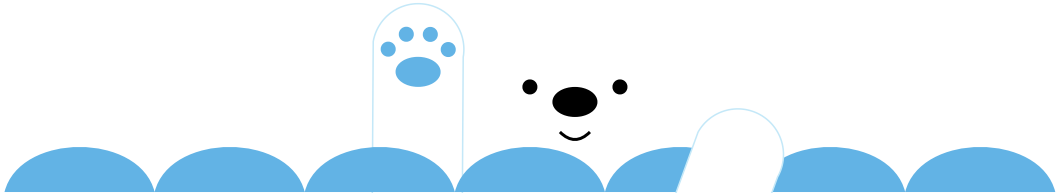
BiScuitOS



MDIO (SMI/MIIM)

Bus

20 Dec 2018

GitHub: MDIO

Email: BuddyZhang1 buddy.zhang@aliyun.com

目录

专业术语

MDIO: Management Data Input/Output.

SMI: Serial Management Interface.

MIIM: Media Independent Interface Management.

MAC: Media Access Control.

PHY: Ethernet physical layer.

MII: Media Independent Interface.

RMII: Reduced media-independent interface.

GMII: Gigabit media-independent interface.

RGMII: Reduced Gigabit media-independent interface.

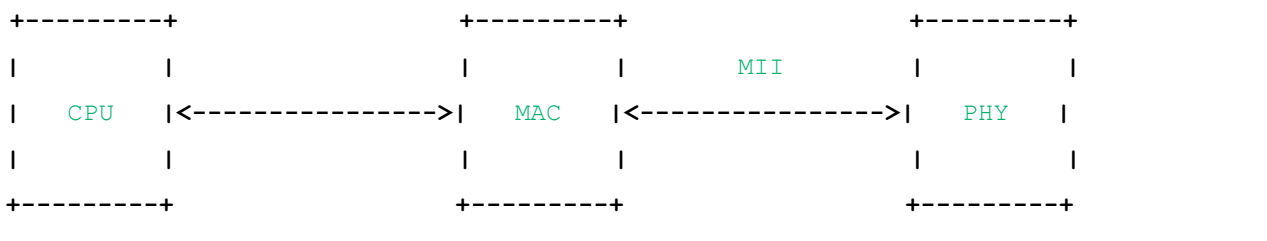
XGMII: 10-Gigabit media-independent interface.

SGMII: Serial Gigabit media-independent interface.

XAUI: 10 Gigabit Attachment Unit Interface.

RXAUI: Reduced Pin eXtended Attachment Unit Interface.

MDIO 原理



MII

MII 是一个标准接口，用于连接 MAC 和 PHY。MII 是 IEEE-802.3 定义的以太网标准，MII 接口可以同时控制多个 PHY。

MII 包含两个接口：

> **1. 一个数据接口，用户 MAC 和 PHY 之间收发 Ethernet 数据**
>
> **2. 一个管理接口，这个管理接口通常称为 MDIO，MIIM 或者 SMI。这个接口用于 MAC 从 PHY 读取相关管理寄存器的值，或者往 PHY 管理寄存器上写入数据。**

原始的 MMI 传输网络数据部分使用两对 4-bit 线 (4 根用于发送数据，4 根用于接收数据)，数据只有 100 Mib/s 的吞吐量。在原始 MII 基础上，拓展支持了衰减信号和增加 速度，当前各种新传输接口：RMII，RGMII，XGMII，SGMII。这些数据信号接口虽然速度 等有所改变，但 MMIM 部分还是共同使用 MIDO 接口进行管理数据的传输。

MDIO

MDIO 也被称为 MIIM，或者 SMI，它是 IEEE802.3 定义标准 MII 接口的一部分，用于 MAC 配置 PHY。MDIO 具有两个信号线，分别如下：

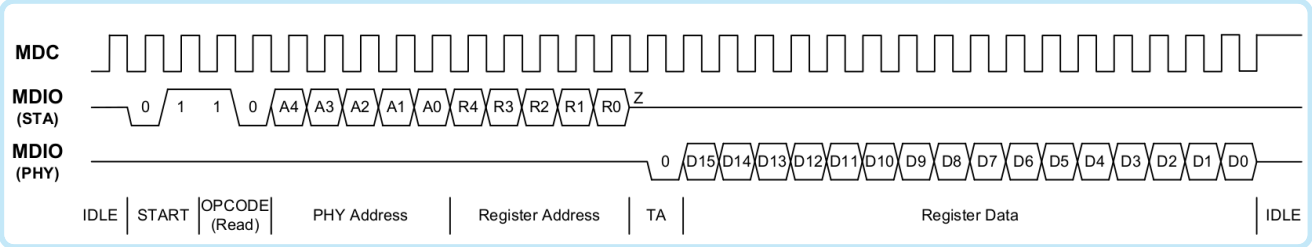
> **1. MDC 时钟线：MDIO 的时钟信号，由 MAC 驱动 PHY**
>
> **2. MDIO 数据线：双向数据线，用于在 MAC 和 PHY 之间传输配置信息**

MDIO 总线只支持 MAC 作为主设备，PHY 作为从设备。MIDO 支持两种时序，分别为 **Clause 22** 和 **Clause 45**.

