第一章　分立模块实验

实验一　LCD 上的字符显示

【实验目的】

1、 了解 SPLC501 的使用方法及相关函数

2、 学习利用 SPLC501 显示字符

【实验设备】

1)装有 μ'nSP™ IDE 仿真环境的 PC 机一台。

2)μ'nSP™十六位单片机实验箱一个。

【实验原理】

利用 SPLC501 显示字符，详细资料见 SPLC501 数据手册

【实验步骤】

1)根据硬件连接图连接好硬件。（实验箱默认的 LCD 连接方式）

2)将 μ'nSP™ IDE 打开后，建立一个新工程。

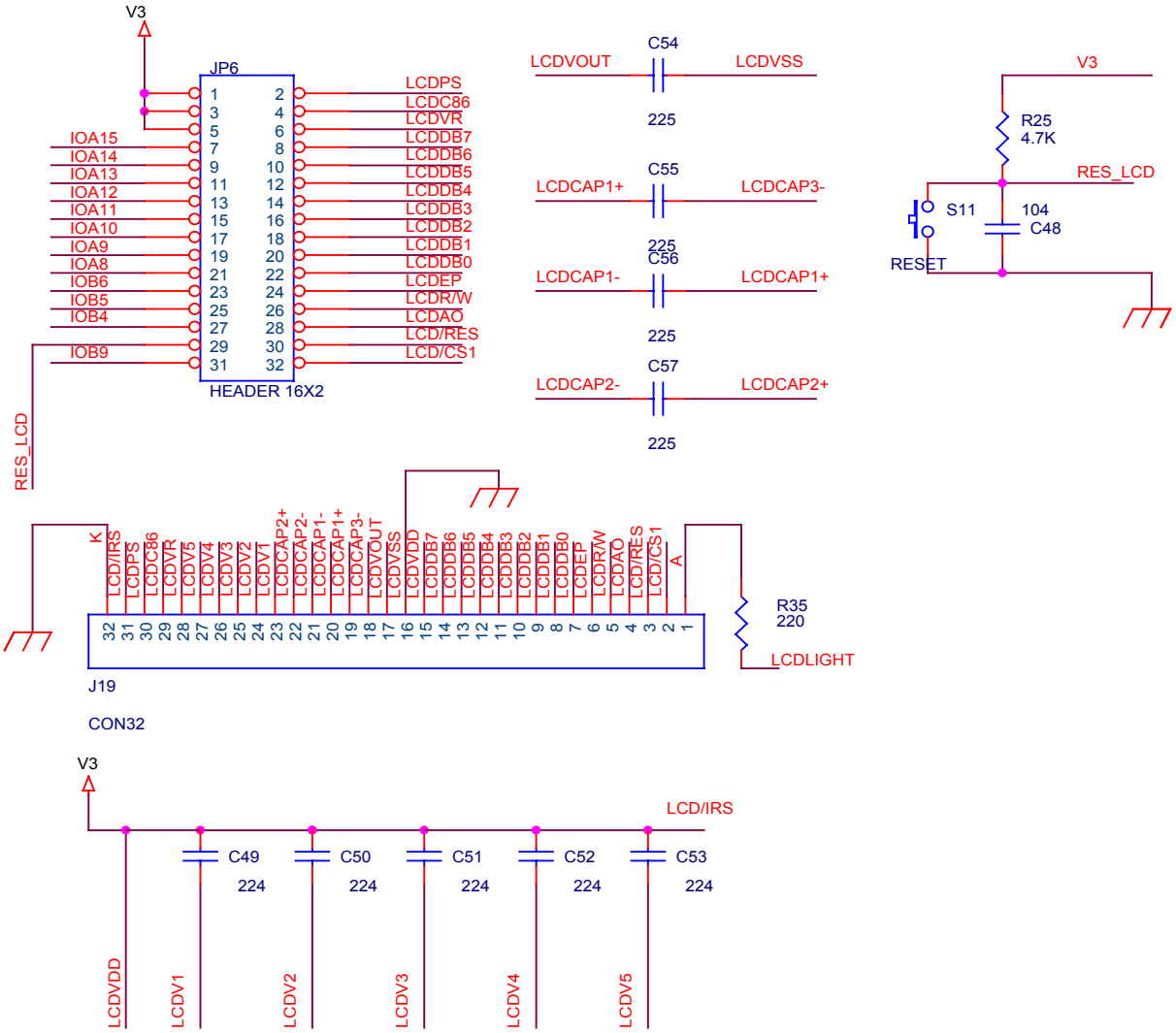
3)在该项目的源文件夹(SOURCE FILES)下建立一个新的 C 语言文件。

4)编写程序代码。

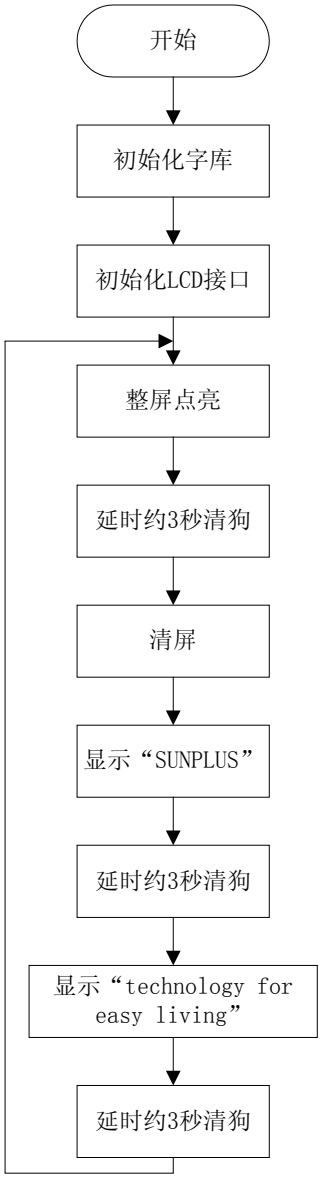
5)编译程序，软件调试。

6)注意观察 LCD 的现象

【硬件连接图】


V3
JP6
HEADER 16X2
IOA15
IOA14
IOA13
IOA12
IOA11
IOA10
IOA9
IOA8
IOB6
IOB5
IOB4
IOB9
LCDPS
LCDC86
LCDVR
LCDDB7
LCDDB6
LCDDB5
LCDDB4
LCDDB3
LCDDB2
LCDDB1
LCDDB0
LCDEP
LCDR/W
LCDAO
LCD/RES
LCD/CS1
RES_LCD
LCDVOUT
C54
LCDVSS
225
LCDCAP1+
C55
LCDCAP3-
225
LCDCAP1-
C56
LCDCAP1+
225
LCDCAP2-
C57
LCDCAP2+
225
R25
4.7K
RES_LCD
S11
104
C48
RESET
K
LCD/IRS
LCDPS
LCDC86
LCDVR
LCDV5
LCDV4
LCDV3
LCDV2
LCDV1
LCDCAP2+
LCDCAP2-
LCDCAP1-
LCDCAP1+
LCDCAP3-
LCDVOUT
LCDVSS
LCDVDD
LCDDB7
LCDDB6
LCDDB5
LCDDB4
LCDDB3
LCDDB2
LCDDB1
LCDDB0
LCDEP
LCDR/W
LCDAO
LCD/RES
LCD/CS1
A
R35
220
LCDLIGHT
J19
CON32
V3
LCD/IRS
C49
C50
C51
C52
C53
224
224
224
224
224
LCDVDD
LCDV1
LCDV2
LCDV3
LCDV4
LCDV5

【程序流程图】



【程序及其特殊函数说明】

FG_ClearScreen

调用方式：FG_ClearScreen ();

功能说明：清除或填充全屏。

参数： DG_CLS_ERASE(default), DG_CLS_FILL

示　　例

FG_ClearScreen (DG_CLS_FILL)

Function Name: FG_PutStr

参数: StrPtr, Font, StartX, StartY

功能说明: 输出字符串从 (StartX, StartY).

Mode:

DG_CHAR_COVER(default)

DG_CHAR_INVERSE

DG_CHAR_XOR

Memory Modified: None
用　法:FG_PutStr(char *StrPtr,short Font,short StartX,short StartY)
示　　例:
FG_PutStr(StringPointer,Tiny,5,6)
　　FG_PutStr("Hello",Large)

实验二　LCD 上的汉字显示

【实验目的】

1、 了解 SPLC501 的使用方法及相关函数

2、学习利用 SPLC501 显示汉字

【实验设备】

1)装有 μ'nSP™ IDE 仿真环境的 PC 机一台。

2)μ'nSP™十六位单片机实验箱一个。

【实验原理】

利用 SPLC501 显示汉字

【实验步骤】

1)根据硬件连接图连接好硬件。(实验箱默认的 LCD 连接方式)

2)将μ'nSP™ IDE 打开后，建立一个新工程。

3)在该项目的源文件夹(SOURCE FILES)下建立一个新的 C 语言文件。

4)编写程序代码。

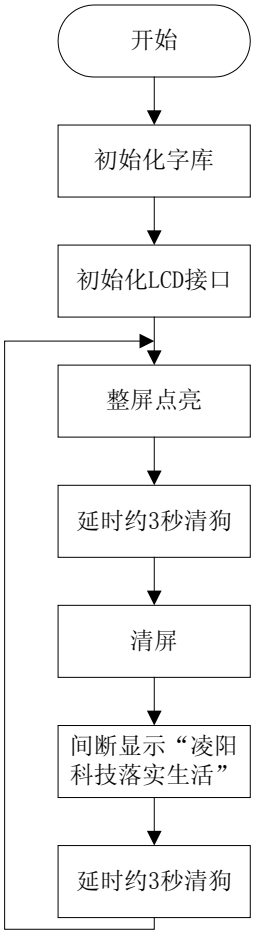
5)编译程序，软件调试。

6)注意观察 LCD 的现象

【硬件连接图】

同实验一硬件连接。

【程序流程图】



【程序及其特殊函数说明】

FG_ClearScreen

调用方式：FG_ClearScreen ()；

功能说明：清除或填充全屏。

参数： DG_CLS_ERASE(default), DG_CLS_FILL

示　　例

FG_ClearScreen (DG_CLS_FILL)

FG_PutBitmap

调用方式： FG_PutBitmap ()；

功能说明：画 bmp 图。

参数：BmpX,BmpY,BmpNum,Mode

功能说明：画位图从 (BmpX,BmpY) 点，用指定模式

Mode:

DG_BMP_COVER(default)

DG_BMP_INVERSE

DG_BMP_XOR

Memory Modified: None

用　法：FG_PutBitmap(short BmpNum,short BmpX,short BmpY,short Mode)

示　　例：

```
FG_PutBitmap(0,10,20,DG_BMP_COVER)
FG_PutBitmap(1,10,20)
FG_PutBitmap(2,DG_BMP_XOR)
```

实验三 LCD 上的图片显示

【实验目的】

1、了解 SPLC501 的使用方法及相关函数

2、学习利用 SPLC501 显示图片

【实验设备】

1)装有 μ'nSP™ IDE 仿真环境的 PC 机一台。

2)μ'nSP™十六位单片机实验箱一个。

【实验原理】

利用 SPLC501 显示图片，图片的性质为位图。采用位图转换工具，将位图图片生成二进制码。

【实验步骤】

1)根据硬件连接图连接好硬件。（实验箱默认的 LCD 连接方式）

2)将 μ 'nSP™ IDE 打开后，建立一个新工程。

3)在该项目的源文件夹(SOURCE FILES)下建立一个新的 C 语言文件。

4)编写程序代码。

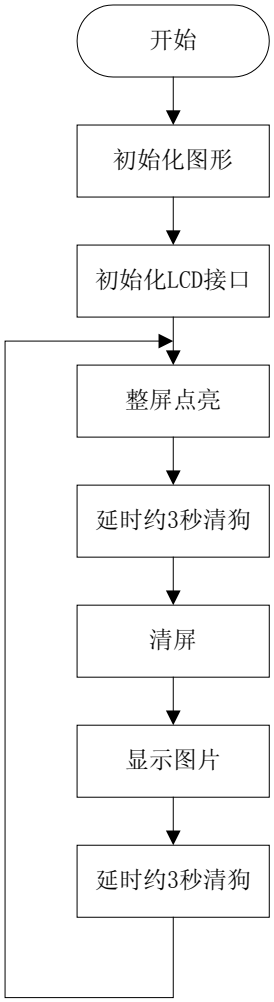
5)编译程序，软件调试。

6)注意观察 LCD 的现象

【硬件连接图】

同实验一硬件连接。

【程序流程图】



【程序及其特殊函数说明】

FG_ClearScreen

调用方式：FG_ClearScreen ();

功能说明：清除或填充全屏。

参数： DG_CLS_ERASE(default), DG_CLS_FILL

示　　例

FG_ClearScreen (DG_CLS_FILL)

FG_PutBitmap

调用方式： FG_PutBitmap ();

功能说明：画 bmp 图。

参数：BmpX,BmpY,BmpNum,Mode

功能说明：画位图从 (BmpX,BmpY) 点，用指定模式

Mode:

DG_BMP_COVER(default)

DG_BMP_INVERSE

DG_BMP_XOR

Memory Modified: None

用　法：FG_PutBitmap(short BmpNum,short BmpX,short BmpY,short Mode)

示　　例：

```
FG_PutBitmap(0,10,20,DG_BMP_COVER)
FG_PutBitmap(1,10,20)
FG_PutBitmap(2,DG_BMP_XOR)
FG_PutBitmap(3)
```

实验四 LCD 上的动态图片显示

【实验目的】

1、 解 SPLC501 的使用方法及相关函数

2、学习利用 SPLC501 显示动态图片

【实验设备】

1)装有 μ'nSP™ IDE 仿真环境的 PC 机一台。

2)μ'nSP™十六位单片机实验箱一个。

【实验原理】

利用 SPLC501 显示动态图片，原理与显示图片相同，动态图片需要多帧图片。

【实验步骤】

1)根据硬件连接图连接好硬件。（实验箱默认的 LCD 连接方式）

2)将 μ 'nSP™ IDE 打开后，建立一个新工程。

3)在该项目的源文件夹(SOURCE FILES)下建立一个新的 C 语言文件。

4)编写程序代码。

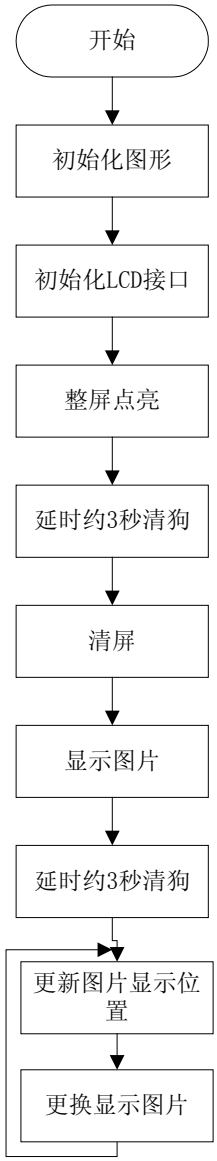
5)编译程序，软件调试。

6)注意观察 LCD 的现象

【硬件连接图】

同实验一硬件连接。

【程序流程图】



【程序及其特殊函数说明】

FG_ClearScreen

调用方式：FG_ClearScreen ();

功能说明：清除或填充全屏。

参数： DG_CLS_ERASE(default), DG_CLS_FILL

示　　例

FG_ClearScreen (DG_CLS_FILL)

FG_PutBitmap

调用方式： FG_PutBitmap ();

功能说明：画 bmp 图。

参数： BmpX,BmpY,BmpNum,Mode

功能说明：画位图从 (BmpX,BmpY) 点，用指定模式

Mode:

DG_BMP_COVER(default)

```
    DG_BMP_INVERSE
    DG_BMP_XOR
Memory Modified: None
用　法: FG_PutBitmap(short BmpNum,short BmpX,short BmpY,short Mode)
示　　例:
    FG_PutBitmap(0,10,20,DG_BMP_COVER)
    FG_PutBitmap(1,10,20)
    FG_PutBitmap(2,DG_BMP_XOR)
    FG_PutBitmap(3)
```

实验五 LCD 上的几何图形显示

实验目的】

1、了解 SPLC501 的使用方法及相关函数

2、学习利用 SPLC501 显示几何图形

【实验设备】

1)装有 μ'nSP™ IDE 仿真环境的 PC 机一台。

2)μ'nSP™十六位单片机实验箱一个。

【实验原理】

利用 SPLC501 显示椭圆，正方形。

【实验步骤】

1)根据硬件连接图连接好硬件。（实验箱默认的 LCD 连接方式）

2)将 μ 'nSP™ IDE 打开后，建立一个新工程。

3)在该项目的源文件夹(SOURCE FILES)下建立一个新的 C 语言文件。

4)编写程序代码。

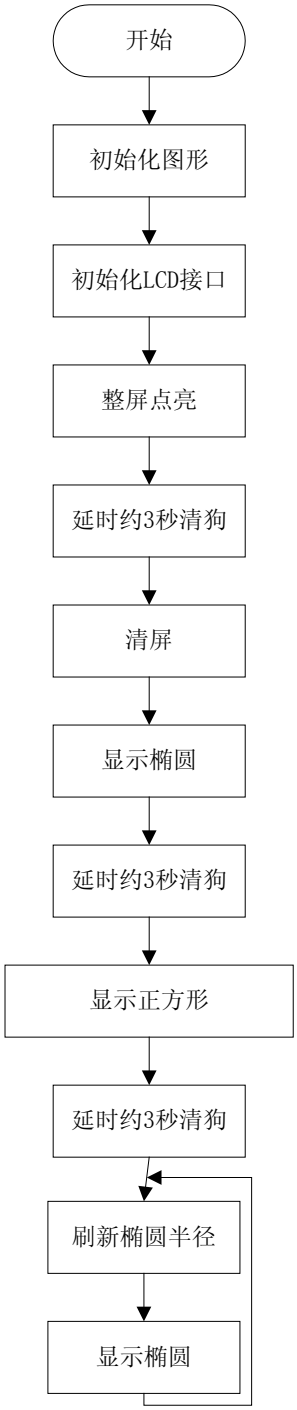
5)编译程序，软件调试。

6)注意观察 LCD 的现象

【硬件连接图】

同实验一硬件连接。

【程序流程图】



【程序及其特殊函数说明】

;===

FG_ClearScreen

调用方式：FG_ClearScreen ();

功能说明：清除或填充全屏。

参数： DG_CLS_ERASE(default), DG_CLS_FILL

示　　例

FG_ClearScreen (DG_CLS_FILL)

;===

FG_InitGraphic

调用方式： FG_InitGraphic ()；

功能说明：初始化端口和 splc501c。

参数：

示　　例：

FG_InitGraphic()

;===

FG_SetLineStyle

参数: Mode

功能说明: 设定现模式

Mode:

DG_LINE_COVER　　//(default)

DG_LINE_ERASE

DG_LINE_HOLLOW

DG_LINE_DOTTED

DG_LINE_HOLLOW_ERASE

DG_LINE_DOTTED_COVER

DG_LINE_SOLID_XOR

DG_LINE_DOTTED_XOR

Memory Modified: R_GraphicMode

用　法:

FG_SetLineStyle(short Mode)

示　　例:

FG_SetLineStyle(DG_LINE_DOTTED_COVER)

;===

Function Name: FG_Ellipse

参数: OriginX, OriginY, RadiusX, RadiusY

功能说明: 用设定模式（R_GraphicMode.）画椭圆

Memory Modified: None

用　法:

FG_Ellipse(short OriginX,short OriginY,short RadiusX,short RadiusY)

