

第七章

SAMSUNG S3C2410 通信与接口实验

本章将介绍几个简单的基础开发实例，既能帮助读者掌握 MDK 嵌入式软件的基本开发过程，同时也能让读者了解 ARM 处理器的基本结构、指令集、存储系统以及基本接口编程。

本书所有的实例均在 Embest EduKit-IV (选用 S3C2410 处理器子板) 教学实验平台上运行通过。Embest EduKit-IV 是一款功能强大的 32 位的嵌入式开发板，可任意选用以下处理器子板：ARM9 的三星的 S3C2410A 芯片、ARM10 的 Intel 的 XScale 芯片和 Cortex M3 芯片。该实验板除了提供键盘、LED、LCD、触摸屏和串口等一些常用的功能模块外，还具有 IDE 硬件接口、PCI 接口、CF 存储卡接口、以太网接口、USB 接口、IrDA 接口、IIS 接口、SD 卡接口以及步进电机等丰富的接口模块，并可根据用户要求扩展 GPRS 等模块。Embest EduKit-IV 实验平台能给用户在 32 位 ARM 嵌入式领域进行实验和开发提供丰富的支持和极大的便利。

7. 1 IIC 读写 EEPROM 实验

7. 1. 1 实验目的

- Ø 通过实验掌握 IIC 串行数据通信协议的使用；
- Ø 通过实验掌握 EEPROM 器件的读写访问方法；
- Ø 通过实验掌握 S3C2410X 处理器的 IIC 控制器的使用。

7. 1. 2 实验设备

- Ø 硬件：Embest EduKit-IV 平台，ULINK2 仿真器套件，PC 机；
- Ø 软件：µVision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

7. 1. 3 实验内容

- Ø 编写程序对实验板上 EEPROM 器件 AT24C02 进行读写访问；
- Ø 写入 EEPROM 某一地址，再从该地址读出，输出到超级终端；
- Ø 把读出内容和写入内容进行比较，检测 S3C2410X 处理器通过 IIC 接口，是否可以正常读写 EEPROM 器件 AT24C02。

7. 1. 4 实验原理

1. IIC 接口以及 EEPROM

IIC 总线为同步串行数据传输总线，其标准总线传输速率为 100kb/s，增强总线可达 400kb/s。总线驱动能力为 400pF。S3C2410X RISC 微处理器能支持多主 IIC 总线串行接口。下图为 IIC 总线的内部结构框图。

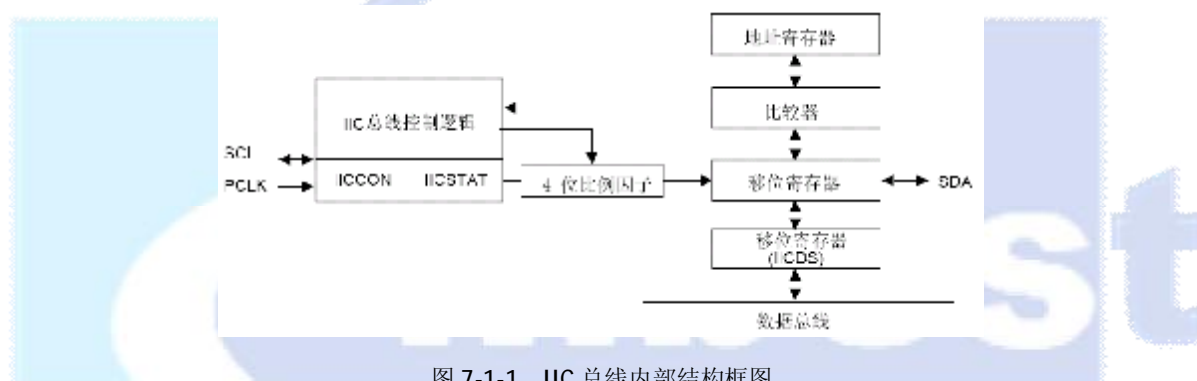


图 7-1-1 IIC 总线内部结构框图

IIC 总线可构成多主和主从系统。在多主系统结构中，系统通过硬件或软件仲裁获得总线控制使用权。应用系统中 IIC 总线多采用主从结构，即总线上只有一个主控节点，总线上的其它设备都作为从设备。IIC 总线上的设备寻址由器件地址接线决定，并且通过访问地址最低位来控制读写方向。

目前，通用存储器芯片多为 EEPROM，其常用的协议主要有两线串行连接协议（IIC）和三线串行连接协议。带 IIC 总线接口的 EEPROM 有许多型号，其中 AT24CXX 系列使用十分普遍。产品包括 AT2401/02/04/08/16 等，其容量（字节数 x 页）分别为 128x8/256x8/512x8/1024x8/2048x8，适用于 2V~5V 的低电压的操作。具有低功耗和高可靠性等优点。

AT24 系列存储器芯片采用 CMOS 工艺制造，内置有高压泵，可在单电压供电条件下工作。其标准封装为 8 脚 DIP 封装形式，如图 7-1-2。

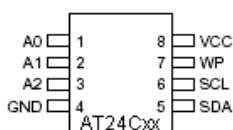


图 7-1-2 AT24 系列 EEPROM 的 DIP8 封装示意图

各引脚的功能说明如下:

SCL ---- 串行时钟。遵循 ISO/IEC7816 同步协议; 漏极开路, 需接上拉电阻。

在该引脚的上升沿, 系统将数据输入到每个 EEPROM 器件, 在下降沿输出。

SDA ---- 串行数据线; 漏极开路, 需接上拉电阻。

双向串行数据线, 漏极开路, 可与其他开路器件“线或”。

A0、A1、A2 ---- 器件/页面寻址地址输入端。

在 AT24C01/02 中, 引脚被硬连接; 其他 AT24Cxx 均可接寻址地址线。

WP ---- 读写保护。

接低电平时可对整片空间进行读写; 高电平时不能读写受保护区。

Vcc/GND ---- 一般输入+5V 的工作电压。

2. IIC 总线的读写控制逻辑

开始条件 (START_C) 在开始条件下, 当 SCL 为高电平时, SDA 由高转为低。

停止条件 (STOP_C) 在停止条件下, 当 SCL 为高电平时, SDA 由低转为高。

确认信号 (ACK) 在接收方应答下, 每收到一个字节后便将 SDA 电平拉低。

数据传送 (Read/Write)

IIC 总线启动或应答后 SCL 高电平期间数据串行传送; 低电平期间为数据准备, 并允许 SDA 线上数据电平变换。总线以字节 (8bit) 为单位传送数据, 且高有效位(MSB)在前。IIC 数据传送时序如图 7-1-3 所示:

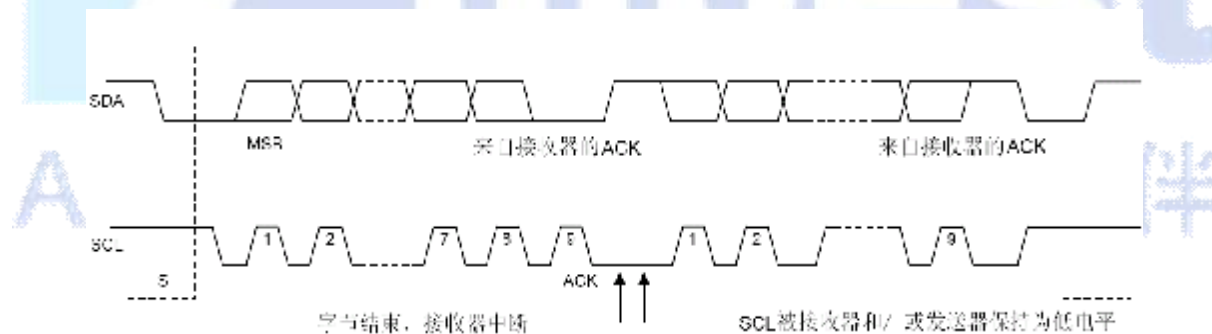


图 7-1-3 IIC 总线信号的时序

3. EEPROM读写操作

本实验平台设计了 IIC 总线扩展接口, 用于各 IIC 通信设备, 也通过扩展接口提供给用户定制模块。本实验中使用 S3C2410X 处理器内置的 IIC 控制器作为 IIC 通信主设备, AT24C02 EEPROM 为从设备。电路设计如图 7-1-4 所示:

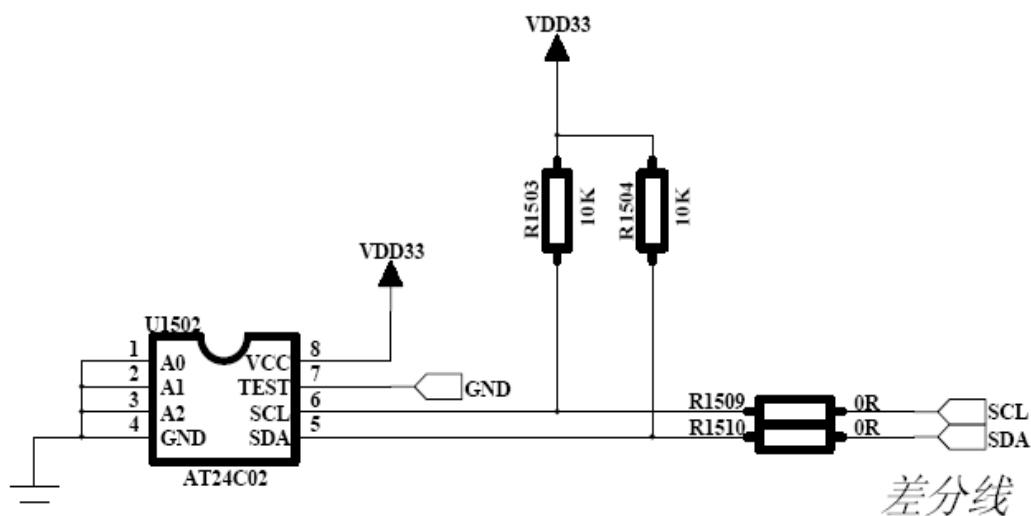


图 7-1-4 AT24C02 控制电路

AT24C02 的存储容量为 256×8 个字节 (2K)，器件地址是 1010，A0、A1、A2 三位地址线决定了芯片的访问地址与要访问的部件。在本实验平台中，A0、A1、A2 分别接地，故在本实验平台中只能操作 256 个字节的存储单元。

AT24C02 由输入缓冲器和 EEPROM 阵列组成。由于 EEPROM 的半导体工艺特性写入时间为 5-10ms，如果从外部直接写入 EEPROM，每写一个字节都要等候 5-10ms，成批数据写入时则要等候更长的时间。具有 SRAM 输入缓冲器的 EEPROM 器件，其写入操作变成对 SRAM 缓冲器的装载，装载完后启动一个自动写入逻辑将缓冲器中的全部数据一次写入 EEPROM 阵列中。对缓冲器的输入称为页写，缓冲器的容量称为页写字节数。AT24C02 的页写字节数为 8，占用最低 3 位地址。写入不超过页写字节数时，对 EEPROM 器件的写入操作与对 SRAM 的写入操作相同；若超过页写字节数时，应等候 5-10ms 后再启动一次写操作。

由于 EEPROM 器件缓冲区容量较小（只占据最低 3 位），且不具备溢出进位检测功能，所以，从非零地址写入 8 个字节数或从零地址写入超过 8 个字节数会形成地址翻卷，导致写入出错。

(1) AT24C02 写操作

AT24C02 支持字节写和页写两种模式。

在字节写模式下，主器件发送起始命令和从器件地址信息（R/W 位置零）给从器件，在从器件产生应答信号后，主器件发送 AT24C02 的字节地址，主器件在收到从器件的另一个应答信号后，再发送数据到被寻址的存储单元。AT24C02 再次应答，并在主器件产生停止信号后开始内部数据的擦写，在内部擦写过程中，AT24C02 不再应答主器件的任何请求。如图 7-1-5，是 AT24C02 的字节写模式的时序图。

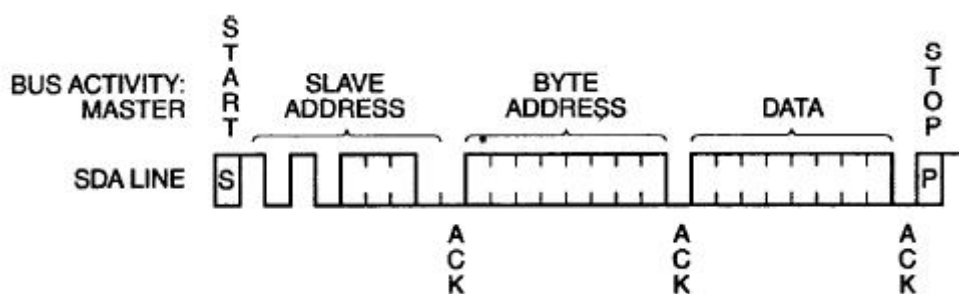


图 7-1-5 AT24C02 的字节写时序图

在页写模式下，AT24C02 可以一次写入 8 个字节的数据。页写操作的启动和字节写一样，不同在于传送了一字节数据后并不产生停止信号，主器件被允许发送 7 个额外的字节，每发送一个字节数据后 AT24C02 产生一个应答位并将字节地址低位加 1，高位保持不变。如果在发送停止信号之前，主器件发送超过 8 个字节，地址计数器将自动翻转，先前写入的数据被覆盖。接收到 8 个字节数据和主器件发送的停止信号后，AT24C02 启动内部写周期将数据写到数据区，所有接收的数据在一个写周期内写入 AT24C02。如图 7-1-6，是 AT24C02 的页写模式的时序图。

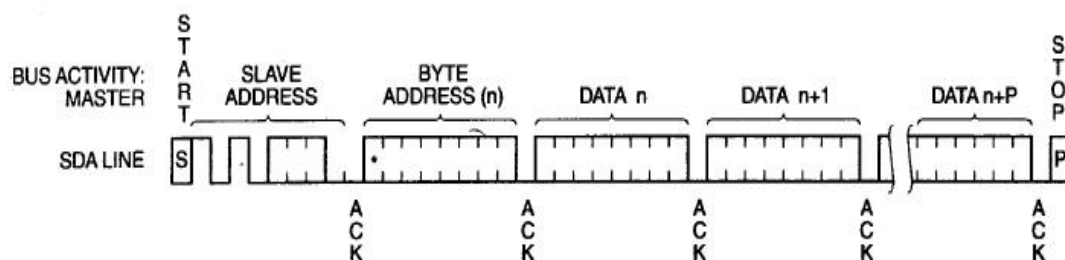


图 7-1-6 AT24C02 的页写时序图

(2) AT24C02 读操作

AT24C02 支持立即地址读、选择读和连续读三种模式。

在立即地址读模式下，AT24C02 的地址计数器内容为最后操作字节的地址加 1，如是上次读/写的操作地址为 N ，则立即读的地址为 $N+1$ 。如果 $N=255$ ，则计数器将翻转到 0 且继续输出数据。AT24C02 接收到从器件地址信号后（R/W 位置 1），它首先发送一个应答信号，然后发送一个 8 位字节数据。主器件不需要发送一个应答信号，但要产生一个停止信号。

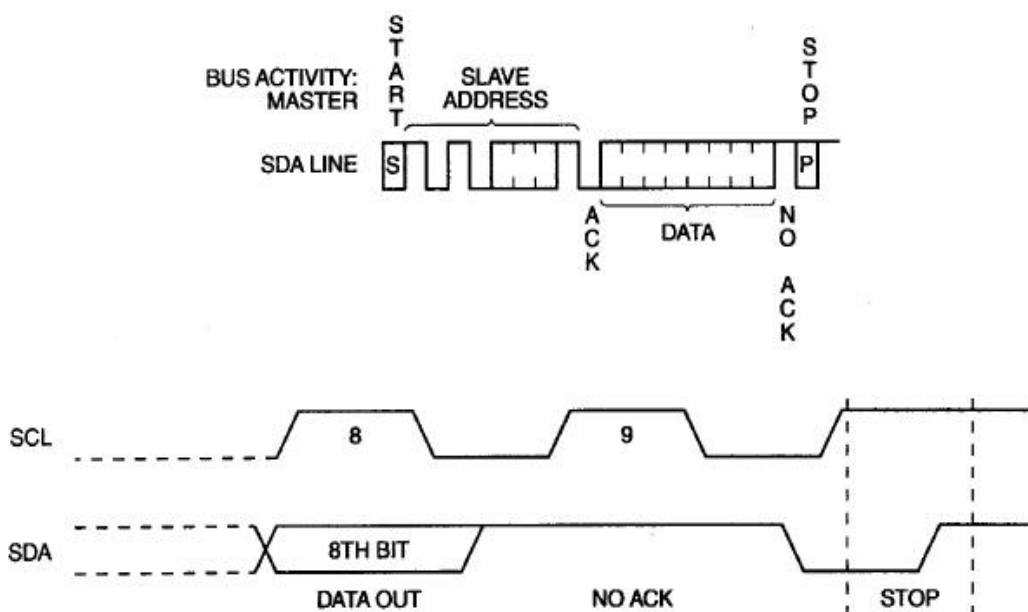


图 7-1-7 AT24C02 的立即地址读模式时序图

在选择读模式下，允许主器件对 AT24C02 的任意字节进行读操作，主器件首先通过发送起始信号、从器件地址和它想读取的字节数据的地址执行一下伪写操作。在 AT24C02 应答之后，主器件重新发送起始信号和从器件地址，此时 R/W 置 1，AT24C02 响应并发送应答信号，然后输出所要求的一个 8 位字节数据，从器件不能发送应答信号但产生一个停止信号。

