



DM9000 uP / ISA MAC Programming Guide

Contents:

1. How to Read / Write DM9000 Registers
2. Driver INITIALIZE
 - 2-1. Software RESET (NCR)
 - 2-2. PHY RESET (Internal)
 - 2-3. Registers setting
3. How to access Serial EEPROM (SRAM)
 - 3-1. How to READ SRAM
 - 3-2. How to WRITE SRAM
4. How to check I / O MODE
5. How to access PHY (Internal)
 - 5-1. PHY Registers Offset read / write
 - 5-2. How to READ PHY
 - 5-3. How to WRITE PHY
 - 5-4. How to select PHY MODE for media type
 - 5-5. How to check LINK Status (EXT PHY)
6. How to map HASH Table and NODE (MAC) Address
7. How to Transmit packets
8. How to Receive packets
 - 8-1. How to identify RX a packet READY
 - 8-2. Get RX a packet STATUS and LENGTH
 - 8-3. Get RX a packet DATA
9. Flow Control
10. Others
 - 10-1. How to enable Sample Frame Wake_Up function
 - 10-2. How to fill Sample Frame patterns
 - 10-3. Serial EEPROM Format



1. How to Read / Write DM9000 Registers

DAVICOM NIC DM9000 supports 8 different sets of I/O base (INDEX_PORT and DATA_PORT address) in host ISA mode: 0x300/0x304, 0x310/0x314, 0x320/0x324, 0x330/0x334, 0x340/0x344, 0x350/0x354, 0x360/0x364 and 0x370/0x374.

I/O base default is 0x300/0x304: INDEX_PORT address and DATA_PORT address. You must choice one set above for I/O access latched from strap pins (see Datasheet ch.9.1) or loaded from Serial EEPROM uP control Bit[11:9] (see Datasheet ch.7). INDEX_PORT address is the register address of DM9000 to select one register. DATA_PORT address is the access address mapping to the data port after the command register setting, then pin.92 CMD =1 High to R/W in this command cycle.

For example:

You want to READ Register 01: I/O out a Byte 1h to INDEX_PORT to select REG_01,
READ_Reg = ior(IOaddr = INDEX_PORT = 300h, reg = 1); ; I/O in REG_01 a Byte from
DATA_PORT

```
int ior(int IOaddr, int reg)
{
    delay(STD_DELAY);
    outp(IOaddr, reg);
    delay(STD_DELAY);
    return inp(IOaddr + 4);          /* You can use inpw() for inport 16-bit word */
}
```

You want to WRITE 1h to Register 05: I/O out a Byte 5h to INDEX_PORT to select REG_05,
iow(IOaddr = 300h, reg = 5h, datab = 1h); ; I/O in a Byte 1h to DATA_PORT, write 1h to REG_05

```
void iow(int IOaddr, int reg, int datab)
{
    outp(IOaddr, reg);
    delay(STD_DELAY);
    outp(IOaddr + 4, datab);       /* You can use outpw() for outport 16-bit word */
    delay(STD_DELAY);
}
```



2. Driver INITIALIZE

Recommendation reset_board procedure of the DM9000 initialization:

- (1) PHY RESET
- (2) PHY POWER_DOWN
- (3) DISABLE all interrupts & RX / TX
- (4) Software RESET
- (5) PHY ENABLE
- (6) Software RESET
- (7) Registers setting

2-1. Software RESET (NCR)

Software RESET:

YOU WRITE NCR REG_00 RST bit NCR.0 = 1 ON then WAIT 20uS OK or DETECT the RST bit of NCR REG_00 to be 0 OFF, and Software RESET OK.

```
iow(IOaddr, 0x00, 1);          /* Software RESET starting */
delay(20); iow(IOaddr, 0x00, 0); /* Software RESET end or do following line */
do {} while(READ_REG= ior(IOaddr, 0x00) ); /* DETECT RESET OK */
```

2-2. PHY RESET (Internal)

DM9000 status default is to cut down the Power of Internal PHY by bit GEPIO0 output logic_1. If Internal PHY is used normally, activate Internal PHY by the following steps:

PHY RESET :

YOU WRITE REG_00 (PHY Offset 0) one word data 8000h into Serial EEPROM.

```
phy_write(IOaddr, 0x00, 8000h); /* RESET PHY (see ch.5-3) */
```

PHY POWER_DOWN :

YOU WRITE GPR REG_1F GEPIO0 bit GPR.0 =1 then POWER DOWN PHY.

```
iow(IOaddr, 0x1F, 1);          /* POWER DOWN PHY */
```

PHY ENABLE :

YOU WRITE GPCR REG_1E GEP_CNTL bit GPCR.0 = 1 to PHY ON, and WRITE GPR REG_1F GEPIO0 bit GPR.0 = 0 to POWER ON PHY for AUTO FAST 100/10M & AUTO Full/Half Duplex, then WAIT 64uS OK.

```
iow(IOaddr, 0x1E, 1);          /* PHY Register ON */
iow(IOaddr, 0x1F, 0);          /* PHY POWER ON */
delay(64);                      /* WAIT 64uS for Internal PHY Ready */
```

