以太网控制模组说明书

Technology for easy living

凌阳科技股份有限公司
SUNPLUS TECHNOGY CO., LTD.

目录

以太网控制板说明书

摘要

想象一下：您准备欣赏电视节目，在您说“看电视，新闻联播。”这句话的时候，电视打开了，频道调整到中央一台，窗帘拉上了，房间的光线逐渐调整到最适宜看电视的程度……或者，在您回家的路上，自动为您提前打开空调，调整室内温度到合适程度，接通通风装置；调节好室内的光线；把您预先准备好的食物用微波炉加热。这样，您一回来就可以享受到清新的空气、舒适的环境，或许还有一杯热茶……

这看起来就想科幻影片里描写的一样。可是，事实上这样的生活已经近在咫尺。随着 IT 产业的发展和人们生活水平的提高，“智能住宅”、“家庭自动化”等技术的发展正受到人们的密切关注。

DM9000 是一款以太网控制芯片，在网络中它可自动获得同设定 MAC 地址一致的 IP 包，完成 IP 包的收发，再用一个单片机来结合完成上层协议，就构成了一个完整的网络终端。这里提供一个采用 SPCE061A 和 DM9000 构成的 Web Server.

一、DM9000 控制板简介

基本参数

额定频率：50Hz；
额定电压：直流（DC）5V；
额定功率：2W；

产品执行标准

凌阳大学计划项目标准。

技术特性

1、和 MCU 连接模式有 ISA 8 bit / ISA 16 bit 模式，并且支持 3.3V 和 5V 的 I/O 控制。可方便和不同电压和界面的 MCU 连接。。

