# APPLICATION NOTE

## PMAC
## 100/10BASE PCI MAC CONTROLLER

## 1. INTRODUCTION

This application note provides a guideline on implementing the interconnection of the network interface card (NIC) for MII, 100Base-TX and 10Base-T Fast Ethernet Network via the MX98713 PCI bus Media Access Controller (the PMAC).

In details, this document presents hardware design and layout recommendations which can help you to quickly implement all the network media interfaces mentioned. Meanwhile, we will discuss the methods on how to gain access on the Boot PROM, EEPROM and NWay registers through software interface.

As you can find in the MX98713 driver diskette, MXIC already provided a complete set of drivers for easier and more efficient way to interface with MX98713 on the most popular Network Operating Systems or environment. Nevertheless, there are still some special applications or environment not covered in the attached driver diskette. Driver developers, however, could still refer to the section of driver programming guide to accomplish the required driver.

## 2. PRODUCT OVERVIEW

The MX98713 PMAC implements the MAC layer and NWay functions on a single chip in accordance with the IEEE 802.3u standard. The PMAC highly integrates with direct PCI bus interface, including PCI bus master capability. Furthermore, the 10Base transceiver and low pass filter module are also supported. This is an advanced design and fully supports standard MII, 100Base-TX and 10Base-T media. For detailed product specification information, please refer to the MX98713 data sheet.

## 3. SYSTEM APPLICATION BLOCK DESCRIPTION

Figure 1 shows the completed functional block diagram of the MX98713. The provided external interconnection interfaces include Symbol (SYM), Twisted-Pair (TP), Media Independent Interface (MII), EEPROM interface, Boot ROM interface and PCI Bus interface. Based on these interfaces, we will describe the detailed implementation consideration when designing a PCI bus NIC via the PMAC.

### 3.1 SYM INTERFACE WITH NWAY EXTERNAL CONNECTION

The one of advantage features of the MX98713 is the NWay auto-negotiation capability. Through a pair of single RJ-45 connector and transformer, both 100Base and 10Base signals can share the same media. Figure 2 depicts the external interconnection block diagram, showing the implementation of the SYM interface with the NWay function, and Figure 3 is the corresponding recommended schematic circuits. The NWay allows NICs to automatically establish the highest available protocol, including operating speed and duplex mode, as well as media type for both transmission and reception.

It should be specially mentioned that, to lower total system cost, the PMAC has an embedded MCC/TP transceiver which can provide a 10Base-T direct media interface and need not any

external transceiver and low pass filter.

When you design the SYM interface with MX98713, a suggested component is the MX98704 that provides a direct-couple interconnection. During transmission, the PMAC converts the 125MHz NRZI encoded bit stream to 5-bit symbols and, during reception, it converts 5-bit symbols into NRZI decoded bit stream at 125MHz, accompanied with recovering the RX clock by using PLL circuit.