2-3. Registers setting

- 1- Program NCR, Network Control Register, REG_00.
Choose normal mode by REG_00 bit[2:1] = 0:0, and 0:1 is MAC Loopback Mode.
Depend on the needs, choose the network operation : the PHY Int. / Ext. choice,
Wake_Up Event Enable, FULL-Duplex Mode choice, Loopback Mode choice.
Please refer to DM9000 Datasheet Page13 ch.6.1 about NCR REG_00.
- 2- Clear Network / Interrupt Status by writing NSR / ISR REG_01 / FE bits High.
NSR & ISR will be automatically cleared after a (bits High) WRITE operation.
- 3- Read Serial EEPROM data.
You can get much useful information from Serial EEPROM: Node (MAC)
Address, Driver Control Information, and so on. (see ch. 3-1 & ch.10-3)
- 4- Set NODE Address (6-byte PAR Registers: REG_10h~REG_15h).
- 5- Set HASH Table (8-byte MAR Registers: REG_16h~REG_1Dh). (see ch.6)
- 6- Set RTFCR, RX/TX Flow Control Register REG_0A if you need Flow Threshold.
- 7- Set IMR, Interrupt Mask Register, REG_FF bit7 = 1 to Enable the automatic
return function of the memory R/W address pointer in NIC SRAM.
Depend on your OS and DDK to handle Mask Registers of NIC Interrupts.
- 8- Program IMR REG_FF bit0 / 1 High ON to Enable TX / RX Interrupt Masks.
Before do this, YOU register your Interrupts handler routines. The steps
depend on the driver implementation. For example: if your driver need the
Interrupt after transmitted a packet, please let REG_FF bit1 PTM = 1 ON;
if you want the DM9000 to automatically generate an Interrupt after received
a packet, please let IMR REG_FF bit0 PRM = 1 ON Enable.
- 9- Program RXCR, RX Control Register, REG_05 to activate NIC receiver.
You must set RXCR REG_05 bit0 RXEN = 1 ON to Enable RX function.
The choices of other bits depend on the needs, please refer to DM9000
Datasheet Page15 ch.6.7 about RXCR REG_05. And, NIC activates now.
- 10-For example :
/* Registers setting */



```
/* Program operating registers depending on your application */
iow(IOaddr, 0x00, 40h);          /* NCR.6 WAKE UP bit ON Enable or not */
iow(IOaddr, 0x02, 0);           /* clear TX Polling if you need Special */
iow(IOaddr, 0x05, REG_05 | 10h); /* discard RX CRC Error Packet if you need*/
iow(IOaddr, 0x0A, REG_0A | 1);  /* Flow Control FLCE bit ON (see ch.9-1) */
iow(IOaddr, 0x0F, REG_0F | 24h); /* WUCR LINK WAKE_UP bits ON or not */
iow(IOaddr, 0x01, 2Ch);         /* clear Network Status latched bits */
iow(IOaddr, 0xFE, 0Fh);         /* clear Interrupt Status latched bits */

/* MAC Node Address and FILTER Hash Table */
dm9000_hash_table(dev);         /* map HASH Table (see ch.3-1 & ch.6 ) */

/* Activate DM9000 */
iow(IOaddr, 0x05, REG_05 | 1);  /* RXCR.0 RXEN bit ON Enable */
iow(IOaddr, 0xFF, 83h);         /* IMR.7 PAR bit ON and TX & RX INT MASK ON */

/* Finding ID and NIC existing if you need beginning */

for (PID=9000h,VID=0A46h, IOaddr=0x300; IOaddr<0x380; IOaddr+=0x10) {
    /* Read if (PID==9000h) then check VID=0x0A46 */
    outp(IOaddr, 0x2A); ID = inp (IOaddr+ 4);
    outp(IOaddr, 0x2B); ID |= (inp(IOaddr+ 4) << 8);
    if (ID == PID) {
        /* Read if (VID==0A46h) then reset_board */
        outp(IOaddr, 0x28); ID = inp (IOaddr+ 4);
        outp(IOaddr, 0x29); ID |= (inp(IOaddr+ 4) << 8);
        if (ID == VID) { /* Temporary functions or Record */
        }
    }
}
}
```



3. How to access Serial EEPROM (SR0M)

DAVICOM DM9000 supports the Serial EEPROM (SR0M) interface of 93C46 and provides a very easy method to access it.

DM9000 NIC takes care all detail steps, so YOU just R/W the 0Bh~0Eh registers which map on the 93C46 address to really READ / WRITE word data in SR0M. 93C46 is a 64 word SR0M (1024 bits) and its word address maps on DM9000 REG_0C 6 bits of bit5:0, but REG_0C bit7:6 are “don't care” for SR0M access.

3-1. How to READ SR0M

SR0M READ steps:

1. WRITE the word address, you want to READ from, into REG_0C.
2. WRITE 04h into REG_0B to start the SR0M READ operation:
REG_0B bit[3] = 0: Select Serial EEPROM Mode (SR0M Mode is normal 0).
REG_0B bit[2] = 1: Issue READ command of Serial EEPROM / PHY.
3. READ REG_0B and WAIT until ERRE bit[0] = 0 OK or following step 4.
4. WAIT at least 150uS then WRITE 0 into REG_0B to clear READ command.
5. READ: the SR0M data High_byte is in REG_0E and Low_byte is in REG_0D.

