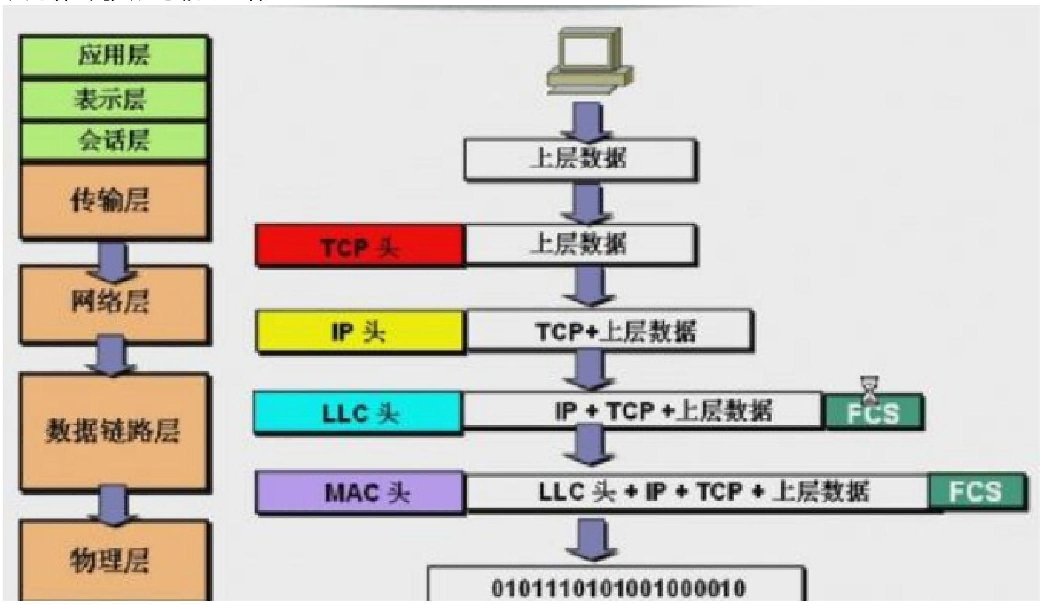
专题18-网卡搭建新通道

一、网卡工作原理

1.1、网络模型

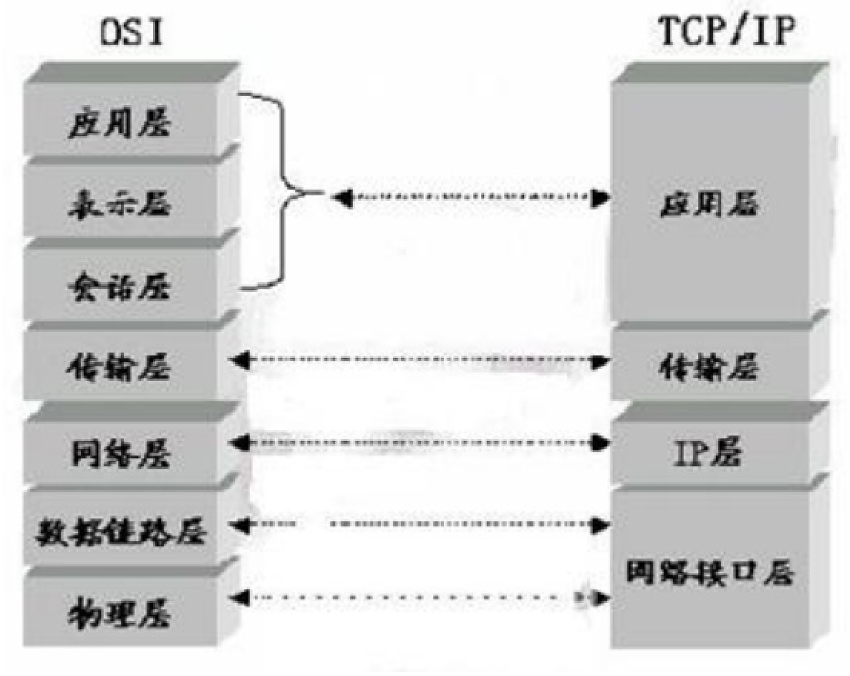
1.1.1、OSI七层模型

OSI(Open System Interconnection)，开放式系统互联参考模型 。它把网络协议从逻辑上分为了7层。通过七个层次使不同的系统网络之间实现可靠的通讯。



