

AN10859

LPC1700 Ethernet MII Management (MDIO) via software

Rev. 01 — 6 August 2009

Application note

Document information

| Info | Content |
|-----------------|--|
| Keywords | LPC1700, Ethernet, MII, RMII, MIIM, MDIO |
| Abstract | This code example demonstrates how to emulate an Ethernet MII Management (MDIO) via software on the LPC1700. |

Revision history

| Rev | Date | Description |
|-----|----------|------------------|
| 01 | 20090806 | Initial version. |

Contact information

For additional information, please visit: <http://www.nxp.com>

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

The LPC1700 Ethernet block contains a full featured 10 Mbps or 100 Mbps Ethernet MAC (Media Access Controller) designed to provide optimized performance through the use of DMA hardware acceleration. Features include a generous suite of control registers, half or full duplex operation, flow control, control frames, hardware acceleration for transmit retry, receive packet filtering and wake-up on LAN activity. Automatic frame transmission and reception with Scatter-Gather DMA off-loads many operations from the CPU.

The Ethernet block interfaces between an off-chip Ethernet PHY using the RMII (Reduced Media Independent Interface) protocol and the on-chip MIIM (Media Independent Interface Management) serial bus, also referred to as MDIO (Management Data Input/Output).

MDIO is a simple two-wired serial interface used to access a set of control and status registers inside the PHY chip. It consist of two pins; Management Data Clock (MDC) which has a maximum clock rate of 2.5 MHz (according to the standard, although some devices support higher rates) and no minimum rate, and the Management Data Input/Output (MDIO) which is bidirectional and may be shared by up to 32 devices.

Fig 1 shows the MDIO frame format.

| 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 frame format

In the read operation, the station management entity (EMAC) sends a sequence of 32 contiguous logic ones on MDIO to provide the PHY chip the required synchronization. After this, the EMAC sends the Start bits, Operation Code bits, Device Address bits (address of the destination PHY chip), and the Register Address bits (address of the PHY's internal register). With this information, the managed entity (PHY chip) should provide the requested Data, but previously, an idle bit time (turnaround) is inserted in order to avoid contention on the MDIO line.

For the write operation, as all data is sent by the EMAC, there is no need for MDIO line contention, so the idle bit time is replaced by a high bit, in order to fill the turnaround time.

Note: some PHYs may not require the sync sequence for every frame, but it's included for the PHYs that do require it.

The MDIO line requires a pull-up resistor, pulling MDIO high during IDLE and turnaround.

Fig 2 shows the timing relationship for a typical read operation.

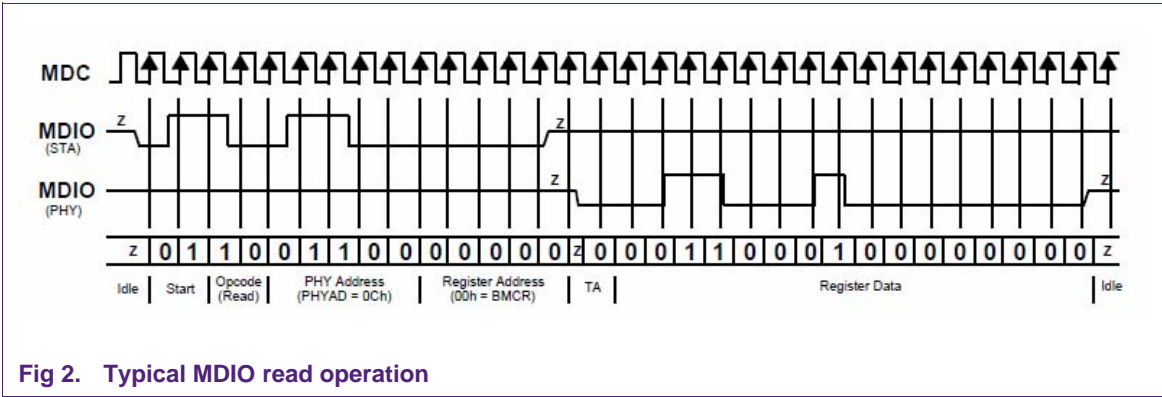
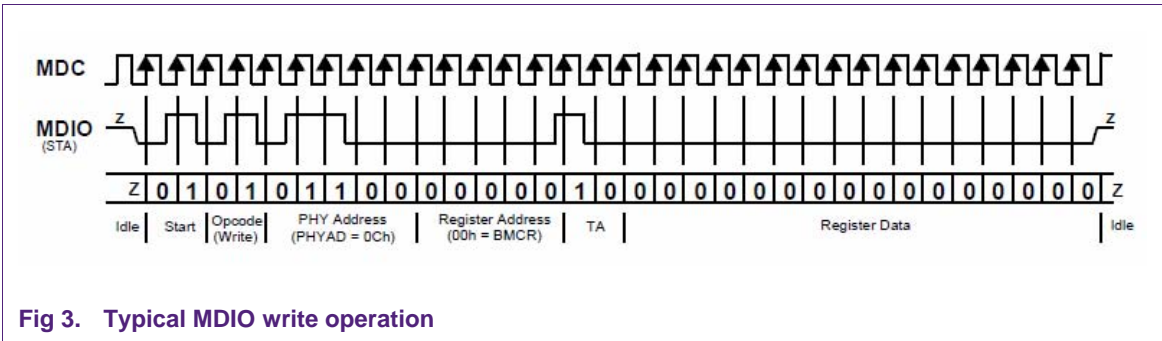


