聯傑國際股份有限公司

www.axwdragon.com

DM9000A uP MAC編程指導

章節：

第一章、DM9000A 的簡介

DM9000A 爲 DAVICOM 研發之三合一之網路晶片，有下列特點：

1. 封裝採用 LQFP 48 管腳封裝，所佔用之面積和高度小。
2. 使用電壓爲 3.3V (內含一個 2.5V 的變壓器)，最大耗用電流爲 92mA，十分省電。
3. 和 MCU 連接模式有 uP 8bit / 16 bit 模式，並且支持 3.3V 和 5V 的 I/O 控制。可方便和不同電壓和界面的 MCU 連接。
4. 內置 AUTO MDI-X 功能 10/100M PHY，支持多種連接模式；電端口支持 10/100M 自適應模式 (N-WAY) 及 固定 10/100M 全半雙工模式；另提供光端口 100M 全雙工，可以用 5V 或 3.3V 光模塊 。
5. 支持 EEPROM (93C46 / 93LC46)，可供存放系統所需資訊。
6. 支持 GPIO * 6 於 8bit 模式，可供開發人員創意使用。
7. 支持快速傳送封包模式，可使 CPU 節省 CPU 資源增進 CPU 效能。
8. 支持 TCP/IP 加速功能，可使 CPU 節省 CPU 資源增進 CPU 效能。

第二章、讀、寫 DM9000A 寄存器

DM9000A 基本在操作上，就是對 DM9000A 的寄存器做讀寫的動作。而使用的管腳如下：

管腳號	管腳名稱	功能說明
35	IOR#	資料讀取管腳
36	IOW#	資料寫入管腳
37	CS#	DM9000A 使能管腳
32	CMD	控制輸出入寄存器資料或寄存器位置 接高電位時，資料管腳輸出入爲寄存器的資料端口 接低電位時，資料管腳輸出入爲寄存器的索引端口
34	INT	中斷使能管腳 DM9000A 產生中斷時，此管腳會輸出訊號。
16~18, 10~14	SD0~7	資料管腳 bit 0 ~ 7
31,29~24 ,22	SD8~15	資料管腳 bit 8 ~ 15

DM9000A 可使用 8bit / 16bit 二種模式，在 16bit 模式時 6 個 GPIO 是無法使用的，請注意。。

而 CMD 管腳功能是在設定 DM9000A 現在讀寫動作是對資料端口或是索引端口。若 CMD 爲低電位時，此時讀寫動作是對 DM9000A 索引端口動作。若 CMD 爲高電位時，此時讀寫動作是對 DM9000A 資料端口動。 uP 模式可以自定一個 GPIO 來設置，後續範例連接到 SA2。

而以上的 IOR# , IOW# , CS , INT 皆可以使用 EEPROM 來改變其使能的默認值。

IOR# 默認低使能
IOW# 默認低使能
CS# 默認低使能
INT 默認使能輸出高電`位

後面示範的方式，以 CMD 連接 SA2 ， CS# 和 IO 0x0300 ~ 0x03FFh 位置連動。

範例：

(1) 讀取 DM9000A 寄存器 0x00 的值：

```
unsigned char   reg , reg_data;
reg = 0x00;                          ;;設定所要讀取的寄存器位置到 reg 之中
outportb(0x0300 , reg);              ;;將此位置先填入 DM9000A 之中
reg_data = inportb(0x0304);          ;;將此位置的資料讀出
```

(2) 將 0x83 寫入 DM9000A 寄存器 0xFF ：

```
unsigned char   reg , reg_data;
reg = 0xff;                    ;;設定所要寫入的寄存器位置到 reg 之中
reg_data = 0x83;               ;;設定要寫入寄存器的資料
outportb(0x0300 , reg);        ;;將此位置先填入 DM9000A 之中
outportb(0x0304 , reg_data);   ;;將此位置的資料寫入
```

未來範例之中所提到的位置，皆以 IOaddr(索引端口) 和 IOdata(資料端口) 來代替 DM9000A 的 I/0 base address ，另外此一指導其 I/0 base address 以 IOaddr = 0x0300 、 IOdata = 0x0304

後面爲了減少閱讀上的困難，在未來改用下面二個 **ior , iow** 函數來減少上述許多動作。

說明：

(1)**ior** 以取代讀取 DM9000A 讀取動作：

```
unsigned char ior(unsigned char reg)
{
    outportb(IOaddr, reg);
    return (inportb(IOdata));
}
```

若以上面範例(1)來示範說明：REG_DATA = ior(0x00);

(2)**iow** 以取代寫入 DM9000A 動作：

```
void iow(unsigned char reg, unsigned char reg_data)
{
    outportb(IOaddr , reg);
    outportb(IOdata , reg_data);
}
```

若以上面範例(2)來示範說明：iow(0xff , 0x83);

第三章、DM9000A 寄存器總表

DM9000A 運用下面一些設定來控制所有的運作。

寄存器名稱	寄存器說明		位置	默認值
NCR	Network Control Register	網路界面控制	00h	00h
NSR	Network Status Register	網路界面資訊	01h	00h
TCR	TX Control Register	封包傳送控制	02h	00h
TSR I	TX Status Register I	封包傳送資訊 –1	03h	00h
TSR II	TX Status Register II	封包傳送資訊 –2	04h	00h
RCR	RX Control Register	封包接收控制	05h	00h
RSR	RX Status Register	封包接收資訊	06h	00h
ROCR	Receive Overflow Counter Register	接收溢出計數	07h	00h
BPTR	Back Pressure Threshold Register	Back Pressure 條件設置	08h	37h
FCTR	Flow Control Threshold Register	Flow Control 條件設置	09h	38h
FCR	TX/RX Flow Control Register	流量控制設置	0Ah	00h
EPCR	EEPROM & PHY Control Register	EEPROM / PHY控制	0Bh	00h
EPAR	EEPROM & PHY Address Register	EEPROM / PHY讀寫位置	0Ch	40h
EPDRL	EEPROM & PHY Low Byte Data Register	EEPROM / PHY資料 -L	0Dh	XXh
EPDRH	EEPROM & PHY High Byte Data Register	EEPROM / PHY資料 -H	0Eh	XXh
WCR	Wake Up Control Register	喚醒控制	0Fh	00h
PAR	Note Address Register	Note 位置設置	10h 11h 12h 13h 14h 15h	XXh by EEPROM
MAR	Multicast Address Register	Multicast 設置	16h 17h 18h 19h 1Ah 1Bh 1Ch 1Dh	XXh XXh XXh XXh XXh XXh XXh XXh
GPCR	General Purpose Control Register	GPIO 界面控制	1Eh	01h
GPR	General Purpose Register	GPIO 界面資訊	1Fh	XXh
VID	Vendor ID	廠商 ID 號	28h 29h	XXh [46h] XXh [0Ah]
PID	Product ID	產品 ID 號	2Ah 2Bh	XXh [00h] XXh [90h]

CHIPR	CHIP Revision	IC 版本號	2Ch	18h
TCR2	TX Control Register 2	封包傳送控制 -2	2Dh	00h
ETXCSR	Early Transmit Control / Status Register	快速傳送封包設置	30h	00h
TCSCR	Transmit Check Sum Control Register	TCP/IP 傳送檢驗核自動計算設置	31h	00h
RCSCSR	Receive Check Sum Control Status Register	TCP/IP 接收檢驗核自動比對設置	32h	00h
MRCMDX	Memory Data Read Command Without Address Increment Register	內存讀取控制，不移內存動讀取位置	F0h	XXh
MRCMD	Memory Data Read Command With Address Increment Register	內存讀取控制，移動內存讀取位置	F2h	XXh
MRRL	Memory Data Read_ address Register Low Byte	內存讀取位置 –L	F4h	00h
MRRH	Memory Data Read_ address Register High Byte	內存讀取位置 -H	F5h	00h
MWCMDX	Memory Data Write Command Without Address Increment Register	內存寫入控制，不移動內存寫入位置	F6h	XXh
MWCMD	Memory Data Write Command With Address Increment Register	內存寫入控制，移動內存寫入位置	F8h	XXh
MWRL	Memory Data Write_ address Register Low Byte	內存寫入位置 –L	FAh	00h
MWRH	Memory Data Write _ address Register High Byte	內存寫入位置 -H	FBh	00h
TXPLL	TX Packet Length Low Byte Register	傳送封包大小設置 -L	FCh	XXh
TXPLH	TX Packet Length High Byte Register	傳送封包大小設置 -H	FDh	XXh
ISR	Interrupt Status Register	中斷資訊設置	FEh	00h
IMR	Interrupt Mask Register	中斷條件設置	FFh	00h

在後續文章之中，會使用寄存器名稱來代替寄存器位置，以增加閱讀上的方便。

基本上，DM9000A 寄存器可以分爲下列幾大類：

基本控制區 →

網路流量控制區 →

EEPROM / PHY 控制區 →

WOL 控制區 →

PAR , MAR 設置區 →

GPIO 控制區 →

封包資料接收區 →

封包資料傳送區 →

中斷設置區 →

第四章、DM9000A 的初始化

DM9000A 在正常工作之前需要做一些相關的設定，才能正常動作，建議的順序如下：

1. DM9000A 軟件重置
2. 清除中斷設定，以免因中斷導致 DM9000A 初始化動作不正常
3. 設定 DM9000A 相關連接界面
4. 設定 Note Address 位置
5. 設定 Multicast 設置
6. 其他相關設置
7. 開啓接收資料功能

現我們分別就上述細詳說明：

1.DM9000A 軟件重置

```
iow(NCR , 0x03);          ;;將 DM9000A 進行軟件重置
dellay(20);               ;;等待 DM9000A 重置完成
iow(NCR , 0x00);          ;;將 DM9000A 回復正常工作狀態
iow(NCR , 0x03);          ;;將 DM9000A 進行軟件重置
dellay(20);               ;;等待 DM9000A 重置完成
iow(NCR , 0x00);          ;;將 DM9000A 回復正常工作狀態
```

上面軟件重置動作有重複二次，以確保 DM9000A 重置成功。

2.清除中斷設定，以免因中斷導致 DM9000A 初始化動作不正常

```
iow(0xff , 0x80);         ;;將 DM9000A 中斷功能關閉
```

3.設定 DM9000A 相關連接界面

```
iow(0x2df , 0x80);        ;;將 LED 設置在 LED1 MODE
gpio_set(0x00 , 0x01);    ;; GPIO 和 內部 PHY 做相連控制(參考 5 章)
gpio_w(0x00 , 0x00);      ;;開啓內部的 PHY 電源(參考 5 章)
phy_w(0x00 , 0x8000);     ;;將 PHY 重置(參考 6 章)
```

4.設定 Note Address 位置

mac_set(); ;;設定 Note Address (參考 8 章)

5.設定 Multicast 設置

unsigned char broadcast[6] = {0xff , 0xff , 0xff , 0xff , 0xff , 0xff};

multicast_set(0x01, broadcast); ;;設定 multicast (參考 9 章)

