APPLICATION NOTE

1. INTRODUCTION

The purpose of this application note is to describe the implementation of a PCI bus master 100 Base-TX Fast Ethernet node using MXIC' highly integrated single chip Fast Ethernet NIC controller MX98725. In details, this document presents product overview, programming guide, hardware design and layout recommendations that can help you to quickly and smoothly implement a Fast Ethernet adapter card.

As you can find in the MX98725 driver diskette, MXIC already provided a complete set of high quality drivers for easier and more efficient way to interface with MX98725 on the most popular Network Operating Systems. Nevertheless, there are still some special applications or environment not covered in the MX98725 driver diskette. Driver developers, however, could still refer to the section of driver programming guide to accomplish the required driver. It is recommended that you are familiar with the MX98725 data sheet before reading this guide.

2. PRODUCT OVERVIEW

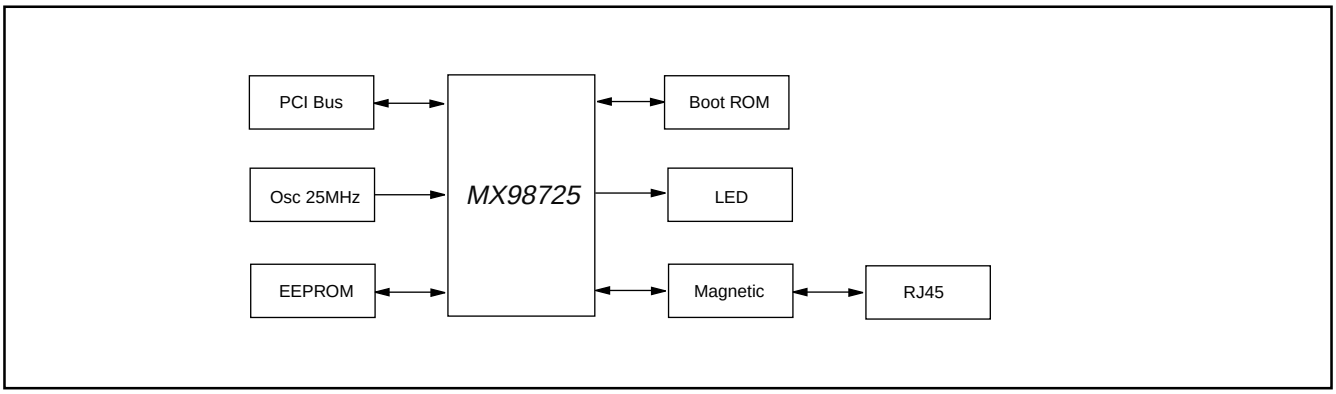
The MX98725 implements the 10/100Mbps MAC layer and Physical layer on a single chip in accordance with the IEEE 802.3 standard.

The MX98725 highly integrates with direct PCI bus interface, including PCI bus master with DMA channel capability, direct EEPROM as well as Boot ROM interface, and large on chip transmit/receive FIFOs. Also, the MX98725 is equipped with intelligent IEEE802.3u-compliant Nway auto-negotiation capability allowing a single RJ-45 connector to link with the other IEEE802.3u-compliant device without re-configuration. To optimize operating bandwidth, network data integrity and throughput, the proprietary Adaptive Network Throughput Control (ANTC) function is implemented. For detailed product specification information, please refer to the MX98725 data sheet.

3. HARDWARE DESIGN CONSIDERATIONS

3.1 SYSTEM APPLICATION BLOCK DIAGRAM

A system block diagram for the MX98725 based Fast Ethernet adapter card is shown as following:



3.2 PCI CONNECTION

The MX98725 provides direct PCI bus interface to PCI connector. Board designers should especially take care of the four pins of TDI, TDO, PRSNT1# & PRSNT2# that are only related to PCI bus connector. Boards that do not implement JTAG Boundary Scan should tight TDI and TDO together to prevent the scan chain from being broken.