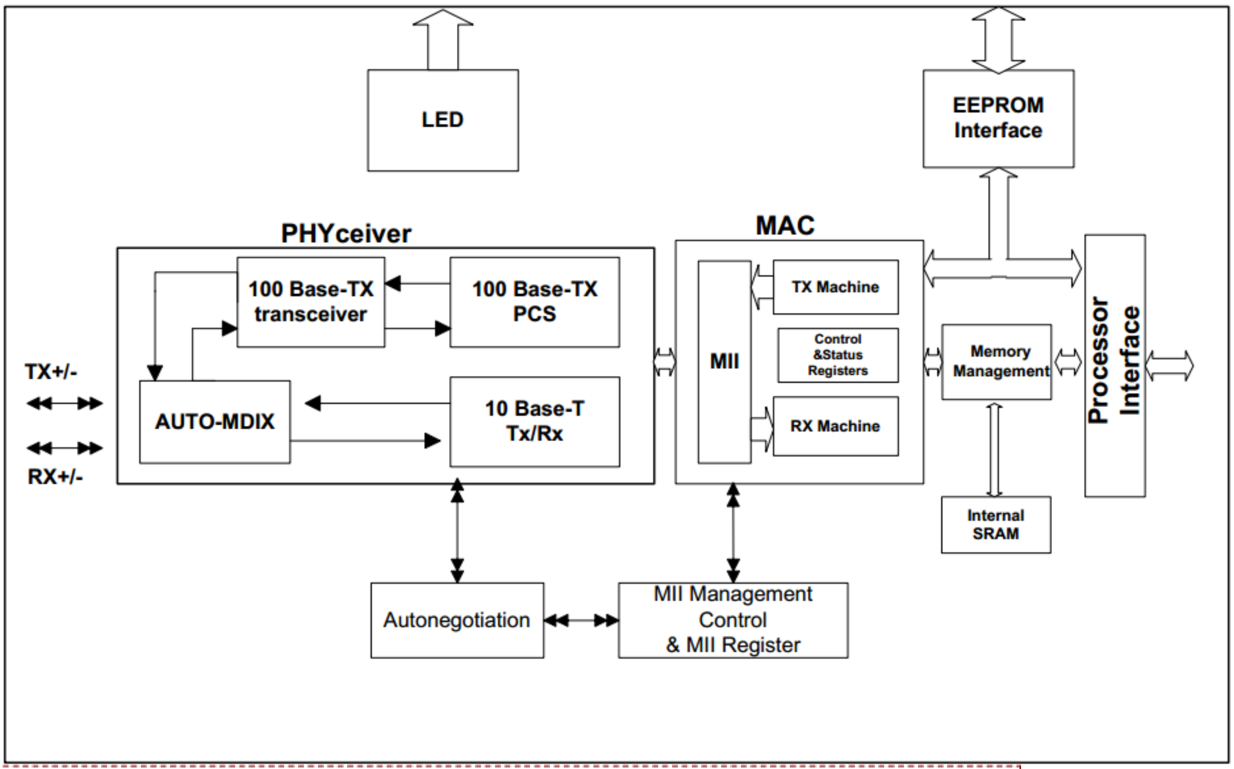
1.1.2、Linux四层模型

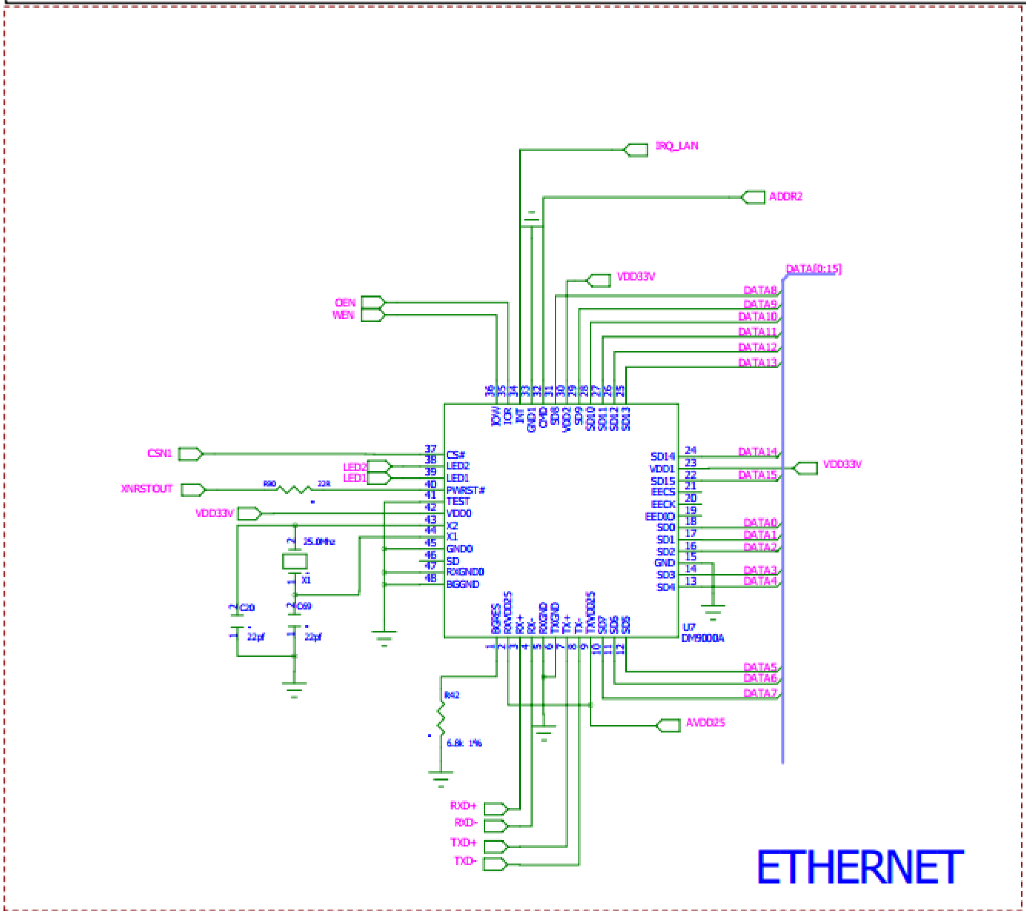
OSI参考模型的过于 庞大、复杂招致了许多批评。与此对照，由技术人员自己开发的TCP/IP协议栈获得了更为广泛的应用。