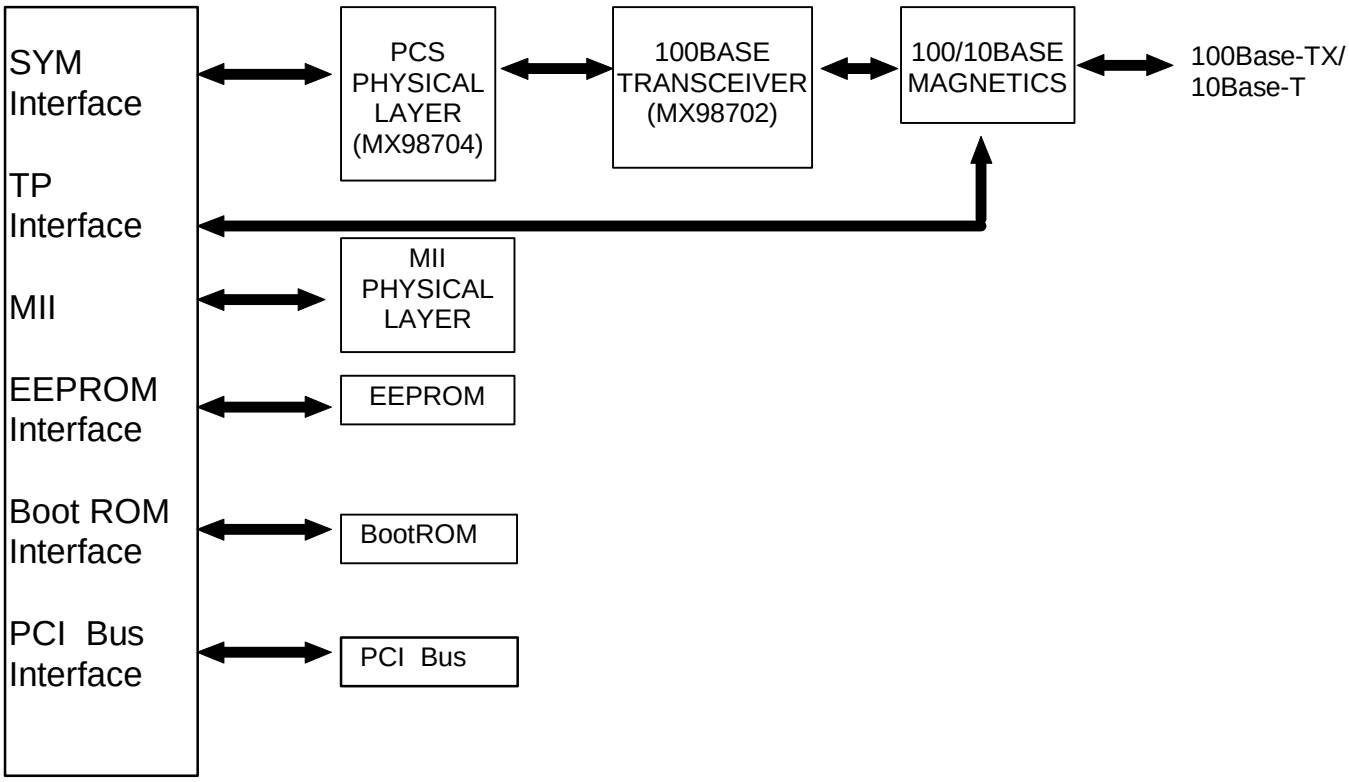
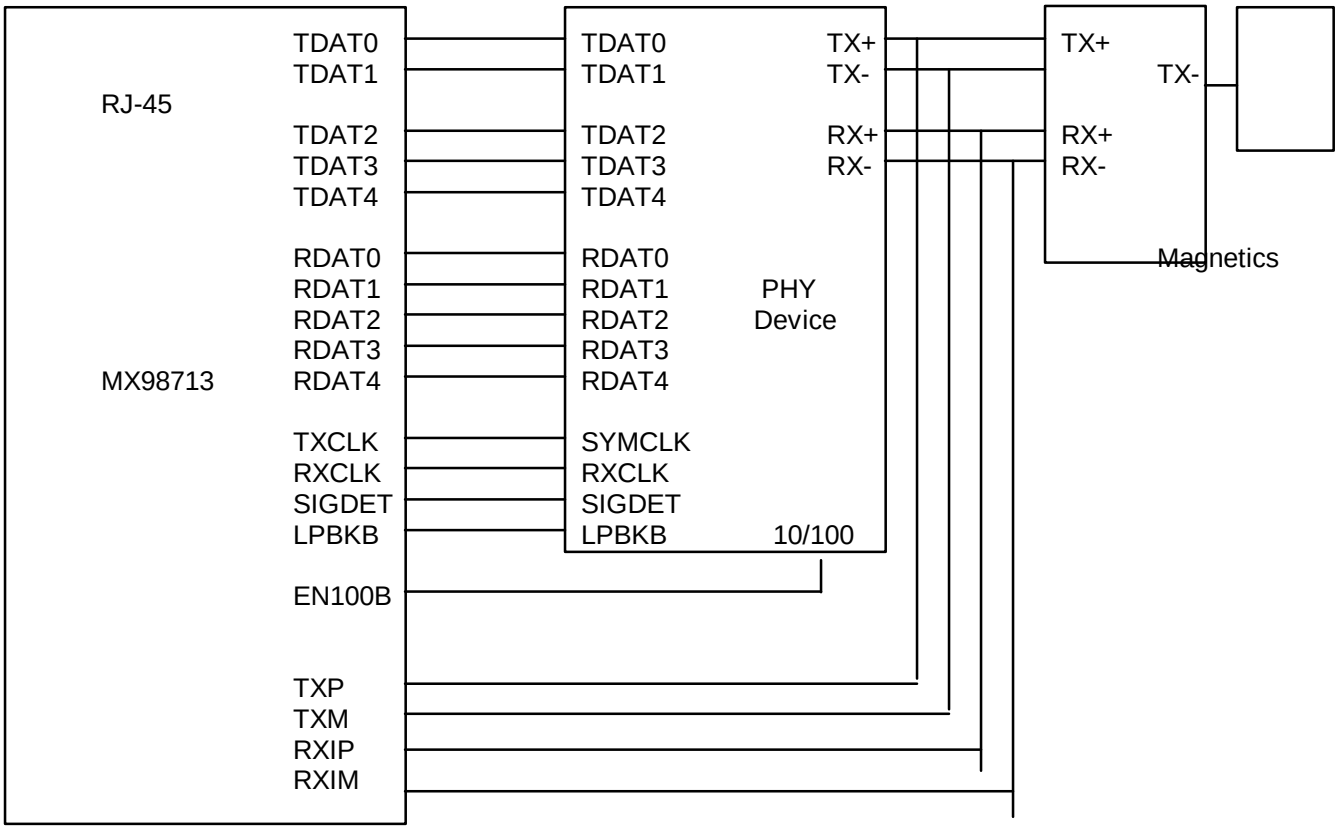



**Figure 1. The MX98713 functional block diagram.**

**Figure 2. SYM interface with NWay function interconnection block diagram**




### 3.1.1 LAYOUT RECOMMENDATION ABOUT MX98704

1. The length of data lines for both transmission (TXD0_4) and reception (RXD0_4) lines should be routed as short as possible and should have similar routing characteristics.

