



10/100M Controller (DM9102 serial) + 1M PHY

(DM9801) Program Guide

Content:

1. How many kinds of Adapters?
2. Find Adapter
 - 2-1. Method 1
 - 2-2. Method 2
3. How to program MAC controller registers?
4. How to select External PHY (DM9801) or Internal MII (10/100M Ethernet PHY)
 - 4-1. Check EXT MII device (DM9801/DM9802) present or not.
 - 4-2. Program to Home Run and Long Run Mode
 - 4-3. How to auto sense the media mode between 10/100M Ethernet and 1M Home net? The priority algorithm
5. How to Reset 10/100M Ethernet (+ DM9801 Home Run)
 - 5-1. Software Reset 10/100M Ethernet Controller
 - 5-2. Software Reset (External) PHY
 - 5-3. Hardware Rest External PHY
6. How to program PHY register
7. How to read SROM
 - 7-1. What is SROM (Serial ROM) for?
 - 7-2. Format of SROM
8. How to program SROM
 - 8-1. SROM Read Operation
 - 8-2. SROM Write Operation
 - 8-3. SROM_CRC Calculation Algorithm
 - 8-4. ID_BLOCK_CRC Calculation Algorithm
9. How to program DM9102 serial ID table
 - 9-1. What is the ID table?
 - 9-2. What is a setup frame?
 - 9-3. Perfect Filtering
 - 9-4. Hash Filtering
 - 9-5. How to calculate the 9 bit CRC and set the right bit
 - 9-6. Hash Filtering only

- 10. Transmit/Receive Descriptor
 - 10-1. Transmit Descriptor
 - 10-2. Receive Descriptor
- 11. New functions for DM9102D
 - 11-1. IP/TCP/UDP Checksum Offload
 - 11-2. Zero Copy Supporting

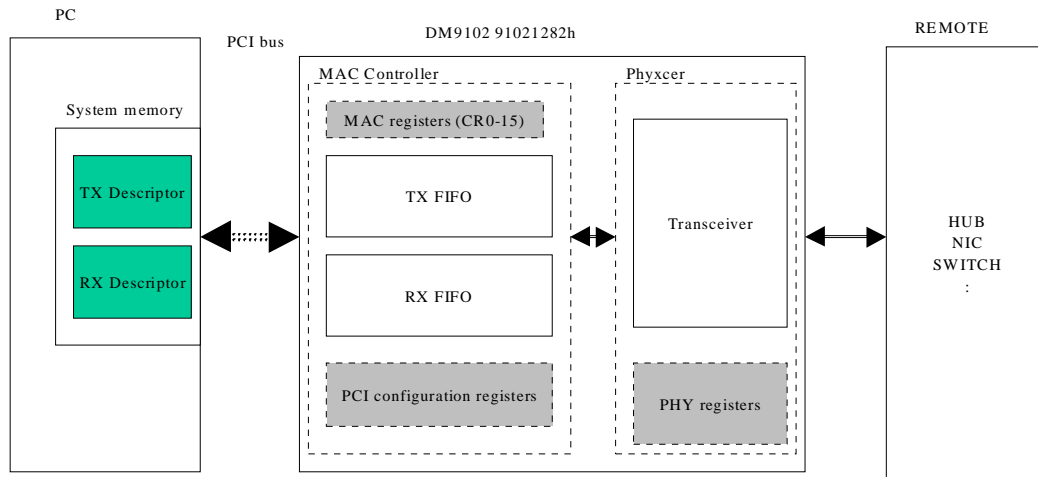


Fig. 1



DM9102 Serials Programming Guide

10/100M PCI Single Chip Ethernet NIC controller

1. How many kinds of Adapters?

- 1-1. 10/100M Ethernet Adapter (a RJ-45 connector and a DM9102x Chip)
- 1-2. 1/10/100M Ethernet + Home Run (one RJ-45 connector for Ethernet, two RJ-11s for Home run and DM9102x and DM9801 chips)
- 1-3. 1/10/100M Ethernet + Long Run (one RJ-45 connector for Ethernet, two RJ-11 ones for Long run and DM9102x and DM9802 chips)
- 1-4. 1M Home Run (two RJ-11 connectors for Home run and DM9102x and DM9801 Chips)
- 1-5. 1M Long Run (two RJ-11 connectors and DM9102x and DM9802 Chips)

Mode	DM9102x	DM9801	DM9802	RJ-45	RJ-11
10/100M Ethernet	✓	x	x	1	0
1/10/100M Ethernet +Home Run	✓	✓	x	1	2
1/10/100 Ethernet +Long Run	✓	x	✓	1	2
1M Home Run	✓	✓	x	0	2
1M Long Run	✓	x	✓	0	2

PS:

Home Run: Meet Home PNA V1.1 spec. The transmitted distance is 1000ft for AWG26 cable.

Long Run: Meet TUT Long Run V1.0 spec. The transmitted distance is 2000ft for AWG26 cable.



2. Find out the Adapter Type:

2-1. Check PCI BIOS

2-1-1. Put **0B101h** into AX. (AH=> PCI_FUNCTION_ID,
AL=> PCI_BIOS_PRESENT)

2-1-2. Call INT 1ah.

2-1-3. Check EDX: “**204943350h**” to identify this system and find the PCI adapter.

2-2. Method 1 (Refer PCI BIOS 2.1 spec.)

2-2-1. From “Check PCI BIOS”, we can make sure that we find a PCI device. Then you need to check Vendor ID and Device ID.

2-2-2. Use **Read Configuration DWord** (Check Vendor ID & Device ID)

2-2-2-1. The information of Vendor ID and Device ID are put in DM9102A PCI register 0h. The bit 16:31 of PCIID register are the product number assigned by Davicom. The bit 0:15 of PCIID register are a registered number from SIG.

2-2-2-2. Put **0B10Ah** into AX. (AH=>PCI_FUNCTION_ID,
AL=> READ_CONFIG_DWORD)

2-2-2-3. Put **0h** into DI. (DI=> Register Number)

2-2-2-4. Put **0h** into BX. (BH=>Bus Number, BL=>Device Number in upper 5bits, Function Number in low 3 bits)

2-2-2-5. Call INT 1ah.

2-2-2-6. CF: **1**:error, **0**:success.

2-2-2-7. If success, check ECX :”**91021282h**”. Find DM9102A Adapter.

If fail, check the value of BX by a loop and add 8 to BX from 0x0000 to 0x0100. The Bus Number is depended on the programmer. Usually, the Bus Number is 0 or 1 for Ethernet/HomeNet card application. The value of Bus Number can be 0 from 255.

Example:

DM9102x vendor ID: 91021282 h

```
;; Check PCI BIOS ?  
;; -----  
mov     ax, 0b101h  
int     1ah  
mov     al, 1           ;; errcode=1  
cmp     edx, 20494350h
```



DM9102 Serials Programming Guide

10/100M PCI Single Chip Ethernet NIC controller

```

                                j ne    PCB_not_present    ;; Not support PCI
BIOS
                                ;; Check vendor ID, device ID. To find DM9102 device
                                ;; -----
                                mov     di, 0                ;; Read
vendor/device ID
                                mov     ax, 0b10ah          ;; Try to read the
PCI
                                mov     bx, [bp+14]          ;; Got Second Arg,
handle no.
                                int     1ah                 ;; configuration
space
                                jc      not_present
                                cmp     ecx, VendorDeviceID ;; Check the card
ID
                                je      NIC_DM9102
                                jmp     not_present
```

2-3. Method 2 (Refer PCI BIOS 2.1 spec.)

2-3-1. From “Check PCI BIOS”, we can make sure that we find a PCI device.

Then you need to check Vendor ID and Device ID.

2-3-2. Use Class Code

2-3-2-1. Put **0B103h** into AX, put **Class Code** into ECX, put **Index** into SI.

(Ethernet Class Code: **020000h**, Index: **0~N**)

(Index: it presents the order of Ethernet NIC card and you can get any information of any Ethernet NIC card from here. The value can be set by software programmer.)

2-3-2-2. Call INT 1ah.

2-3-2-3. Check return code AH, 00 h => SUCCESSFUL

86h => DEVICE_NOT_FOUND

2-3-2-4. Return BH => Bus Number (0...255)

BL => Device Number in upper 5 bits, Function Number in bottom 3 bits

2-3-3. Use **Read Configuration DWord** (Check Vendor ID, Device ID)

2-3-3-1. Put **0B10Ah** into AX. (AH=>PCI_FUNCTION_ID,

AL=> READ_CONFIG_DWORD)

2-3-3-2. Put **0h** into DI. (DI=> Register Number)



DM9102 Serials Programming Guide

10/100M PCI Single Chip Ethernet NIC controller

2-3-3-3. Put Xh into BX. (BH=>Bus Number, BL=>Device Number in upper 5bits, Function Number in low 3 bits)
(The BX value from 2-2-1-4)

2-3-3-4. Call INT 1ah.