示　　例:

FG_Ellipse(20,20,5,4)

FG_Ellipse(R_Var1,R_Var2,R_Var3,R_Var4)

;===

Function Name: FG_ClearEllipse

参数: OriginX, OriginY, RadiusX, RadiusY,Mode

功能说明: 填充或清除椭圆区域

Mode: DG_CLEAR_ERASE(default),DG_CLEAR_FILL

Memory Modified: None

用　法:
FG_ClearEllipse(short OriginX,short OriginY,short RadiusX,short RadiusY,short Mode)
示　　例:
FG_ClearEllipse(20,20,5,4,DG_CLEAR_ERASE)
FG_ClearEllipse(R_Var1,R_Var2,R_Var1,R_Var2,R_Var3,R_Var4)

;==
Function Name: FG_Rectangle
参数: StartX,StartY,EndX,EndY
功能说明: 用设定模式（R_GraphicMode.）画矩形
　　StartX,StartY 是右下脚点 EndX,EndY 是左上脚点
Memory Modified: None
用　法:
FG_Rectangle(short StartX,short StartY,short EndX,short EndY)
示　　例:
FG_Rectangle(0,0,20,20)
FG_Rectangle(20,20)
FG_Rectangle(R_Var1,R_Var2,R_Var3,R_Var4)
FG_Rectangle(R_Var1,R_Var2)

实验六 USB 通讯实验

【实验目的】

1) 通过实验了解实验箱 USB 模块的硬件连接。
2) 掌握简单的 USB 通讯：实现红灯亮、灭和路灯亮灭。
3) 掌握编程中常用函数的使用。

【实验设备】

1) 装有 WINDOWS 系统和 μ'nSP™ IDE 仿真环境的 PC 机一台。
2) μ'nSP™ 十六位单片机实验箱一个。

【实验原理】

用 USB 模组和 SPCE061A 最小系统实现 USB 简单通讯，并实现两点功能：

1. 通过 USB 通讯，PC 端应用程序能够控制 LED 灯的亮灭；
2. PC 端应用程序发送小于 65 字节的字符串给 SPCE061A,SPCE061A 接收 PC 发送的小于 65 字节字符串后将接收到的字符串发送给 PC，PC 接收字符并显示在界面。

【实验步骤】

1. 接好硬件，包括与 MCU 的接线，电源跳线，USB 线。
2. 在 IOA0，IOA1 口接 LED 灯。
3. 将提供的 example 1 的 firmware 下载到单片机（SPCE061A）
4. 按照提示安装驱动程序（如果未装驱动）
5. 等 USB 通讯指示灯亮后，运行 P C 端应用软件，如图 8 所示：

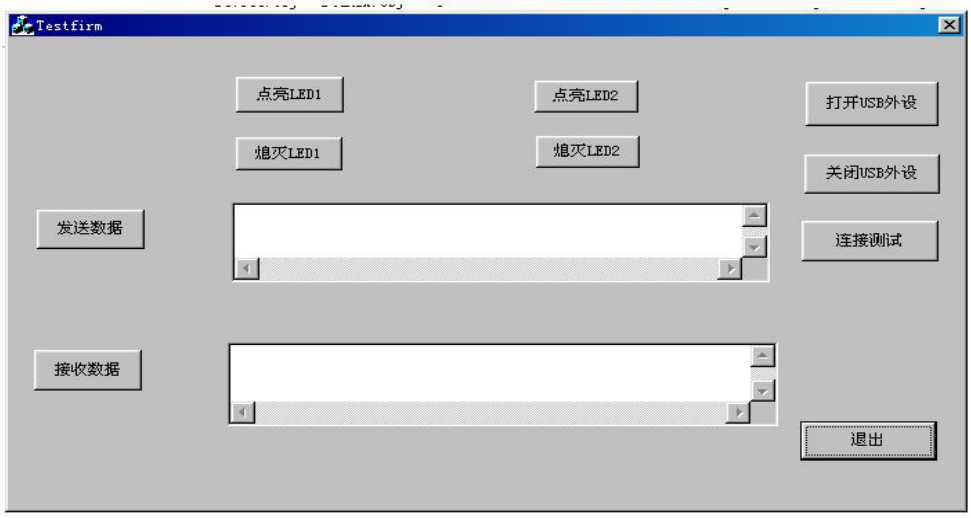


图1 PC 端软件界面图

6. 接着点击“打开 USB 外设”按钮，如果出现如下提示框，执行第 7 步操作，否则点击“确定”然后执行第 3 步操作，重新开始。

7. 点击“连接测试”按钮，如果出现如下提示框，执行第 8 步操作，否则点击“确定”然后执行第 3 步操作，重新开始。

8. 开始 USB 通讯测试

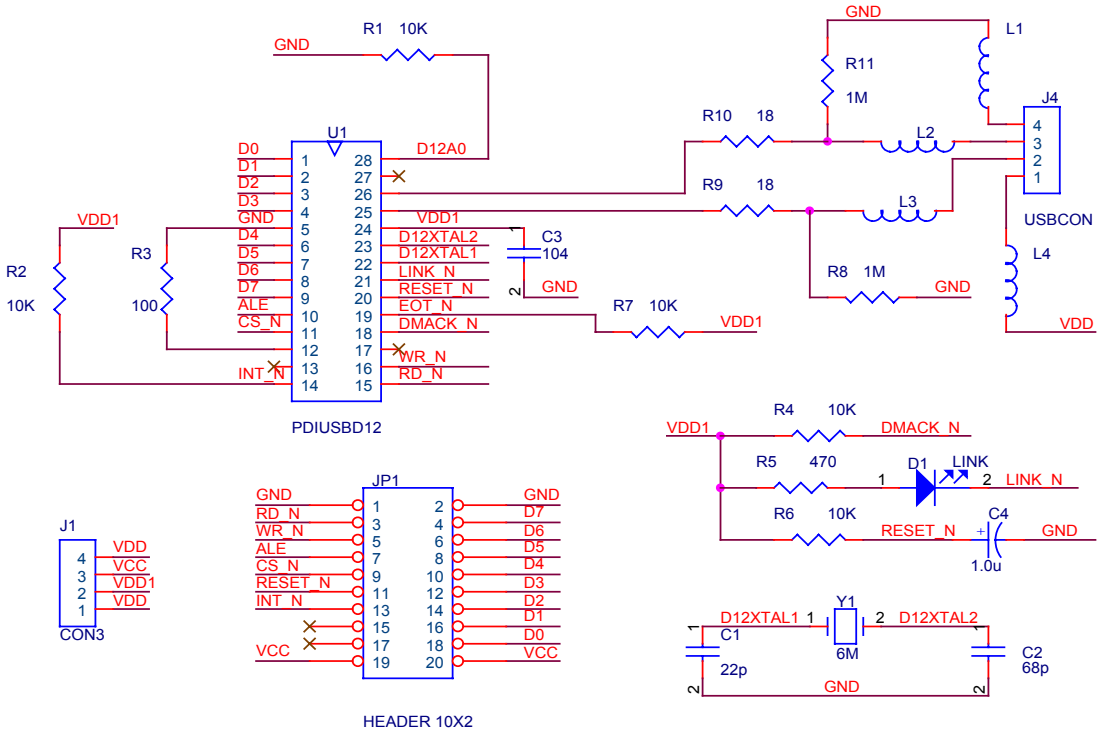
1）通过点击“点亮 LED1”，“熄灭 LED1”，“点亮 LED2”，“熄灭 LED”来测试

2）通过发送、接收数据（数据量小于 65byte）来测试 USB 通讯

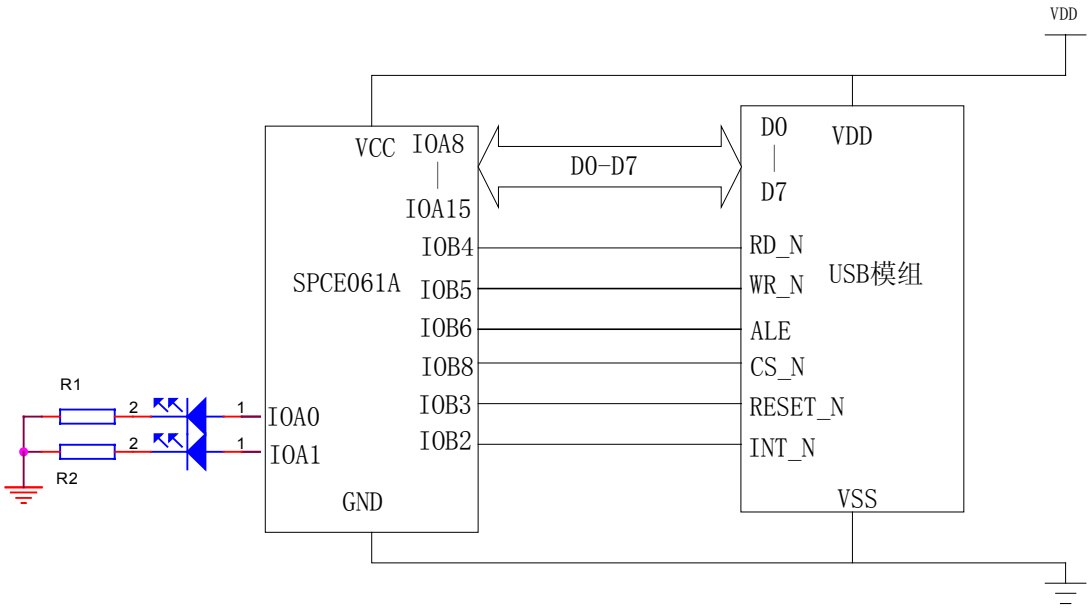
测试完毕后点击“关闭 USB 外设”按钮

【硬件连接图】

1、 模组原理图



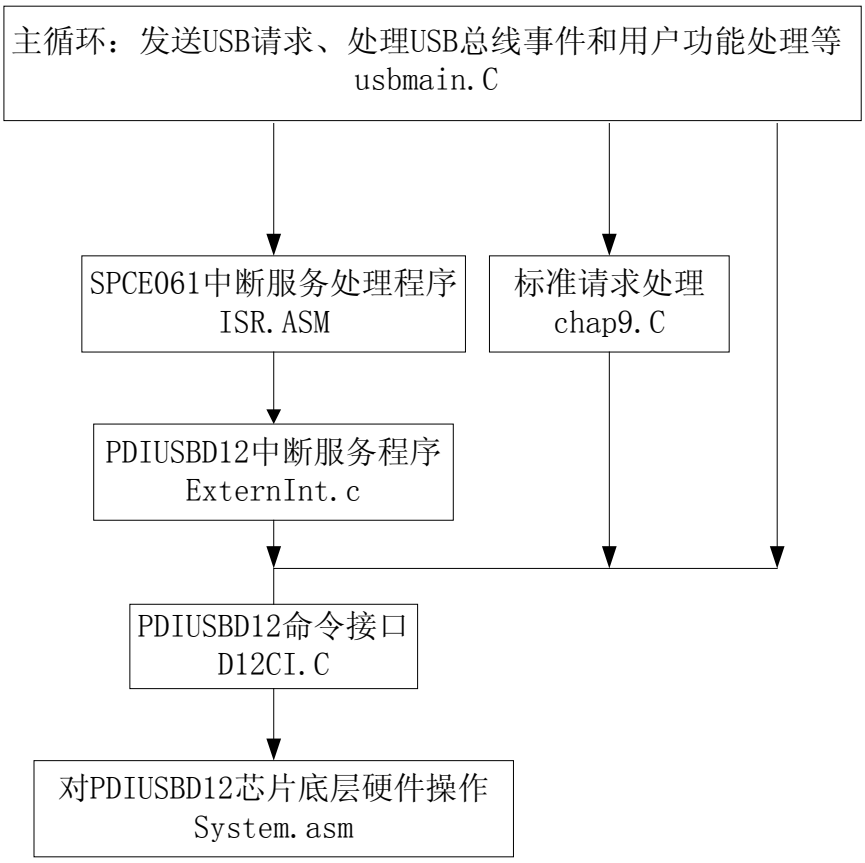
2、 和 SPCE061A 连接图



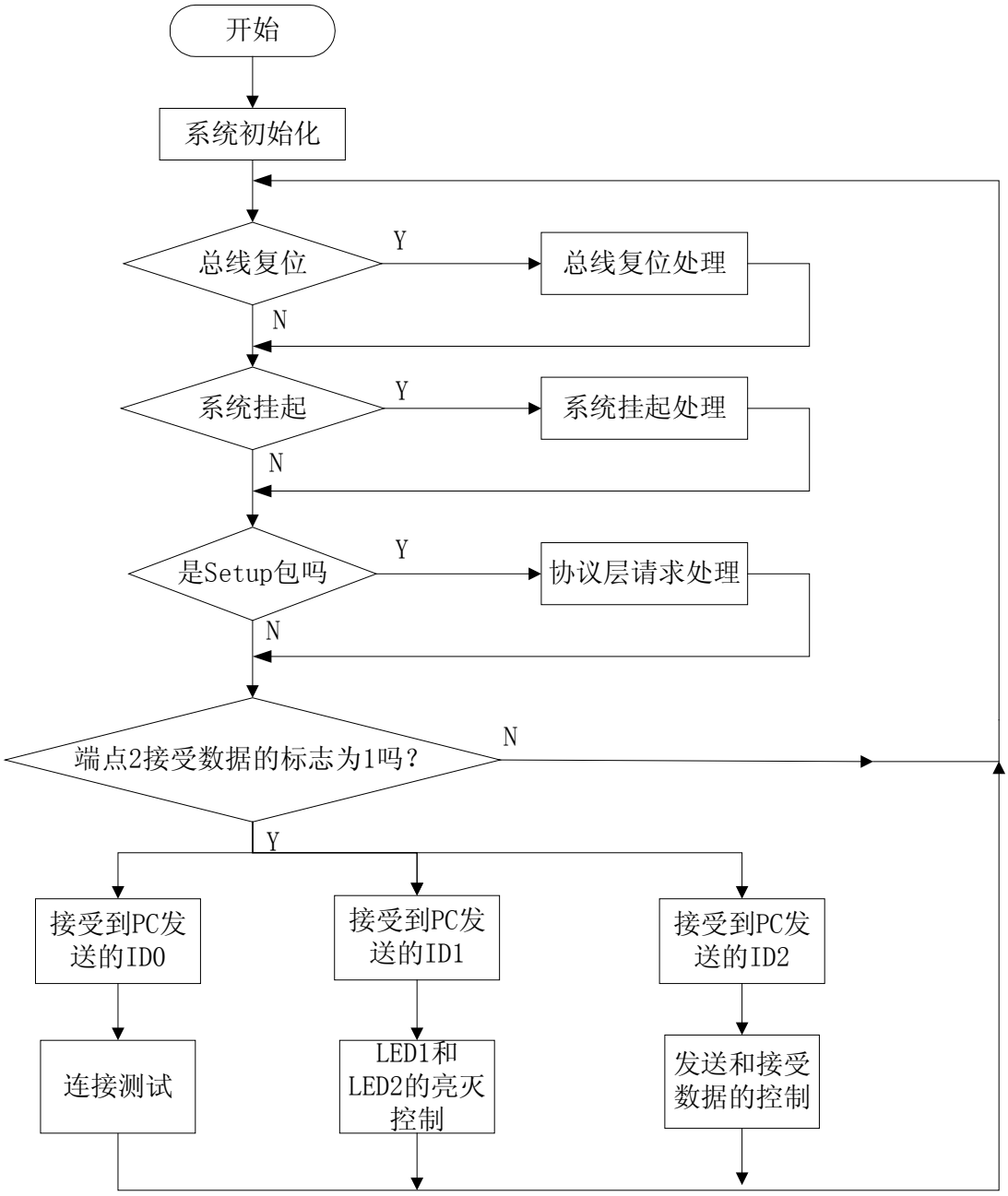
【程序流程图】

固件设计介绍：

1、 固件结构和数据流向


主循环：发送USB请求、处理USB总线事件和用户功能处理等
usbmain.C
SPCE061中断服务处理程序
ISR.ASM
标准请求处理
chap9.C
PDIUSBD12中断服务程序
ExternInt.c
PDIUSBD12命令接口
D12CI.C
对PDIUSBD12芯片底层硬件操作
System.asm

程序流程：



【程序及其特殊函数说明】

程序清单：

chap9.c：协议处理

ExternInt.c：处理 D12 的中断

D12CI.c：处理 D12 芯片的控制命令

System.asm：系统初始化

2、常用函数介绍：

2.1Void F_USB_Isr（void）

该函数为处理分析 D12 芯片的中断源，主要是设置相应 D12 的中断源标志，用户只要知道有以

下四种中断源标志位即可。

Ep1_ReceiveDataFlag：该标志位为 1，表示 PC 主机向 MCU 发送数据，即 pc 数据已经发送到端点 1 的 buffer，等待 MCU 读取。

Ep1_SendDataFlag： 该标志位为 1，表示 PC 主机请求 MCU 发送数据到 D12 端点 1 的 Buffer。

Ep2_ReceiveDataFlag：该标志位为 1，表示 PC 主机向 MCU 发送数据，即 pc 数据已经发送到端点 2 的 buffer，等待 MCU 读取。

Ep2_SendDataFlag： 该标志位为 1，表示 PC 主机请求 MCU 发送数据到 D12 端点 1 的 Buffer。

2.2 用户对 D12 的操作主要有以下四个函数：

1） unsigned int F_D12_ReadLastTransactionStatus(unsigned int bEndp);

【参　数】端点号，取值范围为 0－5

【返回值】参见 PDIUSBD12 用户手册，来源：www.unsp.com.cn

【功　能】清 D12 的端点中断源

2） unsigned int F_D12_ReadEndpoint(unsigned int endp, unsigned int len, unsigned int * buf);

【参　数】1.端点号 2.数据长度 3. 数据缓冲区的地址

【返回值】读取到的数据实际长度。

【功　能】读 D12 中 Buffer 的数据，该函数要与 F_D12_ReadLastTransactionStatus（）函数配合使用，建议使用 F_D12_ReadEndpointAndClrD12Int（）函数。

3） unsigned int F_D12_WriteEndpoint(unsigned int endp, unsigned int len, unsigned int * buf);

【参　数】1.端点号 2.数据长度 3. 数据缓冲区的地址

【返回值】写入缓冲区的实际数据长度。

【功　能】写数据到 D12 的 Buffer

4） unsigned int F_D12_ReadEndpointAndClrD12Int(unsigned int endp, unsigned int len, unsigned int * buf);