2. The transmitting and receiving clock should be carefully handled to avoid any noise. A popular method to isolate possible noise from other signal line is ground line.
3. The PLL components and path lines should be placed as near as possible to the MX98704.

### 3.1.2 LAYOUT RECOMMENDATION ABOUT MX98702

After the MX98702 is connected with MX98704, the NIC could convert the 125MHz NRZI bit stream to three-level (MLT-3) coding scheme. Since the signal speed between the MX98704 and the MX98702 is very high (125MHz) and sensitive, care should be taken that the signal lines be terminated at the receiving ends, as depicted in Figure 3.

The detailed layout consideration about the MX98702 includes:

1. The line length of the line pairs SD+/-, RD+/- and TD+/- between the MX98704 and the MX98702 should be kept as short as possible.
2. The space between each of the intra-line-pairs of SD+/-, RD+/- and TD+/- should be kept as near as possible and should have similar routing characteristics.
3. Keep all the inter-line-pairs among SD+/-, RD+/-, TD+/-, RX+/- and TX+/- away from each other to avoid signal crosstalk.

Both the MX98704 and the MX98702 provide the easy accessing loopback diagnostic function. Although it is not necessary under normal operating environment, this function provides a very convenient way to debug the whole NIC.

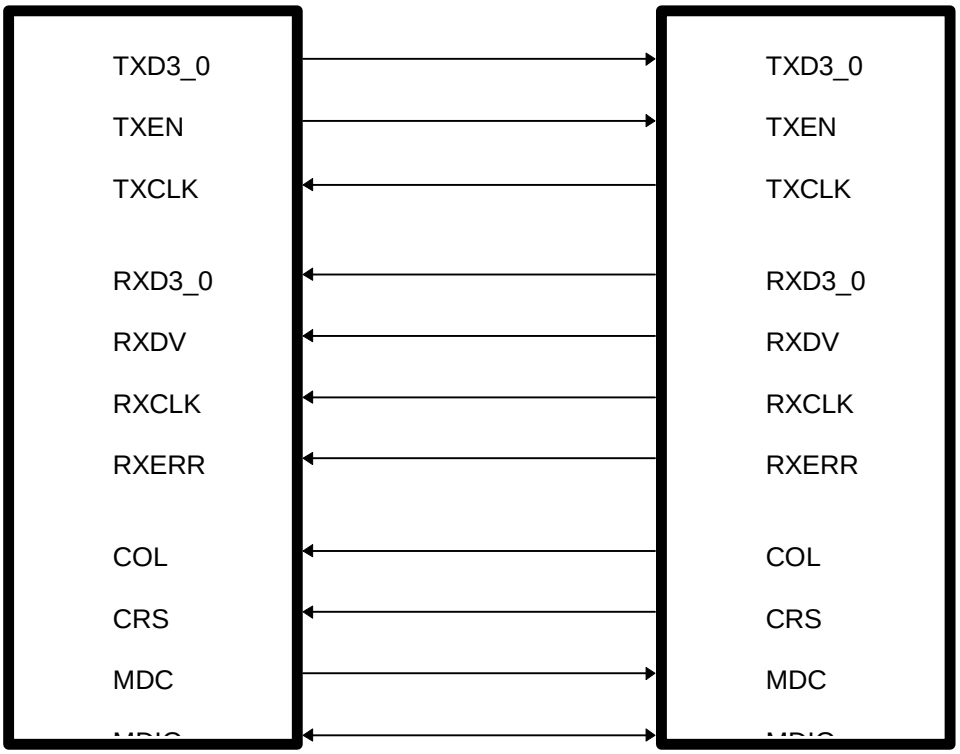
For isolating purpose, a magnetic component or transformer is placed between the MX98702 and the physical media (Twisted Pair). The purpose of the transformer connected to RJ-45 phone jack is to reduce EMI and RF noise.

**Figure 3. The SYM mode connection with NWay function schematic circuit**

## 3.2 MII AND 100BASE-T4 NETWORK IMPLEMENTATION

### 3.2.1 MII PORT INTERCONNECTION

In setting the CSR6_PS = 1, CSR6_PCS = 0 of the MX98713, the MII port is selected instead of the SYM interface. The MII port can be connected to any physical device with standard MII interface. Figure 4 shows the standard interconnection between the MX98713 and the most typical device through MII signal.




**MX98713 (PMAC)** **PHY Device**

**Figure 4. MII connection**

**Figure 5. 100Base T4 connection**

### 3.2.2 THE T4 NETWORK IMPLEMENTATION

The 100Base-T4 can be implemented through the MII port also. Figure 5 depicts the complete 100Base-T4 interface interconnection. The BCM5000 is a typical component that performs all the physical layer interface functions for both 100Base-T4 and 10Base-T. In T4 operation mode, the BCM5000 receives and transmits data on network in the form of 8B6T and then decodes them back to MII interface or vice versa. It can also be programmed to operate under 10Base-T through the MDC and MDIO signals. For more information about such application, please refer to the BCM5000 specification. After BCM5000, a T4 magnetic component ,Valor SF6036 (a typical component) is directly cascaded for isolating purposes.