2-3-3-5. CF: **1**:error, **0**:success.

2-3-3-6. If success, check ECX :”91021282h”. Find DM9102A Adapter.

Example:

```
        xor     si, si                ;; index
Keep_search_dm910x:
        mov     ax, 0b103h           ;; Find PCI class
code
        mov     ecx, 020000h        ;; Ethernet class
code
        int     1ah                 ;;
        jc     not_present          ;;
        cmp     ah, 86h             ;;
        je     DeviceSearchEnd      ;;
        ;; Ethernet adapter found
        mov     di, 0               ;; Read PCI vendor ID
        mov     ax, 0b10ah          ;;
        int     1ah                 ;;
        cmp     ecx, DM9100VID      ;; DM9100 Vendor ID
        je     NIC_DM9100
```

Ps:

Bus Number: A number in the range 0,...,255, it uniquely selects a PCI bus.

Device ID: A predefined field in configuration space, it and the Vendor ID uniquely identify the device .

Device Number: A number in the range 0,...,31, it uniquely selects a device on a PCI bus.

Vendor ID: A predefined field in configuration space, it and the Device ID uniquely identify the device .

Function Number: A number in the range 0,...,7, it uniquely selects a function of a multi-function PCI device.(Ethernet Card = 0)



3. How to program MAC controller registers:

The DM9102x implement 16 Control and Status registers (CR0-15), which can be accessed by the host. These CRs are double long word aligned. All CRs are set to their default values by hardware or software reset unless otherwise specified. All Control and Status Registers with their definitions and offset from IO or memory Base Address are shown up. You can get the clear information from “DM9102x Data Sheet”.

Example:

- i. If you want to force the Adapter in “10M Half-Duplex” mode.
You need to set CR6 to “0x0000” and PHY register 0 to “0x0000”.
- ii. If you want to force the Adapter in “10M Full-Duplex” mode.
You need to set CR6 to “0x80200” and PHY register 0 to “0100”.
- iii. If you want to force the Adapter in “100M Half-Duplex” mode.
You need to set CR6 to “0x80000” and PHY register 0 to “0x2000”.
- iv. If you want to force the Adapter in “100M Full-Duplex” mode.
You need to set CR6 to “0x80200” and PHY register 0 to “2100”.
- v. If you want to force the Adapter in “1M Home Run” mode.
You need to set CR6 to “0x40000” and PHY register 0 to “0x0000”.

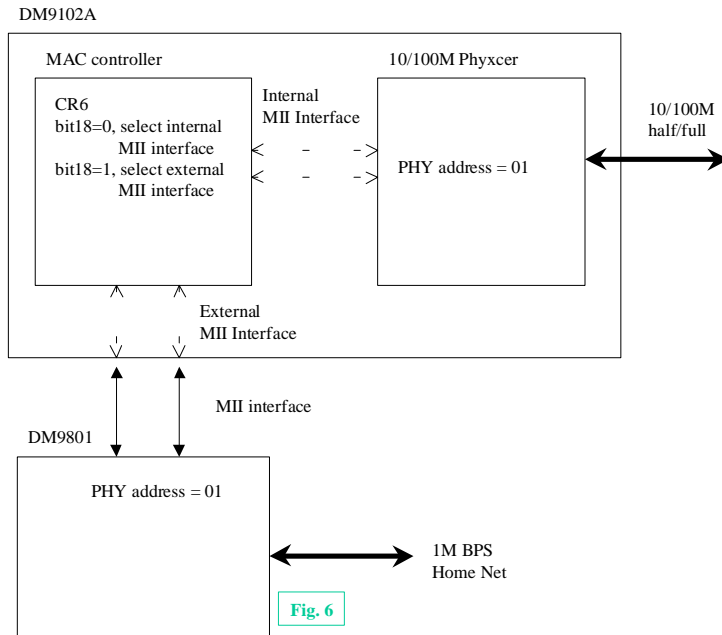
Ps: Please read or write CR registers with Double Words.

Ps: Interrupt can be generated as a result of various events. CR5 contains all the status bits that might cause an interrupt. CR7 contains an enable bit for each of the events that can cause an interrupt. The interrupts are divided into two groups: the normal interrupts and the abnormal interrupts. Interrupt bits are cleared by writing a “1” to the bit position. When all enable bits in one of the interrupt groups are cleared, the corresponding summary bit is cleared.

4. How to select External PHY (DM9801) or Internal MII

(10/100M Ethernet PHY)

Please refer to the Fig. 6.



Normal mode : Only used Internal MII for 10/100M Phyxcer, CR6 bit18 no means.

Internal mode with External MII mode : Support internal MII for 10/100M Phyxcer and external MII for DM9801. When CR6 bit18 equals 0, it selects internal MII interface for 10/100M Phyxcer. When CR6 bit18 equals 1, it selects external MII interface for DM9801.

To set Normal or External MII mode, is depend on DM9102x test1, test2, clkrun#, ma8, ma9 pin.

	Test1(pin 75)	Test 2(pin 71)	Clkrun#(pin 36)	Ma8(pin 84)	Ma9(pin 85)
Normal mode	0	1	X	0	0
Internal mode with External MII mode	0	0	0	0	1/0 ★

If you want to use DM9801, you must make sure your NIC work on the internal mode with external MII mode.



DM9102 Serials Programming Guide

10/100M PCI Single Chip Ethernet NIC controller



MA9 =1 (set up by hardware; Mode can not be changed.)

MA9 =0 & MII_Mode =1(CR6<18>) (select external MII interface; Mode can be changed by software.)

4-1. Check EXT MII device (DM9801/DM9802) present or not.

4-1-1: Let CR6 bit18=1

4-1-2: Read PHY REG2 and check it must be 0181h

4-1-3: Read PHY REG3 and mask off least 4 bits then it must be b900h

Read (PHY REG3) and 0xfff0h ; Skip **Model revision** (REG3<3:0>)

4-1-4: When pass step2 and 3, it is DM9801 or DM9802

4-2. Program to Home Run and Long Run Mode

4-2-1. Let CR6 bit18=1, set to the external MII interface

4-2-2. Let CR6 bit9=0, set to the half duplex mode

4-2-3. **First you have to** check PHY register 31 is 110fh or not. If it is 110fh, the PHY register is DM9802 (Long Run). If not, the PHY register is DM9801 (Home Run).

a. Home Run: Using CR12 to reset DM9801 and after 1-2sec read DM9801 register Add 3 to 24<7:0> and write it to register 25<7:0>. Then change DM9801 register 25<15:8> to f0h, and change register 16<31:0> to 1005h.

b. Long Run: Change register 16 to 0005h, and change register 25<7:0> to 02h.

4-2-3. Set the **Power/Speed, Ignored Remote Command, Disable 25% Increment of Noise.**

Mode	PHY register 16 Bit
Ignored Remote Command	15
Disable 25% increment of noise	12
Remote Command Low Power	11
Remote Command High Power	10
Remote Command Low Speed	9
Remote Command High Speed	8
Network Card Power	2
Network Card Speed	1

a. Home Run default value: Disable 25% increment of Noise + Ignored Remote Command + Low Power, High Speed.
(PHY register 16: 9005h)



DM9102 Serials Programming Guide

10/100M PCI Single Chip Ethernet NIC controller

- b. Long Run default value: Ignored Remote Command + Low Power, High Speed.
(PHY register 16: 8005h)
- c. Ignored Remote Command: If remote terminate send a command to set Power/Speed of your card. The PHY of your card can “Ignored” or not “Ignored” the command.
- d. Remote Command Power/Speed: If you want to set the Power/Speed of remote terminate card. You can set the PHY register 16<11:8> of your card to set the Power/Speed of remote terminate card.
- e. Network Card Speed/Power: You can set yourself card in High/Low of Power/Speed by these bits.
. If you want to set the remote termination Power/Speed, you can refer the following settings: (in Home Run mode)
 - i. High Power, High Speed: 9507h
 - ii. High Power, Low Speed: 9603h
 - iii. Low Power, High Speed: 9905h
 - iv. Low Power, Low Speed: 9A01h

4-3. How to auto sense the media mode between 10/100M Ethernet and 1M Home net?

4-3-1. The priority algorithm:

10/100M phyxcer is the high priority. If 10/100M phyxcer links failed, select DM9801 Home Net.

