DP83815,DP83816

AN-1351 MAC Address Programming for DP83816 MacPHYTER-II and DP83815 MacPHYTER

Literature Number: SNLA070

MAC Address programming for DP83816 MacPHYTER™II and DP83815 MacPHYTER™

National Semiconductor®
Application Note 1351
November 2004

1.0 Scope

This Application Note explains what a MAC (Media Access Control) Address is, how to obtain one, and finally how to program it into the DP83816 MacPHYTER™II and DP83815 MacPHYTER™ devices. For programming the address, there are two possible methods which the document covers. But first let's define what a MAC Address is.

2.0 MAC Address definition

A MAC (Media Access Control) address is the unique hardware or physical address of a device connected to a network. The MAC address uniquely identifies each node on a network.

Specifically, in Ethernet, the MAC address is known as the Ethernet Address, which is the unique ID serial number of the Ethernet device in one's computer. MAC Addresses are used in a Local Area Network (LAN) by computers to communicate with each other.

More generally this is a standardized data link layer (layer 2) address that is required for every port or device that connects to a LAN. Other devices in the network use these addresses to locate specific ports in the network and to create and update routing tables and data structures.

MAC addresses are 6 bytes long and are controlled by the IEEE. They consist of a 48-bit hexadecimal (hex) number (12 characters), of the form XX-XX-XX-XX-XX-XX, where the X's are either digits or letters from A-F.

The 24 most significant bits of the address are also known as the "Organizationally Unique Identifier (OUI)" or "company_id". When requesting a MAC address, most organizations will have a fixed unique OUI assigned to them. This is then concatenated with another 24 bits to form a unique MAC address also known as a hardware address, MAC-layer address or physical address.

The National Semiconductor Corporation (NSC) 24-bit assigned OUI in hex format is:

08-00-17

An example of a NSC MAC Address in hex format is:

08-00-17-0B-62-35

3.0 Obtaining a MAC Address

To obtain a MAC address for your organization, please refer to the following IEEE links:

http://standards.ieee.org/regauth/oui/forms/

http://standards.ieee.org/regauth/oui/index.shtml

4.0 Programming the MAC Address

On the DP83816 and DP83815, the MAC address can be programmed into the device using either one of two methods.

In the first method, the MAC address is first programmed into the EEPROM which is then loaded into the device. In the second method, the MAC address is programmed directly into the device by using its RFCR (Receive Filter/Match Control Register - offset 48h) and RFDR (Receive Filter/Match Data Register - offset 4Ch) registers.

The use of either method depends on the target application of the device (see Section 5.0).

4.1 USING THE EEPROM

The 1st method involves converting the MAC address into a certain format, so that it can be loaded from the EEPROM into the device. After the MAC Address is converted, it is programmed into the EEPROM's corresponding register bits, shown in Table 1. When an EEPROM load occurs, the address is read and placed into the Perfect Match Register (PMATCH) of the device, also shown in Table 1.

TABLE 1.

EEPROM register address / bit(s)	PMATCH bit(s)
0006h / bit [0]	PMATCH [0]
0007h / bits [15:0]	PMATCH [1:16]
0008h / bits [15:0]	PMATCH [17:32]
0009h / bits [15:1]	PMATCH [33:47]

An EEPROM load automatically takes place when the device is powered up. Additionally, a reload or a manual load of the EEPROM can be forced or initiated by setting bit 2 of PTSCR (PCI Test Control Register - offset 0Ch).

Section 4.1.1 explains how the conversion is done.

4.1.1 MAC address conversion

Consider the following MAC Address in hex format:

08-00-17-0B-62-35

where Octet 0 or the Most Significant Byte (MSB) is 08h, and Octet 5 or the Least Significant Byte (LSB) is 35h, as shown in Table 2.

TABLE 2.

Octet 0	Octet 1	Octet 2	Octet 3	Octet 4	Octet 5
08	00	17	0B	62	35

Table 3 shows the MAC address octets converted into binary by listing out each bit value.

TABLE 3.