【参　数】1.端点号 2.数据长度 3. 数据缓冲区的地址

【返回值】读取到的数据实际长度。

【功　能】读 D12 中 Buffer 的数据，该函数比 F_D12_ReadEndpoint（）多一个清中断的操作。

3、注意事项

当 pc 端程序执行 Readfile（）的时候，MCU 只有使能 D12 的端点 Buffer 的时候，才会产生中断。当标志位 Ep1_SendDataFlag 或 Ep2_SendDataFlag 为 1 时， EasyUSB11.lib 中已经清中断了，用户不需再清中断。

```
main()
{
    unsigned int uiSendDataFlag=0;
    unsigned int uiReadEp2DataLength=0;
    unsigned int aIdFlag[2];
    F_System_Initial();
    F_Reconnect_USB();                          //PDIUSBD12 芯片的软连接
    F_Interrupt_On();
    while(1)
    {
    if (bEPPflags.bits.bus_reset)               //总线复位处理
    {
```

```
        bEPPflags.bits.bus_reset = 0;                   //清标志
    }
    if (bEPPflags.bits.suspend)                         //总线挂起处理
    {
        bEPPflags.bits.suspend= 0;                      //清标志
    }
    if (bEPPflags.bits.setup_packet)                    //协议处理
    {
        bEPPflags.bits.setup_packet = 0;                //清标志
        F_Control_Handler();
    }
    if(bEPPflags.bits.Ep1_ReceiveDataFlag==1)
    {
        bEPPflags.bits.Ep1_ReceiveDataFlag=0;
        F_D12_ReadEndpointAndClrD12Int(2, 2,aIdFlag);
        if(aIdFlag[0]==ID0)                             //连接测试
        {
            F_D12_WriteEndpoint(5,1,aIdFlag);
        }
        else if(aIdFlag[0]==ID1)
        {
                if(aIdFlag[1]==1)
                    F_TurnOnFirstLed();                 //点亮 LED 灯
                if(aIdFlag[1]==0)
                    F_TurnOffFirstLed();                //熄灭 LED 灯
                if(aIdFlag[1]==3)
                    F_TurnOnSecondLed();                //点亮 LED 灯
                if(aIdFlag[1]==2)
                    F_TurnOffSecondLed();               //熄灭 LED 灯
        }
        else if(aIdFlag[0]==ID2)
        {
                if(aIdFlag[1]==1)                       //应答
                {
                    F_D12_WriteEndpoint(5,1,aIdFlag);
                }
                else if(aIdFlag[1]==2)                  //回送数据
                {
                    F_D12_WriteEndpoint(5,uiReadEp2DataLength,MainEpBuf);
                }
        }
    }
        if(bEPPflags.bits.Ep1_SendDataFlag==1)
        {
```

```
            bEPPflags.bits.Ep1_SendDataFlag=0;
        }
        if(bEPPflags.bits.Ep2_ReceiveDataFlag==1)
        {
            bEPPflags.bits.Ep2_ReceiveDataFlag = 0;   //清标志
            uiReadEp2DataLength=F_D12_ReadEndpointAndClrD12Int(4, 64,MainEpBuf);
        }
        if(bEPPflags.bits.Ep2_SendDataFlag==1)
        {
            bEPPflags.bits.Ep2_SendDataFlag=0;
        }
        F_Clear_WatchDog();                          //清除 WatchDog
    }
}
```

实验七 SPR4096 中的 FLASH 的擦除及其读写

【实验目的】

1)通过实验了解 SIO 的基本使用方法。

2)了解和体验通过 SIO 扩展 FLASH。

3)学习使用 SPR4096 的 FLASH 的读写和擦除。

【实验设备】

1)装有 WINDOWS 系统和 μ'nSP™ IDE 仿真环境的 PC 机一台。

2) μ'nSP™ 十六位单片机实验箱一个。

【实验原理】

SPR4096 是一个高性能的 4M-bit（512K×8-bit）FLASH，分为 256 个扇区（Sector）每个扇区为 2K-byte。SPR4096 还内置了一个 4K×8-bit 的 SRAM。

SPR4096 串行接口的工作频率可达 5MHz。SPR4096 有两个电源输入端 VDDI 和 VDDQ。VDDI 是给内部 FLASH 和控制逻辑供电的；VDDQ 是专门为 I/O 供电的。供电电压为 VDDQ: 2.25V ~ 3.6V, VDDI: 2.25V ~ 2.75V。

SPR4096 按串行接口模式工作，要把 CF2～CF0 均接高电平。CF7 为低电平时选中 FLASH，高电平时选中 SRAM。本实验中 CF7 与 SPCE061A 的 IOB11 相连，IOB11 输出低电平选择 SPR4096 的 FLASH。

实验现象为，FLASH 擦除后，与 IOA0 连接的 LED 被点亮；写入一个 WORD 后，与 IOA0～IOA3 连接的 LED 被点亮；读出的数据若与写入的数据相等，则点亮与 IOA0～IOA7 连接的 LED，否则熄灭与 IOA0～IOA7 连接的 LED。

【实验步骤】

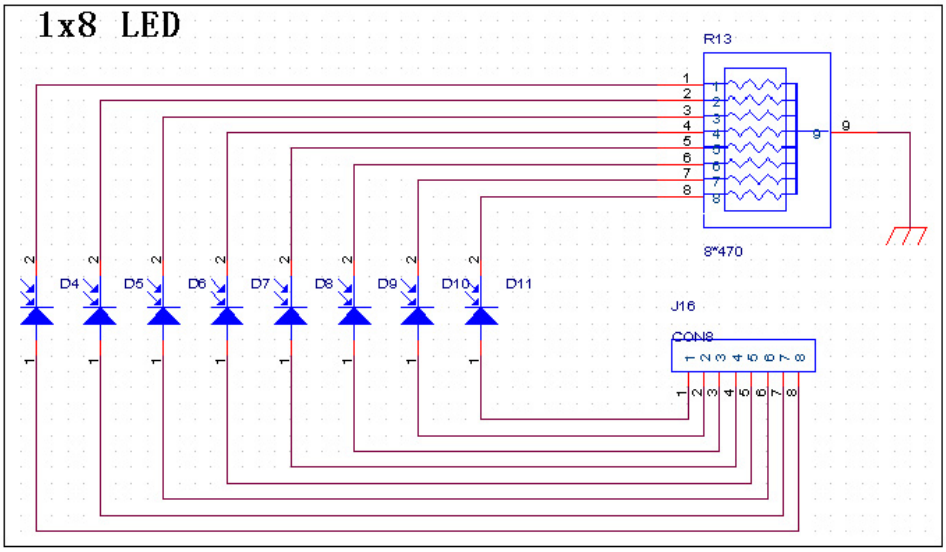
1）把 JP10 的三个短路线接好，注意 CF7 选择 B11。

2）用排线把 J26 和 J16 接上。

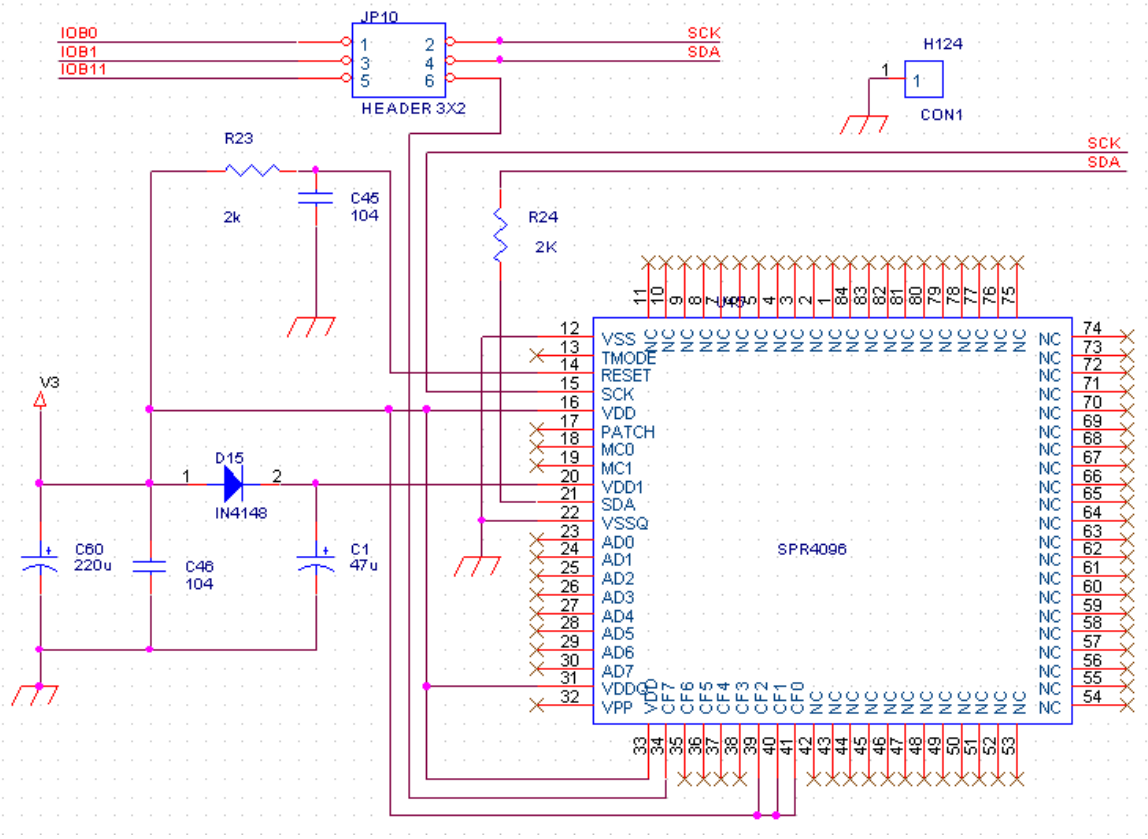
3）下载实验程序，观察现象。

【硬件连接图】

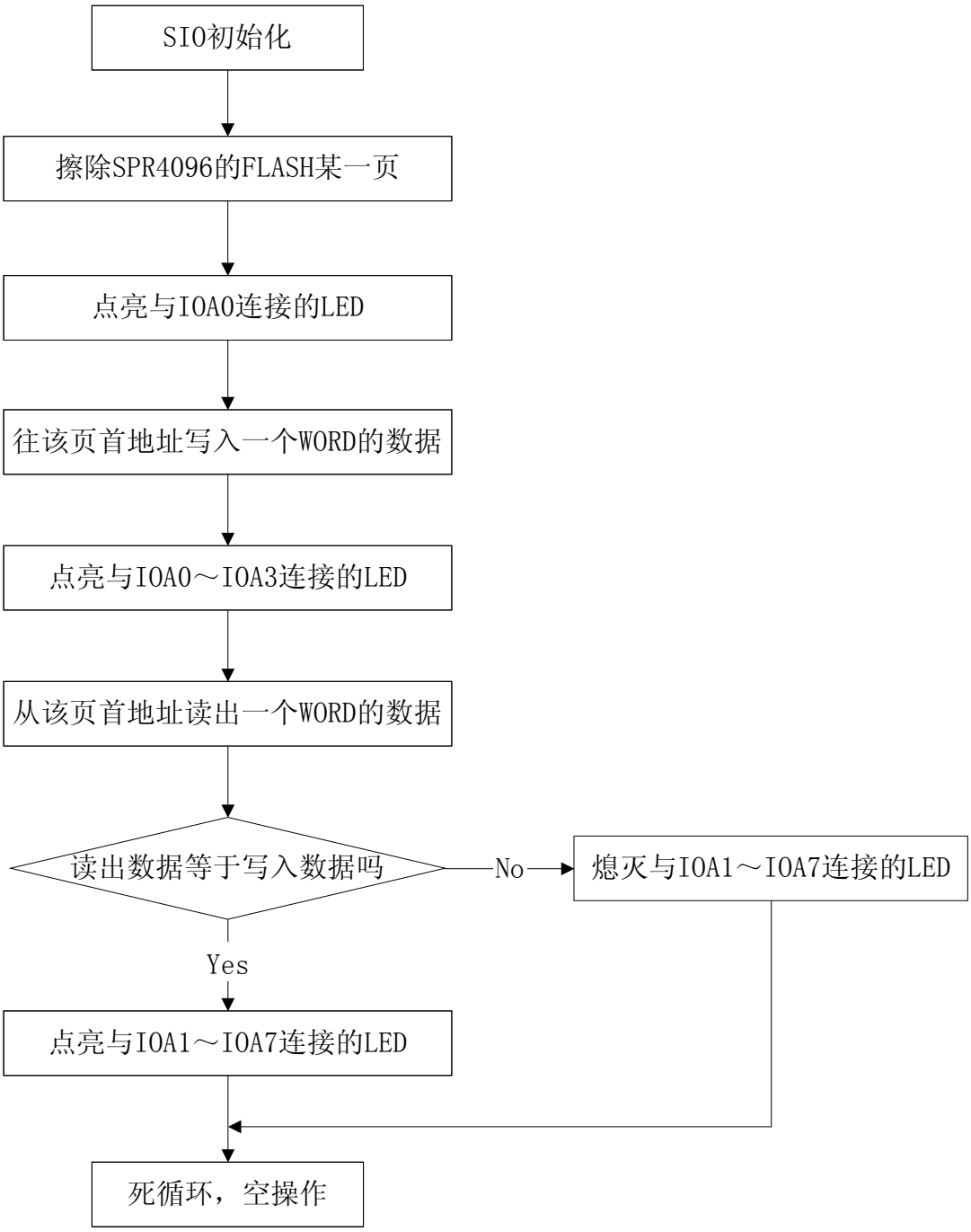
实验箱上发光二极管的原理图如下：



实验箱上 SPR4096 接口原理图如下：



【程序流程图】



【程序及其特殊函数说明】

本程序主要结合实验箱资源来完成一个 SPR4096 的 FLASH 操作，都采用实验箱上分配给各模块的 IO 资源，发光二极管由 IOA 的低八位控制。

本 SPR4096 模块中有 7 个接口函数，如下：

1. void SP_SIOInitial(void)
   功　能：初始化 SIO
   参　数：无
   返回值：无
2. void SP_SIOMassErase(void)
   功　能：擦除 SPR4096 的所有扇区

参　数：无
返回值：无

3. void　SP_SIOSectorErase(unsigned int uiSector)
功　能：擦除 SPR4096 的一个扇区
参　数：uiSector，为扇区的编号，0～255 可选
返回值：无
4. void　SP_SIOSendAByte(unsigned long int ulAddr,unsigned int uiData)
功　能：往 SPR4096 写入一个 Byte 的数据
参　数：ulAddr 为写入的地址，uiData 为写入的数据
返回值：无
5. void　SP_SIOSendAWord(unsigned long int ulAddr,unsigned int uiData)
功　能：往 SPR4096 写入一个 Word 的数据
参　数：ulAddr 为写入的地址，uiData 为写入的数据
返回值：无
6. unsigned int　SP_SIOReadAByte(unsigned long int ulAddr)
功　能：从 SPR4096 读出一个 Byte 的数据
参　数：ulAddr 为读出的地址
返回值：读出的数据
7. unsigned int　SP_SIOReadAWord(unsigned long int ulAddr)
功　能：从 SPR4096 读出一个 Byte 的数据
参　数：ulAddr 为读出的地址
返回值：读出的数据

实验八 SPR4096 中的 SRAM 的读写

【实验目的】

1)通过实验了解 SIO 的基本使用方法。
2)了解和体验通过 SIO 扩展 SRAM。
3)学习使用 SPR4096 的 SRAM 的读写。

【实验设备】

1)装有 WINDOWS 系统和 μ'nSP™ IDE 仿真环境的 PC 机一台。
2) μ'nSP™ 十六位单片机实验箱一个。

【实验原理】

SPR4096 是一个高性能的 4M-bit（512K×8-bit）FLASH，分为 256 个扇区（Sector）每个扇区为 2K-byte。SPR4096 还内置了一个 4K×8-bit 的 SRAM。

SPR4096 串行接口的工作频率可达 5MHz。SPR4096 有两个电源输入端 VDDI 和 VDDQ。VDDI 是给内部 FLASH 和控制逻辑供电的；VDDQ 是专门为 I/O 供电的。供电电压为 VDDQ: 2.25V ~ 3.6V, VDDI: 2.25V ~ 2.75V。

SPR4096 按串行接口模式工作，要把 CF2～CF0 均接高电平。CF7 为低电平时选中 FLASH，高电平时选中 SRAM。本实验中 CF7 与 SPCE061A 的 IOB11 相连，IOB11 输出高电平选择 SPR4096 的 SRAM。

实验现象为，往 SRAM 首地址写入一个 WORD 后，与 IOA0～IOA3 连接的 LED 被点亮；然后

从 SRAM 首地址读出一个 WORD，若读出的数据与写入的数据相等，则点亮与 IOA0～IOA7 连接的 LED，否则熄灭与 IOA0～IOA7 连接的 LED。