4-3-2. 10/100M phyxcer link check:

CR12 is a byte general purpose data I/O registers, Bit0~7 are the data I/O ports. Bit8 is the control bit. Bit8=1, set bit0~7 port direction: 0:input, 1:output, Bit8=0, bit0~7 are the data I/O port. In the driver init program, driver need to the following setting.

Write 0180h to CR12 ; Set bi0~6 are input port and bit 7 is output port

CR12 bit7: output ports, connect to 10/100M phyxcer or 1M Home Run **reset** pin.

CR12 bit0: input port, 10M-link status, and “1” link ok.

CR12 bit1: input port, 100M-link status, and “1” link ok.

CR12 bit2: input port, Full-duplex.



DM9102 Serials Programming Guide

10/100M PCI Single Chip Ethernet NIC controller

4-3-3.1M Phyxcer link check:

Since Link Status (pin 7) of DM9801 is not connected to pin 89 of DM9102x, the way to know the link status of DM9801 is to check DM9801 Register 1 <2> (link status). The link status of DM9801 has latch function, so reading twice is necessary to get the right data. If pin 7 of DM9801 is connected to pin 89 of DM9102x, the link status can be checked by CR12<6>.

If the 1M link ok, you need to set CR6<9> to “0” in “Half-Duplex” mode.

ps: CR12<6> responds back the real time link status of 1M Home Net (about 1ms), so it is not necessary to do twice reading as PHY Register 1<2>.



5.How to Reset 10/100M Ethernet (+ DM9801 Home Run):

5-1. Software Reset 10/100M Ethernet Controller:

Use CR0 (CR0=1h) Software Reset, software reset command has no effect on the PCI configuration register or on CR6<18> external MII_Mode.

5-2. Software Reset (External) PHY:

5-2-1.Internal PHY: When you use the Internal PHY register, you can use the PHY register 0<15> to “Software Reset” Internal PHY. You just need to write the PHY register 0<15> to be “1”, then Internal PHY will be software reset by DM9102x. And the PHY register 0<15> will automatically clear when the Software Rest complete.

5-2-2.External PHY (DM9801):

When you use the External PHY register(DM9801), you can use PHY register 0 <15> to “Software Reset” External PHY. You also just need to set the PHY register 0<15> to be “1” , then External PHY will automatically clear when the Software Rest complete.

5-3. Hardware Rest PHY:

When you need to “Hardware Rest” PHY. You can use CR12<7> to “Hardware Reset” PHY, first you can write “1” to CR12<7> after initialization. Then the PHY of DM9102A will be Hardware reset. After Hardware reset “OK”, you need to write “0” to CR12<7>.

6. How to program PHY register:

The serial management interface to obtain and control the status of PHY management register set through an MDC and MDIO. The Management Data Clock (MDC) is equipped with a maximum clock rate of 2.5MHz. In read/write operation, the management data frame is 64-bit long start with 32 contiguous logic one bits (preamble) synchronization clock cycles on MDC.

The Start of Frame Delimiter (SFD) is indicated by a <01> pattern followed by the operation code (OP): <01> indicates Read operation and <10> indicates Write operation. For read operation, a 2-bit turnaround (TA) field between Register Address field and Data field is provided for MDIO to avoid contention. "Z" stands for the state of high impedance. Following turnaround time, a 16-bit data is read from or written onto management registers.

The MDC is control by CR9 <16>, and The MDIO is control by CR9 <17> (Data out), <19> (Data in).

The CR9 <18> is MII Management Read/Write mode selection.

Please refer the Fig. 11 and 12. (Management Interface – Read/Write)

p.s. Please read and write PHY registers with Double Words.

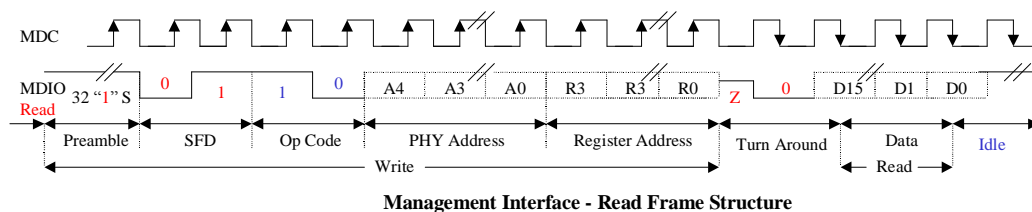


Fig. 11

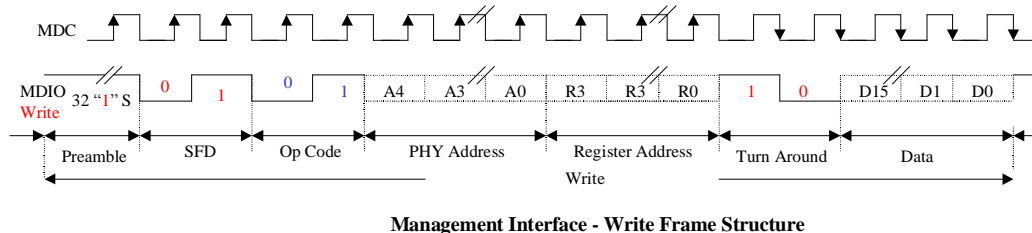


Fig. 12



7. How to read SROM:

7-1. What is SROM (Serial ROM) for?

SROM is used to provide the DM9102x information to the driver. It's space must be 64 words or more to store the configuration data. There are two kinds of SROMs used for the DM9102x 10/100M Controller. One is V3.0, the other is V4.01. There is a Sub-Vendor ID block in SROM which may be used to store LAN card type in the future (10/100M Ethernet, 10/100M Ethernet + 1M Home Run ...,etc.). V4.01 SROM can support the Adapter to select Media Mode (10 Half-Duplex, 100 Full-Duplex, 1M Home Run, 1M Long Run, ...,etc.). The default is Auto Detect Media mode in **0000h**. The function of Adapter to select Media mode also provides the capability of selecting the LAN card type. When the LAN card types of Sub-Vendor ID and Adapter to select Media Mode conflict, it is Adapter to select Media Mode to decide the type because it has higher priority.

7-2. Format of SROM:

7-2-1. V3.0

Field Name	Offset	Size
Subsystem ID block	0	18
SROM version	18	1
Controller count	19	1
Controller_0 Information	20	n
Controller_1 Information	20+n	M
: (depends on controller count)	:	:
CRC checksum	126	2

7-2-2-1. Subsystem ID Block

Subsystem ID Block		Byte Offset
Sub Vendor ID		0
Subsystem ID		2
Reserved		4
Reserved		6
NCE	Auto_load_control	8



DM9102 Serials Programming Guide

10/100M PCI Single Chip Ethernet NIC controller

PCI Vendor ID		10
PCI Device ID		12
PMCSR	PMC	14
Reserved	ID_block_CRC	17,16

The Subsystem ID offered the system vendor information. The content will be transferred into the PCI configuration space during hardware reset function. The auto transfer function must to enable the Auto_Load_Control bits.

The Example of DM9102x SROM format, Please reference to DM9102x Datasheet.

7-2-2. V4.01

Field Name	Offset (Bytes)	Size (Bytes)	Value (Hex)	Commentary
Sub-Vendor ID	0	2	0291	0291h : 10/100M Ethernet 0198h : 1/10/100M HomeRun 0298h : 1/10/100M LongRun 0398h : 1M HomeRun 0498h : 1M LongRun



DM9102 Serials Programming Guide

10/100M PCI Single Chip Ethernet NIC controller

Field Name	Offset (Bytes)	Size (Bytes)	Value (Hex)	Commentary
Adapter Media Capability	34	2	000F	Bit 0..7 : Ethernet Capability Bit 0 : 10M Half-Duplex, Bit 1 : 10M Full-Duplex, Bit 2 : 100M Half-Duplex, Bit 3 : 100M Full-Duplex, Bit 4 : 1G Half-Duplex, Bit 5 : 1G Full-Duplex, Bit 6..7 : Reserved Bit 8..15 : HomeNet Capability Bit 8 : 1M HomeRun, Bit 9 : 1M LongRun, Bit 10 : 10M HomeRun, Bit 11 : 10M LongRun, Bit 12..15 : Reserved (000Fh : 10/100M Ethernet 010Fh : 1/10/100M HomeRun 020Fh : 1/10/100M LongRun 0100h : 1M HomeRun 0200h : 1M LongRun)
Adapter Selected Media	36	2	0000	Bit 0..7 : Ethernet Capability Bit 0 : 10M Half-Duplex, Bit 1 : 10M Full-Duplex, Bit 2 : 100M Half-Duplex, Bit 3 : 100M Full-Duplex, Bit 4 : 1G Half-Duplex, Bit 5 : 1G Full-Duplex, Bit 6..7 : Reserved Bit 8..15 : HomeNet Capability Bit 8 : 1M HomeRun, Bit 9 : 1M LongRun, Bit 10 : 10M HomeRun, Bit 11 : 10M LongRun, Bit 12..15 : Reserved (PS : The default value 0000h à Autodetect Media)

8. How to program SROM

First you need to check the SROM is “idle”.