6.其他相關設置(請參考 n 章寄存器說明)

ior(0x01); ;;清除 NSR 相關資訊

iow(0x02 , 0x00); ;;清除 TCR 相關資訊(參考 13 章)

ior(0x07); ;;清除 ROCR 相關資訊(參考 14 章)

iow(0x0a ,0x28); ;;設置 FCR 起動流控功能(參考 14 章)

phy_w(0x04 , 0x05e1); ;;將 PHY 連接模式及流控(參考 14 章)

iow(0xfe , 0xff); ;;清除 ISR 相關資訊

iow(0xff , 0x83); ;;設置 IMR 功能

7.開啓接收資料功能

iow(0x05 , 0x31); ;;開啓接收功能(參考 13 章)

第五章、GPIO 設置及使用

DM9000A 提供 6 個 GPIO 口，只能在 8 bit 模式可以使用。使用 GPIO 其操作非常簡單，只需設置 GPCR , GPR 這二個寄存器即可。

GPCR 這個寄存器可設置這 GPIO 此一工作時間時爲輸出或輸入端口。,若此 bit 設置 1 GPIO 端口爲輸出端口、設置 0 GPIO 端口爲輸入端口。而 GP4、GP5、GP6 這 3 個 pin 只有支持輸出端口功能，並且輸入端口功能。

GPR 會依 GPCR 寄存器設置，而有不同的動作。 GPR 和 GPCR 一樣控制。若 GPCR 設置爲 1 ，GPR 設置爲 1 輸出高電位、設置爲 0 輸出低電位。若 GPCR 爲 0 。 GPR 此時爲輸入資訊 1 爲輸入高電位、 0 爲輸入低電位。

特別注意， GPR bit 0 (PHYPD) 和 內部 PHY 電源是連動的，若是將 PHYPD 設置爲 0 開啓內部 PHY 的電源，若設置爲 1 則關閉內部 PHY 電源。

範例：

(1) 將 GP2 設置爲輸出端口、GP3 設置成輸入端口

```
iow(GPCR , 0x04);                    ;;將 GP2 設成輸出口，GP3 設置成輸入口
```

(2) 將 GP2 輸出高電位

```
iow(GPR , 0x04);                     ;;使 GPIO 輸出高電位
```

(3) 從 GP3 輸入資訊

```
GP_3 = (ior(GPR) >> 3) & 0x01;       ;;從 GPIO 得入高低電位
```

我們可以使用下面函數來使 GPIO 使用更方便。

(1)**gpio_set** 以取代 GPIO 設定動作：

```
void gpio_set(unsigned char s_gpio , unsigned char gp_io)
{
  if (gp_io == 0x01)
    iow(GPCR , ior(GPCR) & ~(0x01 << s_gpio));
  else
    iow(GPCR , ior(PGCR) | (0x01 << s_gpio));
}
```

若以上面範例(1)來示範說明：

```
gpio_set(0x02 , 0x01);
```

gpio_set(0x03 , 0x00);

(2)**gpio_w** 以取代 GPIO 輸出控制動作：

```
void gpio_w(unsigned char s_gpio , unsigned char gp_hl)
{
  if (gp_hl == 0x01)
    iow(GPR , ior(GPR) | (0x01 << s_gpio));
  else
    iow(GPR , ior(GPR) & ~(0x01 << s_gpio));
}
```

若以上面範例(2)來示範說明：

gpio_w(0x02 , 0x01);

(3)**gpio_r** 以取代 GPIO 輸入動作：

```
unsigned char gpio_r(unsigned char s_gpio)
{
 retrun( (ior(GPR)  >>  s_gpio) & 0x01);
}
```

若以上面範例(2)來示範說明：

GP_3 = gpio_r(0x03);

第六章、設定 PHY 寄存器值

DM9000A 內部有一高效能 AUTO MDI-X 10/100M 的 PHY，其支援模式如下： 10M HALF , 10M FULL , 100M HALF , 100M FULL , 自適應 (N-WAY) , FIBER MODE。相互連接使用。而讀、寫 PHY 寄存器的方式十分簡單。此相關設置的寄存器如下：

EPCR (設定 PHY 動作)
EPAR (讀取或寫入 PHY 的位置，DM9000A 所支援的 PHY 寄存器最大值到 32 個。；所以使用 bit 4 ~ 0 來設定現在是選擇這 32 個的那一個。 DM9000A 內置的 PHY 位置必需設爲 01)
EPDRL (爲讀取或寫入 EPAR 所設定 PHY 寄存器資料的 Low Byte。)
EPDRH (爲讀取或寫入 EPAR 所設定 PHY 寄存器資料的 High Byte。)

請注意 DM9000A 在讀寫 PHY 寄存器時，是以 word 的方式讀寫。

故要將相對寄存器的資料寫入 PHY 寄存器時，必需將其值放入 EPDRL / EPDRH 之中。

例如：將 0x3100 寫入 PHY 寄存器之中！

則將 0x31 寫入 EPDRH , 0x00 寫入 EPDRL 之中

而讀取回 PHY 寄存器的值時，其值也會按 word 的方式放在 EPDRL / EPDRH 之中。

例如：讀從 EPDRH 取得值爲 0x01, EPDRL 取得值爲 0xe1

則從 PHY 寄存器取回的值爲 0x01e1

1. 如何從 PHY 讀取 PHY 寄存器值。

要讀取 PHY 寄存器步驟如下：

(1)要讀取 PHY 寄存器位置放到 EPAR bit 0 ~ 4，並將要讀取的 PHY 位置放到 EPARD

(2)將 EPCR 設爲 0x0C ，使 DM9000A 去讀取 PHY 資料

EPCR bit3 = 1，將模式設定 PHY

EPCR bit2 = 1，開行進行 PHY 寄存器讀取動作

(3) 等待最少 1~200 us 或重覆讀取 EPCR ，並等待 EPCR bit 0 等於 0 爲止

(4)將 EPCR 讀取 PHY 動作清除

(5)此時 EPAR 所選取 PHY 寄存器的 Low Byte 置於 EPDRL、 High Byte 置於 EPDRH.

範例：讀取內置 PHY 第 4 個值寄存器。.

```
unsigned int P_data;
iow(EPAR , 0x04 | 0x40);              ;;將要讀取寄存位設置於 EPAR
iow(EPCR , 0x0c);                     ;;開始進行讀取
delay(20);                            ;;減少下面指令對 BUS 動作
while((ior(EPCR) & 0x01) == 0x01);    ;;反覆確定 DM9000A 是否完成
iow(EPCR , 0x00);                     ;;讀取結束，回復爲正常模式
P_data = (ior(EPDRH) << 8) | ior(EPDRL);   ;;取回讀取之後的資料
```

爲了後續讀取上的方便，使用下面函數：

```
unsigned int phy_r( unsigned char phy_offset)
{
  iow(EPAR , phy_offset | 0x40);      ;;將要讀取寄存位設置於 EPAR
  iow(EPCR , 0x0c);                   ;;開始進行讀取
  delay(20);                          ;;減少下面指令對 BUS 動作
  while((ior(EPCR) & 0x01) == 0x01);  ;;反覆確定 DM9000A 是否完成
  iow(EPCR , 0x00);                   ;;讀取結束，回復爲正常模式
  return( (ior(EPDRH) << 8) | ior(EPDRL));  ;;取回讀取之後的資料
}
```

而以上面的範例，使用下面命令：

```
P_data = phy_r(0x04);
```

2.如何將資料寫入至 PHY 寄存器

要寫入 PHY 寄存器步驟如下：

(1) 要讀取 PHY 寄存器位置放到 EPAR bit 0 ~ 4，並將要讀取的 PHY 位置放到 EPARD
(2)將要寫入資料的 Low Byte 置於 EPDRL、 High Byte 置於 EPDRH.
(3)將 EPCR 設爲 0x0A ，使 DM9000A 去寫入 PHY 資料
EPCR bit3 = 1，將模式設定 PHY
EPCR bit1 = 1，開行進行 PHY 寄存器寫入動作
(4)等待最少 1~200 us 或重覆讀取 EPCR ，並等待 EPCR bit 0 等於 0 爲止
(5)將 EPCR 寫入 PHY 動作清除

範例：將 0x2100 寫入 PHY 寄存器第 0 個。.

```
iow(EPAR , 0x00 | 0x40);              ;;將要寫入寄存位設置於 EPAR
iow(EPDRH , 0x21);                    ;;資料的 High Byte 置入 EPDRH
iow(EPDRL , 0x00);                    ;;資料的 Low Byte 置入 EPDRL
iow(EPCR , 0x0a);                     ;;開始進行寫入
delay(150);                           ;;減少下面指令對 BUS 動作
while((ior(EPCR) & 0x01) == 0x01);    ;;反覆確定 DM9000A 是否完成
iow(EPCR , 0x00);                     ;;寫入完成，回復正常模式
```

爲了後續讀取上的方便，使用下面函數：

```
void phy_w (unsigned char phy_offset , unsigned int reg_data)
{
  iow(EPAR , phy_offset | 0x40);      ;;將要寫入寄存位設置於 EPAR
  iow(EPDRH , reg_data >> 8);         ;;資料的 High Byte 置入 EPDRH
  iow(EPDRL , reg_data & 0xff);       ;;資料的 Low Byte 置入 EPDRL
  iow(EPCR , 0x0a);                   ;;開始進行寫入
  delay(150);                         ;;減少下面指令對 BUS 動作
  while((ior(EPCR) & 0x01) == 0x01);  ;;反覆確定 DM9000A 是否完成
  iow(EPCR , 0x00);                   ;;寫入完成，回復正常模式
}
```

而以上面的範例，使用下面命令：

```
phy_w( 0x00 , 0x2100);
```

3.設置 DM9000A 不同的連接模式

DM9000A 內置的 PHY 提供 5 種不同的連接模式 10M 半雙工 / 10M 全雙工/ 100M 半雙工/ 100M 全雙工/ 自適應(N-WAY)，可以依照應用面需求來設置。

最主要設定的地方，是運用到 PHY 寄存器 0 和 4 ，PHY 寄存器 4 需要在 PHY 寄存器 0 bit 12 (自適應啓動)有設置才有使用。所以 PHY 寄存器 0 基本上有下列連接模式：

PHY 寄存器 0

0x0000	*phy_w(0x00 , 0x0000);*	固定 10M 半雙工
0x0100	*phy_w(0x00 , 0x0100);*	固定 10M 全雙工
0x2000	*phy_w(0x00 , 0x2000);*	固定 100M 半雙工
0x2100	*phy_w(0x00 , 0x2100);*	固定 100M 全雙工
0x1000	*phy_w(0x00 , 0x1000);*	自適應模式，需參考 PHY 寄存器 4

PHY 寄存器 4

Bit 5	支援 10M Half	設置之後可以支持 10M 半雙工
Bit 6	支援 10M Full	設置之後可以支持 10M 全雙工
Bit 7	支援 100M Half	設置之後可以支持 100M 半雙工
Bit 8	支援 100M Hull	設置之後可以支持 100M 全雙工
Bit 10	支援全工模式之流量控制	設置之後可以支持全工模式之流量控制
例如：設置成 10M / 100M 半雙工、全雙工皆支援，並且支援全工之流量		
0x05e1	*phy_w(0x04 , 0x05E1);*	