2、支持多种连接模式；电端口支持10M HALF/ 10M FULL / 100MHALF/100M FULL / AUTO (N-WAY) 。

3、拥有 4Mbit 串行数据存储器及其接口。

功能介绍

1）可通过此设备和凌阳单片机互连到局域网或者广域网进行通讯。

2）支持多种连接模式；电端口支持 10M HALF/ 10M FULL / 100MHALF/100M FULL / AUTO (N-WAY)。

3）可在此卡上进行数据存储。

4）通过 SPCE061A 与此设备搭配可完成一个简单的 WEB 服务器。

5）通过 SPCE061A 与此设备搭配可完一些简单的控制。

连接操作方式

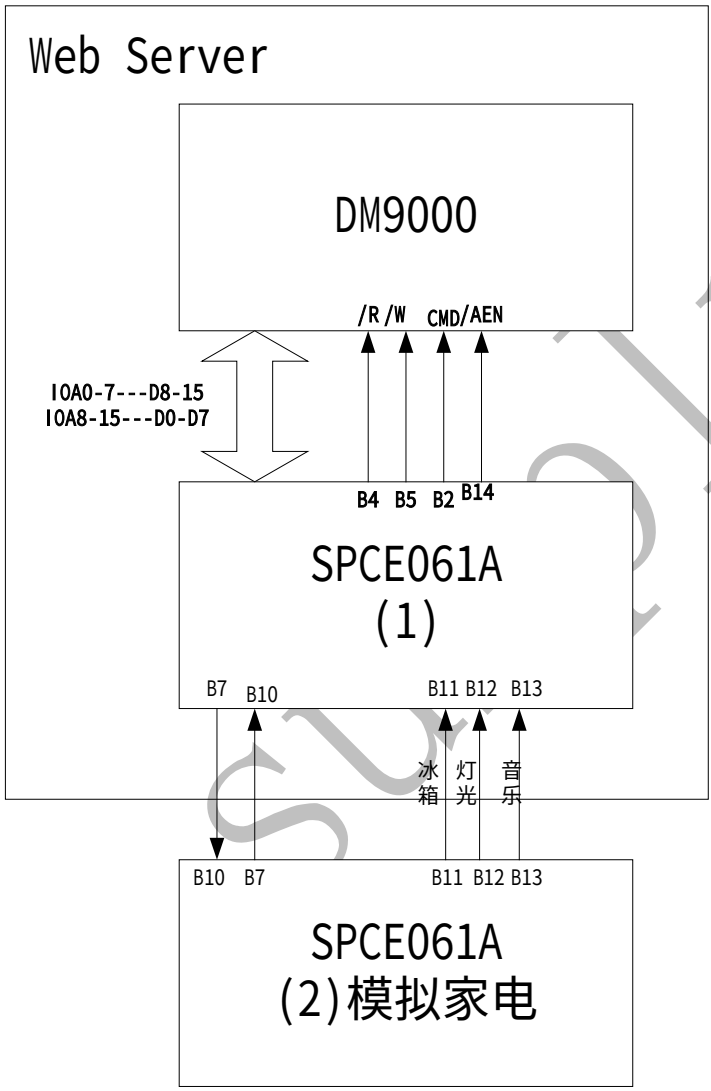

图 1 整体连接图

模块分配及其外观

外观如图 1-1 所示：

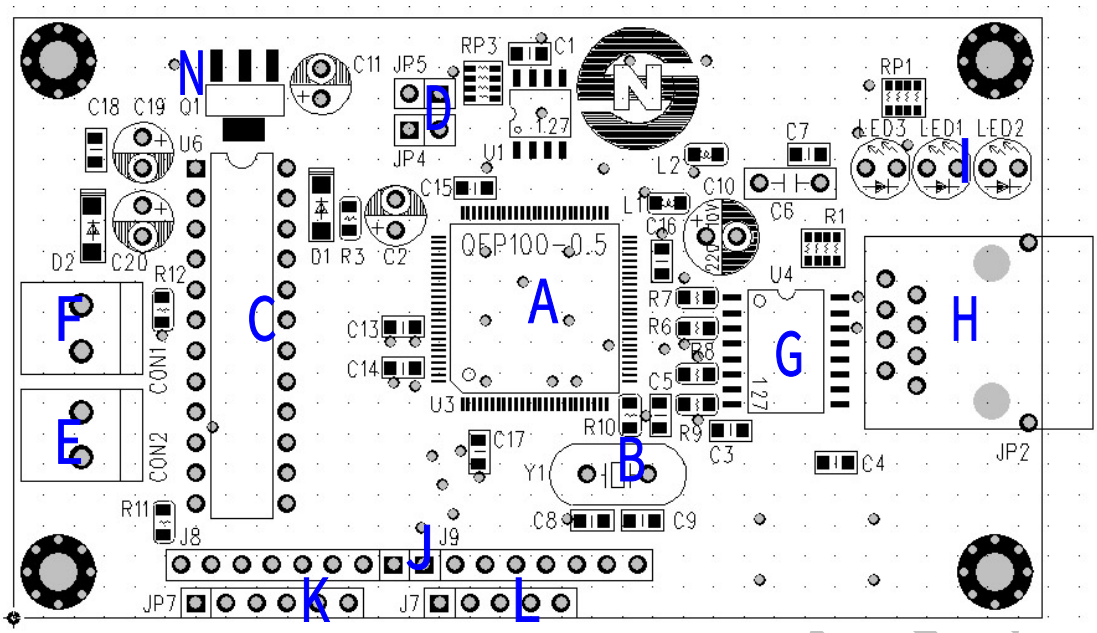

图 2　外观

各模块说明如表 1-1。

A	DM9000	H	RJ-45 接座
B	25MHz 晶振	I	连接状态指示灯
C	SPR4096	J	DM9000 数据端口引出
D	控制跳线	K	SPR4096 操作端口
E	电源输入口	L	DM9000 控制端口引出
F	电源输出口	N	5V to 3.3V 变压器
G	10/100 滤波变压器		

表 1-1　操作面板

控制跳线说明：

JP4 打开时为选用 EEPROM,关闭时不选。

JP5 打开时为 8bit 模式，关闭时为 16bit 模式

注意事项

1、 电源供应必须保证在 200mA 以上。

2、 与 MCU 配合时数据线同控制线不要超过 20cm。

3、 编程之前请根具需要设定 JP5 和 JP6。

主要元件介绍

SPR4096

封装采用 Dip24 管脚封装。

SPR4096 使整个系统增加了 4M 存储空间，这同凌阳的语音录放相配合将产生很好的效果，大大增加了此系统的数据存储空间，可进行网络上的一些稳定数据的存储。

DM9000

1．封装采用LQFP 100 管脚封装，所占用之面积和高度小。
2．使用电压为3.3V ，最大耗用电流为100mA，十分省电。
3．和MCU 连接模式有ISA 8 bit / ISA 16 bit / uP 8bit / uP 16 bit / uP 32 bit / 68K 模式，并且支持3.3V 和5V 的I/O 控制。可方便和不同电压和界面的MCU 连接。
4．内置10/100M PHY，支持多种连接模式；电端口支持10M HALF/ 10M FULL / 100MHALF/100M FULL / AUTO (N-WAY)；另提供光端口100M HALF / 100M FULL 。
5．支持EEPROM (93C46)，可供存放系统所需信息。

二、DM9000 寄存器总表

寄存器名称	寄存器说明	寄存器位置	默认值
NCR	网络界面控制	00h	00h
NSR	网络界面信息	01h	00h
TCR	封包传送控制	02h	00h
TSR I	封包传送信息－1	03h	00h
TSR II	封包传送信息－2	04h	00h
RCR	封包接收控制	05h	00h
RSR	封包接收信息	06h	00h
ROCR	接收溢出计数	07h	00h
BPTR	条件设置	08h	37h
FCTR	条件设置	09h	38h
FCR	流量控制设置	0Ah	00h
EPCR	EEPROM / PHY 控制	0Bh	00h
EPAR	EEPROM / PHY 读写位置	0Ch	40h
EPDRL	EEPROM / PHY 资料-L	0Dh	XXh
EPDRH	EEPROM / PHY 资料-H	0Eh	XXh
WCR	唤醒控制	0Fh	00h
PAR	MAC 位置设置	10h	XXh by EEPROM
		11h	
		12h	
		13h	
		14h	
MAR	Multicast 设置	16h	XXh
		17h	XXh
		18h	XXh
		19h	XXh
		1Ah	XXh
		1Bh	XXh
		1Ch	XXh

GPCR	GPIO 界面控制	1Eh	01h
GPR	GPIO 界面信息	1Fh	XXh
VID	厂商 ID 号	28h	XXh
		29h	XXh
PID	产品 ID 号	2Ah	XXh
		2Bh	XXh
CHIPR	IC 版本号	2Ch	00h
MRCMDX	内存读取控制，不移动读取位置	F0h	XXh
MRCMD	内存读取控制，移动内存读取位置	F2h	XXh
MRRL	内存读取位置-L	F4h	00h
MRRH	内存读取位置-H	F5h	00h
MWCMDX	内存写入控制，不移动内存写入位置	F6h	XXh
MWCMD	内存写入控制，移动内存写入位置	F8h	XXh
MWRL	内存写入位置-L	FAh	00h
MWRH	内存写入位置-H	FBh	00h
TXPLL	传送封包大小设置-L	FCh	XXh
TXPLH	传送封包大小设置-H	FDh	XXh
ISR	中断信息设置	FEh	00h
IMR	中断条件设置	FFh	00h

三、DM9000 基本功能说明

3.1 读取、写入寄存器方式

DM9000 控制读取和写入寄存器方式，十分容易。DM9000 有数据和索引端口。而这二个端口由CMD 管脚控制，若CMD 接高电位时为控制资料端口， CMD 接低电位为控制索引端口。

要读写任何一个寄存器时，方式如下：

(1) 将AEN , SA4 ~9 使DM9000 使能
(2) 设置CMD 管脚为低电位
(3) 将要读写的寄存器的位置填入索引端口(IOW#)
(4) 设置CMD 管脚为高电位
(5) 将要读写的寄存器的资料填入或读出资料端口(IOW# , IOR#)

3.2 内存工作原理

DM9000 共有16K Byte(0000h ~3FFFh) 内存，而读写内存由MWCMD , MRCMD 这二个寄存器来控制。而MWRL , MWRH 寄存器提供现在写入内存的位置，MRRL , MRRH 寄存器提供现在读取内存的位置。而内存每次移动依工作模式，每次移动一个Byte(8 bit) 或二个Byte(16 bit) 或四个Byte (32 bit)

3.3 封包传送工作原理

内存中默认值有3K Byte (0000h ~ 0BFFh) 提供给传送功能使用。而传送一个封包流程如下：

(1)将要传送封包的长度，填入到TXPLL , TXPLH 寄存器
(2)将要传送封包的资料由MWCMD 寄存器填入内存中
(3)由TCR 寄存器使DM9000 送出封包资料
(4)若内存的写入位置超过0BFFh 时，自动将下一个位置回复到0000h