You can select the SROM access for memory interface (CSR9 <11>), and enable the memory write (CSR9<13>). Do the 25 cycles clock for SROM is ready to read. After you check the SROM is idle, you can start to read SROM in the read clock. Please reference Fig. 7,8,9,10.

8-1. SROM Read Operation:

SROM Read Operation

Serial ROM chip select (CR9<0>) =>Cs
 Serial ROM serial clock (CR9<1>) =>Sk
 Serial ROM data in (CR9<2>) =>Din
 Serial ROM data out (CR9<3>) =>Dout

Read operation consist of three phases:

1. Command phase - 3 bits (binary code of **110**)
2. Address phase - 6 bits for 256-bit to 1Kb ROMs, 8 bits for 2Kb to 4Kb ROMs.
3. Data phase - 16 bits

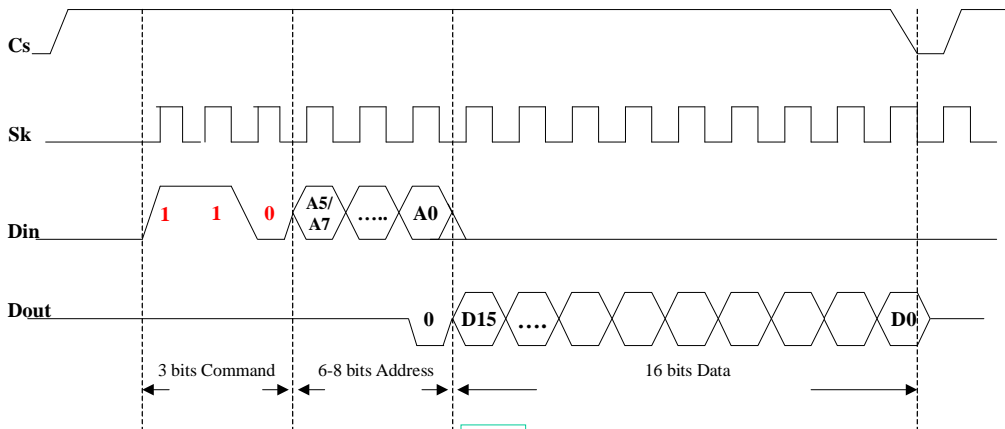
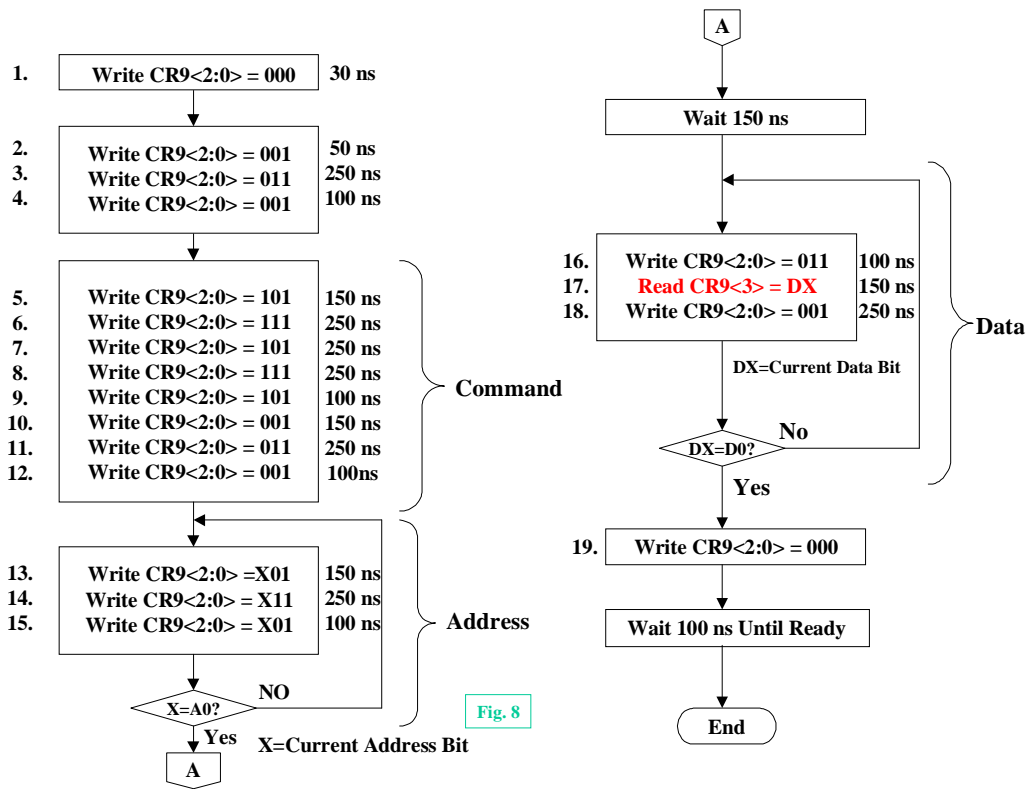


Fig. 7



8-2.SROM Write Operation:

SROM Write Operation

Serial ROM chip select (CR9<0>) =>Cs

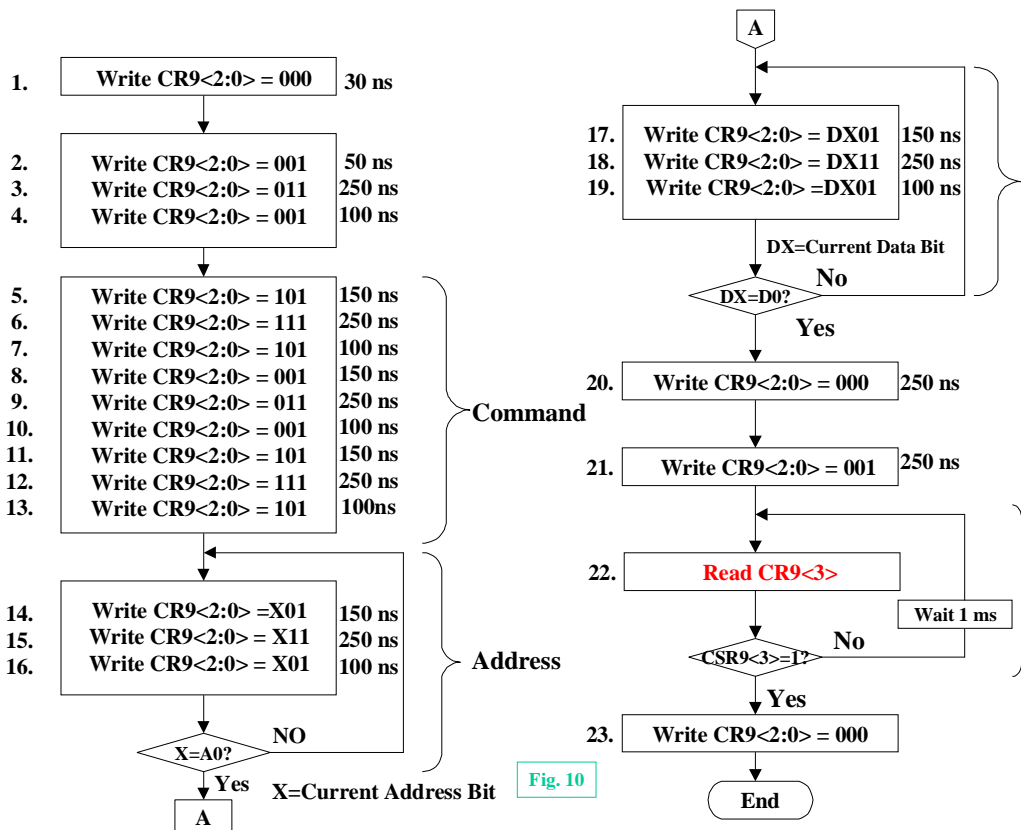
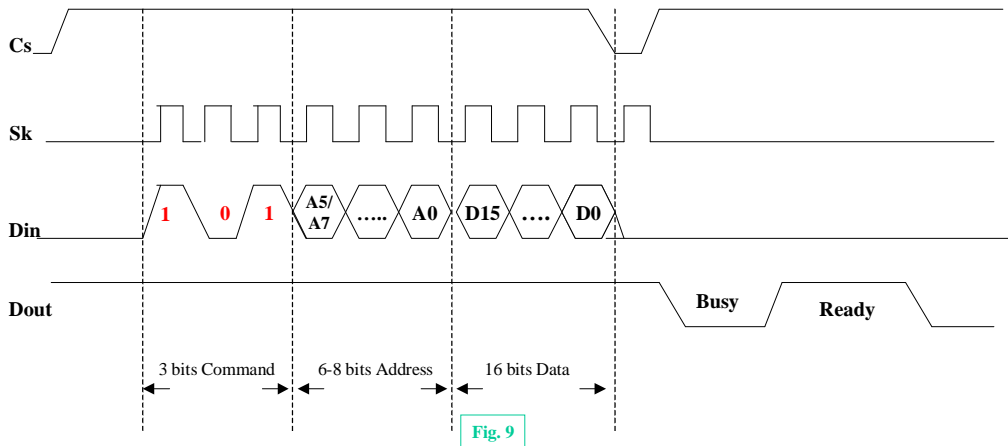
Serial ROM serial clock (CR9<1>) =>Sk