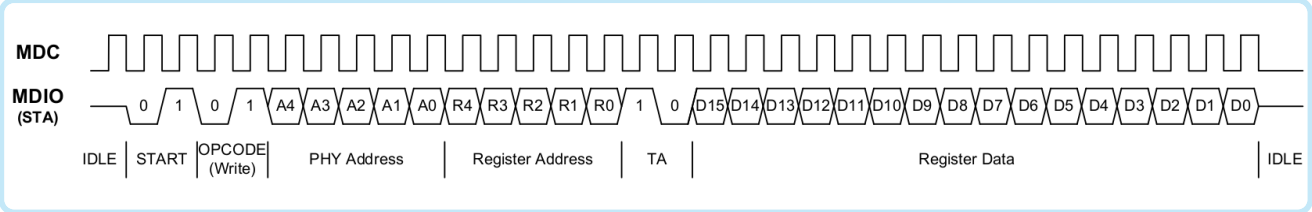
MDIO clause 22

MDIO clause 22 是 MDIO 使用的一种信号时序，在这个信号时序模式，MAC 先向 MDIO 信号线上拍 32 个周期，接着传输 16 bit 的控制位。16 个信号位包含了两个开始位， 2 个访问控制位，5 bits 的 PHY 地址，5 bits 的寄存器地址，以及 2 bits 的翻转位。 当进行写操作的时候，MAC 在接下来的周期中提供地址和数据；当进行读操作的时候， PHY 会翻转 MDIO 之后向 MDIO 信号线上发送数据。

MDIO Read



MDIO Write



MDIO Clause 45

MDIO 也支持 Clause 45 时序，其是 Clause 22 的拓展协议。与 Clause 22 不同的是， 16 位中的起始位：Clause 22 是 00， 而 Clause 45 是 01.

详细 MDIO Clause 45 和 Clause 22 时序，请查阅：

了解与MDIO/MDC接口相关的22号、45号条款

MDIO Clause 45 And Clause 22

Uboot 中访问 MDIO (SMI/MIIM)

项目开发中，经常要在 uboot 阶段通过 MDIO 总线去配置 PHY 或者 SWITCH。开发者可以参照本节内容，通过源码或者工具去访问 MDIO 总线。

工具访问 MDIO

uboot 提供了 **mii** 工具去操作 MDIO 总线，用法如下：

```
mii - MII utility commands
Usage:
mii device                                - list available devices
mii device <devname>                      - set current device
mii info   <addr>                         - display MII PHY info
mii read   <addr> <reg>                   - read  MII PHY <addr> register <reg>
mii write  <addr> <reg> <data>            - write MII PHY <addr> register <reg>
mii modify <addr> <reg> <data> <mask> - modify MII PHY <addr> register <reg>
                                          updating bits identified in <mask>
mii dump   <addr> <reg>                   - pretty-print <addr> <reg> (0-5 only)
```

列出可用的网卡

mii device 命令可以列出所有可用的网卡，命令模式如下：

```
mii device
```

例如，Soc 上可用的网卡：

```
ZynqMP> mii device
MII devices: 'eth0'
Current device: 'eth0'
ZynqMP>
```

列出 PHY 信息

当找打可用的网卡之后，可以使用 “mii info” 命令查看 PHY 相关的信息，命令模式如下：

```
ZynqMP> mii info 0
PHY 0x00: OUI = 0x5043, Model = 0x1D, Rev = 0x01, 1000baseT, FDX
ZynqMP>
```

MDIO 读

mii read 命令可以进行 MDIO 读操作，该操作可以访问 PHY 或 SWITCH 的寄存器， 命令模式如下：

```
mii read <phy_id> <reg_id>
```

例如，Soc 的 MDIO 0 总线上包含一个 MV88e6185 PHY，PHY ID 寄存器地址为 0x2 和 0x3，通过 mii 工具读操作如下：

```
ZynqMP> mii read 0 0x3
0DD1
ZynqMP> mii read 0 0x2
0141
ZynqMP>
```

MDIO 写

mii write 命令可以进行 MDIO 写操作，该操作可以写 PHY 或 SWITCH 的寄存器， 命令模式如下：

```
mii write <phy_id> <reg_id> <data>
```

例如，向 MDIO 0 上，PHY 0 的 0x16 寄存器写值，该寄存器是切页寄存器：

```
ZynqMP> mii read 0 0x16
0000
ZynqMP> mii write 0 0x16 0x1
ZynqMP> mii read 0 0x16
0001
ZynqMP>
```