Octet	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Octet 0	0	0	0	0	1	0	0	0
Octet 1	0	0	0	0	0	0	0	0
Octet 2	0	0	0	1	0	1	1	1
Octet 3	0	0	0	0	1	0	1	1
Octet 4	0	1	1	0	0	0	1	0
Octet 5	0	0	1	1	0	1	0	1

In Table 4, the orientation of the bits is swapped in each octet as such:

TABLE 4.

Octet	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Octet 0	0	0	0	1	0	0	0	0
Octet 1	0	0	0	0	0	0	0	0
Octet 2	1	1	1	0	1	0	0	0
Octet 3	1	1	0	1	0	0	0	0
Octet 4	0	1	0	0	0	1	1	0
Octet 5	1	0	1	0	1	1	0	0

Finally all the bits are shifted to the left by 1 and mapped to the PMATCH (PM) register as shown in Table 5:

TABLE 5.

PM bits	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
[0]								0
[1:8]	0	0	1	0	0	0	0	0
[9:16]	0	0	0	0	0	0	0	1
[17:24]	1	1	0	1	0	0	0	1
[25:32]	1	0	1	0	0	0	0	0
[33:40]	1	0	0	0	1	1	0	1
[41:47]	0	1	0	1	1	0	0	

The octets are then converted back to hex format and assigned into the proper EEPROM locations as shown in Table 6:

TABLE 6.

EEPROM register address / bits	PMATCH bit(s)	Converted MAC Address
0006h / bit [0]	PMATCH [0]	0
0007h / bits [15:0]	PMATCH [1:16]	2001
0008h / bits [15:0]	PMATCH [17:32]	D1A0
0009h / bits [15:0] (assuming bit 0 = 0)	PMATCH [33:47]	8D58

An excel sheet that does the above conversion calculations can be requested online through the following link:

http://www.national.com/feedback/newfeed.nsf/Techsupport?openform

Alternatively, NSC's DOS Console Diagnostic utility, used to program the EEPROM registers including the MAC address, can be downloaded at:

http://www.national.com/appinfo/networks/files/diag_exec.zip

4.1.2 EEPROM Programming

To program the EEPROM, one of two methods are available. Either using an EEPROM programmer or using the DP83816 (or DP83815) itself to do the programming.

In the DP83816 (or DP83815), the MEAR (EEPROM Access Register - offset 08h) controls the EEPROM pins on the device. MEAR[3] (EESEL: EEPROM Chip Select bit) controls the value of the EESEL pin (pin 128). MEAR[2] (EECLK: EEPROM Serial Clock bit) controls the value of the EECLK pin (pin 2). MEAR [1] (EEDO: EEPROM Data Out bit) returns the current state of the EEDO pin (pin 138). MEAR [0] (EEDI: EEPROM Data In bit) controls the value of the EEDI pin (pin 1).

An example of using those bits is as follows:

write 00000008h to MEAR ** Chip select high

write 00000008h to MEAR ** CLK LOW and Data in 0

write 0000000Ch to MEAR ** CLK HIGH and Data out 0

write 00000009h to MEAR ** CLK LOW and Data in 1

write 0000000Eh to MEAR ** CLK HIGH and Data out 1

write 00000000h to MEAR ** Chip select low

Please refer to sec. 4.2.3 in the datasheet for more details on the MEAR register. Appendix A contains sample C++ code showing how the EEPROM is accessed through the device. This is a subset of the Diag utility code available at the following web link:

http://www.national.com/appinfo/networks/macphyter2.html

under "DOS Console Diagnostic (Object and Source files)".

4.1.3 Reference EEPROM

Please use the FM93C46 EEPROM, referenced in the DP83815 and DP83816 datasheets, or a similar part. This EEPROM has a MICROWIRE™ Synchronous Bus interface and uses a special instruction set to communicate with the device.

4.2 USING RFCR AND RFDR

The 2nd method uses the RFCR and RFDR registers to write the MAC address into the device as shown in the example below:

1) Consider the same MAC Address in hex format:

08-00-17-0B-62-35

where Octet 0 or the Most Significant Byte (MSB) is 08h, and Octet 5 or the Least Significant Byte (LSB) is 35h, as shown in Table 2.

TABLE 7.