建議切換 PHY 連接模式範例：

(1) 切換到 100M Full 模式

```
gpio_set(0x00 , 0x01);          ;;將內部的 PHY 電源關閉
phy_w( 0x00 , 0x2100);          ;;切換 PHY 連接模式
gpio_set(0x00 , 0x00);          ;;將內部的 PHY 電源開啓
```

(2) 切換到 10/100M 自適應模式

```
gpio_set(0x00 , 0x01);          ;;將內部的 PHY 電源關閉
phy_w(0x04 , 0x05e1);           ;;將自適應模式及流控資料填入
phy_w(0x00 , 0x1000);           ;;切換 PHY 到自適應模式
gpio_set(0x00 , 0x00);          ;;將內部的 PHY 電源開啓
```

4.如何檢查 PHY 連接狀況

PHY 的連接狀況，可以使用 NSR bit 6 來監看，其 LINK 狀況。

```
LINK_STATUS = (ior( NSR ) >> 6) & 0x01;   ;;爲 1 爲 link 正常，0 爲 link 失敗
```

5.MAC 如何依 PHY 連接狀況設定工作模式

MAC 可依 PHY 的連接狀況，設定成全雙工和半雙工二種工作方式。使用內部的 PHY 時 PHY 寄存器和 DM9000A NCR bit 3 是連動的。所以不用另外處理。只要內部的 PHY 連接到全雙工模式或半雙工模式。會自動改變 DM9000A NCR bit 3 的值。

第七章、使用 EEPROM 資料

DM9000A 支援 EEPROM 爲 93C46 可以運用 DM9000A pin 19 , 20 , 21 這三個管腳和下列寄存器非常簡單的方式來讀取或寫入使用其資料。 DM9000A 和 EEPROM 相關的寄存器如下：

EPCR (設定 EEPROM 動作)
EPAR (讀取或寫入 EEPROM 的位置，DM9000A 可使用 93C46 這一款的 EEPROM ，其內容是 64 個 Word 資料(128 Byte)；使用 bit 5 ~ 0 來設定選擇這 64 個 word 的那一個。而 DM9000A 所需使用 0 ~ 7 共 8 個 word。8 ~ 63 word 爲提供給使用者自行使用。)
EPDRL (爲讀取或寫入 EPAR 所設定 word 資料的 Low Byte。)
EPDRH (爲讀取或寫入 EPAR 所設定 word 資料的 High Byte。)

這兒的使用的寄存器和 PHY 的寄存器是一樣的，DM9000A 在這二個功能上寄存器是共用的。

請特別注意，DM9000A 在讀寫 EEPROM 以 word 方式並且以 big-endian (大字節結尾) 方式來進行讀寫資料，此時 EPDRL 皆會存取偶數的 Byte ， EPDRH 皆會存取奇數的 Byte 。這一地方和讀寫 PHY 的方式有些不一樣。

故要將相對資料寫入 EEPROM ， 需將其值放入 EPDRL / EPDRH 之中。

例如：將值 0x5522 寫入 EEPROM 之中！

則將 0x55 寫入 EPDRH (奇數字節), 0x22 寫入 EPDRL 之中(偶數字節)

而讀取 EEPROM 值時，其值也會按 word 的方式放在 EPDRL / EPDRH 之中。

例如：讀從 EPDRH (奇數字節) 取得值爲 0x33, EPDRL (偶數字節) 取得值爲 0x44

則從 EEPROM 取回的值爲 0x3344

DM9000A 存放在 EEPROM 中的資料有：Note Address、Auto Load Control、Vendor ID、Product ID、Pin Control、Wake-up Mode Control。可以依系統相關設定來調配 (請參考附件格式) 。

1. 如何從 EEPROM 讀取資料：

EEPROM 讀取資料的步驟如下：

(1)將要讀取的 EPPROM word 位置先到 EPAR

(2)將 EPCR 設爲 0x04 ，使 DM9000A 去讀取 EEPROM 資料：

EPCR bit3 設爲 0，將模式設定爲 EEPROM

EPCR bit2 設爲 1，使 DM9000A 進行讀取 EEPROM 資料

(3)重待最少 20~500 us 或重覆讀取 EPCR ，至到 EPCR bit0 等於 0 爲止

(4)將 EPCR 讀取 EEPROM 動作清除。

(5)此時 EPAR 所選取 WORD 的 Low Byte 置於 EPDRL、High Byte 置於 EPDRH

範例：讀取 EEPROM 第 2 個 WORD 的值。

```
unsigned int E_data;
iow(EPAR , 0x02);                    ;;將要讀取寄存位設置於 EPAR
iow(EPCR , 0X04);                    ;;開始進行讀取
delay(150);                          ;;減少下面指令對 BUS 動作
while((ior(EPCR) & 0x01) == 0x01);   ;;反覆確定 DM9000A 是否完成
iow(EPCR , 0x00);                    ;;讀取結束，回復爲正常模式
E_data = (ior(EPDRH) << 8) | ior(EPDRL);  ;;取回讀取之後的資料
```

爲了後續讀取上的方便，使用下面函數：

```
unsigned int eeprom_r(unsigned char rom_offset)
{
  iow(EPAR , rom_offset);            ;;將要讀取寄存位設置於 EPAR
  iow(EPCR , 0x04);                  ;;開始進行讀取
  delay(150);                        ;;減少下面指令對 BUS 動作
  while((ior(EPCR) & 0x01) == 0x01); ;;反覆確定 DM9000A 是否完成
  iow(EPCR , 0x00);                  ;;讀取結束，回復爲正常模式
  return( (ior(EPDRH) << 8) | ior(EPDRL));  ;;取回讀取之後的資料
}
```

而以上面的範例，使用下面命令：

```
E_data = eeprom_r(0x02);
```

2. 如何寫入資料至 EEPROM

寫入資料至 EEPROM 的步驟如下：

(1)將要寫入的 EPPROM word 位置先到 EPAR.

(2)將要寫入 EPAR 所選取 WORD 的 Low byte 置於 EPDRL、High Byte 置於 EPDRH

(3)將 EPCR 設爲 0x12 ，使 DM9000A 進行寫入 EEPROM 資料

EPCR bit4 設爲 1，將 EEPROM 寫入模式啓動

EPCR bit3 設爲 0，將模式設定爲 EEPROM

EPCR bit1 設爲 1，使 DM9000A 進行寫入 EEPROM 資料

(4)等待最少 20~500 us 或重覆讀取 EPCR ，至到 EPCR bit0 等於 0 爲止

(5)將 EPCR 寫入 EEPROM 動作清除

範例：寫入 EEPROM 第 3 個 WORD 的值，存入 0xaa55 。

```
iow(EPAR , 0x03);                    ;;將要寫入寄存位設置於 EPAR
iow(EPDRH , 0xaa);                   ;;資料的 High Byte 置入 EPDRH
iow(EPDRL , 0x55);                   ;;資料的 Low Byte 置入 EPDRL
iow(EPCR , 0x12);                    ;;開始進行寫入
delay(150);                          ;;減少下面指令對 BUS 動作
while((ior(EPCR) & 0x01) == 0x01);   ;;反覆確定 DM9000A 是否完成
iow(EPCR , 0x00);                    ;;寫入完成，回復正常模式
```

爲了後續讀取上的方便，使用下面函數：

```
void eeprom_w(unsigned char rom_offset , unsigned int rom_data)
{
  iow(EPAR , phy_offset);            ;;將要寫入寄存位設置於 EPAR
  iow(EPDRH , reg_data >> 8);        ;;資料的 High Byte 置入 EPDRH
  iow(EPDRL , reg_data & 0xff);      ;;資料的 Low Byte 置入 EPDRL
  iow(EPCR , 0x12);                  ;;開始進行寫入
  delay(150);                        ;;減少下面指令對 BUS 動作
  while((ior(EPCR) & 0x01) == 0x01); ;;反覆確定 DM9000A 是否完成
  iow(EPCR , 0x00);                  ;;寫入完成，回復正常模式
}
```

而以上面的範例，使用下面命令：

```
eeprom_w(0x03 , 0xaa55);
```

DM9000A EEPROM 格式

名稱	Word	Byte 位置	說明
Note address	0	0 ~ 5	6 Byte 網路設備 Ethernet Note Address
Auto Load Control	3	6 ~ 7	Bit 0 = 1 : 使用 EEPROM 設定 Vendor ID 和 Product ID Bit 2 = 1 : 起動 uP control (WORD 6) bit 0 ~ 4 選項 Bit 6 = 1 : 起動 Wake-Up mode control (WORD 7) bit 0 ~ 3 選項 (此設定只有在 8 bit 模式下有效) Bit 10 = 1 : 起動 Wake-Up mode control (WORD 7) bit 7 選項 Bit 12 = 1 : 起動 Wake-Up mode control (WORD 7) bit 8 選項 Bit 14 = 1 : 起動 Wake-Up mode control (WORD 7) bit 14~15 選項 Bit 1 , 3 , 4 , 5, 7 , 8 , 9 , 11 , 13 , 15 ：系統保留，請設爲 0 。
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	Bit 0 : CS# (PIN 37) 起動時爲高電位或低電位 (默認爲低電位) = 1 起動時爲低電位 = 0 起動時爲高電位 Bit 1 : IOR# (PIN 35) 起動時爲高電位或低電位 (默認爲低電位) = 1 起動時爲低電位 = 0 起動時爲高電位 Bit 2 : IOW# (PIN 36) 起動時爲高電位或低電位 (默認爲低電位) = 1 起動時爲低電位 = 0 起動時爲高電位 Bit 3 : INT (PIN 34) 起動時爲高電位或低電位 (默認爲高電位) = 1 起動時爲低電位 = 0 起動時爲高電位 Bit 4 : INT (PIN 34) 起動時爲瞬間輸出或持續輸出 (默認爲持續) = 1 起動時瞬間輸出 (open-collected) = 0 起動時持續輸出 (force output) Bit 5 ~ 8 ：系統保留，請設爲 1 。 Bit 9 ~ 15 ：系統保留，請設爲 0 。
Wake-Up mode control	7	14 ~ 15	Bit 0 : WAKEUP (PIN 22) 起動時爲高電位或低電位 (默認爲高電位) = 1 起動時爲低電位 = 0 起動時爲高電位 Bit 1 : WAKEUP (PIN 22) 輸出爲 Pulse 或 Level (默認爲 Level) = 1 起動時爲 Pulse = 0 起動時爲 Level Bit 2 : 支持 Magic Packet Wakeup 喚醒模式 (默認爲不支持) = 1 設定支持 = 0 設定不支持 Bit 3 : 支持 Link Chang Wakeup 喚醒模式 (默認爲不支持) = 1 設定支持 = 0 設定不支持 (以上設定只有在 8 bit 模式下有效) Bit 7 : LED 顯示模式 (默認爲 LED Mode 0) = 1 LED Mode 1 = 0 LED Mode 0 Bit 8 : 內置的 PHY 啓動 (默認爲不啓動) = 1 啓動 = 0 不啓動 Bit 14 : 內置的 PHY AUTO MDI-X 功能啓動 (默認爲啓動) = 1 啓動 = 0 不啓動