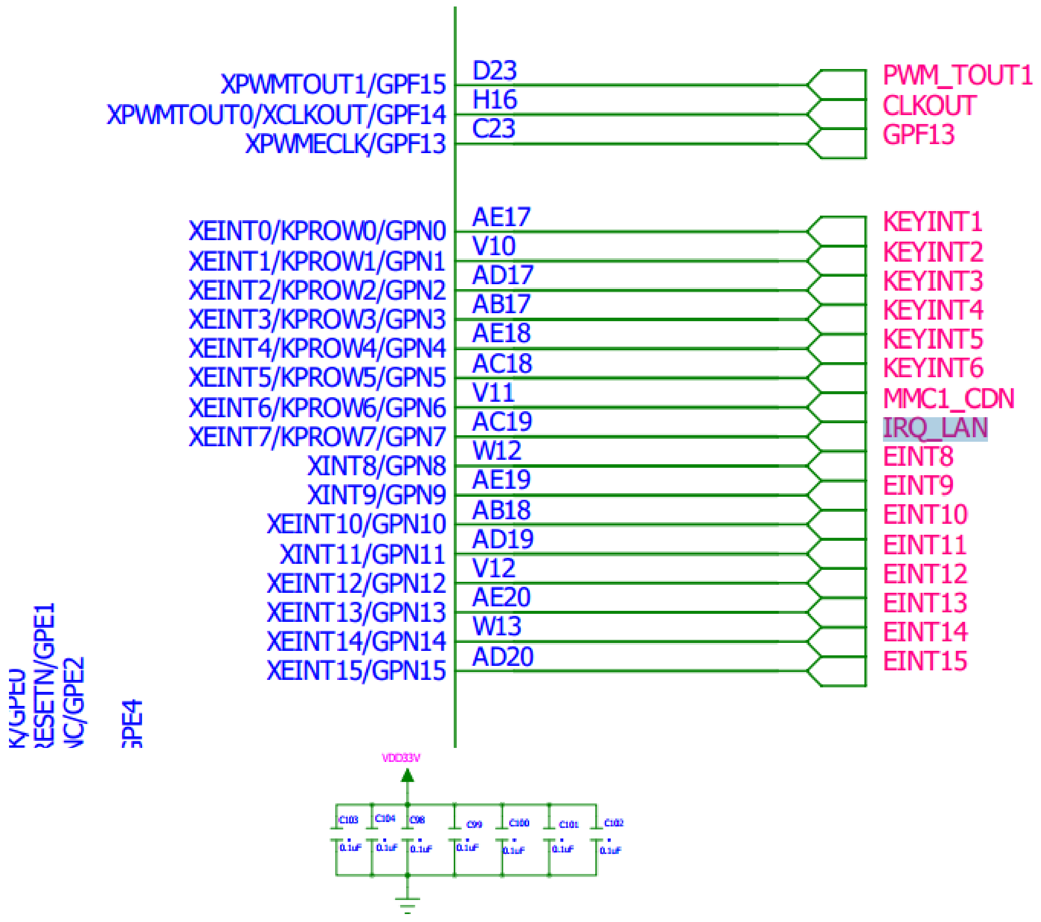


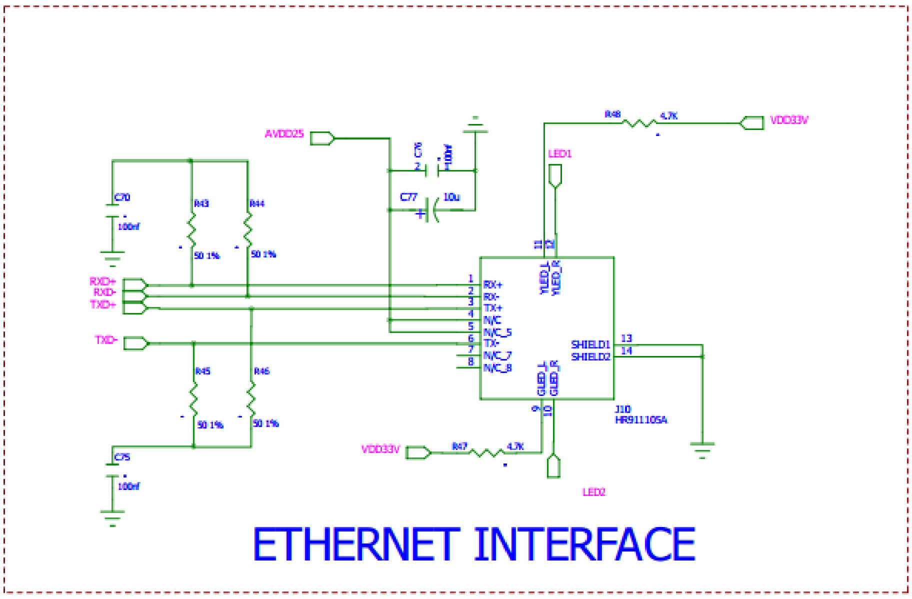
1.2、网卡硬件结构

1.2.1、硬件总体架构

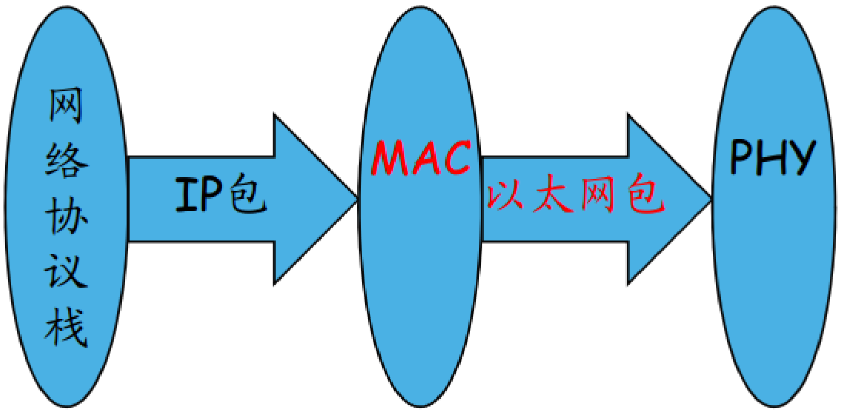

LED
EEPROM Interface
PHYceiver
MAC
100 Base-TX transceiver
100 Base-TX PCS
AUTO-MDIX
10 Base-T Tx/Rx
TX+/-
RX+/-
MII
TX Machine
Control &Status Registers
RX Machine
Memory Management
Internal SRAM
Processor Interface
Autonegotiation
MII Management Control & MII Register


IRQ_LAN
ADDR2
VDD33V
DATA[0:15]
OEN
WEN
CSN1
XNRSTOUT
VDD33V
LED2
LED1
25.0Mhz
X1
C20
C69
22pf
22pf
R42
6.8k 1%
U7
DM9000A
AVDD25
RXD+
RXD-
TXD+
TXD-
ETHERNET


XPWMTOUT1/GPF15 D23 PWM_TOUT1
XPWMTOUT0/XCLKOUT/GPF14 H16 CLKOUT
XPWMECLK/GPF13 C23 GPF13
XEINT0/KPROW0/GPN0 AE17 KEYINT1
XEINT1/KPROW1/GPN1 V10 KEYINT2
XEINT2/KPROW2/GPN2 AD17 KEYINT3
XEINT3/KPROW3/GPN3 AB17 KEYINT4
XEINT4/KPROW4/GPN4 AE18 KEYINT5
XEINT5/KPROW5/GPN5 AC18 KEYINT6
XEINT6/KPROW6/GPN6 V11 MMC1_CDN
XEINT7/KPROW7/GPN7 AC19 IRQ_LAN
XINT8/GPN8 W12 EINT8
XINT9/GPN9 AE19 EINT9
XEINT10/GPN10 AB18 EINT10
XINT11/GPN11 AD19 EINT11
XEINT12/GPN12 V12 EINT12
XEINT13/GPN13 AE20 EINT13
XEINT14/GPN14 W13 EINT14
XEINT15/GPN15 AD20 EINT15
RESETN/GPE1
NC/GPE2
GPE4
VDD33V
C103 C104 C98 C99 C100 C101 C102
0.1uF

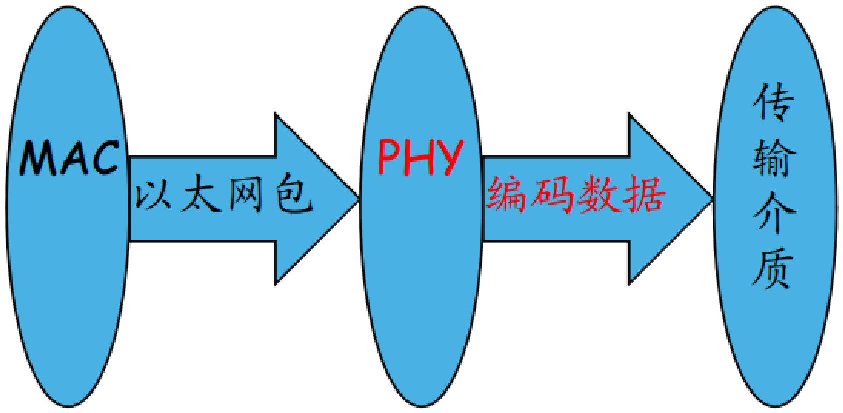

AVDD25
VDD33V
R48 4.7K
LED1
C76 100nf
C77 10u
C70 100nf
R43 R44
50 1%
RXD+
RXD-
TXD+
TXD-
RX+
RX-
TX+
N/C
N/C_5
TX-
N/C_7
N/C_8
YLED_L
YLED_R
GLED_L
GLED_R
SHIELD1
SHIELD2
J10
HR911105A
R45 R46
C75 100nf
R47 4.7K
LED2
ETHERNET INTERFACE

