SMI（MDC/MDIO）总线接口介绍

**1. MDIO接口**

SMI：串行管理接口（Serial Management Interface），也被称作MII管理接口（MII Management Interface），包括MDC和MDIO两条信号线。MDIO是一个PHY的管理接口，用来读/写PHY的寄存器，以控制PHY的行为或获取PHY的状态，MDC为MDIO提供时钟。

MDIO原本是为MII总线接口定义的，MII用于连接MAC和PHY，包含两种信号接口：

1. 一个数据接口用于MAC和PHY之间接收和发送以太网帧数据。

2. 一个PHY管理接口，即MDIO，用于读写每个PHY的控制寄存器和状态寄存器，以达到控制PHY行为和监控PHY状态的目的。

MDIO是双向的，只支持**一个**MAC连接**最多32个**PHY的连接方式，且MAC作为master，PHY作为slave。在写PHY寄存器的时候，由MAC驱动MDIO向PHY写入数据；在读PHY寄存器时，前半段由MAC驱动发送寄存器地址，后半段由PHY驱动回复寄存器的值。

MDC要求由MAC输出，是非周期性的，即不要求提供固定频率的时钟，对于PHY芯片则作为输入，以在**上升沿**触发MDIO的读写。MDC的时钟频率可以是DC-2.5MHz，即最小的时钟周期为400ns。

**2. MDIO数据传输协议**

MDIO数据格式定义在IEEE 802.3以太网标准中，如下图所示（数据传输顺序为从左至右）：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Preamble  (32bits) | Start  (2bits) | OP Code  (2bits) | PHYAD  (5bits) | REGAD  (5bits) | Turn Around  (2bits) | Data  (16bits) | Idle |
| Read | 1.......1 | 01 | 10 | A4A3A2A1A0 | R4R3R2R1R0 | Z0 | D15.......D0 | Z\* |
| Write | 1.......1 | 01 | 01 | A4A3A2A1A0 | R4R3R2R1R0 | 10 | D15.......D0 | Z\* |

上图中\*表示高阻态，这时MDIO的状态由一个外部的1.5KΩ电阻决定。

**Preamble+Start**：32bits的前导码以及2bit的开始位。

**OP Code**：2bits的操作码，10表示读，01表示写。

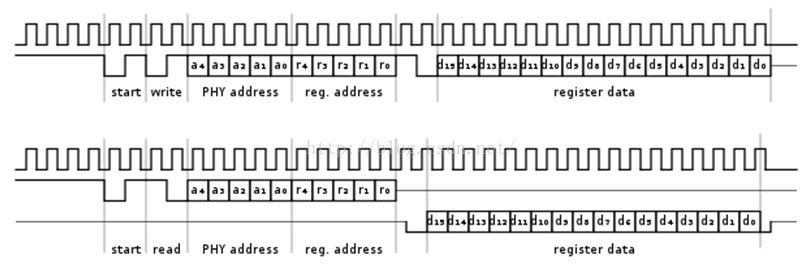
**PHYAD**：5bits的PHY地址，一般PHY地址从0开始顺序编号，例如6口switch中PHY地址为0-5。

**REGAD**：5bits的寄存器地址，即要读或写的寄存器。

**Turn Around**：2bits的TA，在读命令中，MDIO在此时由MAC驱动改为PHY驱动，并等待一个时钟周期准备发送数据。在写命令中，不需要MDIO方向发生变化，则只是等待两个时钟周期准备写入数据。

**Data**：16bits数据，在读命令中，PHY芯片将读到的对应PHYAD的REGAD寄存器的数据写到Data中，在写命令中，MAC将要写入对应PHYAD的REGAD寄存器的值写入Data中。

**Idle**：空闲状态，此时MDIO无源驱动，处高阻状态，但一般用上拉电阻使其处在高电平。



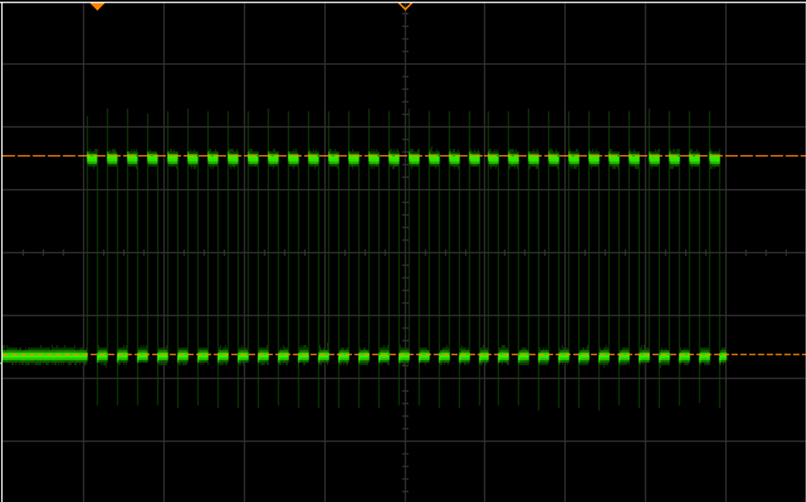
 需要注意的是，为了保证PHY能准确采样，当MAC向MDIO写数据的时候，需要在MDC的上升沿之前就把数据写到MDIO上，要求等待10ns以上再发出一个MDC的上升沿。同样的，为了保证MAC能准确采样，当PHY向MDIO写数据时，PHY要先发出一个MDC的上升沿，然后等待0-300ns（小于上面提到的400ns）再向MDIO写数据。