Octet 0	Octet 1	Octet 2	Octet 3	Octet 4	Octet 5
08	00	17	0B	62	35

2) To program the address into the PMATCH register:

- write 0000h to RFCR (offset 48h) (this selects the appropriate internal receive filter for Octets 1-0)
- write 0008h (Octets 1-0) to RFDR (offset 4Ch)
- write 0002h to RFCR (offset 48h) (this selects the appropriate internal receive filter for Octets 3-2)
- write 0B17h (Octets 3-2) to RFDR (offset 4Ch)
- write 0004h to RFCR (offset 48h) (this selects the appropriate internal receive filter for Octets 5-4)
- write 3562h (Octets 5-4) to RFDR (offset 4Ch)

3) To read the address from the device:

- write 0000h to RFCR (offset 48h)
- read RFDR (offset 4Ch), this will give Octets 1-0 of the MAC address (flip them to read Octets 0-1)
- write 0002h to RFCR (offset 48h)
- read RFDR (offset 4Ch), this will give Octets 3-2 of the MAC address (flip them to read Octets 2-3)
- write 0004h to RFCR (offset 48h)
- read RFDR (offset 4Ch), this will give Octets 5-4 of the MAC address (flip them to read Octets 4-5)

Note that the MAC address is read out as such:

00-08-0B-17-35-62

and each set of octets has to be properly flipped so the actual address, 08-00-17-0B-62-35, can be obtained.

Since the RFCR also controls the receive filter, once the MAC address programming is complete, the RFCR needs to be re-configured with the correct settings for the receive operation to function properly.

5.0 Summary

The first method, using an EEPROM, is targeted for applications that use generic, non-unique software found in most applications that usually do not require any software modifications.

The second method, using the RFCR and RFDR registers, targets applications/systems that require unique software codes for each device.

Since the IEEE standards require every single node to have a unique MAC address, the application will define the optimal programming method to be used.

