

AN10859

软件仿真LPC1700 以太网MII管理(MDIO)

Rev. 01 – 2009年8月6日

应用笔记

文档信息

信息	内容
关键字	LPC1700, 以太网, MII, RMII, MIIM, MDIO
摘要	这个代码样例演示了在LPC1700上如何通过软件仿真以太网MII管理 (MDIO)



版本历史记录

版本	日期	描述
01	20090806	初始版本

联络信息

更多信息，请访问：<http://www.nxp.com>

如欲了解NXP销售办公室地址，请发送电子邮件至：saleaddresses@nxp.com

1. 引言

LPC1700以太网模块包含了全功能的10Mbps或100Mbps以太网MAC（媒体存取控制器），被设计成通过使用DMA硬件加速来提供优化性能。特性包括：一套丰富的控制寄存器，半/全双工操作，流控制，控制帧，硬件加速发送再试，接收包过滤和LAN激活唤醒。通过DMA的分散-收集功能自动收发，减轻了CPU很多运行负担。

这个以太网模块有个接口,连接使用RMII（简化媒体独立接口）协议的片外以太网PHY与片内MIIM（媒体独立接口管理）串行总线，此接口也称作MDIO（管理数据输入/输出）。

MDIO是一种简单的双线串行接口，可访问PHY芯片上的一系列控制和状态寄存器。它包括：2个PIN；管理数据时钟（MDC），其最大时钟速率为2.5MHz（依据标准，虽然一些设备支持更高频率）且无最小速率，以及管理数据输入/输出，是双向的且可同时连接最多32个设备。

Fig 1 表明了MDIO的帧格式。

MII Management Serial Protocol	<idle><sync><start><op code><device addr><reg addr><turnaround><data><idle>
Read Operation	<idle><sync><01><10><AAAA><RRRR><Z0><xxxx xxxx xxxx xxxx><idle>
Write Operation	<idle><sync><01><01><AAAA><RRRR><10><xxxx xxxx xxxx xxxx><idle>

Fig 1. MDIO 帧格式

在读取操作中，站管理实体（EMAC）在MDIO上发送了一串32位紧密相邻的逻辑位给PHY芯片以提供必要的同步。之后，EMAC发送起始位，操作码位，设备地址位（目的PHY芯片的地址），和寄存器地址位（PHY内部寄存器地址）。有了这些信息，被管理实体（PHY芯片）应该提供被请求的数据，但是在此之前，要插入空闲位时间（周转时间）以避免MDIO线上争用。