Layout consideration about BCM5000:

1. The length of data lines TXD0_4 and RXD0_4 should be kept as short as possible and should have similar routing characteristics.
2. The space between each of the intra-line-pair of 8B6T signal TDn+/- and RDn+/- should be kept as near as possible and should have similar routing characteristics.
3. The space among the inter-line-pair of 8B6T signal TDn+/- and RDn+/- should be kept away from each others to avoid crosstalk.
4. The 20MHz crystal and relative components should be placed closer to the BCM5000.

### 3.3 EEPROM INTERFACE IMPLEMENTATION

Since the MX98713 supports a direct serial EEPROM interface, a straightforward interconnecting scheme is shown in Figure 6.

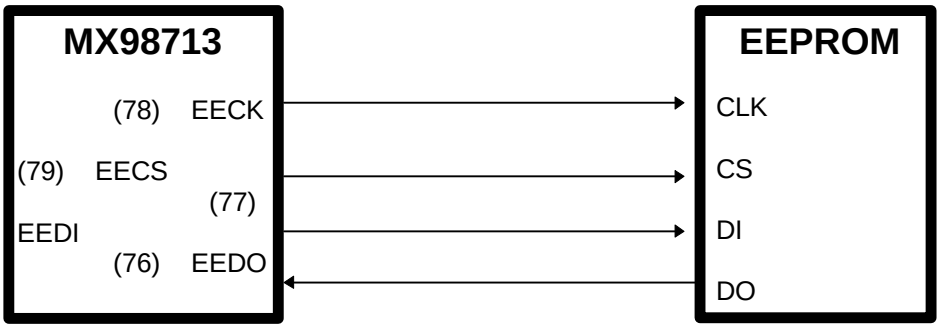



**Figure 6. Scheme of EEPROM interface connection**

In the following, a C-language module is listed completely to provide the EEPROM accessing routine through the CSR9 register of the MX98713.

```
/*******************************************************
*          Read all content from EEPROM (93C46)
*******************************************************/
eeprom_read()
```

```
{
        unsigned int i,address,eeval;
        char  bit;
        for (address=0;address<64;address++){
                NIC_write_reg(&csr9,(unsigned long)0x04800);
                eeprom_serial_in(0);
                eeprom_serial_in(1);       // command
                eeprom_serial_in(1);
                eeprom_serial_in(0);

                for(i=0;i<6;i++){              // address serial in
                        bit= ((address>>(5-i)) & 0x01) ? 1 :0 ;
                        eeprom_serial_in(bit);
                }
                eeval=0;
                for(i=0;i<16;i++){             // data serial out
                        NIC_write_reg(&csr9,(unsigned long)0x04803);
                        NIC_read_reg(&csr9);
                        eeval += (((unsigned long)0x008 & csr9.value)>>3)<<(15-i);
                        NIC_write_reg(&csr9,(unsigned long)0x04801);
                }
                NIC_write_reg(&csr9,(unsigned long)0x04800);

                c46[address*2]= eeval&0x0ff;
                c46[address*2+1]= (eeval>>8)&0x0ff;
        }
}

/*********************************************************
*       Write a word to EEPROM
*********************************************************/
eeprom_write(unsigned int address,unsigned int data)
{
        unsigned int i;
        char  bit;
        eeprom_wen();
        NIC_write_reg(&csr9,(unsigned long)0x04800);
        eeprom_serial_in(0);

        eeprom_serial_in(1);       // command
        eeprom_serial_in(0);
        eeprom_serial_in(1);

        for(i=0;i<6;i++){              // address serial in
                bit= ((address>>(5-i)) & 0x01) ? 1 :0 ;
                eeprom_serial_in(bit);
        }
        for(i=0;i<16;i++){          // data serial in
                bit= ((data>>(15-i)) & 0x01) ? 1 :0 ;
                eeprom_serial_in(bit);
        }
        NIC_write_reg(&csr9,(unsigned long)0x04800);
        NIC_write_reg(&csr9,(unsigned long)0x04801);
        i=0;
```

```
        do{
                i++;
                NIC_read_reg(&csr9);
        } while((!(csr9.value&0x08)) && (i<10000) );
        NIC_write_reg(&csr9,(unsigned long)0x04800);
        if(i==10000)  prstring (" Writing EEPROM error !!",  24, 10, Attr_N);
        eeprom_wds();
}

eeprom_wen()
{
        NIC_write_reg(&csr9,(unsigned long)0x04800);
        eeprom_serial_in(0);
        eeprom_serial_in(1);
        eeprom_serial_in(0);
        eeprom_serial_in(0);
        eeprom_serial_in(1);
        eeprom_serial_in(1);
        eeprom_serial_in(0); // X
        eeprom_serial_in(0); // X
        eeprom_serial_in(0); // X
        eeprom_serial_in(0); // X
        NIC_write_reg(&csr9,(unsigned long)0x04800);
}

eeprom_wds()
{
        NIC_write_reg(&csr9,(unsigned long)0x04800);
        eeprom_serial_in(0);
        eeprom_serial_in(1);
        eeprom_serial_in(0);
        eeprom_serial_in(0);
        eeprom_serial_in(0);
        eeprom_serial_in(0);
        eeprom_serial_in(0); // X
        eeprom_serial_in(0); // X
        eeprom_serial_in(0); // X
        eeprom_serial_in(0); // X
        NIC_write_reg(&csr9,(unsigned long)0x04800);
}

/*********************************************************
*       Serial inject a bit to EEPROM
*********************************************************/
eeprom_serial_in(unsigned int bit2)
{
        NIC_write_reg(&csr9,(unsigned long)0x04801+ 4*bit2);
        NIC_write_reg(&csr9,(unsigned long)0x04803+ 4*bit2);   // x
        NIC_write_reg(&csr9,(unsigned long)0x04801+ 4*bit2);
}
```