For example:

READ SR0M word address 2 (map to Address 4&5 of MAC Address):

1. WRITE 02h to REG_0C
`iow(IOaddr, 0x0C, 2h);`
2. WRITE 04h to REG_0B
`iow(IOaddr, 0x0B, 4h);`
3. WAIT until REG_0B ERRE bit[0] = 0 OK or following step 4
`do {} while((READ_REG= ior(IOaddr, 0x0B)) &1); /* DETECT SR0M Ready */`
4. DELAY at least 150uS then WRITE 0 to REG_0B
`delay(150);`
`iow(IOaddr, 0x0B, 0);`
5. REG_0E keeps the SR0M data High_byte for you to READ it
`READ_High_Byte_Reg = ior(IOaddr, 0x0E);`
6. REG_0D keeps the SR0M data Low_byte for you to READ it
`READ_Low_Byte_Reg = ior(IOaddr, 0x0D);`

3-2. How to WRITE SROM

SROM WRITE steps:

1. WRITE the word address, you want to WRITE, into REG_0C.
2. WRITE the written data High_byte into REG_0E and WRITE the written data Low_byte into REG_0D.
3. WRITE 12h into REG_0B to start the SROM WRITE operation:
REG_0B bit4 = 1: WRITE Serial EEPROM Enable.
REG_0B bit3 = 0: select Serial EEPROM Mode (SROM Mode is normal 0).
REG_0B bit1 = 1: Issue WRITE Serial EEPROM / PHY command.
4. READ REG_0B and WAIT until ERRE bit[0] = 0 OK or following step 5.
5. WAIT at least 150uS then WRITE 0 into REG_0B to clear WRITE command.

For example:

WRITE AA55h into SROM word address 8
(map to RESERVED Offset 16&17 in SROM):

1. WRITE 03h to REG_0C
`iow(IOaddr, 0x0C, 8h);`
2. WRITE data AAh (High_byte) to REG_0E and data 55h (Low_byte) to REG_0D
`iow(IOaddr, 0x0E, 0AAh);`
`iow(IOaddr, 0x0D, 55h);`
3. WRITE 12h to REG_0B
`iow(IOaddr, 0x0B, (2h | 10h));`
4. WAIT until REG_0B bit0 = 0 OK or following step 5
`do {} while((READ_REG= ior(IOaddr, 0x0B)) &1); /* DETECT SROM OK */`
5. DELAY at least 150uS or WAIT until REG_0B ERRE bit[0] = 0 OK
`delay(150);`
`iow(IOaddr, 0x0B, 0);`



Or WRITE MAC Address 00E0630E5678 into SROM word address 0&1&2:

```
srom_write( IOaddr, 0x00, E000h ); /* Low byte = 00h, High byte = E0h */  
srom_write( IOaddr, 0x01, 0E63h ); /* Low byte = 63h, High byte = 0Eh */  
srom_write( IOaddr, 0x02, 7856h ); /* Low byte = 56h, High byte = 78h */
```

```
void srom_write( int IOaddr, int offset, unsigned int wdata )  
{  
    /* WRITE SROM Register (see ch.5-3) */  
    iow(IOaddr, 0x0C, offset | 40h );  
  
    /* WRITE SROM data */  
    iow(IOaddr, 0x0D, (wdata & 0xFF) ); /* Low byte of 16-bit word */  
    iow(IOaddr, 0x0E, ( (wdata >> 8) & 0xFF) ); /* High byte of 16-bit word */  
  
    /* Issue SROM WRITE command */  
    iow(IOaddr, 0x0B, (2h | 10h) );  
  
    /* WAIT SROM WRITE complete then clear SROM WRITE command */  
    delay(150);  
    iow(IOaddr, 0x0B, 0);  
}
```

PS. You want to READ MAC Address from SROM word address 0&1&2 (see ch.6):

```
for (i = 0; i < 3; i++) { /* READ MAC Address from SROM */  
    iow(IOaddr, 0x0C, i);  
    iow(db, 0x0B, 4h);  
    delay(150);  
    iow(IOaddr, 0x0B, 0);  
    /* READ SROM word data into DEVICE_address */  
    dev->dev_addr[i*2] = ior(IOaddr, 0x0D); /* Low_byte of 16-bit word */  
    dev->dev_addr[i*2+1] = ior(IOaddr, 0x0E); /* High_byte of 16-bit word */ }  
}
```



4. How to check I / O MODE

4-1. How to Read NIC I / O MODE

The DM9000 supports 3 I/O MODE: Byte / Word / Dword of memory command to READ / WRITE the H/W DATA_PORT data. And the memory data width is 8-bit / 16-bit / 32-bit on the interface between the processor and RX / TX SRAM.

Depend on H/W DATA BUS designed, you just make sure to check I/O MODE connecting right for work (automatically detecting H/W DATA_PORT on DATA BUS to keep on ISR IOMODE by pin IO16 / IO32).

Please READ ISR Register FEh bits ISR.6&7 IOMODE to check NIC I/O MODE :

Bit 7	Bit 6	I / O MODE (Byte / Word / Dword)
0	0	16-bit mode (WORD MODE)
0	1	32-bit mode (DWORD MODE)
1	0	8 - bit mode (BYTE MODE)

For example:

```
IO_Mode = ior(IOaddr, 0xFE) >> 6;          /* ISR bit7:6 keeps I/O MODE */
```



5. How to access PHY (Internal)

The DM9000 provides a very easy method to access PHY data, and it's very similar to access SROM data (see ch.3).