			Bit 15 : 將 LED1 變成 IO16 管腳使用 (在 16 bit 模式有效，默認不啓動) = 1 啓動 = 0 不啓動 Bit 4 , 5 , 6 , 9 , 10 , 11 , 12 , 13 : 系統保留，請設爲 0 。
USE-ABLE	8~63	16 ~ 127	提供使用者自行使用

建議 EEPROM 設置值：(前 6 組 XX 爲該設備的 Note Address)

“XX XX XX XX XX XX 45 54 46 0A 00 90 E7 01 80 41 “

若以 eeprom_r 命令來讀取資料，結果如下：

```
Printf("\n eeprom addr 0x03 == %04x " , eeprom_r(0x03));
Printf("\n eeprom addr 0x04 == %04x " , eeprom_r(0x04));
Printf("\n eeprom addr 0x05 == %04x " , eeprom_r(0x05));
Printf("\n eeprom addr 0x06 == %04x " , eeprom_r(0x06));
Printf("\n eeprom addr 0x07 == %04x " , eeprom_r(0x07));

eeprom addr 0x03 == 5445
eeprom addr 0x04 == 0a46
eeprom addr 0x05 == 9000
eeprom addr 0x06 == 01e7
eeprom addr 0x07 == 4180
```

第八章、如何設定 Note Address 位置

在一個標準的 IEEE 802.x 封包之中，需要一個專屬獨一無二的位置；而其位置稱爲 Note Address。而設置 Note Address 才知封包要送給那一網絡設備，並且才知此一封包是否是送給 DM9000A 的封包。

那要如何設置 Note Address ，可以用下面二種方法來指定：

(1)使用 EEPROM 來指定，DM9000A 在電源重置時，會自動連接下載 EEPROM 前 8 個 WORD.。做一些基本的設定。EEPROM 前 3 個 WORD 可提供設置 Note Address ，並且自動載入到 PAR (寄存器 0x10 ~ 0x15)。

(2)若不想使用或未連接 EEPROM 時，可以自行指定其 PAR 資料。

注意 DM9000A PAR 值，儘供接收處理使用。在傳送資料時，需程序自行將 Note Address 填入封包之中。

範例：將 Note Address 位置設置成 “ 00:60:6e:00:90:00” 。

```
iow(PAR , 0x00);
iow(PAR + 1 , 0x60);
iow(PAR + 2 , 0x6e);
iow(PAR + 3 , 0x00);
iow(PAR + 4 , 0x90);
iow(PAR + 5 , 0x00);
```


我們也可以用函數來將程式變得更有可讀性：

```
void mac_set(void)
   {
      iow(PAR , 0x00);
      iow(PAR + 1 , 0x60);
      iow(PAR + 2 , 0x6e);
      iow(PAR + 3 , 0x00);
      iow(PAR + 4 , 0x90);
      iow(PAR + 5 , 0x00);
   }
```

第九章、如何設定 Multicast 設置

在多許應用環境之中，可能需要同時發送給不同的網路設備。這種特殊的功能，稱爲多播功能 (Multicast)，而 DM9000A 可以選擇接收或不接收其封包。以減輕系統資源的消耗。

而 Multicast 位置，在 IEEE 802.3 規定上爲 Note Address 第一個 byte 最小的 bit 若爲 1 就是 Multicast 位置，其大小共有 137,470,998,445,313 個。而 DM9000A 運用 Hash Table 方式，將之分爲 64 組。以增進處理的速度。請注意在 RCR bit 1 需設 0 , bit 3 需設 0 才能使用。

那要如何設置 Multicast，前面有提到此一功能需要用到 Hash Table 來做運算，DM9000A 內部硬件已設計會自行做算運檢查。而設置的方法就是將所要設置 Multicast 位置，使用 CRC-32 方法來計算，並將運算完後的 CRC 檢核碼最小 byte 和 0x3f 做一次 and 的運算。求出來的值就是 Hash Table 的值。

下面的程序，是示範如何設定使用 Hash Table 將 Broadcast 設置。

```
unsigned long int crc ,carry ;
unsigned int i , j;
unsigned char mac[6] = { 0xff , 0xff , 0xff , 0xff , 0xff , 0xff};
unsigned char Hash_data , iTemp;

crc = 0xffffffff;
for(i = 0 ; i < 6 ; i++)
  {
    carry = (crc ^ mac[i]) & 0xff;
    for(j = 0 ; j < 8 ; j++)
                                {
      if(carry & 1)
        carry = (carry >> 1) ^ 0xedb88320;
      else
        carry >>= 1;
    }
    crc = ((crc >> 8) & 0x00ffffff) ^ carry;
  }

Hash_data = crc & 0x3f;
iTemp = ior(MAR + (Hash_data /8));
iow(MAR + (Hash_data /8) , 0x01 << (Hash_data % 8));
```

我們也可以用函數來將程式變得更有可讀性：

```
void multicast_set(unsigned char Add_Del , unsigned char *set_mac)
    {
      unsigned long int crc ,carry ;
      unsigned int i , j;
      unsigned char Hash_data , iTemp;

      crc = 0xffffffff;
      for(i = 0 ; i < 6 ; i++)
        {
          carry = (crc ^ *set_mac++) & 0xff;
          for(j = 0 ; j < 8 ; j++)
           {
             if(carry & 1)
               carry = (carry >> 1) ^ 0xedb88320;
             else
               carry >>= 1;
           }
           crc = ((crc >> 8) & 0x00ffffff) ^ carry;
        }

     Hash_data = crc & 0x3f;
     iTemp = ior(MAR + (Hash_data / 8));

     if (Add_Del == 0x01)
       iow(MAR + (Hash_data / 8) ,iTemp | (0x01 << (Hash_data % 8)));
     else
       iow(MAR + (Hash_data / 8) , iTemp & ~(0x01 << (Hash_data %8)));
    }
```

而以上面的範例，使用下面命令：

```
unsigned char broadcast[6] = {0xff , 0xff , 0xff , 0xff , 0xff , 0xff};
multicast_set(0x01, broadcast);
```

第十章、檢查現在所使用的 I/O 模式

DM9000A 支持 2 種 I/O 模式，分別爲 Byte / Word ，在讀寫資料的方式也支援 8-bit / 16-bit ，在硬件和軟件上需要相互配合(即設置模式需一致)。

在程序中要如何去確認現在是使用何種模式呢！可以使用 ISR 來查看：

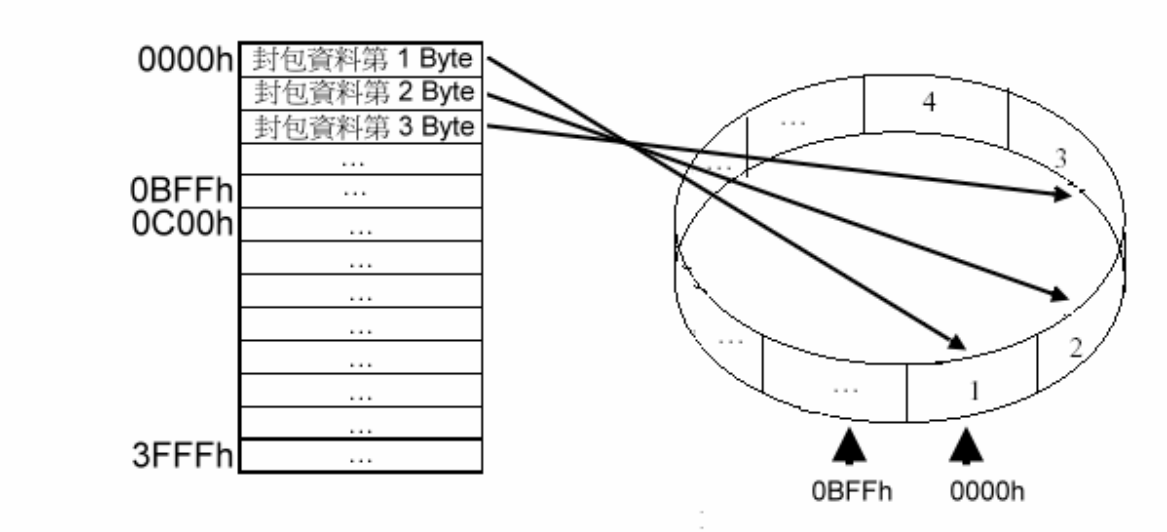
unsigned char IO_chk;

IO_chk = ior(ISR) >> 7; **;; IO_chk 若爲 0x01 工作模式爲 8 bit 模式**
;; IO_chk 若爲 0x00 工作模式爲 16 bit 模式

後面會用 ***DM9000A_E.Work_mode*** *來取代 IO_chk 這個參數*

第十一章、如何傳送封包

在傳送個封包之前,需將其封包資料存放在 DM9000A 的傳送內存中 0000h~0BFFh 。若是寫入位置超 過 0BFFh 時，DM9000A 會自動將位置移到 0000h 的位置。



將封包資料存放在 MWCMD 中，DM9000A 會自動將其資料存向其傳送內存中。另外還需將要傳送封包的大小存放在 TXPLL(low_byte) 和 TXPLH(high_byte)。之後再將 TCR bit 0 設爲 1 ，此時開始進行封包的傳送。而在傳送完成後，會將傳送是否成功的資訊放在 TSRI , TSRII 中。放的順序爲 TSRI -> TSRII -> TSRI -> TSRII …。所以需要依照 NSR bit 2 ~ 3 來判斷現在是 TSRI 或 TSRII 傳送完成。

如何傳送一個封包流程：

(1)檢查現在是使用那一種 IO 動作 (Byte , Word)，請參考 9 章。

在未來將資料搬移的作動，皆要以此模式進行，並且此一模式，需和硬體一致。

(2)將封包長度存入至 TXPLL 和 TXPLH 中：

```
iow(TXPLL , TX_LEN  &  0xff );          ;;將要傳送的長度 Low Byte 填入 TXPLL 中
iow(TXPLH, (TX_LEN  >> 8)  &  0xff );   ;;將要傳送的長度 High Byte 填入 TXPLH 中
```

(3)開始將封包的資料搬移到傳送內存中：(這兒不使用 iow 改使用 outportb、outport 這樣速度可以增快很多)

```
                                        ;;TX_DATA[TX_LEN]： 傳送的資料
                                        ;;TX_LEN：傳送資料長度以 Byte 計算

unsigned char     *tx_data8;            ;;設定 8 bit 資料指針
unsigned int      *tx_data16;           ;;設定 16 bit 資料指針

outportb( IOaddr, MWCMD );              ;;將位置指向 MWCMD
I / O Byte Mode:
  tx_data8 = TX_DATA;                   ;;將指針指向傳送資料
  for (i = 0; i < TX_LENGTH; i++)       ;;將 TX_DATA 中的資料移到傳送內存中
    outportb( IOdata, *(tx_data8++));   ;;每次一個 Byte

I / O Word Mode:
  tx_data16 = TX_DATA;                  ;;將指針指向傳送資料
  Tmp_Length = ( TX_LENGTH + 1 ) / 2;   ;;以 word 方式，重新計算搬移資料次數
  for (i = 0; i < Tmp_Length; i++)      ;;將 TX_DATA 中的資料移到傳送內存中
    outport( IOdata, *(tx_data16++));   ;;每次一個 word
```