1.2.2、MAC


网络协议栈
IP包
MAC
以太网包
PHY

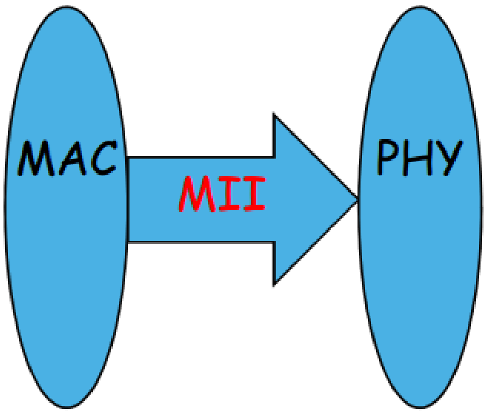
MAC主要负责数据帧的构建、数据差错检 查、传送控制等。

1.2.3、PHY



PHY是物理接口收发器，属于物理层，当它收 到MAC过来的数据时，它会去加上校验码，然后按照物理层的规则进行数据编码，再发送到传输介质上。接收过程则相反

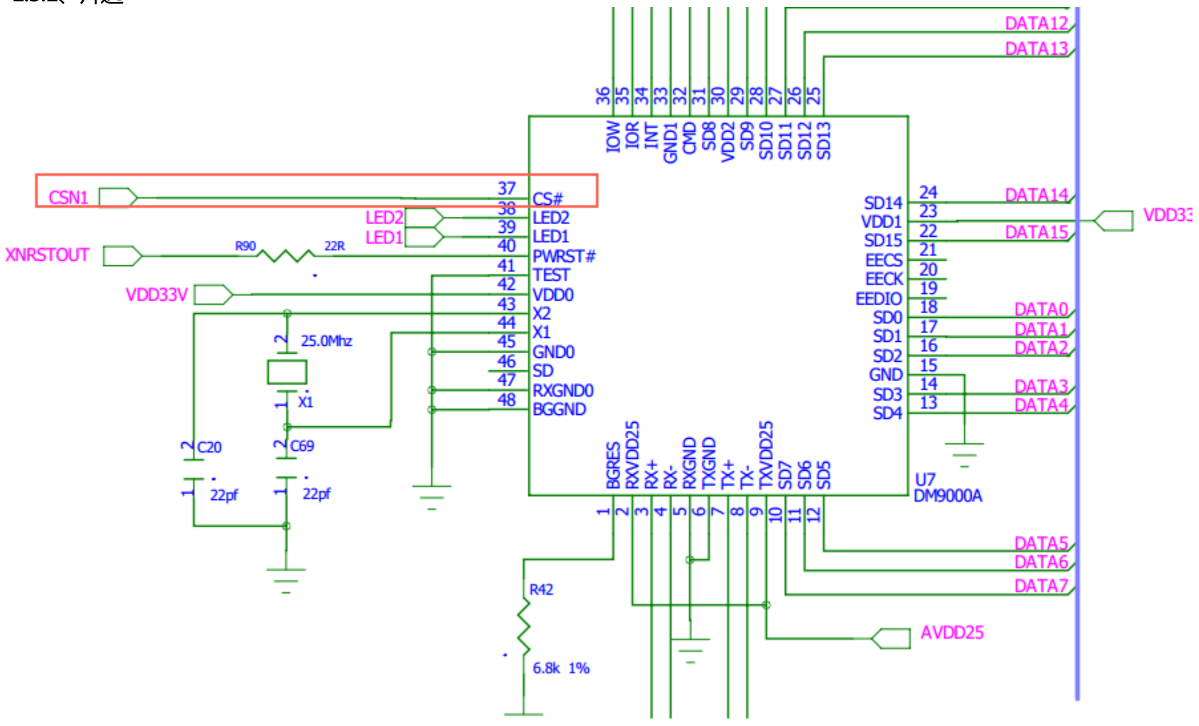
1.2.4、MII接口

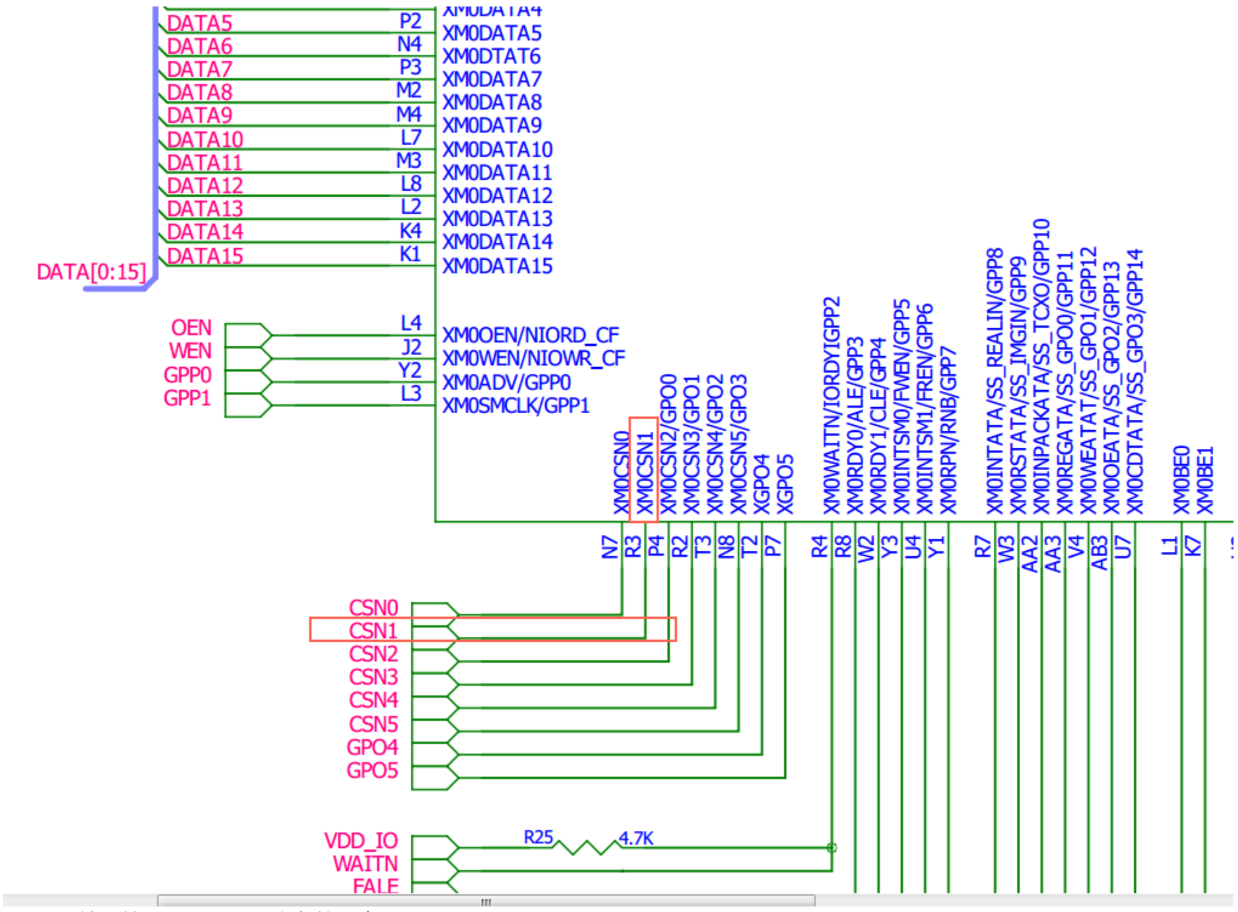


MII：媒体独立接口, “媒体独立”表明MAC一定情况下，任何类型的PHY设备都可以正常工作。

1.3、DM9000工作特性

1.3.1、片选

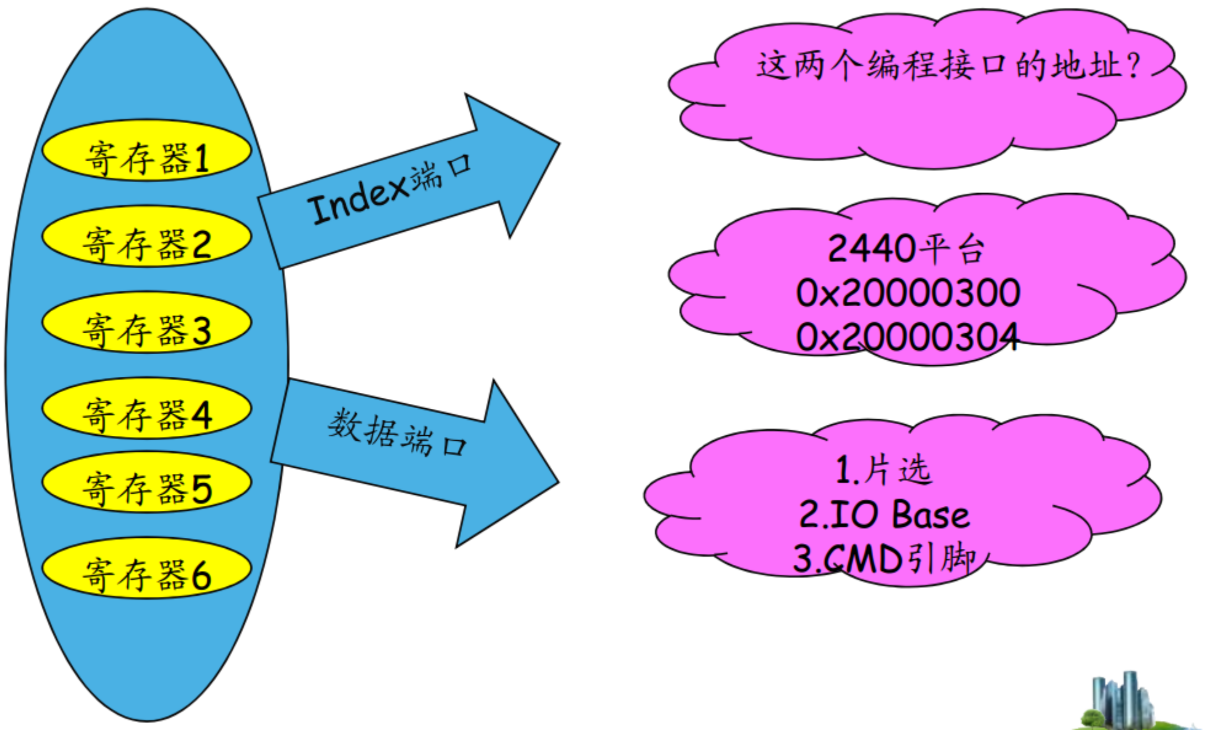




使用的是XM0CSN1,内存位置为：

```
#define CONFIG_DRIVER_DM9000 1 /* we have a SMC9115 on-board */
#define CONFIG_DM9000_BASE 0x18000300 /*XMOCSN1*/
#define DM9000_DATA 0x18000304 /*ADDR2*/
#define DM9000_IO CONFIG_DM9000_BASE
#define CONFIG_DM9000_USE_16BIT 1
```