【实验步骤】

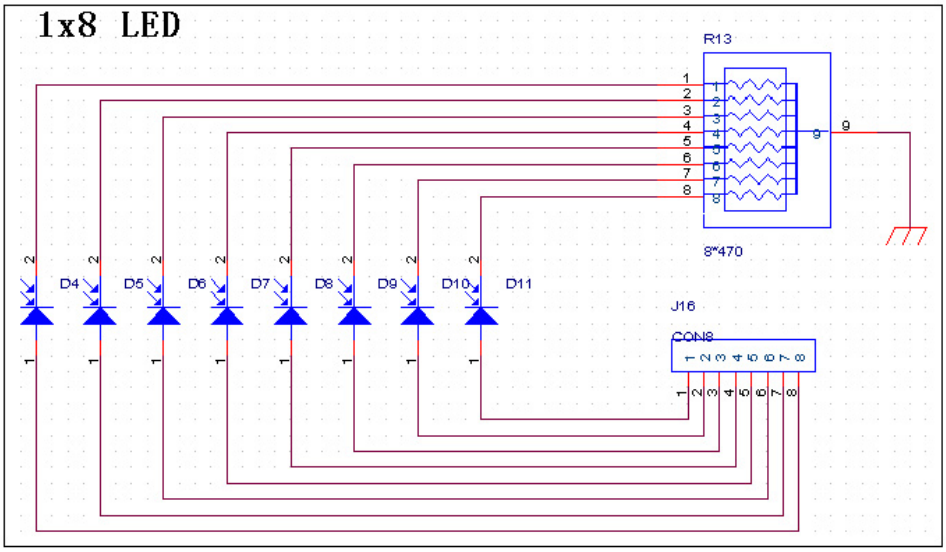
1）把 JP10 的三个短路线接好，注意 CF7 选择 B11。

2）用排线把 J26 和 J16 接上。

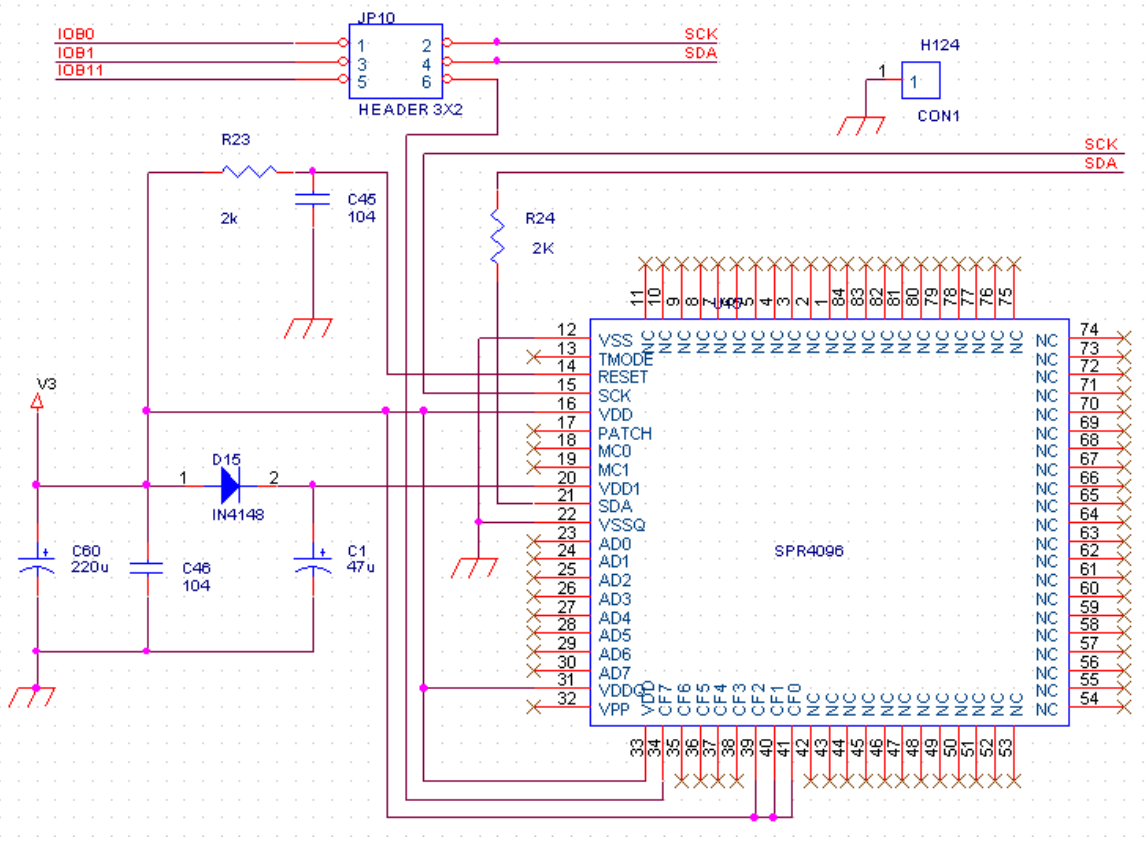
3）下载实验程序，观察现象。

【硬件连接图】

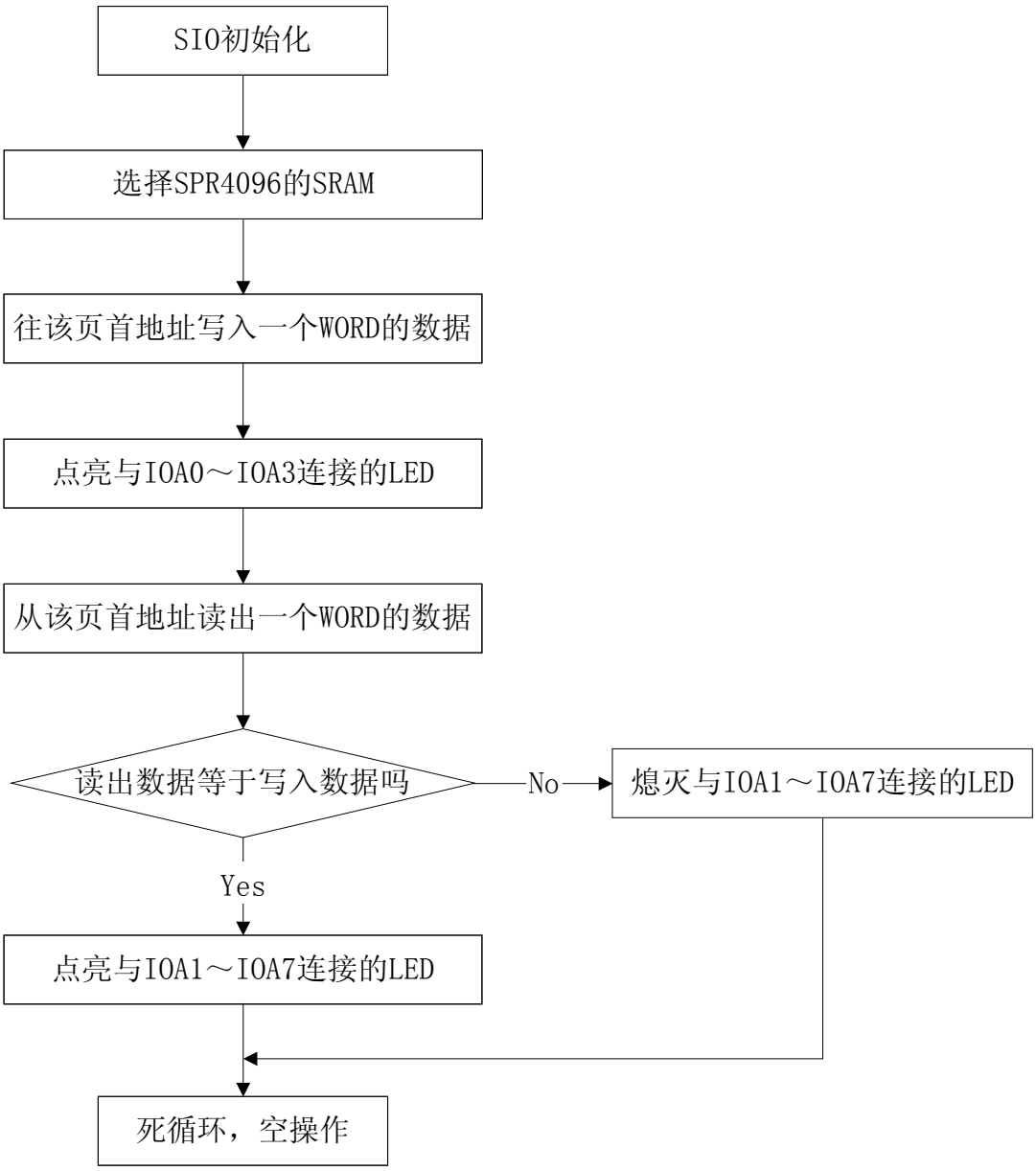
实验箱上发光二极管的原理图如下：



实验箱上 SPR4096 接口原理图如下：


JP10
IOB0
IOB1
IOB11
SCK
SDA
HEADER 3X2
H124
CON1
R23
2k
C45
104
R24
2K
V3
D15
IN4148
C60
220u
C46
104
C1
47u
SPR4096
VSS
TMODE
RESET
SCK
VDD
PATCH
MC0
MC1
VDD1
SDA
VSSQ
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
VDDQ
VPP
NC

【程序流程图】



【程序及其特殊函数说明】

本程序主要结合实验箱资源来完成一个 SPR4096 的 SRAM 操作，都采用实验箱上分配给各模块的 IO 资源，发光二极管由 IOA 的低八位控制。

第二章 综合实验

实验一 6 位 7 段 LED 数码管显示实验

【实验要求】

1)初始化时，使 6 位 LED 均显示 8，显示时间为 1s。

2)从第一个 LED 开始，从 0 显示到 9，0.5s 刷新一次。直到最后一个 LED。

【实验目的】

1)熟悉并进一步掌握定时器中断的使用和时基信号的使用。

2)进一步巩固 I/O 口的使用方法。

3)了解 6 位 7 段 LED 数码管的使用。

【实验设备】

1)装有 u'nsp IDE 仿真环境的 PC 机一台。

2)μ'nSP™十六位单片机实验箱一个。

【实验原理】

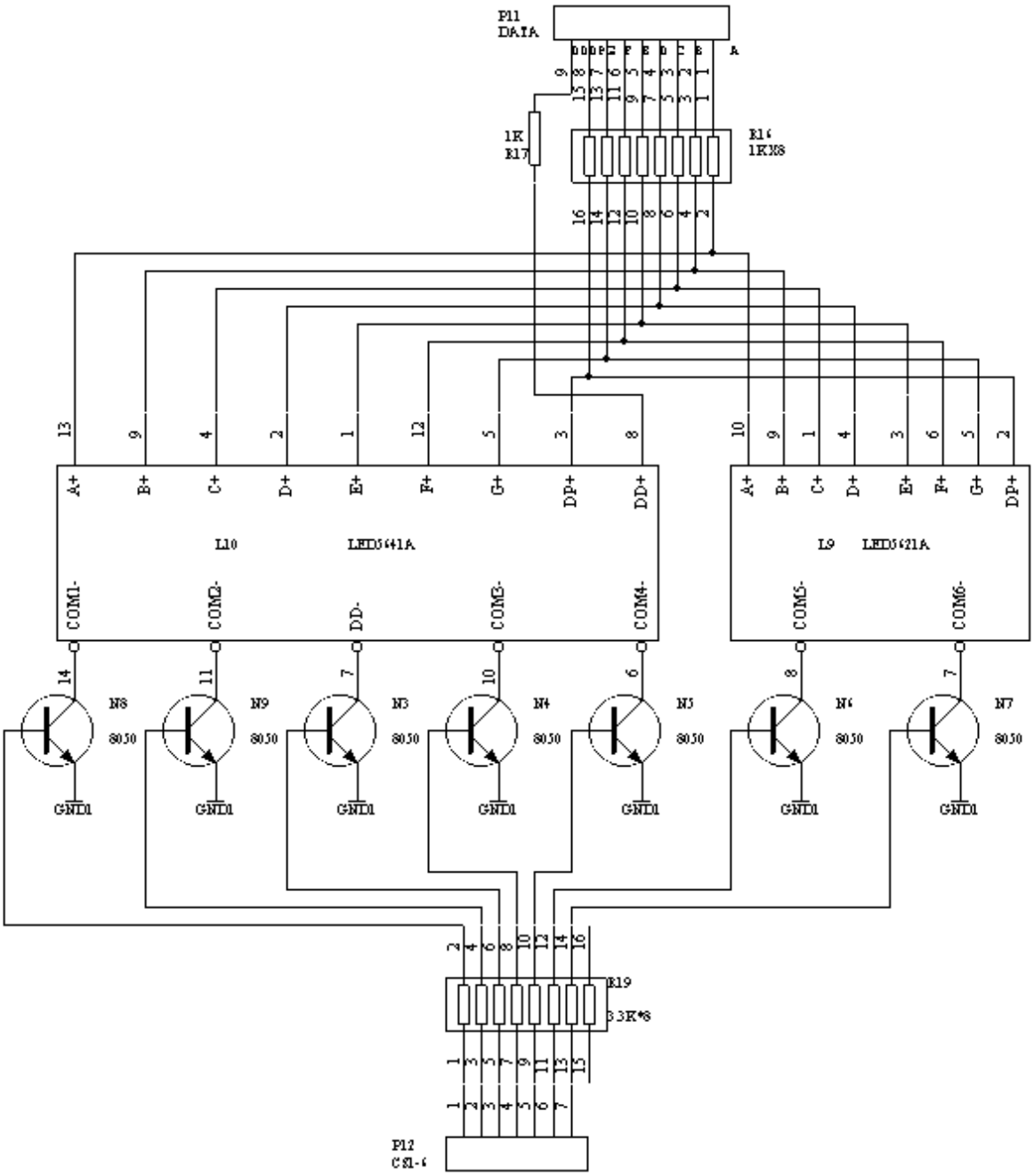
通过对 I/O 口的控制，初始化时点亮所有的数码管，即 6 位 LED 数码管均显示 8。1s 后，从第一位数码管开始从 0 显示到 9，刷新时间为 0.5s。直到最后一个数码管。1s 的时间使用定时器 A（FIQ）；0.5s 的时间使用 2HZ 的时基信号（IRQ5）。

【硬件连接图】

A0—A6　接　A---G

A8—A13　接　CS1—CS6

B0—B7　接　KEY



【实验步骤】

⑴按硬件电路原理图进行连接。

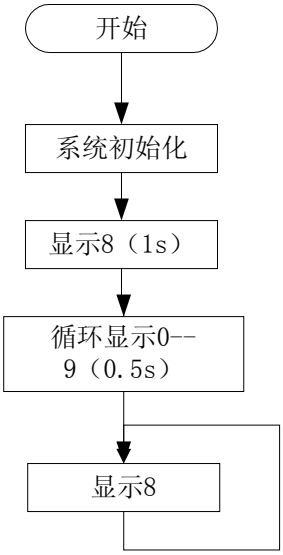
⑵画程序流程图。

⑶编写程序。

⑷调试程序。

⑸结合硬件调试，实现最终功能。

【主程序流程图】



【程序范例】

```
//*************************************/
//程序名称：main.c
//描述:   A0---A6 接 LED a--g
//A8--A14  接 LED CS1--6
//FIQ:定时器 A 为 5s   用于所有 LED 显示 8 的等待时间
//IRQ5: 2Hz 中断   用于每个 LED 显示 0—9 的数字刷新
//*************************************//
#include “hardware.h"                          //加头文件
typedef   unsigned char uchar
#define       true   1
#define       false 0
#define       DIG    6                        //定义共 6 个 LED 数码管
#define       Show_Value 10                   //共显示十个数字 0~9
uchar Interrupt_2Hz_flag
uchar TimeAFlag
int main()
{
      uchar dig                                 //LED 数码管的位数
      uchar show_value                          //显示的数值
      Interrupt_2Hz_flag = false
      TimeAFlag = false
      SP_Init_IOA()                             //初始化 A 口
      SP_Export(Port_IOA_Data,0xffff)           //A 口输出高电平
      SP_INT_TIMEA()                            //初始化定时器 A 并打开中断
```

```
    while(TimeAFlag != 4)                    // ‘8‘显示 1 秒
    SP_INT_IRQ5()                            //初始化中断为 2Hz 定时中断源
    for(dig =0;dig<DIG;dig++)                //显示 0~9
    {
        for(show_value=0;show_value<Show_Value;show_value++)
            {
                while(Interrupt_2Hz_flag != true)
            Interrupt_2Hz_flag = false
            show(dig,show_value)
        }
    }
    while(1)
    {
        SP_Export(Port_IOA_Data,0xffff)      //显示 8
    }
}
//************************//
//程序名称：system.asm
//描述：初始化函数和显示函数
//************************//
.include hardware.inc
.external   _Interrupt_2Hz_flag              //1 秒标识符
.ram
.public    sum
.var sum                                     // 进入中断的计数器
.data
//’0’,’1’,’2’,’3’,’4’,’5’,’6’,’7’,’8’,’9’的代码
address: .dw   0x00bf,0x0086,0x00db,0x00cf,0x00e6,0x00ed,0x00fc,0x0087,0x00ff,0x00ef
Dig:      .dw   0x0100,0x0200,0x0800,0x1000,0x2000,0x4000
.code                                        //选中 LED 管

//**********************//
//描述：初始化 A 口
//**********************//
.public _SP_Init_IOA                         //初始化 A 口为同相高电平输出口
_SP_Init_IOA: .proc
    r1 = 0xffff
    [P_IOA_Attrib] = r1
    [P_IOA_Dir] = r1
    [P_IOA_Data] = r1
    retf
.endp
//************************//
//向端口送数据
```

```
//***********************//
.public _SP_Export                          //输出的子程序
    _SP_Export: .PROC
                PUSH BP,BP TO [SP]              //堆栈保护
                BP = SP + 1
                PUSH R1,R2 TO [SP]
                R1 = [BP+3]
                R2 = [BP+4]
                [R1] = R2                       //读出数值
                POP R1,R2 FROM [SP]
               POP BP,BP FROM [SP]              //出栈
                RETF
                .ENDP
    //***************************//
    //描述：初始化中断为 2Hz 定时中断源
    //***************************//
    .public _SP_INT_IRQ5                        //初始化中断为 2Hz 定时中断源
    _SP_INT_IRQ5: .proc
          fiq off                               //关中断。
          r1 = 0x0004                           //允许 2Hz 的中断
    [P_INT_Ctrl] = r1
    INT IRQ;                                    //开中断
    retf
.endp;

//***************************//
//描述：初始化中断为 0.25s 定时中断源
//***************************//
.public _SP_INT_TIMEA;
_SP_INT_TIMEA: .proc
    R1 = 0x0000                                 //系统时钟选择 Fosc
    [P_SystemClock] = R1
    R1= 0xFFFF
    R1 = 0x0034
    [P_TimerA_Ctrl]=R1                          //TimerA 选择 4096Hz
    R1-= 0x0400
    [P_TimerA_Data]=R1                          //设置中断时间为 4*256/4096＝0.25(s)

R1 = C_FIQ_TMA
    [P_INT_Ctrl] = R1;                          //TimeA 中断的设置
    INT FIQ;
    retf
.endp
```

```
//*****************************//
//描述：数据显示函数
//入口：1、LED 的位数（DIG）
//          2、LED 的显示值
//无出口数据
//*****************************//
.public _show;
_show: .proc
  push bp to    [sp];                                  //弹出入口参数共两个入口参数
       bp = sp + 1
  loop:
       r1 = [bp+3]                                    //取出第一个入口参数
       r3 = [bp+4]                                    //取出第二个入口参数
       r2 =    r1 + Dig                               //取 LED 管的片选地址
       r2    = [r2]
       r4 =    r3 + address                           //取显示数据的地址
       r4 = [r4]
       r2 |= r4
       [P_IOA_Data] = r2;                             //使选中的数码管显示数据
       pop bp from [sp]
       retf;
  .endp
//************************************//
//程序名称：isr_TimeA_2Hz.asm
//描述：定时器 A 为 0.25s 中断
//2Hz 为 0.5s 中断
//*********************************************************************//
// Function: Fast Interrupt Service routine Area
//     Service for      (1)PWM FIQ
//                      (2)Timer A FIQ
//                      (3)Timer B FIQ
//     User's FIQ must hook on here
//     _FIQ:                                          // Fast interrupt entrence
//    _IRQ5:                                          // interrupt entrence
//*********************************************************************//
.include hardware.inc                                // include io information
.DEFINE C_IRQ_1024Hz              0x0010;   //1024Hz IRQ4
       .DEFINE C_IRQ_2048Hz               0x0020;   //2048 IRQ4
       .DEFINE C_IRQ_4096Hz               0x0040;   //4096 IRQ4
       .TEXT
       .public _FIQ;
       .public _IRQ5;
       .external sum;
       .external _Interrupt_2Hz_flag;
```

```
    .external _TimeAFlag;
    .external _Clear_WatchDog;
    _FIQ:
        push r1,r4 to [sp];
        call _Clear_WatchDog;
        r1 = C_FIQ_TMA;
        test r1,[P_INT_Ctrl];
        jne   L_FIQ_TimerA;                    // Timer A FIQ entrence
        r1 = C_FIQ_TMB;
        test r1,[P_INT_Ctrl];
        jne L_FIQ_TimerB;                      // Timer B FIQ entrence
        L_FIQ_PWM:                             // PWM FIQ entrence
        //-----------------------------------------------------------------
        // hook PWM FIQ subroutine here and define it to be external
        //-----------------------------------------------------------------
        r1 = C_FIQ_PWM;
        [P_INT_Clear] = r1;
        pop r1,r4 from [sp];
       reti;
 L_FIQ_TimerA:
        call _Clear_WatchDog;
        r1 = C_FIQ_TMA;
        [P_INT_Clear] = r1;

        r1 = [_TimeAFlag];
        r1 += 1
        [_TimeAFlag] = r1;
        R1=0xFFff;
        [P_IOA_Data]=R1;
        pop r1,r4 from [sp];
        reti;

L_FIQ_TimerB:
        [P_INT_Clear] = r1;
        pop r1,r4 from [sp];
        reti;

////////////////////////////////////////////////////////////////////////////////
// Function: Interrupt Service routine Area
//     Service for      IRQ5
//     User's IRQ must hook on here
////////////////////////////////////////////////////////////////////////////////

_IRQ5:                                         //定时 0.25 秒的中断程序
```

```
        push r1,r4 to [sp];

        r1 =0x0008;
        test r1,[P_INT_Ctrl];
        jnz   L_4Hz;                          // Timer A FIQ entrence

        r1 = 0x0004;
        [P_INT_Clear] = r1;                   //清中断

loop0:
        r1 = 0x0001;
        [_Interrupt_2Hz_flag] = r1;           //设置中断标识
        r1 = 0
        [sum] = r1
        pop r1,r4 from [sp];
        reti;

L_4Hz:
        r1 = 0x0008;
        [P_INT_Clear] = r1;                    //清中断
        pop r1,r4 from [sp];
        reti;
```