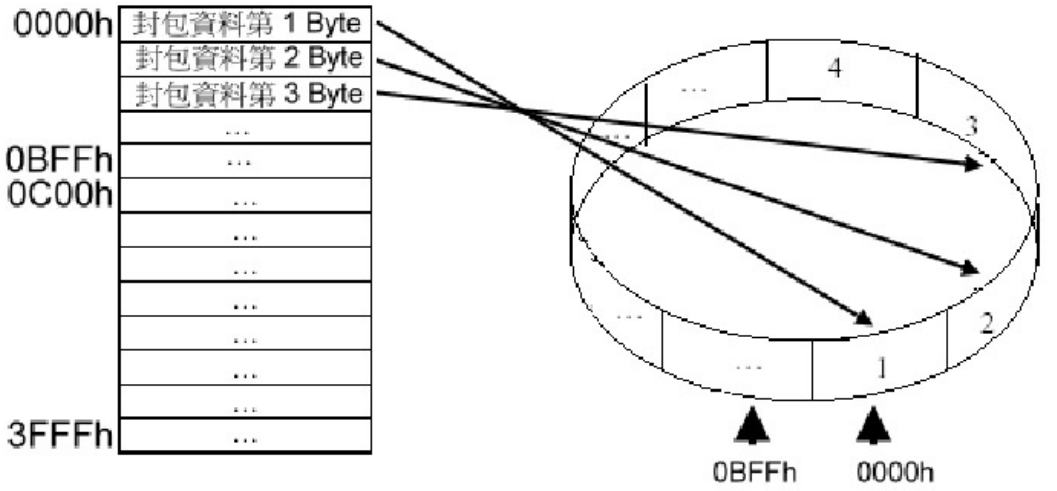

图3 传包说明

3.4 封包接收工作原理

内存中默认值有13K Byte (0C00h ~ 03FFh) 提供给接收功能使用。在每一个封包，会有4 个Byte 存放一些封包相关资料。第1 个Byte 是封包是否已存放在接收内存，若值为“01h” 为封包已存放于接收内存，若为“00h” 则RX RAM 尚未有封包存放。在读取其它Byte 之前，必需要确定第1 个byte 是否为“01h”。第2 个Byte 则为这个封包的一些相关讯息，它他的格式像RSR 寄存器的格式。第3 和4 个Byte 是存放这个封包的长度大小。4 个BYTE 封包讯息封包资料4 个BYTE 封包讯息封包资料。

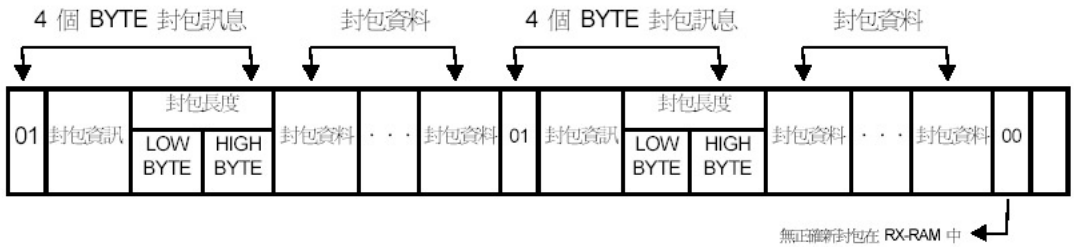

图4 封包示意图

.如何收接一个封包的流程如下：

(1) 检查MRCMDX 寄存器值是否为01 ，若有则有封包进入需读取

(2) 读取MRCMD 将前四个Byte 封包讯息读入

(3) 由前四个Byte 封包讯息取待得封包长度(以Byte 为单位)， 连续读取MRCMD 将封包资料移到系统内存之中

(4) 若读取位置超过 3FFFh 时，自动会移到 0C00h。

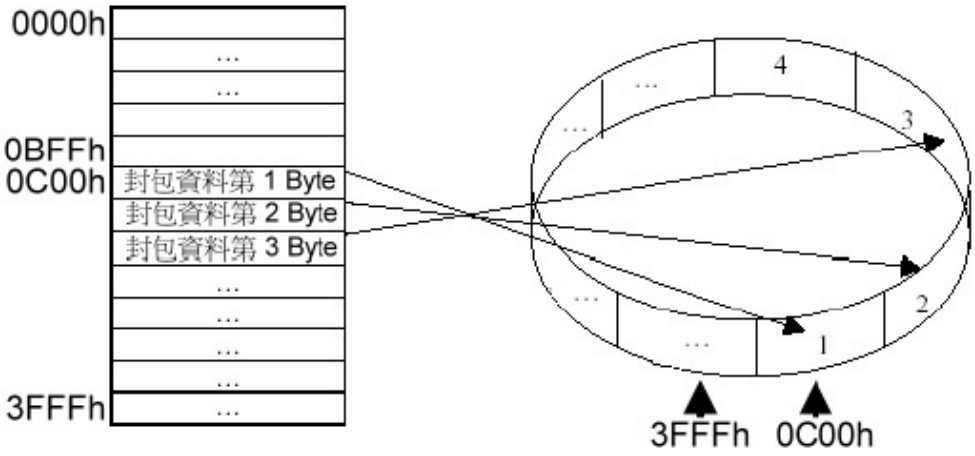

图 5　收包说明

四、DM9000 的初始化

DM9000 在正常工作之前需要做一些相关的设定，才能正常动作，建议的顺序如下：

DM9000 软件重置

```
iow(NCR , 0x03);将DM9000 进行软件重置
dellay(20);等待DM9000 重置完成
iow(NCR , 0x00);将DM9000 回复正常工作状态
iow(NCR , 0x03);将DM9000 进行软件重置
dellay(20);等待DM9000 重置完成
iow(NCR , 0x00);将DM9000 回复正常工作状态
```

上面软件重置动作有重复二次，以确保DM9000重置成功。

清除中断设定

清除中断设定，以免因中断导致 DM9000 初始化动作不正常

```
iow(0xff , 0x80);将DM9000 中断功能关闭
```

设定 DM9000 相关连接界面(内部 PHY ，外部 MII ，

Reverse-MII)

```
    gpio_set(0x00 , 0x01);              //将内部的PHY电源关闭
    phy_w(0x04 , 0x01e1);               //将自适应模式资料填入
    phy_w(0x00 , 0x1000);               //切换PHY 到自适应模式
    gpio_set(0x00 , 0x00);              //将内部的PHY 电源开启
```

设定 Physical Address 位置

```
void MACSet(void)
{
    unsigned char MAC[6] = { 0x12 , 0x34 , 0x56 , 0x78 , 0x00, 0x00};
    WriteRegister(PAR,MAC[0]);
    WriteRegister(PAR1,MAC[1]);
    WriteRegister(PAR2,MAC[2]);
    WriteRegister(PAR3,MAC[3]);
    WriteRegister(PAR4,MAC[4]);
    WriteRegister(PAR5,MAC[5]);
}
```