对于写操作，当所有数据由EMAC发出后，没有MDIO线争用的必要，所以空闲位时间被高位取代，以填充周转时间。

注意：有些PHYs或许不需要同步序列给每个帧，但被需要同步的PHYs涵盖了。

MDIO线需要上拉电阻，在IDLE和周转期间拉高MDIO。

Fig 2表明典型读操作的时序关系。

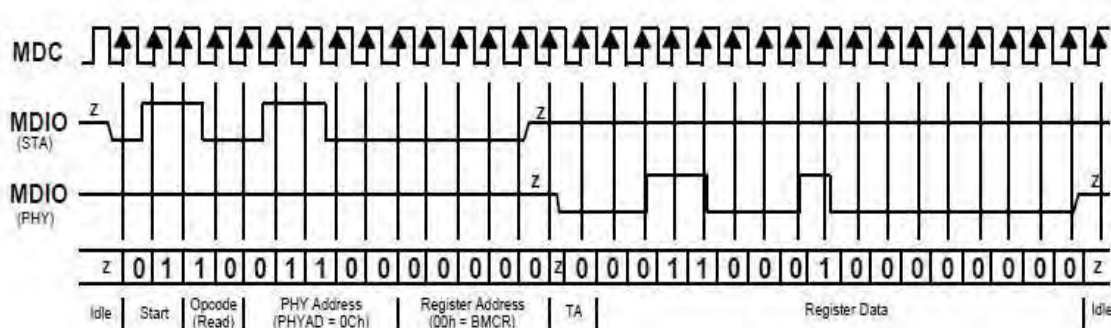


Fig 2. 典型MDIO读操作

Fig 3 表明典型写操作的时序关系。

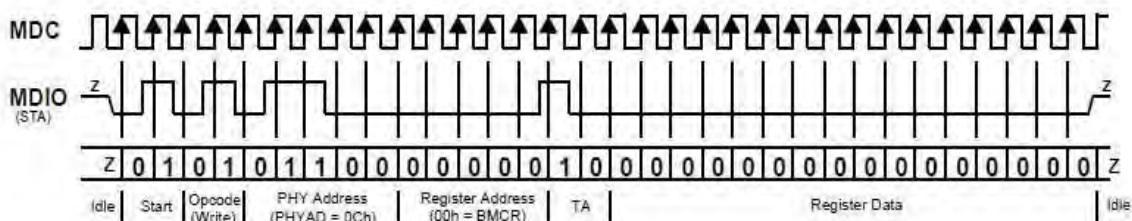


Fig 3. 典型MDIO写操作

2. MDIO软件实现

管理数据输入输出协议由软件实现并且代码包含在`mdio.c`和`mdio.h`文件中。见相关软件zip文件。

`output_MDIO()`函数用来通过要写入的值（比特0或比特1）驱动MDIO线和产生MDC时钟给某个特定位置。Fig 4 给出了这个函数实现。

```
static void output_MDIO (U32 val, U32 n) {  
  
    /* Output a value to the MII PHY management interface. */  
    for (val <= (32 - n); n; val <= 1, n--) {  
        if (val & 0x80000000) {  
            GPIO2->FIOSET = MDIO;  
        }  
        else {  
            GPIO2->FIOCLR = MDIO;  
        }  
        delay ();  
        GPIO2->FIOSET = MDC;  
        delay ();  
        GPIO2->FIOCLR = MDC;  
    }  
}
```

Fig 4. output_MDIO() 函数实现

请注意在这个特定实现中，我们使用P2.8给MDC和P2.9给MDIO，不过这些口也可以改成任何其它GPIO PIN。

*delay()*函数需要根据核运行时的特有频率进行调整。在这个样例中，CPU速率是72MHz，MDC时钟频率大约是2.5MHz。记住这应该是用作标准状态的最高频率，除非所用的特殊PHY支持更高的时钟。

为了在读操作中从PHY芯片读取数据，使用了()函数。它产生16个脉冲给MDC时钟和读由PHY驱动的MDIO线。Fig 5给出了它的实现。

```
static U32 input_MDIO (void) {  
  
    /* Input a value from the MII PHY management interface. */  
    U32 i, val = 0;  
  
    for (i = 0; i < 16; i++) {  
        val <= 1;  
        GPIO2->FIOSET = MDC;  
        delay ();  
        GPIO2->FIOCLR = MDC;  
        if (GPIO2->FIOPIN & MDIO) {  
            val |= 1;  
        }  
    }  
    return (val);  
}
```

Fig 5. input_MDIO()函数实现

*turnaround_MDIO()*函数用来产生空闲位，使MDIO线处于三态状态。代码如图Fig 6所示。

```
static void turnaround_MDIO (void) {  
  
    /* Turnaround MDO is tristated. */  
    GPIO2->FIODIR  &= ~MDIO;  
    GPIO2->FIOSET   = MDC;  
    delay ();  
    GPIO2->FIOCLR   = MDC;  
    delay ();  
}
```

Fig 6. turnaround_MDIO()函数实现

利用以上函数支持，我们可以按照如Fig 1所示的帧格式来再现MDIO读处理。Fig 7给出了代码实现。

```
U32 mdio_read(int PhyReg) {
    U32 val;

    /* Configuring MDC on P2.8 and MDIO on P2.9 */
    GPIO2->FIODIR |= MDIO;

    /* 32 consecutive ones on MD0 to establish sync */
    output_MDIO (0xFFFFFFFF, 32);

    /* start code (01), read command (10) */
    output_MDIO (0x06, 4);

    /* write PHY address */
    output_MDIO (DP83848C_DEF_ADR >> 8, 5);

    /* write the PHY register to write */
    output_MDIO (PhyReg, 5);

    /* turnaround MD0 is tristated */
    turnaround_MDIO ();

    /* read the data value */
    val = input_MDIO ();

    /* turnaround MDIO is tristated */
    turnaround_MDIO ();

    return (val);
}
```