Appendix A

```
/*********************************************************************
*
* EEPROM.C
*
* This is a collection of routines to interface to the EEPROM
*  for the MacPhyter Chip
*
*********************************************************************/

// Include files

#include <dos.h>
#include <time.h>
#include <sys\timeb.h>
#include <stdio.h>
#include <conio.h>
#include <memory.h>
#include <time.h>
#include "GLOBAL.H"
#include "NICSTUFF.H"
#include "PCIBIOS.H"

// Function prototypes

#include "FUNCTION.H"

UINT16 eepromDefaultValue[11] = {
                    0xD008,         // 0000h, Subsystem Vendor ID
                    0x0400,         // 0001h, Subsystem Device ID
                    0x2CD0,         // 0002h, Minimum Grant, Maximum Latency
                    0xCF82,         // 0003h, Configuration/Power Management Stuff
                    0x0000,         // 0004h, SecureOn Password
                    0x0000,         // 0005h, SecureOn Password
                    0x0000,         // 0006h, SecureOn Password
                    0x0000,         // 0007h, Ethernet ID
                    0x0000,         // 0008h, Ethernet ID
                    0x0000,         // 0009h, Ethernet ID
                    0xA098          // 000Ah, WOL, Receive Filter/Match
};
```

```
UINT32 OUID; // Organizationally Unique Identifier
UINT32 startMACAddr;
UINT32 endMACAddr;
UINT32 currentMACAddr;

// Globals
extern unsigned short RegBase;
extern unsigned short batchMode;

extern UINT16 flip_16 (UINT16 us);

/************************************************
*       Function:   DisplayEEPROM
*       Purpose:display EEPROM contents
*       Returns:    void
************************************************/
void DisplayEEPROM()
{
        int i=0;
        WORD value;

        for( i=0; i<0xC; i++ )
        {
                ReadEEWord( RegBase + MEAR, i, &value );

                printf( "\n%01X= %04X  ", i, value );
        }

        printf( "\n\n" );
}

/************************************************
*       Function:   ReadMACAddress
*       Purpose:    load readable form of the MAC address
*       Returns:    void
************************************************/
void ReadMACAddress(  UINT16 *nicAddress )
{
        UINT32 i;
        UINT16 mask;
```

```
        UINT16 word1 = 0;
        UINT16 word2 = 0;
        UINT16 word3 = 0;
        UINT16 word4 = 0;

//
// Read 16 bit words 6 - 9 from the EEProm.  They contain the hardwares MAC
// address in a rather cryptic format.
//
ReadEEWord( RegBase + MEAR, 0x06, &word1 );
ReadEEWord( RegBase + MEAR, 0x07, &word2 );
ReadEEWord( RegBase + MEAR, 0x08, &word3 );
ReadEEWord( RegBase + MEAR, 0x09, &word4 );

//
// Decode the cryptic format into what we can use a word at a time.
//
nicAddress[ 0 ] = word1 & 1;
nicAddress[ 1 ] = word2 & 1;
nicAddress[ 2 ] = word3 & 1;

i = 15;
mask = 0x2;
while ( i-- )
        {
                if ( word2 & 0x8000 )
                {
                        nicAddress[ 0 ] |= mask;
                }
                word2 = word2 << 1;
                mask = mask << 1;
}

i = 15;
mask = 0x2;
        while ( i-- )
        {
                if ( word3 & 0x8000 )
                {
                        nicAddress[ 1 ] |= mask;
                }
```

```
                word3 = word3 << 1;
                mask = mask << 1;
    }

    i = 15;
    mask = 0x2;
    while ( i-- )
        {
                if ( word4 & 0x8000 )
                {
                        nicAddress[ 2 ] |= mask;
                }
                word4 = word4 << 1;
                mask = mask << 1;
    }
}

/************************************************
*       Function:WriteMACAddress
*       Purpose:Write readable form of the MAC address to EEPROM
*       Returns: 1 if FAIL, 0 if Succees
************************************************/
int writeMacAddress( unsigned *nicAddress )
{
        UINT32      mask;
        UINT32      I;
        int         nbits = 9;
        unsigned    value, sixOff, nineOff, checkSum = 0, word1, word2, word3;
        unsigned    sword[4];

    ReadEEWord( RegBase + MEAR, 0x06, &sixOff );
        sixOff &= ~1;

    ReadEEWord( RegBase + MEAR, 0x09, &nineOff );
        nineOff &= 1;

    word1 = nicAddress[ 0 ];
    word2 = nicAddress[ 1 ];
    word3 = nicAddress[ 2 ];

    writeEeEnable();
```

```
        value = sixOff;
        if( word1 & 1 ) value |= 1;
sword[0] = value;
WriteEEWord( RegBase + MEAR, 0x06, value );

        value = 0;
l = 15;
mask = 0x2;
        while( l-- )
        {
                if( word1 & 0x8000 ) value |= mask;
                mask <<= 1;
                word1 <<= 1;
        }

if( word2 & 1 ) value |= 1;
sword[1] = value;
WriteEEWord( RegBase + MEAR, 0x07, value );

        value = 0;
l = 15;
mask = 0x2;
        while( l-- )
        {
                if( word2 & 0x8000 ) value |= mask;
                mask <<= 1;
                word2 <<= 1;
        }

if( word3 & 1 ) value |= 1;
sword[2] = value;
WriteEEWord( RegBase + MEAR, 0x08, value );

value = nineOff;
l = 15;
mask = 0x2;
        while( l-- )
        {
                if( word3 & 0x8000 ) value |= mask;
                mask <<= 1;
```

```
                word3 <<= 1;
        }

    sword[3] = value;
    WriteEEWord( RegBase + MEAR, 0x09, value );

    writeEeDisable();

    for( l=0; l<4; l++ )
    {