设定 Multicast 设置

```
void multicast_set(unsigned char Add_Del , unsigned char *set_mac)
{
    unsigned long int crc ,carry ,k;
    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;
    }
    k=crc >>4;
```

```
    k=crc >>4;
//  crc = ((crc >> 8) & 0x00ffffff) ^ carry;
    crc = (k & 0x00ffffff) ^ carry;
    }
    Hash_data = crc & 0x3f;
    iTemp = ReadRegister(MAR + (Hash_data / 8));
    if (Add_Del == 0x01)
    WriteRegister(MAR + (Hash_data / 8) ,iTemp | (0x01 << (Hash_data % 8)));
    else
    WriteRegister(MAR + (Hash_data / 8) , iTemp & ~(0x01 << (Hash_data %8)));
}
```

其它相关设置

```
    ReadRegister(0x01); //清除NSR 相关信息
    WriteRegister(0x02 , 0x00); //清除TCR 相关信息(参考13 章)
    ReadRegister(0x07); //清除ROCR 相关信息(参考14 章)
    WriteRegister(0x0a , ReadRegister(0x0a) | 0x29); //设置FCR 起动流控功能
    WriteRegister(0xfe , 0x0f); //清除ISR 相关信息
    WriteRegister(0xff , 0x83); //设置IMR 功能
```

开启接收资料功能

```
WriteRegister(0x05 , 0x39); //开启接收功能
```

五、检查现在所使用的 I/O 模式

DM9000 支持3 种I/O 模式，分别为Byte / Word / Dword ，在读写资料的方式也支持8-bit /16-bit / 32-bit，在硬件和软件上需要相互配合(即设置模式需一致)。在程序中要如何去确认现在是使用何种模式呢！可以使用ISR 来查看：

```
unsigned char IO_chk;
IO_chk = ior(ISR) >> 6; ;; IO_chk 若为0x00 工作模式为16 bit 模式
;; IO_chk 若为0x01 工作模式为32 bit 模式
;; IO_chk 若为0x02 工作模式为8 bit 模式
```

六、如何传送封包

在传送个封包之前，需将其封包资料存放在DM9000 的传送内存中0000h~0BFFh 。若是写入位置超过0BFFh 时，DM9000 会自动将位置移到0000h 的位置。将封包资料存放在MWCMD 中，DM9000 会自动将其资料存向其传送内存中。另外还需将要传送封包的大小存放在TXPLL(low_byte) 和TXPLH(high_byte)。之后再将TCR bit 0 设为1 ，此时开始进行封包的传送。而在传送完成后，会将传送是否成功的信息放在TSRI ，TSRII 中。放的顺序为TSRI -> TSRII -> TSRI ->TSRII …。所以需要依照NSR bit 2 ~ 3 来判断现在是TSRI 或TSRII 传送完成。

如何传送一个封包流程：

1.检查现在是使用那一种IO 动作(Byte , Word , Dword)，请参考9 章。
在未来将资料搬移的作动，皆要以此模式进行，并且此一模式，需和硬件一致。
2.开始将封包的资料搬移到传送内存中：(这儿不使用iow 改使用outportb、outport、outportl 这样速度可以增快很多)

```
TX_DATA[TX_LEN] : 传送的资料
TX_LEN : 传送资料长度以Byte 计算
unsigned char *tx_data8;设定8 bit 资料指针
unsigned int *tx_data16;设定16 bit 资料指针
unsigned long int *tx_data32;设定32 bit 资料指针
outportb( IOaddr, MWCMD );将位置指向MWCMD
I / O Byte Mode:
tx_data8 = TX_DATA; ;;将指针指向传送资料
for (i = 0; i < TX_LENGTH; i++) ;;将TX_DATA 中的资料移到传送内存中
outportb( IOdata, *(tx_data8++)); ;;每次一个Byte
I / O Word Mode:
tx_data16 = TX_DATA; ;;将指针指向传送资料
Tmp_Length = ( TX_LENGTH + 1 ) / 2; ;;以word 方式，重新计算搬移资料次数
for (i = 0; i < Tmp_Length; i++) ;;将TX_DATA 中的资料移到传送内存中
outport( IOdata, *(tx_data16++)); ;;每次一个word
I / O Dword Mode:
tx_data32 = TX_DATA; ;;将指针指向传送资料
Tmp_Length = ( TX_LENGTH + 3 ) / 4; ;;以dword 方式，重新计算搬移资料次数
for (i = 0; i < Tmp_Length; i++) ;;将TX_DATA 中的资料移到传送内存中
outportl( IOdata, *(tx_data32++) ); ;;每次一个dword ， outprotl 为32 bit 输出函数
```

3.将封包长度存入至TXPLL 和TXPLH 中：

```
iow(TXPLL , TX_LEN & 0xff ); ;;将要传送的长度Low Byte 填入TXPLL 中
iow(TXPLH, (TX_LEN >> 8) & 0xff ); ;;将要传送的长度High Byte 填入TXPLH 中
```

4.开始传送封包：

```
iow(TCR, ior(TCR) | 0x01 ); ;;发送传送数据指令
```

5.检查是否传送完成：
方式如二种，可依情况来选择使用：
(1)使用TCR 来检查
TX_send = ior(TCR) & 0x01 ; ;; = 0 传送结束，= 1 传送中
(2)使用ISR 检看方式来检查
TX_send = ior(ISR) & 0x02; ;; = 2 传送结束，= 0 传送中
6.检查是否传送成功：
使用NSR , TSR I , TSRII 来检查是否传送成功：

```
TX_CHK = ior(NSR) ;
If ((TX_CHK & 0x04) == 0x04)
{
if (ior(TSRI) & 0xfc) == 0x00)
printf(“ \n TSR I 传送成功“ );
else
printf(“ \n TSR I 传送失败“ );
}
else
{
if (ior(TSRII) & 0xfc) == 0x00)
printf(“ \n TSR II 传送成功“ );
else
printf(“ \n TSR II 传送失败“ );
}
unsigned char *tx_data8; ;;设定8 bit 资料指针
unsigned int *tx_data16; ;;设定16 bit 资料指针
unsigned long int *tx_data32; ;;设定32 bit 资料指针
```