The DM9000 takes care all detail steps, so YOU just READ / WRITE the PHY Registers which map to the register address (Offset) that you want to access it.

5-1. PHY Registers Offset read / write

The PHY of the DM9000 only supports 18 registers which are map to REG_0C bit[4:0].

The PHY address PHY_ADR bit[1:0] is defined in REG_0C bit[7:6] to select PHY.

The bit width of the PHY is 16-bit WORD in SROM: EE_PHY_H REG_0E is High Byte Data and EE_PHY_L REG_0D is Low Byte Data.

The default register address of the PHY is "0x001B": Offset 27 in SROM, and it's named Offset 0 in PHY.

The most three significant bits of the PHY are force to "0x000B".

5-2. How to READ PHY

PHY READ steps:

1. WRITE the PHY Register Offset into REG_0C bit[4:0] and the PHY address bit[1:0] = 01 into REG_0C bit[7:6].
2. WRITE 0Ch into REG_0B to start the PHY READ operation:
 - REG_0B bit[3] = 1: select PHY Mode (if 0 select SROM Mode, see ch.3-1)
 - REG_0B bit[2] = 1: Issue READ command of Serial EEPROM / PHY.
3. READ REG_0B and WAIT until ERRE bit[0] = 0 OK or following step 4.
4. WAIT at least 150uS then WRITE 0 into REG_0B to clear READ command.
5. READ: the PHY data High_byte is in REG_0E and Low_byte is in REG_0D.

For example:

READ the PHY Register Offset 4h (PHY Media Mode):

1. WRITE Offset = 4h to REG_0C bit[4:0] and 01 into REG_0C bit[7:6]
iow(IOaddr, 0x0C, (Offset = 4h | 40h));
2. WRITE 0Ch to REG_0B
iow(IOaddr, 0x0B, (4h | 8h));
3. WAIT until REG_0B ERRE bit[0] = 0 OK or following step 4
do {} while((READ_REG= ior(IOaddr, 0x0B)) &1); /* DETECT PHY Ready */
4. DELAY at least 150uS then WRITE 0 to REG_0B
delay(150);
iow(IOaddr, 0x0B, 0);
5. READ REG_0E : High_byte of PHY and REG_0D : Low_byte of PHY
READ_High_Byte_Reg = ior(IOaddr, 0x0E);
READ_Low_Byte_Reg = ior(IOaddr, 0x0D);

Or READ the PHY Register Offset 0x04: READ_phy = phy_read(IOaddr, 0x04);
unsigned int phy_read(int IOaddr, int offset)

```
{
    /* Write PHY Register*/
    iow(IOaddr, 0x0C, offset | 40h );

    /* Issue PHY READ command */
    iow(IOaddr, 0x0B, (4h | 8h) );

    /* WAIT PHY READ complete then clear PHY READ command */
    delay(150);
    iow(IOaddr, 0x0B, 0);

    /* READ PHY Register data */
    return ior(IOaddr, 0x0D) | (ior(IOaddr, 0x0E) << 8);
}
```

5-3. How to WRITE PHY

PHY WRITE steps:

1. WRITE the PHY Register Offset into REG_0C bit[4:0] and the PHY address bit[1:0] = 01 into REG_0C bit[7:6].
2. WRITE High_byte of the written data into REG_0E and WRITE Low_byte of the written data into REG_0D.
3. WRITE 0Ah into REG_0B to start the PHY WRITE operation:
REG_0B bit[3] = 1: select PHY Mode (if 0 select SROM Mode, see ch.3-2)
REG_0B bit[1] = 1: Issue WRITE command of Serial EEPROM / PHY.
4. READ REG_0B and WAIT until ERRE bit[0] = 0 OK or following step 5.
5. WAIT at least 150uS then WRITE 0 into REG_0B to clear WRITE command.

For example:

WRITE data 2100h into the PHY Register Offset 0h (100M FULL Duplex Mode):

1. WRITE Offset = 0h to REG_0C bit[4:0] and 01 into REG_0C bit[7:6]
`iow(IOaddr, 0x0C, (Offset = 0h | 40h));`
2. WRITE data 21h (High_byte) to REG_0E and data 0 (Low_byte) to REG_0D
`iow(IOaddr, 0x0E, 21h);` (see ch.5-4 if phy_reg4 = 101h)
`iow(IOaddr, 0x0D, 00h);`
3. WRITE 0Ah to REG_0B
`iow(IOaddr, 0x0B, (2h | 8h));`
4. WAIT until REG_0B ERRE bit[0] = 0 OK or following step 4
`do {} while((READ_REG= ior(IOaddr, 0x0B)) &1);` /* DETECT PHY OK */
5. DELAY at least 150uS then WRITE 0 to REG_0B
`delay(150);`
`iow(IOaddr, 0x0B, 0);`