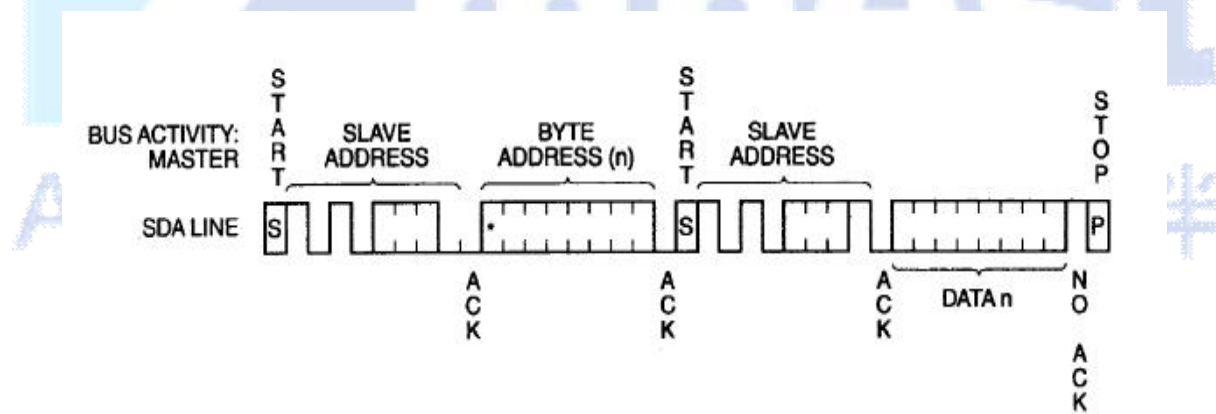


图 7-1-8 AT24C02 的选择读模式时序图

在连续读模式下（连续读操作可通过立即读或选择性读操作启动），AT24C02 发送完一个 8 位字节数据后，主器件产生一个应答信号为响应，告知 AT24C02 主器件要求更多的数据，对应每个主机产生的应答信号 AT24C02 将发送一个 8 位数据字节。当主器件不发送应答信号而发送停止位时结束此操作。从 AT24C02 输出的数据按顺序由 N 到 N+1 输出。读操作时地址计数器在 AT24C02 整个地址内增加，这样整个寄存器区域都可以在一个读操作内全部读出。当读取的字节超过 255，计数器将翻转到零并继续输出数据字节。

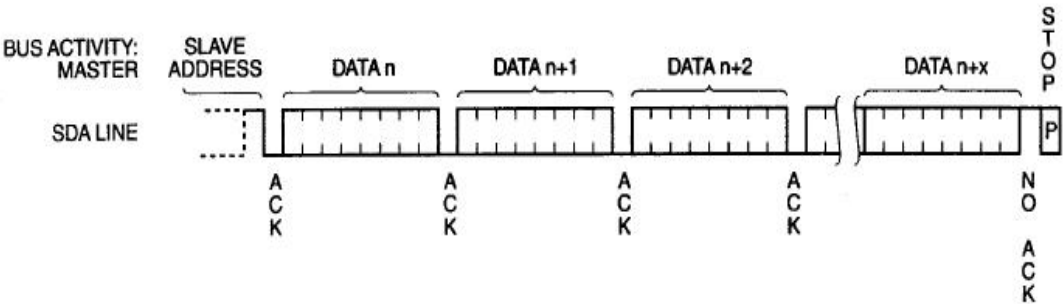


图 7-1-9 AT24C02 的连续读模式时序图

4. S3C2410 处理器 IIC 接口

S3C2410X 处理器为用户进行应用设计提供了支持多主总线的 IIC 接口。处理器提供符合 IIC 协议的设备连接的双向数据线 IICSDA 和 IIC_SCL，在 IIC_SCL 高电平期间，IICSDA 的下降沿启动上升沿停止。S3C2410X 处理器可以支持主发送、主接收、从发送、从接收四种工作模式。在主发送模式下，处理器通过 IIC 接口与外部串行器件进行数据传送，需要使用到如下寄存器：

IIC 总线控制寄存器 IICCON

寄存器	地址	读/写	描述	复位值
IICCON	0x54000000	R/W	IIC 总线控制寄存器	0x0X

IICCON	位	描述	初始值
应答使能 ^[注 1]	[7]	IIC总线应答使能位 0：禁止，1：使能 在输出模式下，IICSDA在ACK时间被释放 在输入模式下，IICSDA在ACK时间被拉低	0
输出时钟源选择	[6]	IIC总线发送时钟预分频选择位 0: IICCLK = fPCLK /16 1: IICCLK = fPCLK /512	0
发送/接收中断使能 ^[注3]	[5]	IIC总线中断使能位 0: 禁止，1：使能	0
中断未决位 ^[注2]	[4]	IIC总线未处理中断标志。不能对这一位写入1，置1是系统自动产生的。当这位被置1，IIC_SCL信号将被拉低，IIC传输也停止了。如果想要恢复操作，将该位清零。 0: 1) 当读出0时，没有发生中断 2) 当写入0时，清除未决条件并恢复中断响应 1: 1) 当读出1时，发生了未决中断 2) 不可以进行写入操作	0
发送时钟值	[3:0]	发送时钟预分频器的值，这四位预分频器的值决定了IIC总线	Undefined

		进行发送的时钟频率，对应关系如下： $Tx\ clock = IICCLK/(IICCON[3:0]+1)$.	
--	--	---	--

注:

1. 在 Rx 模式下访问 EEPROM 时，为了产生停止条件，在读取最后一个字节数据之后不允许产生 ACK 信号。

2. IIC 总线上发生中断的条件： 1) 当一个字节的读写操作完成时； 2) 当一个通常的通话发生或者是从地址匹配上时； 3) 总线仲裁失败时。

3. 如果 IICCON[5]=0, IICCON[4]就不能够正常工作了。因此，建议务必将 IICCON[5]设置为 1，即使你暂时并不用 IIC 中断。

IIC 总线状态寄存器 IICSTAT (地址: 0x54000004)

IICSTAT	位	描述	初始值
模式选择	[7:6]	IIC总线主从，发送/接收模式选择位。 00: 从接收模式； 01: 从发送模式； 10: 主接收模式； 11: 主发送模式	00
忙信号状态/起始/停止条件	[5]	IIC总线忙信号状态位 0: 读为0，表示状态不忙；写入0，产生停止条件 1: 读为1，表示状态忙；写入1，产生起始条件 IICDS中的数据在起始条件之后自动被送出	0
串行数据输出使能	[4]	IIC总线串行数据输出使能/禁止位 0 : 禁止发送/接收； 1: 使能发送接收	0
仲裁状态位	[3]	IIC总线仲裁程序状态标志位 0 : 总线仲裁成功 1: 总线仲裁失败	0
从地址状态标志位	[2]	IIC总线从地址状态标志位 0: 在探测到起始或停止条件时，被清零； 1: 如果接收到的从器件地址与保存在IICADD中的地址相符，则置1	0
0地址状态标志位	[1]	IIC总线0地址状态标志位 0: 在探测到起始或停止条件时，被清零； 1: 如果接收到的从器件地址为0，则置1	0
应答位状态标志	[0]	应答位（最后接收到的位）状态标志 0: 最后接收到的位为0 (ACK接收到了) 1: 最后接收到的位为1 (ACK没有接收到)	0

IIC 总线地址寄存器 IICADD (地址: 0x54000008)

IICADD	位	描述	初始值
从器件地址	[7:0]	7位从器件地址：如果IICSTAT中的串行数据输出使能位为0，IICADD就变为写使能。IICADD总为可读	XXXXXXXX

IIC 总线发送接收移位寄存器 IICDS（地址：0x5400000C）

IICDS	位	描述	初始值
数据移位寄存器	[7:0]	IIC接口发送/接收数据所使用的8位数据移位寄存器：当IICSTAT中的串行数据输出使能位为1，则IICDS写使能。IICDS总为可读	XXXXXXXX

5 软件设计

本实验的内容就是将 0~F 这 16 个数按顺序写入到 EEPROM（AT24C02）的内部存储单元中，然后在依次将他们读出，并通过实验板的串口 UART0 输出到在 PC 机上运行的 windows 自带超级终端上。在本实验中，EEPROM 是被作为 IIC 总线上的从设备来进行处理的，其工作过程涉及到 IIC 总线的主发送和主接收两种工作模式。下图 7-1-10 和图 7-1-11 所示详细说明了这两种工作模式下的程序流程。关于它们的具体实现可以参考实验参考程序中的 write_24c040()和 read_24c040()这两个函数。

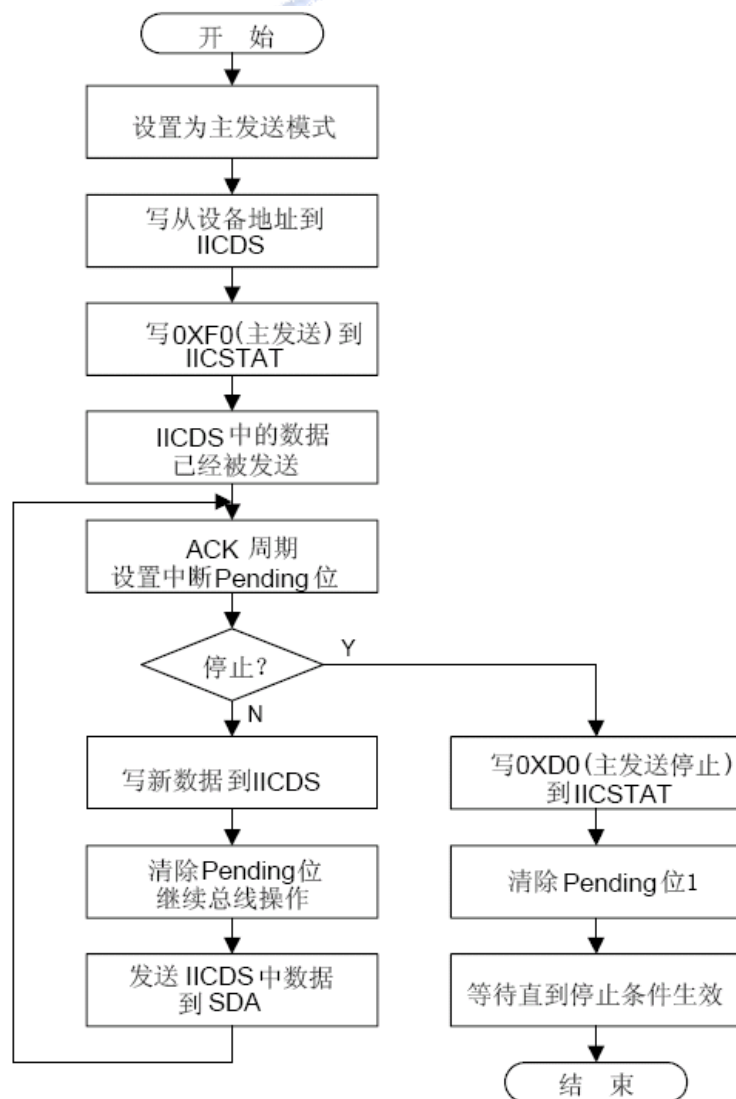


图 7-1-10 IIC 主发送程序设计流程图（S3C2410X）

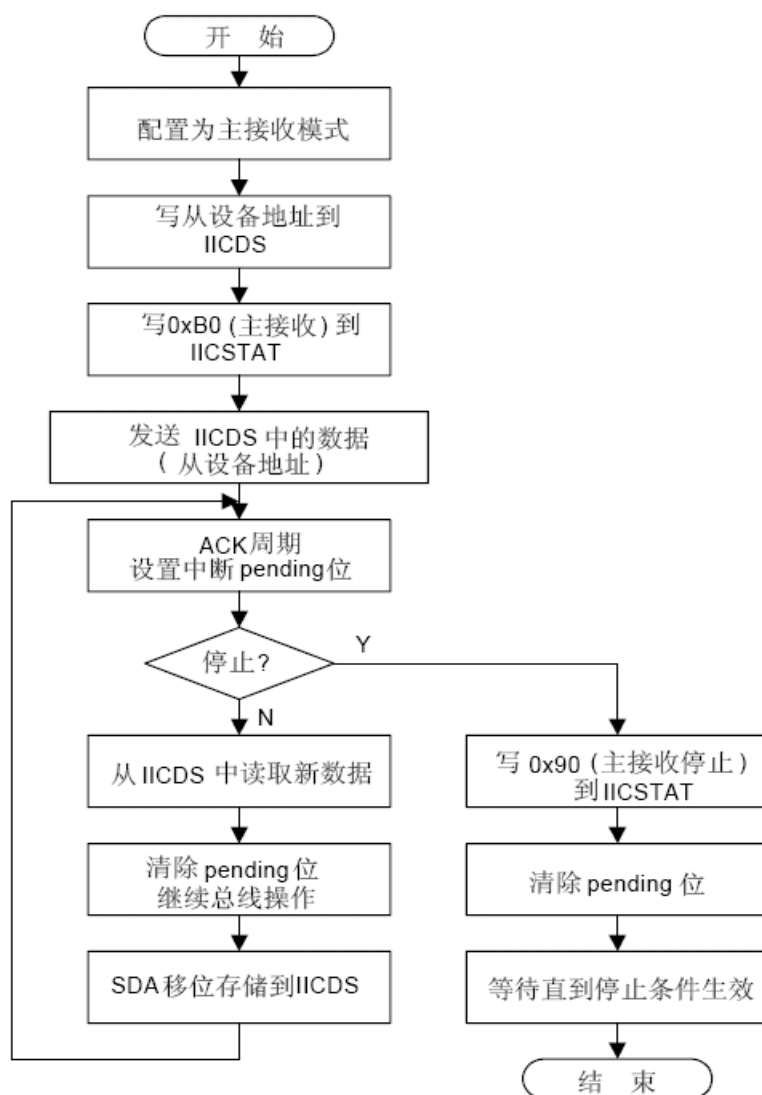


图 7-1-11 IIC 主接收程序设计流程图 (S3C2410X)

7. 1. 5 实验步骤

11. 准备实验环境

使用 ULINK2 仿真器连接 Embest EduKit-IV 实验平台的主板 JTAG 接口; 使用 Embest EduKit-IV 实验平台附带的交叉串口线, 连接实验平台主板上的 COM2 和 PC 机的串口 (一般 PC 只有一个串口, 如果有多个请自行选择, 笔记本没有串口设备的可购买 USB 转串口适配器扩充); 使用 Embest EduKit-IV 实验平台附带的电源适配器, 连接实验平台主板上的电源接口。

12. 串口接收设置

在 PC 机上运行 windows 自带的超级终端串口通信程序, 或者使用实验平台附带光盘内设置好

了的超级终端实验光盘路径，设置超级终端：波特率 115200、1 位停止位、无校验位、无硬件流控制，或者使用其它串口通信程序。（注：超级终端串口的选择根据用户的 PC 串口硬件不同，请自行选择，如果 PC 机只有一个串口，一般是 COM1）

13. 打开实验例程

1) 拷贝实验平台附带光盘实验光盘路径文件夹到 MDK 的安装路径：Keil\ARM\Boards\Embest\
（如果本实验之前已经拷贝，可以跳过这一步）。（注：用户也可拷贝工程到任意目录，本实验为了便于教学，故统一实验路径）；

2) 运行 μ Vision IDE for ARM 软件，点击菜单栏“Project”，选择“Open Project...”，在弹出的对话框选择实验例程目录 7.1_EEPROM_Test 子目录下的 EEPROM_Test.Uv2 工程。

3) 默认打开的工程在源码编辑窗口会显示实验例程的说明文件 readme.txt，详细阅读并理解实验内容。

4) 工程提供了两种运行方式：一是下载到 SDRAM 中调试运行，二是固化到 Nor Flash 中运行。用户可以在工具栏 Select Target 下拉框中选择在 RAM 中调试运行还是固化 Flash 中运行。如下图所示：

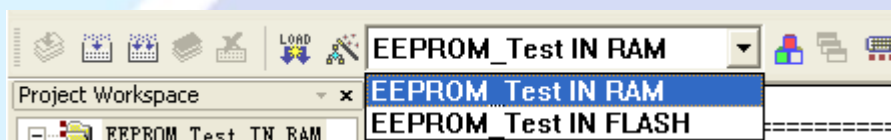



图 7-1-12 选择运行方式

下面实验将介绍下载到 SDRAM 中调试运行，所以我们在 Select Target 下拉框中选择 EEPROM_Test IN RAM。

5) 接下来开始编译链接工程，在菜单栏“Project”选择“Build target”或者“Rebuild all target files”编译整个工程，用户也可以在工具栏单击“”或者“”进行编译。

6) 编译完成后，在输出窗口可以看到编译提示信息，比如““.\\SDRAM\\EEPROM_Test.axf" - 0 Error(s), 1 Warning(s).”，如果显示“0 Error(s)”即表示编译成功。


7) 拨动实验平台电源开关，给实验平台上电，单击菜单栏 Debug->Start/Stop Debug Session 项将编译出来的映像文件下载到 SDRAM 中，或者单击工具栏“”按钮来下载。