实验二 4*4 键盘输入在 LED 数码管上的显示

【实验要求】

1)在实验一基础上添加 4*4 键盘，使键盘输入的操作通过 LED 给予显示。

2)键盘实现的功能如键盘图。

【实验目的】

1)了解 4*4 键盘的使用方法。

2)进一步了解键唤醒的使用方法。

【实验设备】

1)装有 u'nsp IDE 仿真环境的 PC 机一台。

2)μ'nSP™十六位单片机实验箱一个。

【实验原理】

通过对键盘的操作在 LED 数码管上给予显示及相应操作。键盘扫描在时基中断中进行

键盘界面如下：

7	8	99	F1
44	55	66	F2
11	22	33	F3
DEL	00	F4	**ENT**

0~9:数字键

DEL：删除键，删除前一个数字。LED 上无数字时无响应。

ENTER：确认键，当进行时钟和日期设置时，按确认键才可显示输入时钟和日期。

F1、F2 和 F3、F4：保留键

【硬件连接图】

见综合实验一

【实验步骤】

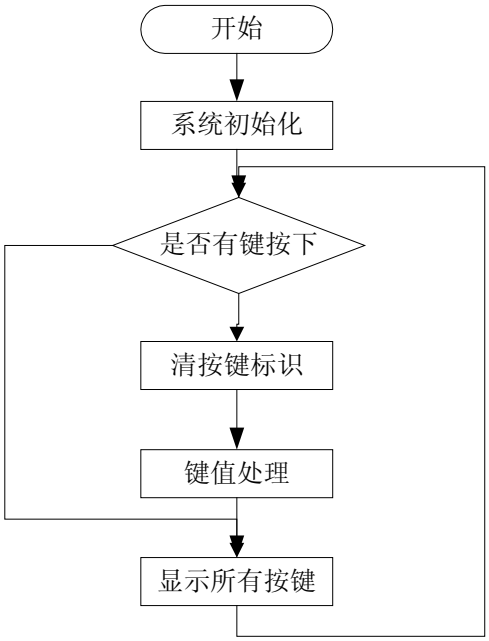
1)画程序流程图（包括各个键的处理）。

2)编写程序。

3)调试程序。

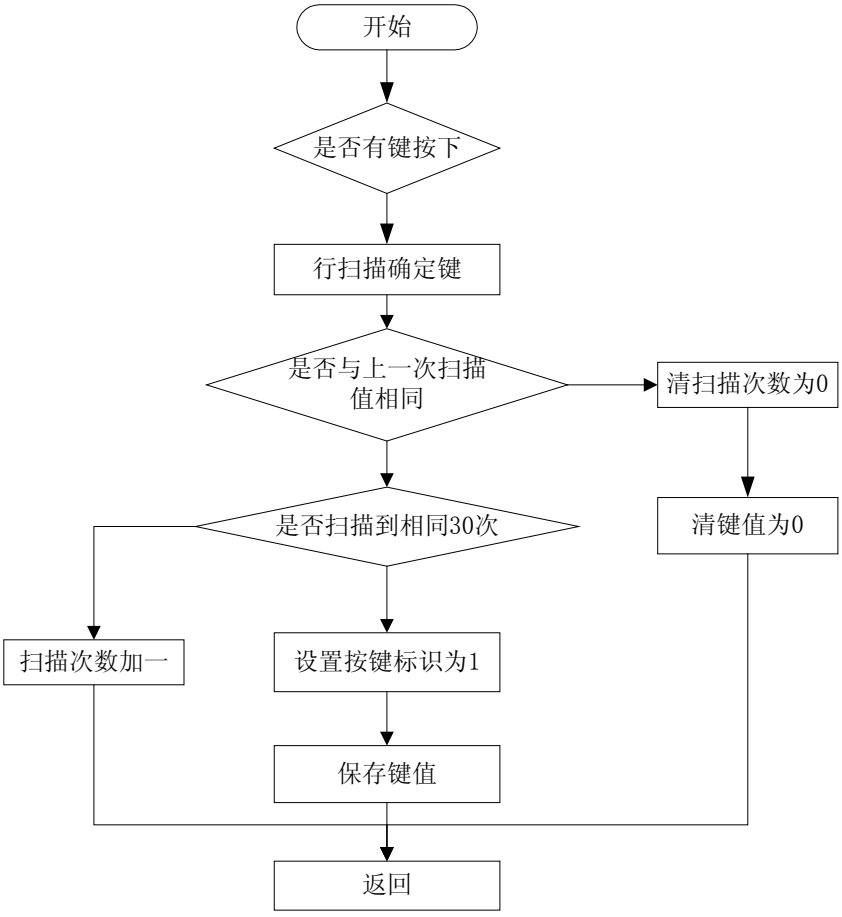
4)结合硬件调试，实现最终功能。

【主程序流程图】



【键盘扫描流程图】

在 128Hz 中断中的 键盘扫描



【程序范例】:程序代码详见光盘。

实验三 时钟实验

【实验要求】

1)通过键盘设置时钟并在所设置的时钟基础上继续增加。

2)设置时钟的输入显示，通过确认键来完成。

【实验目的】

1)加深了解定时中断和时基中断的使用

2)对 SPCE061 的熟练使用

【实验设备】

1)装有 u'nsp IDE 仿真环境的 PC 机一台。

2)μ'nSP™十六位单片机实验箱一个。

【实验原理】

在实验一和实验二基础上，通过按键实现时钟的设置。

【实验步骤】

1)画程序流程图（包括各个键的处理）。

2)编写程序。

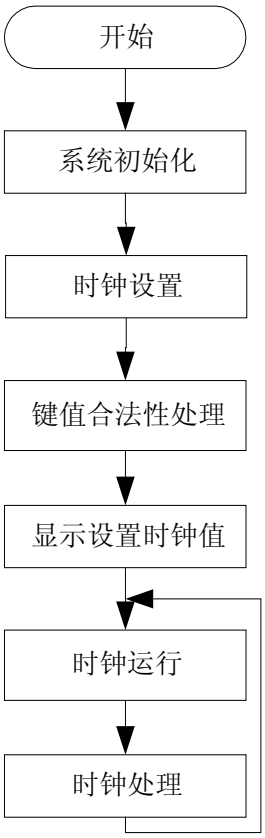
3)调试程序。

4)结合硬件调试，实现最终功能。

【硬件连接图】

见综合实验一

【主程序流程图】



【程序范例】:程序代码详见光盘。

实验四 LED 点阵模块

【实验目的】

进一步掌握 SPCE061 在 IDE 环境下的 C 及汇编语音编程。

【实验设备】

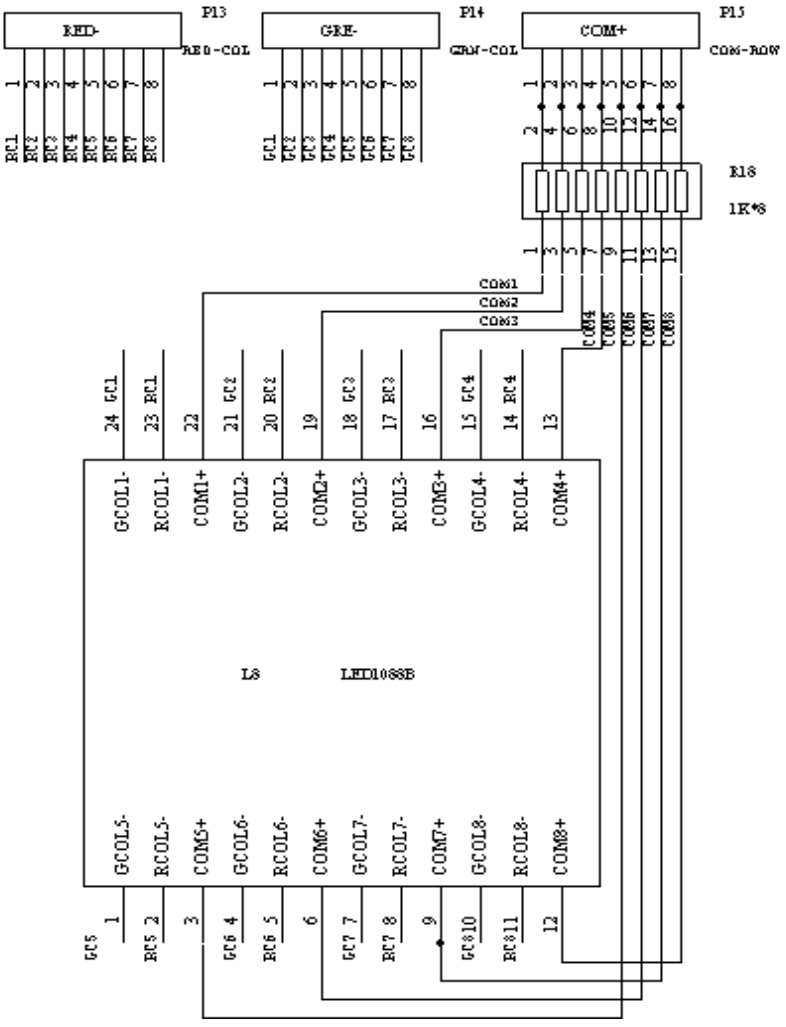
1)装有 u'nsp IDE 仿真环境的 PC 机一台

2)μ'nSP™十六位单片机实验箱一个

【实验原理】

用行驱动和列驱动的点亮 LED 点阵模块，当扫描频率大于 50HZ 时，人眼就能看到没有闪烁的字符或图形。

【硬件原理图】



图中 A0—A7 接 RED_Col

A8—A15 接 GREEN_Col

B0—B7 接 COM_Row ；

注：在综合实验五中

B0—B7 接 KEY

B8—B15 接 COM_Row

【实验步骤】

1)如图连接 LED 到仿真板的 PortA,PortB，接通电源。

2)打开 u'nSP IDE 开发环境，建立一个新工程。

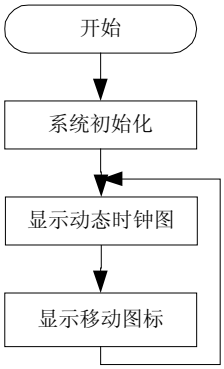
3)在源文件夹(Source File)下建立一个 C 语言源文件。

4)编写程序代码。

5)编译程序、调试程序。

5)运行程序，跟踪运行结果，观察各寄存器的状态、LED 模块的显示。

【主程序流程图】



【程序范例】：

```
    #include "hardware.h"
    unsigned char Pattern[15][8]=                    //要显示的时钟图形的数据
    {
        { 0x1C,0x22, 0x51,0x4F, 0x41,0x22, 0x1C,0x00 },
        { 0x1C,0x2a, 0x49,0x4F, 0x41,0x22, 0x1C,0x00 },
        { 0x1C,0x22, 0x45,0x4F, 0x41,0x22, 0x1C,0x00 },
        { 0x1C,0x22, 0x41,0x4F, 0x41,0x22, 0x1C,0x00 },
        { 0x1C,0x22, 0x41,0x4F, 0x45,0x22, 0x1C,0x00 },
        { 0x1C,0x22, 0x41,0x4F, 0x49,0x2a, 0x1c,0x00 },
        { 0x1C,0x22, 0x41,0x4F, 0x51,0x22, 0x1c,0x00 },
        { 0x1C,0x22, 0x41,0x7F, 0x41,0x22, 0x1C,0x00 },
        { 0x20,0x3C, 0x23,0x61, 0x23,0x3C, 0x20,0x00 },
        { 0x08,0x1C, 0x3E,0x08, 0x08,0x3E, 0x1C,0x08 },
        { 0x78,0x48, 0x4F,0x49, 0x4F,0x48, 0x78,0x00 },
        { 0x78,0x78, 0x4F,0x49, 0x4B,0x78, 0x78,0x00 },
        { 0x7F,0x47, 0x57,0x50, 0x57,0x47, 0x7F,0x00 },
        { 0x7E,0x78, 0x78,0x78, 0x78,0x78, 0x7E,0x00 },
        { 0x3E,0x22, 0x3E,0x00, 0x2E,0x2A, 0x3E,0x00 },
  };

void InitialPort()
{
    SP_Init_IOB(0xffff,0xffff,0xffff)          //B 口为带数据缓存器的高电平输出
    SP_Init_IOA(0xffff,0xffff,0);              //A  口的设置.
}
```

```
void delay(char n)
{
    int i,j ;                                   //延时
    for(i=0;i<n;i++)                            //i 从 0 记数到 n.
    for(j = 0;j<50;j++);                        //j 从 0 记数到 50
}
void Clock(void)                                //时钟图形
{
        int i,j,k,SelCol,Reload = 1;
    for(i=0;i<8;i++)
    {
        for(k=0;k<80;k++)
            {
            SelCol=Reload;
            for(j = 0;j<8;j++)
                {
                    SP_Export(Port_IOA_Data,SelCol);
                SP_Export(Port_IOB_Data,Pattern[i][j]);
                    delay(1);
                SelCol = SelCol<<2;             //左移 2 位
                }
        }
    }
}
void WalkMan(void)                              //人的走动图形
{
    char state = 0;//0:左  1:右
    unsigned int i,j,k,l=0,SelCol = 2,Reload = 1;
    unsigned char     ManIcon[8]= { 0x20,0x94,0xCA,0x7A,0xCA,0x94,0x20,0x00 };
    for(j=0;j<20;j++)                           //j 从 0 记数到 20
    {
        switch(state)
        {
                case 0:
                Reload = Reload << 2;           // 状态 0，图形向左移
                if(Reload == 0)
                    {
                    Reload = 0x8000;            //0x8000    shift left
                    state = 1;                  //left state
                }
                break;
            case 1:                             //状态 1，图形向右移
                Reload = Reload >> 2;           //右移 2 位
                if(Reload == 0)
```

```
                {
                    Reload = 0x0001;
                    state = 2;
                }
                break;
            case 2:                                  //状态 2，图形向左溢出屏
                l++;
                Reload = 1;
                    if(l==8)
                {
                    state = 0;
                    l = 0;
                }
                break;
        }
        for(k=0;k<8;k++)
        {
            SelCol = Reload;
            for(i = l;i<8;i++)
            {
              SP_Export(Port_IOA_Data,SelCol);
               //A 口输出
              SP_Export(Port_IOB_Data,ManIcon[i]);
                  //B 口输出
               delay(1);                             //延时
               SelCol = SelCol<<2;                   //左移 2 位
            }
        }
    }
}
int   main(void)
    {
    InitialPort();                                   //调处始化程序
    while(1)
        {
            WalkMan();                                   //显示为人的图形
        Clock();                                     //显示为时钟的图形
    }
        return 0;
}
```