Serial ROM data in (CR9<2>) =>Din

Serial ROM data out (CR9<3>) =>Dout

Read operation consist of three phases:

1. Command phase - 3 bits (binary code of 101)
2. Address phase - 6 bits for 256-bit to 1Kb ROMs, 8 bits for 2Kb to 4Kb ROMs.
3. Data phase - 16 bits





8-3.SROM_CRC Calculation Algorithm:

```
Unsigned short CalcSromCrc (unsigned char *SromData);
#define DATA_LEN 126 //1024 bits SROM
struct{
    unsigned char SromData[DATA_LEN];
    unsigned short SromCRC;
} Srom;
unsigned short CalcSromCrc(unsigned char *SromData)
{
#define POLY 0x04C11DB6L
    unsigned long crc = 0xFFFFFFFF;
    unsigned long FlippedCRC = 0;
    unsigned char CurrentByte;
    unsigned Index;
    unsigned Bit;
    unsigned Msb;
    int I;
    for (Index = 0; Index < DATA_LEN; Index++)
    {
        CurrentByte = SromData[Index];
        For (Bit = 0; Bit <8; Bit++)
        {
            Msb = (crc >> 31) &1;
            Crc <<= 1;

            if (Msb ^ (CurrentByte & 1))
            {
                crc ^= POLY;
                crc|= 0x00000001;
            }
            CurrentByte >>= 1;
        }
    }
    for ( I = 0; I<32; I++)
    {
        FlippedCRC <<= 1;
        Bit = crc &1;
```



```
    crc >>= 1;
    FlippedCRC += Bit;
}

crc = FlippedCRC ^ 0xFFFFFFFF;
return (crc & 0xFFFF)
}
```

8-4.ID_BLOCK_CRC Calculation Algorithm:

```
/*
** This program calculates the CRC which sums the Serial ROM
ID Block header and the Magic
** Information Block.
** In the case of the ID Block header, this serial ROM header
of 9 words is read upon reset of
** the chip. If the CRC result of these 9 words equals 0, it
means the data has been read correctly
** CRC is an 8 bit crc. Polynom is X8 + X2 + X + 1.
** Note that contrary to a regular CRC, this CRC is calculated
on the data stream which flows in this manner.
*/
main()
{
#define POLY 0x6
# define LEN 9      /* for ID Block */
unsigned short DAT[LEN];
int I, Word, n;
char Bit;
unsigned char Bit Val;
unsigned char crc;
n=0;
crc = -1;
for (Word=0; Word<LEN; Word++)
for (Bit=15; Bit>=0; Bit --)
{
    if ((Word == (LEN-1)) && (Bit ==7))
    {
        /*
```



DM9102 Serials Programming Guide

10/100M PCI Single Chip Ethernet NIC controller

```
        ** Insert the correct CRC result into input data stream
in place.
        */
        DAT[LEN-1] = (DAT[LEN-1] & 0xff00) | (unsigned short) crc;
        break;
    }
    n++;
    BitVal = ((DAT[Word] >> Bit) & 1) ^ ((crc >> 7) & 1);
    crc = crc << 1;
    if (BitVal == 1)
    {
        crc ^= POLY;
        crc |= 0x01;
    }
}
```



9. How to program DM9102x ID table:

9-1. What is the ID table?

DM9102x lookup the ID table to decide receive the coming packet or not. DM9102x supports 4 filter mode and set by TDES1 FMB1/FMB0.

9-2. What is a setup frame?

Driver uses the setup frame to program MAC filtering mode and fill ID table. First setup frame must be sent before the reception process is started. A setup frame is 192 bytes. The setup frame does not affect the reception process state, but during setup frame processing, the DM9102x is disengaged from the Ethernet wire.

The setup must be allocated in a single buffer that is long word aligned. First segment (TDES1<29>) and last segment (TDES1<30>) must both 0. When the setup frame load is completed, the 10/100 MAC controller (DM9102x) closed the setup frame descriptor by clearing its ownership bit and by setting all other bits to 1.

9-2-1. First Setup Frame:

The setup frame must be processed before the reception process is started, except when it operates in promiscuous filtering mode.

9-2-2. Subsequent Setup Frames:

Subsequent setup frame maybe queued to the 10/100M MAC controller (DM9102x) despite the reception process state. To ensure correct setup processing, these packets may be queued at beginning of the transmit descriptor's ring or following a descriptor with a **zero-length** buffer.

For the descriptor with a **zero-length**, it should contain the following information:

TDES0<31>	1 (Adapter-owned descriptor)
TDES1<30>	0 (Last segment bit 0)
TDES1<29>	0 (First segment bit 0)
TDES1<27>	1 (Setup Frame)
TDES1<21:11>	0 (Transmit buffer 2 empty)
TDES1<10:0>	0 (Transmit buffer 1 empty)

The address filtering bits (TDES1<22> and TDES1<28>) should the same as in the previous packet. The subsequent setup frames are processed after all preceding frames have been transmitted and the current frame reception is complete.

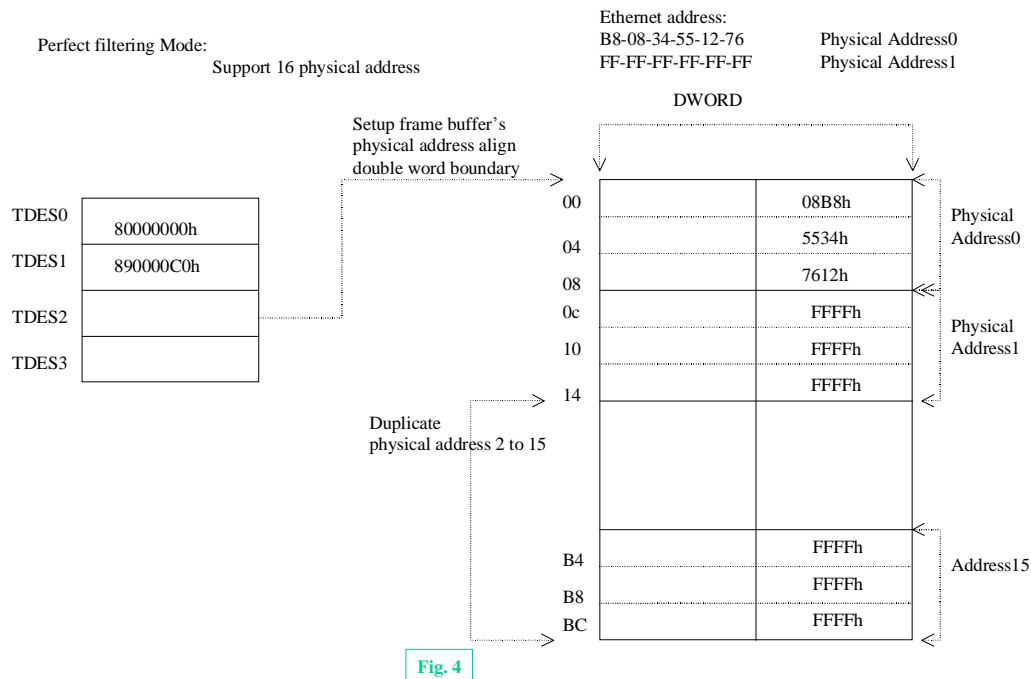
FMB1	FMB0	Filtering Mode	Description
0	0	Perfect Filtering	Support 16 physical address. Receive if match.
0	1	Hash Filtering	Support 1 physical address and 512 hash address. Receive if match.