Or WRITE data 8000h into the PHY Register Offset 0h (PHY RESET):

```
phy_write(IOaddr, 0x00, 8000h);          /* RESET PHY */

void phy_write(int IOaddr, int offset, unsigned int wdata)
{
    /* WRITE PHY Register (see ch.3-2) */
    iow(IOaddr, 0x0C, offset | 40h );

    /* WRITE PHY data */
    iow(IOaddr, 0x0D, (wdata & 0xFF) );    /* Low byte of 16-bit word */
    iow(IOaddr, 0x0E, ( (wdata >> 8) & 0xFF) ); /* High byte of 16-bit word */

    /* Issue PHY WRITE command */
    iow(IOaddr, 0x0B, (2h | 8h) );

    /* WAIT PHY WRITE complete then clear PHY WRITE command */
    delay(150);
    iow(IOaddr, 0x0B, 0);
}
```



5-4. How to select PHY MODE for media type

10M Half Duplex Mode:

```
PHY REG_04 = 21h;  
PHY REG_00 = 0000h;
```

10M FULL Duplex Mode:

```
PHY REG_04 = 41h;  
PHY REG_00 = 1100h;
```

100M Half Duplex Mode:

```
PHY REG_04 = 81h;  
PHY REG_00 = 2000h;
```

100M FULL Duplex Mode:

```
PHY REG_04 = 101h;  
PHY REG_00 = 2100h;
```

For example:

Running the phy_mode function of 100M FULL Duplex Mode following:

```
phy_reg4 = 101h;  
phy_reg0 = 2100h;  
phy_write(IOaddr, 4, phy_reg4);           /* Set PHY Media Mode */  
phy_write(IOaddr, 0, phy_reg0);  
iow(IOaddr, 0x1E, 1);                     /* Let GEPIO0 output */  
iow(IOaddr, 0x1F, 0);                     /* Enable PHY */
```



5-5. How to check LINK Status (EXT PHY)

If you want to know the LINK Status of the Internal PHYceiver, please READ the bit NSR.6 LINKST of NSR, Network Status Register 01h, to check the LINK Status is 1 = OK or 0 = FAIL.

If you use External PHYceiver chip, you should set the bit NCR.7 EXT_PHY of NCR, Network Control Register 00h, to Enable External PHY first. Then you can READ the bit LINKST of NSR to check the LINK Status is 1 = OK or 0 = FAIL.

For example:

```
iow(IOaddr, 0x00, REG_00 | EXT_PHY);    /* External PHY Enable if used */  
  
Link_Status = ior(IOaddr, 0x01)>> 6 & 1;
```



6. How to map HASH Table and NODE (MAC) Address

The DM9000 has a physical address (MAC Address) map on REG_10h~ REG_15h to get the 6-byte values from SROM word address 0&1&2.

Note: DA (MAC Address) first byte is on REG_10h, and the last byte is on REG_15h.

The DM9000 also provides a 64-bit Hash Table in REG_16h~ REG_1Dh to accept the incoming frame with Multicast Address.

Following the useful sample & CRC-function, you program the physical (MAC) address and map the Multicast Address to 64-bit Hash Table:

```
/* dev_addr[6] : the Node (MAC) Address , map on Physical Address Register PAB0~PAB5  
   mc_list[8] : the Multicast Address , map on Multicast Address Register MAB0~MAB7   */
```

```
static void dm9000_hash_table(struct DEVICE *dev)  
{  
    struct dev_mc_list *mcptr = dev->mc_list;  
    int mc_cnt = dev->mc_count;  
    u32 hash_val;  
    u16 i, ofset, hash_table[4];  
  
    /* READ MAC Address from SROM */  
    for (i = 0; i < 3; i++)    {  
        iow(IOaddr, 0x0C, i);  
        iow(db, 0x0B, 4h);  
        delay(150);  
        iow(IOaddr, 0x0B, 0);  
        /* READ SROM word data into DEVICE_address */  
        dev->dev_addr[i*2] =    ior(IOaddr, 0x0D);    /* Low_byte of 16-bit word */  
        dev->dev_addr[i*2+1] =    ior(IOaddr, 0x0E);    /* High_byte of 16-bit word */    }  
  
    /* set Node (MAC) Address */  
    for (i = 0, ofset = 0x10; i < 6; i++, ofset++)  
        iow(IOaddr, ofset, dev->dev_addr[i]); /* WRITE MAC Address into 6-byte PAB0~PAB5 */
```



```
/* clear Hash Table 4 words*/
for (i = 0; i < 4; i++)
    hash_table[i] = 0000h;

/* Broadcast Address */
hash_table[3] = 8000h;

/* HASH Table calculating for 64-bit Multicast Address */
for (i = 0; i < mc_cnt; i++, mcptr = mcptr->next)    {
    hash_val = cal_CRC( (char *)mcptr->dmi_addr, 6, 0) & 3Fh;
    hash_table[hash_val / 16] |= (u16) 1 << (hash_val % 16);    }

/* WRITE HASH Table into 8-byte MAB0~MAB7 (Multicast Address) for MAC MD Table */
for (i = 0, ofset = 0x16; i < 4; i++)    {
    iow(IOaddr, ofset++, hash_table[i] & 0xFF);
    iow(IOaddr, ofset++, (hash_table[i] >> 8) & 0xFF);    }
}

/* Calculate the CRC value of the RX packet
flag =1 : return the reverse CRC (for the received packet CRC)
0 : return the normal CRC (for Hash Table index)    */
unsigned long cal_CRC(unsigned char * Data, unsigned int Len, u8 flag)
{
    unsigned long Crc = 0xFFFFFFFF;

    while (Len--){
        Crc = CrcTable[(Crc ^ *Data++) & 0xFF] ^ (Crc >> 8);    }

    if (flag)
        return ~Crc;
    else
        return Crc;
}
```