IEEE 802.3还定义了扩展的SMI数据格式，包括read，write以及set address和readincrement，不过我们在此不做讨论。

PHY和MAC芯片通常都内置MDIO读写的实现，我们只需要按照硬件手册布线，按照软件手册来操作MDIO的读写即可，一般不需要自己实现MDIO的读写操作。

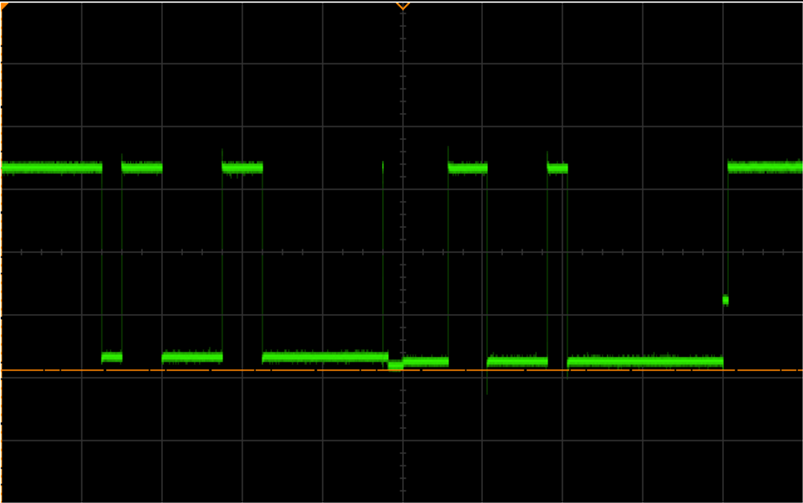
下面以获取和设置某个PHY的速率和双工模式为例，我们来看一下实际MDIO的波形。Register 0是PHYControl寄存器，可以设置PHY的速率和双工模式。

MDC的波形如下：



我在抓波形的时候没有将MDC和MDIO一起抓，所以看不出两者的关系，抱歉~

1. 读取PHY6的register 0：

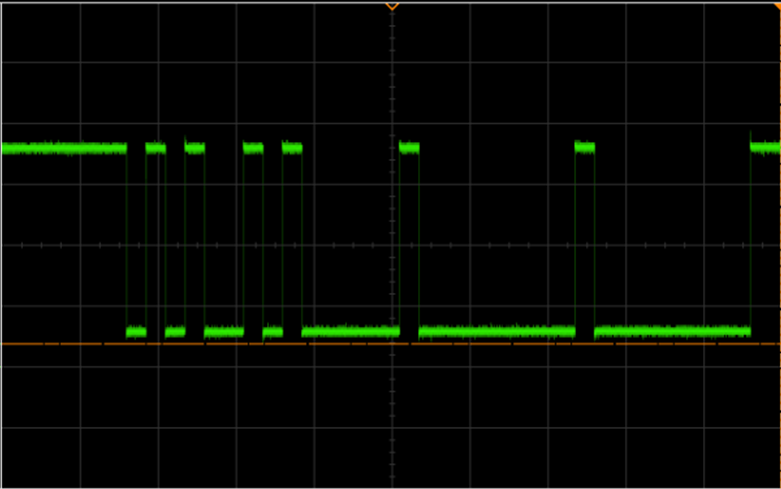


上面中MDIO上的数据为：

http://img.blog.csdn.net/20160610222643999

T表示Turn Around，可以看到在读数据的时候发生了两次TA：一次在MAC发送完REGAD后开始接收PHY的data之前，MDIO由MAC驱动改为PHY驱动；一次在PHY发送完data后，MDIO重新改为MAC驱动的时候。

2. 将PHY5设置为强制10M全双工，设置PHY5的register 0为0x0100：



上图中MDIO上的数据为：

http://img.blog.csdn.net/20160610222855484

写的时候一直都是MAC驱动MDIO，因此不存在MDIO源的改变。

**3. 尝试用GPIO模拟MDC/MDIO**

在某些芯片上可能没有提供MDC/MDIO接口，可以通过GPIO（General Purpose Input/Output）来模拟，GPIO可实现串行输入输出，且一般CPU上会提供很多GPIO接口供用户自定义使用。

每组SMI需要两个GPIO口分别来模拟MDC和MDIO，首先需要保证这两个GPIO口不作其他用途，且相应的复用模式设置为GPIO模式。

模拟MDC很简单，就是将相应GPIO设置为输出模式（由MAC提供时钟），在MDIO上发送和读取数据时提供时钟即可，对时钟频率没有太严格的要求，在前面已做了说明。具体来说，向GPIO写0即为低电平，向GPIO写1即为高电平，电平持续时间可通过usleep等待或执行数条空指令来实现。

模拟MDIO就是实现一套read/write时序，也通过向GPIO写0或写1实现，实现TA则为改变GPIO的输入/输出模式，而read的后半段即为从GPIO上读数据。

需要用到的GPIO寄存器为GPIO\_DIR，GPIO\_DATA和GPIO\_IE：

**GPIO\_DIR**：用于配置GPIO的方向为输入或输出，当GPIO为输入时，MAC可读取GPIO上的数据，当GPIO为输出时，MAC可向GPIO上写数据。

**GPIO\_DATA**：当GPIO为输入时，GPIO\_DATA即为GPIO上的数据，当GPIO为输出时，向GPIO\_DATA写数据即写到了GPIO上。

**GPIO\_IE**：GPIO作为输入时可作为中断源，在用GPIO模拟SMI时，要关闭相应GPIO线的中断。

这三个寄存器的详细用法要参考相应芯片的datasheet。

对于GPIO模拟SMI的具体软件实现，这里不再介绍，有了上述基础知识，应该很容易实现。 对于Linux系统，内核中提供了一个mdio-bitbang.c，里面实现了一套软件实现的MDIO/MDC接口时序可供参考。