Both pins PRSNT1# and PRSNT2# should be connected to ground indicating that the board is physically presenting in a PCI slot and providing information about the total power requirements (less than 7.5W) of the board.

3.3 OSCILLATOR

The MX98725 is designed to operate with a 25MHz-oscillator module. The clock specification of this oscillator should meet 25MHz +/- 50PPM.

Furthermore, if flash memory is used, one can change its contents via CSR9 and CSR10 (Refer to MX98725 data sheet).
Detailed software programming example is described in section 4.6.

3.4 BOOT ROM

The MX98725 supports a direct boot ROM interfaces allowing diskless workstations to remotely download operating system from network server. For proper operation, the access time of adapted EPROM should not excess 240ns.

Furthermore, if flash memory is used, user could easily change its contents via CSR9 and CSR10. Detailed information of CSR9 and CSR10 please refer to MX98725 data sheet. In Section 4.6, an example of flash memory programming methods is listed.

3.5 SERIAL EEPROM

The MX98725 provides pins EECS, BPA0 (EECK), BPA1 (EEDI) and BPD0 (EEDO) for directly accessing the serial EEPROM. BPA0-1 and BPD0 serve as SK (EECK), DI (EEDI) and DO (EEDO) respectively. The contents of the EEPROM include the ID information of the MX98725 (VendorID, DeviceID, Sub-vendorID, Sub-deviceID and MAC ID), and the configuration parameters for software driver. The EEPROM contents should be programmed according to MXIC' definition as mentioned in Appendix A. Detailed software programming example is described in section 4.5.

3.6 PROGRAMMABLE LED SUPPORT

The MX98725 provides four pins LED0SEL, LED1SEL, LED2SEL and LED3SEL to control display LED. Displayed messages are programmable through setting CSR9 bit 28~31 to serve as Activity or Linkspeed, Goodlink or Link/Activity, TX or Collision and RX or Full/Half Duplex LED respectively. The maximum sinking current of these output pins is 16mA. Current limiting resistor (560Ω) should be added to ensure proper operation. The following indicates the configuration setting example table for LED display programming.

CSR9 <28:31>	LED0	LED1	LED2	LED3
0000	Activity	Goodlink	TX	RX
1111	Linkspeed	Link/Act	Collision	F/H duplex

3.7 NETWORK INTERFACE TO MAGNETIC COMPONENT

For isolating and impedance matching purpose, an isolating transformer with 1:1 transmit and 1:1 receive turns ratio is required for transmit and receive twisted pair interface. In Appendix B, several transformers that we had verified successfully with MX98725 are listed for quick reference purpose.

3.8 OPTIMIZED EQUALIZER COMPONENTS

MXIC' Fast Ethernet solution utilizes adaptive equalizer to compensate the attenuation and phase distortion induced by different lengths of cable. To optimize transmit and receive signal quality, pins RTX and RTX2EQ should be connected to external resistors 560Ω (±1%) and 1.4KΩ (±1%) and then to ground respectively.

3.9 Remote-Power-On and ACPI application

MX98725 fully supports Remote-Power-On and ACPI that meets PC98 requirement for power-sensitive applications. To implement such advance features, two necessary conditions should be met at first:

(1) After the AC power being plugged in, power supply must be continuously drained at least 400mA from auxiliary power, no matter the switch is in ON or OFF states.

(2) Motherboard BIOS must support Remote-Power-On.

The auxiliary power and PCI bus power are decoupled via a power switch (FDC6324L). After the AC power core being plugged in, the MX98725 will automatically load network ID from EEPROM and begin to scan incoming packet. Once receiving magic packet from network, PMEB and EXTSTARTB of MX98725 will be asserted low and LANWAKE will be asserted high to wake up the host.

PMEB will be asserted when one of the following conditions meet:
(1) The MX98725 is in D1 state, PMCSR<8> and PMEB enable, and a magic packet is received.
(2) The MX98725 is in D3-cold (power off) state, and a magic packet is received.