7. How to Transmit packets

Before transmit a packet, the data of the packet must be stored in TX_RAM which is in the DM9000 internal SRAM address 0000h~0BFFh by using the WRITE command to MWCMD REG_F8. The length of the packet are defined High_byte in REG_FD and Low_byte in REG_FC. Set the bit[0] TXREQ (Transmit Request) of TXCR REG_02 will transmit the packet. The DM9000 will set a completion flag latched on NSR REG_01 bit[2] TX1END =1 in toggle, and generate an interrupt latched on ISR REG_FE bit[1] PTS =1 if IMR REG_FF bit[1] PTM = 1, to indicate that the packet is transmitted.

The DM9000 supports Byte / Word / Dword memory command to WRITE the data to TX_RAM depend on the system application automatically. You can get this memory data width from I / O MODE bit[7:6] IOMODE of ISR REG_FE. (see ch.4-1)

7-1. Transmit a packet

CHECK the memory data width I / O MODE: (see ch.4-1)

Byte Mode: if ISR REG_FE bit[7:6] = 10
Word Mode: if ISR REG_FE bit[7:6] = 00 or 11
Dword Mode: if ISR REG_FE bit[7:6] = 01

WRITE ransmitted data into TX RAM:

/ TX_DATA[] : Transmitted data, TX_LENGTH : Transmitted data length */*

I / O Byte Mode:

```
out( IOaddr, 0xF8 );           /* Select MWCMD REG_F8 */  
for (i = 0; i < TX_LENGTH; i++)/* Loop WRITE a byte data each into TX RAM */  
    out( IOaddr+4, TX_DATA[i] );
```

I / O Word Mode:

```
    Tmp_Length = ( TX_LENGTH + 1 ) / 2;
```



聯傑國際股份有限公司

07/06/08

```
outp( IOaddr, 0xF8 );
```

```
/* Select MWCMD REG_F8 */
```



```
for (i = 0; i < Tmp_Length; i++) /* Loop WRITE a word data each into TX RAM */
    outpw( IOaddr+4, ( (unsigned int *) TX_DATA)[i] );
```

I / O Dword Mode:

```
    Tmp_Length = ( TX_LENGTH + 3 ) / 4;
    outp( IOaddr, 0xF8 );          /* Select MWCMD REG_F8 */
    for (i = 0; i < Tmp_Length; i++) /* Loop WRITE a dword data each into TX RAM
*/
        outpl( IOaddr+4, ( (unsigned long *) TX_DATA)[i] ); // outpl() is MACRO for 32-bit
```

WRITE transmitted data length into DM9000:

```
/* WRITE transmitted data length High_byte to REG_FD */
    iow( IOaddr, 0xFD, HIGH_BYTE(TX_LENGTH) );
/* WRITE transmitted data length Low_byte to REG_FC */
    iow( IOaddr, 0xFC, LOW_BYTE(TX_LENGTH) );
```

START to send TX:

```
/* WRITE a TX Request command TXREQ into TXCR REG_02 */
    iow( IOaddr, 0x02, 1 );
```

CHECK a completion flag:

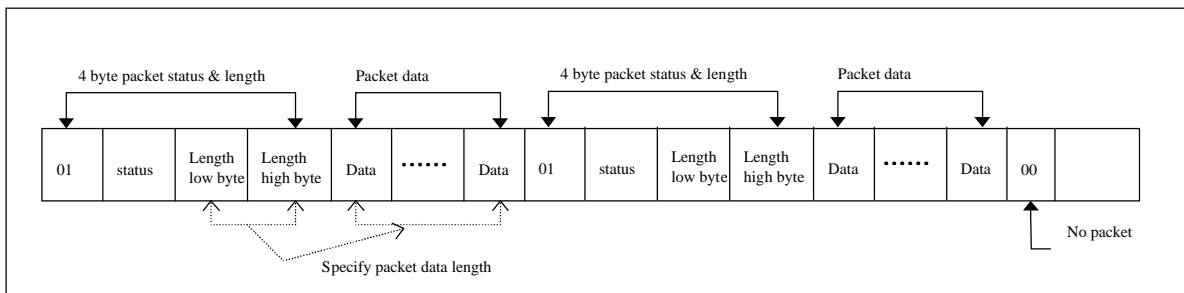
This routine can be moved to the interrupt handler if your driver is used the interrupt driven. If your driver is used the polling method, you can put this routine into the TX function for waiting TX completed.

```
    outp( IOaddr, 0x01 );          /* Select NSR REG_01 */
    :repeat
    Tmp_REG_01 = inp( IOaddr + 4 ); /* READ NSR TX1END = 1 OK */
    :until if( (Tmp_REG_01 >> 2) & 1 ) /* just for bit[2] TX1END equals "1" */
```

8. How to Receive packets

The received data is stored in RX_RAM which is in the DM9000 internal SRAM address 0C00h~3FFFh. There are 4 bytes header data for each packet in the RX_RAM. The first byte is the packet ready flag. If this byte is “01h”, it means a packet is READY to receive. If this byte is “0”, it means no any packet is in RX_RAM. Before you read the others byte, you must make sure the first byte is “01h”. The second byte is the STATUS of the ready packet. Its format is like as REG_06. According this status byte, the ready packet is a good packet or an error packet can be identified. The third and fourth bytes are the LENGTH of this ready packet. The others byte are the ready packet’s DATA.