Uboot 源码访问 MDIO

有时项目需求，需要在 uboot 源码中访问 MDIO，开发者可以参考下面代码进行 MDIO 操作：

```
/*
* MDIO (SMI/MIIM) read/write on Uboot
*
* (C) 2018.12.20 BiscuitOS <buddy.zhang@aliyun.com>
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License version 2 as
* published by the Free Software Foundation.
*/
#include <common.h>
#include <miiphy.h>

/*
```

```
* MDIO read
*  @phy: PHY ID
*  @reg: Register address.
*  @data: read buffer.
*
* If succeed, return 0.
*/
static int mdio_read(unsigned char phy, unsigned char reg,
                                              unsigned short *data)
{
    const char *devname;

    /* use current device */
    devname = miiphy_get_current_dev();

    if (miiphy_read(devname, phy, reg, &data) != 0) {
        printf("Error reading from the PHY %d reg %d\n", phy, reg);
        return -1;
    }
    return 0;
}

/*
* MDIO write
*  @phy: PHY ID
*  @reg: Register address.
*  @data: data need to write.
*/
static int mdio_write(unsigned char phy, unsigned char reg,
                                               unsigned short data)
{
    const char *devname;

    /* use current device */
    devname = miiphy_get_current_dev();

    if (miiphy_write(devname, phy, reg, data) != 0) {
        printf("Error writing to the PHY %d reg %d\n", phy, reg);
        return -1;
    }
    return 0;
}
```

Kernel 中访问 MDIO

内核中 MDIO 的驱动一般和 PHY 驱动或者网卡驱动一同使用，开发者可以参考下面驱动模型来使用 MDIO

```
/*
* MDIO/SMI/MIIM on Kernel
*
* (C) 2018.12.20 BuddyZhang1 <buddy.zhang@aliyun.com>
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License version 2 as
* published by the Free Software Foundation.
*/
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/mii.h>
#include <linux/phy.h>
#include <linux/phy_fixed.h>

#define MII_BUS         "mii_demo"

static struct mii_bus *mii_demo;

extern int swphy_read_reg(int reg, const struct fixed_phy_status *state);

/* mdio read entence */
static int mii_demo_read(struct mii_bus *bus, int phys_addr, int reg)
{
    struct fixed_phy_status state;

    /* setup real MDIO control register and transfer data */

    return swphy_read_reg(reg, &state);
}

/* mdio write entence */
static int mii_demo_write(struct mii_bus *bus, int phy_addr, int reg,
                          unsigned short val)
{
    return 0;
}
```

```
static int __init mdio_demo_init(void)
{
    int ret;

    /* Allocate MII bus */
    mii_demo = mdiobus_alloc();
    if (mii_demo == NULL) {
        ret = -ENOMEM;
        goto err;
    }

    /* setup mii bus */
    snprintf(mii_demo->id, MII_BUS_ID_SIZE, "mii_demo-0");
    mii_demo->name  = MII_BUS;
    mii_demo->read  = &mii_demo_read;
    mii_demo->write = &mii_demo_write;

    /* Register mdio bus */
    ret = mdiobus_register(mii_demo);
    if (ret)
        goto err_alloc;

    return 0;

err_alloc:
    mdiobus_free(mii_demo);

err:
    return ret;
}

static void __exit mdio_demo_exit(void)
{
    mdiobus_unregister(mii_demo);

    mdiobus_free(mii_demo);
}

module_init(mdio_demo_init);
module_exit(mdio_demo_exit);
MODULE_LICENSE("GPL v2");
```

Makefile

```
obj-m += mdio.o

KERNELDIR ?= /lib/modules/$(shell uname -r)/build

PWD       := $(shell pwd)

ROOT := $(dir $(M))
DEMOINCLUDE := -I$(ROOT)../include -I$(ROOT)/include

GCCVERSION = $(shell gcc -dumpversion | sed -e 's/\.\([0-9][0-9]\)/\1/g' -e 's/\.\

GCC49 := $(shell expr $(GCCVERSION) \>= 40900)

all:
        $(MAKE) -C $(KERNELDIR) M=$(PWD) modules

install: all
        $(MAKE) -C $(KERNELDIR) M=$(PWD) modules_install
        depmod -a

clean:
        rm -rf *.o *.o.d *~ core .depend .*.cmd *.ko *.ko.unsigned *.mod.c .tmp_ver
        .cache.mk *.save *.bak Modules.* modules.order Module.markers *.bin

CFLAGS_mdio.o := -Wall $(DEMOINCLUDE)

ifeq ($(GCC49),1)
        CFLAGS_mdio.o += -Wno-error=date-time
endif

CFLAGS_mdio.o := $(DEMOINCLUDE)
```

上述源码可以之间编译到内核源码树或外部编译，外部编译适用如下命令：

```
make clean
make
sudo insmod mdio.ko
```

MDIO 驱动模型分析

内核为 MDIO 子系统提供了以下接口供 MDIO 的分配，注册，释放，和销毁的操作。当 MDIO 注册成功，网卡或 CPU 的 MAC 就可以通过 MDIO 总线去配置 PHY 或者 SWITCH。

