号

enable-输入: generator disable(FALSE)/generator enable(TRUE)

para-输入：包生成器配置参数(详细参见 yt8614Q_pkg_gen_t 说明)

4.2.28 s32 yt8614Q_pkg_checker_enable(u32 lport, BOOL enable, yt8614Q_pkg_gen_checker_dir_t dir)

功能：收发包统计配置

参数：lport-输入：全局端口号

enable-输入: checker disable(FALSE)/checker enable(TRUE)

dir-输入：收发包统计的 data source 设定(详细参见 yt8614Q_pkg_gen_checker_dir_t 说明)

4.2.29 s32 yt8614Q_fiber_rx_powerdown(u32 lport, BOOL enable)

功能：fiber rx power down or power on 配置

参数：lport-输入：全局端口号

enable-输入: fiber rx power on(FALSE)/fiber rx power down(TRUE)

4.3 YT8618 驱动 API 说明

4.3.1 寄存器读写 API

s32 yt8618_read_reg(struct yt8618_phy_dev_info *info, struct yt8618_phy_reg *param);

s32 yt8618_write_reg(struct yt8618_phy_dev_info *info, struct yt8618_phy_reg *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。

4.3.2 s32 yt8618_phy_soft_reset(u32 lport, u8 type);

功能：软复位指定端口

参数：lport-输入：全局端口号

type-输入：SOFT_RESET_UTP(0)/SOFT_RESET_QSGMII(1)/SOFT_RESET_SGMII(2)

4.3.3 s32 yt8618_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。

4.3.4 s32 yt8618_fiber_enable(u32 lport, BOOL enable);

功能：指定端口光口模式 powerdown/正常(TRUE)

参数：lport-输入：全局端口号

enable: powerdown(FALSE)/正常(TRUE)

4.3.5 s32 yt8618_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(FALSE)/正常(TRUE)

4.3.6 s32 yt8618_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(FALSE)/使能(TRUE)

4.3.7 s32 yt8618_fiber_speed_set(u32 lport, int fiber_speed);

功能：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: SPEED_1000M（2）/SPEED_100M(1)

4.3.8 s32 yt8618_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(FALSE)/使能(TRUE)

4.3.9 s32 yt8618_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(FALSE)/使能(TRUE)

4.3.10 s32 yt8618_qsgmii_sgmii_link_status_get(u32 lport, BOOL if_qsgmii, BOOL *linkup);

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(TRUE), SGMII(FALSE)

linkup-输出: linkdown(0)/linkup(1)

说明：对于 yt8618，只有一个 QSGMII，一个 SGMII，所以端口号 0-3 的结果是一样的。

4.3.11 int yt8618_combo_media_priority_set(u32 lport, BOOL 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(FALSE)/光口优先(TRUE)

4.3.12 int yt8618_combo_media_priority_get(u32 lport, BOOL *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(FALSE)/光口优先(TRUE)

4.3.13 s32 yt8618_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(FALSE)/使能(TRUE)

4.3.14 s32 yt8618_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(FALSE)/使能(TRUE)