【程序练习】

分别用 C 语言和汇编语言实现字符的滚动，翻屏，反相显示。

实验五 4*4 键盘在 LED 点阵上的应用

【实验要求】

1）数字键在 LED 点阵上有数字显示

2）每个数字键均有图形映射，即每一显示所按数字键后均有一图形相应显示。

3）数字键显示 1s。

【实验目的】

进一步了解 SPCE061 的应用。

【实验设备】

1)装有 u'nsp IDE 仿真环境的 PC 机一台。

2)μ'nSP™十六位单片机实验箱一个。

【实验原理】

键盘上数字键在 LED 点阵上的显示。即每一显示所按数字键后均有一图形相应显示。

【硬件连接图】

见综合实验四。

【实验步骤】

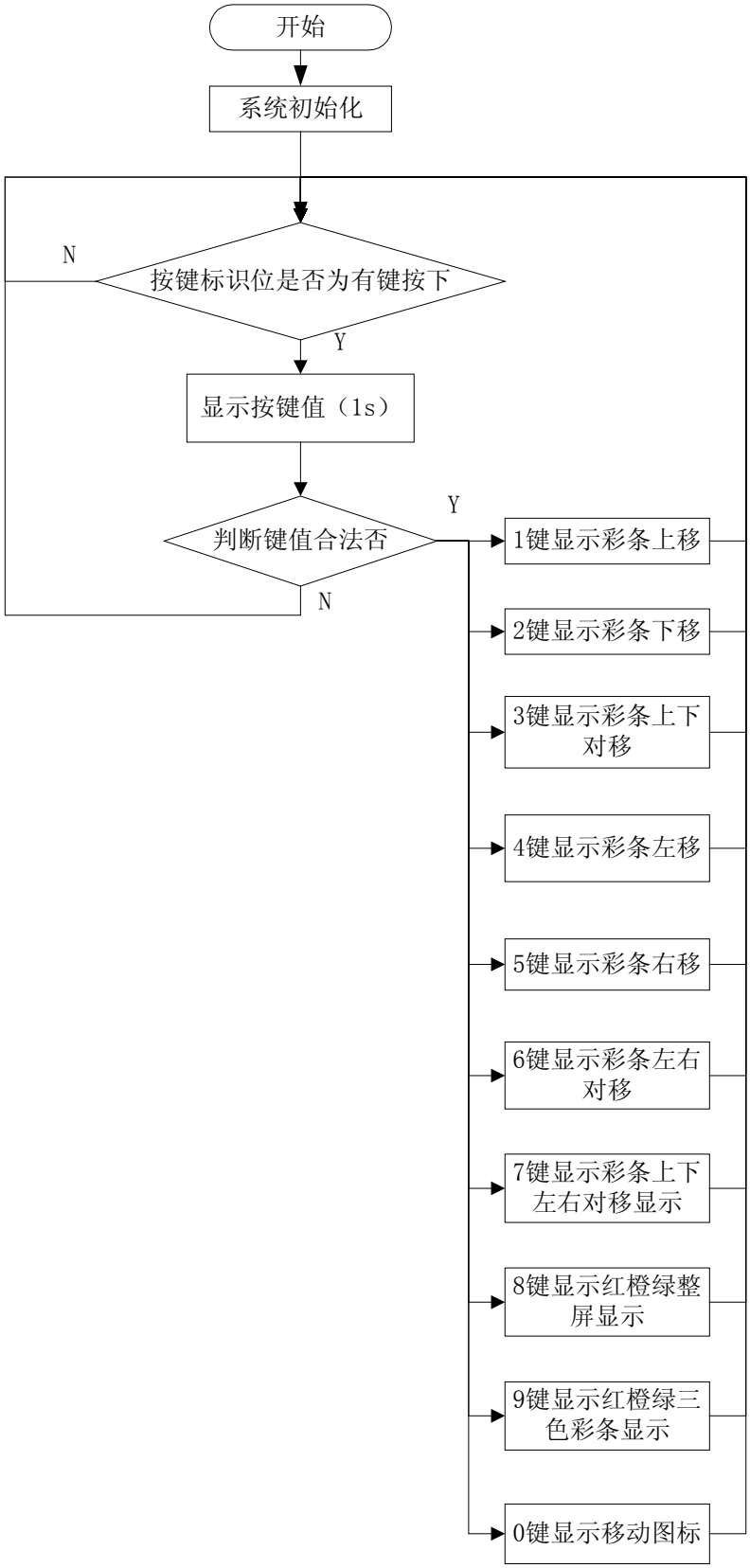
⑴画程序流程图（包括各个键的处理）。

⑵编写程序。

⑶编译程序、调试程序。

⑷结合硬件调试，实现最终功能。

【主程序流程图】



【程序范例】程序代码详见光盘。

实验六 4*4 键盘播放语音

【实验目的】

进一步了解 SPCE061 的外部应用及其语音播放功能。

【实验设备】

1)装有 u'nsp IDE 仿真环境的 PC 机一台。

2)μ'nSP™十六位单片机实验箱一个。

【实验原理】

键盘上数字键在 LED 数码上的显示。同时有语音播放。

【硬件连接图】

如实验一 ：A0—A6 接 A—G；A8—A13 接 CS1—CS6；B0—B8 接 KEY

【实验步骤】

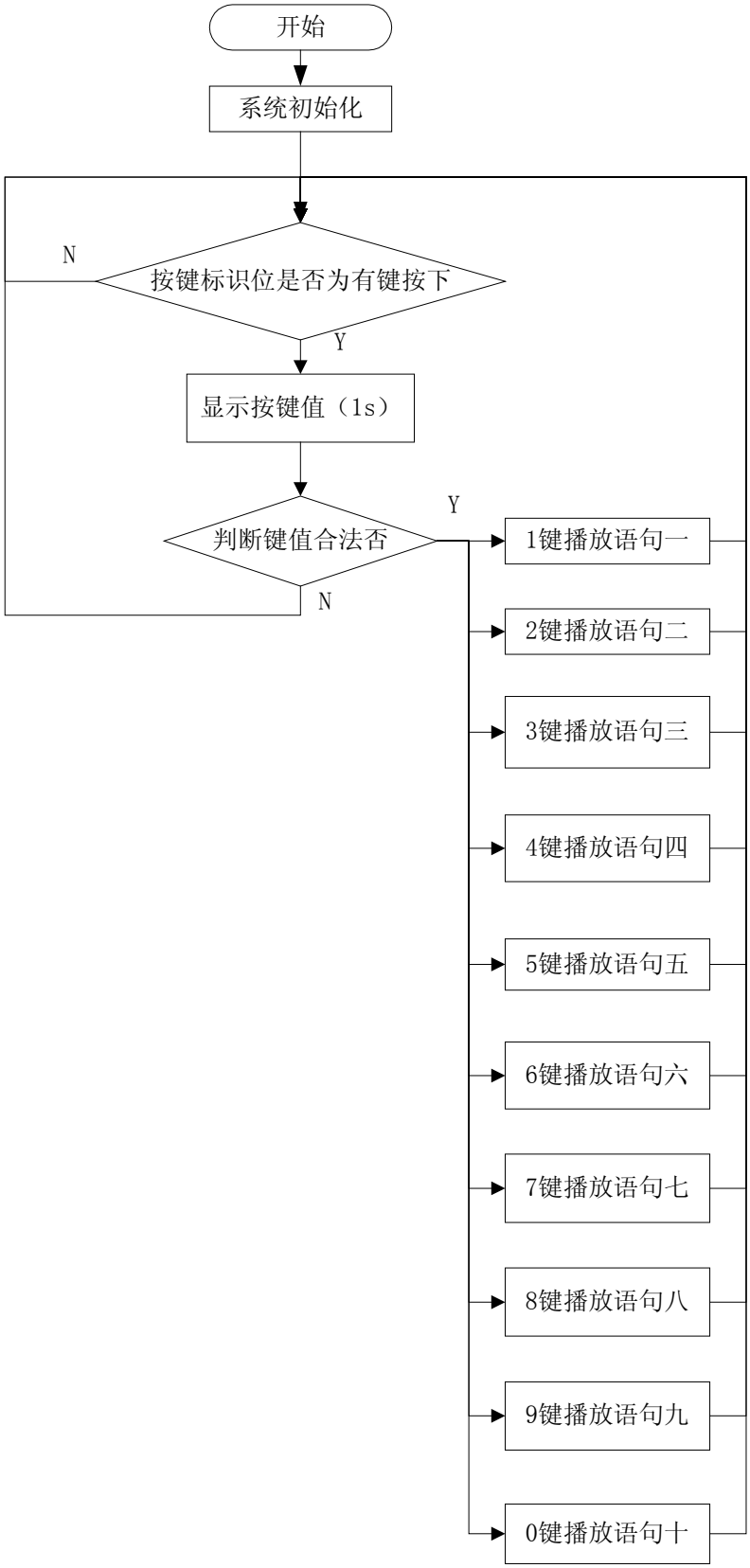
⑴画程序流程图（包括各个键的处理）。

⑵编写程序。

⑶调试程序。

⑷结合硬件调试，实现最终功能。

【主程序流程图】



【程序范例】

程序代码详见光盘。

实验七 并口扩展 ROM（M27 C4001）

【实验目的】

1)了解 M27c4001 ROM 芯片的使用方法

2)了解在 SPCE061A 的 I/O 直接外挂 M27C4001 的编程方法。

【实验设备】

1)装有 u'nsp IDE 仿真环境的 PC 机一台。

2)μ'nSP™十六位单片机实验箱一个。

3）M27C4001 一个。

【实验原理】

在 SPCE061A 的 I/O 直接外挂 M27C4001，实现对 M27C4001 的数据（读）访问。烧录语音数据到 M27C4001 中，然后通过 SPCE061A 手动播放模式播放语音。

注意：Rom 中的语音数据应先写到 Rom 中，并清楚它的存放地址。

【硬件连接图】

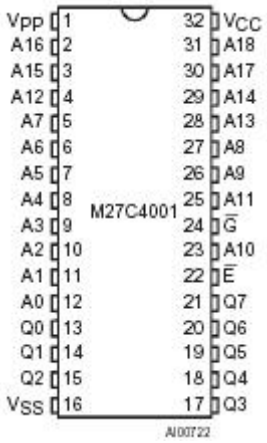
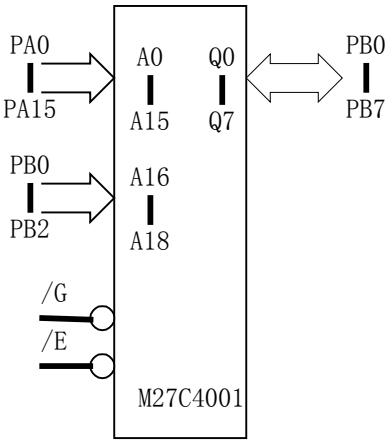


Table 1. Signal Names

A0-A18	Address Inputs
Q0-Q7	Data Outputs
E	Chip Enable
G	Output Enable
V_{PP}	Program Supply
V_{CC}	Supply Voltage
V_{SS}	Ground



注意：在这里为了节约管脚/G 和/E 段可以接地。

【实验步骤】

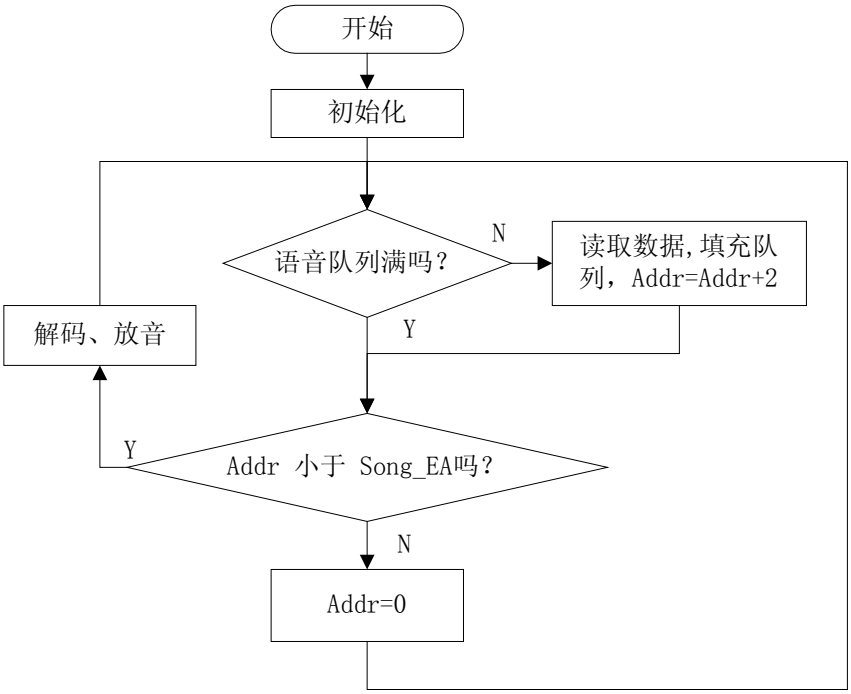
1）画程序流程图。

2）编写程序。

3）编译程序、调试程序。

4）结合硬件调试，实现最终功能。

【主程序流程图】



主程序流程图

【程序范例】

程序代码详见光盘

实验八 并口扩展 SRAM（HM628128DLP5）

【实验目的】

1)通过实验掌握并行扩展 SRAM 的方法，了解 HM628126 存储器的性能参数、各种指标及使用方法。

2)通过实验巩固 I/O 端口的设置。

3)通过实验学习掌握通用函数的编写方法。

4)通过实验参考程序了解学习汇编函数与 C 进行参数传递的方法。

【实验设备】

1)装有 u'nsp IDE 仿真环境的 PC 机一台

2)μ'nSP™十六位单片机实验箱一个

【实验原理】

要对单片机扩展存储器件，需首先仔细了解此存储器件的各样的性能指标。SRAM 静态存储器 HM628128 为 5V 电压供电，其容量为 128k×8bit，具有 17 条地址线 8 条数据线，访问速度为 55/70/85/100ns。其管脚排列及说明如图 1 所示。

HM628128ALP/ALFP Series

Pin Name	Function
A0 – A16	Address
I/O0 – I/O7	Input/output
$\overline{CS1}$	Chip select 1
CS2	Chip select 2
$\overline{WE}$	Write enable
$\overline{OE}$	Output enable
NC	No connection
V_{CC}	Power supply
V_{SS}	Ground

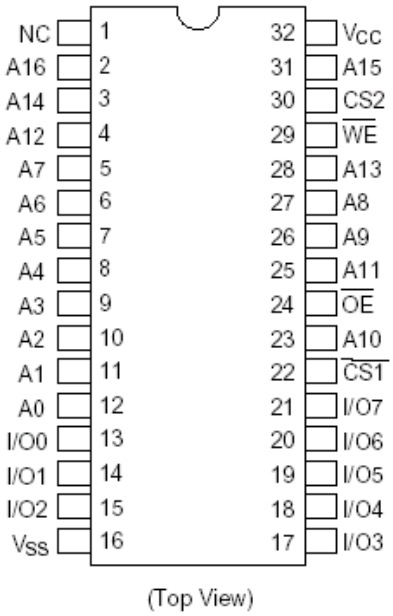


图 1　HM628128 管脚排列及说明

读写控制管脚的功能表如图 2 所示。

$\overline{CS1}$	CS2	$\overline{OE}$	$\overline{WE}$	Mode	V_{CC} Current	I/O Pin	Ref. Cycle
H	X	X	X	Standby	I_{SB}, I_{SB1}	High-Z	—
X	L	X	X	Standby	I_{SB}, I_{SB1}	High-Z	—
L	H	H	H	Output disable	I_{CC}	High-Z	—
L	H	L	H	Read	I_{CC}	Dout	Read cycle
L	H	H	L	Write	I_{CC}	Din	Write cycle (1)
L	H	L	L	Write	I_{CC}	Din	Write cycle (2)

Note: X: H or L

图 2 读写控制引脚的功能

要实现对存储器的进行读取功能，那么在功能上可以分为三个功能模块：存储器端口初始化、读数据、写数据。

根据存储器的管脚功能，在存储器端口初始化中需完成一下内容：

单片机连接选端 $\overline{CS1}$、读写控制端 $\overline{WE}$、$\overline{OE}$ 的引脚均设置为输出状态并置为高电平，使得存储器数据及地址端口均为高阻状态。

单片机连接存储器的数据线的引脚、地址线的引脚均设置为输出高状态。

对于读存储器的时序图如图 3 所示。

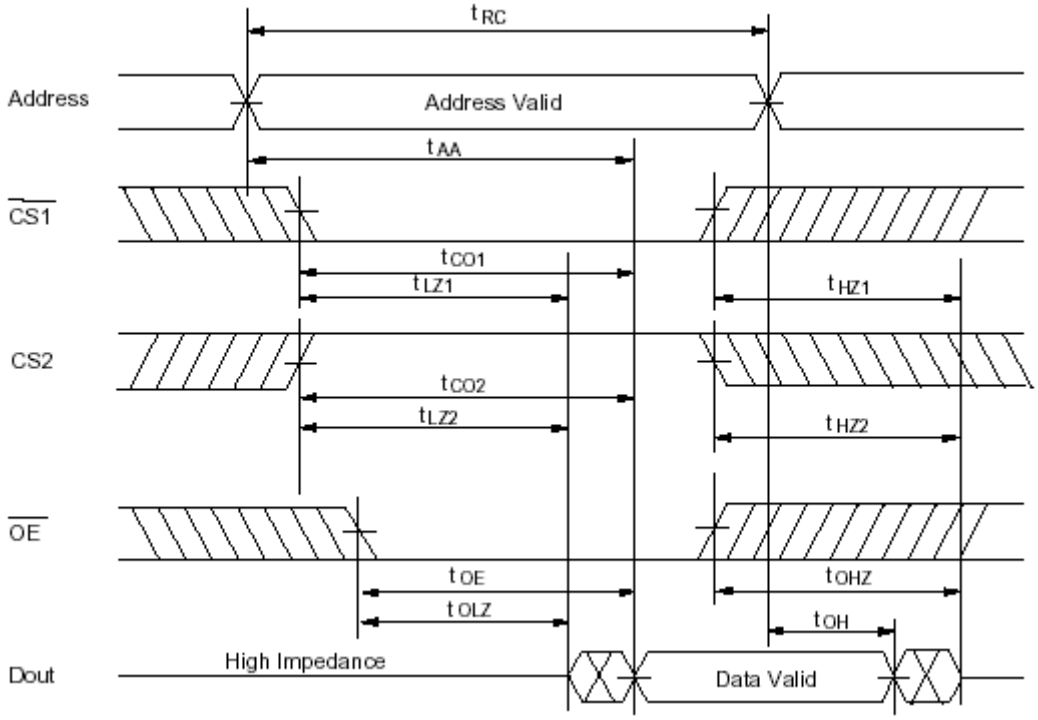


图 3 读时序图（WE=H）

图 3 所示读操作时序图中，相应的时间要求如图 4 所示。

		HM628128A									
		-5		-7		-8		-10			
Parameter	Symbol	Min	Max	Min	Max	Min	Max	Min	Max	Unit	Notes
Read cycle time	t_{RC}	55	—	70	—	85	—	100	—	ns	
Address access time	t_{AA}	—	55	—	70	—	85	—	100	ns	
Chip selection to output valid	t_{CO1}	—	55	—	70	—	85	—	100	ns	
	t_{CO2}	—	55	—	70	—	85	—	100	ns	
Output enable to output valid	t_{OE}	—	30	—	35	—	45	—	50	ns	
Chip selection to output in low-Z	t_{LZ1}	5	—	10	—	10	—	10	—	ns	2, 3
	t_{LZ2}	5	—	10	—	10	—	10	—	ns	2, 3
Output enable to output in low-Z	t_{OLZ}	5	—	5	—	5	—	5	—	ns	2, 3
Chip deselection to output in high-Z	t_{HZ1}	0	20	0	25	0	30	0	35	ns	1, 2, 3
	t_{HZ2}	0	20	0	25	0	30	0	35	ns	1, 2, 3
Output disable to output in high-Z	t_{OHZ}	0	20	0	25	0	30	0	35	ns	1, 2, 3
Output hold from address change	t_{OH}	5	—	10	—	10	—	10	—	ns	

Notes: 1. t_{HZ} and t_{OHZ} are defined as the time at which the outputs achieve the open circuit conditions and are not referred to output voltage levels.
2. At any given temperature and voltage condition, t_{HZ} max is less than t_{LZ} min both for a given device and from device to device.
3. This parameter is sampled and not 100% tested.

图 4 读时序的时间指标

通过上面的存储器读的时序图以及相应的时间指标，在单片机中编写读取存储器的程序，需按顺序完成下列过程：

把地址线端口设为输出状态，并根据所给的地址设置端口电平。

片选信号、读信号有效。（$\overline{CS1}=L$、$\overline{OE}=L$）

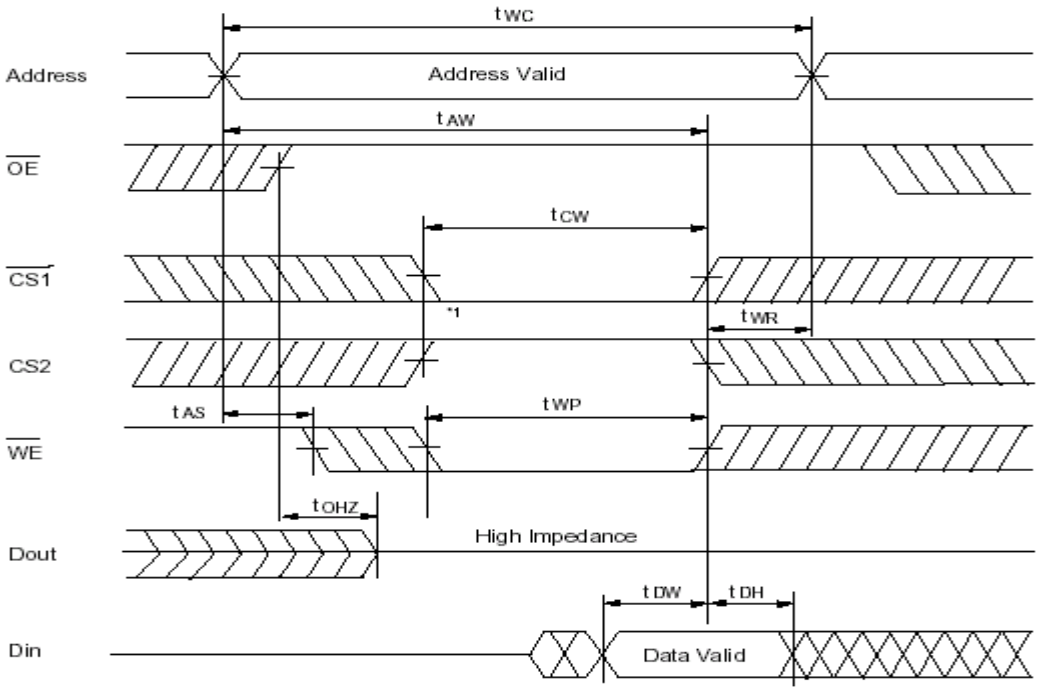
等待一个 NOP，使得数据电平稳定。

读取数据端口数据。

片选信号、读信号无效。（$\overline{CS1}=H$、$\overline{OE}=H$）

地址线设为具有上拉的输入状态。

对于写存储器有两种情形：$\overline{OE}=H$、$\overline{WE}=L$ 和 $\overline{OE}=L$、$\overline{WE}=L$。为简单起见，本实验仅讨论和实现 $\overline{OE}=H$、$\overline{WE}=L$ 的情况下的写操作，其时序图如图 5 所示，相应的时间指标如图 6 所示。



Notes: 1. If the $\overline{CS1}$ goes low simultaneously with $\overline{WE}$ going low or after the $\overline{WE}$ going low, the outputs remain in a high impedance state.