1. mdiobus_alloc() 分配一个 mdio bus 结构
2. mdiobus_free() 释放一个 mdio bus
3. mdiobus_register() 注册一个 mdio bus
4. mdiobus_unregister() 注销一个 mdio bus

用户空间工具访问 MDIO (SMI/MIIM)

用户空间提供了多个工具操作 PHY，例如 **mii-diag**, **mii-tool** 和 **ethtool**，这些工具都可以操作一个可用的 PHY。使用如下：

mii-diag

```
# mii-diag --help
Usage: mii-diag [-aDfrRvVw] [-AF <speed+duplex>] [--watch] <interface>.

  This program configures and monitors the transceiver management registers
  for network interfaces.  It uses the Media Independent Interface (MII)
  standard with additional Linux-specific controls to communicate with the
  underlying device driver.  The MII registers control and report network
  link settings and errors.  Examples are link speed, duplex, capabilities
  advertised to the link partner, status LED indications and link error
  counters.

   The common usage is
      mii-diag eth0

   The default interface is "eth0".
Frequently used options are
   -A  --advertise <speed|setting>
   -F  --fixed-speed <speed>
        Speed is one of: 100baseT4, 100baseTx, 100baseTx-FD, 100baseTx-HD,
                         10baseT, 10baseT-FD, 10baseT-HD
   -s  --status     Return exit status 2 if there is no link beat.

Less frequently used options are
   -a  --all-interfaces  Show the status all interfaces
               (Not recommended with options that change settings.)
   -D  --debug
   -g  --read-parameters        Get driver-specific parameters.
   -G  --set-parameters PARMS   Set driver-specific parameters.
```

```
       Parameters are comma separated, missing elements retain existing value.
   -M  --msg-level LEVEL        Set the driver message bit map.
   -p  --phy ADDR               Set the PHY (MII address) to report.
   -r  --restart        Restart the link autonegotiation.
   -R  --reset          Reset the transceiver.
   -v  --verbose        Report each action taken.
   -V  --version        Emit version information.
   -w  --watch          Continuously monitor the transceiver and report changes.

   This command returns success (zero) if the interface information can be
   read.  If the --status option is passed, a zero return means that the
   interface has link beat.
```

mii-tool

```
mii-tool --help
usage: mii-tool [-VvRrwl] [-A media,... | -F media] [interface ...]
       -V, --version               display version information
       -v, --verbose               more verbose output
       -R, --reset                 reset MII to poweron state
       -r, --restart               restart autonegotiation
       -w, --watch                 monitor for link status changes
       -l, --log                   with -w, write events to syslog
       -A, --advertise=media,...   advertise only specified media
       -F, --force=media           force specified media technology
media: 1000baseTx-HD, 1000baseTx-FD,
       100baseT4, 100baseTx-FD, 100baseTx-HD,
       10baseT-FD, 10baseT-HD,
       (to advertise both HD and FD) 1000baseTx, 100baseTx, 10baseT
```

ethtool

```
ethtool --help
```

用户空源码间访问 MDIO (SMI/MIIM)