## 3.4 EEPROM FORMAT

The EEPROM format is described in detailed as below:

| Offset(Hex) | Descriptions |
|---|---|
| 20~6f | Reserved. |
| 70 | Network ID index to indicates the starting address of Network ID in length of continuous 6 bytes. The content of this field could be in range of 00h~1ah. |
| 71 | Reserved. |
| 72 | Operating mode selection; 0: PCS mode, else: MII mode's physical address |
| 73 | IC revision ID; 0:revision E and before, 1:revision G and after |
| 74~79 | Reserved, all bits should be set to 0 |
| 7a | LSB of Device ID |
| 7b | MSB of Device ID |
| 7c | LSB of Vendor ID |
| 7d | MSB of Vendor ID |
| 7e~7f | Check sum |

### 3.4.1 THE CONTENTS OF EEPROM

The dumped EEPROM contents is listed as below:

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 aa bb cc 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
14 00 00 01 00 00 00 00 00 00 12 05 d9 10 0e de

For the example mentioned above, the parameters of a typical adapter card will be:

Vendor ID : 0x10d9
Device ID : 0x0512
Network ID : Offset = 0x14 (then the ID address is: 00 00 00 aa bb cc)
checksum : 0x0ede
operating mode : 0 (Symbol mode)
IC revision ID : 1 (revision G and after)

### 3.4.2 THE OPTIONS FOR VENDOR ID AND DEVICE ID:

Since you could choose the driver provided by MXIC or developed by yourself. Meanwhile, although the MX98713 has built-in the intelligent Nway function, you could bypass it and adapt external Nway through MII interface. The possible options are summarized in the following:

| combination | vender ID | device ID | S/M value |
|---|---|---|---|
| MXIC driver + MX98713 with Nway | 10d9 | 0512 | 0 |
| MXIC driver + MX98713 + external Nway | 10d9 | 0512 | MII address |
| vendor driver + MX98713 with Nway | vendor's own ID | vendor's own ID | 0 |

| | | | |
|---|---|---|---|
| vendor driver + MX98713 + external Nway | vendor's own ID | vendor's own ID | MII address |

### 3.5 PCI BUS CONNECTION

Figure 7 shows the interconnection between the MX98713 and host system through PCI bus. The MX98713 has a complete set of host system connections to achieve direct interfacing with the PCI bus control and address/data signals. The length of each connection should be limited to under 1.5 inches.

### 3.6 BOOT PROM CONNECTION

Figure 8 shows the interconnection between PMAC and boot PROM. The boot PROM size may be 8K, 16K, 32K or 64K, and default size is set to be 16K bytes. Boot PROM base address is defined in PCI bus' configuration memory at offset 33h-30h. For detailed information, please refer to the MX98713 PMAC product specification.

**Figure 7. PCI Interface**

**Figure 8. MX98713 Power, Boot PROM, PCI Bus, LED connection**

# 4. ABOUT THE DRIVER

## 4.1 DRIVER OVERVIEW

As you can find in the released driver diskette, there is the “release.txt” file in root directory describing the released driver contents, revision and date. Meanwhile, we have prepared the “readme.txt” file to demonstrate the installation procedure and consideration in root directory as well as each subdirectory that contains various family of drivers. In general, you may find the required information from these text files.

## 4.2 DRIVER PROGRAMMING GUIDE

In this section, three most important C-language source code modules are completely listed. As you can find in the attached driver diskette, MXIC already presented a full set of high performance drivers for the most popular network operating systems. If you, however, still want to develop driver for special applications or environment, the following Initialization, Transmission and Reception modules should become your kernel code.

### 4.2.0 UNDOCUMENTED REGISTERS FOR DEVELOPING YOUR OWN DRIVER

**Before you follow the programming guide to write your own driver code, there are registers located beyond offset 78h of MX98713’s IO space that are not documented in data sheet but reserved for new features and test purposes. At offset 80h in IO space is CSR16 which must be set to “0x0F37xxxx” to bring the chip into normal operation mode before any initialization process can be started by driver. These undocumented registers are reserved for test and new feature like Magic Packet Detection in the future. Here is a list of all registers in this reserved IO area :**

| | | |
|---|---|---|
| **offset 80h** | **CSR16** | **Magic Packet/Test mode register ( default “0x0F37xxxx” )** |
| **offset 88h** | **CSR17** | **Reserved** |
| **offset 90h** | **CSR18** | **Reserved** |
| **offset 98h** | **CSR19** | **Reserved** |
| **offset A0h** | **CSR20** | **Reserved** |
| **offset A8h** | **CSR21** | **Reserved** |
| **offset B0h** | **CSR22** | **Reserved** |