Fig 3 shows the timing relationship for a typical write operation.



2. MDIO software implementation

The Management Data Input/Output protocol is implemented by software and the code is included in *mdio.c* and *mdio.h* files. See the associated software zip file.

The *output_MDIO()* function is used to drive the MDIO line with the desired value (bit 0 or bit 1) and generate the MDC clock for that particular bit. Fig 4 shows this function implementation.

```
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. Implementation of the output_MDIO() function

Please note that in this particular implementation, we are using P2.8 for MDC and P2.9 for MDIO, but these lines can be changed to any other GPIO pins.

The **delay()** function would need adjustments according to the particular frequency at which the core is running. In this example, the CPU speed is 72 MHz, and the MDC clock frequency is about 2.5 MHz. Remember that this should be the maximum frequency used as the standard states, unless the particular PHY used supports higher clocks.

In order to read the data from the PHY chip in a read operation, the **input_MDIO()** function is used. It generates 16 pulses for the MDC clock and reads the MDC line driven by the PHY. [Fig 5](#) shows its implementation.

```
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. Implementation of the input_MDIO() function

The ***turnaround_MDIO()*** function is used to generate the idle bit, where the MDIO line is tri-stated. [Fig 6](#) shows the code.

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

Fig 6. Implementation of the turnaround_MDIO() function

Utilizing the above support functions, we can reproduce the MDIO read transaction following the frame format as shown in [Fig 1](#). [Fig 7](#) shows code implementation.

```
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. Implementing the MDIO read transaction

In the same manner, we can implement the MDIO write transaction, as shown in [Fig 8](#).

```

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. Implementing the MDIO write transaction

The last two functions, ***mdio_read()*** and ***mdio_write()***, can be used in order to access the PHY registers from code. The following section shows an example using these functions.

3. Example using MDIO software implementation

We will use the EasyWeb example as a reference, and show the required steps to implement the MDIO interface via software in that code. The code is available as a Keil project, and was tested using a Keil MCB1750 evaluation board. Ensure that the E/C and E/U jumpers are set appropriately, as those jumpers connect P2.8 and P2.9 pins to the PHY chip.

The first step is to detect if the code is running on an LPC175x device. In such a case, we need to activate the MDIO implementation via software. For this, we are using an IAP (In Application Programming) API call, which requires the following declarations:

```

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

```


The above code declares the IAP entry and the **dev_175x** variable which serves as a flag. Then, we need to make the IAP call and check if the device is an LPC175x. All this code is implemented in the **Init_EMAC()** function in the **emac.c** file. See [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 RMI management routines. */
    dev_175x = __TRUE;
}
```

Fig 9. Detecting the LPC175x device

The next step is to configure the pin connect block accordingly, i.e., for the LPC175x, the P2.8 and P2.9 pins should be configured as GPIO. Initially, both pins are configured as an output. [Fig 10](#) shows the code.

```
/* Power Up the EMAC controller. */
SC->PCOMP |= 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. Configuring the pin connect block

The last step consists of modifying the **write_PHY()** and **read_PHY()** functions, in order to call the **mdio_write()** and **mdio_read()** functions respectively, if the device is an LPC175x. [Fig 11](#) and [Fig 12](#) show these implementations.

```
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. Calling the mdio_write() function

```
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. Calling the mdio_read() function

4. Testing the EasyWeb example

In order to test the example, connect an Ethernet cable between the MCB1750 board and the PC. By default, the board is assigned with a static IP 192.168.0.100, so the PC should be configured with any IP within the same subnet (e.g., 192.168.0.99). If a different IP address is required for the board, the new value can be configured in the **tcpip.h** file.

Build and download the example code to the board. Test the connectivity using the ping 192.168.0.100 command from the command prompt. If everything is working, open the browser and use <http://192.168.0.100> to connect to the webserver.

In [Fig 13](#) the screenshot shows MDC clock with a frequency of 2.5 MHz.

In [Fig 14](#) the screenshot shows an MDIO write transaction, from the following instruction;

```
/* Put the DP83848C in reset mode */
```

```
write_PHY (PHY_REG_BMCR, 0x8000);
```

The Op.Code is 01 (write), the PHY address is 00001, the Register address is 0 (BMCR register), and the data written is 0x8000, which means this transaction is writing 1 in bit 15 (Reset bit).