The ready packet format is like as the following block diagram:



8-1. How to identify RX a packet READY

REG_F0: Memory data read command without the increment of the Read_Pointer of the RX_RAM. This register is used for check packet’s first byte.

Reading twice REG_F0 command to get the final updated data, because the DM9000 read the current Read_Pointer data of the internal memory to a internal latched register after a memory read command (like READ REG_F0 or REG_F2).

<For example> check any packet READY

```

outp( IOaddr, 0xF0);          /* Select MRCMDX REG_F0 */
:repeat
Tmp_REG_F0 = inp( IOaddr + 4 ) /* Dumy read, skip the old data on REG_F0 */
Tmp_REG_F0 = inp( IOaddr + 4 ) /* Get the most updated data */
:until if( Tmp_REG_F0)        /* Get packets in RX FIFO now if "01" READY */

```

8-2. Get RX a packet STATUS and LENGTH

REG_F2: Memory data read command with the increment of the Read_Pointer of the RX_RAM. The Read_Pointer increased after reading the memory read command REG_F2. The size of the increment of the Read_Pointer is depend on system application which the IO BUS width is Byte / Word / Dword to increase 1/ 2 / 4 bytes respectively.



You can get the RX status, length and packet's data by reading MRCMD REG_F2 command.

For example: Get the RX STATUS and the RX LENGTH

I/O Byte Mode:

```
outp( IOaddr, 0xF2 );           /* Select MRCMD REG_F2 */
RX_Status = inp(IOaddr+4) + (inp(IOaddr+4) << 8); /* Get first word : RX STATUS */
RX_Length = inp(IOaddr+4) + (inp(IOaddr+4) << 8); /* Get second word: RX LENGTH */
```

I/O Word Mode:

```
outp( IOaddr, 0xF2 );           /* Select MRCMD REG_F2 */
RX_Status = inpw( IOaddr + 4 ); /* Get first word : RX STATUS */
RX_Length = inpw( IOaddr + 4 ); /* Get second word: RX LENGTH */
```

I/O DWord Mode:

```
outp( IOaddr, 0xF2 );           /* Select MRCMD REG_F2 */
(unsigned long)tmp_data = inpl( IOaddr + 4 ); /* Get a Dword data for 32-bit */
RX_Status = (unsigned int)(tmp_data & 0xFFFF); /* Get first word : RX STATUS */
RX_Length = (unsigned int)( (tmp_data >> 16) & 0xFFFF ); /* Get second word: RX LENGTH */
```

8-3. Get RX a packet DATA

```
char RX_data[2048];           /* Declare a data structure to store RX DATA */
```

I/O Byte Mode:

```
outp( IOaddr, 0xF2 );           /* Select MRCMD REG_F2 */
for( i = 0 ; i < RX_Length ; i++ ) /* LOOP get RX DATA */
    RX_data[ i ] = inp( IOaddr + 4 );
```

I/O Word Mode:

```
outp( IOaddr, 0xF2 );           /* Select MRCMD REG_F2 */
Tmp_Length = ( RX_Length + 1 ) / 2; /* Word READ loop count */
for( i = 0 ; i < Tmp_Length ; i++ ) /* LOOP get RX DATA */
    ( (unsigned int *)RX_data )[ i ] = inpw( IOaddr + 4 );
```

I/O DWord Mode:

```
outp( IOaddr, 0xF2 );           /* Select MRCMD REG_F2 */
Tmp_Length = ( RX_Length + 3 ) / 4; /* Dword READ loop count */
for( i = 0 ; i < Tmp_Length ; i++ ) /* LOOP get RX DATA */
    ( (unsigned long *)RX_data )[ i ] = inpl( IOaddr + 4 ); /* inpl() is MACRO inport 32-bit long word */
```



9. Flow Control

9-1. How to setup Flow Control

Please select REG_0A RTFCR, RX / TX Flow Control Register, WRITE bit RTFCR.0 FLCE = 1 ON to Enable Flow Control of the DM9000 advanced functions.

The DM9000 automatically responds TX Pause Packet from my destination address to make a pause for Flow Control slowly to prevent packet's overflow.

For example:

```
iow(IOaddr, 0x0A, 1);          /* Flow Control FLCE bit ON Enable */
```

9-2. How to force Flow Control Pause Packet

You WRITE bit RTFCR.5 TXPEN = 1 ON to force TX Pause Packet Enable, and NIC automatically sends TX Pause Packet with pause_time = 0000h / FFFFh depend on bit[7] TXP0 or bit[6] TXPF setting, then auto clears the bit TXP0 or TXPF (see DM9000 Datasheet Page 16 ch.6.11 about RTFCR REG_0A).