用户空间在源码中直接访问 MDIO 的方法比较难找，但这里提供了一个思路，开发者可以 参照这个源码，就可以在用户空间操作 MDIO 总线。

首先，开发者可以编译一个内核驱动，然后这个驱动将 MDIO 总线导出到用户空间，源码 如下：

```
/*
* MDIO Userland Procedure-Interface
*
* (C) 2018.12.20 BuddyZhang1 <buddy.zhang@aliyun.com>
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License version 2 as
* published by the Free Software Foundation.
*/
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/device.h>
#include <linux/stat.h>
#include <linux/slab.h>
#include <linux/platform_device.h>

#include <linux/err.h>
#include <linux/io.h>
#include <asm/uaccess.h>
#include <linux/bitops.h>
#include <linux/phy.h>

#define DEV_NAME "mdio_demo"

/* mdio_demo from real Ethernet card, and export it here! */
extern struct mii_bus *mdio_demo;

/* Parse input string
*  Value[0]: DeviceAddress. Value[1]: Register. Value[2]: date in Byte.
*
*  Read/Write operation
*  CMD: <r/w>,<DevAddr>,<RegAddr>,[Value],
*       r: Read special PHY/SERDES register
*       w: Write data to special PHY/SERDES register.
*       DevAddr: Device address
*       RegAddr: Register address
*       Value:   value what to write.
*/
static int parse_input_string(const char *string, int *value, int *flag)
{
    int nr;
    char *tmp;
    char *buffer, *leg;
```

```
    int i = 0;

    buffer = (char *)kmalloc(strlen(string) + 1, GFP_KERNEL);
    leg = buffer;
    memset(buffer, 0, strlen(string) + 1);
    /* Copy original data */
    strcpy(buffer, string);

    while ((tmp = strstr(buffer, ","))) {
        int data;
        char dd[20];

        nr = tmp - buffer;
        tmp++;
        strncpy(dd, buffer, nr);
        dd[nr] = '\0';
        if (strncmp(dd, "r", 1) == 0) {
            *flag = 1;
        } else if (strncmp(dd, "w", 1) == 0) {
            *flag = 2;
        } else {
            sscanf(dd, "%d", &data);
            value[i++] = data;
        }
        buffer = tmp;
    }
    kfree(leg);
    return 0;
}

/* Dump all PHY's all register */
static ssize_t mdio_demo_show(struct device *dev,
                    struct device_attribute *attr, char *buf)
{
    ssize_t size = 0;
    int phy, reg;

    for (phy = 0; phy < 32; phy++) {
        for (reg = 0; reg < 32; reg++) {
            unsigned short val;

            if (((reg % 16) == 0) && (reg != 0))
                printk("\n");
            val = mdio_demo->read(mdio_demo, phy, reg);
            printk("%#04x ", val);
```

```
        }
    }
    printk("\n");

    return size;
}

/* Read/Write PHY register */
static ssize_t mdio_demo_store(struct device *dev,
            struct device_attribute *attr, const char *buf, size_t size)
{
    int flag = 0; /* 1: read 2: write */
    int value[10];

    parse_input_string(buf, value, &flag);
    /* Read data from Port-Register */
    if (flag == 1) {
        unsigned short reg;

        reg = mdio_demo->read(mdio_demo, value[0], value[1]);
        /* Output message into dmesg */
        printk("\r\nRead: Port - Dev[%#x] Reg[%#x] Value[%#x]\n",
                                                value[0], value[1], reg);
    } else if (flag == 2) { /* Write data to Port-Register */

        mdio_demo->write(mdio_demo, value[0], value[1], value[2]);
        /* Output message into dmesag */
        printk("\r\nWrite: Port - Dev[%#x] Reg[%#x] value[%#x]\n",
                                            value[0], value[1], value[2]);
    } else {
        printk(KERN_ERR "Unknown operation from Port register\n");
    }
    return size;
}

static struct device_attribute mdio_demo_attr =
       __ATTR_RW(mdio_demo);

/* probe platform driver */
static int mdio_demo_probe(struct platform_device *pdev)
{
    int err;

    err = device_create_file(&pdev->dev, &mdio_demo_attr);
    if (err) {
```

```
        printk("Unable to create device file for reg***.\n");
        return -EINVAL;
    }

    return 0;
}

/* remove platform driver */
static int mdio_demo_remove(struct platform_device *pdev)
{
    device_remove_file(&pdev->dev, &mdio_demo_attr);

    return 0;
}

/* platform device information */
static struct platform_device mdio_demo_device = {
    .name = DEV_NAME,  /* Same as driver name */
    .id   = -1,
};

/* platform driver information */
static struct platform_driver mdio_demo_driver = {
    .probe  = mdio_demo_probe,
    .remove = mdio_demo_remove,
    .driver = {
        .name = DEV_NAME, /* Same as device name */
    },
};

/* init entence */
static __init int mdio_demo_init(void)
{
    int ret;


    /* register device */
    ret = platform_device_register(&mdio_demo_device);
    if (ret)
        return ret;

    /* register driver */
    return platform_driver_register(&mdio_demo_driver);
}
```

```
/* Exit entence */
static __exit void mdio_demo_exit(void)
{
    platform_driver_unregister(&mdio_demo_driver);
    platform_device_unregister(&mdio_demo_device);
}

module_init(mdio_demo_init);
module_exit(mdio_demo_exit);

MODULE_LICENSE("GPL v2");
```

Makefile

```
obj-m += mdio.o

KERNELDIR ?= /lib/modules/$(shell uname -r)/build

PWD       := $(shell pwd)

ROOT := $(dir $(M))
DEMOINCLUDE := -I$(ROOT)../include -I$(ROOT)/include

GCCVERSION = $(shell gcc -dumpversion | sed -e 's/\.\([0-9][0-9]\)/\1/g' -e 's/\.\

GCC49 := $(shell expr $(GCCVERSION) \>= 40900)

all:
        $(MAKE) -C $(KERNELDIR) M=$(PWD) modules

install: all
        $(MAKE) -C $(KERNELDIR) M=$(PWD) modules_install
        depmod -a

clean:
        rm -rf *.o *.o.d *~ core .depend .*.cmd *.ko *.ko.unsigned *.mod.c .tmp_ver
        .cache.mk *.save *.bak Modules.* modules.order Module.markers *.bin

CFLAGS_mdio.o := -Wall $(DEMOINCLUDE)

ifeq ($(GCC49),1)
        CFLAGS_mdio.o += -Wno-error=date-time
endif

CFLAGS_mdio.o := $(DEMOINCLUDE)
```