Fig 7. MDIO读处理实现

按同样的方式，我们可以实现MDIO写处理，如图Fig 8所示。

```
void mdio_write(int PhyReg, int Value) {  
  
    /* Configuring MDC on P2.8 and MDIO on P2.9 */  
    GPIO2->FIODIR |= MDIO;  
  
    /* 32 consecutive ones on MD0 to establish sync */  
    output_MDIO (0xFFFFFFFF, 32);  
  
    /* start code (01), write command (01) */  
    output_MDIO (0x05, 4);  
  
    /* write PHY address */  
    output_MDIO (DP83848C_DEF_ADR >> 8, 5);  
  
    /* write the PHY register to write */  
    output_MDIO (PhyReg, 5);  
  
    /* turnaround MDIO (1,0)*/  
    output_MDIO (0x02, 2);  
  
    /* write the data value */  
    output_MDIO (Value, 16);  
  
    /* turnaround MD0 is tristated */  
    turnaround_MDIO ();  
}
```

Fig 8. MDIO写处理实现

最后两个函数，`mdio_read()`和`mdio_write()`，可用来从代码访问PHY寄存器。以下章节给出了使用这些函数的例子。

3. 使用MDIO软件实现的样例

我们将使用EasyWeb样例作为参考，给出通过其中的代码软件实现MDIO接口必需的步骤。代码存在于Keil工程中，且在Keil MCB1750评估板上测试过了。确认E/C和E/U跳线正确配置了，因为它们是连接P2.8和P2.9 到PHY芯片的。

第一步是检测代码是否运行在LPC175x设备上。这种情况下，我们必须通过软件激活MDIO实现。为此，我们使用了IAP（在应用编程）的API调用，它需要以下声明：

```
typedef void (*IAP)(U32 *cmd, U32 *res);  
IAP iap_entry = (IAP)0x1FFF1FF1;  
static char dev_175x;
```


以上代码声明了IAP入口和用作标志的`dev_175x`变量。然后，我们需要进行IAP调用和检查设备是否是LPC175x。所有这些代码是`emac.c`文件中的`init_EMAC()`函数实现的。见Fig 9。

```
U32 pb[2];

dev_175x = __FALSE;
/* Read device ID with IAP */
pb[0] = 54;
iap_entry (&pb[0], &pb[0]);
if ((pb[1] >> 24) == 0x25) {
    /* Use software RMII management routines. */
    dev_175x = __TRUE;
}
```

Fig 9. LPC175x设备检测

接下来的步骤是配置相应的pin连接到模块，也就是说，对于LPC175x，P2.8和P2.9应该配置为GPIO。这两个pin初始配置为输出。Fig 10显示了代码。

```
/* Power Up the EMAC controller. */
SC->PCONP |= 0x40000000;

/* Enable P1 Ethernet Pins. */
PINCON->PINSEL2 = 0x50150105;
if (dev_175x == __FALSE) {
    /* LPC176x devices, no MDIO, MDC remap. */
    PINCON->PINSEL3 = (PINCON->PINSEL3 & ~0x0000000F) | 0x00000005;
}
else {
    /* LPC175x devices, use software MII management. */
    PINCON->PINSEL4 &= ~0x000F0000;
    GPIO2->FIODIR |= MDC;
}
```

Fig 10. 配置pin连接到模块

最后一步是修改`write_PHY()`和`read_PHY()`函数，以实现当设备是LPC175x时，分别调用`mdio_write()`和`mdio_read()`函数。Fig 11和Fig 12显示了实现过程。

```
void write_PHY (int PhyReg, int Value)
{
    unsigned int tout;

    if (dev_175x == __TRUE) {
        /* Software MII Management for LPC175x. */
        mdio_write(PhyReg, Value);
    }
    else {

        EMAC->MADR = DP83848C_DEF_ADR | PhyReg;
        EMAC->MWTD = Value;
        .....
    }
}
```

Fig 11. 调用mdio_write()函数

```
unsigned short read_PHY (int PhyReg)
{
    unsigned int tout, val;

    if (dev_175x == __TRUE) {
        /* Software MII Management for LPC175x. */
        val = mdio_read(PhyReg);
    }
    else {

        EMAC->MADR = DP83848C_DEF_ADR | PhyReg;
        EMAC->MCMD = MCMD_READ;
        .....
    }
}
```