1	0	Inverse Filtering	Support 16 physical address. Receive if not match.
1	1	Hash-only Filtering	Support 512 hash address. Receive if match.

9-3. Perfect Filtering:

9-3-1. How to set the setup frame for the perfect filtering mode?

Please refer the Fig. 4.



In this example, we set two address, B8-08-34-55-12-76 and FF-FF-FF-FF-FF-FF.

Driver needs to put those two Ethernet addresses into the transmitted buffer physical address 0/1. Driver must duplicate the one Ethernet address to all unused physical address filed. In this case, we duplicated FF-FF-FF-FF-FF-FF to all unused fields.

```

TDES1=890000C0h (SETF=1, FMB1/FMB0=00, CE=1, Buffer
Length=c0h)
TDES0=80000000h (OWN=1, ready to send)
CR6=00002000h (TXSC=1, start to send this setup frame)
Check TDES0 OWN bit until OWN=0
TDES0=7FFFFFFFh, if setting is successful.

```

Note:

When set a setup frame, TDES1 both BD and ED must be zero.

The setup frame buffer must be aligned the double word boundary.

9-4. Hash Filtering:

How to set the setup frame for the hash filtering mode?

Please refer the Fig. 5.

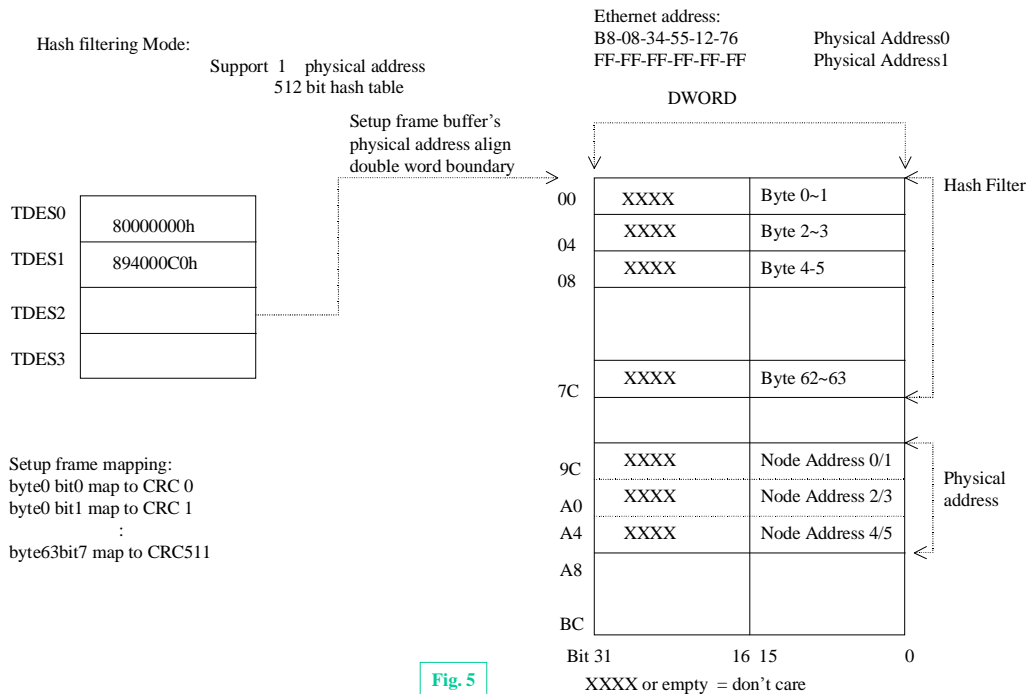


Fig. 5

This mode support 1 physical address (6 bytes) and 512 bit hash table (64 bytes)

The hash table occupies the setup frame first 32 double word but only used 64 byte.

Use the 6-byte multicast address to calculate a 9 bit CRC to index this 512-bit hash table.

CRC 0 index to hash table is the bit0 of first byte.

9-5. How to calculate the 9 bit CRC and set the right bit

```

hash_val = cal_CRC((char ) node_address, 6, 0) & 0x1ff; /* Got
9 bits CRC */
hash_table[hash_val/16] |= (u32) 1 << (hash_val % 16);
/*
Calculate the CRC valude 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;
}

```

TDES1=894000C0h (SETF=1, FMB1/FMB0=01, CE=1, Buffer Length=c0h)

TDES0=80000000h (OWN=1, ready to send)

CR6=00002000h (TXSC=1, start to send this setup frame)

Check TDES0 OWN bit until OWN=0

TDES0=7FFFFFFFh, if setting is successful.

9-6 Hash Filtering only:

It is like as Hash Filtering mode except it did not has the 6-byte physical address. Please refer Fig.5. (Hash Filtering mode)

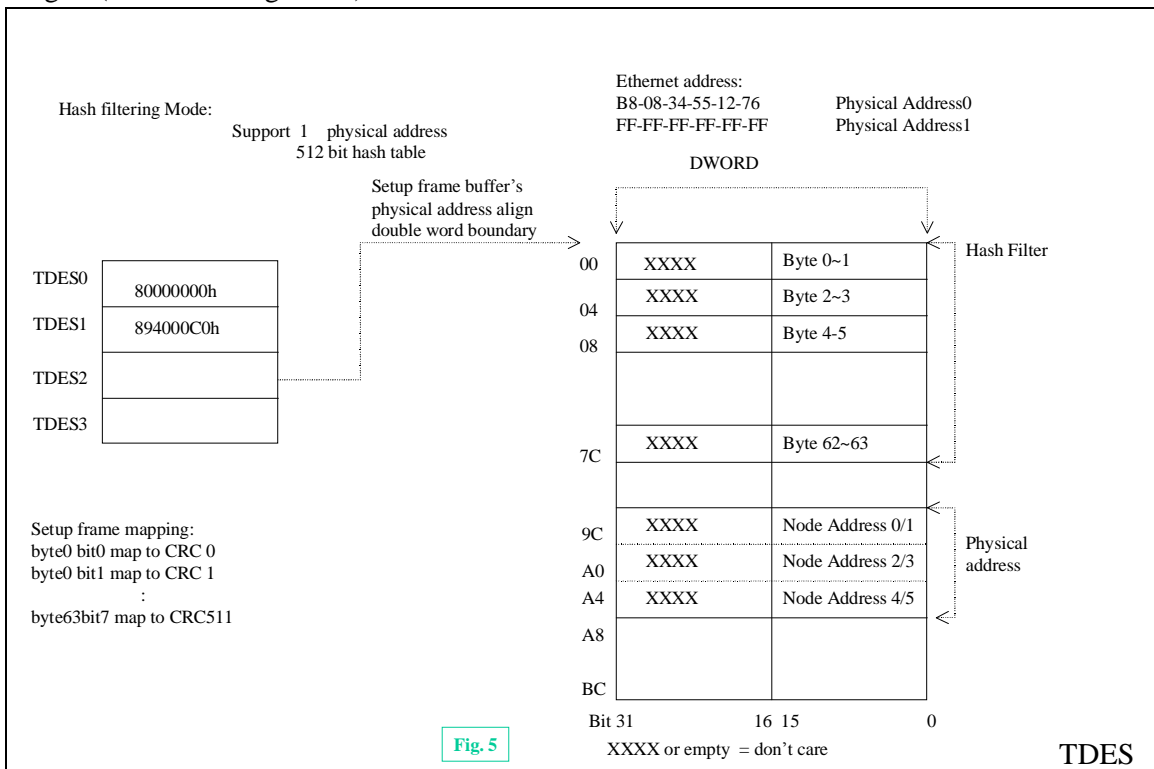


Fig. 5



DM9102 Serials Programming Guide

10/100M PCI Single Chip Ethernet NIC controller

1=994000C0h (SETF=1, FMB1/FMB0=11, CE=1, Buffer Length=c0h)

TDES0=80000000h (OWN=1, ready to send)

CR6=00002000h (TXSC=1, start to send this setup frame)