        ReadEEWord( RegBase + MEAR, 6+l, &value );
                if( value != sword[l] )
                {
                        printf( "\nEEPROM programming failed!  Enter Alternate MAC address.\n" );
                        break;
                }
    }

    if (doEeCheckSum())
        {
                return 1;
        }

        if (batchMode)
        {
                return writeMacFile();
        }

        return 0;
}

int writeID( int ssId, BYTE venDev )
{
        int idOffset = 0;

        // Calculate the offset:
        if (venDev == 'D') idOffset = 1;

    writeEeEnable();
    WriteEEWord( RegBase + MEAR, idOffset, ssId );
    writeEeDisable();
```

```
        return doEeCheckSum();
}

/************************************************
*       Function:   WriteEEWord
*       Purpose:    write a 16-bit value into the specified eeprom location
*       Returns:    void
************************************************/
void WriteEEWord( unsigned short pMear, UINT32 wordOffset, UINT16 value )
{
        UINT32 mask;
        UINT32 l;
        int     nbits = 9;
        UINT32 invalue = (UINT32) value;

   outpd( RegBase + MEAR, 0 );          /* clock out CS low */
   mdelay( 5 );

   outpd( RegBase + MEAR, EECLK );
   mdelay( 5 );

   invalue = ((UINT32)( 0x0140|wordOffset) << 16 ) | invalue;

   nbits = 25;

   mask = ((UINT32)1) << (nbits-1);

   while (nbits--)
        {
      /* assert chip select, and setup data in */
      l = ( invalue & mask ) ? EECS | EEDI : EECS;

      outpd( RegBase + MEAR, l );
           mdelay( 5 );

      outpd( RegBase + MEAR, l | EECLK );
      mdelay( 5 );

      mask >>= 1;
   }
```

```
    // IOW32( mear, 0 );          /* terminate write  */
    outpd( RegBase + MEAR, 0 );              // terminate operation
    mdelay( 5 );

    /* wait for operation to complete */
    outpd( RegBase + MEAR, EECS );       /* assert CS */
    mdelay( 5 );

    outpd( RegBase + MEAR, EECS | EECLK );
    mdelay( 5 );

    for (I = 0; I < 10; I++)
            {
       mdelay( 5 );
       if( inpd( RegBase + MEAR ) & EEDO )
          break;
    }

    outpd( RegBase + MEAR, 0 );          /* clock out CS low */
    mdelay( 5 );

    outpd( RegBase + MEAR, EECLK );
    mdelay( 5 );
}

/***********************************************
*           Function:   checkSumCheck
*           Purpose:    Use the EEBIST bit on MacPhyter to
*                                       validate the EEPROM checksum
*           Returns:    1 if FAIL, 0 if Succees
***********************************************/
int checkSumCheck()
{
           unsigned long pciTestReg = EEBIST_EN;

           outpd( RegBase + PCITEST_CNTRL, EEBIST_EN );

           while (pciTestReg & EEBIST_EN)
           {
                      pciTestReg = inpd( RegBase + PCITEST_CNTRL );
```

```
        }

        if (inpd( RegBase + PCITEST_CNTRL ) & EEBIST_FAIL)
        {
                printf( "\n!!WARNING:  Checksum FAILED!\n" );
                return( 1 );
        }
        else
        {
                printf( "\nEEBIST Test Passes!\n" );
        }

        return( 0 );
}

/***********************************************
*       Function:   ReadEEWord
*       Purpose:    Read 16-bit value from the specified location
*       Returns:    void
***********************************************/
void ReadEEWord( unsigned short pMear, UINT32 offset, UINT16* pEeData )
{
        UINT16 value;

   outpd( RegBase + MEAR, 0 );
   mdelay( 5 );

   outpd( RegBase + MEAR, EECLK );      // clock out no chipselect
   mdelay( 5 );

   eeput( (UINT32)(EEread|offset), 9 );

   value = eeget( 16 );

   outpd( RegBase + MEAR, 0 );          // terminate read
   mdelay( 5 );

   outpd( RegBase + MEAR, EECLK );      // clock out no chipselect
   mdelay( 5 );

   *pEeData = ( UINT16 )value;
```

```
    outpd( RegBase + MEAR, 0 );          // terminate read
    mdelay( 5 );
}

/************************************************
*           Function:   EEPUT
*           Purpose:    Clock OUT data to EEPROM
*           Returns:    void
************************************************/
void eeput( UINT32 value, int nbits )
{
            UINT32 mask = ((UINT32)1) << (nbits-1);
            UINT32 l;

    while( nbits-- )
            {
        // assert chip select, and setup data in
        l = ( value & mask ) ? EECS | EEDI : EECS;
        outpd( RegBase + MEAR, l );
        mdelay( 5 );

        outpd( RegBase + MEAR, l | EECLK );
        mdelay( 5 );

        mask >>= 1;
    }
}

/************************************************
*           Function:   EEGET
*           Purpose:    Clock IN data from EEPROM
*           Returns:    void
************************************************/
UINT16 eeget( int nbits )
{
            UINT16 mask = 1 << (nbits-1);
            UINT16 value = 0;

    while( nbits-- )
            {
```

```
        // assert chip select, and clock in data
        outpd( RegBase + MEAR, EECS );
        mdelay( 5 );

        outpd( RegBase + MEAR, EECS | EECLK );
                mdelay( 5 );

        if( inpd( RegBase + MEAR ) & EEDO )
                value |= mask;
                mdelay( 5 );

        mask >>= 1;
    }

    return( value );
}

/***********************************************
*           Function:   WriteEeEnable
*           Purpose:    ENABLE EEPROM Write Mode
*           Returns:    void