8) 下载完成后，单击菜单栏 Debug->Run 项运行程序，或者单击工具栏“”按钮来全速运行程序。用户也可以使用进行单步调试程序。

9) 全速运行后，用户可以在超级终端看到程序运行的信息，出现“Please input words, then press Enter”提示后输入想要发送的数据，并已回车作为发送字符串的结尾标志。

10) 用户可以 Stop 程序运行，使用 μ Vision IDE for ARM 的一些调试窗口跟踪查看程序运行的信息。

注：如果在第 4) 步用户选择在 Flash 中运行，则编译链接成功后，单击菜单栏 Flash->Download

项将程序固化到 NorFlash 中，或者单击工具栏按钮“”固化程序，从实验平台的主板拔出 JTAG 线，给实验平台重新上电，程序将自动运行。

14. 观察实验结果

在 PC 机上观察超级终端程序主窗口，可以看到如下界面：

```

*****
**                               **
**      英蓓特 EduKit 系列嵌入式教学系统平台      **
**                               **
**      Embest EduKit Series Embedded Teaching Platform      **
**                               **
*****

IIC operate Test Example

IIC Test using AT24C02

Write char 0-f into AT24C02

Read 16 bytes from AT24C02

0 1 2 3 4 5 6 7 8 9 a b c d e f
    
```

15. 完成实验练习题

理解和掌握实验后，完成实验练习题。

7. 1. 6 实验参考程序

1. 初始化及测试主程序

```

/*-----*/
/*****
* name:      eeprom_test
* func:      test iic
* para:      none
* ret: none
* modify:
* comment:
*****/

Void eeprom_test(void)
{
    UINT8T      szData[256];
    //UINT8T      szBuf[256];
    
```

```

    unsigned int    i;
    uart_printf("\n IIC operate Test Example");
    uart_printf("\n IIC Test using AT24C02 ");
    uart_printf("\n Write char 0-f into AT24C02\n");
    f_nGetACK = 0;
    // Enable interrupt
    rINTMOD  = 0x0;
    rINTMSK &= ~BIT_IIC;
    pISR_IIC = (unsigned)iic_int_24c04;
    // Initialize iic
    rIICADD = 0x10;                // S3C2410X slave address
    rIICCON = 0xaf;                // Enable ACK, interrupt, SET IICCLK=MCLK/16
    rIICSTAT = 0x10;                // Enable TX/RX
    // Write 0 - 16 to 24C04
    for(i=0; i<16; i++)
        iic_write_24c040(0xa0, i, i);
    // Clear array
    for(i=0; i<16; i++)
        szData[i]=0;
    // Read 16 byte from 24C04
    for(i=0; i<16; i++)
        iic_read_24c040(0xa0, i, &(szData[i]));
    // Printf read data
    uart_printf(" Read 16 bytes from AT24C02\n");
    for(i=0; i<16; i++)
        uart_printf(" %2x ", szData[i]);
    rINTMSK |= BIT_IIC;
    uart_printf("\n end.\n");
}

```

2. 中断服务程序

```

/*****
* name:      iic_int_24c04()
* func:      IIC interrupt handler
* para:      none
* ret: none
* modify:

```



```

* comment:
*****/

void iic_int_24c04(void)
{
    ClearPending(BIT_IIC);
    f_nGetACK = 1;
}

```

3. IIC 写 AT24C02 程序

```

/*****
* name:      iic_write_24c040
* func:      write data to 24C040
* para:      unSlaveAddr  --- input, chip slave address
*            unAddr       --- input, data address
*            ucData        --- input, data value
*****/

void iic_write_24c040(UINT32T unSlaveAddr,UINT32T unAddr,UINT8T ucData)
{
    f_nGetACK = 0;
    // Send control byte
    rIICDS = unSlaveAddr;                // 0xa0
    rIICSTAT = 0xf0;                     // Master Tx,Start
    while(f_nGetACK == 0);               // Wait ACK
    f_nGetACK = 0;
    //Send address
    rIICDS = unAddr;
    rIICCON = 0xaf;                      // Resumes IIC operation.
    while(f_nGetACK == 0);               // Wait ACK
    f_nGetACK = 0;
    // Send data
    rIICDS = ucData;
    rIICCON = 0xaf;                      // Resumes IIC operation.
    while(f_nGetACK == 0);               // Wait ACK
    f_nGetACK = 0;
    // End send
    rIICSTAT = 0xd0;                     // Stop Master Tx condition
    rIICCON = 0xaf;                      // Resumes IIC operation.
}

```

```

        delay(10);                                // Wait until stop condition is in effect.
    }

```

4. IIC 读 AT24C02 程序

```

/*****
* name:      read_24c040
* func:      read data from 24C040
* para:      unSlaveAddr --- input, chip slave address
*            unAddr      --- input, data address
*            pData       --- output, data pointer
* ret:      none
*****/

void read_24c040(UINT32T unSlaveAddr,UINT32T unAddr,UINT8T *pData)
{
    char cRecvByte;
    f_nGetACK = 0;
    // Send control byte
    rIICDS = unSlaveAddr;                                // 0xa0
    rIICSTAT = 0xf0;                                     // Master Tx,Start
    while(f_nGetACK == 0);                               // Wait ACK
    f_nGetACK = 0;
    // Send address
    rIICDS = unAddr;
    rIICCON = 0xaf;                                       // Resumes IIC operation.
    while(f_nGetACK == 0);                               // Wait ACK
    f_nGetACK = 0;
    // Send control byte
    rIICDS = unSlaveAddr;                                // 0xa0
    rIICSTAT = 0xb0;                                     // Master Rx,Start
    rIICCON = 0xaf;                                       // Resumes IIC operation.
    while(f_nGetACK == 0);                               // Wait ACK
    f_nGetACK = 0;
    rIICCON = 0x2f;
    delay(1);
    // Get data
    cRecvByte = rIICDS;
    // End receive
}

```

```
        rIICSTAT = 0x90;                                // Stop Master Rx
        condition
        rIICCON = 0xaf;                                // Resumes IIC operation.
        delay(10);                                     // Wait until stop condition is in effect.
        *pData = cRecvByte;
    }
```

7. 1. 7 练习题

编写程序往 AT24C02 芯片上存储某天日期字符串,再读出,并通过串口或液晶屏输出。

7. 2 以太网通信试验

7. 2. 1 实验目的

- Ø 通过实验了解以太网通讯原理和驱动程序开发方法;
- Ø 通过实验了解 IP 网络协议和网络应用程序开发方法。

7. 2. 2 实验设备

- Ø 硬件: Embest EduKit-IV 平台, ULINK2 仿真器套件, PC 机;
- Ø 软件: µVision IDE for ARM 集成开发环境, Windows 98/2000/NT/XP。

7. 2. 3 实验内容

熟悉以太网控制器 DM9000,在内部以太局域网上基于 TFTP/IP 协议,下载文本文件到开发板中。

7. 2. 4 实验原理

1. 以太网通讯原理

以太网是由 Xeros 公司开发的一种基带局域网碰撞检测 (CSMA/CD) 机制,使用同轴电缆作为

传输介质，数据传输速率达到 10M；使用双绞线作为传输介质，数据传输速率达到 100M/1000M。现在普遍遵从 IEEE802.3 规范。

结构

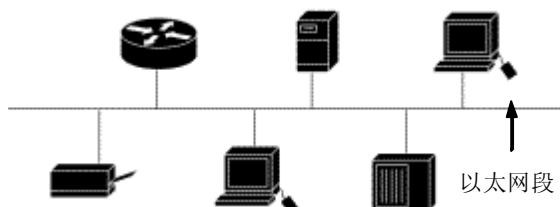


图 7-2-1 以太网结构示意图

类型

以太网/IEEE802.3——采用同轴电缆作为网络媒体，传输速率达到 10Mbps；

100Mbps 以太网——又称为快速以太网，采用双绞线作为网络媒体，传输速率达到 100Mbps；

1000Mbps 以太网——又称为千兆以太网，采用光缆或双绞线作为网络媒体。

工作原理

以太网的传输方法，也就是以太网的介质访问控制（MAC）技术称为载波监听多路存取和冲突检测（CSMA / CD），下面我们分步来说明其原理：

载波监听：当你所在的网站（计算机）要向另一个网站发送信息时，先监听网络信道上有无信息正在传输，信道是否空闲。

信道忙碌：如果发现网络信道正忙，则等待，直到发现网络信道空闲为止。

信道空闲：如果发现网络信道空闲，则向网上发送信息。由于整个网络信道为共享总线结构，网上所有网站都能够收到你所发出的信息，所以网站向网络信道发送信息也称为“广播”。但只有你想要发送数据的网站识别和接收这些信息。

冲突检测：网站发送信息的同时，还要监听网络信道，检测是否有另一台网站同时在发送信息。如果有，两个网站发送的信息会产生碰撞，即产生冲突，从而使数据信息包被破坏。

遇忙停发：如果发送信息的网站检测到冲突，则立即停止发送，并向网上发送一个“冲突”信号，让其它网站也发现该冲突，从而摒弃可能一直在接收受损的信息包。

多路存取：如果发送信息的网站因“碰撞冲突”而停止发送，就需等待一段时间，再回到第一步，重新开始载波监听和发送，直到数据成功发送为止。

所有共享型以太网上的网站，都是经过上述六步步骤，进行数据传输的。由于 CSMA / CD 介质访问控制法规定在同一时间里，只能有一个网站发送信息，其它网站只能收听和等待，否则就会产生“碰撞”。所以当共享型网络用户增加时，每个网站在发送信息时产生“碰撞”的概率增大，当网络用户增加到一定数目后，网站发送信息产生的“碰撞”会越来越多，想发送信息的网站不断地进行：监听—Λ 发送—Λ 碰撞—Λ 停止发送—Λ 等待—Λ 再监听—Λ 再发送……

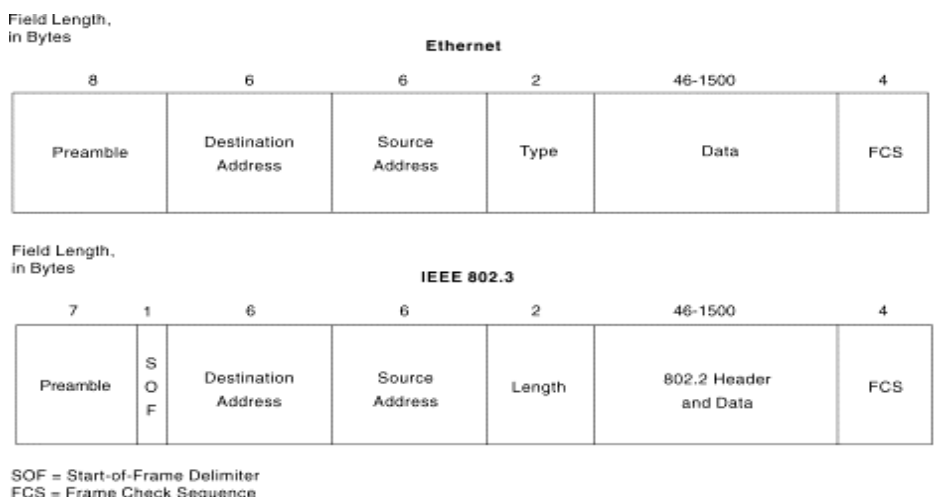


图 7-2-2 以太网/IEEE 802.3 帧的基本组成

以太网/IEEE802.3 帧的结构

如上图所示，以太网和 IEEE802.3 帧的基本结构如下：

前导码：由 0、1 间隔代码组成，可以通知目标站作好接收准备。IEEE 802.3 帧的前导码占用 7 个字节，紧随其后的是长度为 1 个字节的帧首定界符（SOF）。以太网帧把 SOF 包含在了前导码当中，因此，前导码的长度扩大为 8 个字节。

帧首定界符（SOF）：IEEE 802.3 帧中的定界字节，以两个连续的代码 1 结尾，表示一帧的实际开始。

目标和源地址：表示发送和接收帧的工作站的地址，各占据 6 个字节。其中，目标地址可以是单址，也可以是多点传送或广播地址。

类型（以太网）：占用 2 个字节，指定接收数据的高层协议。

长度（IEEE 802.3）：表示紧随其后的以字节为单位的数据段的长度。

数据（以太网）：在经过物理层和逻辑链路层的处理之后，包含在帧中的数据将被传递给在类型段中指定的高层协议。虽然以太网版本 2 中并没有明确作出补齐规定，但是以太网帧中数据段的长度最小应当不低于 46 个字节。

数据（IEEE 802.3）：IEEE 802.3 帧在数据段中对接收数据的上层协议进行规定。如果数据段长度过小，使帧的总长度无法达到 64 个字节的最小值，那么相应软件将会自动填充数据段，以确保整个帧的长度不低于 64 个字节。

帧校验序列（FSC）：该序列包含长度为 4 个字节的循环冗余校验值（CRC），由发送设备计算产生，在接收方被重新计算以确定帧在传送过程中是否被损坏。

2. IP 网络协议原理

TCP/IP 协议是一组包括 TCP（Transmission Control Protocol）协议和 IP（Internet Protocol）协议，UDP（User Datagram Protocol）协议、ICMP（Internet Control Message Protocol）协议和

其他一些协议的协议组。

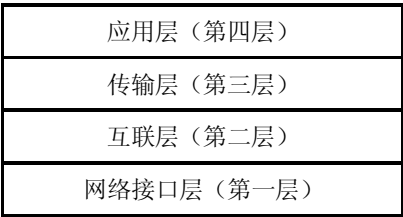


图 7-2-3 TCP/IP 协议层次

TCP/IP 协议采用分层结构，共分为四层，每一层独立完成指定功能，如上图所示：

网络接口层：负责接收和发送物理帧，它定义了将数据组成正确帧的规程和在网络中传输帧的规程，帧是指一串数据，它是数据在网络中传输的单位。网络接口层将帧放在网上，或从网上把帧取下来。

互联层：负责相邻结点之间的通信，本层定义了互联网中传输的“信息包”格式，以及从一个节点通过一个或多个路由器运行必要的路由算法到最终目标的“信息包”转发机制。主要协议有 IP、ARP、ICMP、IGMP 等。

传输层：负责起点到终点的通信，为两个用户进程之间建立、管理和拆除有效的端到端连接。主要协议有 TCP、UDP 等。

应用层：它定义了应用程序使用互联网的规程。应用程序通过这一层访问网络，主要遵从 BSD 网络应用接口规范。主要协议有 SMTP、FTP、TELNET、HTTP 等。

IP

网际协议 IP 是 TCP/IP 的心脏，也是网络层中最重要的协议。

IP 层接收由更低层（网络接口层例如以太网设备驱动程序）发来的数据包，并把该数据包发送到更高层——TCP 或 UDP 层；相反，IP 层也把从 TCP 或 UDP 层接收来的数据包传送到更低层。IP 数据包是不可靠的，因为 IP 并没有做任何事情来确认数据包是按顺序发送的或者没有被破坏。

当前 IP 协议有 IPv4 和 IPv6 两个版本，IPv4 正被广泛使用，IPv6 是下一代高速互联网的基础协议。下面这个表是 IPv4 的数据包格式：

0	4	8	16	32
版本	首部长度	服务类型	数据包总长	
	标识	DF MF	碎片偏移	
	生存时间	协议	首部校验和	
	源 IP 地址			
	目的 IP 地址			
	选项			

数据

IP 协议头的结构定义如下:

```

struct      ip_header
{
    UIntip_v:4;                //协议版本
    UInt   ip_hl:4;            //协议头长度
    UInt8   ip_tos;            //服务类型
    UInt16  ip_len;            //数据包长度
    UInt16  ip_id;             //协议标识
    UInt16  ip_off;            //分段偏移域
    UInt8   ip_ttl;            //生存时间
    UInt8   ip_p;              //IP 数据包的高层协议
    UInt16  ip_sum;            //校验和
    Struct in_addr ip_src, ip_dst; //源和目的 IP 地址
};

```

ip_v: IP 协议的版本号, IPv4 为 4, IPv6 为 6。

ip_hl: IP 包首部长度,这个值以 4 字节为单位.IP 协议首部的固定长度为 20 个字节,如果 IP 包没有选项,那么这个值为 5。

ip_tos: 服务类型,说明提供的优先权。

ip_len: 说明 IP 数据的长度,以字节为单位。

ip_id: 标识这个 IP 数据包。

ip_off: 碎片偏移，这和上面 ID 一起用来重组碎片的。

ip_ttl: 生存时间, 每经过一个路由时减一, 直到为 0 时被抛弃。

ip_p: 协议,表示创建这个 IP 数据包的高层协议.如 TCP,UDP 协议。

ip_sum: 首部校验和，提供对首部数据的校验。

ip_src,ip_dst: 发送者和接收者的 IP 地址。

IP 地址实际上是采用 IP 网间网层通过上层软件完成“统一”网络物理地址的方法,这种方法使用统一的地址格式,在统一管理下分配给主机。Internet 网上不同的主机有不同的 IP 地址,在 IPv4 协议中,每个主机的 IP 地址都是由 32 比特,即 4 个字节组成的。为了便于用户阅读和理解,通常采用“点分十进制表示方法”表示,每个字节为一部分,中间用点号分隔开来。如 211.154.134.93 就是嵌入开发网 WEB 服务器的 IP 地址。每个 IP 地址又可分为两部分。网络号表示网络规模的大小,主机号表示网络中主机的地址编号。按照网络规模的大小,IP 地址可以分为 A、B、C、D、E 五类,其中 A、B、C 类是三种主要的类型地址,D 类专供多目传送用的多目地址,E 类用于扩展备用地址。