我们也可以用函数来将程序变得更有可读性：

```
unsigned char S_NetPack(unsigned int SP_len , unsigned char *SPData)
{
unsigned int *SPData16;
unsigned log int *SPData32;
unsigned int iTemp , iTemp2 ;
unsigned char iTemp3;
iow(TXPLL , SP_len ); /* 将封包长度设置*/
iow(TXPLH , (SP_len >> 8));
iTemp3 = ior(ISR) >> 6;
/* 将系统内存的资料，搬到DM9000 中*/
/* 这边不用iow() 改用outport 增加速度，减少IO 动作*/
outportb(IOaddr , MWCMD);
switch(iTemp3)
{
case 0x00 : /* 16 bit 工作模式*/
```

```
iTemp2 = (SP_len + 1) / 2; /* 以word 方式重新计算读取次数*/
SPData16 = SPData;
for(iTemp = 0x0000 ; iTemp < iTemp2 ; iTemp++ )
outport(IOdata , *(SPData16++));
break;
case 0x01 : /* 32 bit 工作模式*/
iTemp2 = (SP_len + 3) / 4; /* 以dword 方式重新计算读取次数*/
SPData32 = SPData;
for(iTemp = 0x0000 ; iTemp < iTemp2 ; iTemp++ )
outportl(IOdata , *(SPData32++));
break;
case 0x10 : /* 8 bit 工作模式*/
for(iTemp = 0 ; iTemp < SP_len ; iTemp++)
outportb(IOdata , *(SPData++));
break;
}
iow(TCR , 0x01); /* 开始进行传送*/
while((ior(ISR) & 0x02) == 0); /* 检查是否有传送完成*/
iow(ISR , 0x02); /* 清除传送完成中断*/
if ((ior(NSR) & 0x04) > 0) /* 传送完成为TSR1 or TSR2 */
{
if (ior(TSR1) == 0x00) return (1);
}
else
{
if (ior(TSR2) == 0X00) return (1);
}
return (0);
}
```

而以上面的范例，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);
```

七、如何接收封包

DM9000 接收到的封包，会存放于DM9000接收内存在于0C00h~3FFFh 中。若是读取位置超过3FFFh时，DM9000 会自动将位置移到0C00h 的位置。

在每一个封包，会有4 个Bytes 存放一些封包相关资料。第1 个Byte 是封包是否已存

放在接收内存，若值为“01h” 为封包已存放于接收内存，若为“00h” 则接收内存尚未有封包存放。在读取其它Byte 之前，必需要确定第1 个byte 是否为“01h”。第2 个Byte 则为这个封包的一些相关讯息，它他的格式像RSR 的格式。第3 和4 个Byte 是存放这个封包的长度大小。

4 个BYTE 封包讯息封包资料4 个BYTE 封包讯息封包资料

1.如何收接一个封包的流程：

(1) 检查现在是使用那一种IO 动作(Byte , Word , Dword)。

在未来将资料搬移的作动，皆要以此模式进行，并且此一模式，需和硬件一致。

RX_IN_CHK = ior (ISR) / 0Xfe; = 0 有封包接收= 1 无封包接收

(3) 确认封包是否为正常的封包：

此时运用到读取接收内存不移动位置MRCMDX。在使用MRCMDX 最好连续读取二次，以确保读取的封包为最新的封包。

ior(MRCMDX); ;;第一次读取到信息，先不理。

RX_P_CHK = ior(MRCMDX) & 0xff; ;; = 1 为正常封包， != 1 为异常封包*/

(4) 若是正常的封包，取得封包相关信息和长度：

此时运用到读取接收内存并移动位置MRCMD。这时取读时，其位置会随着使用MRCMD 的次数自

动移动，这一点和MRCMDX 不一样，请注意。

```
outportb( IOaddr, MRCMD ); ;;将位置指向MRCMD
I / O Byte Mode:
RX_Status = inportb(IOdata) | (inprotb(IOdata) << 8); ;;取得此一封包相关信息
RX_Length = inportb(IOdata) | (inp(IOdata) << 8); ;;取得此一封包的长度
I / O Word Mode:
RX_Status = inport( IOdata ); ;;取得此一封包相关信息
RX_Length = inport( IOdata ); ;;取得此一封包的长度
I / O DWord Mode:
(unsigned long int) tmp_data = inportl( IOdata ); ;;取得前四个byte 的信息
inport 为32 bit 输入指令
RX_Status = tmp_data & 0xFFFF; ;;转换取得此一封包相关信息
RX_Length = (tmp_data >> 16) & 0xFFFF; ;;转换取得此一封包长度
```

(5) 取得封包相关信息和长度，开始接收资料：

在你取得这个封包的信息、长度之后，再使用REG_F2 取得封包所包含的资料。

```
unsigned char RX_DATA[2048]; ;;系统内存放置接收回来封包的位置
unsigned char *rx_data8; ;;设定8 bit 资料指针
unsigned int *rx_data16; ;;设定16 bit 资料指针
unsigned long int *rx_data32; ;;设定32 bit 资料指针
outportb( IOaddr, MRCMD ); ;;将位置指向MRCMD
I / O Byte Mode:
rx_data8 = RX_DATA; ;;将指针指向接收数据的位置
for( i = 0 ; i < RX_Length ; i++ ) ;;将接收内存中的资料移到RX_DATA 中
*(rx_data8++) = inp( IOdata ); ;;每次一个byte
```

I / O Word Mode:

```
rx_data16 = RX_DATA; ;;将指针指向接收数据的位置
Tmp_Length = ( RX_Length + 1 ) / 2; ;;以word 方式，重新计算搬移资料次数
for( i = 0 ; i < Tmp_Length ; i++ ) ;;将接收内存中的资料移到RX_DATA 中
*(rx_data16++) = inp( IOdata ); ;;每次一个word
```

I / O DWord Mode:

```
rx_data32 = RX_DATA; ;;将指针指向接收数据的位置
Tmp_Length = ( RX_Length + 3 ) / 4; ;;以dword 方式，重新计算搬移资料次数
for( i = 0 ; i < Tmp_Length ; i++ ) ;;将接收内存中的资料移到RX_DATA 中
*(rx_data32++) = inp( IOdata ); ;;每次一个dword
```

我们也可以用函数来将程序变得更有可读性：