1.3.2、对外接口



二、 DM9000驱动程序设计（移植）直接使用的u-boot里的dm9000x.h文件

```
#include "type.h"
#include "dm9000x.h"
```

```
#include "arp.h"

#define CONFIG_DM9000_BASE      0x18000300 /*XMOCSN1*/
#define DM9000_DATA             0x18000304 /*ADDR2*/
#define DM9000_IO               CONFIG_DM9000_BASE
#define CONFIG_DM9000_USE_16BIT 1
#define DM9000_BASE             CONFIG_DM9000_BASE
#define DM9000_PPTR             (*((volatile unsigned short *)(DM9000_IO)))
#define DM9000_PDATA            (*((volatile unsigned short *)(DM9000_DATA)))

#define SROM_BW                 (*((volatile unsigned long *)0x70000000))
#define SROM_BC1                (*((volatile unsigned long *)0x70000008))

/* ----------------------------------------------------------------------- */
#define DM9000_Tacs     (0x0)   // 0clk         address set-up
#define DM9000_Tcos     (0x4)   // 4clk         chip selection set-up
#define DM9000_Tacc     (0xE)   // 14clk        access cycle
#define DM9000_Tcoh     (0x1)   // 1clk         chip selection hold
#define DM9000_Tah      (0x4)   // 4clk         address holding time
#define DM9000_Tacp     (0x6)   // 6clk         page mode access cycle
#define DM9000_PMC      (0x0)   // normal(1data)page mode configuration
/* ----------------------------------------------------------------------- */

#define GPNCON                  (*((volatile unsigned long *)0x7F008830))

#define DM9000_ID               0x90000A46
#define DM9KS_REG05             (RXCR_Discard_LongPkt|RXCR_Discard_CRCPkt)
#define DM9KS_DISINTR           IMR_SRAM_antoReturn


unsigned char * buffer = &arpbuf;

unsigned char host_mac_addr[6] = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff};
unsigned char mac_addr[6] = {9,8,7,6,5,4};
//unsigned char host_ip_addr[4] = {10,100,15,99};
//unsigned char ip_addr[4] = {10,100,15,202};

unsigned char host_ip_addr[4] = {192,168,1,100};
unsigned char ip_addr[4] = {192,168,1,202};

unsigned short packet_len;

```

2.1、配置SROM BANK 4 位宽和时序

REGISTER	ADDRESS	R/W	DESCRIPTION	RESET VALUE
MEM_SYS_CFG	0x7E00_F120	R/W	Memory Subsystem configuration register	0x0000_0080
RESERVED	0x7E00_F124	R/W	RESERVED	0x0000_0000
QOS_OVERRIDE1	0x7E00_F128	R/W	DMC1 QOS Override register	0x0000_0000
MEM_CFG_STAT	0x7E00_F12C	R	Memory Subsystem setup status register	0x0000_0000

MEM_SYS_CFG	BIT	DESCRIPTION	RESET VALUE
RESERVED	[31:15]	RESERVED	0x0000_0
INDEP_CF	[14]	Use CF interface independently. 0 = Use memory port 0 shared by EBI. 1 = Use independent CF interface.	0
RESERVED	[13]	Should be '0'	0
BUS_WIDTH	[12]	Select initial state of SROMC CS0 memory bus width. 0 = 8-bit data width. 1 = 16-bit data width. If NOR booting (OM[4:1] = 0101) is selected, this setting is ignored and 16-bit data width is selected. Even this bit is set to 0 or 1, this selects only reset value of DataWidth0 of SROM_BW SFR in SROMC. Bus width of CS0 for SROMC follows DataWidth0 setting.	0

[illegible] should be set to 0.

	MP0_CS_CFG						
	[5]	[4]	[3]	[2]	[1]	[0]	
Xm0CSn[0]	-	-	-	-	-	-	SROMC CS0
Xm0CSn[1]	-	-	-	-	-	-	SROMC CS1
Xm0CSn[2]	-	-	-	-	1	-	SROMC CS2
	-	-	-	-	0	-	OneNANDC CS0
	-	-	-	-	0	-	NFCON CS0
Xm0CSn[3]	-	-	1	-	-	-	SROMC CS3
	-	-	0	-	-	-	OneNANDC CS1
	-	-	0	-	-	-	NFCON CS1
Xm0CSn[4]	-	0	-	-	-	-	SROMC CS4
	-	1	-	-	-	-	CFCON CS0
Xm0CSn[5]	0	-	-	-	-	-	SROMC CS5
	1	-	-	-	-	-	CFCON CS1

6.6.1 SROM BUS WIDTH & WAIT CONTRL REGISTER(SROM_BW)

Register	Address	R/W	Description	Reset Value
SROM_BW	0x70000000	R/W	SROM Bus width & wait control	0x0000_000x

SROM_BW	Bit	Description	Initial State
Reserved	[31:24]	Reserved	0
ByteEnable5	[23]	nWBE / nBE(for UB/LB) control for Memory Bank5 0 = Not using UB/LB (XrnWBE[1:0] is dedicated nWBE[1:0]) 1 = Using UB/LB (XrnWBE[1:0] is dedicated nBE[1:0]	0
WaitEnable5	[22]	Wait enable control for Memory Bank5 0 = WAIT disable 1 = WAIT enable	0
Reserved	[21]	Reserved	0
DataWidth5	[20]	Data bus width control for Memory Bank5	0

DataWidth5	[20]	Data bus width control for Memory Bank5 0 = 8-bit 1 = 16-bit	0
ByteEnable4	[19]	nWBE / nBE(for UB/LB) control for Memory Bank4 0 = Not using UB/LB (XrnWBE[1:0] is dedicated nWBE[1:0]) 1 = Using UB/LB (XrnWBE[1:0] is dedicated nBE[1:0]	0
WaitEnable4	[18]	Wait enable control for Memory Bank4 0 = WAIT disable 1 = WAIT enable	0
Reserved	[17]	Reserved	0
DataWidth4	[16]	Data bus width control for Memory Bank4 0 = 8-bit 1 = 16-bit	0
ByteEnable3	[15]	nWBE / nBE(for UB/LB) control for Memory Bank3 0 = Not using UB/LB (XrnWBE[1:0] is dedicated nWBE[1:0]) 1 = Using UB/LB (XrnWBE[1:0] is dedicated nBE[1:0]	0
WaitEnable3	[14]	Wait enable control for Memory Bank3 0 = WAIT disable 1 = WAIT enable	0
Reserved	[13]	Reserved	0
DataWidth3	[12]	Data bus width control for Memory Bank3 0 = 8-bit 1 = 16-bit	0
ByteEnable2	[11]	nWBE / nBE(for UB/LB) control for Memory Bank2 0 = Not using UB/LB (XrnWBE[1:0] is dedicated nWBE[1:0]) 1 = Using UB/LB (XrnWBE[1:0] is dedicated nBE[1:0]	0
WaitEnable2	[10]	Wait enable control for Memory Bank2 0 = WAIT disable 1 = WAIT enable	0