(4)將封包長度存入至 TXPLL 和 TXPLH 中：

```
iow(TXPLL , TX_LEN  &  0xff );          ;;將要傳送的長度 Low Byte 填入 TXPLL 中
iow(TXPLH, (TX_LEN  >> 8)  &  0xff );   ;;將要傳送的長度 High Byte 填入 TXPLH 中
```

(5)開始傳送封包：

```
iow(TCR, ior(TCR)  | 0x01 );            ;;發送傳送資料指令
```

(6)檢查是否傳送完成：

方式如二種，可依情況來選擇使用：

(1)使用 TCR 來檢查

```
TX_send = ior(TCR) & 0x01 ;             ;; = 0 傳送結束 , = 1 傳送中
```

(2)使用 ISR 來檢查

```
TX_send = ior(ISR) & 0x02;              ;; = 2 傳送結束 , = 0 傳送中
```

(7)檢查是否傳送成功：

使用 NSR , TSR I , TSRII 來檢查是否傳送成功：

```
TX_CHK = ior(NSR) ;
  If ((TX_CHK & 0x04) == 0x04)
   {
     if (ior(TSRI) & 0xfc) == 0x00)
       printf(" \n   TSR I 傳送成功 ");
     else
       printf(" \n   TSR I 傳送失敗 ");
   }
  else
   {
     i f (ior(TSRII) & 0xfc) == 0x00)
       printf(" \n   TSR II 傳送成功 ");
     else
       printf(" \n   TSR II 傳送失敗 ");
   }
```

我們也可以用函數來將程式變得更有可讀性：

```
unsigned char   S_NetPack(unsigned int SP_len , unsigned char *SPData)
{
  unsigned int          *SPData16;
  unsigned char         *SPData8;
  unsigned int          iTemp , iTemp2 ;

  SPData16 = SPData ;
  SPData8   = SPData;
  iow(TXPLL , SP_len );                          /* 將封包長度設置 */
  iow(TXPLH , (SP_len >> 8));

  /* 將系統內存的資料，搬到 DM9000A 中 */
  /* 這邊不用 iow() 改用 outport 增加速度，減少 IO 動作  */

  outportb(IOaddr , MWCMD);
  if (DM9000A_E.Work_mode)                       /* 檢查爲 8 / 16 bit 工作模式 */
    {
      iTemp2 = (SP_len + 1) / 2;                 /* 以 word 方式重新計算讀取次數 */
      for(iTemp = 0x0000 ; iTemp < iTemp2 ; iTemp++ )
        outport(IOdata , *(SPData16++));
    }
  else
   {
      for(iTemp = 0 ; iTemp < SP_len ; iTemp++)
        outportb(IOdata , *(SPData8++));
    }
```

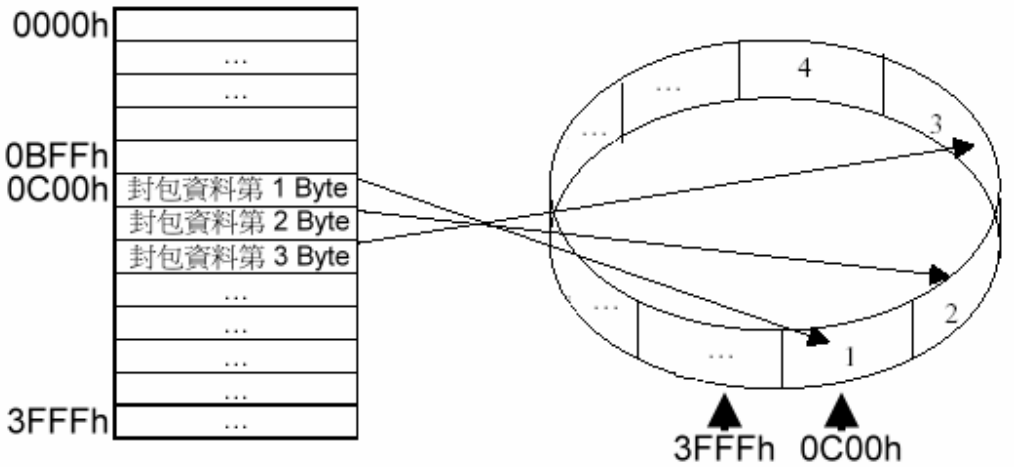
```
    iow(TCR , 0x01);                          /* 開始進行傳送 */
    while((ior(ISR) & 0x02) == 0);            /* 檢查是否有傳送完成 */
    iow(ISR , 0x02);                          /* 清除傳送完成中斷 */

    if ((ior(NSR) & 0x04) > 0)                /* 傳送完成爲 TSR1 or TSR2 */
      {
        if (ior(TSR1) == 0x00) return (1);
      }
    else
      {
        if (ior(TSR2) == 0X00) return (1);
      }
    return (0);
  }
```

而以上面的範例，使用下面命令：

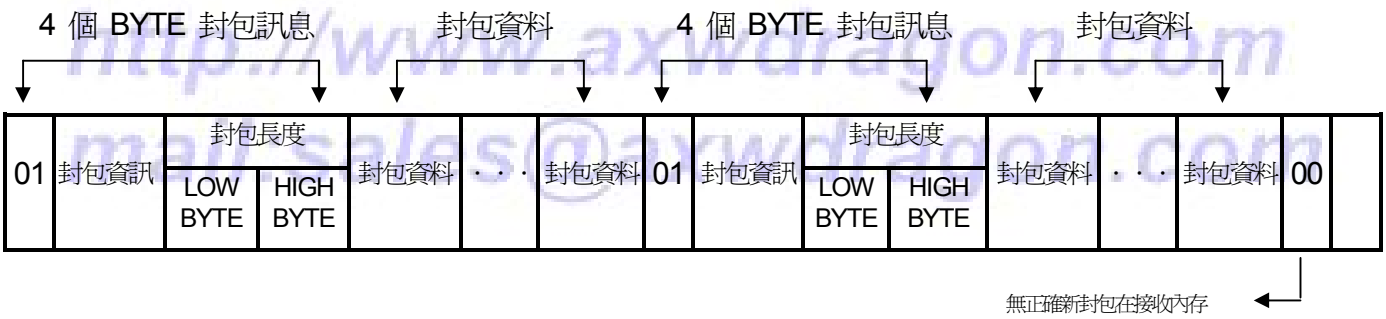
```
unsigned char TX_data[255] ;
unsigned char iTemp;

for(iTemp = 0x00 , iTemp < 0xff , iTemp++)
  TX_data[iTemp] = iTemp;
S_NetPack(0xff , TX_data);
```

第十二章、如何接收封包

DM9000A 接收到的封包，會存放於 DM9000A接收內存在於 0C00h~3FFFh 中。若是讀取位置超過 3FFFh 時，DM9000A 會自動將位置移到 0C00h 的位置。



在每一個封包，會有 4 個 Bytes 存放一些封包相關資料。第 1 個 Byte 是查看封包是否已正確存放在接收內存之中，若值爲 “01h” 爲封包已正確存放於接收內存，若爲 “00h” 則接收內存尚未有正確封包存放。在讀取其它 Byte 之前，必需要確定第 1 個 byte 是否爲 “01h”。第 2 個 Byte 則爲這個封包的一些相關訊息，它他的格式像 RSR 的格式。第 3 和 4 個 Byte 是存放這個封包的長度大小。



如何收接一個封包的流程：

(1) 檢查現在是使用那一種 IO 動作 (Byte , Word)，請參考 8。
在未來將資料搬移的作動，皆要以此模式進行，並且此一模式，需和硬體一致。

(2) 檢查是否有接收到封包：

此時先要確認系統是使用查尋式的方式還是使用 INT 中斷的方式，若是查尋式的方式，可定期用下面命令查尋，是否有接受到封包。

```
RX_IN_CHK =   ior (ISR) I 0Xfe;          ;; = 0  有封包接收  = 1 無封包接收
```

(3) 確認封包是否爲正常的封包：

此時運用到 讀取接收內存不移動位置 MRCMDX。在使用 MRCMDX 最好連續讀取二次，以確保讀取的封包爲最新的封包。

```
ior(MRCMDX);                             ;;第一次讀取到資訊，先不理。
RX_P_CHK =   ior(MRCMDX)   & 0xff;       ;; = 1 爲正常封包， != 1 爲異常封包 */
```

(4) 若是正常的封包，取得封包相關資訊和長度：

此時運用到 讀取接收內存並移動位置 MRCMD。這時取讀時，其位置會隨著使用 MRCMD 的次數自動移動，這一點和 MRCMDX 不一樣，請注意。(這兒不使用 ior 改使用 inportb、inport 這樣速度可以增快很多)

```
outportb( IOaddr, MRCMD );               ;;將位置指向 MRCMD
```

I / O Byte Mode:

```
RX_Status = inportb(IOdata) | (inprotb(IOdata) << 8);   ;;取得此一封包相關資訊
RX_Length = inportb(IOdata) | (inprotb(IOdata) << 8);   ;;取得此一封包的長度
```

I / O Word Mode:

```
RX_Status = inport( IOdata );            ;;取得此一封包相關資訊
RX_Length = inport( IOdata );            ;;取得此一封包的長度
```

(5) 取得封包相關資訊和長度，開始接收資料：

在你取得這個封包的資訊、長度之後，再使用 REG_F2 取得封包所包含的資料。

```
unsigned char RX_DATA[2048];             ;;系統內存放置接收回來封包的位置
unsigned char     *rx_data8;             ;;設定 8 bit 資料指針
unsigned int      *rx_data16;            ;;設定 16 bit 資料指針
```

```
outportb( IOaddr, MRCMD );          ;;將位置指向 MRCMD
```

I / O Byte Mode:

```
rx_data8 = RX_DATA;                 ;;將指針指向接收資料的位置
for( i = 0 ; i < RX_Length ; i++ )  ;;將接收內存中的資料移到 RX_DATA 中
  *(rx_data8++) = inportb( IOdata );  ;;每次一個 byte
```

I / O Word Mode:

```
rx_data16 = RX_DATA;                ;;將指針指向接收資料的位置
Tmp_Length = ( RX_Length + 1 ) / 2;  ;;以 word 方式，重新計算搬移資料次數
for( i = 0 ; i < Tmp_Length ; i++ )  ;;將接收內存中的資料移到 RX_DATA 中
  *(rx_data16++) = inport( IOdata );  ;;每次一個 word
```