TCP

如果 IP 数据包中有已经封好的 TCP 数据包, 那么 IP 将把它们向“上”传送到 TCP 层。TCP 将包排序并进行错误检查, 同时实现虚电路间的连接。TCP 数据包中包括序号和确认, 所以未按照顺序收到的包可以被排序, 而损坏的包可以被重传。

TCP 将它的信息送到更高层的应用程序, 例如 Telnet 的服务程序和客户程序。应用程序轮流将信息送回 TCP 层, TCP 层便将它们向下传送到 IP 层, 设备驱动程序和物理介质, 最后到接收方。下面是 TCP 协议的数据包头格式:

0	4	8	10	16	24	32
源端口				目的端口		
序列号						
确认号						
首部长度		保留		窗口		
		U A P S F				
		R C S Y I				
		G K H N N				
校验和				紧急指针		
选项				填充字节		

关于 TCP 协议的详细情况, 请查看 RFC793 文档。

TCP 对话通过三次握手来初始化。三次握手的目的是使数据段的发送和接收同步; 告诉其它主机其一次可接收的数据量, 并建立虚连接。

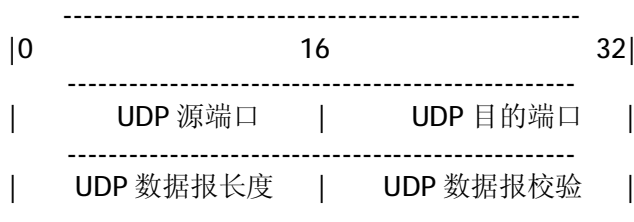
我们来看看这三次握手的简单过程:

- (1) 初始化主机通过一个同步标志置位的数据段发出会话请求。
- (2) 接收主机通过发回具有以下项目的数据段表示回复: 同步标志置位、即将发送的数据段的起始字节的序号、应答并带有将收到的下一个数据段的字节序号。
- (3) 请求主机再回送一个数据段, 并带有确认序号和确认号。

UDP

UDP 与 TCP 位于同一层, 但对于数据包的顺序错误或重发不处理。因此, UDP 不被应用于那些使用虚电路的面向连接的服务, UDP 主要用于那些面向查询---应答的服务, 例如 NFS。相对于 FTP 或 Telnet, 这些服务需要交换的信息量较小。使用 UDP 的服务包括 NTP (网落时间协议) 和 DNS (DNS 也使用 TCP)。

UDP 协议的数据包头格式为:



UDP 协议适用于无须应答并且通常一次只传送少量数据的应用软件。

ICMP

ICMP 与 IP 位于同一层，它被用来传送 IP 的控制信息。它主要是用来提供有关通向目的地址的路径信息。ICMP 的“Redirect”信息通知主机通向其他系统的更准确的路径，而“Unreachable”信息则指出路径有问题。另外，如果路径不可用了，ICMP 可以使 TCP 连接“体面地”终止。PING 是最常用的基于 ICMP 的服务。

ARP

要在网络上通信，主机就必须知道对方主机的硬件地址。地址解析就是将主机 IP 地址映射为硬件地址的过程。地址解析协议 ARP 用于获得在同一物理网络中的主机的硬件地址。

解释本地网络 IP 地址过程：

- (1) 当一台主机要与别的主机通信时，初始化 ARP 请求。当该 IP 断定 IP 地址是本地时，源主机在 ARP 缓存中查找目标主机的硬件地址。
- (2) 要是找不到映射的话，ARP 建立一个请求，源主机 IP 地址和硬件地址会被包括在请求中，该请求通过广播，使所有本地主机均能接收并处理。
- (3) 本地网上的每个主机都收到广播并寻找相符的 IP 地址。
- (4) 当目标主机断定请求中的 IP 地址与自己的相符时，直接发送一个 ARP 答复，将自己的硬件地址传给源主机。以源主机的 IP 地址和硬件地址更新它的 ARP 缓存。源主机收到回答后便建立起了通信。

TFTP 协议

TFTP 是一个传输文件的简单协议，一种简化的 TCP/IP 文件传输协议，它基于 UDP 协议而实现，支持用户从远程主机接收或向远程主机发送文件。此协议设计的时候是进行小文件传输的。因此它不具备通常的 FTP 的许多功能，它只能从文件服务器上获得或写入文件，不能列出目录，不进行认证，它传输 8 位数据。

因为 TFTP 使用 UDP，而 UDP 使用 IP，IP 还可以使用其它本地通信方法。因此一个 TFTP 包中会有以下几段：本地媒介头，IP 头，数据报头，TFTP 头，剩下的就是 TFTP 数据了。TFTP 在 IP 头中不指定任何数据，但是它使用 UDP 中的源和目标端口以及包长度域。由 TFTP 使用的包标记(TID)在这里被用做端口，因此 TID 必须介于 0 到 65,535 之间。

初始连接时候需要发出 WRQ（请求写入远程系统）或 RRQ（请求读取远程系统），收到一个确定应答，一个确定的可以写出的包或应该读取的第一块数据。通常确认包包括要确认的包的包号，每个数据包都与一个块号相对应，块号从 1 开始而且是连续的。因此对于写入请求的确定是一个比较特殊的情况，因此它的包的包号是 0。如果收到的包是一个错误的包，则这个请求被拒绝。创建连接时，通信双方随机选择一个 TID，因为是随机选择的，因此两次选择同一个 ID 的可能性就很小了。每个包包括两个 TID，发送者 ID 和接收者 ID。在第一次请求的时候它会将请求发到 TID 69，也就是服务器的 69 端口上。应答时，服务器使用一个选择好的 TID 作为源 TID，并用上一个包中的 TID 作为目的 ID 进行发送。这两个被选择的 ID 在随后的通信中会被一直使用。

此时连接建立，第一个数据包以序列号 1 从主机开始发出。以后两台主机要保证以开始时确定的 TID 进行通信。如果源 ID 与原来确定的 ID 不一样，这个包会被认识为发送到了错误的地址而被抛弃。

网络应用程序开发方法

进行网络应用程序开发有两种方法：一是采用 BSD Socket 标准接口，程序移植能力强；二是采用专用接口直接调用对应的传输层接口，效率较高。

BSD Socket 接口编程方法

Socket（套接字）是通过标准的文件描述符和其它程序通讯的一个方法。每一个套接字都用一个半相关描述：{协议，本地地址、本地端口}来表示；一个完整的套接字则用一个相关描述：{协议，本地地址、本地端口、远程地址、远程端口}，每一个套接字都有一个本地的由操作系统分配的唯一套接字号。

传输层专用接口编程方法

网络协议都可以直接提供专用函数接口给上层或者跨层调用，用户可以调用每个协议代码中特有的接口实现快速数据传递。

实验板的网络协议包提供的就是 TFTP 协议的专用接口，应用程序可以通过它接收用户从主机上使用 TFTP 传递过来的数据。主要接口函数有：

int DM9000DBG_GetFrame(UINT8T *pbData, unsigned int *pwLength)——接收用户数据，网络协议包自动完成连接过程，并从网络获取用户传递过来的数据包，每次实际接受到的包传递给参数 len。

dm9000_send (UINT8T *pbData,unsigned int length)——传递给用户的数据。pbData 只想所要传输数据的首地址，length 存放所要传输数据的字节数。

波特率由一个专用的 UART 波特率分频寄存器（UBRDIVn）控制，计算公式如下：

$$\text{UBRDIVn} = (\text{int})(\text{ULK}/(\text{bps} \times 16)) - 1$$

$$\text{或者 } \text{UBRDIVn} = (\text{int})(\text{PLK}/(\text{bps} \times 16)) - 1$$

其中:时钟选用 ULK 还是 PLK 由 UART 控制寄存器 UCONn[10]的状态决定。如果 UCONn[10]=0, 用 PLK 作为波特率发生, 否则选用 ULK 做波特率发生。UBRDIVn 的值必须在 1 到 (216-1) 之间。

例如: ULK 或者 PLK 等于 40MHz, 当波特率为 115200 时,

$$UBRDIVn = (int)(40000000/(115200 \times 16)) - 1 = (int)(21.7) - 1 = 21 - 1 = 20$$

3. 硬件设计

DM9000 介绍

DM9000 (A) 是一个全集成、功能强大、性价比高的快速以太网 MAC 控制器, 它带有一个通用处理器接口、EEPROM 接口、10/100 PHY 和 16KB 的 SRAM (13KB 作为接收 FIFO, 3KB 作为发送 FIFO)。它采用单电源供电, 可兼容 3.3V、5V 的 IO 接口电平。

DM9000(A)同样支持 MII(Media Independent Interface 介质无关)接口, 连接到 HPNA(Home Phone-line Networking Alliance 家用电话网络联盟)设备上或其它支持 MII 的设备。其 16 位模式的封装图如图 7-2-4。

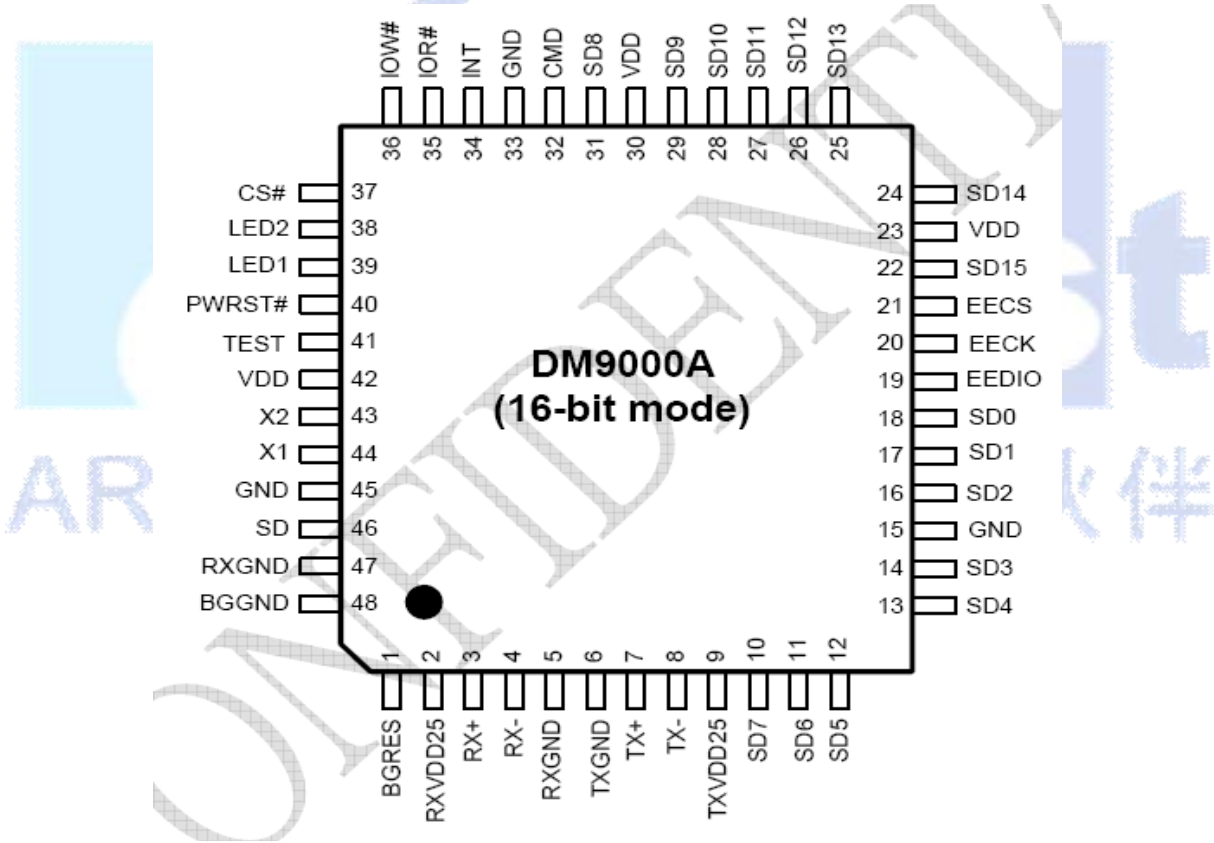


图 7-2-4 DM9000 封装图

各引脚功能以及和处理器的连接状况如表 7-2-1 所示。

表 7-2-1 引脚功能表

处理器总线	DM9000	引脚号	数据传	功能描述
-------	--------	-----	-----	------

信号	信号		输方向	
nRD	IOR#	35	输入	处理器读命令。此引脚默认下是低电平，可以通过 EEPROM 设定来来修改它的极性
nWR	IOW#	36	输入	处理器写命令。此引脚默认下是低电平，可以通过 EEPROM 设定来来修改它的极性
nCS/nAEN	CS#	36	输入	片选信号。通过此引脚上一个低电平信号来选通 DM9000。可以通过 EEPROM 设定来来修改它的极性
SD0~7	SD0~7	18,17,16,14,13,12,11,10	输入 / 输出	数据数据总线 0~7
SD8~15	SD8~15	31,29,28,27,26,25,24,22	输入 / 输出	书记总线 8~16
CMD	CMD	32	输入	命令类型：在此命令周期内。当其为低电平时，为访问 INDEX 端口。当其为高电平时，为访问数据端口。
INT	INT	34	输出	中断请求。此引脚默认下是高电平。可以修改 EEPROM 设定来改变它的极性和输出类型。

硬件连接



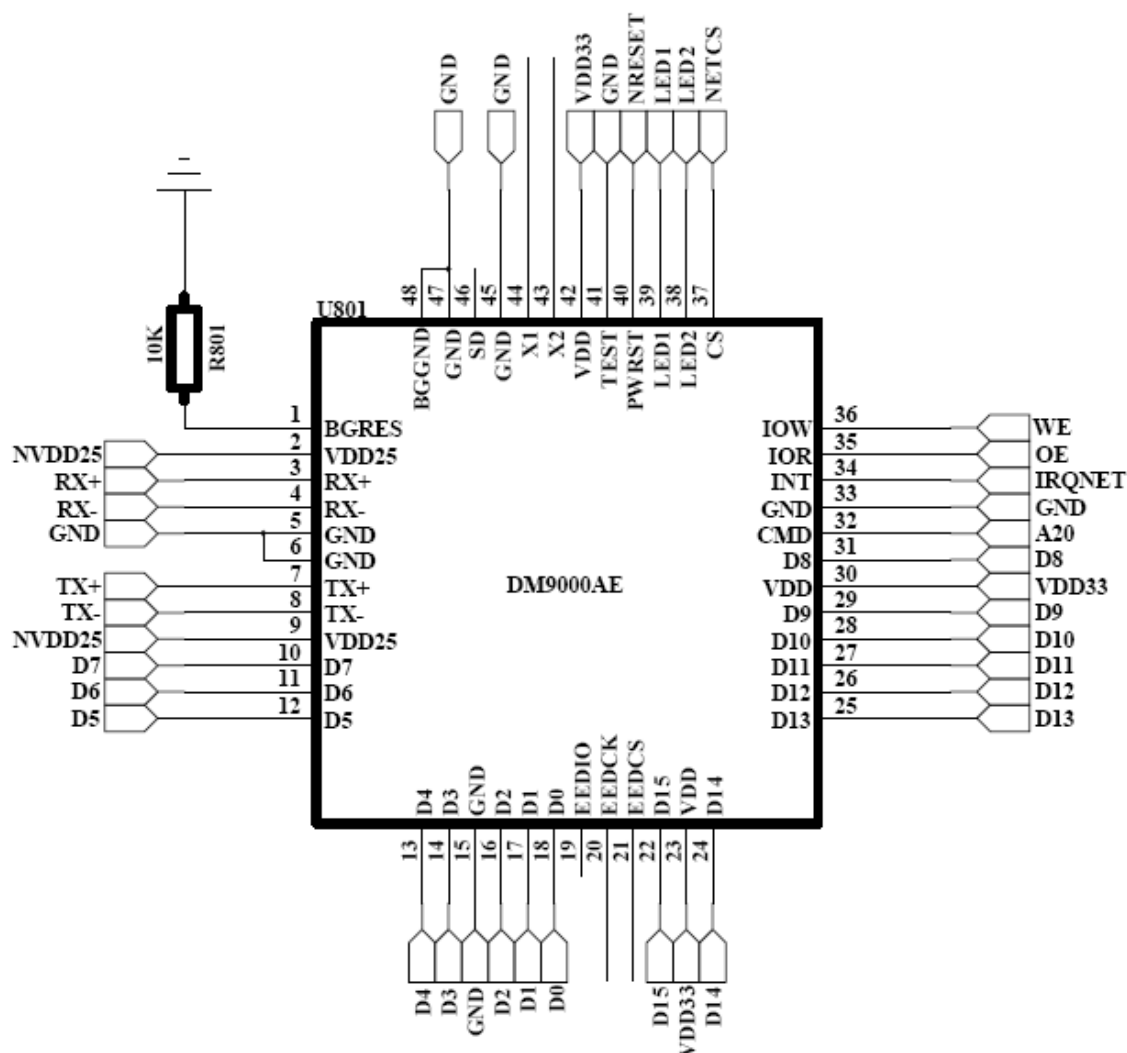


图 7-2-5 DM9000 封装图

其中外部连接的 D0~D15 为 CPU 的数据总线。RX+、RX-、TX+ 和 TX- 为网口相连用于传输数据的 4 根线。WE 和 OE 为 CPU 的写信号和读信号。NETCS 为通过 CPLD 和 CPU 相连，CMD 为 CPU 的 A20 地址信号。通过 CPLD 编码后，DM9000 的 INDEX 端口的地址为 0x20000000，其数据 DATA 端口地址为 0x20100000。IRQNET 为连接到 IO3 的中断，也是通过 CPLD 来控制它。此程序不是通过中断方式，而是通过查询方式来判断是否收到数据。