### 4.2.1 INITIALIZATION

```
InitializeTheTransmitRing()
{
        unsigned int        i, j;
        unsigned long       physicaladdress;

        for( i=0 ; i<NumTXBuffers ; i++ ){
                /* memory allocation for tx descriptor_buffer  (allign 4)  */
                tx_resource[i]=(struct TX_RESOURCE  *)((((unsigned int )tx_temp[i])+4)& 0xfffc);
        }

        for( i=0 ; i<NumTXBuffers ; i++ ){
                /* initialize the own bit to host tdes0 */
                tx_resource[i]->ownership=0x00;
                tx_resource[i]->tstatus=0x0000;
```

```
                tx_resource[i]->tdes0_unused=0x00;

                /* fill buffer_1_address tdes2*/
                get_ea((void far *)(tx_resource[i]->tx_buffer_data),&physicaladdress);
                tx_resource[i]->buff_1_addr=physicaladdress;

                /* fill buffer_2_address tdes3*/
                if(i==NumTXBuffers-1) j=0;
                else  j=i+1;
                get_ea((void far *)tx_resource[j],&physicaladdress);
                tx_resource[i]->buff_2_addr=physicaladdress;
        }
}

InitializeTheReceiveRing()
{
        unsigned int          i, j;
        unsigned long         physicaladdress;
        for( i=0 ; i<NumRXBuffers ; i++ ){
                /* memory allocation for tx descriptor_buffer  (allign 4)  */
                rx_resource[i]=(struct RX_RESOURCE  *)((((unsigned int )rx_temp[i])+4) & 0xfffc);
        }
        for( i=0 ; i<NumRXBuffers ; i++ ){
                /* set own bit to chip     rdes0  */
                rx_resource[i]->frame_length=RDES0_OWN_BIT;
                rx_resource[i]->rstatus=0x0000;

                /* fill rdes1  */
                rx_resource[i]->command=RDES1_BUFF-RX_BUFFER_SIZE+rxpkt_size[i];

                /* fill buffer_1_address   rdes2  */
                get_ea((void far *)(rx_resource[i]->rx_buffer_data),&physicaladdress);
                rx_resource[i]->buff_1_addr=physicaladdress;

                /* fill buffer_2_address   rdes3  */
                if(i==NumRXBuffers-1) j=0;
                else          j=i+1;
                get_ea((void far *)rx_resource[j],&physicaladdress);
                rx_resource[i]->buff_2_addr=physicaladdress;
        }
}

initialize()
{
        unsigned long  physicaladdress;
        unsigned int     i;

        InitializeTheTransmitRing ();
        InitializeTheReceiveRing ();
        NIC_write_reg(&csr0,CSR0_L_SWR); // Clear  Status Register
        for(i=0;i<6;i++)      perfect[i]=sa[i];
        setup_frame(TDES1_SETUP_LAST,perfect);
        delay(50);
        NIC_write_reg(&csr0,csr0shadow);
```

```
        NIC_write_reg(&csr16,(unsigned long)0x0f37fec8);     //set IC to normal mode
        get_ea((void far *)rx_resource[0],&physicaladdress);
        NIC_write_reg(&csr3,physicaladdress);
        get_ea((void far *)tx_resource[0],&physicaladdress);
        NIC_write_reg(&csr4,physicaladdress);
        NIC_write_reg(&csr6,csr6shadow);
        NIC_write_reg(&csr5,(unsigned long)0xffffffff);     //Clear status register
}
```

### 4.2.2 TRANSMISSION MODULE

```
bmtx()
{
        unsigned char editmode, j;
        struct TX_RESOURCE  *tx_pointer;

        initialize();
        fill_pattern(6);                                    // fill pattern
        NIC_write_reg(&csr6,csr6.value&(~CSR6_ST)); // stop
        NIC_read_reg(&csr6);
        NIC_write_reg(&csr6,csr6.value|CSR6_SF);        // store and forward
        NIC_read_reg(&csr0);
        NIC_write_reg(&csr0,csr0.value|0x020000);       // TAP=01

        tx_pointer= tx_resource[0];
        j=0;
        editmode=1;

        while (editmode)   {
                if( (tx_pointer->ownership & 0x80 )==0 )  {
                        j++;
                        j %= tx_pkt_num;
                        if (tx_pointer->command & TDES1_LS_BIT )
                        tx_error_detect(tx_pointer->tstatus);
                        tx_pointer->ownership |= 0x80 ;
                        tx_pointer= tx_resource[j];
                }

                if (kbhit()) {
                        keycode_get();
                        if (M_code!=0)  {
                                switch (M_code)  {
                                        case 0x1b :                         /*** ESC   ***/
                                                editmode=0;
                                                cll(0,0,79,24);
                                                break;
                                        case 0x20 :
                                                NIC_read_reg(&csr6);    // CSR6_ST
                                        NIC_write_reg(&csr6,csr6.value^CSR6_ST);
                                                break;
                                        default   :
                                                break;
```

```
                                        }
                                }
                        }
                }
}
```