Fig 12. 调用mdio_read()函数

4. 测试EasyWeb样例

为了测试这个样例，在MCB1750板和PC间连上以太网线。板子默认分配了一个静态IP 192.168.0.100，因此PC应该配置相同子网内的任何IP（比如，192.168.0.99）。若需要给板子分配别的IP地址，可以在`tcpip.h`文件中重新设置一个值。

构建然后下载这个样例代码到板子上。在命令行中通过`ping 192.168.0.100`命令测试连通性。如果所有的都正常，打开浏览器，输入`http:// 192.168.0.100`连接到网页服务器。

Fig 13的截图表明MDC时钟频率是2.5MHz。

Fig 14的截图显示了一个MDIO写处理，来自以下指令：

```
/* Put the DP83848C in reset mode */
write_PHY (PHY_REG_BMCR, 0x8000);
```

Op.Code是01(write),PHY地址是0001，寄存器地址是0（BMCR寄存器），所写的的数据是0x8000，意思是这个处理是在bit 15写入1（Reset bit）。

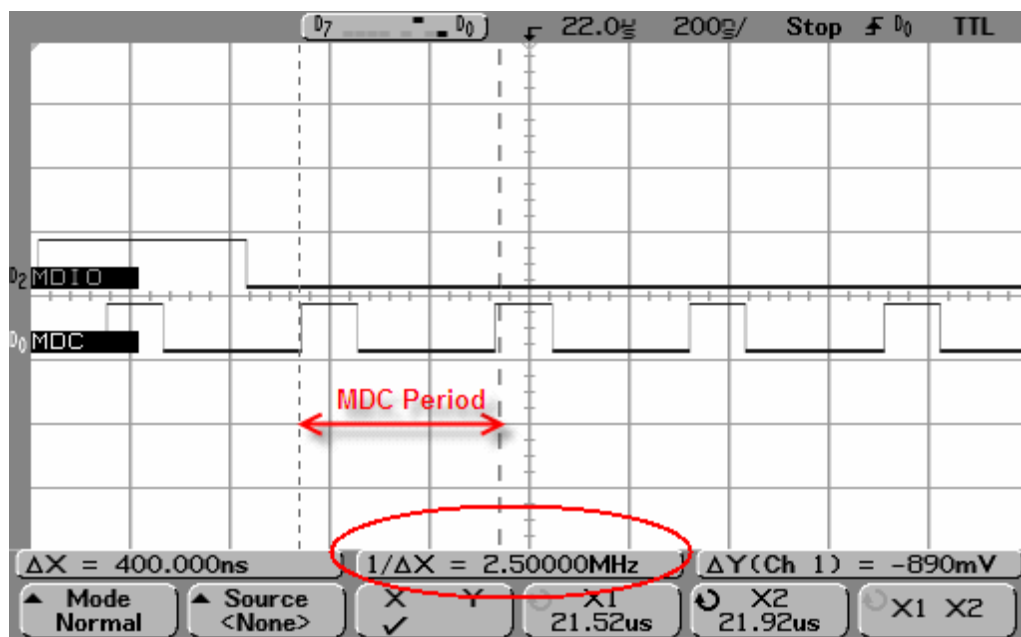


Fig 13. MDC时钟2.5MHz

最后，Fig 15显示了MDIO读处理的截图。这个例子中，Op.Code是10(read),访问的寄存器是PHY Identifier Register #2（地址0x03）,值为0x5C90。