PMEB can be disserted by:
(1) Clearing PMCSR<8>
(2) Writing 1 to PMCSR<15>
(3) Turn on the host power

4. DRIVER PROGRAMMING GUIDE

This chapter will provide you the necessary information for programming driver for the MX98725 based node. Initialization module is introduced first that describes how MX98725 is initialized before any other operations can commence, then followed by actual implementation examples for both transmit and receive operations. Programming differences between MX98713, MX98713A and MX98725 are also included that will help you to upgrade your own driver to support all MXIC' NIC product series.

4.1 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 rx descriptor_buffer (allign 4) */
         rx_resource[i]=
            (struct RX_RESOURCE *)((((unsigned int)rx_temp[i])+4)&
0xfffc);
        }

        for (i=0; i<NumRXBuffers; i++) {
         /* set the 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;

        NIC_read_reg(&csr6);
        NIC_write_reg(&csr6,csr6.value&(~(CSR6_SR|CSR6_ST)));
        delay(10);
        InitializeTheTransmitRing (6);
        InitializeTheReceiveRing (6);
        NIC_write_reg(&csr0,CSR0_L_SWR);
```

```
        delay(50);

        NIC_write_reg(&csr0,csr0shadow);
        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_read_reg(&csr16);
        NIC_write_reg(&csr7,csr7shadow);
        NIC_write_reg(&csr16,csr16shadow);
        //Clear status register
        NIC_write_reg(&csr5,(unsigned long)0xffffffff);
        NIC_write_reg(&csr6,csr6shadow);
        NIC_read_reg(&csr6);
        setup_frame(TDES1_SETUP_LAST,perfect);
  }
```

4.2 TRANSMITION 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
forware
        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: quit
                             editmode=0;
                             break;
                       case 0x20:
                             NIC_read_reg(&csr6);
                             NIC_write_reg(&csr6,csr6.value^CSR6_ST);
                             break
                    default: break;
                    }
              }
            }
        }
  }
```

4.3 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: quit
                             editmode=0;
                             break;
                    default: break;
                    }
              }
            }
        }
  }
```

4.4 CODING DIFFERENCE BETWEEN MX98713, MX98713A, MX98715 AND MX98725

4.4.1 SPEED SELECTION

Speed selection of MX98713 are controlled by internal Nway registers. All the MII management commands should have the following structure:

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

For detailed programming example, please refer to MX98713 application note.

As for MX98713A, MX98715 and MX98725, Internal NWay registers are removed and protocol selection is controlled by Operation Mode Register (CSR6) and 10Base-T Control Register (CSR14)

NWay Active	100F	100H	10F	10H	
CSR6_PS	0	1	1	0	0
CSR6_PCS	X	1	1	X	X
CSR6_FD	1	1	0	1	0
CSR14_ANE	1	0	0	0	0

4.4.2 ELSE REGISTERS SETTING FOR DEVELOPING YOUR OWN DRIVER

There is an obvious change in setting the contents of CSR16 in comparison with MX98713. In MX98713, the offset 80h in IO space that is CSR16 must be set to "0x0f37XXXX" to bring this controller into normal operation mode before any initialization process can be started by driver. However, in MX98713A, MX98715 and MX98725, driver developers must set the CSR16 to be "0x0b3cXXXX" .

In summary, the contents of CSR16 for MXIC 100Base NIC controllers should be set differently as follow:

MX98713 = 0x0f37XXXX
MX98713A = 0x0b3cXXXX
MX98715 = 0x0b3cXXXX
MX98725 = 0x0b3cXXXX

Meanwhile, you could directly access the Nway auto-negotiation status from CSR20. Detailed format please refer to MX98725 data sheet.

4.5 EEPROM ACCESSING

The following is a reference code for accessing the contents of EEPROM that stores ID information and node configuration for the MX98725.