将上面的源码编译进内核之后，开发者可以在用户空间按如下方法对 MDIO 进行读写操作。

MDIO 读

MDIO 读命令格式如下：

```
echo "r,<phy_id>,<reg_id>," > /sys/devices/platform/mdio_demo/mdio_demo_reg
dmesg | tail -n 5
```

例如，从 PHY 0 上读取 ID 寄存器

```
# echo "r,0x0,0x2," > /sys/devices/platform/mdio_demo/mdio_demo_reg
# dmesg | tail -n 5

Read: Port - Dev[0x0000] Reg[0x0002] Value[0x0141]
```

MDIO 写

MDIO 写命令格式如下：

```
echo "w,<phy_id>,<reg_id>,<data>," > /sys/devices/platform/mdio_demo/mdio_demo_reg
dmesg | tail -n 5
```

例如，往 PHY 0 的 22 号寄存器上写值

```
# echo "r,0x0,0x16," > /sys/devices/platform/mdio_demo/mdio_demo_reg
# echo "w,0x0,0x16,0x48," > /sys/devices/platform/mdio_demo/mdio_demo_reg
# echo "r,0x0,0x16," > /sys/devices/platform/mdio_demo/mdio_demo_reg
# dmesg | tail -n 10

Read: Port - Dev[0x0000] Reg[0x0016] Value[0x0000]

Write: Port - Dev[0x0000] Reg[0x0016] Value[0x0048]

Read: Port - Dev[0x0000] Reg[0x0016] Value[0x0048]
```

Dump 操作

将所有 PHY 的所有寄存器都 Dump 出来，命令模式如下：

```
cat /sys/devices/platform/mdio_demo/mdio_demo_reg
```