我們也可以用函數來將程式變得更有可讀性：

```
/* 檢查內存是否有封包進入 */
unsigned char CheckNetPack(void)
{
  DM9KREG_r(MRCMDX);
  switch(DM9KREG_r(MRCMDX))              /* 檢查內存是否還有封包 */
    {
      case 0x00 :
        {
          DM9KREG_w(ISR , 0x0c);         /* 清除 ISR 有關接收資訊 */
          return(0);                     /* 尚無封包進入 */
            break;
        }
      case 0x01 : return(1);break;       /* 有封包進入 */
      default    : DM9000A_reset(); return(0);   /* 內存出錯，重置 DM9000A */
    }
}
```

```
/* DM9000A 接收封包函數 */
unsigned int R_NetPack(unsigned char *RPData)
{
   unsigned char     RX_good , RX_status;
   unsigned int      iTemp , iTemp2 , RP_len;
   unsigned int      *RPData16;
   unsigned char     *RPData8;

   /* 這邊不用 DM9KREG_r() 改用 DM9Kaddr , DM9Kdata 增加速度，減少 IO 動作 */
   /*  讀取 FIFO 現在封包相關資訊 */
   RP_len = 0x0000;
   RPData16 = RPData;
   RPData8   = RPData;

   outportb(DM9000A_E.DM9Kaddr , MRCMD);
    if (DM9000A_E.Work_mode)                          /* 檢查爲 8 / 16 bit 工作模式 */
      {
          iTemp2 = inport(DM9000A_E.DM9Kdata);
          RX_good = iTemp2 &0xff ;                     /* 取回是否爲好 */
          RX_status = iTemp2 >> 8;                     /* 取得此一封包相關資訊 */
          RP_len = inport(DM9000A_E.DM9Kdata);         /* 取得此一封包長度 */
          iTemp2 = (RP_len + 1) / 2;                   /* word 方式計算讀取次數 */
                                                       /* 將資料搬到系統內存之中 */
          for(iTemp = 0x0000 ; iTemp < iTemp2 ; iTemp++ )
            *(RPData16++) = inport(DM9000A_E.DM9Kdata);
      }
   else
      {
          RX_good = inportb(DM9000A_E.DM9Kdata);       /* 取回是否爲好 */
          RX_status = inportb(DM9000A_E.DM9Kdata);     /* 取得此一封包相關資訊 */
          RP_len = inportb(DM9000A_E.DM9Kdata) & 0x00ff; /* 取得此一封包長度 */
          RP_len |= (inportb(DM9000A_E.DM9Kdata) << 8) & 0xff00;
```

```
                                                    /* 將資料搬到系統內存之中 */
            for(iTemp = 0x0000 ; iTemp < RP_len ; iTemp++ )
              *(RPData8++) = inportb(DM9000A_E.DM9Kdata);
          }
        return (RP_len);                            /* 回傳封包長度，包含 CRC */
    }
```

而以上面的範例，使用下面命令：

```
unsigned char RX_data[2048] ;
unsigned int iTemp;
if (CheckNetPack() == 0x01)
  iTemp = R_NetPack(RX_data);

if(iTemp > 0x00) printf("封包接收完成");
```

第十三章、傳送、接收寄存器設置

DM9000A 在傳送和接收過程中，有二個重要的寄存器，分別爲 TCR , RCR。此章分別將之細述如下：

TCR 爲傳送寄存器，每一個 bit 說明如下：

Bit	名稱	默認值	說明
7	RESERVED	0,唯讀	不使用
6	TJDIS	0,可讀寫	是否傳送 Jabber 封包 (> 2048 bytes 封包) = 1 不傳送 Jabber 封包 = 0 可傳送 Jabber 封包
5	EXCECM	0, 可讀寫	連續碰撞模式控制 (若連續接收到 “碰撞封包” (Collision) 超過 15 次) = 1 保持現在傳送的封包 = 0 放棄現在傳送的封包
4	PAD_DIS2	0, 可讀寫	TSR II 封包內容不足 60 bytes 自動填滿 = 1 不自動填滿 = 0 自動填滿
3	CRC_DIS2	0, 可讀寫	TSR II 在封包最後加上 4 bytes CRC 檢查碼 = 1 不自動加上 CRC = 0 自動加上 CRC
2	PAD_DIS1	0, 可讀寫	TSR I 封包內容不足 60 bytes 自動填滿 = 1 不自動填滿 = 0 自動填滿
1	CRC_DIS1	0, 可讀寫	TSR I 在封包最後加上 4 bytes CRC 檢查碼 = 1 不自動加上 CRC = 0 自動加上 CRC
0	TXREQ	0, 可讀寫	傳送傳送內存中的資料 = 1 開始進行傳送 = 0 現無進行傳送 (在傳送完成之後，會自動清除爲 0)

TCR 寄存器一般在傳送設置，會設置成 0x01 。

RCR 爲接收寄存器，每一個 bit 說明如下：

Bit	名稱	默認值	說明
7	RESERVED	0, 唯讀	不使用
6	WTDIS	0, 可讀寫	關閉看門狗 (是否接收到 > 2048 bytes 封包) = 1 關閉看門狗功能 = 0 啓動看門狗功能
5	DIS_LONG	0, 可讀寫	不接收超長封包功能 (> 1522 bytes 封包) = 1 不接收超長封包 = 0 可接收超長封包
4	DIS_CRC	0, 可讀寫	不接收 CRC 錯誤封包 = 1 不收CRC錯誤封包 = 0 可收CRC錯誤封包
3	ALL	0, 可讀寫	接收所有的 Multicast = 1 接收所有 Multicast = 0 依 MAR 16~1Dh 設定接收
2	RUNT	0, 可讀寫	接收超短封包 (< 64 bytes 封包) = 1 接收超短封包 = 0 不接收超短封包
1	PRMSC	0, 可讀寫	接收所有位置封包 = 1 接收所有位置封包 = 0 只接收 PAR , MAR 設定封包
0	RXEN	0, 可讀寫	啓動接收封包功能 = 1 接收封包 = 0 停止接收封包

RCR 寄存器一般在接收時，會設置成 0x31 。

第十四章、如何使用快速傳送模式

在一般內嵌的 MCU 以速度效能並不高，固若可以減少 DM9000 一些 I/O 動作，實質可以提昇一些在操作之效能，而「快速傳送模式」就可以使 DM9000 達到此一目的。

使用快速傳送模式，會使用到的寄存器和標準傳送模式的寄存器一樣 (MWCMD , TXPLL , TXPLH , TCR , ISR,TSRI , TSRII)，另外加上了 EXTCSR 寄存器。這個寄存器功能說明如下：

ETXCSR 爲快速傳送封包設置寄存器，每一個 bit 說明如下：

Bit	名稱	默認值	說明
7	ETE	0, 可讀寫	啓動快速傳送封包模式 = 1 啓動 = 0 不啓動 (啓動時不可將 TCP/IP 加速功能啓動，否測會造成傳送資料錯誤)
6	ETS2	0, 唯讀	傳送封包 -2 快速傳送封包模式是否完成 = 1 未完成 = 0 完成
5	ETS1	0, 唯讀	傳送封包 -1 快速傳送封包模式是否完成 = 1 未完成 = 0 完成
4:2	RESERVED	0, 唯讀	不使用
1:0	ETT	0, 可讀寫	當傳送封包資料寫入內存多少比例時，自動開始傳送封包 bit 1 bit 0 比例設置 0 0 12.5 % 0 1 25 % 1 0 50 % 1 1 75 %

一般在傳送模式流程如下：

1. 將封包大小填入 TXPLL , TXPLH
2. 將封包資料填入 MWCMD ，依 BYTE , WORD 方式分次數寫入。直到資料寫入完整
3. 將 TCR 下達傳送命令
4. 查看 ISR , TSRI , TSRII 看傳送是否完成

若要傳送其他封包，需要將上述 1 ~ 4 的動作重覆一次

快速傳送模式流程會變動如下：

1. 將封包大小填入 TXPLL , TXPLH
2. 將封包資料填入 MWCMD ，依 BYTE , WORD 方式分次數寫入。
   DM9000A 會依照 EXTCRS 的 ETT 設置，開始進行傳送封包。並不用等到資料寫入完整。
3. 查看 ISR , EXTCRS 看傳送是否完成

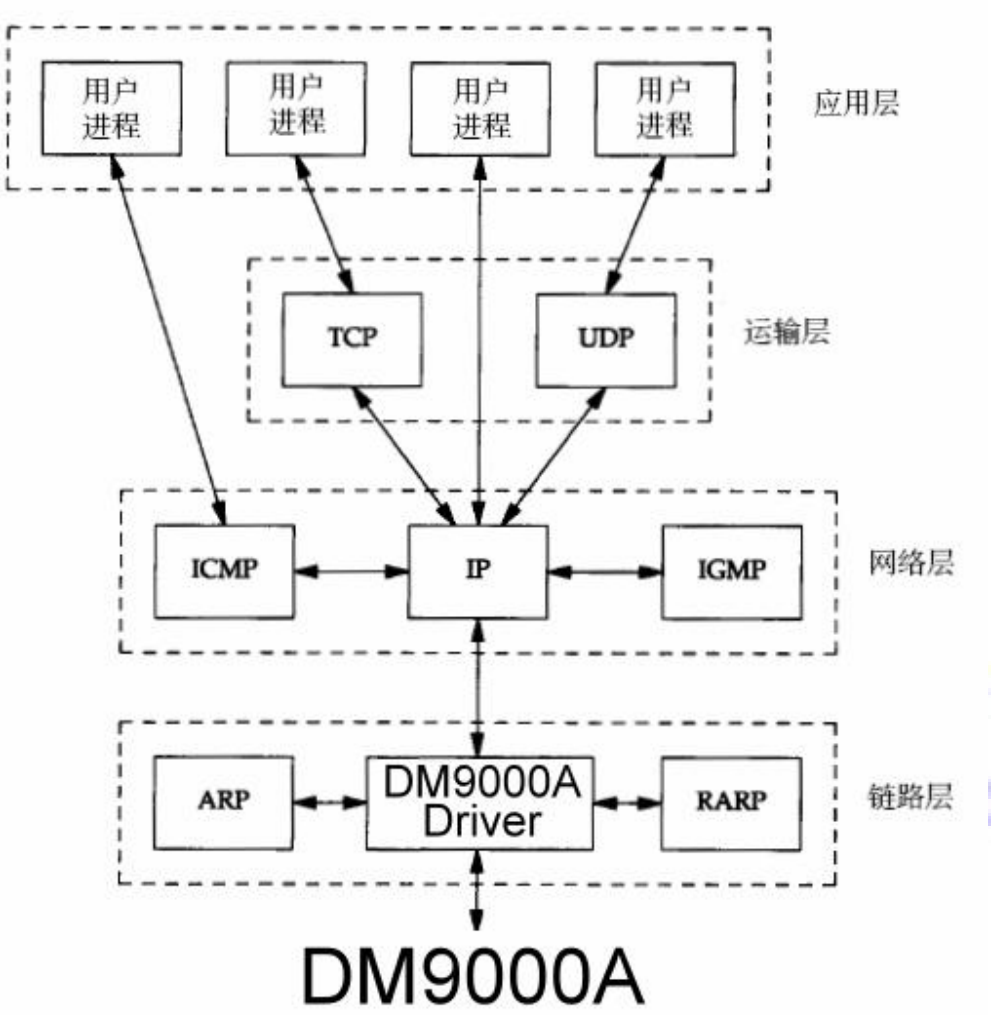
若要傳送其他封包，需要將上述 1 ~3 的動作重覆一次

若以快速傳送模式大量的傳送封包，在每次傳送時，可以減少 2～4 個動作，以增進系統效能。而 EXTCRS 的 ETT 設置時要注意，此設置需看 MCU 的效能，若 MCU 效能不高或是 ETT 設置比例太低， DM9000A 會不斷的傳送 CRC 有錯誤的封包，可能會造成整體網路有問題 (網路風暴) 。