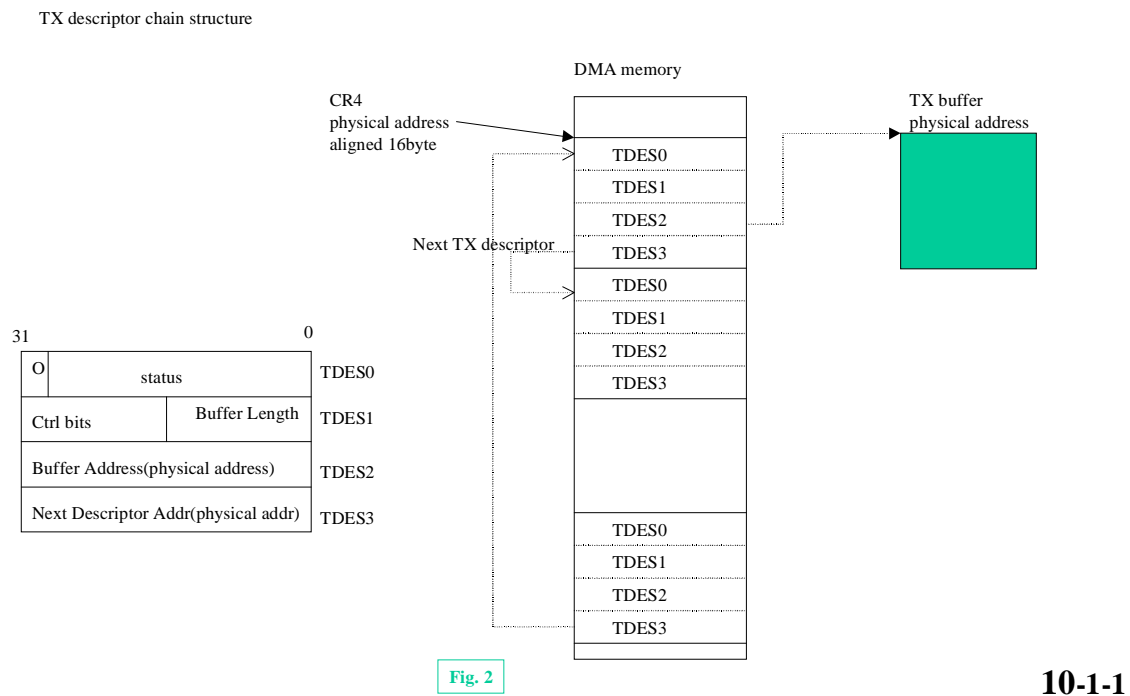
Check TDES0 OWN bit until OWN=0

TDES0=7FFFFFFFh, if setting is successful.

10. Transmit/Receive Descriptor:

There are two kinds of buffer in the host, the transmit buffer and receive buffer. These buffers are in many distributed regions in the host memory. They are linked together and controlled by the descriptor list. The content of each descriptor includes pointer to the buffer, count of the buffer, command and status for the packet to be transmit or received. Each descriptor list starts from the address setting of **CR3** (receive descriptor base address) and **CR4** (transmit descriptor base address). The descriptor list is Chain structure. Please refer Fig. 2,3.

10-1. Transmit Descriptor :



.TDES0: Owner Bit of Transmit Status

<31>: 1 => owned by 10/100M Controller (DM9102x)

0 => owned by host

This bit should be set when the transmitting buffer is filled with data and ready to be transmit. It will reset by 10/100M Controller (DM9102x) after transmitting the whole data buffer.

<30:0>: These bits content includes status of transmitted frame (reference DM9102x data sheet).

10-1-2.TDES1: Transmit buffer control and buffer size

<28/22>: Filtering Mode Bit1/0

FMB1	FMB0	Filtering Type
0	0	Perfect Filtering
0	1	Hash Filtering
1	0	Inverse Filtering
1	1	Hash-only Filtering

<10:0> :Buffer length

10-1-3.TDES2: Buffer Starting Address indicates the **physical** starting address of buffer.

10-1-4.TDES3: Address indicates the **next** descriptor starting address.

10-2. Receive Descriptor:

RX descriptor chain structure

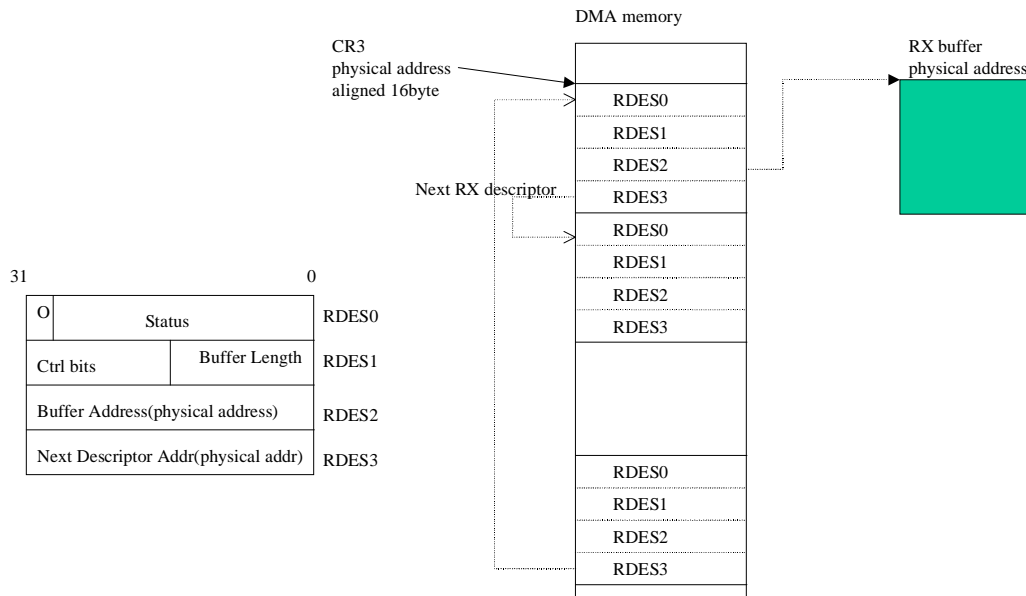


Fig. 3

10-2-1.RDES0: Owner bit of received status

<31>: 1 => owned by 10/100M Controller (DM9102x)

0 => owned by host

This bit should be reset after packet reception is completed.

The host will set this bit after received data is removed.

<30:0>: These bits content includes status of receive frame (reference DM9102x data sheet).

10-2-2.RDES1: Receive buffer control and Buffer Size

<24>: Chain Enable

<10:0>: Buffer Length

10-2-3.RDES2: Buffer Starting Address indicates the **physical** address of buffer.

10-2-4.RDES3: Next Descriptor Address.

10-3. How to initialization Transmit/Receive descriptor chain structure:

First hardware and software rest, let TX/RX process are placed in the “STOP” state.

10-3-1.Read/Write suitable value for PCI configuration registers.

10-3-2.Write CR3 and CR4 to provide the starting address of TX/RX descriptor list.

10-3-3.Write CR0 to 0h.

10-3-4.Write CR7 to FFFE1841h (enable transmit and receive interrupt).

10-3-5.Write CR6 to 02042002h (start receive/transmit processes). TX/RX processes will enter the running state and attempted to acquire descriptor from the respective descriptor lists.

10-3-6.Wait for any interrupt.

10-4. The processes of Transmit and Receive data buffer: (please refer Fig. 13 &14)