5. 结论

软件MDIO提供了灵活性，允许使用任何可用的GPIO达到此目的。它的实现简单易懂，结合已提供的设备检测机制使用它，硬件和软件MDIO都能为用户很清楚透明地工作。

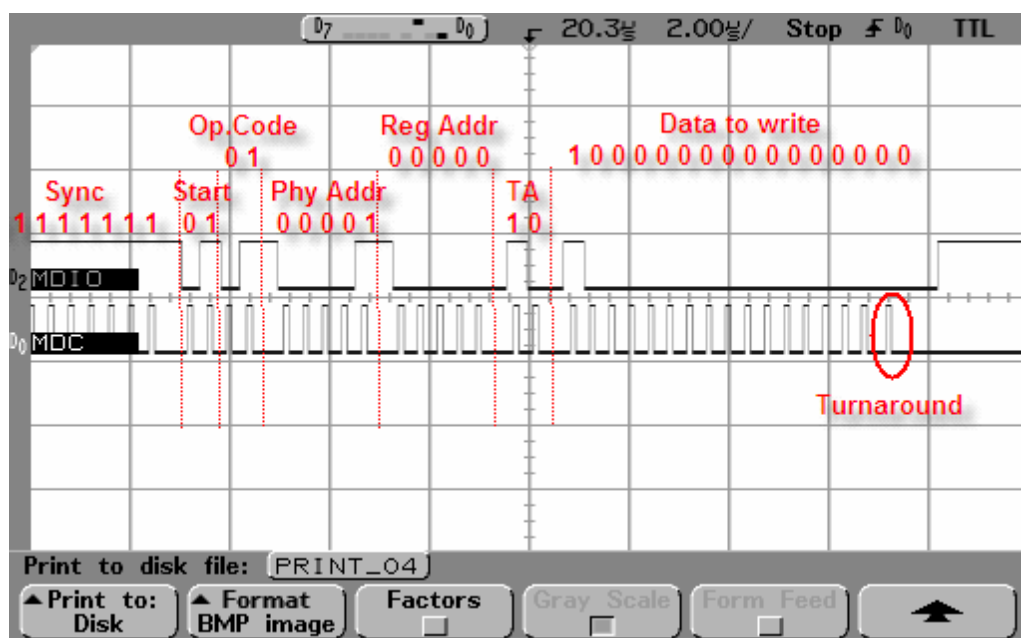


Fig 14. MDIO写处理

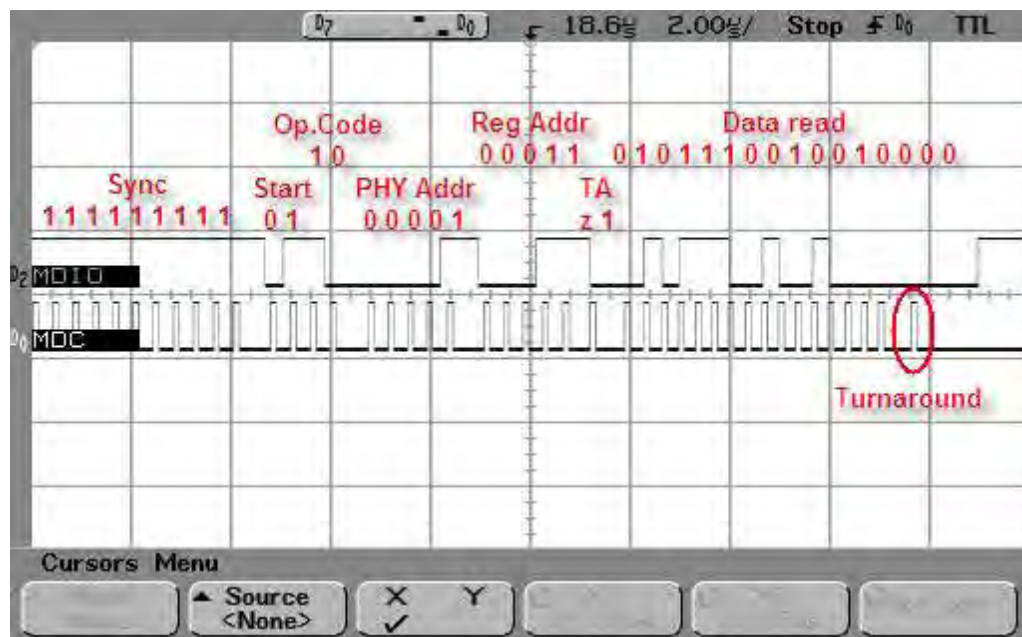


Fig 15. MDIO读处理

6. 免责声明

有限保修和责任— 本文档中的信息被认为是准确和可靠的。然而，对于信息的准确性和完整性，恩智浦半导体公司不给予任何陈述或担保，明示或暗示，对于此类信息的使用后果不负任何责任。