4. DM9000 软件设计

如何访问芯片

在命令周期内，可以通过操作 CS 引脚，再结合 IOR 或 IOW 引脚来访问 DM9000。这些引脚在默认下都是低电平，也可以通过 EEPROM 中的设定来修改它们的电平极性以满足在不同处理器上的应用。

通过激活 CS 角、IOW/IOR 角来写/读 INDEX 或 DATA 端口。此程序就是通过访问处理器的 0x20000000 地址来访问 INDEX 端口、0x20100000 地址来访问 DATA 端口。

如何初始化 DM9000

1) 设定 INDEX 和 DATA 端口地址

```
dwEthernetIOBase   = 0x20000000;

dwEthernetDataPort = 0x20100000;
```

2) 看是否探测到网卡，即能否读出网卡信息

```
r = Probe(); /*Detect DM9000 */
```

3) 内部 PHY 加电

默认情况下，PHY 处于掉电模式。可以写 0 到 PHYPD 来对内部 PHY 加电

```
WRITE_REG1(0x1f, 0); /* GPR (reg_1Fh) bit GPIO0=0 pre-activate PHY */
```

4) 通过软件重启 DM9000

```
/* do a software reset */
WRITE_REG1(0x0, 3); /* NCR (reg_00h) bit[0] RST=1 & Loopback=1, reset on */
DM9000_Delay(1000);
WRITE_REG1(0x0, 3); /* NCR (reg_00h) bit[0] RST=1 & Loopback=1, reset on */
DM9000_Delay(1000);
```

5) 设定 IMR 寄存器的 Bit[7]来使能对收发缓冲区的读写时，队列对首指针的自动修改功能

```
WRITE_REG1(0xff, 0x80); /* Enable SRAM automatically return */
```

6) 操作某些控制寄存器，来清除传输和中断状态位

```
WRITE_REG1(0x01, 0x2c); /* clear TX status */

WRITE_REG1(0xfe, 0x0f); /* Clear interrupt status */
```

7) 写网卡的物理地址到相关寄存器中

```
/* Set Node address */
WRITE_REG1(0x10, (UINT8T)(mac[0] & 0xFF));
WRITE_REG1(0x11, (UINT8T)(mac[0] >> 8));
WRITE_REG1(0x12, (UINT8T)(mac[1] & 0xFF));
WRITE_REG1(0x13, (UINT8T)(mac[1] >> 8));
WRITE_REG1(0x14, (UINT8T)(mac[2] & 0xFF));

WRITE_REG1(0x15, (UINT8T)(mac[2] >> 8));
```

8) 若是通过中断方式判断是否收到或发出数据，则使能并初始化网卡中断

9) 通过编写 RCR 寄存器来使能收数据

```
WRITE_REG1(0x05, 0x30 | 1); /* Discard long packet and CRC error packets*/ /* RX enable */
```

10) 等待 DM9000 连接好

```
while(1)
{
    temp=READ_REG1(0x01)&0x40;
    if(temp) {break; }
}
```

如何传输数据包

在传输一个数据包之前，此数据包必须已经存放在处于内部 SRAM 中 0~0XBFF 的输出缓冲队列中。即首先写 0XF8 到 INDEX 端口，再将数据包长度的高字节和低字节分别写入 TXPLH 和 TXPLL 寄存器。再向数据端口写要发送的数据，数据也就保存到输出队列了。最后再设置 TXRQ 寄存器的 Bit[0]来请求传输此数据包。

当数据传输完成时，会通过以下方式指示数据包的传输完成。若设定了 IMR 寄存器的 Bit[1]来使能了传输完成中断，则会产生中断并在 ISR 寄存器中的 Bit[1]置 1。而且 NSR 寄存器中的 TX1END Bit[2]或 TX2END Bit[3]将会置 1（若为 1 号包则置 TX1END 位，若为 2 号包则置 TX2END 位）。

传输数据包的具体步骤为：

- 1) 核查内存数据宽度为 8 位还是 16 位

```
/* I/O mode */
```

```
DM9000_iomode = READ_REG1(0xfe) >> 6; /* ISR bit7:6 keeps I/O mode */
```

- 2) 将包的数据写入到发送缓冲队列中

```
IOWRITE(dwEthernetIOBase, 0xf8); /* data copy ready set */
/*pbData[]:要传输的数据。Length:pbData[]的长度*/
/* copy data to FIFO */
If(DM9000_iomode==DM9000_BYTE_MODE)
{
    for (i = 0; i < length; i++)
        IOWRITE(dwEthernetDataPort, ((UINT8T *)pbData)[i]);
}
Else If(DM9000_iomode== DM9000_WORD_MODE)
{
    tmplen = (length+1)/2;
    for (i = 0; i < tmplen; i++)
        IOWRITE16(dwEthernetDataPort, ((UINT16T *)pbData)[i]);
}
else
    uart_printf("[DM9000][TX]Move data error!!!");
```

- 3) 将数据包长度的高字节和低字节分别写入 TXPLH 和 TXPLL 寄存器

```
/*set packet leng */
```

```
WRITE_REG1(0xfd, (length >> 8) & 0xff);
```

```
WRITE_REG1(0xfc, length & 0xff);
```

- 4) 开始传输数据包


```
/* start transmit */  
  
WRITE_REG1(0x02, 1);
```

5) 等待传输完成

若使用的中断方式，则在中断服务程序中应该清除对应的中断标志位。若为查询方式，则等待传输完成标志位置位后应该清除传输完成标志位（此程序使用的为查询方式）。

```
/*wait TX complete*/  
while(1)  
{  
    if (READ_REG1(0xfe) & 2) { //TX completed  
        WRITE_REG1(0xfe, 2);  
        break;  
    }  
}
```

如何接受数据包

接受到的数据包（包含有效数据和填充数据）保存在处于内部 SRAM0X0C00~0X3FF(13K)的接受缓冲队列中。每个接受到的包有 4 个字节的 MAC 头，通过 MRCMDX 和 MRCMD 两个寄存器来读取所到报的信息。

在前 4 个字节的 MAC 头中，第一个字节用来核查接受缓冲队列中是接受到新包。若为 0x01，则代表有新包；若为 00，则无新包。在度随后字节信息前，要确保 MAC 头的第一个字节的最低位 Bit[0]为 1。若第一个字节的 Bit[1:0]既不是 01 也不是 00，则必须重启 DM9000MAC/PHY，使系统总线 DM9000 恢复到稳定状态。第二个字节保存接受包的状态信息，其格式和 RSR 寄存器对应。根据这些状态位，可以核查此包是正确包还是错误包。第三和四个字节保存的是接受包的长度。其它字节为要接受的数据。下图描述了接收包的帧格式。

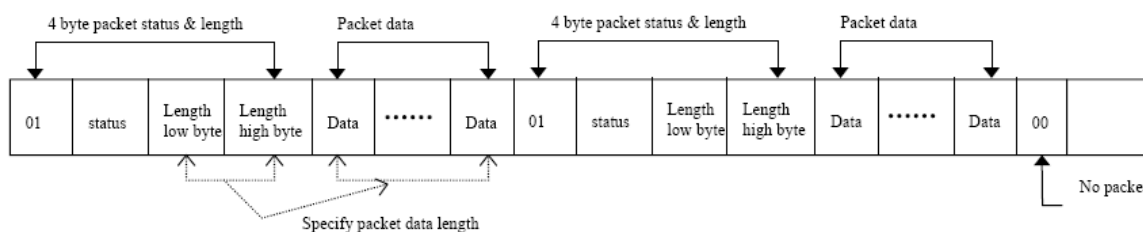


图 7-2-6 接收包的模块图

接收数据包的具体步骤如下：

1) 等待数据接收完成

若使用的中断方式，则在中断服务程序中应该清除对应的中断标志位。若为查询方式，则等待接收完成标志位置位后应该清除接收完成标志位（此程序使用的为查询方式）。

```
RxRead=READ_REG1(0xFE);
```

```

if(RxRead&0x01==0) return -1;

/* clean ISR */

WRITE_REG1(0xfe,READ_REG1(0xfe));

```

2) 读包的第一个字节，看是否为有效包

向 INDEX 端口写入 0xF0 来发出读接收缓冲区命令后，再读取 DATA 端口可以取得缓冲区数据，而且队列指针不自加。MRCMDX 寄存器仅用来读取接收包准备标志（是否有包，包是否有效）。

```

/* read the first byte*/
RxRead = READ_REG1(0xf0);
RxRead = IOREAD(dwEthernetDataPort);
/* the fist byte is ready or not */
if ((RxRead & 3) != 1) /* no data */
{
    return -1;
}

```

3) 读取包的状态和长度

向 INDEX 端口写入 0XF2 来发出读接收缓冲区命令后。再读取 DATA 端口可以取得缓冲区数据，而且队列指针自加。自加的字节数依据 DM9000 的数据总线宽度而定。MRCMD 寄存器仅用来读取接收包的接收状态、长度和随后包数据。

```

IOWRITE(dwEthernetIOBase, 0xf2); /* set read ptr ++ */
if (DM9000_iomode== DM9000_BYTE_MODE)
{
    status = IOREAD(dwEthernetDataPort)+(IOREAD(dwEthernetDataPort)<<8);
    rxlen = IOREAD(dwEthernetDataPort) + (IOREAD(dwEthernetDataPort)<<8);
}
else if (DM9000_iomode==DM9000_WORD_MODE)
{
    status = IOREAD16(dwEthernetDataPort);
    rxlen = IOREAD16(dwEthernetDataPort);
}
else
    uart_printf("[DM9000]Get status and rxlen error!!!");

```

4) 接收包数据

当通过向 INDEX 端口写入 0XF2 来发出读接收缓冲区命令后，就可以根据包长度通过读数据端口将接收缓冲区的数据读到指定位置。

```

/* move data from FIFO to memory */
If(switch (DM9000_iomode==DM9000_BYTE_MODE)

```

```
{
    tmplen = rxlen ;
    for (i = 0; i < tmplen; i++)
        ((UINT8T *)pbData)[i] = IOREAD(dwEthernetDataPort);
}
else if (DM9000_iomode==DM9000_WORD_MODE)
{
    tmplen = (rxlen+1)/2;
    for (i = 0; i < tmplen; i++)
        ((UINT16T *)pbData)[i] = IOREAD16(dwEthernetDataPort);
}
```

7. 2. 5 实验步骤

1. 准备实验环境

使用 ULINK2 仿真器连接 Embest EduKit-IV 实验平台的主板 JTAG 接口; 使用 Embest EduKit-IV 实验平台附带的交叉串口线, 连接实验平台主板上的 COM2 和 PC 机的串口(一般 PC 只有一个串口, 如果有多个请自行选择, 笔记本没有串口设备的可购买 USB 转串口适配器扩充); 使用 Embest EduKit-IV 实验平台附带的电源适配器, 连接实验平台主板上的电源接口。用直连网线连接 PC 机器和开发板。

2. 串口接收设置

在 PC 机上运行 windows 自带的超级终端串口通信程序, 或者使用实验平台附带光盘内设置好了的超级终端, 设置超级终端: 波特率 115200、1 位停止位、无校验位、无硬件流控制, 或者使用其它串口通信程序。(注: 超级终端串口的选择根据用户的 PC 串口硬件不同, 请自行选择, 如果 PC 机只有一个串口, 一般是 COM1)

3. 打开实验例程

1) 拷贝实验平台附带光盘 DISK3_S3C2410\03-Codes\01-MDK\Mini2410-IV 文件夹到 MDK 的安装路径: Keil\ARM\Boards\Embest\ (如果本实验之前已经拷贝, 可以跳过这一步)。(注: 用户也可拷贝工程到任意目录, 本实验为了便于教学, 故统一实验路径);

2) 运行 μ Vision IDE for ARM 软件, 点击菜单栏 “Project”, 选择 “Open Project...”, 在弹出的对话框选择实验例程目录 7.2_TFTP_Test 子目录下的 TFTP_Test.Uv2 工程。

3) 默认打开的工程在源码编辑窗口会显示实验例程的说明文件 readme.txt, 详细阅读并理解实验内容。

4) 工程提供了两种运行方式: 一是下载到 SDRAM 中调试运行, 二是固化到 Nor Flash 中运行。

用户可以在工具栏 Select Target 下拉框中选择在 RAM 中调试运行还是固化 Flash 中运行。如下图所示:

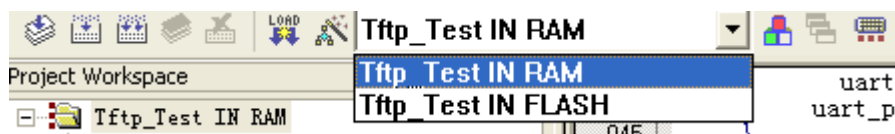



图 6-5-1 选择运行方式

下面实验将介绍下载到 SDRAM 中调试运行，所以我们在 Select Target 下拉框中选择 Uart_Test IN RAM。

5) 接下来开始编译链接工程，在菜单栏“Projiet”选择“Build target”或者“Rebuild all target files”编译整个工程，用户也可以在工具栏单击“”或者“”进行编译。


6) 编译完成后，在输出窗口可以看到编译提示信息，比如““.SDRAM\TFTP_Test.axf” - 0 Error(s), 1 Warning(s).”，如果显示“0 Error(s)”即表示编译成功。

7) 拨动实验平台电源开关，给实验平台上电，单击菜单栏 Debug->Start/Stop Debug Session 项将编译出来的映像文件下载到 SDRAM 中，或者单击工具栏“”按钮来下载。

8) 下载完成后，单击菜单栏 Debug->Run 项运行程序，或者单击工具栏“”按钮来全速运行程序。用户也可以使用进行单步调试程序。

9) 全速运行后，用户可以在超级终端看到程序运行的信息，出现“Please input words, then press Enter”提示后输入想要发送的数据，并已回车作为发送字符串的结尾标志。

10) 用户可以 Stop 程序运行，使用 μVision IDE for ARM 的一些调试窗口跟踪查看程序运行的信息。

注：如果在第 4) 步用户选择在 Flash 中运行，则编译链接成功后，单击菜单栏 Flash->Download 项将程序固化到 NorFlash 中，或者单击工具栏按钮“”固化程序，从实验平台的主板拔出 JTAG 线，给实验平台重新上电，程序将自动运行。

4. 观察实验结果

在执行到第 8) 步时，可以看到超级终端上输出等待输入字符：

```
boot success...

FS2410XP TFTP Test,please Enter 'ESC' to exit

Mini TFTP Server 1.0 (IP : 192.168.2.111 PORT: 69)

Type tftp -i 192.168.2.111 put filename at the host PC

detected DM9000...

dwEthernetIOBase = 20000000
```

```
dwEthernetDataPort = 20100000

id_val = 90000a46

INFO: Probe: DM9000 is detected.

INFO:CHIP Revision is:25

DM9000: MAC Address: 0:0:50:18:51:18

dm9000_hash_table

link stauts = 40

INFO: Init: DM9000_Init OK.

Starting the TFTP download...

Copy data!

Copy data end!

download 0x16 bytes to 0x30008000

0123456789!@#$^&*()_+

Press any key to continue...
```

以上为用户分别传输了在 C 盘下的一个文本文件。文件中的内容为：0123456789!@#\$^&*()_+

5. 完成实验练习题

理解和掌握实验后，完成实验练习题。

7. 2. 6 实验参考程序

1. DM9000 初始化程序

```
int DM9000DBG_Init(void )
{
    dwEthernetIOBase    = 0x20000000;
    dwEthernetDataPort = 0x20100000;

    r = Probe(); /*Detect DM9000 */
    MacAddr[0] = 0x0000;
    MacAddr[1] = 0x1232;
    MacAddr[2] = 0x1233;
```



```

if(r == FALSE)    return FALSE;

/* set the internal PHY power-on, GPIOs normal */
WRITE_REG1(0x1f, 0); /* GPR (reg_1Fh) bit GPIO0=0 pre-activate PHY */
DM9000_Delay(1000);

/* do a software reset */
WRITE_REG1(0x0, 3); /* NCR (reg_00h) bit[0] RST=1 & Loopback=1, reset on */
DM9000_Delay(1000);
WRITE_REG1(0x0, 3); /* NCR (reg_00h) bit[0] RST=1 & Loopback=1, reset on */
DM9000_Delay(1000);
/* I/O mode */
DM9000_iomode = READ_REG1(0xfe) >> 6; /* ISR bit7:6 keeps I/O mode */

/* Program operating register */
WRITE_REG1(0x0, 0);
WRITE_REG1(0x02, 0); /* TX Polling clear */
WRITE_REG1(0x2f, 0); /* Special Mode */
WRITE_REG1(0x01, 0x2c); /* clear TX status */
WRITE_REG1(0xfe, 0x0f); /* Clear interrupt status */
/* Set address filter table */
dm9000_hash_table(MacAddr);
/* Activate DM9000A/DM9010 */
WRITE_REG1(0x05, 0x30 | 1); /* Discard long packet and CRC error packets */ /* RX enable */
WRITE_REG1(0xff, 0x80); /* Enable SRAM automatically return */
/* wait link ok */
while(1)
{
    temp=READ_REG1(0x01)&0x40;
    if(temp) break;
}
return r;
}