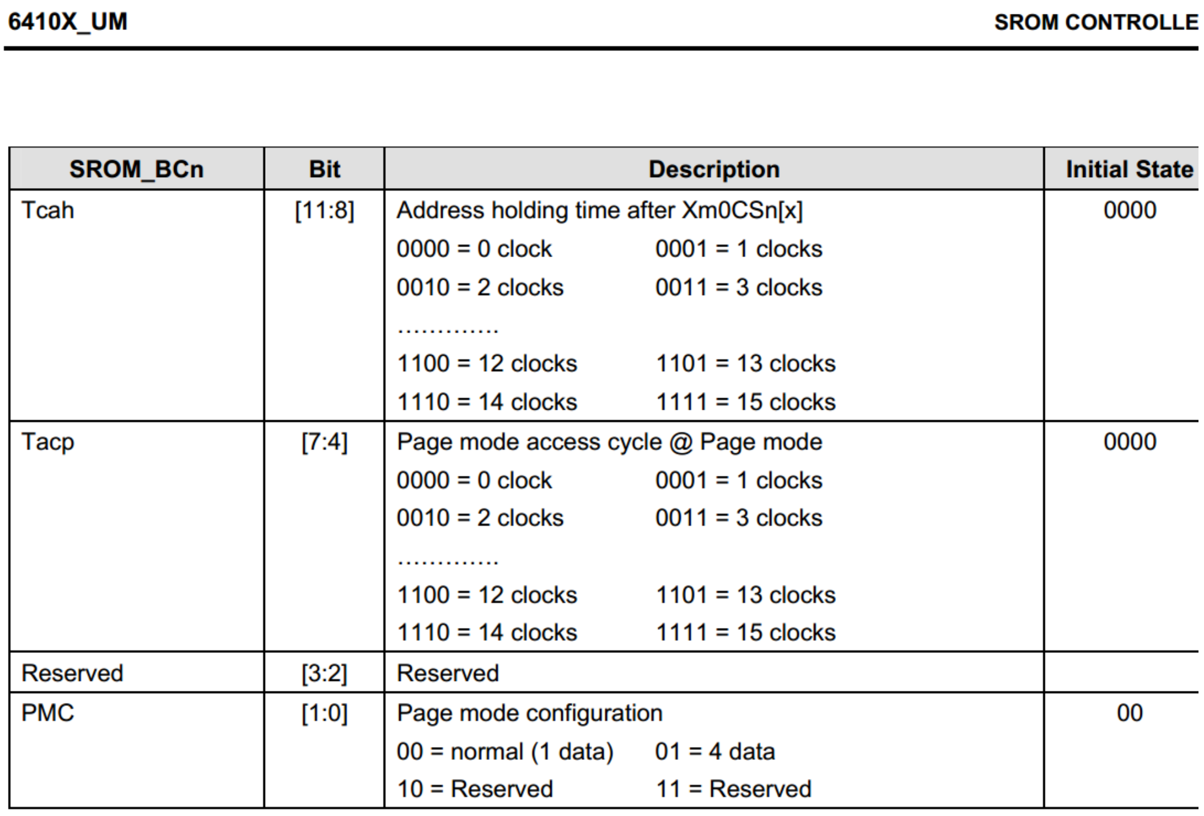
SROM_BW	Bit	Description	Initial State
Reserved	[9]	Reserved	0
DataWidth2	[8]	Data bus width control for Memory Bank2 0 = 8-bit 1 = 16-bit	0
ByteEnable1	[7]	nWBE / nBE(for UB/LB) control for Memory Bank1 0 = Not using UB/LB (XrnWBE[1:0] is dedicated nWBE[1:0]) 1 = Using UB/LB (XrnWBE[1:0] is dedicated nBE[1:0]	0
WaitEnable1	[6]	Wait enable control for Memory Bank1 0 = WAIT disable 1 = WAIT enable	0
Reserved	[5]	Reserved	0
DataWidth1	[4]	Data bus width control for Memory Bank1 0 = 8-bit 1 = 16-bit	0
ByteEnable0	[3]	nWBE / nBE(for UB/LB) control for Memory Bank0 0 = Not using UB/LB (XrnWBE[1:0] is dedicated nWBE[1:0]) 1 = Using UB/LB (XrnWBE[1:0] is dedicated nBE[1:0]	0
WaitEnable0	[2]	Wait enable control for Memory Bank0 0 = WAIT disable 1 = WAIT enable	0

Reserved	[1]	Reserved	0
DataWidth0	[0]	Data bus width control for Memory Bank0. Reset value is configured by OM setting. 0 = 8-bit 1 = 16-bit	H/W Set

6.6.2 SROM BANK CONTROL REGISTER (SROM_BC : XRCSN0 ~ XRCSN2)

Register	Address	R/W	Description	Reset Value
SROM_BC0	0x70000004	R/W	SROM Bank0 control register	0x000F_0000
SROM_BC1	0x70000008	R/W	SROM Bank1 control register	0x000F_0000
SROM_BC2	0x7000000C	R/W	SROM Bank2 control register	0x000F_0000
SROM_BC3	0x70000010	R/W	SROM Bank3 control register	0x000F_0000
SROM_BC4	0x70000014	R/W	SROM Bank4 control register	0x000F_0000
SROM_BC5	0x70000018	R/W	SROM Bank5 control register	0x000F_0000

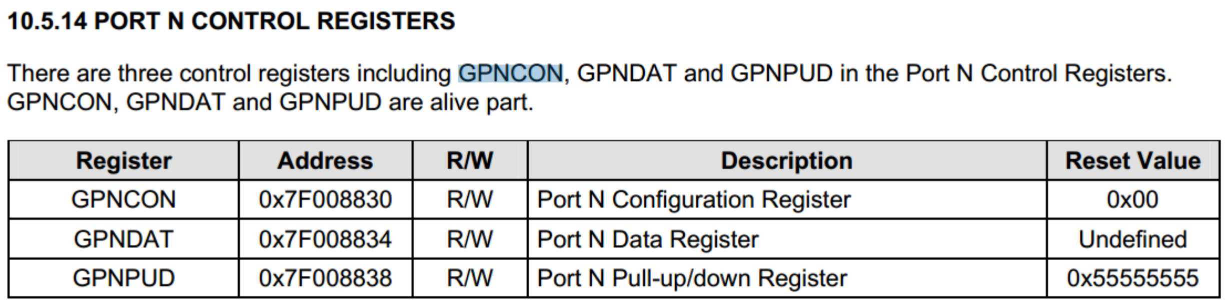
SROM_BCn	Bit	Description	Initial State
Tacs	[31:28]	Adress set-up before Xm0CSn[x] 0000 = 0 clock 0001 = 1 clocks 0010 = 2 clocks 0011 = 3 clocks …………. 1100 = 12 clocks 1101 = 13 clocks 1110 = 14 clocks 1111 = 15 clocks	0000
Tcos	[27:24]	Chip selection set-up before nOE 0000 = 0 clock 0001 = 1 clocks 0010 = 2 clocks 0011 = 3 clocks …………. 1100 = 12 clocks 1101 = 13 clocks 1110 = 14 clocks 1111 = 15 clocks	0000
Reserved	[23:21]	Reserved	000
Tacc	[20:16]	Access cycle 00000 = 1 clock 00001 = 2 clocks 00010 = 3 clocks 00011 = 4 clocks …………. 11100 = 29 clocks 11101 = 30 clocks 11110 = 31 clocks 11111 = 32 clocks	01111
Tcoh	[15:12]	Chip selection hold on nOE 0000 = 0 clock 0001 = 1 clocks 0010 = 2 clocks 0011 = 3 clocks …………. 1100 = 12 clocks 1101 = 13 clocks 1110 = 14 clocks 1111 = 15 clocks	0000



SROM_BCn	Bit	Description	Initial State
Tcah	[11:8]	Address holding time after Xm0CSn[x] 0000 = 0 clock 0001 = 1 clocks 0010 = 2 clocks 0011 = 3 clocks …………. 1100 = 12 clocks 1101 = 13 clocks 1110 = 14 clocks 1111 = 15 clocks	0000
Tacp	[7:4]	Page mode access cycle @ Page mode 0000 = 0 clock 0001 = 1 clocks 0010 = 2 clocks 0011 = 3 clocks …………. 1100 = 12 clocks 1101 = 13 clocks 1110 = 14 clocks 1111 = 15 clocks	0000
Reserved	[3:2]	Reserved	
PMC	[1:0]	Page mode configuration 00 = normal (1 data) 01 = 4 data 10 = Reserved 11 = Reserved	00

```
void dm9000_cs_init(void)
{
	int SROM_BW_value, SROM_BC1_value;

	 //Data bus width control for Memory Bank1
	SROM_BW &= (~(0xf<<4));
	SROM_BW |= (0x1<<4);
	SROM_BW_value = SROM_BW;^M
	printf("\n\r SROM_BW_value is : %d \n\r", SROM_BW_value);

	//SROM BANK CONTROL REGISTER
	SROM_BC1 = ((DM9000_Tacs<<28)+(DM9000_Tcos<<24)+(DM9000_Tacc<<16)+(DM9000_Tcoh<<12)+
(DM9000_Tah<<8)+(DM9000_Tacp<<4)+(DM9000_PMC));
	SROM_BC1_value = SROM_BC1;^M
	printf("\n\r SROM_BC1_value is : %d \n\r", SROM_BC1_value);
}

```

2.2、设置DM9000中断引脚



10.5.14 PORT N CONTROL REGISTERS

There are three control registers including GPNCON, GPNDAT and GPNPUD in the Port N Control Registers. GPNCON, GPNDAT and GPNPUD are alive part.