### 4.2.3 RECEPTION MODULE

```
bmrx()
{
        unsigned char editmode,i,j;
        unsigned long physicaladdress;
        struct RX_RESOURCE   *rcv_pointer;
        initialize();
        rcv_pointer= rx_resource[0];
        j=0;
        editmode=1;
        while (editmode)   {

                /* if data received */
                if(( rcv_pointer->frame_length & 0x8000 )==0)  {
                        j++;
                        j %= 6;
                        if(rcv_pointer->rstatus & RDES0_LS )
                        rx_error_detect(rcv_pointer->rstatus);
                        rcv_pointer->frame_length|=0x8000;
                        rcv_pointer= rx_resource[j];
                }
                if (kbhit())  {
                        keycode_get();
                        if (M_code!=0)  {
                                switch (M_code) {
                                        case 0x1b :                     /*** ESC   ***/
                                                editmode=0;
                                                cll(0,0,79,24);
                                                break;
                                        default   : break;
                                }
                        }
                }
        }
}
```

## 5. ABOUT THE LOOPBACK AND DIAGNOSTICS

The loopback mode allows in-circuit testing or debugging of the NIC using the MX98713.

### 5.1 INTERNAL LOOPBACK

There are two kinds of internal loopback modes provided and mentioned below:

**100Mbps loopback**: Set the NWay command register as 0.13=1, 0.12= 0, 0.14=0 and set CSR6_LOM=01 to enter internal loopback mode . This loopback mode is performed on 5B signal layer.

**10Mbps loopback** : Set the NWay command register as 0.13=0 , 0.12=0, 0.14=1 and set CSR6_LOM=10 to enter external loopback mode. The loopback mode is performed on embedded MCC layer.

Both the two loopback operation modes provide a self-diagnosis capability on the MX98713 itself. For detailed information about the loopback operation, please refer to the portions of CSR6 and NWay register of the MX98713 PMAC product specification.

## 5.2 100MBPS EXTERNAL LOOPBACK

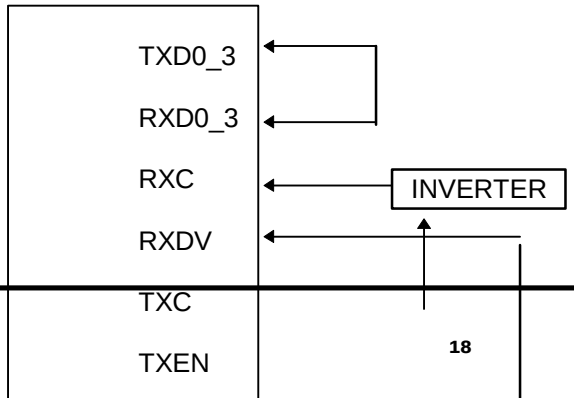
There are two kinds of 100Mbps external loopback modes provided and mentioned below:

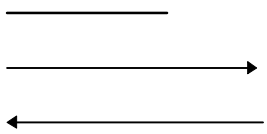
**External loopback through FCG:** Set the MX98713 CSR6_LOM=10 and set the NWay command register as 0.13=1, 0.12= 0, 0.14=0 as well as set the MX98704 /LB to low to enter external loopback mode. The MX98704 will then disconnect all the signals behind it and loop the transmit data path back into the receive path. This operation provides a diagnosis on the circuit between the MX98704 and the MX98713.

**External loopback through Transceiver:** The MX98713 could be forced to be in the external loopback mode by setting both the LBEN of the MX98704 and the MX98702 to high. Under such setting mode, the signal pair of TXO+/- will loop back to signal pair of RXI+/-. This operation mode provides a diagnosis on the circuit between the MX98713 and the MX98702.

## 5.3 MII EXTERNAL LOOPBACK CONNECTION

The MII interface external loopback operation depends on the selection of transceiver and it is for the MX98713 MII interface verification purpose only. The loopback interconnection is provided as shown in Figure 9. Such connection mode provides a direct way to verify the MII interface function.





**MX98713 (PMAC)**

**Figure 9. MII external loopback connection**

## 6. ABOUT THE NWAY REGISTER ACCESS

The MII management is under control by MDC and MDIO pins of the MX98713 and will generate the read or write command to the internal register of the MX98713. All the commands should have the frame structure:

<PRE><ST><OP><PHYAD><REGAD><TA><DATA><IDLE>

The complete C-language routines are provided to gain access to the NWay internal registers as mentioned below. For detailed information on how to manage the NWay function, please refer to the MX98713 PMAC product specification.

```
/************************************************
*  read MII register
*  phyad -> physical address
*  regad -> register address
************************************************ */
unsigned int mii_read(phyad,regad)
unsigned char phyad,regad;
{
        unsigned int i,value;
        unsigned int bit;
        mii_pre_st();                   // PRE+ST
        mii_serial_in(1);               // OP
        mii_serial_in(0);

        for (i=0;i<5;i++)   {           // PHYAD
                bit= ((phyad>>(4-i)) & 0x01) ? 1 :0 ;
                mii_serial_in(bit);
        }

        for (i=0;i<5;i++)   {           // REGAD
                bit= ((regad>>(4-i)) & 0x01) ? 1 :0 ;
                mii_serial_in(bit);
        }
        mii_serial_out();                       // TA_Z
        if((bit=mii_serial_out()) !=0 )         // TA_0
```

```
        return(FAIL);

        value=0;
        for (i=0;i<16;i++)  {                   // READ DATA
                bit=mii_serial_out();
                value += bit<<15-i ;
        }
        mii_serial_in(0);                       // dumy clock
        mii_serial_in(0);                       // dumy clock
        return(value);
}

/*****************************************
*   Write MII register
*   phyad -> physical address
*   regad -> register address
*   value -> value to be write
***************************************** */
mii_write(phyad,regad,value)
unsigned char phyad,regad;
unsigned int  value;


{
        unsigned int i;
        char  bit;
        mii_pre_st();                           // PRE+ST
        mii_serial_in(0);                       // OP
        mii_serial_in(1);

        for (i=0;i<5;i++)   {                   // PHYAD
                bit= ((phyad>>(4-i)) & 0x01) ? 1 :0 ;
                mii_serial_in(bit);
        }

        for (i=0;i<5;i++)   {                   // REGAD
                bit= ((regad>>(4-i)) & 0x01) ? 1 :0 ;
                mii_serial_in(bit);
        }

        mii_serial_in(1);                       // TA_1
        mii_serial_in(0);                       // TA_0

        for (i=0;i<16;i++)  {                   // OUT DATA
                bit= ((value>>(15-i)) & 0x01) ? 1 : 0 ;
                mii_serial_in(bit);
        }
        mii_serial_in(0);                       // dumy clock
        mii_serial_in(0);                       // dumy clock
}

/*****************************************
*   preamble + ST
*****************************************/
```

```
mii_pre_st()
{
        unsigned char i;
        NIC_write_reg(&csr9,(unsigned long)0x02000); // SWO+~SSR
        for(i=0;i<32;i++)                               // PREAMBLE
                mii_serial_in(1);
        mii_serial_in(0);                       // ST
        mii_serial_in(1);
}

/* ****************************************************************
*   Inject a bit to NWay register through CSR9_MDC,MDIO
****************************************************************/
mii_serial_in(char bit_MDO)       // inject data into mii PHY
{
        if(bit_MDO){
                NIC_write_reg(&csr9,(unsigned long)0xfff02000 | CSR9_MDO);//           SWO
                NIC_write_reg(&csr9,(unsigned long)0xfff12000 | CSR9_MDO);// MDC+SWO
                NIC_write_reg(&csr9,(unsigned long)0xfff02000 | CSR9_MDO);//           SWO
        }
        else{
                NIC_write_reg(&csr9,(unsigned long)0xfff02000 );//     SWO
                NIC_write_reg(&csr9,(unsigned long)0xfff12000 );// MDC+SWO
                NIC_write_reg(&csr9,(unsigned long)0xfff02000 );//     SWO
        }
}



/************************************************************************
*    read  a bit from NWay register through CSR9_MDC,MDIO
************************************************************************/
mii_serial_out()                        // read data from mii PHY
{
        unsigned int bit_MDI;
        NIC_write_reg(&csr9,CSR9_MMD|CSR9_SRO );                // MMD+SRO
        NIC_read_reg (&csr9);
        NIC_write_reg(&csr9,CSR9_MMD|CSR9_SRO|CSR9_MDC); // MD+MDC+SRO
        NIC_write_reg(&csr9,CSR9_MMD|CSR9_SRO );                // MMD+SRO
        bit_MDI =  (csr9.value & CSR9_MDI) ? 1 : 0 ;
        return(bit_MDI);
}
```

## 7. RECOMMENDATION OF POWER SUPPLY CONNECTION

The recommended power supply schematic is shown in Figure 8. The GND indicates a direct connection to the ground plane. The VCC indicates a direct connection to the power plane.

## MACRONIX INTERNATIONAL CO., LTD.

**HEADQUARTERS:**
No. 3, Creation Road III, Science-Based Industrial Park, Hsin Chu, Taiwan, R.O.C.
TEL:+886-3-578-8888
FAX:+886-3-578-8887

**TAIPEI OFFICE:**
12F, No. 4, Min-Chuan E.Rd., Sec 3, Taipei, Taiwan, R.O.C.
TEL:+886-3-509-3300
FAX:+886-3-509-2200

**EUROPE OFFICE:**
Koningin Astridlaan 59, Bus 1, 1780 Wemmel, Belgium
TEL:+32-2-456-8020
FAX:+32-2-456-8021

**SINGAPORE OFFICE:**
5 Jalan Masjid Kembangan Court #01-12 Singapore 418924
TEL:+65-747-2309
FAX:+65-748-4090

**MACRONIX AMERICA, INC.**
1338 Ridder Park Drive, San Jose, CA95131 U.S.A.
TEL:+1-408-453-8088
FAX:+1-408-453-8488

**JAPAN OFFICE:**
NFK Kawasaki Building, 8F, 1-2 Higashida-cho, Kawasaki-ku
Kawasaki-shi, Kawasaki-ken 210, Japan
TEL:+81-44-246-9100
FAX:+81-44-246-9105