Transmit Buffer Management State Transition

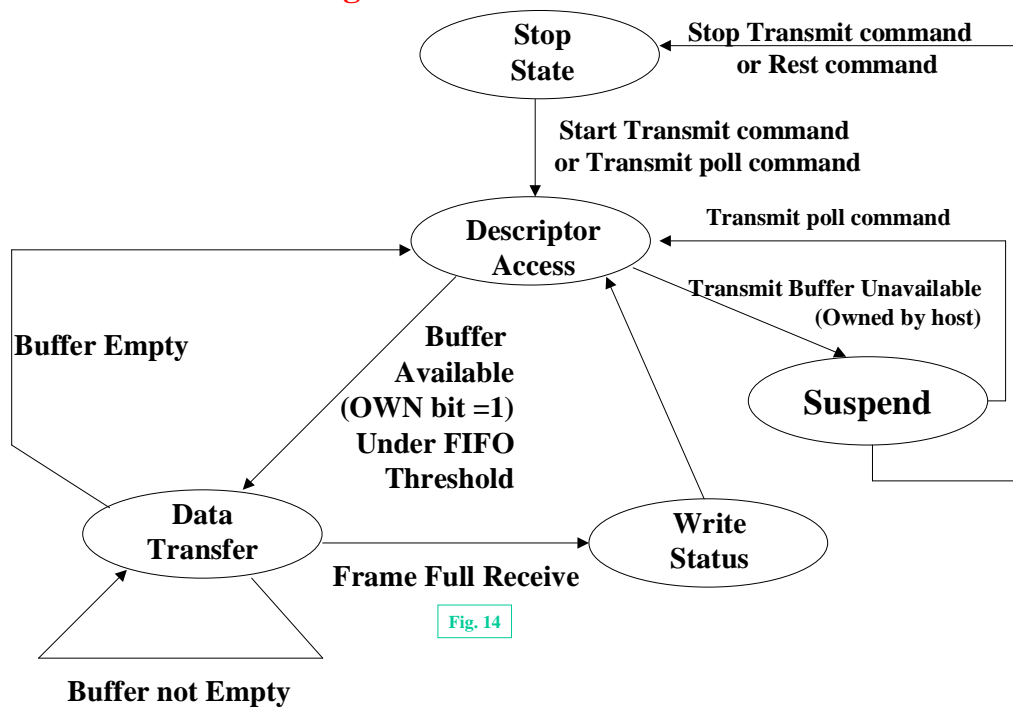
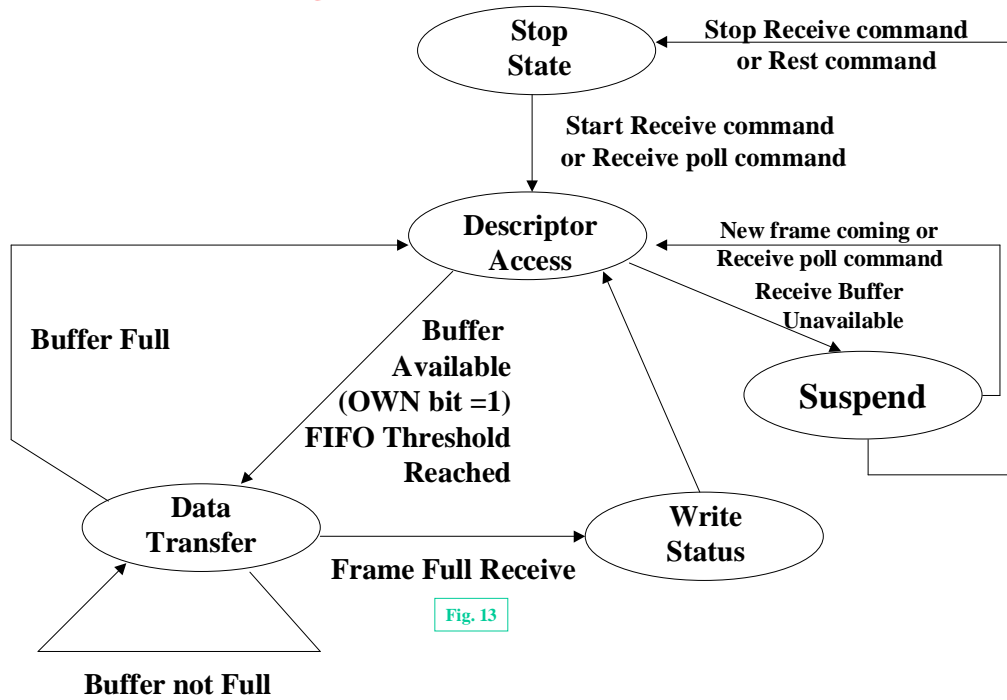


Fig. 14

Receive Buffer Management State Transition





11. New functions for DM9102D :

Only DM9102D chip support these two functions, checksum offload and zero-copy. These functions can reduce CPU utilization and increase network throughputs.

11-1. IP/TCP/UDP Checksum Offload

TCP/UDP/IP offload engine (TOE) is one of the technologies that can reduce the amount of TCP/UDP/IP processing handled by microprocessor. Let CPU have more resource to handle other things and increasing the throughput of the system.

Bit [31-27] of CR15 defines checksum offload functions. The default values of these are "0". It means disable checksum offload functions when chip power-on and reset.

Bit 31 : In transmitting, enable **IP/TCP/UDP** checksum depend-on TX descriptor control.

Bit 30 : In transmitting, enable **IP** checksum to all packets.

Bit 29 : In transmitting, enable **TCP** checksum to all packets.

Bit 28 : In transmitting, enable **UDP** checksum to all packets.

Bit 27 : In receiving, report **IP/TCP/UDP** checksum status to RDES0.

If per-packet checksum in bit 31 of CR 15 is enabled, the bit [21-19] of TDES1 (Transmit descriptor) must be set to enable the IP/TCP/UDP packet checksum generation. If user wants to enable TX checksum function, user could add the following code in user's source code.

```
cr15_data |= (1 << 30) ; //enable TX IP/TCP/UDP checksum
outl(cr15_data, ioaddr + CR15_offset);
```

In TX routine,

```
if ((skb->nh.iph->protocol) == IPPROTO_TCP )
    csum_flag=(1<<21) | (1<<20); /*TCP/IP checksum*/
else if((skb->nh.iph->protocol) == IPPROTO_UDP)
    csum_flag=(1<<21) |(1<<19); /*UDP/IP checksum*/
else
    printk("TX CHECKS_HW error");
```

```
/* Transmit Packet Process */
txptr->tdes1 = cpu_to_le32(0xe1000000|csum_flag|skb->len);
txptr->tdes0 = cpu_to_le32(0x80000000); /* Set owner bit */
outl(0x1, dev->base_addr + DCR1); /* Issue Tx polling */
```

If bit 27 of CR 15 is enabled, the bit [13-12] of RDES0 (Receive descriptor) present the IP/TCP/UDP status :



- 0X – IP checksum OK
- 1X – IP checksum FAIL
- X0 –TCP or UDP checksum OK
- X1 – TCP or UDP checksum FAIL

If user wants to enable RX checksum function, user could add the following code to user's source code.

```
cr15_data |= (1 << 27) ; //enable RX IP/TCP/UDP checksum
outl(cr15_data, iaddr + CR15_offset);
```

In RX routine,

```
if ((db->version ==PCI_DM9102D_VERSION) && !(rdes0 & 0x3000))
    skb->ip_summed = CHECKSUM_UNNECESSARY;
```

11-2. Zero Copy Supporting

Conventional TCP/IP communication incurs a high cost to copy data between kernel buffers and user process virtual memory at the socket layer. This situation has motivated development of techniques to reduce or eliminate data copying by page remapping between the user process and the kernel when size and alignment properties allow. In Linux environment, driver can use two APIs, `pci_map_signal()` and `pci_unmap_signal()`, to implement zero-copy function.

For example, kernel wants to transfer the packet and calls TX routine. Driver can use the API “`pci_map_single`” to map pointer to TDES2.

```
txptr->tdes2 = cpu_to_le32(pci_map_single(db->pdev,
skb->data, skb->len, PCI_DMA_TODEVICE));
```

Then driver can use the API “`pci_unmap_signal`” to release the buffer when completed the packet transmit.

```
pci_unmap_single(db->pdev, txptr->tdes2, (txptr->tdes1 &
0x7FF), PCI_DMA_TODEVICE);
```

In the hand of RX routine, it is also the same TX routine. When driver allocates RX buffers, driver can use the API “`pci_map_signal`” to do it and write the pointer which is API's return value to RDES2.

```
static void allocate_rx_buffer(struct dmfe_board_info *db)
{
```



DM9102 Serials Programming Guide

10/100M PCI Single Chip Ethernet NIC controller

```
struct rx_desc *rxptr;
struct sk_buff *skb;

rxptr = db->rx_insert_ptr;

while(db->rx_avail_cnt < RX_DESC_CNT) {
    if ( ( skb = dev_alloc_skb(RX_ALLOC_SIZE) ) == NULL )
        break;
    rxptr->rx_skb_ptr = skb;
    rxptr->rdes2 = cpu_to_le32( pci_map_single(db->pdev,
skb->tail, RX_ALLOC_SIZE, PCI_DMA_FROMDEVICE) );
    wmb();
    rxptr->rdes0 = cpu_to_le32(0x80000000);
    rxptr = rxptr->next_rx_desc;
    db->rx_avail_cnt++;
}

db->rx_insert_ptr = rxptr;
```

When completed packet reception, driver can the API “pci_unmap_signal” to release the RX buffer.

```
static void dmfe_rx_packet(struct DEVICE *dev, struct
dmfe_board_info * db)
{
    struct rx_desc *rxptr;
    struct sk_buff *skb;
    u32 rdes0;

    rxptr = db->rx_ready_ptr;
    while(!((rdes0=le32_to_cpu(rxptr->rdes0)) & 0x80000000)) /*
check own bit */
    {

        db->rx_avail_cnt--;
        db->interval_rx_cnt++;
    }
}
```



DM9102 Serials Programming Guide

10/100M PCI Single Chip Ethernet NIC controller

```
pci_unmap_single(db->pdev, le32_to_cpu(rxptr->rdes2),
RX_ALLOC_SIZE, PCI_DMA_FROMDEVICE);
:
:
:
}
```

```
// start of frame <01>
```

```
write(EDI, 0);
```

```
write(EECK, 0);
```

```
write(EECK, 1);
```

```
write(EDI, 1);
```

```
write(EECK, 0);
```

```
write(EECK, 1);
```