```
/*************************************
*  Read all content from EEPROM
*************************************/
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++){       //dat 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 !!");
    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);
    eeprom_serial_in(0);
    eeprom_serial_in(0);
    eeprom_serial_in(0);
    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);
    eeprom_serial_in(0);
    eeprom_serial_in(0);
    eeprom_serial_in(0);
    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)0x04800+4*bit2);
    NIC_write_reg(&csr9,(unsigned long)0x04803+4*bit2);
    NIC_write_reg(&csr9,(unsigned long)0x04801+4*bit2);
}
```

4.6 Flash ROM accessing

Following is an example code for accessing Macronix's flash ROM MX28F2000.

```
 #define  CYCLE1_ADDRESS    (unsigned long)0x00005555
#define  CYCLE2_ADDRESS    (unsigned long)0x00002aaa

 FlashWrite (unsigned long FlashAddress, unsigned char
FlashData)
{
    NIC_write_reg(&csr10, FlashAddress);
    NIC_write_reg(&csr9, (unsigned long)(0x00003000 |
FlashData));
}

unsigned char FlashRead (unsigned long FlashAddress)
{
    NIC_write_reg(&csr10, FlashAddress);
    NIC_write_reg(&csr9, (unsigned long)(0x00005000));
    NIC_read_reg(&csr9);
    return ((unsigned char)(0xff & csr9.value));
}

ResetToReadMode ()
{
    _disable();
    FlashWrite (CYCLE1_ADDRESS, (unsigned char) 0xff);
    FlashWrite (CYCLE1_ADDRESS, (unsigned char) 0xff);
    _enable();
}

unsigned char Read_ID_OK ()
{
    unsigned char manufactureID;
    unsigned char deviceID;

    ResetToReadMode ();

    _disable();
    FlashWrite (CYCLE2_ADDRESS, (unsigned char) 0x90);
    manufactureID = FlashRead (CYCLE2_ADDRESS);
    FlashWrite (CYCLE2_ADDRESS, (unsigned char) 0x90);
    deviceID = FlashRead (CYCLE2_ADDRESS+1);
    _enable();

    ResetToReadMode ();
    if ((manufactureID!=0xc2) | (deviceID!=0x2a)) return 0;
    else return 0xff;
}


 SetUpByteWriteCmd (unsigned long wraddr, unsigned char
wrdata)
{
    unsigned char tmp;
```

```
    _disable();
    FlashWrite (wraddr, (unsigned char) 0x40);
    FlashWrite (wraddr, wrdata);
    _enable();
    tmp = FlashRead (wraddr);
    while (tmp != FlashRead(wraddr)) tmp=FlashRead(wraddr);
    ResetToReadMode ();
}

ChipErase ()
{
    unsigned char tmp;

    _disable();
    FlashWrite (CYCLE1_ADDRESS, (unsigned char) 0x30);
    FlashWrite (CYCLE1_ADDRESS, (unsigned char) 0x30);
    _enable();

    delay100us ();
    tmp = FlashRead (CYCLE1_ADDRESS);
    while (tmp != FlashRead(CYCLE1_ADDRESS))
          tmp=FlashRead(CYCLE1_ADDRESS);
    ResetToReadMode ();
}

/*************************************************************/
main(argc,argv)
/*************************************************************/
int argc;
char *argv[];
{
    unsigned long i;

    // MX28F2000 manufactureID and DeviceID check
    if (!Read_ID_OK())
    {
      printf("\nRead ID failed !!!\n");
      exit(0);
    }

    //  MX28F2000 erased
    ChipErase ();

    //  MX28F2000 filled with all ' 0 '
    for (i=0; i<=0x3ffff; i++)
    {
      SetUpByteWriteCmd (i, 0);
      ResetToReadMode ();
    }