图 5 $\overline{OE}$ = H 、$\overline{WE}$ = H 存储器的写入时序图

		-5		-7		-8		-10			
Parameter	Symbol	Min	Max	Min	Max	Min	Max	Min	Max	Unit	Notes
Write cycle time	t_{WC}	55	—	70	—	85	—	100	—	ns	
Chip selection to end of write	t_{CW}	50	—	60	—	75	—	80	—	ns	2
Address setup time	t_{AS}	0	—	0	—	0	—	0	—	ns	3
Address valid to end of write	t_{AW}	50	—	60	—	75	—	80	—	ns	
Write pulse width	t_{WP}	40	—	50	—	55	—	60	—	ns	1, 7
Write recovery time	t_{WR}	0	—	0	—	0	—	0	—	ns	4
Write to output in high-Z	t_{WHZ}	0	20	0	25	0	30	0	35	ns	5, 6
Data to write time overlap	t_{DW}	25	—	30	—	35	—	40	—	ns	
Data hold from write time	t_{DH}	0	—	0	—	0	—	0	—	ns	
Output active from end of write	t_{OW}	5	—	5	—	5	—	5	—	ns	6
Output disable to output in High-Z	t_{OHZ}	0	20	0	25	0	30	0	35	ns	5

Notes:
1. A write occurs during the overlap of a low $\overline{CS1}$, a high CS2, and a low $\overline{WE}$. A write begins at the latest transition among $\overline{CS1}$ going low, CS2 going high, and $\overline{WE}$ going low. A write ends at the earliest transition among $\overline{CS1}$ going high, CS2 going low, and $\overline{WE}$ going high. t_{WP} is measured from the beginning of write to the end of write.
2. t_{CW} is measured from the later of $\overline{CS1}$ going low or CS2 going high to the end of write.
3. t_{AS} is measured from the address valid to the beginning of write.
4. t_{WR} is measured from the earliest of $\overline{CS1}$ or $\overline{WE}$ going high or CS2 going low to the end of write cycle.
5. During this period, I/O pins are in the output state; therefore, the input signals of the opposite phase to the outputs must not be applied.
6. This parameter is sampled and not 100% tested.
7. In the write cycle with $\overline{OE}$ low fixed, t_{WP} must satisfy the following equation to avoid a problem of data bus contention. $t_{WP} \geq t_{DW}$ min + t_{WHZ} mix

图 6 存储器的写入时序图中的时间指标

我们在编写存储器写操作的时候需要遵循以下步骤：

把地址线端口设为输出状态，并根据所给的地址设置端口电平。

片选信号、写信号有效。（$\overline{CS1}=L$ 、 $\overline{WE}=L$ ）

把数据端口设为输出状态，并根据说给的数据设置端口电平。

等待一个 NOP，使得数据线上电平稳定。

片选信号、写信号无效。（$\overline{CS1}=H$ 、 $\overline{WE}=H$ ）

地址线设为具有上拉输入状态。

根据以上对 HM628128 芯片的介绍，可以设计管脚接线的电路图如图 7 所示。

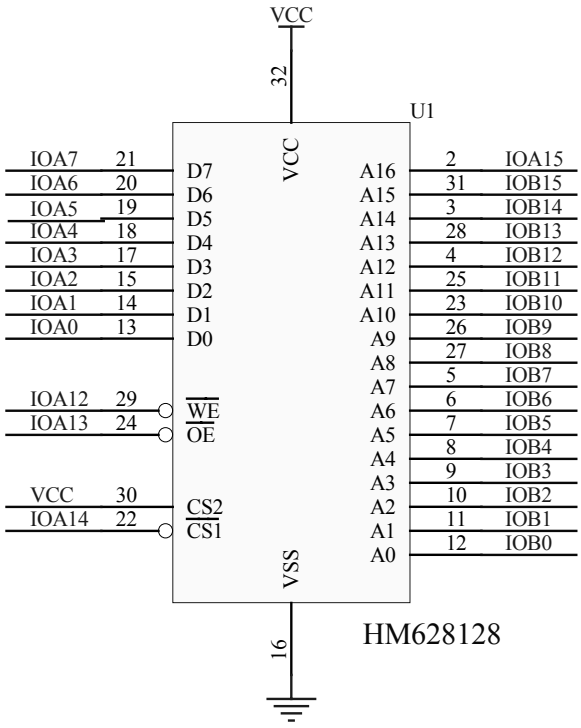


图 7 存储器扩展 HM628128 管脚接线图

由于 SPCE061 上电初始状态下，各端口处于输入下拉状态，所以按照此硬件接线方式，在单片机上电后，存储器的 0x00000 地址的初始数据一定为 0x00。

【实验步骤】

1）设计并画出电路图。

2）画出程序流程图并编写程序代码。

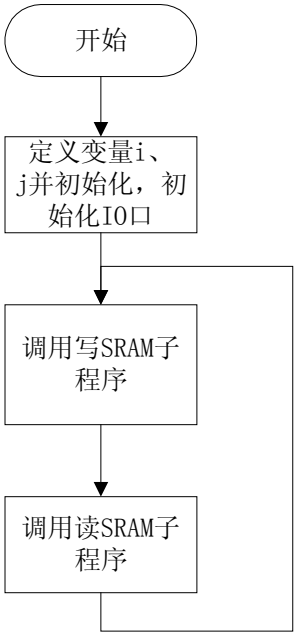
3）按照电路图连接硬件。

4）调试程序。

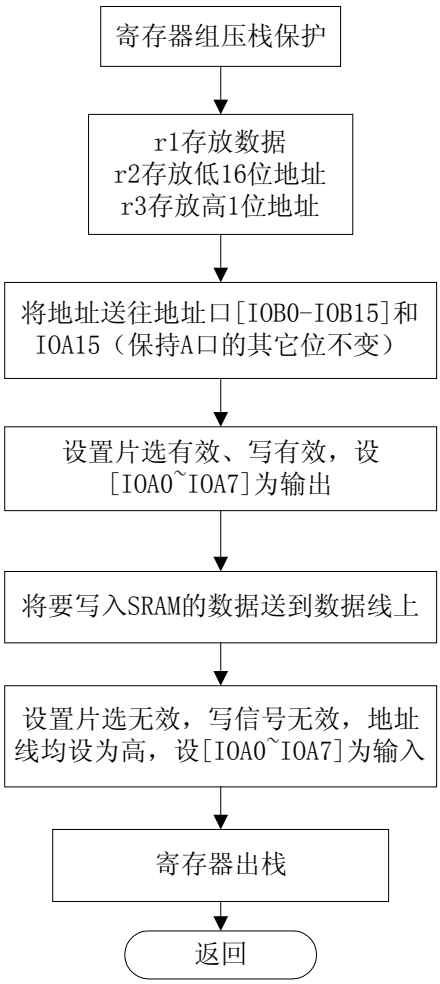
5）检查程序是否正确，若不正确返回第一步开始检查 bug。

【程序流程图】

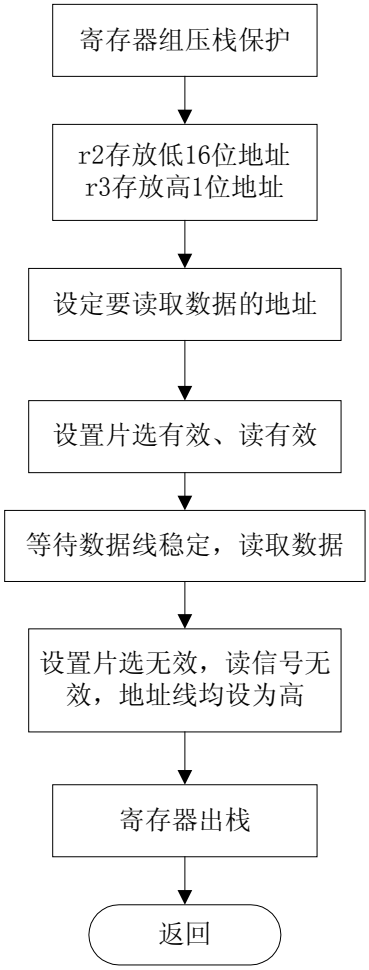
主程序流程图：



写SRAM子程序流程图：



读SRAM子程序流程图：



【程序范例】

```
//关于读写程序在文件 sram.asm 中，调试测试程序在 C 语言文件中 main.c 中。
//汇编程序文件 sram.asm 的内容如下：

//************************************************************
//访问 SRAM:HM628128 需要三个函数:
//          1、初始化
//          2、读函数
//          3、写函数
//硬件接线方法：
//          IOA0~IOA7 共 8 个端口为数据线
//          IOB0~IOB15、IOA15 共 17 个端口为地址线
//          IOA12 为写控制线/WE，低电平有效
//          IOA13 为读控制线/OE，低电平有效
//          IOA14 为片选控制/CS1，低电平有效
//************************************************************

.include hardware.inc

//************************************************************
//访问 SRAM:HM628128 初始化函数
//1、片选以及读写引脚均设置为输出状态并置为高电平。
//2、数据线的引脚设为下拉输入状态
//3、连接地址线的的引脚设置为输出高状态。
//汇编调用函数名：F_sram628128_Init
//C 语言调用函数名：void F_sram628128_Init(void)
//使用寄存器：r1
//返回值：无
//************************************************************
.public _F_sram628128_Init;
.public F_sram628128_Init
.code
_F_sram628128_Init:     .proc
F_sram628128_Init:
      //设置 IOA12、13、14 为高电平输出
      //设置 IOA0~IOA7、IOA15 为下拉输入

      //采用"读－修改－写"的方法，不影响其他端口的设置
      r1 = [P_IOA_Dir];
      r1 |= 0xf000;                              //设置 IOA12、13、14、15 为输出
      r1 &= 0xff00;                              //设置 IOA0~IOA7 为输入
      [P_IOA_Dir] = r1;

      r1 = [P_IOA_Attrib];
```

```
    r1 |= 0xf0ff;                          //设置 IOA12、13、14、15 为同相输出
                                           //设置 IOA0~IOA7 为悬空输入
    [P_IOA_Attrib] = r1;

    r1 = [P_IOA_Data];
    r1 |= 0xf000;                          //设置 IOA12、13、14、15 为输出高电平
    [P_IOA_Data] = r1;

    //设置 IOB0~IOB15 为带数据缓存器的高电平输出.
    r1 = 0xffff;
    [P_IOB_Dir] = r1;
    [P_IOB_Attrib] = r1;
    [P_IOB_Data] = r1;
  retf;
    .endp
//*************************************************************
//访问 SRAM:HM628128 读取函数
//汇编调用函数名：F_sram628128_Read
//参数参数传递：
//    输入：
//    r2 存放地址低 16 位
//    r3 存放地址高 1 位
//    输出：
//    r1 存放读取的数据
//调用方法：
//    r2 =  地址低 16 位
//    r3 =  地址高 1 位
//    call F_sram628128_Read
//    8 位数据  = r1
//使用寄存器：r1,r2,r3
//*************************************************************
.public F_sram628128_Read;
F_sram628128_Read:    .proc
    //地址低 16 位端口设定
    [P_IOB_Data] = r2;                     //IOB 输出低 16 位地址
                                           //地址高第 17 位设定
    r1 = [P_IOA_Data];                     //取 IOA 端口状态
    r1 = r1 lsl 1;                         //左移一位
    r3 = r3 ror 4;                         //把 r3.0 位移到 SB.0
    r1 = r1 ror 1;                         //SB.0 移到 r1.15
    [P_IOA_Data] = r1;                     //实现了位传送：r3.0=>[P_IOA_Data].15

    //设置片选有效、读有效
    r1 = [P_IOA_Data];
```

```
    r1 &= 0x9fff;
    [P_IOA_Data] = r1;

    nop;                                    //等待数据线稳定
    r1 = [P_IOA_Data];
    r1 &= 0x00ff;                           //从低 8 位取数据

    //设置片选无效、读无效,IOA15 为高
    r2 = [P_IOA_Data];
    r2 |= 0xf000;                           //设置 IOA12、13、14、15 为输出高电平
    [P_IOA_Data] = r2;

    r2 = 0xffff;                            //地址线均为高
    [P_IOB_Data] = r2;

    retf;
    .endp

//************************************************************
//访问 SRAM:HM628128 读取函数
//C 语言调用函数函数原型：
//      unsigned int F_sram628128_Read(unsigned long int Addr)
//输入参数：17 位宽度地址
//返回值：8 位数据
//调用方法：
//          unsigned int Data;
//          unsigned long int Addr;
//          Addr = 0x0001abcd;
//          Data = F_sram628128_Read(Addr)
//使用寄存器：r1,r2,r3,r4
//************************************************************
.public _F_sram628128_Read;
_F_sram628128_Read:  .proc
    push r2,r3 to [sp];                     //保护现场
    push bp to [sp];                        //保护 bp 值
    bp = sp + 1;                            //调整 bp 指针指向堆栈栈顶数据
    //栈顶到栈低数据依次为:
//bp,r2,r3,pc,SR,程序段地址,C 函数 32 位整形参数低 16 位，C 函数 32 位整形参数高 16 位
    r2 = [bp+5];                            //取 C 函数 32 位整形参数低 16 位
    r3 = [bp+6];                            //取 C 函数 32 位整形参数高 16 位
    call F_sram628128_Read;                 //调用汇编读函数,读取的数据已经在 r1 中，
                                            //C 的返回值就是 r1 中的数值
    pop bp from [sp];                       //恢复现场
    pop r2,r3 from [sp];
```

```
        retf;
        .endp

//************************************************************
//访问 SRAM:HM628128 写函数
//汇编调用函数名：F_sram628128_Write
//参数参数传递：
//    输入：
//          r2 存放地址低 16 位
//          r3 存放地址高 1 位
//          r1 存放要写入的数据
//调用方法：
//          r2 =  地址低 16 位
//          r3 =  地址高 1 位
//          r1 =  要写入的数据
//          call F_sram628128_Write
//使用寄存器：r1,r2,r3
//************************************************************
.public F_sram628128_Write;
F_sram628128_Write:    .proc
        [P_IOB_Data] = r2;
        //地址高第 17 位设定
        r2 = [P_IOA_Data];                          //取 IOA 端口状态
        r2 = r2 lsl 1;                              //左移一位
        r3 = r3 ror 4;                              //把 r3.0 位移到 SB.0
        r2 = r2 ror 1;                              //SB.0 移到 r1.15
        [P_IOA_Data] = r2;                          //实现了位传送：r3.0=>[P_IOA_Data].15

        //设置片选有效、写有效
        r2 = [P_IOA_Data];
        r2 &= 0xafff;
        [P_IOA_Data] = r2;

        //数据线 IOA0~IOA7 设为输出
        r2 = [P_IOA_Dir];
        r2 |=0x00ff;
        [P_IOA_Dir] = r2;

        r2 = [P_IOA_Data];
        r2 = r2 lsr 4;
        r2 = r2 lsr 4;                              //r2 低 8 位移出
        r1 = r1 lsl 4;
        r1 = r1 lsl 4;                              //r1 高 8 位移出
        r1 = r1 rol 4;
```

```
    r2 = r2 rol 4;
    r1 = r1 rol 4;
    r2 = r2 rol 4;                          //实现了 r1 低 8 位到 r2 低 8 位的传送
    [P_IOA_Data] = r2;                      //要写入的数据送到数据线上

    //设置片选无效、读无效,IOA15 为高
    r2 = [P_IOA_Data];
    r2 |= 0x7000;                           //设置 IOA12、13、14 为输出高电平
    [P_IOA_Data] = r2;
    r2 = 0xffff;
    [P_IOB_Data] = r2;                      //地址线均为高
    r2 = [P_IOA_Data];                      //设置 IOA15 为高
    r2 |= 0x8000;
    [P_IOA_Data] = r2;
    //数据线 IOA0~IOA7 设为输入
    r2 = [P_IOA_Dir];
    r2 &= 0xff00;
    [P_IOA_Dir] = r2;

    retf;
    .endp

//************************************************************
//访问 SRAM:HM628128 写入函数
//C 语言调用函数函数原型：
//void F_sram628128_Read(unsigned int Data,unsigned long int Addr)
//输入参数：8 位宽度数据,17 位宽度地址
//返回值：8 位数据
//调用方法：
//          unsigned int Data;
//          unsigned long int Addr;
//          Data = 0xaa;
//          Addr = 0x0001abcd;
//          F_sram628128_Write(Data,Addr)
//使用寄存器：r1,r2,r3
//************************************************************
.public _F_sram628128_Write;
_F_sram628128_Write: .proc
    push r1,r3 to [sp];                     /保护现场
    push bp to [sp];                        //保护 bp 值
    bp = sp + 1;                            //调整 bp 指针指向堆栈栈顶数据
    //栈顶到栈低数据依次为:
//bp,r1,r2,r3,pc,程序段地址,C 函数第二参数 32 位整形参数低 16 位
//C 函数第二参数 32 位整形参数高 16 位,C 函数第一 16 位参数
```

```
    r1 = [bp+6];                    //取 C 函数第一个 16 位参数
    r2 = [bp+7];                    //取 C 函数第二个 32 位整形参数低 16 位
    r3 = [bp+8];                    //取 C 函数第二个 32 位整形参数高 16 位
    call F_sram628128_Write;        //调用汇编写函数
    pop bp from [sp];               //恢复现场
    pop r1,r3 from [sp];
    retf;
    .endp
```

下面是用 C 语言编写的一个测试程序，不断的往 sram 中写数据，然后再读出来，这样来验证读写得是否正确。程序如下：

文件名：main.c

```
int main(void){
    unsigned int i;                 //定义变量
    unsigned long int j;
    i = 0x00ff;                     //给初值.
    j = 0;
    F_sram628128_Init();            //调用初始化程序
    while(1){
        F_sram628128_Write(i,j);
        i = 0;
        i = F_sram628128_Read(j);
        //此处设断点，观察变量 i,j 的变化，以验证读写程序
        i--;
        j++;
    }
    return 0;
}
```