***********************************************/
void writeEeEnable()
{
            UINT32 mask;
            UINT32 l;
            int         nbits = 9;
    //
    // Start the sequence with Chip Select 0 for a 4 microsecond cycle.
    //
    outpd( RegBase + MEAR, 0 );
    mdelay( 5 );

    outpd( RegBase + MEAR, EECLK );
    mdelay( 5 );

    mask = ((UINT32)1) << (nbits-1);

    while( nbits-- )
            {
        /* assert chip select, and setup data in */
```

```
        l = ( 0x0130 & mask ) ? EECS | EEDI : EECS;
        outpd( RegBase + MEAR, l );
        outpd( RegBase + MEAR, l | EECLK );
        mask >>= 1;
    }

    outpd( RegBase + MEAR, 0 );             // terminate operation
    mdelay( 5 );

    //IOW32( mear, 0 );          /* clock out CS low */
    outpd( RegBase + MEAR, EECLK );
    mdelay( 5 );
}

/************************************************
*           Function:   WriteEEDisable
*           Purpose:    Disable EEPROM write Mode
*           Returns:    void
************************************************/
void writeEeDisable()
{
            UINT32 mask;
            UINT32 l;
            int         nbits = 9;

    //ed->eeput( (uint32)EEwriteDisable, 9 );
    nbits = 9;
    mask = ((UINT32)1) << (nbits-1);

    while( nbits-- )
            {
        /* assert chip select, and setup data in */
        l = ( 0x0100 & mask ) ? EECS | EEDI : EECS;
        outpd( RegBase + MEAR, l );
        outpd( RegBase + MEAR, l | EECLK );
        mask >>= 1;
    }

    outpd( RegBase + MEAR, 0 );             // terminate operation
}
```

```
/***********************************************
*       Function:   doEeCheckSum()
*       Purpose:    Read EEPROM contents, calculate checksum,
*                                   write it, verify the write and validate it.
*       Returns:    1 if FAIL, 0 if Succees
***********************************************/
int doEeCheckSum()
{
        int I;
        WORD value, checkSum;
        unsigned char csLow, csHigh, csSum, csVals[ 11 ];

  // Calculate Checksums:
  for( I=0; I<11; I++ )
  {
        ReadEEWord( RegBase + MEAR, I, &value );

        csLow = value & 0xff;
                value >>= 8;
        csHigh = value & 0xff;

        csVals[ I ] = csLow + csHigh;
  }

  // Calculate Checksums:
  csSum =  0;
  for( I=0; I<11; I++ )
        {
                csSum += csVals[ I ];
        }

  csSum += 0x55;
  checkSum = (~csSum) + 1;
  checkSum = ( checkSum << 8 ) + 0x55;

  writeEeEnable();
  WriteEEWord( RegBase + MEAR, 0x0B, checkSum );
  writeEeDisable();

  ReadEEWord( RegBase + MEAR, 0xB, &value );
  if (value != checkSum)
```

```
        {
                printf( "\n!!WARNING Checksum not programmed correctly!\n" );
                printf( "\n  checkSUM: %04X  actual: %04X", checkSum, value );
        }

   if (checkSumCheck()) // Invalid checksum
        {
                return 1;
        }

        outpd( RegBase + PCITEST_CNTRL, EELOAD_EN ); // Reload configuration information from EEPROM
        delay( 1 );

        return 0;
}

/*****************************************************
*       Function:   WriteEeDefaults:
*       Purpose:    Initialize the EEPROM without touching the
*                                   MAC address.
*
*       Returns:    1 if FAIL, 0 if Succees
*
*     The Following was taken from EPMON Programming script:
*
*     eeprom  0     d008          # subsystem vendor
*     eeprom  1     0400          # subsys id
*     eeprom  2     2cd0          # min gnt, max lat
*     eeprom  3     cf82          # config stuff
*     eeprom  4     0000          # secure on pw stuff
*     eeprom  5     0000          # secure on pw stuff
*     eeprom  6     0000          # LSb is part of Ethernet ID
*     eeprom  7     2001          # Ethernet ID
*     eeprom  8     d1a1          # Ethernet ID
*     eeprom  9     $Loc9         # Ethernet ID
*     eeprom  a     a098
*     eeprom  b     $value        # checksum
*****************************************************************/
int writeEeDefaults()
{
        FILE *stream;
```

```
        char buf[120];
        UINT16 eepromValue[11];
        UINT32 i;
        UINT16 value;
        unsigned char inputString[25];
        int nicAddress[3];

        UINT16 Addressbit;

        // Read EEPROM values from file if the file exists, otherwise set as
        // default values
if ((stream = fopen("EEPROM.TXT", "rt")) != NULL)
        {
                for (i = 0; i <= 0xA; i++)
                {
                        fgets(buf, sizeof(buf), stream);
                        sscanf(buf, "%04X", &eepromValue[i]);
                }
                fclose(stream);
        }
        else
        {
                for (i = 0; i <= 0xA; i++)
                {
                        eepromValue[i] = eepromDefaultValue[i];
                }
        }

        // Read subsystem ID's from file if the file exists, and convert the ID's
        // in readable form to the format that DP83815/816 uses
if ((stream = fopen("SUBSYSID.TXT", "rt")) != NULL)
        {
                for (i = 0; i <= 1; i++)
                {
                        fgets(buf, sizeof(buf), stream);
                        sscanf(buf, "%04X", &value);
                        eepromValue[i] = flip_16(value);
                }
                fclose(stream);
        }
```

```
writeEeEnable();