}
```

5. PCB LAYOUT RECOMMENDATIONS

The MX98725-based adapter board is strongly recommended to separate the power plane into 3 regions, i.e., digital, analog and receive region to isolate from digital noise and noise coupling between the transmitter and the receiver. These power pins in these three regions are shown in the table listed below. Each VDD pin also needs a 0.1uF capacitor located as close to the VDD pin as possible.

Analog Region	Receive Region	Digital Region
3. VDD	147. VDD	Others
4. GND	148. GND	
5. VDD	145. VDD	
6. GND	146. GND	
7. VDD	151. GND	
8. GND	152. VDD	
9. VDD	153. GND	
137. GND	154. GND	
138. VDD		
140. GND		
142. GND		
144. VDD		
143. GND		
155. VDD		
159. GND		

The power plane of designed PCB should be split into three regions. The below figure clearly describes an ideal power plane partition related to MX98725's pinout. To reduce noise incurred among power traces of these three regions, designers should allocate Ferrit Beads on the interface of +5V traces between digital and analog regions as well as between analog and receive regions. Such placement of Ferrit Beads implies that the power traces between digital and receive regions should be clearly isolated; all the power for receive region should be from analog region. All the traces and resistance/capacitance components for these pins of MX98725 should be located in each specified regions. As for placement of the 25MHz oscillator, it is recommended to be apart from receive and analog regions as far as possible to prevent from the impact of harmonic noise.

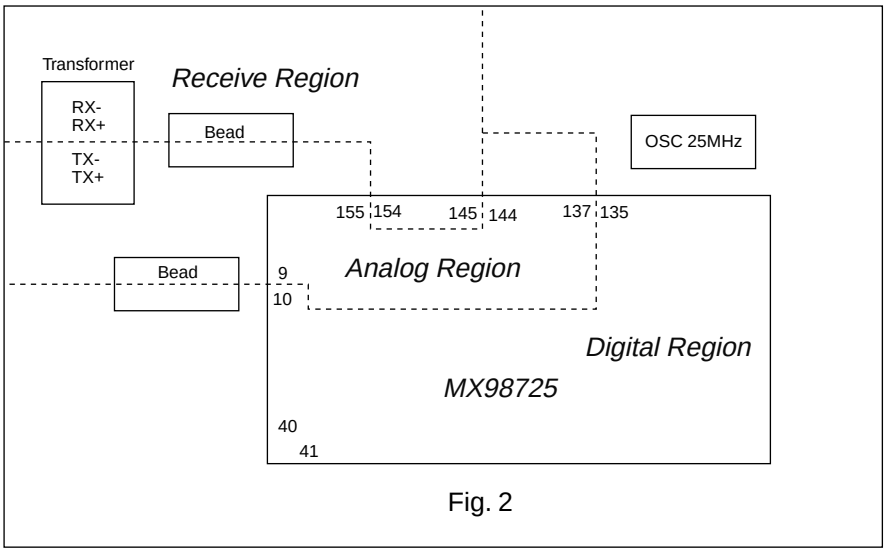


Fig. 2

The MX98725 should be placed as close to the transformer as possible to reduce the length of layout traces and potential noise from coupling on RXIP/N and TXOP/N. The 100Ω resistor between RXIP and RXIN is suggested being located as close to the RXIP and RXIN pins of MX98725 as possible. The 49.9Ω pull up (VDD) resistors for TXOP and TXON pins should be placed as close to the TXOP and TXON pins of MX98725 as possible for impedance match purpose. The receive paths (traces of RXIP/N) need to be in parallel and have equal routing length on the component side of the PCB, and absolutely do not have any through hole on the receive paths to keep the signals clearance.

APPENDIX A: EEPROM FORMAT

BYTE OFFSET (HEX)	DESCRIPTIONS
00-19	Reserved
1a	Magic Packet ID Byte1
1b	Magic Packet ID Byte0
1c	Magic Packet ID Byte3
1d	Magic Packet ID Byte2
1e-39	Reserved
3a	Magic Packet ID Byte5
3b	Magic Packet ID Byte4
3c-59	Reserved
5a	LSB of Sub-Device ID
5b	MSB of Sub-Device ID
5c	LSB of Sub-Vendor ID
5d	MSB of Sub-Vendor ID
5e-6f	Reserved