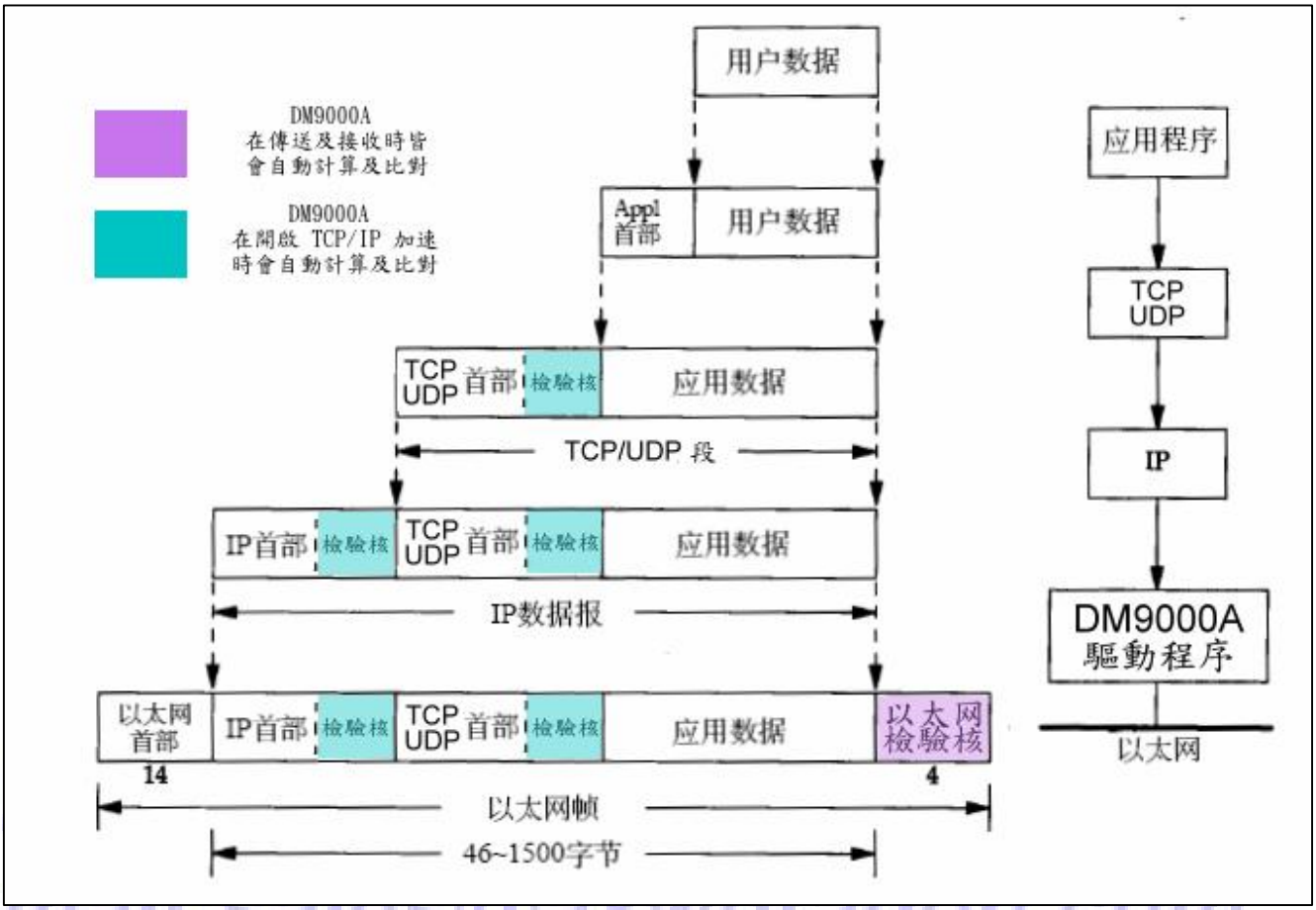
另外還要注意一點「加速傳送模式」和「TCP/IP 加速模式」並不相容，二者只能選擇一個使用，若是使用者可以修改 TCP/IP 代碼，建議使用「TCP/IP 加速模式」

第十五章、如何使用 TCP/IP 加速功能

在網路傳送或接收資料中，除了一般的應用程序所使用之資料外，還有包含一些 IP , TCP , UDP 封包格式的資料。請參看下面的說明：



而每個 IP , TCP, UDP 的封包格式資料中，皆有其檢驗核，以確保資料之正確性。



一般的 TCP/IP 的協議會使用 MCU 的計算來製作及比對其檢驗核。這些檢驗核會消耗大部份 MCU 的效能。DM9000A 對此一方面特別加入的 TCP/IP 加速功能，主要針對 IP , TCP , UDP 其檢驗核自動計算及自動比對，大大減少 MCU 在此所需花費之效能。

在 TCP/IP 加速功能中，分別會使用到 TCSCR , RCSCSR 二個寄存器，下面分別來說明其功能

1. 傳送 TCP/IP 加速功能：

若是可以修改 TCP/IP 協議，可以將其 IP , TCP , UDP 檢驗核計算功能跳過。將 TCSCR 寄存器設置成 0X87，則 DM9000A 在下執傳送命令時，會自動查看現在傳送封包資料是否爲 IP , TCP , UDP 的封包，若是就會自動計算其檢驗核，並且自動寫在其 DM9000A 傳送內存之中後，開始傳送。

TCSCR 爲 TCP/IP 傳送檢驗核自動計算設置寄存器，每一個 bit 說明如下：

Bit	名稱	默認值	說明
7:3	RESERVED	0, 唯讀	不使用
2	UDPCSE	0, 可讀寫	啓動 UDP 檢驗核自動計算模式 = 1 啓動 = 0 不啓動
1	TCPCSE	0, 可讀寫	啓動 TCP 檢驗核自動計算模式 = 1 啓動 = 0 不啓動
0	IPCSE	0, 可讀寫	啓動 IP 檢驗核自動計算模式 = 1 啓動 = 0 不啓動

在 TCP/IP 傳送加速模式，可設置成 0x07。

2. 接收 TCP/IP 加速功能：

若是可以修改 TCP/IP 協議，可以將其 IP , TCP , UDP 檢驗核比對功能跳過。此時可以將 RCSCSR 寄存器設置成 0x02 或 0x03 。

RCSCSR 爲 TCP/IP 接收檢驗核自動比對設置寄存器，每一個 bit 說明如下：

Bit	名稱	默認值	說明
7	UDPS	0, 唯讀	目前接收內存第一筆封包 UDP 檢驗核比對錯誤 (若是 UDP 封包) = 1 比對錯誤 = 0 比對正確
6	TCPS	0, 唯讀	目前接收內存第一筆封包 TCP 檢驗核比對錯誤 (若是 TCP 封包) = 1 比對錯誤 = 0 比對正確
5	IPS	0, 唯讀	目前接收內存第一筆封包 IP 檢驗核比對錯誤 (若是 IP 封包) = 1 比對錯誤 = 0 比對正確
4	UDPP	0, 唯讀	目前接收內存第一筆封包是否爲 UDP 封包 = 1 是 UDP 封包 = 0 不爲 UDP 封包
3	TCPP	0, 唯讀	目前接收內存第一筆封包是否爲 TCP 封包 = 1 是 TCP 封包 = 0 不爲 TCP 封包
2	IPP	0, 唯讀	目前接收內存第一筆封包是否爲 IP 封包 = 1 是 IP 封包 = 0 不爲 IP 封包
1	RCSEN	0, 可讀寫	啓動 TCP/IP 接收檢驗核自動比對模式 = 1 啓動 = 0 不啓動 (啓動時會將接收封包內存的第一個 BYTE 資料格式改變，反應此封包的格式)
0	DCSE	0, 可讀寫	不接收 TCP/IP 接收檢驗核比動比對失敗的封包 = 1 不收比對失敗封包 = 0 可收比對失敗封包

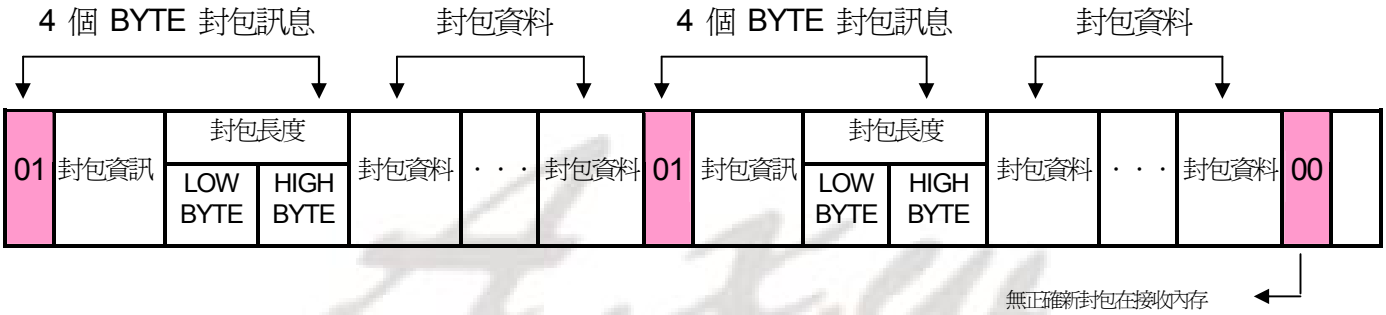
在 TCP/IP 接收加速模式，可設置成 0x03 或 0x02 (建議設置成 0x03)

在開啓 TCP/IP 接收加速模式後，會改變 DM9000A 在接收資料的流程及判別的方法，這一點需要特別的注意。

標準接收模式 和 TCP/IP 接收加速模式 其最大的差異如下：

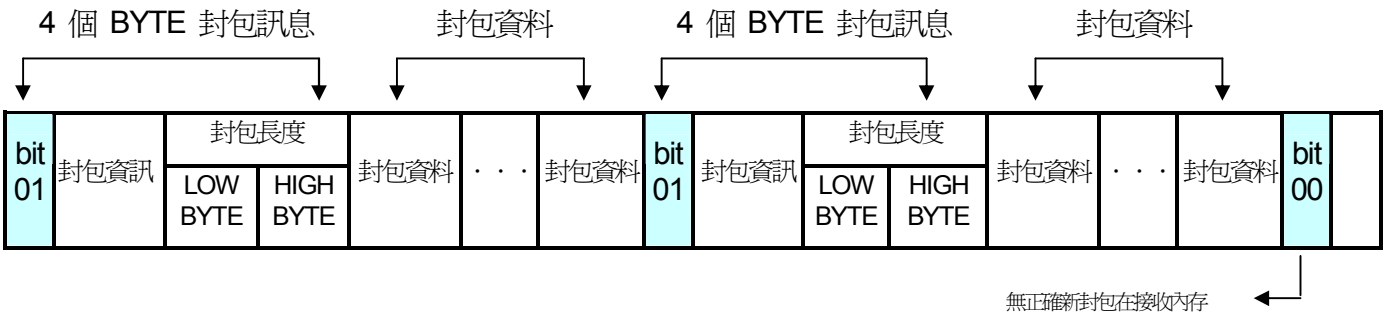
標準的接收模式：

在每一個封包，會有 4 個 Bytes 存放一些封包相關資料。第 1 個 Byte 是查看封包是否已正確存放在接收內存之中，若值爲 “01h” 爲封包已正確存放於接收內存，若爲 “00h” 則接收內存尚未有正確封包存放。在讀取其它 Byte 之前，必需要確定第 1 個 byte 是否爲 “01h”。第 2 個 Byte 則爲這個封包的一些相關訊息，它他的格式像 RSR 的格式。第 3 和 4 個 Byte 是存放這個封包的長度大小。