```
/* 检查内存是否有封包进入*/
unsigned char CheckNetPack(void)
{
DM9KREG_r(MRCMDX);
switch(DM9KREG_r(MRCMDX)) /* 检查内存是否还有封包*/
{
case 0x00 :
{
DM9KREG_w(ISR , 0x0c); /* 清除ISR 有关接收信息*/
return(0); /* 尚无封包进入*/
break;
}
case 0x01 : return(1);break; /* 有封包进入*/
default : DM9000_reset(); return(0); /* 内存出错，重置DM9000 */
}
}
/* DM9000 接收封包函数*/
unsigned int R_NetPack(unsigned char *RPData)
{
unsigned char RX_good , RX_status;
unsigned int iTemp , iTemp2 , RP_len;
unsigned int *RPData16;
unsigned long int *RPData32;
unsigned long int iTemp3;
/* 这边不用DM9KREG_r() 改用DM9Kaddr , DM9Kdata 增加速度，减少IO 动作*/
/* 读取FIFO 现在封包相关信息*/
RP_len = 0x0000;
outportb(DM9000_E.DM9Kaddr , MRCMD);
switch(DM9000_E.Work_mode)
{
case WORK_16 : /* 16 bit 工作模式*/
```

```
iTemp2 = inport(DM9000_E.DM9Kdata);
RX_good = iTemp2 &0xff ; /* 取回是否为好*/
RX_status = iTemp2 >> 8; /* 取得此一封包相关信息*/
RP_len = inport(DM9000_E.DM9Kdata); /* 取得此一封包长度*/
iTemp2 = (RP_len + 1) / 2; /* word 方式计算读取次数*/
RPData16 = RPData;
/* 将资料搬到系统内存之中*/
for(iTemp = 0x0000 ; iTemp < iTemp2 ; iTemp++ )
*(RPData16++) = inport(DM9000_E.DM9Kdata);
break;
case WORK_32 : /* 32 bit 工作模式*/
iTemp3 = inportl(DM9000_E.DM9Kdata);
RX_good = iTemp3 & 0xff ; /* 取回是否为好*/
RX_status = (iTemp3 >> 8) & 0xff; /* 取得此一封包相关信息*/
RP_len = (iTemp3 >> 16) & 0xffff; /* 取得此一封包长度*/
iTemp2 = (RP_len + 3) / 4; /* dword 方式计算读取次数*/
RPData32 = RPData;
/* 将资料搬到系统内存之中*/
for(iTemp = 0x0000 ; iTemp < iTemp2 ; iTemp++ )
*(RPData32++) = inportl(DM9000_E.DM9Kdata);
break;
case WORK_8 : /* 8 bit 工作模式*/
RX_good = inportb(DM9000_E.DM9Kdata); /* 取回是否为好*/
RX_status = inportb(DM9000_E.DM9Kdata); /* 取得此一封包相关信息*/
RP_len = inportb(DM9000_E.DM9Kdata) & 0x00ff; /* 取得此一封包长度*/
RP_len |= (inportb(DM9000_E.DM9Kdata) << 8) & 0xff00;
/* 将资料搬到系统内存之中*/
for(iTemp = 0x0000 ; iTemp < RP_len ; iTemp++ )
*(RPData++) = inportb(DM9000_E.DM9Kdata);
break;
}
if ((RX_status & 0xBF) != 0x00) /* 若封包中有其它错误讯息*/
return(0); /* 此一封包无效，长度0 */
else
return (RP_len); /* 回传封包长度，包含CRC */
}
```

而以上面的范例，使用下面命令：

```
unsigned char RX_data[1536] ;
unsigned int iTemp;
if (CheckNetPack() == 0x01)
iTemp = R_NetPack(RX_data);
if(iTemp > 0x00) printf(“ 封包接收完成 ”);
```

八、web 服务器的实现

8.1 完成一个小的 web 服务器。

一个小的 web 服务器需要如下驱动和一些协议的支持配合。

8.2 网站的制作

我们要先做一组想要的网页，由于不像 PC 那样资源丰富，我们做出来的网页要尽量小，有些可去可留的语句就去掉。作完后将这一组网页以二进制方式传送到 Linux 平台下的用户根目录，并创建一个目录 fs，将网页都拷贝到 fs 目录下，然后

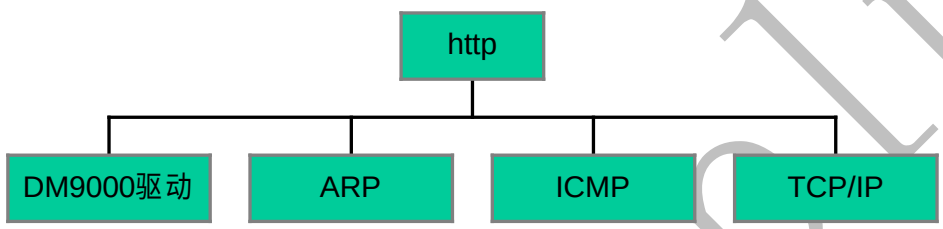

图 6 协议示意图

将 perl 脚本文件 makefsdata 拷贝到用户根目录下，并执行 perl makefsdata，会在用户根目录下生成一个 fsdata.c 文件，这个文件中就已经包含了所有网页上的数据，可以直接在本例中的 demo 中使用了。

8.3 Web Server 的实现

Web Server 的实现流程如下

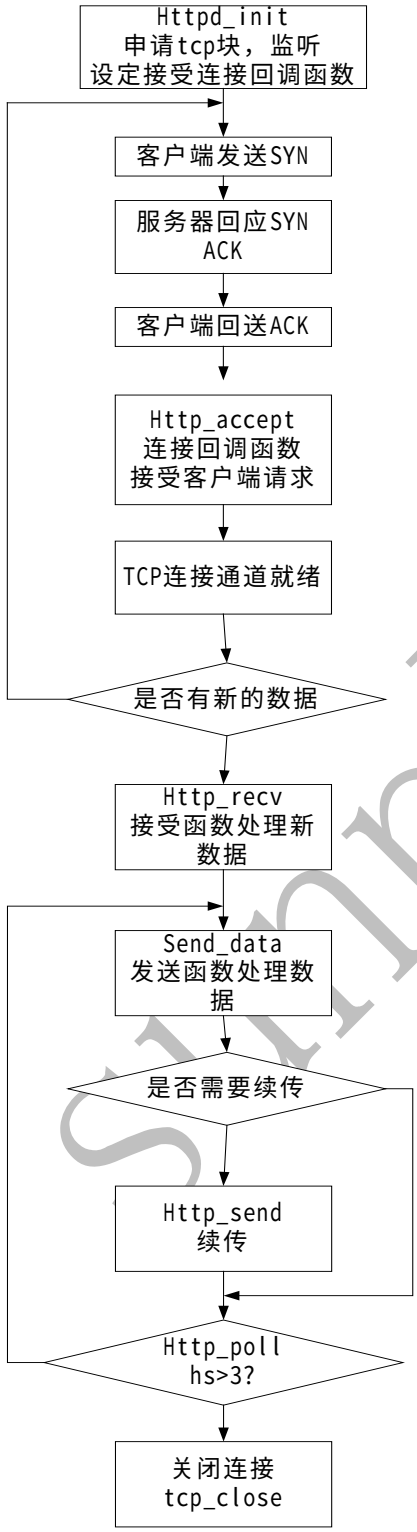

图 7 http 流程图

a、 服务器的初始化

服务器的初始化流程如下所示，执行完毕以后就已经开启了服务了，并且在服务端口监听连接请求，当有合法的客户端请求连接到达，程序会转入接受连接回调函数去进一步处理相关操作。