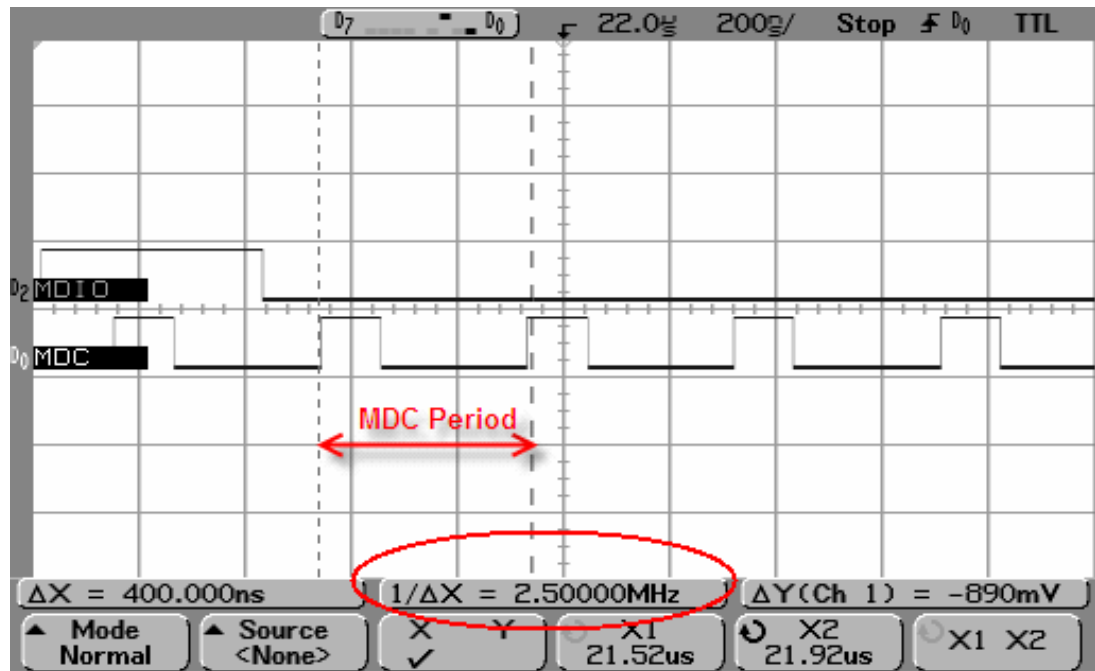


Fig 13. MDC clock of 2.5 MHz

Finally, [Fig 15](#) shows a screenshot of an MDIO read transaction. In this case, the Op.Code is 10 (read), and the addressed register is the PHY Identifier Register #2 (address 0x03), which has a value 0x5C90.

5. Conclusion

Software MDIO provides flexibility as it allows the use of any available GPIO pins for this purpose. Its implementation is straightforward and using it in conjunction with the provided device detection mechanism, both hardware and software MDIO can work transparently for the user.