```

2. 相关协议数据结构

```

//ARP 协议头结构
packed struct arphdr
{
    unsigned short  ar_hrd;        /* format of hardware address */
    unsigned short  ar_pro;        /* format of protocol address */
}

```

```
    unsigned char  ar_hln;          /* length of hardware address */
    unsigned char  ar_pln;          /* length of protocol address */
    unsigned short ar_op;           /* ARP opcode (command) */
    unsigned char  ar_sha[ETH_ALEN]; /* sender hardware address */
    unsigned long  ar_sip;          /* sender IP address */
    unsigned char  ar_tha[ETH_ALEN]; /* target hardware address */
    unsigned long  ar_tip;          /* target IP address */
};

//IP 协议头结构
__packed struct iphdr {

    unsigned ihl:4;
    unsigned version:4;
    unsigned char  tos;
    unsigned short tot_len;
    unsigned short id;
    unsigned short frag_off;
    unsigned char  ttl;
    unsigned char  protocol;
    unsigned short check;
    unsigned long  saddr;
    unsigned long  daddr;
    /*The options start here. */
};

//UDP 协议头结构
__packed struct udphdr {
    unsigned short source;
    unsigned short dest;
    unsigned short len;
    unsigned short check;
};

//TFTP 协议头结构
__packed struct tftphdr {
    short th_opcode;          /* packet type */
    __packed union {
        unsigned short tu_block; /* block # */
        short tu_code;          /* error code */
        char tu_stuff[1];       /* request packet stuff */
    };
};
```

7. 2. 7 练习题

改写 TFTP_TEST 例程，改变实验板 IP 地址，下载的时候改变 Flash 起始地址，重新进行实验，检查是否正确下载。

7. 3 IIS 音频实验

7. 3. 1 实验目的

- Ø 掌握有关音频处理的基础知识；
- Ø 通过实验了解 IIS 音频接口的工作原理；
- Ø 通过实验掌握对处理器 S3C2410X 中 IIS 模块电路的控制方法；
- Ø 通过实验掌握对常用 IIS 接口音频芯片的控制方法。

7. 3. 2 实验设备

- Ø 硬件：Embest EduKit-IV 平台，ULINK2 仿真器套件，PC 机；
- Ø 软件：μVision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

7. 3. 3 实验内容

将从 UART1 接收到的字符串回送显示。

7. 3. 4 实验原理

19. 数字音频基础

采样频率和采样精度

在数字音频系统中，通过将声波波形转换成一连串的二进制数据再现原始声音，这个过程中使用的设备是模拟/数字转换器（Analog to Digital Converter，即 ADC），ADC 以每秒上万次的速率对声波进行采样，每次采样都记录下了原始声波在某一时刻的状态，称之为样本。

每秒采样的数目称为采样频率，单位为 HZ（赫兹）。采样频率越高所能描述的声波频率就越高。系统对于每个样本均会分配一定存储位（bit 数）来表达声波的声波振幅状态，称之为采样精度。采

样频率和精度共同保证了声音还原的质量。

人耳的听觉范围通常是 20Hz~20KHz，根据奈魁斯特（NYQUIST）采样定理，用两倍于一个正弦波的频率进行采样能够真实地还原该波形，因此当采样频率高于 40KHz 时可以保证不产生失真。

CD 音频的采样规格为 16bit，44KHz，就是根据以上原理制定。

音频编码：脉冲编码调制 PCM（Pulse Code Modulation）编码的方法是对语音信号进行采样，然后对每个样值进行量化编码，在“采样频率和采样精度”中对语音量化和编码就是一个 PCM 编码过程。ITU-T 的 64kbit / s 语音编码标准 G.711 采用 PCM 编码方式，采样速率为 8KHz，每个样值用 8bit 非线性的 μ 律或 A 律进行编码，总速率为 64kbit / s。

CD 音频即是使用 PCM 编码格式，采样频率 44KHz，采样值使用 16bit 编码。

使用 PCM 编码的文件在 Windows 系统中保存的文件格式一般为大家熟悉的 wav 格式，实验中用到的就是一个采样 44.100KHz，16 位立体声文件 t.wav。

在 PCM 基础上发展起来的还有自适应差分脉冲编码调制 ADPCM（Adaptive Differential Pulse Code Modulation）。ADPCM 编码的方法是对输入样值进行自适应预测，然后对预测误差进行量化编码。CCITT 的 32kbit / s 语音编码标准 G.721 采用 ADPCM 编码方式，每个语音采样值相当于使用 4bit 进行编码。

其他编码方式还有线性预测编码 LPC（Linear Predictive Coding），低时延码激励线性预测编码 LD-CELP（Low Delay-Code Excited Linear Prediction）等。

目前流行的一些音频编码格式还有 MP3（MPEG Audio Layer-3），WMA（Windows Media Audio），RA（RealAudio），它们有一个共同特点就是压缩比高，主要针对网络传输，支持边读边放。

20. IIS 音频接口

IIS (Inter-IC Sound) 是一种串行总线设计技术，是 SONY、PHILIPS 等电子巨头共同推出的接口标准，主要针对数字音频处理技术和设备如便携 CD 机、数字音频处理器等。IIS 将音频数据和时钟信号分离，避免由时钟带来的抖动问题，因此系统中不再需要消除抖动的器件。

IIS 总线仅处理音频数据，其它信号如控制信号等单独传送，基于减少引脚数目和布线简单的目的，IIS 总线只由三根串行线组成：时分复用的数据通道线，字选择线和时钟线。

continuous serial clock (SCK);

word select (WS);

serial data (SD);

使用 IIS 技术设计的系统的连接配置参见图 7-3-1。

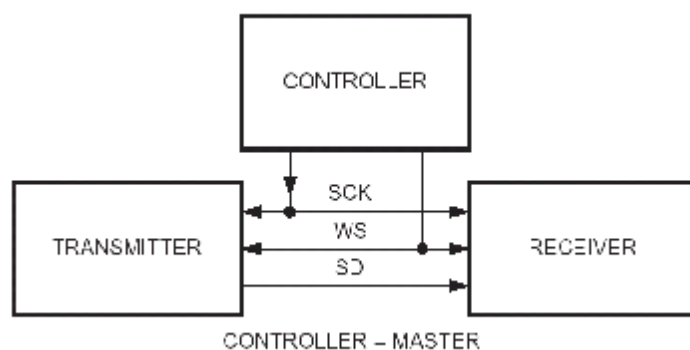


图 7-3-1 IIS 系统连接简单配置图

IIS 总线接口的基本时序参见图 7-3-2。

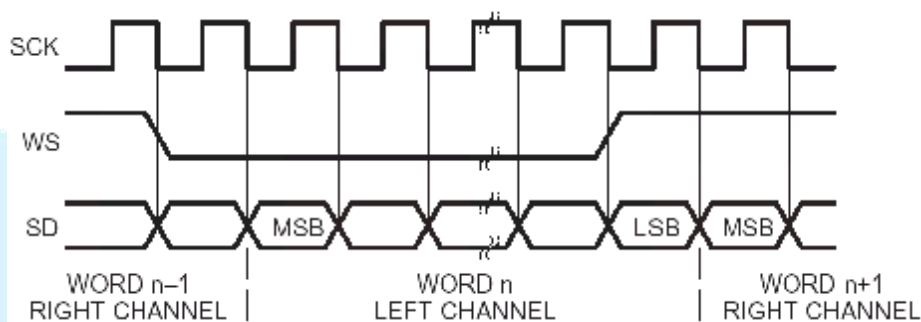


图 7-3-2 IIS 接口基本时序图

WS 信号线指示左通道或右通道的数据将被传输，SD 信号线按高有效位 MSB 到低有效位 LSB 的顺序传送字长的音频数据，MSB 总在 WS 切换后的第一个时钟发送，如果数据长度不匹配，接收器和发送器将自动截取或填充。关于 IIS 总线的其它细节可参见《I2S bus specification》。

在实验中，IIS 总线接口由处理器 S3C2410 的 IIS 模块和音频芯片 UDA1341 硬件实现，我们需要关注的是正确的配置 IIS 模块和 UDA1341 芯片，音频数据的传输反而比较简单。

21. 电路设计原理

S3C2410X 外围模块 IIS 说明

(1) 信号线

处理器中与 IIS 相关的信号线有五根：

- Ø 串行数据输入 IISDI，对应 IIS 总线接口中的 SD 信号，方向为输入。
- Ø 串行数据输出 IISDO，对应 IIS 总线接口中的 SD 信号，方向为输出。
- Ø 左右通道选择 IISLRCK，对应 IIS 总线接口中的 WS 信号，即采样时钟。
- Ø 串行位时钟 IISCLK，对应 IIS 总线接口中的 SCK 信号。
- Ø 音频系统主时钟 CODECLK，一般为采样频率的 256 倍或 384 倍，符号为 256fs 或 384fs，其中 fs 为采样频率。CODECLK 通过处理器主时钟分频获得，可以通过在程序中设定分频

寄存器获取, 分频因子可以设为 1 到 32。CODECLK 与采样频率的对应表格见下表, 实验中需要正确的选择 IISLRCK 和 CODECLK。

表 7-3-1 音频主时钟和采样频率的对应表

IISLRCK (fs)	8,000 KHz	11,025 KHz	16,000 KHz	22,050 KHz	32,000 KHz	44,100 KHz	48,000 KHz	64,000 KHz	88,200 KHz	96,000 KHz
CODECLK (MHz)	256fs									
	2.0460	2.8224	4.0960	5.6448	8.1920	11.2088	12.2880	16.3840	22.5792	24.5760
	384fs									
	3.0720	4.2336	6.1440	8.4672	12.2880	16.9344	18.4320	24.5760	33.8688	36.8640

需要注意的是, 处理器主时钟可以通过配置锁相环寄存器进行调整, 结合 CODECLK 的分频寄存器设置, 可以获得所需要的 CODECLK。

(2) 寄存器

处理器中与 IIS 相关的寄存器有三个:

IIS 控制寄存器 IISCON, 通过该寄存器可以获取数据高速缓存 FIFO 的准备好状态, 启动或停止发送和接收时的 DMA 请求, 使能 IISLRCK、分频功能和 IIS 接口。

IIS 模式寄存器 IISMOD, 该寄存器选择主/从、发送/接收模式, 设置有效电平、通道数据位, 选择 CODECLK 和 IISLRCK 频率。

IIS 分频寄存器 IISPSR。

(3) 数据传送

数据传送可以选择普通模式或者 DMA 模式, 普通模式下, 处理器根据 FIFO 的准备状态传送数据到 FIFO, 处理器自动完成数据从 FIFO 到 IIS 总线的发送, FIFO 的准备状态通过 IIS 的 FIFO 控制寄存器 IISFCON 获取, 数据直接写入 FIFO 寄存器 IISFIF。DMA 模式下, 对 FIFO 的访问和控制完全由 DMA 控制器完成, DMA 控制器自动根据 FIFO 的状态发送或接收数据。

DMA 方式下数据的传送细节请参考处理器手册中 DMA 章节。

音频芯片 UDA1341TS 说明

电路中使用的音频芯片是 PHILIPS 的 UDA1341TS 音频数字信号编译码器, UDA1341TS 可将立体声模拟信号转化为数字信号, 同样也能把数字信号转换成模拟信号, 并可用 PGA (可编程增益控制), AGC(自动增益控制)对模拟信号进行处理; 对于数字信号, 该芯片提供了 DSP(数字音频处理)功能。实际使用中, UDA1341TS 广泛应用于 MD、CD、notebook、PC 和数码摄像机等。

UDA1341TS 提供两组音频输入信号线、一组音频信号输出线, 一组 IIS 总线接口信号, 一组 L3 总线。

IIS 总线接口信号线包括位时钟输入 BCK、字选择输入 WS、数据输入 DATAI、数据输出 DATAO 和音频系统时钟 SYSCLK 信号线。

UDA1341TS 的 L3 总线, 包括微处理器接口数据 L3DATA、微处理器接口模式 L3MODE、微处理器接口时钟 L3CLOCK 三根信号线, 当该芯片工作于微控制器输入模式使用的, 微处理器通过 L3

总线对 UDA1341TS 中的数字音频处理参数和系统控制参数进行配置。处理器 S3C2410X 中没有 L3 总线专用接口，电路中使用 I/O 口连接 L3 总线。L3 总线的接口时序和控制方式参见 UDA1341TS 手册。

电路连接：

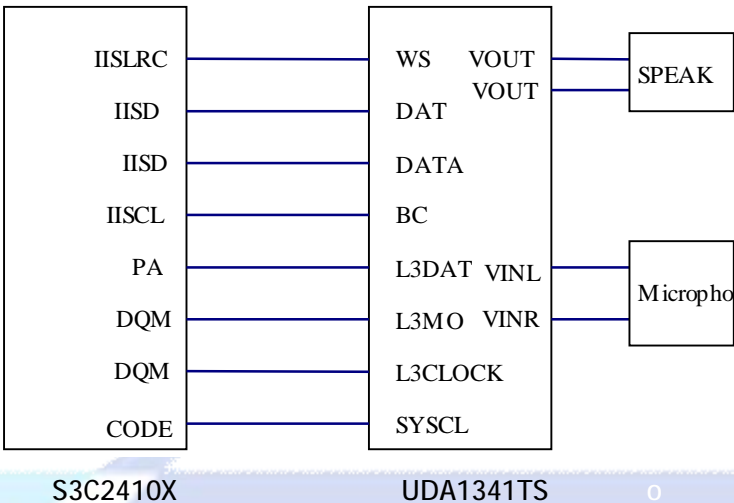


图 7-3-3 IIS 接口电路

7. 3. 5 实验步骤

1. 准备实验环境

使用 ULINK2 仿真器连接 Embest EduKit-IV 实验平台的主板 JTAG 接口；使用 Embest EduKit-IV 实验平台附带的交叉串口线，连接实验平台主板上的 COM2 和 PC 机的串口（一般 PC 只有一个串口，如果有多个请自行选择，笔记本没有串口设备的可购买 USB 转串口适配器扩充）；使用 Embest EduKit-IV 实验平台附带的电源适配器，连接实验平台主板上的电源接口。

2. 串口接收设置

在 PC 机上运行 windows 自带的超级终端串口通信程序，或者使用实验平台附带光盘内设置好了的超级终端，设置超级终端：波特率 115200、1 位停止位、无校验位、无硬件流控制，或者使用其它串口通信程序。（注：超级终端串口的选择根据用户的 PC 串口硬件不同，请自行选择，如果 PC 机只有一个串口，一般是 COM1）

3. 打开实验例程

1) 拷贝实验平台附带光盘 DISK3_S3C2410\03-Codes\01-MDK\Mini2410-IV 文件夹到 MDK 的安装路径：Keil\ARM\Boards\Embest\（如果本实验之前已经拷贝，可以跳过这一步）。（注：用户也可拷贝工程到任意目录，本实验为了便于教学，故统一实验路径）；

2) 运行 μ Vision IDE for ARM 软件, 点击菜单栏 “Project”, 选择 “Open Project...”, 在弹出的对话框选择实验例程目录 7.3_IIS_Test 子目录下的 IIS_Test.Uv2 工程。

3) 默认打开的工程在源码编辑窗口会显示实验例程的说明文件 readme.txt, 详细阅读并理解实验内容。

4) 工程提供了两种运行方式: 一是下载到 SDRAM 中调试运行, 二是固化到 Nor Flash 中运行。用户可以在工具栏 Select Target 下拉框中选择在 RAM 中调试运行还是固化 Flash 中运行。如下图所示:

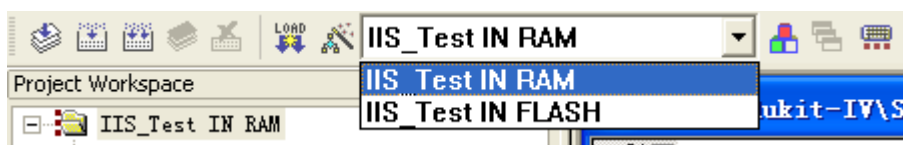


图 7-3-4 选择运行方式

下面实验将介绍下载到 SDRAM 中调试运行, 所以我们在 Select Target 下拉框中选择 Uart_Test IN RAM。

5) 接下来开始编译链接工程, 在菜单栏 “Projiet” 选择 “Build target” 或者 “Rebuild all target files” 编译整个工程, 用户也可以在工具栏单击 “

```
=====
|           WELCOME TO EMBEST EDUKIT IV           |
=====

IIS test example

Menu(press digital to select):

1: play wave file

2: record and play

3: EXIT!

1

Menu(press digital to select):

1: play wave file

2: record and play

3: EXIT!

2

Start recording....

End of record!!!

Press any key to play record data!!!

Play end!!!

Menu(press digital to select):

1: play wave file

2: record and play

3: EXIT!

3

end
```