4.3.15 s32 yt8618_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 CAP_10HALF          (1 << 0)  /* capability 10Mbps Half duplex */
#define CAP_10FULL          (1 << 1)  /* capability 10Mbps Full duplex */
#define CAP_100HALF         (1 << 2)  /* capability 100Mbps Half duplex */
#define CAP_100FULL         (1 << 3)  /* capability 100Mbps Full duplex */
#define CAP_1000HALF        (1 << 4)  /* capability 1000Mbps Half duplex */
#define CAP_1000FULL        (1 << 5)  /* capability 1000Mbps Full duplex */
#define CAP_PAUSE           (1 << 6)  /* capability pause */
#define CAP_ASYM_PAUSE    (1 << 7) /* capability Asymmetric pause */
```

4.3.16 s32 yt8618_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。

4.3.17 s32 yt8618_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(FALSE)/full duplex(TRUE)

4.3.18 s32 yt8618_utp_force_duplex_get(u32 lport, unsigned int *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)/DUPLEX_NO_SENSE(0xffff 自协商使能)

4.3.19 s32 yt8618_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_100M(1)/SPEED_10M(0)

4.3.20 s32 yt8618_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_100M(1)/SPEED_10M(0)/SPEED_UNKNOWN(0xffff)

4.3.21 int yt8618_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。

4.3.22 int yt8618_chip_hard_reset(unsigned int chip_no);

功能：phy 芯片硬复位，硬复位后，芯片所有寄存器都恢复至 default 值
参数：chip_no-输入

4.3.23 4.1.24 int yt8618_utp_SNR_read(u32 lport, u8 *snr0, u8 *snr1, u8 *snr2, u8 *snr3);

功能：utp SNR(Signal Noise Ratio) read. 可以通过返回值判定当前信号质量
参数：lport-输入：全局端口号
snr0-输出：utp 第一对线 snr
snr1-输出：utp 第二对线 snr
snr2-输出：utp 第三对线 snr
snr3-输出：utp 第四对线 snr
说明：1000Mbps 下返回四对线 snr，100Mbps 下仅仅返回一对线的 snr

4.3.24 4.1.25 s32 yt8618_utp_fast_link_down_set(u32 lport, BOOL enable);

功能：utp fast link down 配置
参数：lport-输入：全局端口号
enable: 禁用(FALSE)/使能(TRUE)

4.3.25 s32 yt8618_utp_fast_link_down_get(u32 lport, BOOL *enable);

功能：获取当前 utp fast link down 状态
参数：lport-输入：全局端口号
enable-输出: 禁用(FALSE)/使能(TRUE)

4.3.26 s32 yt8618_utp_smart_downgrade_set(u32 lport, BOOL enable);

功能：utp speed smart downgrade 配置
参数：lport-输入：全局端口号
enable: 禁用(FALSE)/使能(TRUE)

4.3.27 s32 yt8618_utp_smart_downgrade_get(u32 lport, BOOL *enable);

功能：获取当前 utp speed smart downgrade 状态
参数：lport-输入：全局端口号
enable-输出: 禁用(FALSE)/使能(TRUE)

4.3.28 s32 yt8618_utp_eee_set(u32 lport, BOOL enable);

功能：utp eee 配置
参数：lport-输入：全局端口号
enable: 禁用(FALSE)/使能(TRUE)

4.3.29 s32 yt8618_utp_eee_get(u32 lport, BOOL *enable);

功能：获取当前 utp eee 配置状态
参数：lport-输入：全局端口号
enable-输出: 禁用(FALSE-not in EEE)/使能(TRUE-in EEE)

4.3.30 s32 yt8618_utp_lds_set(u32 lport, BOOL enable);

功能：4 对线百兆长距离配置
参数：lport-输入：全局端口号
enable: 禁用(FALSE)/使能(TRUE)

4.3.31 s32 yt8618_utp_lds_get_status(u32 lport, yt8618_utp_lds_status_t *status);

功能：4 对线百兆长距离状态获取
参数：lport-输入：全局端口号
status-输出:
YT8618_4pair_LRE_100M_FULL(0)/YT8618_1000M_FULL(1)/
YT8618_1000M_HALF(2)/YT8618_100M_FULL(3)/
YT8618_100M_HALF(4)/YT8618_10M_FULL(5)/YT8618_10M_HALF(6)/
YT8618_UNKNOWN(7)

4.3.32 s32 yt8618_utp_sleep_set(u32 lport, BOOL enable);

功能：utp sleep 配置

参数：lport-输入：全局端口号

enable: 禁用(FALSE)/使能(TRUE)

4.3.33 s32 yt8618_utp_sleep_get(u32 lport, BOOL *enable);

功能：获取当前 utp sleep 配置状态

参数：lport-输入：全局端口号

enable-输出: 禁用(FALSE)/使能(TRUE)

4.3.34 s32 yt8618_utp_vct(u32 lport, yt8618_utp_vct_t type, yt8618_utp_vct_status_t* status, u16 *channel0, u16 *channel1, u16 *channel2, u16 *channel3);

功能：utp vct 测试

参数：lport-输入：全局端口号

type: YT8618_VCT_INTER_BOARD(0)/YT8618_VCT_NON_INTER_BOARD(1)

status--输出：YT8618_VCT_OPEN(0)/YT8618_VCT_SHORT(1)/YT8618_VCT_LOAD(2)

channel0-输出：第一对线出问题的地点(单位 cm)

channel1-输出：第二对线出问题的地点(单位 cm)

channel2-输出：第三对线出问题的地点(单位 cm)

channel3-输出：第四对线出问题的地点(单位 cm)

说明：当 status 为 YT8618_VCT_LOAD，返回值 channel0~channel3 无意义

4.3.35 s32 yt8618_utp_crossover(u32 lport, yt8618_utp_crossover_config_t type);

功能：utp crossover 配置