Register	Address	R/W	Description	Reset Value
GPNCON	0x7F008830	R/W	Port N Configuration Register	0x00
GPNDAT	0x7F008834	R/W	Port N Data Register	Undefined
GPNPUD	0x7F008838	R/W	Port N Pull-up/down Register	0x55555555

GPNCON	Bit	Description	Initial State
GPN0	[1:0]	00 = Input 01 = Output 10 = Ext. Interrupt[0] 11 = Key pad ROW[0]	00
GPN1	[3:2]	00 = Input 01 = Output 10 = Ext. Interrupt[1] 11 = Key pad ROW[1]	00
GPN2	[5:4]	00 = Input 01 = Output 10 = Ext. Interrupt[2] 11 = Key pad ROW[2]	00
GPN3	[7:6]	00 = Input 01 = Output 10 = Ext. Interrupt[3] 11 = Key pad ROW[3]	00
GPN4	[9:8]	00 = Input 01 = Output 10 = Ext. Interrupt[4] 11 = Key pad ROW[4]	00
GPN5	[11:10]	00 = Input 01 = Output 10 = Ext. Interrupt[5] 11 = Key pad ROW[5]	00
GPN6	[13:12]	00 = Input 01 = Output 10 = Ext. Interrupt[6] 11 = Key pad ROW[6]	00
GPN7	[15:14]	00 = Input 01 = Output 10 = Ext. Interrupt[7] 11 = Key pad ROW[7]	00

```
void dm9000_int_init(void)
{
	//set GPN7 as EINT7
	GPNCON &= ~(0x3 << 14);
	GPNCON |= (0x2 << 14);
}

```

2.3、DM9000初始化（参考u-boot程序）

2.3.1、DM9000寄存器读写函数

```
static void iow(unsigned short reg, unsigned short data)
{
	DM9000_PPTR = reg;
	DM9000_PDATA = data;
}

static unsigned char ior(int reg)
{
	DM9000_PPTR = reg;
	return DM9000_PDATA;
}

```

2.3.2、DM9000复位函数

```
void dm9000_reset(void)
{
	/* set the internal PHY power-on, GPIOs normal, and wait 20ms */
	iow(DM9KS_GPR, GPR_PHYUp);
	mdelay(20); /* wait for PHY power-on ready */
	iow(DM9KS_GPR, GPR_PHYDown);/* Power-Down PHY */
	mdelay(1000);   /* compatible with rtl8305s */
	iow(DM9KS_GPR, GPR_PHYUp);
	mdelay(20);/* wait for PHY power-on ready */

	iow(DM9KS_NCR, NCR_MAC_loopback|NCR_Reset);
	udelay(20);/* wait 20us at least for software reset ok */
	iow(DM9KS_NCR, NCR_MAC_loopback|NCR_Reset);
	udelay(20);/* wait 20us at least for software reset ok */

}

```

2.3.3、DM9000 ID 获取函数

```
static u32 GetDM9000ID(void)
{
	u32	id_val;
	DM9000_PPTR = DM9KS_PID_H;
	id_val = (DM9000_PDATA & 0xff) << 8;
	DM9000_PPTR = DM9KS_PID_L;
	id_val += (DM9000_PDATA & 0xff);
	id_val = id_val << 16;

	DM9000_PPTR = DM9KS_VID_H;
	id_val += (DM9000_PDATA & 0xff) << 8;
	DM9000_PPTR = DM9KS_VID_L;
	id_val += (DM9000_PDATA & 0xff);

	return id_val;
}

```

2.3.4、DM9000捕获函数

```
static void dm9000_capture (void)
{
	u32 ID;

	 ID = GetDM9000ID();
	 if ( ID != DM9000_ID) {
		printf("not found the dm9000 ID:%x\n",ID);
		return;
		} else
		printf("found DM9000 ID:%x\n",ID);

}

```

2.3.5、DM9000MAC初始化函数

```
static void dm9000_mac_init(void)
{
	iow(DM9KS_NCR, 0);
	iow(DM9KS_TCR, 0);/* TX Polling clear */
	iow(DM9KS_BPTR, 0x30|JPT_600us);/* Less 3kb, 600us */
	iow(DM9KS_SMCR, 0);/* Special Mode */
	iow(DM9KS_NSR, 0x2c);/* clear TX status */
	iow(DM9KS_ISR, 0x0f);/* Clear interrupt status */
	iow(DM9KS_TCR2, TCR2_LedMode1);/* Set LED mode 1 */
}

```

2.3.6、DM9000MAC地址填充函数

```
static void dm9000_mac_fill(void)
{
	int i;
	for (i = 0; i < 6; i++) {
		iow(DM9KS_PAR + i, mac_addr[i]);
	}
}

```

2.3.7、DM9000激活函数

```
static void dm9000_activate(void)
{
	/* Activate DM9000A/DM9010 */
	iow(DM9KS_RXCR, DM9KS_REG05 | RXCR_RxEnable);
	iow(DM9KS_IMR, DM9KS_DISINTR);
}

```

2.3.8、DM9000模块初始化函数

```
void dm9000_init(void)
{
	int status;

	//chip select dm9000
	dm9000_cs_init();

	//dm9000 interrupt init
	dm9000_int_init();

	//reset dm9000
	dm9000_reset();

	status = ior(DM9KS_BPTR);
	printf("\n\r NSR register status is : %d\n\r",status);

	//capture dm9000
	dm9000_capture();

	//mac init
	dm9000_mac_init();

	//fill mac address
	dm9000_mac_fill();

	//activate dm9000
	dm9000_activate();
}

```

2.4、DM9000网络数据发送函数

```
extern int eth_send (volatile unsigned char *packet, int length)
{
	int length1 = length;
	int i;

	/*disable interrupt */
	iow(DM9KS_IMR, 0x80);

	/* set packet length  */
	iow(DM9KS_TXPLH, (length1 >> 8) & 0xff);
	iow(DM9KS_TXPLL, length1 & 0xff);

	DM9000_PPTR = DM9KS_MWCMD;/* data copy ready set */
	for (i = 0; i < length; i += 2) {
		DM9000_PDATA = packet[i] | (packet[i+1] << 8);
	}

	/* start transmit */
	iow(DM9KS_TCR, TCR_TX_Request);

	/* wait for tx complete */
	while (1) {
		if (ior(DM9KS_NSR)& (NSR_TX2END|NSR_TX1END))
			break;
	}

	/*clear TX statu */
	iow(DM9KS_NSR, 0x2c);

	/*enable rx interrupt */
	iow(DM9KS_IMR, 0x81);

	return 0;

}