```
void
httpd_init(void)
{
  struct tcp_pcb *pcb;

  pcb = tcp_new();                    //申请新的 tcp 协议控制块
  tcp_bind(pcb, IP_ADDR_ANY, 80);     //将协议控制块与服务端口 80 绑定，这
里的
  pcb = tcp_listen(pcb);              //将端口置于监听状态
  tcp_accept(pcb, http_accept);       //设定好接受连接回调函数
  /*Initial CGI structures*/

  CGI_Init();                         //应用层的部分初始化操作
}
```

b、 连接的接受

服务器与客户端之间建立一次 TCP 连接需要三次握手，客户端发起连接请求（SYN），服务器收到后回应一个(SYN ACK)，然后客户端回送一个响应(ACK)，服务器收到 ACK 后就表示建立了一次连接。在第三次握手时，也就是在服务器收到客户端的 ACK 时，LwIP 会调用事先注册好的连接的接受回调函数，在本例中是 *http_accept()*。函数主体如下所示：

```
static err_t
http_accept(void *arg, struct tcp_pcb *pcb, err_t err)
{
  struct http_state *hs;
  tcp_setprio(pcb, TCP_PRIO_MIN);
  hs = mem_malloc(sizeof(struct http_state));
  if(hs == NULL) {
    return ERR_MEM;
  }
    /* Initialize the structure. */
  hs->file = NULL;
  hs->left = 0;
  hs->retries = 0;
    /* Tell TCP that this is the structure we wish to be passed for our
      callbacks. */
  tcp_arg(pcb, hs);
  /* Tell TCP that we wish to be informed of incoming data by a call
      to the http_recv() function. */
```

```
  tcp_recv(pcb, http_recv);            //接收回调函数
  tcp_err(pcb, conn_err);              //错误处理函数，一般释放资源，关闭连接
  tcp_poll(pcb, http_poll, 1);         //定时查寻函数，执行间隔为 1*slowtimer
  return ERR_OK;
}
```

c、 数据的接收

在客户端的请求被接受以后，就创建了一个 TCP 连接，客户端和服务器可以通过这条通道收发数据。客户端发送过来的 packet 若带有新的数据，LwIP 都会适时的调用数据的接收回调函数来处理这些数据。本例中注册的接收函数是 http_recv()，函数的主体如下所示：

```
err_t
http_recv(void *arg, struct tcp_pcb *pcb, struct pbuf *p, err_t err)
{
  int i;
  char *data;
  struct fs_file file;
  struct http_state *hs;
   int tempdata[58];
  hs = arg;
  if(err == ERR_OK && p != NULL) {

     /* Inform TCP that we have taken the data. */
    tcp_recved(pcb, p->tot_len);          //通知协议栈，应用层成功获取的数据长度
    ……                                    //处理接收到的数据，数据存放在以 p 开头的
                                          //packet 链中，总字节长度为 p->tot_len
      pbuf_free(p);                       //数据处理完毕以后必须释放 packet  p
    ……                                    //略去一些处理过程
    send_data(pcb, hs);                   //发送数据
    tcp_sent(pcb, http_sent);             //注册发送回调函数（用于续传）
      } else {
    pbuf_free(p);
    close_conn(pcb, hs);
      }
   } else {
      pbuf_free(p);
   }
  }

  if(err == ERR_OK && p == NULL) {
    close_conn(pcb, hs);
```

```
  }
  return ERR_OK;
}
```

d、数据的发送以及续传

数据的发送是通过调用 tcp_write()函数来完成的，由于涉及到窗口和发送缓冲区大小的问题，它的调用也要符合一定规则，下面是一个例子：

```
static void
send_data(struct tcp_pcb *pcb, struct http_state *hs)
{
  err_t err;
  u16_t len;
  /* We cannot send more data than space available in the send
      buffer. */
    if(tcp_sndbuf(pcb) < hs->left) {    //判断待发送数据长度是否大于发送缓冲区
      len = tcp_sndbuf(pcb);
  } else {
      len = hs->left;
  }

do {
 err = tcp_write(pcb, hs->file, len, 0); //将数据写入发送队列，参数 0 表示无拷贝
      if(err == ERR_MEM) {
        len /= 2;
      }
  } while(err == ERR_MEM && len > 0);

  if(err == ERR_OK) {
      hs->file += (len >> 1);          //修改待发送数据指针（指针是以字为单位的）
      hs->left -= len;                 //修改待发送数据长度（长度是以字节为单位的）
  }
}
```

预发送的数据应该用一个数据结构来存储，数据结构包括一个数据指针来指向预发送的数据区域首地址，一个长度成员，用来表示预发送的数据长度。当预发送数据的长度大于发送缓冲区大小的时候，就不可能一次将所有待发送数据放置到发送队列中去，这个时候就需要续传。在前面申请的函数 http_sent 就是用来完成续传的。当有新的数据成功的发送出去并且收到客户端的确认后，LwIP 会调用用户注册的发送函数，来发送也许还未发送完毕的数据，这个时候一般发送缓冲区的大小会较大。在 http_sent 函数中可以从

回调参数中获得足够的信息，以确定该发送哪些数据，下面是函数的主体：

```
static err_t
http_sent(void *arg, struct tcp_pcb *pcb, u16_t len)
{
  struct http_state *hs;
  hs = arg;
  hs->retries = 0;
    if(hs->left > 0) {
    send_data(pcb, hs);            //续传一次数据
  } else {
    close_conn(pcb, hs);           //传输完毕关闭连接
  }
  return ERR_OK;
}
```

e、 数据的重传

这里的重传是发生在应用层的事情，因为一旦数据打包交给了 TCP 层，就不再需要应用层去管理一切传输事宜了。当在应用层发送数据的时候，会按照前面的步骤来做，在第 d 步的时候，若发送缓冲区为空，则 TCP 层不会接收来自应用层的数据的。在应用回调 API 函数的情形下，只能利用 LwIP 提供的周期查询函数在另一时刻来尝试重新发送数据，当重试次数大于某一个设定值（3）的时候，强行关闭这个连接。在本例中，这一查询函数为 http_poll，函数主体为：