TCP/IP 接收加速模式：

在每一個封包，會有 4 個 Bytes 存放一些封包相關資料。第 1 個 Byte 是查看封包是否已正確存放在接收內存之中 及 TCP/IP 檢驗核 及 封包格式相關資料，若值最小二個 bit 爲 “0x01” 爲封包已正確存放於接收內存，若爲 “0x00” 則接收內存尚未有正確封包存放。而其他 bit 的格式，和寄存器 RCSCSR 一樣，可以由此格式來查看爲何種封包及檢驗核是否有出錯。若是 RCSCSR 設置成 0x02 時若檢驗核出錯，封包資料還是可以進入接收內存；若是設置成 0x03 時，只要檢驗核出錯則此封包並不會進入接收內存。在讀取其它 Byte 之前，必需要確定第 1 個 byte 是否最小二個 bit 爲 “0x01”。第 2 個 Byte 則爲這個封包的一些相關訊息，它他的格式像 RSR 的格式。第 3 和 4 個 Byte 是存放這個封包的長度大小。



若是第一個 BYTE 的資訊如下，分別代表各種不同的情況：

BYTE 值	UDPS	TCPS	IPS	UDPP	TCPP	IPP	封包檢查	封包情況說明
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1:0	
0x00	0	0	0	0	0	0	00	無正確的封包待接收
0x01	0	0	0	0	0	0	01	收到非 IP,TCP,UDP 封包
0x05	0	0	0	0	0	1	01	收到 IP 封包，檢驗正確
0X0D	0	0	0	0	1	1	01	收到 TCP 封包，檢驗正確
0x15	0	0	0	1	0	1	01	收到 UDP 封包，檢驗正確
有收到 TCP/IP 封包，但是檢驗出錯。								
0x25	0	0	1	0	0	1	01	收到 IP 封包，檢驗錯誤
0x4D	0	1	0	0	1	1	01	收到 TCP 封包，檢驗錯誤
0x95	1	0	0	1	0	1	01	收到 UDP 封包，檢驗錯誤
內存出錯的情況，需重置 DM9000A								
	X	X	X	X	X	X	00	接收內存出錯
	X	X	X	X	X	X	10	接收內存出錯
	X	X	X	X	X	X	11	接收內存出錯
	X	X	X	1/0	1/0	0	XX	接收內存出錯，
	1/0	1/0	1/0	X	X	X	01	若 RCSCSR Bit 0 = 1 ，且接收的 byte bit 5~7 爲非零時，接收內存出錯

X 爲任意值

若開啓 TCP/IP 加速功能時，其封包檢查功能，需修改如下：

```
/* 檢查內存是否有封包進入 */
unsigned char CheckNetPack(void)
{
  unsigned char check_pack;
  outportb(DM9000A_E.DM9Kaddr , MRCMDX);
  check_pack = inportb(DM9000A_E.DM9Kdata);
  check_pack = inportb(DM9000A_E.DM9Kdata);      /* 檢查內存是否還有封包 */

  if(check_pack == 0x00)
    {
      DM9KREG_w(ISR , 0x0c);                     /* 清除 ISR 有關接收資訊 */
      return(0);                                 /* 尚無封包進入 */
    }
```

```
    if(check_pack == 0x01) return(1);                    /* 判斷正確的的封包 */
    if(check_pack == 0x05) return(1);                    /* 判斷正確的的封包 */
    if(check_pack == 0x0d) return(1);                    /* 判斷正確的的封包 */
    if(check_pack == 0x15) return(1);                    /* 判斷正確的的封包 */

    DM9000A_reset();                                     /* 內存出錯，需重置 */
    Returen(0);
}
```

第十六章、網路流量控制

在網路資料在快速傳送時，若 MCU 處理速度不夠快，會造成封包丟失。這時可以開啓網路流量控制(FCR)，功能來減少封包丟失的情況。請注意，此設定需要 DM9000A 所連接的設備必需皆有支持才有用，否則封包還是會丟失的。

FCR 爲流量控制設置寄存器，每一個 bit 說明如下：

Bit	名稱	默認值	說明
7	TXP0	0, 可讀寫	傳送 Flow Control 回復傳送封包 (暫停時間 = 0000H) = 1 傳送回復傳送封包 = 0 不傳送回復傳送封包 (在回復傳送封包傳送完成後，自動清除爲 0)
6	TXPF	0, 可讀寫	傳送 Flow Control 暫停傳送封包 (暫停時間 = FFFFH) = 1 傳送暫停傳送封包 = 0 不傳送暫停傳送封包 (在暫停傳送封包傳送完成後，自動清除爲 0)
5	TXPEN	0, 可讀寫	啓動自動傳送 Flow Control 封包 (全雙工模式下有效) = 1 啓動傳送流量控制 = 0 不啓動傳送流量控制
4	BKPA	0, 可讀寫	啓動 Back Pressure 條件模式 <任意封包> (半雙工模式下有效) = 1 啓動預防溢出模式 = 0 不啓動預防溢出模式 (啓動後，在 BPTR 08h 條件成立，再收到任一封包。就送出 jam 訊號)
3	BKPM	0, 可讀寫	啓動 Back Pressure 條件模式 <專屬封包> (半雙工模式下有效) = 1 啓動預防溢出模式 = 0 不啓動預防溢出模式 (啓動後，在 BPTR 08h 條件成立，收到專門給此 MAC 的封包。才送出 jam 訊號)
2	RXPS	0, 唯讀, 自動清除	有接收到 Flow Control 暫停 / 回復 傳送封包 = 1 有接收到 = 0 未接收到
1	RXPCS	0, 唯讀	現是否在 Flow Control 暫停傳送模式 = 1 在暫停傳送模式 = 0 不在暫停傳送模式
0	FLCE	0, 可讀寫	啓動接收 Flow Control 封包模式 = 1 啓動 = 0 不啓動 (啓動後，若收到 Flow Control 暫停/回復 傳送封包，DM9000A 會依封包設置 暫停/回復傳送封包)

在有開啓流量控制設置，可設置成 0x28 或 0x29。(建議設置成 0x28)

網路流量控制 DM9000A 有支持二種模式 Back Pressure 和 Flow Control 。分別支持在半雙工模式和全雙工模式。分別說明如下：

Back Pressure 模式 ： <半雙工才有支持>

此模式有相關的寄存器有 BPTR 和 FCR 。其工作模式如後：

BPTR 設置其接收內存還剩多少空間時，會自動傳送 jam 訊號給 DM900 連接設備。(虛擬的傳送一個「碰撞」訊號(COL)出來)使其設備對方先暫停傳送封包。此模式在一般設備皆有支特。

Flow Control 模式 ： <全雙工才有支持>

此模式有相關的寄存器有 FCTR 和 FCR 。其工作模式如後：

FCTR 和 BPTR 一樣設置其接收內存還剩多少空間時，會自動傳送要求連接設備暫停傳送的封包。而特殊的地方多了一項，接收內存大於空間時，會自動傳送繼續傳送封包的功能。

另外 DM9000A 除了會自動要求對方暫停。也可以接收連接設備傳送過來的 Flow Control 封包，自動也進行暫停傳送和回復傳送的動作。

在設置 Flow Control 模式時，在內置 PHY 需要將 PHY 寄存器中 REG 4 寄存器的 bit 開啓這樣 DM9000 內置 PHY 才會啓動接收並且傳送 Flow Control 功能。

設置方式如下：

```
gpio_set (0x00 , 0x01);          ;;將內部的 PHY 電源關閉
phy_w( 0x04 , 0x05e1);           ;;將自適應模式及流控資料填入
phy_w( 0x00 , 0x1000);           ;;切換 PHY 到自適應模式
gpio_set(0x00 , 0x00);           ;;將內部的 PHY 電源開啓
```

請特別注意，若連接設備不支持此一模式，此功能無法使用。

第十七章、其他

1. IO 設定值控制順序

DM9000A 一些相關的設定，皆有默認值。也可以使用外部設置或是 EEPROM 來設置。但是如果三者同時存在是依何爲主呢。結果如後：默認值 ＜ 外部設定值 ＜ EEPROM 設置值。

例如：

DM9000A INT 輸出電位爲例，

默認值爲 ：高電位爲使能輸出電位

將 EECK 接高 ：低電位爲使能輸出電位

此時 INT 會以低電位爲使能輸出電位

若是此時 EEPROM 設置 WORD 3 bit 2 = 1 , WORD 6 bit 3 = 0

此時 INT 會以高電位爲使能輸出電位

2.傳送與接收超長偵測

依造 802.3 格式，一個封包大小應在 64 byte ~ 1518 byte (包含 CRC) ，若加上 VLAN 長度可達 1536 Byte 。但是若在傳送與接收到一個超長的封包時，系統要如何偵測：

若是傳送一個超長封包 (> 2048 bytes)，會將 TCR I bit 7 或 TCR II bit 7 設成 1 。

若是收到一個超長封包 (> 2048 bytes)，會將 RSR bit 4 設成 1。

默認值偵測是閉啓的，若想要將之關閉。傳送將 TCR bit 6 設成 1 、接收將 RCR bit 6 設成 1。

3.DM9000A 在不同模式下的效能

DM9000A 效能是由 MCU 處理能力來產生的。而我們推論效能如下：若是 MCU 速度十分快速，完全不會有其他處理時間的話。DM9000A 每一次動作需要時間 10ns + 10ns = 20ns==> 50000000bps ==> 47.68Mbps。(即做一次 IOR# , IOW# 最少所需的時間，請看 datasheet p39.40)

由此推算若是不管其他裝置處理所花的時間。 DM9000A 理論可以達到。

8 BIT 模式	只有在收或送資料	==>	8 * 47.68Mbps	==> 381.44 Mbps *
	同時收和送資料	==>	8 * 47.68Mbps / 2	==> 190.72 Mbps *
16 BIT 模式	只有在收或送資料	==>	16 * 47.68Mbps	==> 762.88 Mbps *
	同時收和送資料	==>	16 * 47.68Mbps / 2	==> 381.44 Mbps *

* 因爲 DM9000A 是 10/100M 的網卡，所以最大值只能達到 100Mbps 。