Arduino 上访问 MDIO

Arduino 上访问 MDIO 的方法可以拓展到 MCU 上访问 MDIO 的方法。这里使用硬件的方式，通过直接拍周期信号产生 MDIO 的读写时序。开发者可以参照如下代码：

```
/*
* MDIO/SMI/MIIM on Arduino
*
* (C) 2018.12.20 BuddyZhang1 <buddy.zhang@aliyun.com>
* (C) Sword <xxx@jjj.com>
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License version 2 as
* published by the Free Software Foundation.
*/
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

#define DIR     4 // 74HCT245 Dir
#define MDC     3 // D3 PIN for MDC
#define MDIO    2 // D2 PIN for MDIO
#define OUT     1
#define IN      0

int smi_init(void)
{
    pinMode(DIR, OUTPUT);
    digitalWrite(DIR, OUT);      //245 A to Y
    pinMode(MDC, OUTPUT);
    pinMode(MDIO, OUTPUT);
    return 0;
}


/*! Generates a rising edge pulse on MDC */
void pulse_mdc(void)
{
    volatile uint8_t i;
    //pinMode(MDC, OUTPUT);
    //i++;
    digitalWrite(MDC, 0);
```

```
    //delay(1);  change from 1Kbit/s to 10Kbit/s
    delayMicroseconds(100);
    i++;
    digitalWrite(MDC, 1);
    //delay(1);
    delayMicroseconds(100);
    i++;
}

/*
* SMI/MDIO/MIIM write
*  @phy: PHY id
*  @reg: Register address.
*  @data: data need to write.
*/
void write_smi(uint8_t phy, uint8_t reg, uint16_t data)
{
    uint8_t byte;
    uint16_t word;

    /* MDIO pin is output */
    digitalWrite(DIR, OUT);
    pinMode(MDIO, OUTPUT);

    digitalWrite(MDIO, 1);
    digitalWrite(MDC, 1);
    for (byte = 0;byte < 32; byte++)
        pulse_mdc();

    /* Stat code */
    digitalWrite(MDIO, 0);
    pulse_mdc();
    digitalWrite(MDIO, 1);
    pulse_mdc();

    /* Write OP Code */
    digitalWrite(MDIO, 0);
    pulse_mdc();
    digitalWrite(MDIO, 1);
    pulse_mdc();

    /* PHY address - 5 bits */
    for (byte = 0x10; byte != 0; byte = byte >> 1) {
        if (byte & phy)
            digitalWrite(MDIO, 1);
```

```
        else
            digitalWrite(MDIO, 0);
        pulse_mdc();
    }

    /* REG address - 5 bits */
    for (byte = 0x10; byte != 0; byte = byte >> 1) {
        if (byte & reg)
            digitalWrite(MDIO, 1);
        else
            digitalWrite(MDIO, 0);

        pulse_mdc();
    }
    /* Turn around bits */
    digitalWrite(MDIO, 1);
    pulse_mdc();
    digitalWrite(MDIO, 0);
    pulse_mdc();

    /* Data - 16 bits */
    for(word = 0x8000; word != 0; word = word >> 1) {
        if (word & data)
            digitalWrite(MDIO, 1);
        else
            digitalWrite(MDIO, 0);

        pulse_mdc();
    }

    /* This is needed for some reason... */
    pulse_mdc();
    /* Stay in 0 state */
    //MDC = 0;
    digitalWrite(DIR, IN);
    pinMode(MDIO, INPUT);
}

/*
* SMI/MDIO/MIIM read
*  @phy: PHY id.
*  @reg: Register address.
*/
uint16_t read_smi(uint8_t phy, uint8_t reg)
{
```

```
uint8_t byte;
volatile uint16_t word, data;
data = 0;

/* MDIO pin is output */
digitalWrite(DIR, OUT);
pinMode(MDIO, OUTPUT);

digitalWrite(MDIO, 1);
digitalWrite(MDC, 1);
for (byte = 0; byte < 32; byte++)
    pulse_mdc();

/* Stat code */
digitalWrite(MDIO, 0);
pulse_mdc();
digitalWrite(MDIO, 1);
pulse_mdc();

/* Read OP Code */
digitalWrite(MDIO, 1);
pulse_mdc();
digitalWrite(MDIO, 0);
pulse_mdc();

/* PHY address - 5 bits */
for (byte = 0x10; byte != 0; ) {
    if (byte & phy) {
        digitalWrite(MDIO, 1);
        pulse_mdc();
    } else {
        digitalWrite(MDIO, 0);
        pulse_mdc();
    }
    byte = byte >> 1;
}

/* REG address - 5 bits */
for (byte = 0x10; byte != 0; ){
    if (byte & reg){
        digitalWrite(MDIO, 1);
        pulse_mdc();
    }else{
        digitalWrite(MDIO, 0);
        pulse_mdc();
```

```
        }
        byte = byte >> 1;
    }

    /* Turnaround bits */

    /* MDIO now is input */
    digitalWrite(DIR, IN);
    pinMode(MDIO, INPUT);
    pinMode(MDC, OUTPUT);
    pulse_mdc();
    pulse_mdc();

    /* Data - 16 bits */
    for(word = 0x8000; word != 0; ) {

        if (digitalRead(MDIO)) {
            data |= word;
        }
        pulse_mdc();
        word = word >> 1;
    }

    /* This is needed for some reason... */
    pulse_mdc();
    /* Stay in 0 state */
    //MDC = 0;
    digitalWrite(DIR, IN);
    pinMode(MDIO, INPUT);

    return data;
}

/* setup entence */
void setup()
{
    uint8_t phy, reg, val, i, sel;
    String inStr = "";
    uint16_t reg_val = 0;

    /* start serial port at 9600 bps: */
    Serial.begin(9600);
    Serial.print("MDIO (SMI/MIIM) Initialization ....\n");
    smi_init();
    for(;;){
```

```
Serial.print("=========================================\r\n");
Serial.print("Arduino MDIO (SMI/MIIM) Bus tools\r\n");
Serial.print("1. Read register\r\n");
Serial.print("2. Write register\r\n");
Serial.print("3. Dump register\r\n");

Serial.setTimeout(100000);
sel = Serial.parseInt();

switch(sel) {
case 1:
    Serial.print("Read-PHY: ");
    Serial.setTimeout(100000);
    phy = Serial.parseInt();
    Serial.print(phy);
    if (phy > 31)
        break;
    Serial.print(" Register: ");
    Serial.setTimeout(100000);
    reg = Serial.parseInt();
    Serial.print(reg);
    if (reg > 31)
        break;
    reg_val = read_smi(phy, reg);
    Serial.print("\n\r\n\rRead-PHY: ");
    Serial.print(phy);
    Serial.print(" Register: ");
    Serial.print(reg);
    Serial.print(" Value [0x");
    Serial.print(reg_val, HEX);
    Serial.print("]\r\n\r\n");
    break;
case 2:
    Serial.print("Write-PHY: ");
    Serial.setTimeout(100000);
    phy = Serial.parseInt();
    Serial.print(phy);
    if (phy > 31)
        break;
    Serial.print(" Register: ");
    Serial.setTimeout(100000);
    reg = Serial.parseInt();
    Serial.print(reg);
    if (reg > 31)
        break;
```

```
            Serial.print(" Value: ");
            reg_val = Serial.parseInt();
            Serial.print(reg_val, HEX);
            write_smi(phy, reg, reg_val);
            Serial.print("\n\r\n\rWrite Port: ");
            Serial.print(phy);
            Serial.print(" Register: ");
            Serial.print(reg);
            Serial.print(" Value [0x");
            Serial.print(reg_val, HEX);
            Serial.print("]\r\n\r\n");
            break;
        case 3:
            Serial.print("Dump Start PHY: ");
            phy = Serial.parseInt();
            Serial.print(phy);
            Serial.print(" End PHY: ");
            Serial.setTimeout(100000);
            i = Serial.parseInt();
            Serial.print(i);
            Serial.print("\r\n");
            for (; phy < i; phy++) {
                Serial.print("/*0x");
                Serial.print(phy, HEX);
                Serial.print("*/");
                Serial.print("{");
                for (reg = 0; reg < 32; reg++) {
                    Serial.print("0x");
                    reg_val = read_smi(phy, reg);
                    Serial.print((reg_val & 0xffff), HEX);
                    if (reg < 31)
                        Serial.print(",");
                    delay(10);
                }
                Serial.print("}\r\n");
            }
            break;
        default:
            Serial.print("input wrong\r\n");
            break;
        }
    }
}

/* loop entence */
```