70	Network ID index: to indicate the starting address of Network ID in length of continuous 6 bytes. The content of this field could be in the range of 00-04h, or 10-14h, or 21-24h, or 31-34h
71-76	Reserved, and should be set to 0
77	LED option: The content of this field will be read by driver for LED setting Bit0: CSR9 Bit 28, LED0SEL Bit1: CSR9 Bit 29, LED1SEL Bit2: CSR9 Bit 30, LED2SEL Bit3: CSR9 Bit 31, LED3SEL
78-79	Reserved, and 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	Reserved, and should be set to 0

APPENDIX B: MAGNETIC COMPONENTS

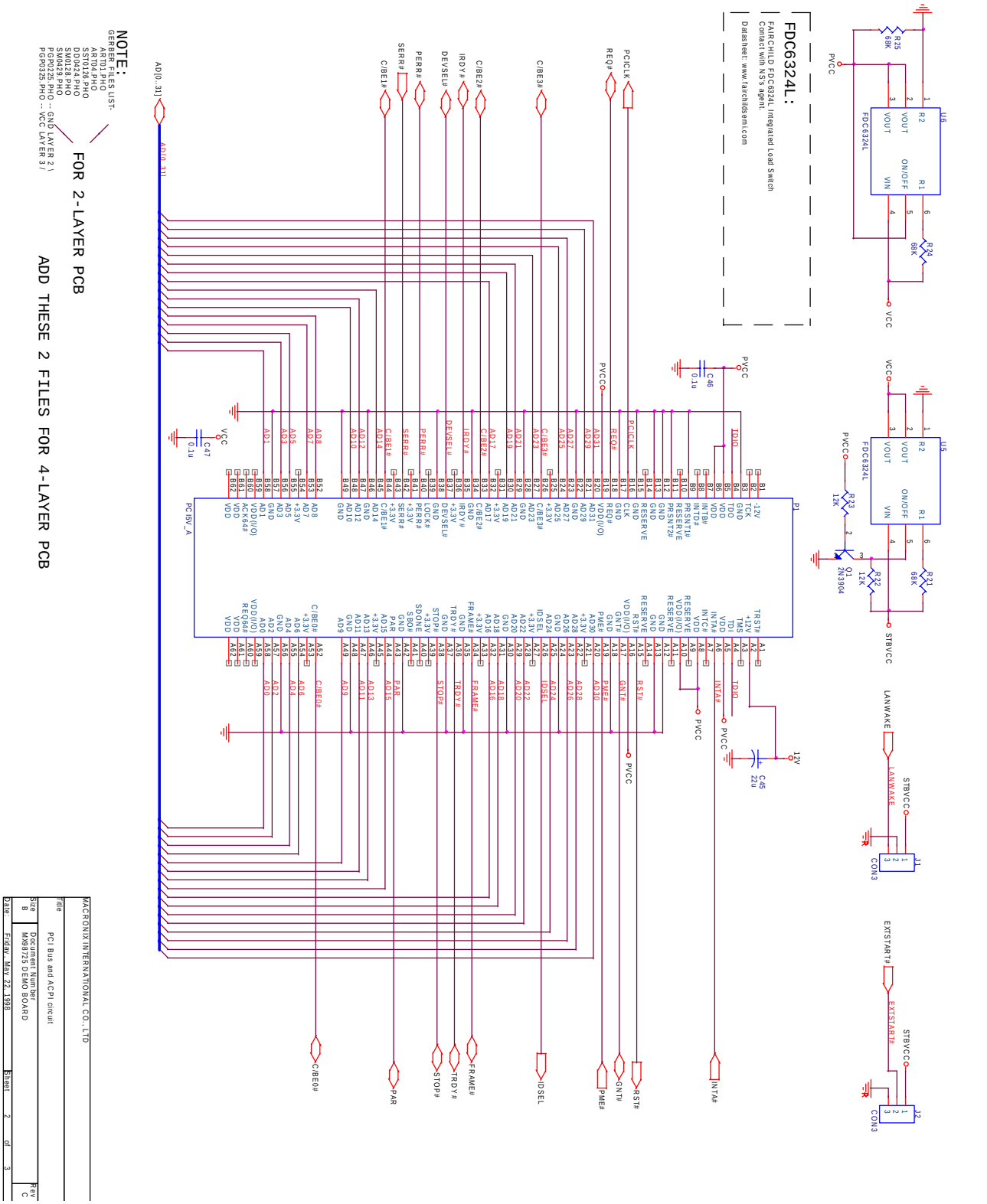
1 BASIC ELECTRICAL SPECIFICATION		
Turn Ratio	Transmit	1:1
	Receive	1:1
OCL	350uH min measured between 0 and 70°C with a 0.1V rms, 100KHz signal at a DC bias between 0 and 8mA.	
LL	0.4uH Max at >1MHz	
Cww	18pF Max	
DCR	0.9W Max per winding	
Isolation Resistance	not less than 1GW @ 2000V rms	
Isolation Voltage	2000V rms Min @ 60Hz for 1 min	
Rise/Fall Time	3ns Min 4ns Max	
Insertion Loss (100 KHz to 100 MHz)	-1.1 dB Max	
CMDR & DCMR (100 KHz to 80 MHz)	38 dB Min	
Cross Talk (100KHz to 80 MHz)	-38 dB Max	
2 REFERENCE VENDORS		
Vendor	Part No	
Valor	ST6118 (PT4171S)	
PE	PE68515	
BelFuse	S558-5999-15	
Delta	LF8200	
Taimic	HSIP-002	

APPENDIX C: THE B.O.M. LIST OF MX98715 DEMO BOARD

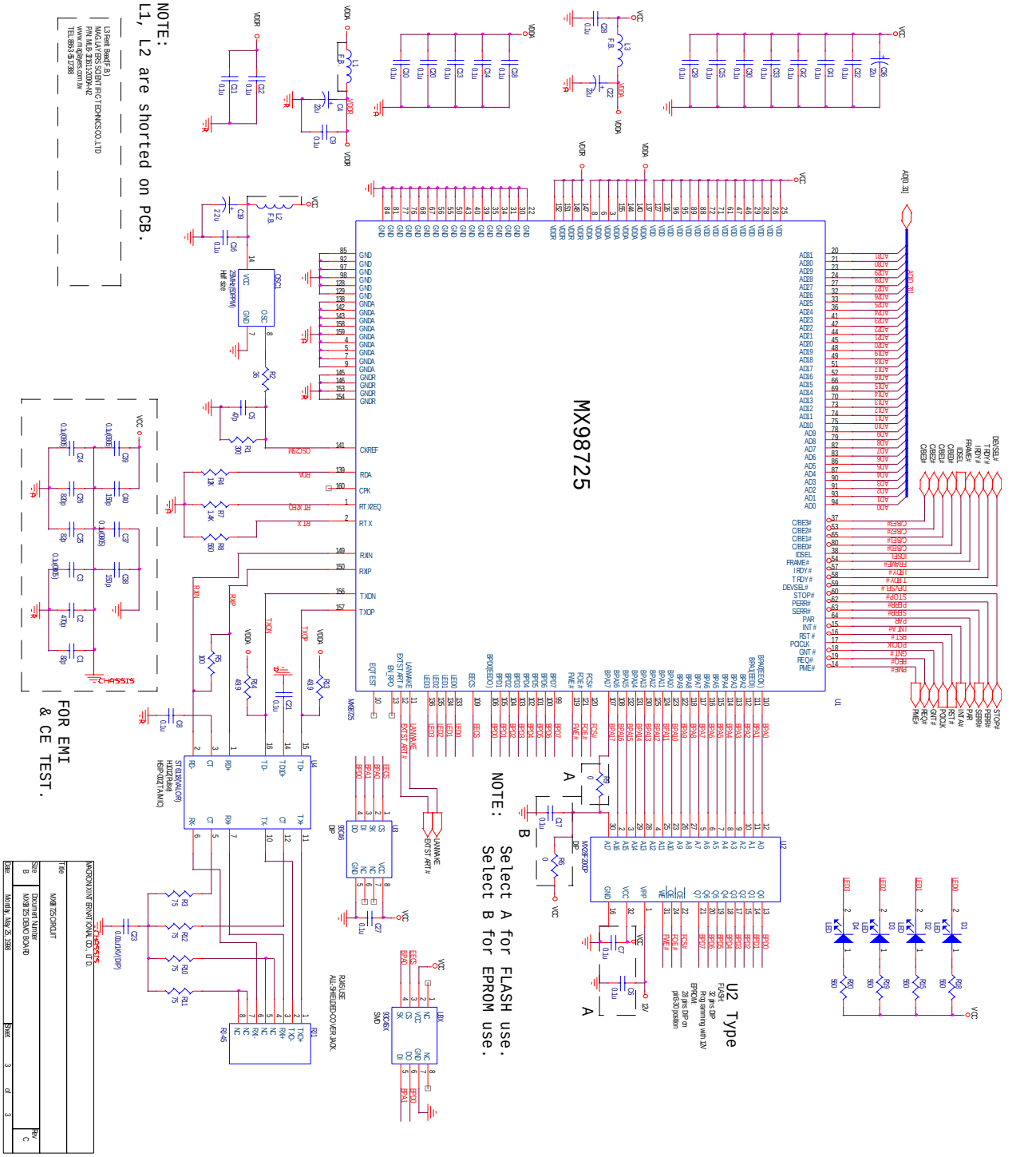
ITEM	QUANTITY	REFERENCE	PART
1	2	C1, C25	82p
2	1	C2	470p
3	4	C3, C24, C37, C39	0.1u (0805)