        // Write into EEPROM
        for (i = 0; i <= 0xA; i++)
        {
                if (i <=5 || i == 0xA)
                {
                        WriteEEWord( RegBase + MEAR, i, eepromValue[i] );
                }
        }

// Save the least significant bit for the MAC address:
ReadEEWord( RegBase + MEAR, 0x6, &Addressbit );
Addressbit &= 1;
WriteEEWord( RegBase + MEAR, 0x6, Addressbit );

// Clear the least significant bit for the MAC address:
ReadEEWord( RegBase + MEAR, 0x9, &Addressbit );
Addressbit &= 0xfffe;
WriteEEWord( RegBase + MEAR, 0x9, Addressbit );

writeEeDisable();

        // Check if the EEPROM values are written correctly
        for (i = 0; i <= 0xA; i++)
        {
                if (i <= 5 || i == 0xA)
                {
                        ReadEEWord( RegBase + MEAR, i, &value );
                        if (value != eepromValue[i])
                        {
                                printf("EEPROM values are not written correctly!\n");
                                return 1;
                        }
                }
        }

        // Write MAC address also if it is in batch mode and the MAC address
        // file exists
        if (batchMode)
        {
```

```
                if (readMacFile(inputString))
                {
                        return 1;
                }
                nicAddress[0] = (UINT16)*( inputString+2 );
                nicAddress[0] = (nicAddress[0] << 8) + (UINT16)*( inputString );
                nicAddress[1] = (UINT16)*( inputString+6 );
                nicAddress[1] = (nicAddress[1] << 8) + (UINT16)*( inputString+4 );
                nicAddress[2] = (UINT16)*( inputString+10 );
                nicAddress[2] = (nicAddress[2] << 8) + (UINT16)*( inputString+8 );
                if (writeMacAddress(nicAddress))
                {
                        return 1;
                }

                // Call findNIC routine again to display the new address
                if (findNIC(TRUE))
                {
                        return 1;
                }

                if (currentMACAddr == endMACAddr)
                {
                        return 1;
                }

                return 0;
        }
        else
        {
                return doEeCheckSum();
        }
}

/***********************************************
*       Function:   readMacFile
*       Purpose:Opens up a file, retrieves MacAddress,
*                                       increments it and returns the new number.
*       Returns:    1 = FAIL; 0 = SUCCESS; loads the input;
***********************************************/
int readMacFile( unsigned char *inputString )
```

```
{
        FILE *stream;
        char buf[120];

        // Open up MACADDR.TXT file in the cwd
  if ((stream = fopen("MACADDR.TXT", "rt")) != NULL)
        {
                fgets(buf, sizeof(buf), stream);
                sscanf(buf, "%06lX%06lX", &OUID, &startMACAddr);
                fgets(buf, sizeof(buf), stream);
                sscanf(buf, "%06lX%06lX", &OUID, &endMACAddr);
                fgets(buf, sizeof(buf), stream);
                sscanf(buf, "%06lX%06lX", &OUID, &currentMACAddr);
                fclose(stream);
        }
        else
        {
           printf("\nUnable to open up MACADDR.TXT in current working directory.");
           return 1;
        }

        // Check if the MAC address is valid
        if (((OUID == 0x000000) && (currentMACAddr == 0x000000))
                || ((OUID == 0xFFFFFF) && (currentMACAddr == 0xFFFFFF)))
        {
           printf("\nInvalid MAC address: %06lX%06lX", OUID, currentMACAddr);
                return 1;
        }
        else if ((currentMACAddr < startMACAddr) || (currentMACAddr > endMACAddr))
        {
           printf("\nMAC address is out of the range!");
                return 1;
        }

        sscanf(buf, "%02X%02X%02X%02X%02X%02X",
         inputString,
         (inputString+2),
         (inputString+4),
         (inputString+6),
         (inputString+8),
         (inputString+10)
```

```
        );

        return 0;
}


/***********************************************
*       Function:   writeMacFile
*       Purpose:Opens up a file, writes the new number.
*       Returns:    1 = FAIL; 0 = SUCCESS; loads the input;
***********************************************/
int writeMacFile(void)
{
        FILE *stream;

        // Stop program if current MAC address reachs the end MAC address
        if (currentMACAddr < endMACAddr)
        {
                currentMACAddr++;

                // Open up MACADDR.TXT file in the cwd
                if ((stream = fopen("MACADDR.TXT", "wt")) != NULL)
                {
                        fprintf(stream, "%06lX%06lX\t\t// Start MAC Address\n", OUID, startMACAddr);
                        fprintf(stream, "%06lX%06lX\t\t// End MAC Address\n", OUID, endMACAddr);
                        fprintf(stream, "%06lX%06lX\t\t// Current MAC Address\n", OUID, currentMACAddr);
                        fclose(stream);
                }
                else
                {
                        printf("\nUnable to open up MACADDR.TXT in current working directory.");
                        return 1;
                }
        }
        else
        {
           printf("\nEnd MAC address is reached.\n");
        }

        return 0;
}
```