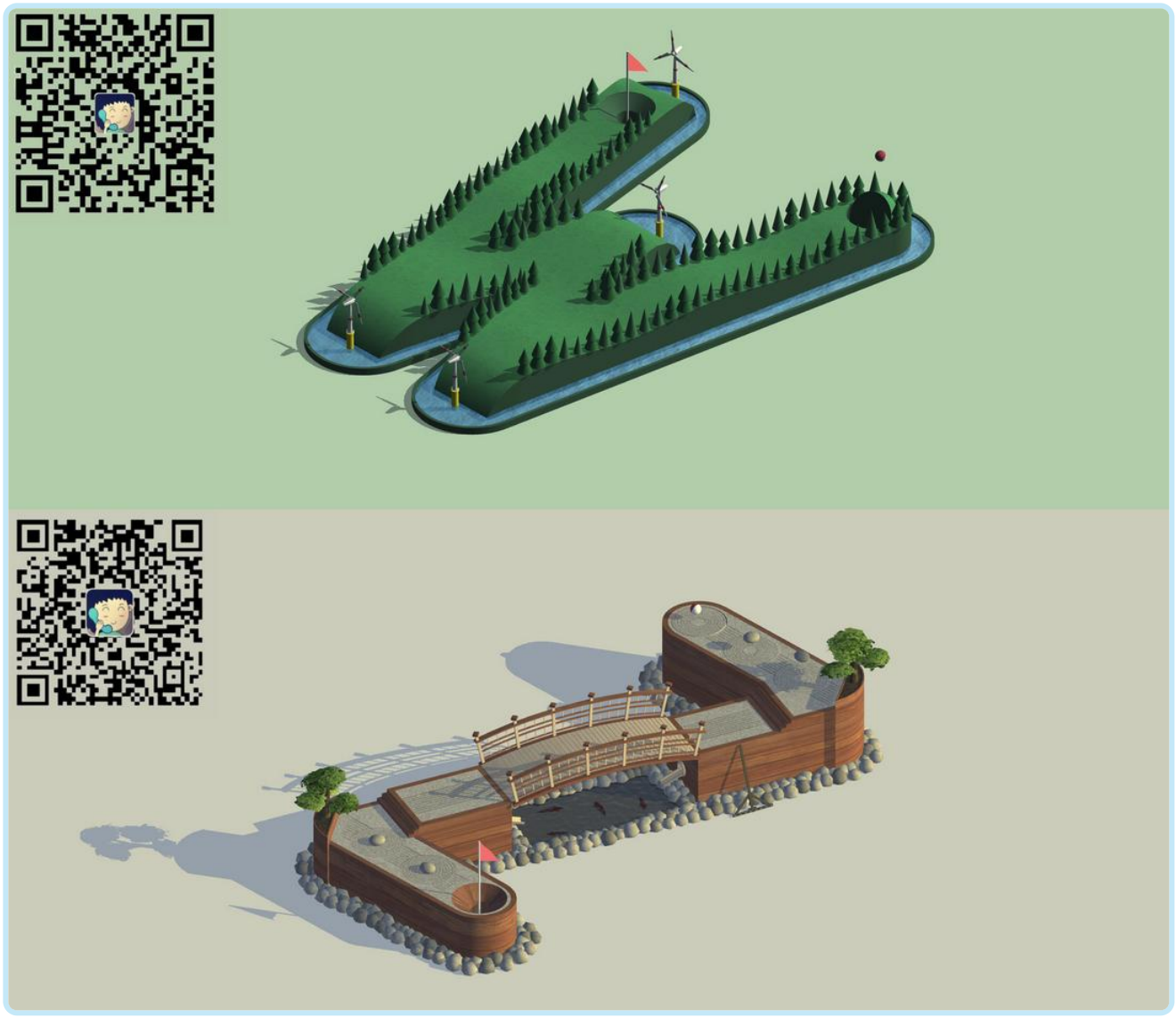
```
void loop()
{
}
```

Arduino 可以作为一个 MDIO调试器，加速网卡，PHY，SWITCH 等外部调试的速度。

附录

> **Media-independent interface**
>
> **Management Data Input/Output**
>
> **MDIO Clause 22 and 45**
>
> **BiscuitOS Home**
>
> **BiscuitOS Driver**
>
> **BiscuitOS Kernel Build**
>
> **Linux Kernel**
>
> **Bootlin: Elixir Cross Referencer**

赞赏一下吧 🙂



« Prev

Next »