在任何情况下，恩智浦半导体不会承担任何间接、意外发生、惩罚性、特别或相关性的损害赔偿（包括单不限于利润损失、储蓄损失、业务中断、有关去除或更换任何产品的费用或返工费用），不管这些损害赔偿是基于侵权（包括疏忽）、保修、违约合同或其他法律理论。

对于客户无论任何理由可能招致的任何损害，恩智浦半导体为在这里所提到的产品的汇总和累积责任应限制在恩智浦半导体商业销售的条款及条件里面。

变更的权利— 恩智浦半导体有权在任何时间对此文件发布的信息（包括单不限于规格和产品说明）做出任何改动。本文件将取代所有之前所公布的信息。

适用性— 恩智浦半导体产品并非为那些用于对生命和安全有重大关系的系统和设备而设计、授权或提供保证，也不用于那些可以合理预见到的因恩智浦半导体的产品的故障会造成人身伤害、甚至死亡、或是严重的财产或环境损害的应用程序中。恩智浦半导体的产品如果应用在此类的设备或应用程序中，恩智浦半导体对所此造成的风险将不承担任何责任，因此这些风险有客户自行承担。

应用— 在这里所描述有关产品的任何应用程序仅用于说明的目的。在没有进一步的测试或修改的情况下，恩智浦半导体对该应用程序对指定用途是否合适不作任何表示或保证。

客户应对其使用恩智浦半导体产品的应用以及产品的设计和运行自行负责，恩智浦半导体不负责协助应用程序或客户的产品设计。同时，客户应自行负责决定恩智浦产品是否适合客户应用、计划产品、计划的应用程序以及第三方客户使用。客户应提供适当的设计和运行的保障措施以尽量减少其产品与应用的相关风险。

因客户的应用或产品的弱点或缺陷所产生的，或因使用其第三方客户的产品而产生的任何缺陷、损失、费用支出和问题，

恩智浦半导体不承担任何责任。客户应负责为其使用恩智浦半导体芯片的产品或应用以及其第三方客户使用产品或应用做必要的测试，以避免使用不当而造成不必要的损失。恩智浦对在此方面不承担任何责任。

限制值— 超过一个或多个限制值（如在IEC60134的绝对值最大额定值）的施压会对设备造成永久的损害。限制值只强调额定功率，这个设备的操作除了应用在此文件中所提到的“推荐工作条件”和“特征”部分之外，恩智浦半导体不担保超过上述要求的操作。恒定或反复超出限制值将永久地和不可逆转地影响设备的质量和可靠性。

商业销售条件— 恩智浦半导体产品的销售适用公布于<http://www.nxp.com/profile/terms>网站上的通用商业销售条款，除非另存一个单独有效的书面协议，在此种情况下，将适用该单独有效的书面协议之条款和条件。关于客户采购恩智浦半导体产品，恩智浦半导体在此明确拒绝适用客户的通用条款和条件。

不构成任何出售要约或许可— 本文中任何部分都不可被翻译或解释成可以开放接受或授予、转让或任何暗示许可版权、专利或其它工业或知识产权的销售产品要约。

出口控制— 本文件以及其项目描述可能受出口管制条例限制。出口可能需事先获得国家机关许可。

非车规级产品— 除非数据手册明确标出此恩智浦半导体产品为车规级，否则该产品不适合于汽车应用。该产品未在汽车产品测试和应用条件下经测试和质量认证。恩智浦半导体对客户将非车规产品运用在汽车设备和应用中不承担任何责任。

当客户使用该产品设计并在使用在需要车规级规格和标准的汽车应用时，(1) 客户在该汽车应用、使用和规格中使用恩智浦半导体产品时，不在恩智浦半导体对该产品的保证范围内；(2) 当在汽车应用中使用超出恩智浦半导体规格的产品，客户应该自行承担风险；(3) 因客户超标准和产品规格使用恩智浦半导体产品导致的影响、损坏和失效产品索赔，客户不能要求恩智浦半导体进行赔偿。

7. 目录

1. 引言	3
2. MDIO软件实现	4
3. 使用MDIO软件实现的样例	8
4. 测试EasyWeb样例	10
5. 结论	11
6. 免责声明	13
7. 目录	14

This translated version is for reference only, and the English version shall prevail in case of any discrepancy between the translated and English versions.

版权所有 2012恩智浦有限公司 未经许可，禁止转载