```

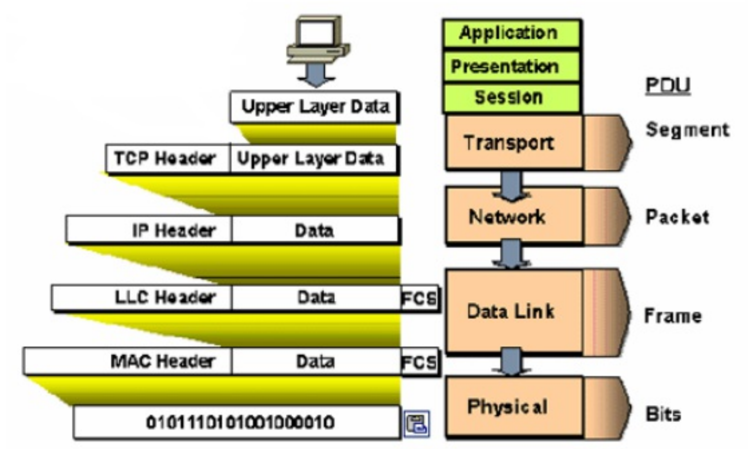
2.4、DM9000网络数据接收函数

```
#define PKT_MAX_LEN	1522
extern int eth_rx (unsigned char * data)
```

```
{
	u8 RxRead;
	u8 status, len;
	u16 tmp;
	int i;

	/*whether is rx interrupt,and clear interrupt statu */
	RxRead = ior(DM9KS_MRCMDX);
	RxRead = ior(DM9KS_ISR);
	RxRead = ior(DM9KS_MRCMDX) & 0xff;

	if (RxRead != 1)  /* no data */
		return 0;
	else
		iow(DM9KS_IMR, 0x01);

	DM9000_PPTR = DM9KS_MRCMD; /* set read ptr ++ */

	/*read statu */
	status = DM9000_PDATA;

	/*read length */
	len = DM9000_PDATA;

	if (len < PKT_MAX_LEN) {
		for (i = 0; i < len; i+=2) {
			tmp = DM9000_PDATA;
			data[i] = tmp & 0xff;
			data[i+1] = (tmp >> 8) & 0xff;
		}
	}

}

void dm9000_int_isr(void)
{
	int i;
	packet_len = eth_rx (buffer);
	arp_process();
}
```

三、ARP协议实现

3.1、以太网通讯



在计算机网络中，数据发送的过程，就是一个把数据按照各层协议层层封装的过程。在这个过程中，最终要使用的协议通常是以太网协议（数据链路层协议）。

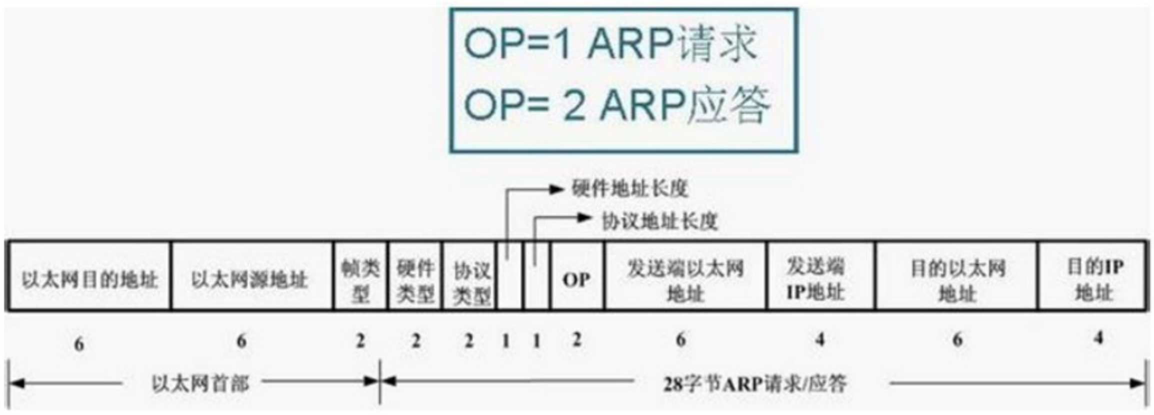
3.2、以太网包格式

目的MAC地址（6）	源MAC地址（6）	类型（2）	数据（46~1500）	CRC（4）

目的MAC地址：接收者的物理地址
源MAC地址：发送者的物理地址
类型：标明高层的数据使用的协议类型
数据：高层的数据
CRC：校验码

3.3、ARP功能描述

在以太网络中，每台计算机的唯一身份标示是MAC地址（物理层的地址），两台计算机要进行通讯，也必须知道对方的MAC地址，但是用户通常只知道对方的IP地址，这个时候，就可以利用ARP（地址解析协议）来向局域网中的所有计算机发送ARP请求包，收到请求包且满足条件的计算机将回复ARP应答包，告知其MAC地址。所以ARP协议是一种利用IP地址或者MAC地址的协议。



ARP包分为请求包和应答包，通过OP字段来区别。

3.4、网络字节序

为了统一网络传输格式，网络包采用的大端模式，但地址不需要改变（即只改变ARP请求包里双字节的数据）。

3.5、ARP实现程序设计

3.5.1、构造ARP网络包

```
#ifndef ARP_H_
#define ARP_H_

typedef struct eth_hdr
{
        unsigned char d_mac[6];
        unsigned char s_mac[6];
        unsigned short type;
}ETH_HDR;

typedef struct arp_hdr
{
        ETH_HDR ethhdr;
        unsigned short hwtype;
        unsigned short protocol;
        unsigned char hwlen;
        unsigned char protolen;
        unsigned short opcode;
        unsigned char smac[6];
        unsigned char sipaddr[4];
        unsigned char dmac[6];
        unsigned char dipaddr[4];
}ARP_HDR;

ARP_HDR arpbuf;

extern unsigned char * buffer;

extern unsigned char host_mac_addr[6];
extern unsigned char mac_addr[6];
extern unsigned char host_ip_addr[4];
extern unsigned char ip_addr[4];
extern unsigned short packet_len;

#endif /* ARP_H_  */

```

3.5.2、发送ARP请求包

```
#include "arp.h"

#define HON(n) (((n & 0xff) << 8 ) | ((n & 0xff00) >> 8))

/*
 * send arp request packet
 */

static int ARP_FLAG;

void arp_request(void)
{
         /*Constructs an arp request packet */
        memcpy(arpbuf.ethhdr.d_mac, host_mac_addr, 6);
        memcpy(arpbuf.ethhdr.s_mac, mac_addr, 6);
        arpbuf.ethhdr.type = HON(0x0806);

        arpbuf.hwtype = HON(1);
        arpbuf.protocol = HON(0x0800);
        arpbuf.hwlen = 6;
        arpbuf.protolen = 4;
        arpbuf.opcode = HON(1);
        memcpy(arpbuf.smac, mac_addr, 6);
        memcpy(arpbuf.sipaddr, ip_addr, 4);
        //arpbuf.dmac[6]
        memcpy(arpbuf.dipaddr, host_ip_addr, 4);

        packet_len = 14 + 28;

         /*call eth_send function */
        eth_send(buffer, packet_len);
}


```

3.5.3、处理ARP应答包，并提取MAC地址

```
/*
 * Parse arp response packet, extract MAC
 */
unsigned char arp_process(void)
{
        int i;

	if (packet_len < 28)
		return 0;

	memcpy(host_ip_addr, arpbuf.sipaddr, 4);
	printf("\n\r host ip is : ");
	for (i = 0; i < 4; i++) {
		printf("%03d", host_ip_addr[i]);
		printf(".");
		}
	printf("\n\r");

	memcpy(host_mac_addr, arpbuf.smac, 6);
	printf("\n\r host mac is : ");
	for (i = 0; i < 6; i++) {
		printf("%02x", host_mac_addr[i]);
		printf("-");
		}
	printf("\n\r");

	ARP_FLAG = 0;
}

void dm9000_arp(void)
{
	ARP_FLAG = 1;
	arp_request();
	//while (ARP_FLAG) ;
}
```