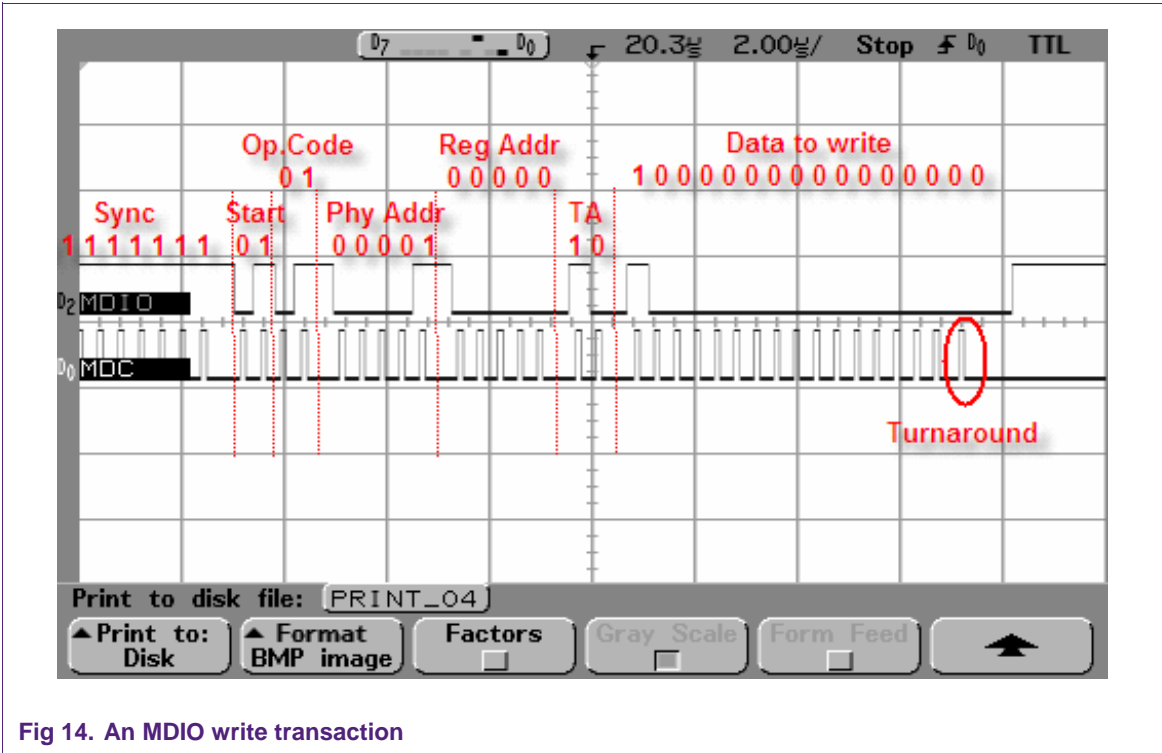


Fig 14. An MDIO write transaction

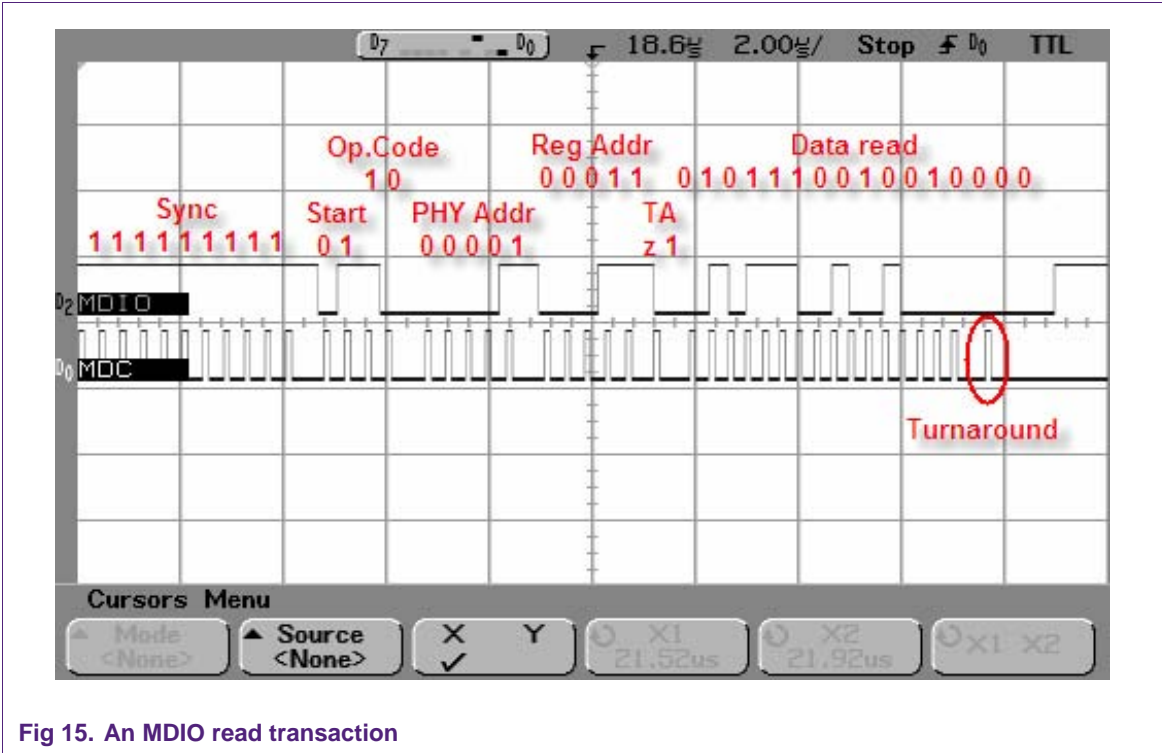


Fig 15. An MDIO read transaction

6. Legal information

6.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

6.2 Disclaimers

General — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of a NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is for the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from national authorities.

6.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

7. Contents

1. Introduction3

2. MDIO software implementation.....4

3. Example using MDIO software implementation
.....8

4. Testing the EasyWeb example 10

5. Conclusion..... 11

6. Legal information 13

6.1 Definitions 13

6.2 Disclaimers..... 13

6.3 Trademarks 13

7. Contents..... 14

Please be aware that important notices concerning this document and the product(s) described herein, have been included in the section 'Legal information'.



© NXP B.V. 2009. All rights reserved.

For more information, please visit: <http://www.nxp.com>
For sales office addresses, email to: salesaddresses@nxp.com