5. 完成实验练习题

理解和掌握实验后，完成实验练习题。

7. 3. 6 实验参考程序

本实验的参考程序如下：

1. 初始化程序

```

/*****
* name:      iis_init
* func:      Initialize IIS circuit
* para:      none
* ret:       none
* modify:
* comment:
*****/

void iis_init(void)
{
    //-----
    //  PORT E GROUP
    //Ports : GPE4   GPE3   GPE2   GPE1   GPE0
    //Signal : I2SSDO I2SSDI CDCLK I2SSCLK I2SLRCK
    //Binary :  10   ,   10   10 ,  10   10
    //-----

    rGPEUP = rGPEUP | 0x1f;    //The pull up function is disabled GPE[4:0] 1 1111
    rGPECON = rGPECON & ~((1<<8)|(1<<6)|(1<<4)|(1<<2)|(1<<0)) |
        (1<<9)|(1<<7)|(1<<5)|(1<<3)|(1<<1);
    //GPE[4:0]=I2SSDO:I2SSDI:CDCLK:I2SSCLK:I2SLRCK
    f_nDMADone = 0;
    init_1341(PLAY);           // initialize philips UDA1341 chip
}

```

2. IIS 控制程序

```

/*****
* name:      iis_test
* func:
* para:      none
* ret: none
* modify:
* comment:
*****/

```



```

*****/

void iis_test(void)
{
    int nSoundLen=155956;
    iis_init();                // initialize IIS
    while((ucInput != '3') )
    {
        uart_printf(" Menu(press digital to select):\n");
        .....
        while((ucInput != '1') & (ucInput != '2') & (ucInput != '3'))
        {
            ucInput = uart_getkey();
        }
        switch(ucInput)
        {
            case '1':
            {
                uart_printf(" %c\n",ucInput);
                memcpy((void *)0x32000000, g_ucWave, nSoundLen);
                iis_play_wave(10,(UINT8T *)0x32000000,nSoundLen);//      nSoundLen = 155956;
                ucInput=0;
                break;
            }
            case '2':
            {
                uart_printf(" %c\n",ucInput);
                iis_record();
                ucInput=0;
                break;
            }
            case '3':
            {
                uart_printf(" %c\n",ucInput);
                uart_printf(" end.\n");
                iis_close();                // close IIS
            }
            default : ;
        }
    }
    .....}
}

```

7. 3. 7 练习题

编写程序实现通过按钮调整音量的大小。

7. 4 USB 接口实验

7. 4. 1 实验目的

- Ø 了解 S3C2410X 处理器 USB 接口相关控制寄存器的使用;
- Ø 了解 USB 接口基本原理;
- Ø 掌握通过 USB 接口与 PC 通讯的编程技术。

7. 4. 2 实验设备

- Ø 硬件: Embest EduKit-IV 平台, ULINK2 仿真器套件, PC 机;
- Ø 软件: µVision IDE for ARM 集成开发环境, Windows 98/2000/NT/XP。

7. 4. 3 实验内容

- Ø 为 S3C2410x 处理器集成的 USB Device 单元编写设备控制器驱动程序;
- Ø 实现 PC 机端 USB 主机与实验板 USB 设备进行数据接收与发送。

7. 4. 4 实验原理

1. USB 基础知识

(1) 定义

通用串行总线协议 USB (Universal Serial Bus) 是由 Intel、Compaq、Microsoft 等公司联合提出的一种新的串行总线标准, 主要用于 PC 机与外围设备的互联。1994 年 11 月发布第一个草案, 1996 年 2 月发布第一个规范版本 1.0, 2000 年 4 月发布高速模式版本 2.0, 对应的设备传输速度也从 1.5Mb/s 的低速和 12Mb/s 的全速提高到如今的 480Mb/s 的高速。其主要特点是:

支持即插即用: 允许外设主机和其它外设工作时进行连接配置使用及移除。

传输速度快: USB 支持三种设备传输速率: 低速设备 1.5 Mb/s、中速设备 12 Mb/s 和高速设备 480Mb/s。

连接方便: USB 可以通过串行连接或者使用集线器 Hub 连接 127 个 USB 设备, 从而以一个串行通道取代 PC 上其他 I/O 端口如串行口、并行口等, 使 PC 与外设之间的连接更容易。

独立供电: USB 接口提供了内置电源。

低成本: USB 使用一个 4 针插头作为标准插头, 通过这个标准插头, 采用菊花链形式可以把多

达 127 个的 USB 外设连接起来，所有的外设通过协议来共享 USB 的带宽。

(2) 组成

USB 规范中将 USB 分为五个部分：控制器、控制器驱动程序、USB 芯片驱动程序、USB 设备以及针对不同 USB 设备的客户驱动程序。

控制器 (Host Controller)，主要负责执行由控制器驱动程序发出的命令，如位于 PC 主板的 USB 控制芯片。

控制器驱动程序(Host Controller Driver)，在控制器与 USB 设备之间建立通信信道，一般由操作系统或控制器厂商提供。

USB 芯片驱动程序(USB Driver)，提供对 USB 芯片的支持，设备上的固件 (Firmware)。

USB 设备(USB Device)，包括与 PC 相连的 USB 外围设备。

设备驱动程序(Client Driver Software)，驱动 USB 设备的程序，一般由 USB 设备制造商提供。

(3) 传输方式

针对设备对系统资源需求的不同，在 USB 规范中规定了四种不同的数据传输方式：

同步传输 (Isochronous)，该方式用来联接需要连续传输数据，且对数据的正确性要求不高而对时间极为敏感的外部设备，如麦克风、喇叭以及电话等。同步传输方式以固定的传输速率，连续不断地在主机与 USB 设备之间传输数据，在传送数据发生错误时，USB 并不处理这些错误，而是继续传送新的数据。同步传输方式的发送方和接收方都必须保证传输速率的匹配，不然会造成数据的丢失。

中断传输 (Interrupt)，该方式用来传送数据量较小，但需要及时处理，以达到实时效果的设备，此方式主要用在偶然需要少量数据通信，但服务时间受限制的键盘、鼠标以及操纵杆等设备上。

控制传输 (Control)，该方式用来处理主机到 USB 设备的数据传输，包括设备控制指令、设备状态查询及确认命令，当 USB 设备收到这些数据和命令后，将依据先进先出的原则处理到达的数据。主要用于主机把命令传给设备、及设备把状态返回给主机。任何一个 USB 设备都必须支持一个与控制类型相对应的端点 0。

批量传输 (Bulk)，该方式不能保证传输的速率，但可保证数据的可靠性，当出现错误时，会要求发送方重发。通常打印机、扫描仪和数字相机以这种方式与主机联接。

(4) 关键词

USB 主机(Host) USB 主机控制总线上所有的 USB 设备和所有集线器的数据通信过程，一个 USB 系统中只有一个 USB 主机，USB 主机检测 USB 设备的连接和断开、管理主机和设备之间的标准控制管道、管理主机和设备之间的数据流、收集设备的状态和统计总线的活动、控制和管理主机控制器与设备之间的电气接口，每一毫秒产生一帧数据，同时对总线上的错误进行管理和恢复。

USB 设备 (Device) 通过总线与 USB 主机相连的称为 USB 设备。USB 设备接收 USB 总线上的

所有数据包, 根据数据包的地址域来判断是否接收; 接收后通过响应 USB 主机的数据包与 USB 主机进行数据传输。

端点 (Endpoint) 端点是位于 USB 设备中与 USB 主机进行通信的基本单元。每个设备允许有多个端点, 主机只能通过端点与设备进行通讯, 各个端点由设备地址和端点号确定在 USB 系统中唯一的地址。每个端点都包含一些属性: 传输方式、总线访问频率、带宽、端点号、数据包的最大容量等。除控制端点 0 外的其他端点必须在设备配置后才能生效, 控制端点 0 通常用于设备初始化参数。USB 芯片中, 每个端点实际上就是一个一定大小的数据缓冲区。

管道 (Pipe) 管道是 USB 设备和 USB 主机之间数据通信的逻辑通道, 一个 USB 管道对应一个设备端点, 各端点通过自己的管道与主机通信。所有设备都支持对应端点 0 的控制管道, 通过控制管道主机可以获取 USB 设备的信息, 包括: 设备类型、电源管理、配置、端点描述等。

2. USB 设备设计原理

USB 设备控制器提供多个标准的端点, 每个端点都支持单一的总线传输方式。端点 0 支持控制传输, 其他端点支持同步传输、批量传输或中断传输中的任意一种。管理和使用这些端点, 实际上就是通过操作相应的控制寄存器、状态寄存器、中断寄存器和数据寄存器来实现。其中, 控制寄存器用于设置端点的工作模式、启用端点的功能等; 状态寄存器用于查询端点的当前状态; 中断寄存器则用于设置端点的中断触发和响应功能; 数据寄存器则是设备与主机交换数据用的缓冲区。

(1) 关于 S3C2410X 的 USB 设备控制器

Embest EduKit-III 实验平台的 S3C2410X 上集成了一个 USB 设备控制器。它具体如下特性:

- 完全兼容 USB 1.1 规范
- 全速 (12Mbps) USB 设备
- 集成 USB 收发器
- 支持控制, 中断和批量传输
- 包含 5 个带有 FIFO 的端点 (1 个带 16 字节 FIFO 的双向控制端点和 4 个带 64 字节 FIFO 的双向支持批量传输的端点)
- 带有支持批量传输的 DMA 接口
- 支持挂起和远程唤醒功能

USB 设备控制器的寄存器说明

所有的寄存器都是通过字节或字方式进行访问, 在 little endian 和 big endian 方式下, 访问的偏移地址会有不同。下表列出了 USB 设备控制器的寄存器 (详细的介绍请参考 S3C2410X 的芯片手册)。

表 6-4-1 S3C2410X USB 设备控制器寄存器

寄存器名	描述	偏移地址
FUNX_ADDR_REG	功能地址寄存器	0x140(L)/0x143(B)

PWR_REG	电源管理寄存器	0x144(L)/0x147(B)
EP_INT_REG	端点中断寄存器	0x148(L)/0x14B(B)
USB_INT_REG	USB 中断寄存器	0x158(L)/0x15B(B)
EP_INT_EN_REG	端点中断使能寄存器	0x15C(L)/0x15F(B)
USB_INT_EN_REG	帧序号低字节寄存器	0x16C(L)/0x16F(B)
FRAME_NUM1_REG	帧序号高字节寄存器	0x170(L)/0x173(B)
FRAME_NUM2_REG	USB 中断使能寄存器	0x174(L)/0x177(B)
INDEX_REG	索引寄存器	0x178(L)/0x17B(B)
EP0_FIFO_REG	端点 0 的 FIFO 寄存器	0x1C0(L)/0x1C3(B)
EP1_FIFO_REG	端点 1FIFO 寄存器	0x1C4(L)/0x1C7(B)
EP2_FIFO_REG	端点 2FIFO 寄存器	0x1C8(L)/0x1CB(B)
EP3_FIFO_REG	端点 3FIFO 寄存器	0x1CC(L)/0x1CF(B)
EP4_FIFO_REG	端点 4FIFO 寄存器	0x1D0(L)/0x1D3(B)
EP1_DMA_CON	端点 1 DMA 控制寄存器	0x200(L)/0x203(B)
EP1_DMA_UNIT	端点 1 DMA 传输单元计数器	0x204(L)/0x207(B)
EP1_DMA_FIFO	端点 1 DMA FIFO 计数器	0x208(L)/0x20B(B)
EP1_DMA_TTC_L	端点 1 DMA 传输单元计数器低字节	0x20C(L)/0x20F(B)
EP1_DMA_TTC_M	端点 1 DMA 传输单元计数器中字节	0x210(L)/0x213(B)
EP1_DMA_TTC_H	端点 1 DMA 传输单元计数器高字节	0x214(L)/0x217(B)
EP2DMA_CON	端点 2 DMA 控制寄存器	0x218(L)/0x21B(B)
EP2DMA_UNIT	端点 2 DMA 传输单元计数器	0x21C(L)/0x21F(B)
EP2DMA_FIFO	端点 2 DMA FIFO 计数器	0x220(L)/0x223(B)
EP2DMA_TTC_L	端点 2 DMA 传输单元计数器低字节	0x224(L)/0x227(B)
EP2DMA_TTC_M	端点 2 DMA 传输单元计数器中字节	0x228(L)/0x22B(B)
EP2DMA_TTC_H	端点 2DMA 传输单元计数器高字节	0x22C(L)/0x22F(B)
EP3DMA_CON	端点 3 DMA 控制寄存器	0x240(L)/0x243(B)
EP3DMA_UNIT	端点 3 DMA 传输单元计数器	0x244 (L)/0x247(B)
EP3DMA_FIFO	端点 3 DMA FIFO 计数器	0x248(L)/0x24B(B)
EP3DMA_TTC_L	端点 3 DMA 传输单元计数器低字节	0x24C(L)/0x24F(B)
EP3DMA_TTC_M	端点 3 DMA 传输单元计数器中字节	0x250(L)/0x253(B)
EP3DMA_TTC_H	端点 3 DMA 传输单元计数器高字节	0x254(L)/0x257(B)
EP4DMA_CON	端点 4 DMA 控制寄存器	0x258(L)/0x25B(B)
EP4DMA_UNIT	端点 4 DMA 传输单元计数器	0x25C(L)/0x25F(B)
EP4DMA_FIFO	端点 4 DMA FIFO 计数器	0x260(L)/0x263(B)
EP4DMA_TTC_L	端点 4 DMA 传输单元计数器低字节	0x264(L)/0x267(B)
EP4DMA_TTC_M	端点 4 DMA 传输单元计数器中字节	0x268(L)/0x26B(B)
EP4DMA_TTC_H	端点 4 DMA 传输单元计数器高字节	0x26C(L)/0x26F(B)
MAXP_REG	端点最大包寄存器	0x18C(L)/0x18F(B)
IN_CSR1_REG/EP0_CSR	输入端点控制状态寄存器 1/端点 0 控制状	0x184(L)/0x187(B)

	态寄存器	
IN_CSR2_REG	输入端点控制状态寄存器 2	0x188(L)/0x18B(B)
OUT_CSR1_REG	输出端点控制状态寄存器 1	0x190(L)/0x193(B)
OUT_CSR2_REG	输出端点控制状态寄存器 2	0x194(L)/0x197(B)
OUT_FIFO_CNT1_REG	端点写输出字节数寄存器 1	0x198(L)/0x19B(B)
OUT_FIFO_CNT2_REG	端点写输出字节数寄存器 2	0x19C(L)/0x19F(B)

(2) 设备驱动程序设计

在所有的操作之前，必须对 S3C2410X 的杂项控制器进行如下设置：

```
rMISCCR=rMISCCR&~(1<<3);    //使用 USB 设备而不是 USB 主机功能
```

```
rMISCCR=rMISCCR&~(1<<13);   //使用 USB 端口 1 模式
```

初始化 USB

在使用 USB 之前必须要进行初始化。USB 主机和 USB 设备接口都需要 48Mhz 的时钟频率。在 S3C2410X 中，这个时钟是由 UPLL（USB 专用 PLL）来提供的。USB 初始化的第一步就是要对 UPLL 控制器进行设置。

```
ChangeUPLValue(40,1,2);        //UCLK=48Mhz
```

在 USB1.x 规范中，规定了 5 种标准的 USB 描述符：设备描述符（Device Descriptor）、配置描述符（Configuration Descriptor）、接口描述符（Interface Descriptor）、端点描述符（EndPoint Descriptor）和字符串描述符（String Descriptor）。每个 USB 设备只有一个设备描述符，而一个设备中可以包含一个或多个配置描述符，即 USB 设备可以有多种配置。设备的每一个配置中又可以包含一个或多个接口描述符，即 USB 设备可以支持多种功能（接口），接口的特性通过接口描述符提供。在 Embest EduKit-III 实验平台的 USB 设备中只有一种配置，支持一种功能。关于设备描述符表的初始化以及配置由下面的两个函数实现：

```
InitDescriptorTable();          //初始化描述符表
```

```
ConfigUsbd();                   //设备的配置
```

在 ConfigUsbd 函数中，将 USB 设备控制器的端点 0 设置为控制端点，端点 1 设置为批量输入端点，端点 3 设置为批量输出端点，端点 2 和端点 4 暂时没有使用，同时，还使能了端口 0，1，3 的中断和 USB 的复位中断。

```
rEP_INT_EN_REG=EP0_INT|EP1_INT|EP3_INT;
```

```
rUSB_INT_EN_REG=RESET_INT;
```

除此之外，初始化过程还对中断服务程序入口等进行了设置。

USB 中断

S3C2410X 能够接收 56 个中断源的请求，当它接收到来自 USB 设备的中断请求时就会将 SRCPND 寄存器的 INT_USBD 置位，经过仲裁之后，中断控制器向内核发送 IRQ 中断请求。

USB 中断服务例程

当内核接收 USB 设备的中断请求之后, 就会转入相应的中断服务程序运行。这个中断服务程序入口是在 USB 初始化时设置的。

```
pISR_USBD = (unsigned)IsrUsbd;
```

USB 读写

USB 设备的读写是通过管道来完成的。管道是 USB 设备和 USB 主机之间数据通信的逻辑通道, 它的物理介质就是 USB 系统中的数据线。在设备端, 管道的主体是“端点”, 每个端点占据各自的管道和 USB 主机通信。