参数：lport-输入：全局端口号

type: YT8618_CROSSOVER_MDI(0)/YT8618_CROSSOVER_MDIX(1)
/YT8618_CROSSOVER_AUTO(2)

4.3.36 s32 yt8618_utp_optimization(u32 lport, yt8618_utp_optimization_data_t odata);

功能：utp 电口指标优化

参数：lport-输入：全局端口号

odata-优化指标(详细参见 yt8618_optimization_data_t 说明)

4.3.37 s32 yt8618_utp_test_mode1000M(u32 lport, yt8618_utp_test_mode_1000m_t type);;

功能：utp template 测试

参数：lport-输入：全局端口号

type-测试类别(详细参见 yt8618_utp_test_mode_1000m_t 说明)

4.3.38 s32 yt8618_utp_test_mode100M(u32 lport, yt8618_utp_test_mode_100m_t type);

功能：utp template 测试

参数：lport-输入：全局端口号

type-测试类别(详细参见 yt8618_utp_test_mode_100m_t 说明)

4.3.39 s32 yt8618_utp_test_mode10M(u32 lport, yt8618_utp_test_mode_10m_t type);

功能：utp template 测试

参数：lport-输入：全局端口号

type-测试类别(详细参见 yt8618_utp_test_mode_10m_t 说明)

4.3.40 s32 yt8618_qsgmii_polarity_reverse(u32 lport, yt8618_polarity_dir_t dir);

功能：qsgmii 极性反转配置

参数：lport-输入：全局端口号

dir-收发方向指定，YT8618_SND(0)/YT8618_RCV(1)

4.3.41 s32 yt8618_sgmii_polarity_reverse(u32 lport, yt8618_polarity_dir_t dir)

功能：sgmii 极性反转配置

参数：lport-输入：全局端口号

dir-收发方向指定，YT8618_SND(0)/YT8618_RCV(1)

4.3.42 s32 yt8618_qsgmii_loopback(u32 lport, yt8618_loopback_t type, BOOL enable);

功能：qsgmii 环回配置

参数：lport-输入：全局端口号

type-环回类型指定: YT8618_INTERNAL_LOOPBACK(0)/YT8618_EXTERNAL_LOOPBACK(1) /YT8618_REMOTE_LOOPBACK(2)

enable: 禁用(FALSE)/使能(TRUE)

4.3.43 s32 yt8618_sgmii_loopback(u32 lport, yt8618_loopback_t type, BOOL enable);

功能：sgmii 环回配置

参数：lport-输入：全局端口号

type-环回类型指定: YT8618_INTERNAL_LOOPBACK(0)/YT8618_EXTERNAL_LOOPBACK(1) /YT8618_REMOTE_LOOPBACK(2)

enable: 禁用(FALSE)/使能(TRUE)

4.3.44 s32 yt8618_qsgmii_main_emphasis(u32 lport, yt8618_qsgmii_main_emphasis_grade_t grade);

功能：qsgmii main cursor 测试

参数：lport-输入：全局端口号

grade-加重等级(详细参见 yt8618_qsgmii_main_emphasis_grade_t 说明)

4.3.45 s32 yt8618_qsgmii_post_emphasis(u32 lport, yt8618_qsgmii_post_emphasis_grade_t grade);

功能：qsgmii post cursor 测试

参数：lport-输入：全局端口号

grade-加重等级(详细参见 yt8618_qsgmii_post_emphasis_grade_t 说明)

4.3.46 s32 yt8618_sgmii_main_emphasis(u32 lport, yt8618_sgmii_main_emphasis_grade_t grade);

功能：sgmii main cursor 测试

参数：lport-输入：全局端口号

grade-加重等级(详细参见 yt8618_sgmii_main_emphasis_grade_t 说明)

4.3.47 s32 yt8618_sgmii_post_emphasis(u32 lport, yt8618_sgmii_post_emphasis_grade_t grade);

功能：sgmii post cursor 测试

参数：lport-输入：全局端口号

grade-加重等级(详细参见 yt8618_sgmii_post_emphasis_grade_t 说明)

4.3.48 int yt8618_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：YT8618_SMI_SEL_PHY + 1

光口：YT8618_SMI_SEL_SDS_SGMII + 1

enable: 禁用(FALSE)/使能(TRUE)

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

4.3.49 s32 yt8618_app_set_chip_base_phy_addr(unsigned int chip_no, unsigned int base_phy_addr)

功能：phy base address 配置

参数：**chip_no**-输入

4.3.50 unsigned int yt8618_app_get_chip_base_phy_addr(unsigned int chip_no)

功能：phy base address 读取

参数：**chip_no**-输入

返回值：当前 chip_no 对应的 phy base address

4.3.51 s32 yt8618_utp_loopback(u32 lport, u8 speed, yt8618_loopback_t type, BOOL enable)

功能：utp loopback 配置

参数：lport-输入：全局端口号

speed-输入: SPEED_1000M(2)/SPEED_100M(1)/SPEED_10M(0)

type-输入:环回类型指定: YT8618_INTERNAL_LOOPBACK(0)/YT8618_EXTERNAL_LOOPBACK(1)/YT8618_REMOTE_LOOPBACK(2)

enable: 禁用(FALSE)/使能(TRUE)

4.3.52 s32 yt8618_fiber_loopback(u32 lport, yt8618_loopback_t type, BOOL enable)

功能：fiber loopback 配置

参数：lport-输入：全局端口号

type-输入:环回类型指定: YT8618_INTERNAL_LOOPBACK(0)/YT8618_EXTERNAL_LOOPBACK(1)/YT8618_REMOTE_LOOPBACK(2)

enable: 禁用(FALSE)/使能(TRUE)

4.3.53 s32 yt8618_pkg_generator_enable(u32 lport, BOOL enable, yt8618_pkg_gen_t para)

功能：包生成器配置

参数：lport-输入：全局端口号

para-输入：包生成器配置参数(详细参见 yt8618_pkg_gen_t 说明)

4.3.54 s32 yt8618_pkg_checker_enable(u32 lport, BOOL enable, yt8618_pkg_gen_checker_dir_t dir)

功能：收发包统计配置

参数：lport-输入：全局端口号

dir-输入：收发包统计的 data source 设定(详细参见 yt8618_pkg_gen_checker_dir_t 说明)