4	4	C4, C22, C36, C45	22u
5	1	C5	47p
6	24	C6, C7, C8, C9, C10, C11, C12, C13, C14, C15, C16, C18, C20, C21, C27, C28, C29, C30, C32, C33, C41, C42, C46, C47	0.1u (0603)
7	1	C19	2.2u
8	1	C23	0.01u/1KV (DIP)
9	1	C26	820p
10	2	C38, C40	150p
11	4	D1, D2, D3, D4	LED
12	2	J1, J2	CON3
13	1	L3	F.B.
14	1	OSC1	25MHz (50PPM) Half size
15	1	Q1	2N3904 (SMD)
16	1	R2	36
17	4	R3, R10, R11, R12	75
18	3	R4, R22, R23	12K
19	1	R5	100
20	1	R9	0
21	1	R7	1.4K
22	5	R8, R15, R18, R18, R20	560
23	2	R13, R14	49.9
24	3	R21, R24, R25	68K
25	1	RJ1	RJ-45
26	1	U1	MX98725
27	1	U2	MX28F2000P
28	1	U3	93C46
29	1	U4	ST6118 (VALOR), H1012 (Pulse), HSIP-002 (TAIMIC)
30	2	U5, U6	PDC6324L

APPENDIX D: MX98725 APPLICATION SCHEMATIC CIRCUITS

D.1 PCI BUS INTERFACE



D.2 MX98725 AND FRONT-END CIRCUITS



REVISION HISTORY

Revision No.	Description	Date
1.1	To modify resistor R7 value from 1.5KΩ το 1.4KΩ P.2, P.11, P.13)	07/10/1998

MACRONIX INTERNATIONAL CO., LTD.

HEADQUARTERS:
TEL:+886-3-578-8888
FAX:+886-3-578-8887

EUROPE OFFICE:
TEL:+32-2-456-8020
FAX:+32-2-456-8021

JAPAN OFFICE:
TEL:+81-44-246-9100
FAX:+81-44-246-9105

SINGAPORE OFFICE:
TEL:+65-747-2309
FAX:+65-748-4090

TAIPEI OFFICE:
TEL:+886-3-509-3300
FAX:+886-3-509-2200

MACRONIX AMERICA, INC.
TEL:+1-408-453-8088
FAX:+1-408-453-8488

CHICAGO OFFICE:
TEL:+1-847-963-1900
FAX:+1-847-963-1909

http : //www.macronix.com