The toggle condition to send a Pause Packet depend on the value HWOT / LWOT on REG_09 FCTR, Flow Control Threshold Register, for checking RX FIFO High or Low Water trigger (see DM6000 Datasheet Page 16 ch.6.10 about FCTR REG_09).

For example:

```
iow(IOaddr, 0x0A, 20h | TXP0); /* Flow Control TXPEN bit ON Enable */
```

Note: Back Pressure Mode is for Half Duplex Mode only, and you set bit BKPM / BKPA on REG_0A RTFCR to Enable (see DM9000 Datasheet Page 15 ch.6.9 about BPTR).

The mask bits for each sample frame pattern are defined in the P-10-2-2.

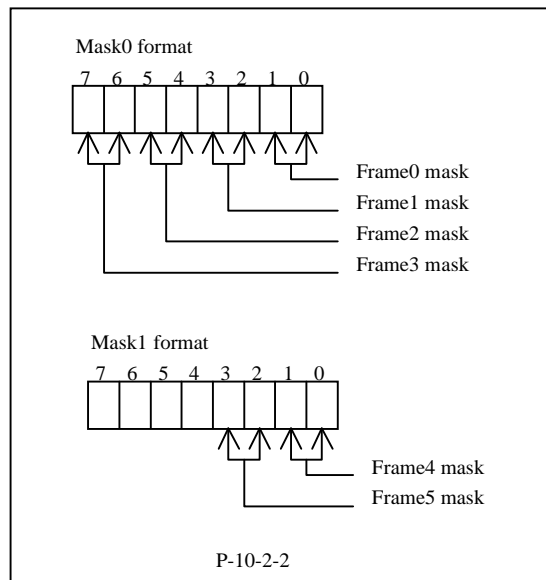
The mask definition please refers P-10-2-3.

Note that don't fill mask bits "11" to the sample frame patterns that are not used.

Mask bits description

bit1	bit0	description
0	0	this byte don't care
0	1	this byte musk check
1	0	this byte don't care
1	1	end of mask

P-10-2-3





10-3. Serial EEPROM Format:

name	Word	offset	description
MAC address	0	0 ~ 5	6 Byte Ethernet address
Auto Load Control	3	6-7	Bit 1:0=01: Update Vendor ID and Product ID Bit 3:2=01: Accept setting of WORD6[8:0] Bit 5:4=01: Accept setting of WORD6[11:9] Bit 7:6=01: Accept setting of WORD7[3:0] Bit 9:8=01: Accept setting of WORD7[6:4] Bit 11:10=01: Accept setting of WORD7[7] Bit 13:12=01: Accept setting of WORD7[8] Bit 15:14=01: Reserved
Vendor ID	4	8-9	2 Byte Vendor ID (Default: 0A46h)
Product ID	5	10-11	2 Byte Product ID (Default: 9000h)
uP control	6	12-13	Bit0: uP CS active low when set (default: active high) Bit1: uP/ISA IOR active low when set (default: active low) Bit2: uP/ISA IOW active low when set (default: active low) Bit3: uP/ISA IRQ active low when set (default: active high) Bit4: uP/ISA IRQ is open-collect (default: force output) Bit5: uP/ISA IOWAIT active low when set (default: active low) Bit6: uP/ISA IOWAIT is open-collect (default: force output) Bit7: uP/ISA IO16 active low when set (default: active low) Bit8: uP/ISA IO16 is open-collect (default: force output) Bit1 1:09: ISA IObase (default: 000 = 300h) 000 : 300h 001 : 310h 010 : 320h 011 : 330h 100 : 340h 101 : 350h 110 : 360h 111 : 370h Bit15:12: Reserved
Wake-Up mode control	7	14-15	Bit0: WOL active low when set (default: active high) Bit1: WOL is pulse mode (default: level mode) Bit2: magic wakeup event enabled when set. (default: no) Bit3: link_change wakeup event enabled when set (default: no) Bit4: magic wakeup event enabled if USB in suspend state (default: yes) Bit5: link_change wakeup event enabled if USB in suspend state (default: yes) Bit6: sample frame wakeup event enabled if USB in suspend state (default: yes) Bit7: LED mode 1 (default: 0) Bit8: internal PHY is enabled after power-on (default: no) Bit15:9: Reserved
RESERVED	8-63	16 ~ 127	USEABLE



11. Transmit Jabber and Receive Watch-dog Timer

You want to transmit Jabber if you let TXCR bit[6] TJDIS = 0 default on REG_02 TXCR, please check TSR_P1 bit[7] TJTO = 1 on REG_03 TSR_P1 when the transmitted frame is more than 2048 bytes:

For example:

```
if( TSR_P1 >> 7 ) Record_TX_Packet1_2048[i] =1; // TX packet[i] >2048 bytes
```

Please refer to DM9000 Datasheet Page13 ch.6.3 about RSR REG_02.

You want to receive Watch-dog Timer if you let RXCR bit[6] WTDIS = 0 default on REG_05 RXCR, please check RSR bit[4] RWTO = 1 on REG_06 RSR when the receiving frame is more than 2048 bytes:

For example:

```
if( (RSR >> 4) &1 ) Record_RX_Packet_2048[i] =1; // RX packet[i] >2048 bytes
```

Please refer to DM9000 Datasheet Page15 ch.6.7 about RSR REG_06.