```
static err_t
http_poll(void *arg, struct tcp_pcb *pcb)
{
  struct http_state *hs;
  hs = arg;
    if(hs == NULL) {
    tcp_abort(pcb);
    return ERR_ABRT;
  } else {
    ++hs->retries;
    if(hs->retries == 3) {
    if (hs->ram) mem_free(hs->head);
    mem_free(hs);                          //释放应用层所占资源
      tcp_abort(pcb);                      //关闭连接
      return ERR_ABRT;
    }
   send_data(pcb, hs);                     //尝试重新发送数据
  }
  return ERR_OK;
}
```

重传的频率为在 b 步中设定的时间，在本例中为 1 个单位的慢时钟长度。具体计算一下，所使用的快时钟是 8Hz，慢始终为 8/5Hz，那么可以计算出来为每 0.625 重试一次。

f、 连接的终止

在关闭连接之前，释放所有的资源，例如所有应用层申请的内存等等，释放完毕后调用 tcp_close()结束连接。

TCP 客户端的软件设计

TCP 客户端的设计大部分与服务器端相同，区别比较大的地方是在连接的初始化上，服务器是等候由客户端来发起连接，而客户端是主动发起一次连接。连接建立起来之后的数据的收发以及续传重传等等，两者就比较一致了。下面以 POP3 的客户端为例说明如何建立一次连接：

```
void pop3_init(struct ip_addr *sevip)
{
  struct tcp_pcb *pcb;
// IP4_ADDR(&sevip, 172,20,5,42);
  pcb = tcp_new();
    tcp_connect(pcb, sevip, 110, pop_connected);        //发起与 POP3 服务器的连接
                                                        //并注册连接建立回调函数
}
```

发起到服务器的连接后，客户会发出 SYN 到服务器，服务器回应 SYN ACK，这时 LwIP 会调用连接建立回调函数，在本例中是 pop_connected，本函数中可以进一步完成一些应用层的初始化操作，以及注册一些其他的回调函数。

```
err_t pop_connected(void *arg, struct tcp_pcb *pcb,err_t err)
{
  struct pop3_state *ps;
  ps = arg;
  ps = mem_malloc(sizeof(struct pop3_state));
  pop3c_init(ps);
  pop3c_changestate(POP3C_CONNECTION_OPENED, ps);
/* Tell TCP that this is the structure we wish to be passed for our
   callbacks. */
  tcp_arg(pcb, ps);                          //注册回调参数
  tcp_sent(pcb, pop3_sent);                  //注册续传函数
/* Tell TCP that we wish to be informed of incoming data by a call
   to the http_recv() function. */
  tcp_recv(pcb, pop3_recv);                  //注册接收回调函数
// tcp_err(pcb, pop3_err);
  tcp_poll(pcb, pop3cd_run, 1);              //注册定时轮寻函数
return ERR_OK;
  }
```

执行完毕连接建立回调函数后 LwIP 会返回一个 ACK 到服务器端，至此一个完整 TCP 连接在客户端与服务器端建立起来了。

8.4 模拟家电处理部分。

为了配合网页完成远程控制和远程监控，用一个 061 来模拟家电，在收到网络终端过来的控制命令后，会把一个 I/O 口置相应状态，网页刷新时会扫描相应 I/O 口，以完成远程监控，这样就形成了一个反馈回路，可很好的完成它的功能。

九、注意事项

★ 要提供足够的电源，建议使用两个电源供电 。

★ 采用多电源供电时，请注意共地，否则几个模块之间没有共同的参考电平，将无法协同完成工作。

★ 数据线不能过长。

十、控制板原理图

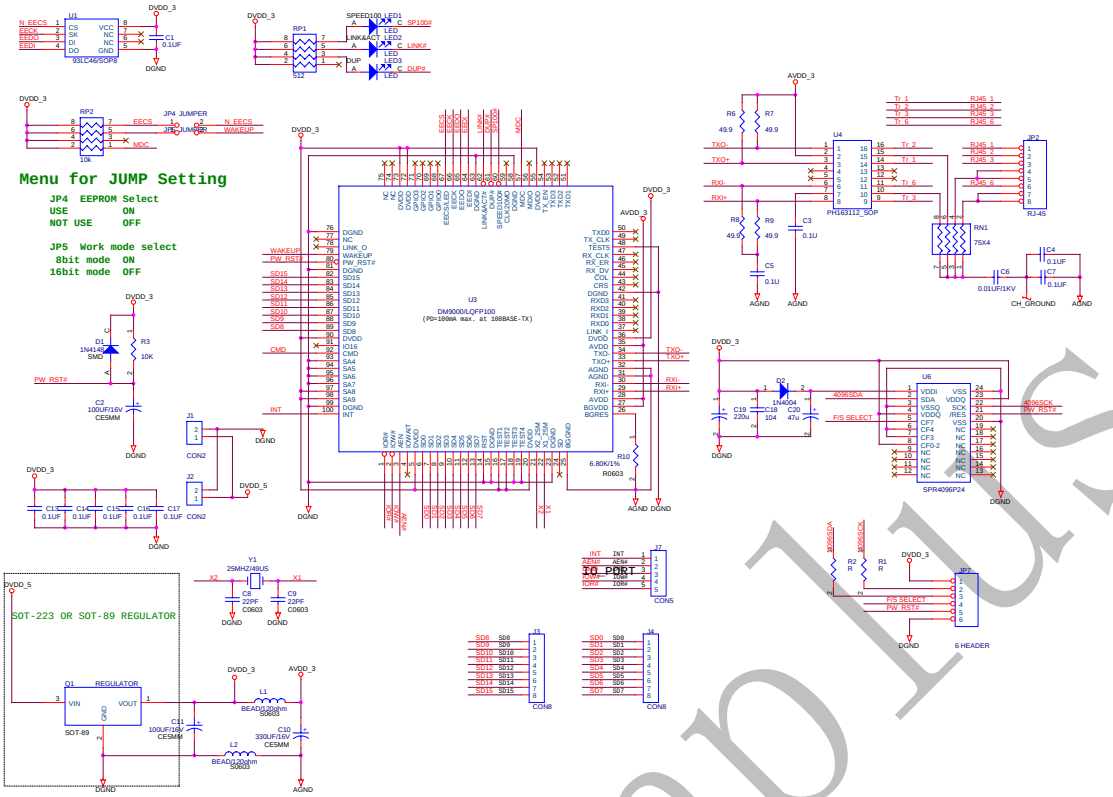

图 8 硬件原理图