National Semiconductor
Americas Customer
Support Center
Email: new.feedback@nsc.com
Tel: 1-800-272-9959

www.national.com

National Semiconductor
Europe Customer Support Center
Fax: +49 (0) 180-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 69 9508 6208
English Tel: +44 (0) 870 24 0 2171
Français Tel: +33 (0) 1 41 91 8790

National Semiconductor
Asia Pacific Customer
Support Center
Email: ap.support@nsc.com

National Semiconductor
Japan Customer Support Center
Fax: 81-3-5639-7507
Email: jpn.feedback@nsc.com
Tel: 81-3-5639-7560

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Audio	www.ti.com/audio	Communications and Telecom	www.ti.com/communications
Amplifiers	amplifier.ti.com	Computers and Peripherals	www.ti.com/computers
Data Converters	dataconverter.ti.com	Consumer Electronics	www.ti.com/consumer-apps
DLP® Products	www.dlp.com	Energy and Lighting	www.ti.com/energy
DSP	dsp.ti.com	Industrial	www.ti.com/industrial
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Security	www.ti.com/security
Logic	logic.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Power Mgmt	power.ti.com	Transportation and Automotive	www.ti.com/automotive
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Mobile Processors	www.ti.com/omap		
Wireless Connectivity	www.ti.com/wirelessconnectivity		
	TI E2E Community Home Page		e2e.ti.com

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265