所有的设备都需要有支持控制传输的端点, 协议将端点 0 定义为设备默认的控制端点。在设备正常工作之前, USB 主机必须为设备分配总线上唯一的设备地址, 并完成读取设备的各种描述符, 根据描述符的需求为设备的端点配置管道, 分配带宽等工作, 另外, 在设备的工作过程中, 主机希望及时的获取设备的当前状态。以上的过程是通过端点 0 来完成的。

USB 设备和主机之间的数据的接收和发送采用的是批量传输方式。端点 1 为批量输入端点, 端点 3 为批量输出端点 (输入和输出以 USB 主机为参考)。端点 3 数据的批量传输由 DMA 接口实现。

本实验假设读者对 USB 通信有一定的认识, 并由于篇幅有限, 本文不能对 USB 规范协议作更详细的介绍, 有兴趣的读者可以参考相关书籍。

7. 4. 5 实验步骤

1. 准备实验环境

使用 ULINK2 仿真器连接 Embest EduKit-IV 实验平台的主板 JTAG 接口; 使用 Embest EduKit-IV 实验平台附带的交叉串口线, 连接实验平台主板上的 COM2 和 PC 机的串口 (一般 PC 只有一个串口, 如果有多个请自行选择, 笔记本没有串口设备的可购买 USB 转串口适配器扩充); 使用 Embest EduKit-IV 实验平台附带的电源适配器, 连接实验平台主板上的电源接口。

2. 串口接收设置

在 PC 机上运行 windows 自带的超级终端串口通信程序, 或者使用实验平台附带光盘内设置好了的超级终端, 设置超级终端: 波特率 115200、1 位停止位、无校验位、无硬件流控制, 或者使用其它串口通信程序。(注: 超级终端串口的选择根据用户的 PC 串口硬件不同, 请自行选择, 如果 PC 机只有一个串口, 一般是 COM1)

3. 打开实验例程

1) 拷贝实验平台附带光盘实验光盘路径文件夹到 MDK 的安装路径: Keil\ARM\Boards\Embest\ (如果本实验之前已经拷贝, 可以跳过这一步)。(注: 用户也可拷贝工程到任意目录, 本实验为了便于教学, 故统一实验路径);

2) 运行 μ Vision IDE for ARM 软件, 点击菜单栏 “Project”, 选择 “Open Project...”, 在弹出的对话框选择实验例程目录 7.4_USB_Test 子目录下的 USB_Test.Uv2 工程。

3) 默认打开的工程在源码编辑窗口会显示实验例程的说明文件 readme.txt, 详细阅读并理解实验内容。

4) 工程提供了两种运行方式: 一是下载到 SDRAM 中调试运行, 二是固化到 Nor Flash 中运行。用户可以在工具栏 Select Target 下拉框中选择在 RAM 中调试运行还是固化 Flash 中运行。如下图所示:

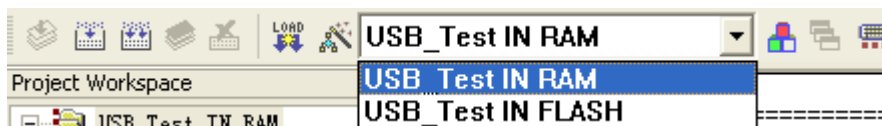



图 6-5-1 选择运行方式

下面实验将介绍下载到 SDRAM 中调试运行, 所以我们在 Select Target 下拉框中选择 USB_Test IN RAM。

5) 接下来开始编译链接工程, 在菜单栏 “Project” 选择 “Build target” 或者 “Rebuild all target files” 编译整个工程, 用户也可以在工具栏单击 “ ” 或者 “ ” 进行编译。


6) 编译完成后, 在输出窗口可以看到编译提示信息, 比如 “. \SDRAM\USB_Test.axf” - 0 Error(s), 1 Warning(s).”, 如果显示 “0 Error(s)” 即表示编译成功。

7) 拨动实验平台电源开关, 给实验平台上电, 单击菜单栏 Debug->Start/Stop Debug Session 项将编译出来的映像文件下载到 SDRAM 中, 或者单击工具栏 “ ” 按钮来下载。

8) 下载完成后, 单击菜单栏 Debug->Run 项运行程序, 或者单击工具栏 “ ” 按钮来全速运行程序。用户也可以使用进行单步调试程序。

9) 全速运行后, 用户可以在超级终端看到程序运行的信息, 此时将实验箱附带的 USB 线将 PC 和实验箱连接起来, 并在 PC 机上运行 DNW 软件, 若是第一次运行, Windows 会提示安装驱动, 选择从指定路径安装, 本实验 USB 驱动在例程目录下的 Driver 中。安装成功后, 在 DNW 的标题栏看到类似[USB:OK]信息并在超级终端上显示 USB host is connected. Waiting a download. 的信息。此时可键入空格。根据超级终端上的菜单进行操作。

10) 用户可以 Stop 程序运行, 使用 μ Vision IDE for ARM 的一些调试窗口跟踪查看程序运行的信息。

注: 如果在第 4) 步用户选择在 Flash 中运行, 则编译链接成功后, 单击菜单栏 Flash->Download 项将程序固化到 NorFlash 中, 或者单击工具栏按钮 “ ” 固化程序, 从实验平台的主板拔出 JTAG 线, 给实验平台重新上电, 程序将自动运行。然后再按步骤 9) 来操作。

4. 观察实验结果

在执行到第 8) 步时, 可以看到超级终端上输出以下字符:

此时将实验箱附带的 USB 线将 PC 和实验箱连接起来，并在 PC 机上运行 DNW 软件，若是第一次运行，Windows 会提示安装驱动，选择从指定路径安装，本实验 USB 驱动在例程目录下的 Driver 中。安装成功后，在 DNW 的标题栏看到类似[USB:OK]信息并在超级终端上显示。

键入空格之后,会有以下菜单:

读者可选择功能进行试验。

理解和掌握实验后，完成实验练习题。

7. 4. 6 实验参考程序

本实验的主控制参考程序如下：

```

/*****
* File:      usb_test.c
* Author:    Embest
* Desc:      USB_Test
* History:
*****/

#include "usb_test.h"

/*-----*/
/*                      global variablese                      */
/*-----*/

volatile UINT32T downloadAddress;

volatile unsigned char *downPt;
volatile UINT32T downloadFileSize;
volatile UINT16T checkSum;
volatile unsigned int err=0;
volatile UINT32T totalDmaCount;

volatile int isUsbdSetConfiguration;

int download_run=0;
UINT32T tempDownloadAddress;
UINT32T test_ram_start;
UINT32T test_ram_end;
int menuUsed=0;

UINT32T g_nKeyPress;
extern char Image$$RW_ZI$$ZI$$Limit[];
UINT32T *pMagicNum=(UINT32T *)Image$$RW_ZI$$ZI$$Limit;
int consoleNum;

void (*run)(void);

/*****
* name:      MemoryTest

```

```
* func:      Test Ram
* para:      none
* ret: none
* modify:
* comment:   comparing the data written to RAM and the data read from RAM
*****/

void MemoryTest(void)
{
    int i;
    UINT32T data;
    int memError=0;
    UINT32T *pt;

    // memory test
    uart_printf("\nEnter a start address of ram to test(0x3...):");
    test_ram_start = uart_getintnum();
    uart_printf("\nEnter a end address of ram to test(0x3...):");
    test_ram_end = uart_getintnum();
    uart_printf("Memory Test(%xh-%xh):WR",test_ram_start,test_ram_end);

    pt=(UINT32T *)(test_ram_start);
    while((UINT32T)pt<(test_ram_end))
    {
        *pt=(UINT32T)pt;
        pt++;
    }

    uart_printf("\b\bRD");
    pt=(UINT32T *)(test_ram_start);

    while((UINT32T)pt<(test_ram_end))
    {
        data=*pt;
        if(data!=(UINT32T)pt)
        {
            memError=1;
            uart_printf("\b\bFAIL:0x%x=0x%x\n",i,data);
            break;
        }
    }
}
```



```
        pt++;
    }

    if(memError==0)uart_printf("\b\bO.K.\n");
}

/*****
* name:      usb_test
* func:      usb monitor entry
* para:      none
* ret: none
* modify:
* comment:
*****/
void usb_test(void)
{
    char *mode;

    rGPHCON = rGPHCON&~(0xf<<18)|(0x5<<18);

    Isr_Init();
    if(*pMagicNum!=0x12345678)
        consoleNum=0;
    else
        consoleNum=1;

    rMISCCR=rMISCCR&~(1<<3);    // USB0 is selected instead of USBH1
    rMISCCR=rMISCCR&~(1<<13);   // USB port 1 is enabled.

    // USB0 should be initialized first of all.
    isUsbdSetConfiguration=0;

    UsbdMain();
    delay(0);

    pISR_SWI=(_ISR_STARTADDRESS+0xf0);
```

```

#if USBDMA
    mode="DMA";
#else
    mode="Int";
#endif

    uart_printf("\n");
    uart_printf("+-----+\n");
    uart_printf("| S3C2410X USB Downloader ver R1.11 APR/29/05 |\n");
    uart_printf("+-----+\n");
    uart_printf("FCLK=%dMHz,%s mode\n",FCLK/1000000,mode);
    uart_printf("USB: IN_ENDPOINT:1 OUT_ENDPOINT:3\n");
    uart_printf("FORMAT: <ADDR(DATA):4>+<SIZE(n+10):4>+<DATA:n>+<CS:2>\n");
    uart_printf("NOTE: 1. Power off/on or press the reset button for 1 sec\n");
    uart_printf("        in order to get a valid USB device address.\n");
    uart_printf("        2. For additional menu, Press any key. \n");
    uart_printf("\n");

    download_run=0; //The default menu is the Download mode.

    while(1)
    {
        if(menuUsed==1) Menu();
        WaitDownload();
    }

}

/*****
* name:      Menu
* func:      usb monitor Menu
* para:      none
* ret: none
* modify:
* comment:
*****/
void Menu(void)
{

```

```
UINT8T key;
menuUsed=1;
g_nKeyPress = 1;
while(g_nKeyPress==1)                                // only for board test to exit
{
    uart_printf("\n##### Select Menu #####\n");
    uart_printf(" [1] Download Only\n");
    uart_printf(" [2] Test SDRAM \n");
    uart_printf(" [3] Change The Console UART Ch.\n");

    while(g_nKeyPress==1)
    {
        key = uart_getkey();
        break;
    }

    switch(key)
    {

    case '1':
        uart_printf("\nDownload Only is selected.\n");
        uart_printf("\nEnter a new temporary download address(0x3...):");
        tempDownloadAddress=uart_getintnum();
        download_run=0;
        uart_printf("\nThe temporary download address is 0x%x.\n\n",tempDownloadAddress);
        return;
    case '2':
        uart_printf("\nMemory Test is selected. please wait a little long time.\n");

        MemoryTest();
        Menu();
        return;
    case '3':
        uart_printf("\nWhich UART channel do you want to use for the console?[0/1]\n");
        if(uart_getch()!='1')
        {
            *pMagicNum=0x0;
            uart_printf("UART ch.0 will be used for console at next boot.\n");
        }
    }
```

```
else
{
    *pMagicNum=0x12345678;
    uart_printf("UART ch.1 will be used for console at next boot.\n");
    uart_printf("UART ch.0 will be used after long power-off.\n");
}

uart_printf("System is waiting for a reset. Please, Reboot!!!\n");
while(1);

default:
    break;
}
}

}

/*****
* name:      WaitDownload
* func:      usb downloader
* para:      none
* ret: none
* modify:
* comment:
*****/

void WaitDownload(void)
{
    UINT32T i;
    UINT32T j;
    UINT16T cs;
    UINT32T temp;
    UINT16T dnCS;
    int first=1;
    float time;
    UINT8T tempMem[16];
    UINT8T key;

    checkSum=0;
    downloadAddress=(UINT32T)tempMem;
    downPt=(unsigned char *)downloadAddress;
```

```
downloadFileSize=0;

/*****
/*   Test program download   */
*****/

if(isUsbdSetConfiguration==0)
{
    uart_printf("USB host is not connected yet.\n");
}

while(downloadFileSize==0)
{
    if(first==1 && isUsbdSetConfiguration!=0)
    {
        uart_printf("USB host is connected. Waiting a download.\n");
        first=0;
    }

    key=uart_getkey();
    if(key!=0)
    {
        Menu();
        first=1;                //To display the message,"USB host ...."
    }

}

timer_initex();
timer_startex();

#if USBDMA

rINTMSK &= ~(BIT_DMA2);

ClearEp3OutPktReady();

// indicate the first packit is processed.
// has been delayed for DMA2 cofiguration.
```

```
if(downloadFileSize>EP3_PKT_SIZE)
{
    if(downloadFileSize<=(0x80000))
    {

        ConfigEp3DmaMode(downloadAddress+EP3_PKT_SIZE-8,downloadFileSize-EP3_PKT_SIZE);

        //wait until DMA reload occurs.
        while((rDSTAT2&0xffff)==0);

        //will not be used.
        rDIDST2=(downloadAddress+downloadFileSize-EP3_PKT_SIZE);
        rDIDSTC2=(0<<1)|(0<<0);
        rDCON2=rDCON2&~(0xffff)|(0);
    }
    else
    {
        ConfigEp3DmaMode(downloadAddress+EP3_PKT_SIZE-8,0x80000-EP3_PKT_SIZE);
        //wait until DMA reload occurs.
        while((rDSTAT2&0xffff)==0);

        if(downloadFileSize>(0x80000*2))//for 1st autoreload
        {
            rDIDST2=(downloadAddress+0x80000-8); //for 1st autoreload.
            rDIDSTC2=(0<<1)|(0<<0);
            rDCON2=rDCON2&~(0xffff)|(0x80000);

            while(rEP3_DMA_TTC<0xffff)
            {
                rEP3_DMA_TTC_L=0xff;
                rEP3_DMA_TTC_M=0xff;
                rEP3_DMA_TTC_H=0xf;
            }
        }
        else
        {
            rDIDST2=(downloadAddress+0x80000-8); //for 1st autoreload.
            rDIDSTC2=(0<<1)|(0<<0);
```



```
{
    if( ((UINT32T)downPt-downloadAddress)>=j)
    {
        uart_printf("\b\b\b\b\b\b\b\b%8d",j);
        j+=0x10000;
    }
}

#endif

time=timer_stopex();
uart_printf("\b\b\b\b\b\b\b\b%8d",downloadFileSize);
uart_printf("%5.1fKB/S,%3.1fS)\n",(float)(downloadFileSize/time/1000.),time);

#if USBDMA

/*****
/*      Verify check sum          */
*****/

uart_printf("Now, Checksum calculation\n");

cs=0;
i=(downloadAddress);
j=(downloadAddress+downloadFileSize-10)&0xffffffc;
while(i<j)
{
    temp=((UINT32T *)i);
    i+=4;
    cs+=(UINT16T)(temp&0xff);
    cs+=(UINT16T)((temp&0xff00)>>8);
    cs+=(UINT16T)((temp&0xff0000)>>16);
    cs+=(UINT16T)((temp&0xff000000)>>24);
}

i=(downloadAddress+downloadFileSize-10)&0xffffffc;
j=(downloadAddress+downloadFileSize-10);
while(i<j)
{
    cs+=*((UINT8T *)i++);
}
```

```
    checksum=cs;
#else
    //checksum was calculated including dnCS. So, dnCS should be subtracted.
    checksum=checksum - *((unsigned char*)(downloadAddress+downloadFileSize-8-2))
        - *((unsigned char*)(downloadAddress+downloadFileSize-8-1));
#endif

    dnCS=*((unsigned char*)(downloadAddress+downloadFileSize-8-2))+
        *((unsigned char*)(downloadAddress+downloadFileSize-8-1))<<8;

    if(downloadFileSize<1024*4)
        for (i=0;i<downloadFileSize;i++)
            uart_printf("%c",*((unsigned char*)(downloadAddress+i)));

    if(checksum!=dnCS)
    {
        uart_printf("Checksum Error!!! MEM:%x DN:%x\n",checksum,dnCS);
        return;
    }

    uart_printf("\n Download O.K.\n\n");
    uart_txempty(consoleNum);

    if(download_run==1)
    {
        rINTMSK = BIT_ALLMSK;
        run=(void (*)(void))downloadAddress;
        run();
    }
}

/*****
* name:      Isr_Init
* func:      Initialization of Isr
* para:      none
* ret: none
* modify:
* comment:
```

```
*****/

void Isr_Init(void)
{
    pISR_UNDEF=(unsigned)debug_undef;
    pISR_SWI  =(unsigned)debug_swi;
    pISR_PABORT=(unsigned)debug_abort_d;
    pISR_DABORT=(unsigned)debug_abort_d;
    rINTMOD=0x0;                                // All=IRQ mode
    rSRCPND   = rSRCPND;                        // clear all interrupt
    rINTPND   = rINTPND;                        // clear all interrupt
    rINTMSK   = BIT_ALLMSK;                     // All interrupt is masked.
    rINTSUBMSK = BIT_SUB_ALLMSK;                // All sub-interrupt is masked.

    pISR_USBD =(unsigned)IsrUsbd;
    pISR_DMA2 =(unsigned)IsrDma2;

    ClearPending(BIT_DMA2);
    ClearPending(BIT_USBD);
}
```

7. 4. 7 练习题

编写程序，通过端点 1 发送自定义的信息到 PC 机上 USB 主机，并使用 DNW 观察。