实验九 按键 DVR

【实验目的】

1)通过实验了解实验箱资源的基本使用方法。

2)了解凌阳音频、LCD 函数的调用以及 SIO 的使用方法。

【实验设备】

1)装有 WINDOWS 系统和 μ'nSP™ IDE 仿真环境的 PC 机一台。

2) μ'nSP™ 十六位单片机实验箱一个。

【实验原理】

通过按 4 乘 4 键盘的 0、1、2 键控制录音，停止，播放，同时 LCD 会有相关提示。

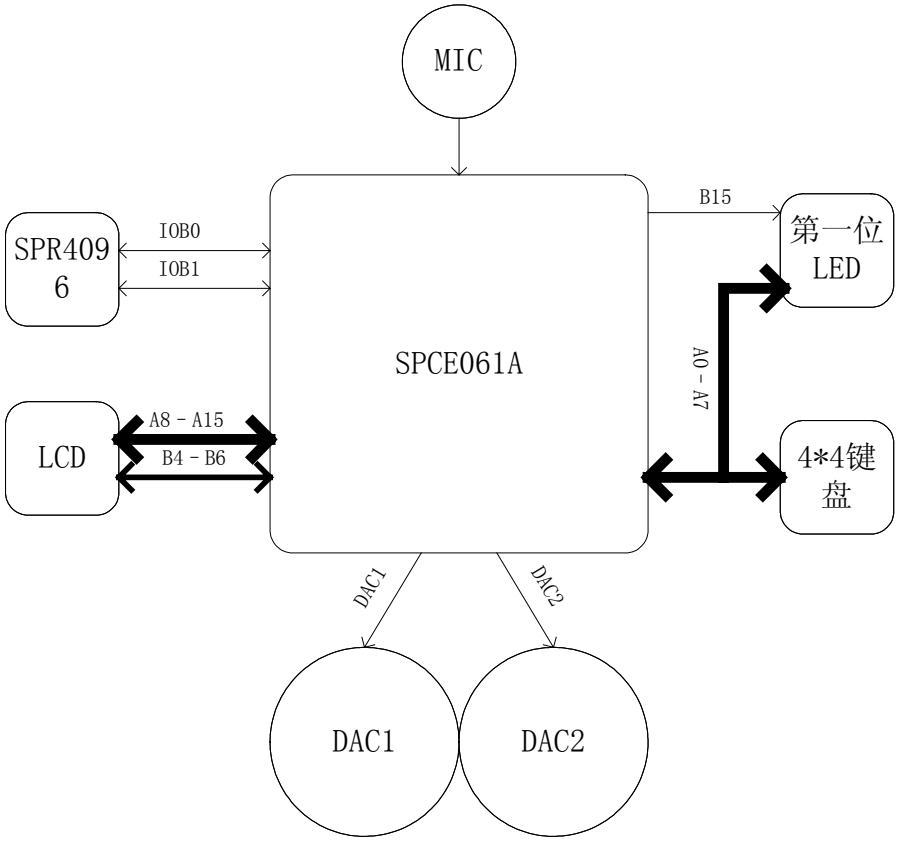
【实验步骤】

1)把 SIO,LCD, 4 乘 4 键盘以及数码管前四位的短结子连接好。

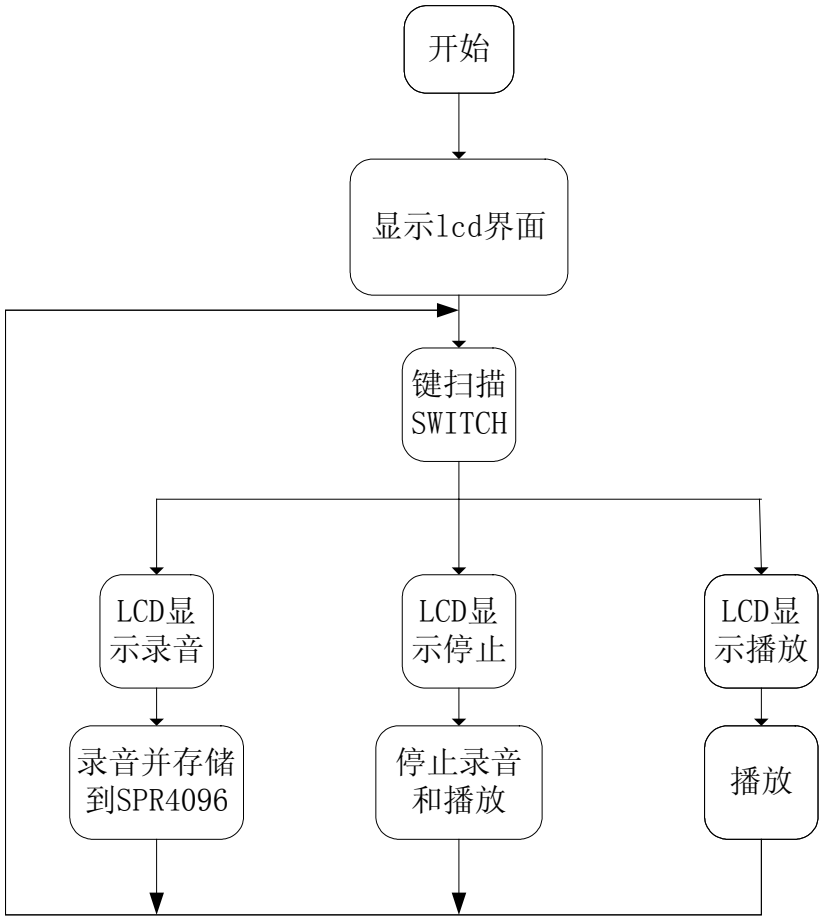
2)下载程序。

3)按键 0 进入录音状态，按键 1 停止，按键 2 开始播放。

【硬件连接图】


MIC
SPR4096
IOB0
IOB1
SPCE061A
B15
第一位LED
A0－A7
LCD
A8－A15
B4－B6
4*4键盘
DAC1
DAC2
DAC1
DAC2

【程序流程图】



【程序及其特殊函数说明】

本程序主要结合实验箱资源来完成一个语音录放的工作，无需连线，都采用实验箱上分配给各模块的 IO 资源。

1、Splc501c 库函数使用方法

FG_InitGraphic FG_ClearScreen
FG_GetBMPMode FG_SetBMPMode FG_PutBitmap
FG_GetLineStyle FG_SetLineStyle FG_LineTo
FG_GetRectStyle FG_SetRectStyle FG_ClearRect FG_Rectangle
FG_GetCircleStyle FG_SetCircleStyle FG_ClearCircle FG_Circle
FG_GetEllipseStyle FG_SetEllipseStyle FG_ClearEllipse FG_Ellipse
FG_MoveTo FG_MoveDelta
FG_GetX FG_GetY
FG_GetCharMode FG_SetCharMode
FG_GetCharWidth FG_GetCharHeight FG_PutChar
FG_PutStr FG_GetStrWidth
FG_PutPixel FG_GetPixel
FG_Bar

完成以上 34 个函数，函数功能基本包括 splc501c 初始化、绘图、放置图片、和完

成西文字符输入功能。本函数库，可作为图形、和文字输入部分程序底层程序加快项目进程。

程序 i/o 口的设定：

本程序需要八根数据线和三根控制线。线的连接可以在 splc501c_io.inc 中进行设定。设定限制数据线必须在一个 port 口的高八位或低八位上顺序 d0~d7 连接 port.0~port.7 或 port.8~port.15。控制线只需在一个 port 口次序可随机设定。

例如：

```
//=============================================================
//SPLC501C Library Pin Definition
//=============================================================
//6800 Control Pin
//-------------------------------------------------------------------------
//User can change    any of control pins at any bit, except user select the IO port
//of control pin the same as control bus.
//=============================================================
//                                FEDCBA9876543210
//.define         C_AOP_Pin          00100000000000b;        //IO AOP
//.define         C_EP_Pin           01000000000000b;        //IO EP
//.define         C_RWP_Pin          10000000000000b;        //IO as RWP
//                                FEDCBA9876543210
.define       C_AOP_Pin              0000000100000000b;          //IO AOP
.define       C_EP_Pin               0000001000000000b;          //IO EP
.define       C_RWP_Pin              0000010000000000b;          //IO as RWP
//=============================================================
//Define the SPLC Data Bus Pin
//Only Two Type Can Select
//=============================================================
//                            FEDCBA9876543210
.define       C_DataBus       1111111100000000b;          //IO Data Bus
.define       C_BusHighLow    1;              //Set Data High/Low Byte 1:High
//.define     C_AddressBus    0000000011111111b;          //IO Data Bus
//.define     C_DataBus       0000000011111111b;          //IO Data Bus
//.define     C_BusHighLow    0;              //Set Data High/Low Byte 0:Low
//=============================================================
//=============================================
//Set Control Pins Port
//=============================================
//Set Control Pins At Port A
.define  P_IO_Control_Data          0x7000;      //P_IOA_Data
.define  P_IO_Control_Buffer            0x7001;      //P_IOA_Buffer
.define  P_IO_Control_Dir           0x7002;      //P_IOA_Dir
.define  P_IO_Control_Attrib        0x7003;      //P_IOA_Attrib
//=============================================================
//Set Control Pins At Port B
```

```
//.define     P_IO_Control_Data       0x7005;     //P_IOB_Data
//.define     P_IO_Control_Buffer         0x7006;     //P_IOB_Buffer
//.define     P_IO_Control_Dir        0x7007;     //P_IOB_Dir
//.define     P_IO_Control_Attrib     0x7008;     //P_IOB_Attrib
//=====================================================
//=====================================================
//Set Control Bus Port
//=====================================================
//Set Control Bus At Port A
//.define     P_IO_Data_Data              0x7000;     //P_IOA_Data
//.define     P_IO_Data_Buffer            0x7001;     //P_IOA_Buffer
//.define     P_IO_Data_Dir               0x7002;     //P_IOA_Dir
//.define     P_IO_Data_Attrib            0x7003;     //P_IOA_Attrib
//=====================================================
//Set Control Bus At Port B
.define  P_IO_Data_Data         0x7005;     //P_IOB_Data
.define  P_IO_Data_Buffer       0x7006;     //P_IOB_Buffer
.define  P_IO_Data_Dir          0x7007;     //P_IOB_Dir
.define  P_IO_Data_Attrib       0x7008;     //P_IOB_Attrib
//=====================================================
```

以上设定数据口在 portb 口的高八位，控制线在 porta 口 AOP—porta.8、EP—porta.9、R/WP—porta.a。当然控制线和数据线可以在一个 port 口。

程序中参数的说明：

1. 位置参数是在一个字节范围内正整数，LCD 显示范围 x∈ [0,127] ， y∈[1,64]。
2. 关于 FG-Rectange、FG-Bar 函数参数是先输入右下角点，再是做上角点。如果次序错误，可能进入死循环。
3. 函数模式一般通过模式设定函数进行设置。有的函数有模式参数，此时模式的设定只在本函数范围内。如果函数运行完成，下一个需要模式的函数，如果未进行模式设定，那么这个函数的模式，由程序模式寄存器 R_GraphicMode 中的标志决定，和上一个函数中模式的设定无关。

例如：FG_PutBitmap(short BmpNum,short BmpX,short BmpY,short Mode)

1.FG_PutBitmap(short BmpNum,short BmpX,short BmpY,short Mode)

4. 如果模式参数错误，将无法正确设定模式。
5. 如果位置参数错误或超出范围，将无法预料结果。
6. 字体的宽度以在 lcd 上实际显示宽度为准。
7. 在有字的函数中字的范围可参照 tiny.bmp 中的字符。如果想输入字母无法字符输入可输入字母在 tiny.bmp 中的序号加 31。输入数字范围[32，227]。如果错误，可能进入死循环。

```
;=====================================================
```

FG_InitGraphic

调用方式： FG_InitGraphic ();

功能说明：初始化端口和 splc501c。

参数：

示　　例：

FG_InitGraphic()

;===

FG_ClearScreen

调用方式：FG_ClearScreen ()；

功能说明：清除或填充全屏。

参数： DG_CLS_ERASE(default), DG_CLS_FILL

示　　例

FG_ClearScreen (DG_CLS_FILL)

;===

FG_GetBMPMode

调用方式：FG_GetBMPMode ()；

功能说明：得到 bmp 图形绘制模式。

参数：

示　　例：

FG_GetBMPMode (index,x,y,mode)

;===

FG_SetBMPMode

调用方式： FG_SetBMPMode ()；

功能说明：设定 bmp 图形绘制模式。

参数：Mode。

Mode: DG_BMP_COVER(default)

DG_BMP_INVERSE

DG_BMP_XOR

Memory Modified: R_GraphicMode

用　法：

FG_SetBMPMode(short Mode)

示　　例：

FG_SetBMPMode(DG_BMP_INVERSE)

;===

FG_PutBitmap

调用方式： FG_PutBitmap ()；

功能说明：画 bmp 图。

参数：BmpX,BmpY,BmpNum,Mode

功能说明：画位图从 (BmpX,BmpY) 点，用指定模式

Mode:

DG_BMP_COVER(default)

DG_BMP_INVERSE

DG_BMP_XOR

Memory Modified: None

用　法：FG_PutBitmap(short BmpNum,short BmpX,short BmpY,short Mode)

示　　例：

FG_PutBitmap(0,10,20,DG_BMP_COVER)

FG_PutBitmap(1,10,20)

```
    FG_PutBitmap(2,DG_BMP_XOR)
    FG_PutBitmap(3)
```

;===

Function Name: FG_SetLineStyle

参数: Mode

功能说明: 设定现模式

Mode:

DG_LINE_COVER //(default)

DG_LINE_ERASE

DG_LINE_HOLLOW

DG_LINE_DOTTED

DG_LINE_HOLLOW_ERASE

DG_LINE_DOTTED_COVER

DG_LINE_SOLID_XOR

DG_LINE_DOTTED_XOR

Memory Modified: R_GraphicMode

用 法:

FG_SetLineStyle(short Mode)

示 例:

FG_SetLineStyle(DG_LINE_DOTTED_COVER)

;===

Function Name: FG_LineTo

参数: StartX,StartY,EndX,EndY

功能说明: 用指定模式（R_GraphicMode.）画线

Mode:

DG_LINE_COVER //(default)

DG_LINE_ERASE

DG_LINE_HOLLOW

DG_LINE_DOTTED

DG_LINE_HOLLOW_ERASE

DG_LINE_DOTTED_COVER

DG_LINE_SOLID_XOR

DG_LINE_DOTTED_XOR

Memory Modified: None

用 法:

FG_LineTo(short StartX,short StartY,short EndX,short EndY)

示 例:

FG_LineTo(0,0,20,20)

FG_LineTo(R_Var1,R_Var2)

;===

Function Name: FG_SetRectStyle

参数: Mode
功能说明: 设置矩形模式
Mode:
DG_RECT_COVER(default)
DG_RECT_ERASE
DG_RECT_HOLLOW
DG_RECT_DOTTED
DG_RECT_HOLLOW_ERASE
DG_RECT_DOTTED_COVER
DG_RECT_SOLID_XOR
DG_RECT_DOTTED_XOR

Memory Modified: R_GraphicMode
用 法:
FG_SetRectStyle(short Mode)
示 例:
FG_SetRectStyle(DG_RECT_DOTTED_COVER)

;==

Function Name: FG_ClearRect
参数: StartX,StartY,EndX,EndY,Mode
功能说明:清除或填充矩形区域
Mode: DG_CLEAR_ERASE(default),DG_CLEAR_FILL

Memory Modified: None
用 法: FG_ClearRect(short StartX,short StartY,short EndX,short EndY,short Mode)
示 例:
FG_ClearRect(0,0,20,20,DG_CLEAR_FILL)
FG_ClearRect(0,0,R_Var1,R_Var2)
FG_ClearRect(20,20,DG_CLEAR_ERASE)
FG_ClearRect(R_Var1,R_Var2)

;==
Function Name: FG_Rectangle
参数: StartX,StartY,EndX,EndY
功能说明: 用设定模式（R_GraphicMode.）画矩形
StartX,StartY 是右下脚点 EndX,EndY 是左上脚点
Memory Modified: None
用 法:
FG_Rectangle(short StartX,short StartY,short EndX,short EndY)
示 例:
FG_Rectangle(0,0,20,20)
FG_Rectangle(20,20)
FG_Rectangle(R_Var1,R_Var2,R_Var3,R_Var4)

FG_Rectangle(R_Var1,R_Var2)

;===

Function Name: FG_SetCircleStyle
参数: Mode
功能说明: 设定圆模式
Mode:
DG_CIRCLE_COVER(default)
DG_CIRCLE_ERASE
DG_CIRCLE_HOLLOW
DG_CIRCLE_DOTTED
DG_CIRCLE_HOLLOW_ERASE
DG_CIRCLE_DOTTED_COVER
DG_CIRCLE_SOLID_XOR
DG_CIRCLE_DOTTED_XOR

Memory Modified: R_GraphicMode
用　法:
FG_SetCircleStyle(short Mode)
示　　例:
FG_SetCircleStyle(DG_CIRCLE_DOTTED_COVER)

;===

Function Name: FG_ClearCircle
参数: OriginX, OriginY, Radius,Mode
功能说明:清除或填充圆的区域
Mode: DG_CLEAR_ERASE(default),DG_CLEAR_FILL

Memory Modified: None
用　法:
FG_ClearCircle(short OriginX,short OriginY,short Radius,short Mode)
示　　例:
FG_ClearCircle(20,20,DG_CLEAR_ERASE)
FG_ClearCircle(R_Var1,R_Var2)

;===
Function Name: FG_Circle
参数: OriginX, OriginY, Radius
功能说明: 用设定模式（R_GraphicMode.）画圆

Memory Modified: None
用　法:
FG_Circle(short OriginX,short OriginY,short Radius)

Mode:
DG_CIRCLE_COVER(default)
DG_CIRCLE_ERASE
DG_CIRCLE_HOLLOW
DG_CIRCLE_DOTTED
DG_CIRCLE_HOLLOW_ERASE
DG_CIRCLE_DOTTED_COVER
DG_CIRCLE_SOLID_XOR
DG_CIRCLE_DOTTED_XOR

示　　例:
FG_Circle(20,20,5)
FG_Circle(R_Var1,R_Var2,R_Var3)

;===

Function Name: FG_SetEllipseStyle
参数: Mode
功能说明: 设定椭圆模式
Mode:
DG_ELLIPSE_COVER(default)
DG_ELLIPSE_ERASE
DG_ELLIPSE_HOLLOW
DG_ELLIPSE_DOTTED
DG_ELLIPSE_HOLLOW_ERASE
DG_ELLIPSE_DOTTED_COVER
DG_ELLIPSE_SOLID_XOR
DG_ELLIPSE_DOTTED_XOR

Memory Modified: R_GraphicMode
用　法:
FG_SetEllipseStyle(short Mode)
示　　例:
FG_SetEllipseStyle(DG_ELLIPSE_DOTTED_COVER)

;===

Function Name: FG_ClearEllipse
参数: OriginX, OriginY, RadiusX, RadiusY,Mode
功能说明: 填充或清除椭圆区域
Mode: DG_CLEAR_ERASE(default),DG_CLEAR_FILL

Memory Modified: None
用　法:
FG_ClearEllipse(short OriginX,short OriginY,short RadiusX,short RadiusY,short Mode)

示　　例:
FG_ClearEllipse(20,20,5,4,DG_CLEAR_ERASE)
FG_ClearEllipse(R_Var1,R_Var2,R_Var1,R_Var2,R_Var3,R_Var4)

;===

Function Name: FG_Ellipse
参数: OriginX, OriginY, RadiusX, RadiusY
功能说明: 用设定模式（R_GraphicMode.）画椭圆

Memory Modified: None
用　法:
FG_Ellipse(short OriginX,short OriginY,short RadiusX,short RadiusY)
示　　例:
FG_Ellipse(20,20,5,4)
FG_Ellipse(R_Var1,R_Var2,R_Var3,R_Var4)

;===

Function Name: FG_MoveTo
参数: NewX,NewY
功能说明: Move (X0,Y0) to a new location.

Memory Modified: X0->NewX, Y0->NewY
用　法:　FG_MoveTo(short NewX,short NewY)
示　　例:
FG_MoveTo(20,30)
FG_MoveTo(R_Var1,R_Var2)

;===

Function Name: FG_MoveDelta
参数: DeltaX,DeltaY
功能说明: Mover (X0,Y0) to a new location. New X0=X0+DeltaX.

Memory Modified: X0->X0+DeltaX, Y0->Y0+DeltaY
用　法:　　FG_MoveDelta(short DeltaX,short DeltaY)
示　　例:FG_MoveDelta(10,10)

;===

Function Name: FG_GetX,FG_GetY
参数:
功能说明:得到当前点值存入 r1

Memory Modified: None
用　法:
FG_GetX()
FG_GetY()

;===

Function Name: FG_SetCharMode
参数: Mode
功能说明:设置字模式
Mode: DG_CHAR_COVER(default)
　　DG_CHAR_INVERSE
　　DG_CHAR_XOR

Memory Modified: R_GraphicMode
用　法:　　FG_SetCharStyle(short Mode)
示　　例:FG_SetCharStyle(DG_CHAR_XOR)

;===

Function Name: FG_GetCharWidth
参数: Char,Font
功能说明: 得到字符宽度存入 r1

Memory Modified: None
用　法:　FG_GetCharWidth(char Char,short Font)
示　　例:　FG_GetCharWidth('A',Tiny)

;===

Function Name: FG_GetCharHeight
参数: Char,Font
功能说明: 得到字符高度存入 r1

Memory Modified: None
用　法:　FG_GetCharHeight(char Char,short Font)
示　　例:　FG_GetCharHeight('a',Large)

;===

Function Name: FG_PutChar
参数: Char,Font,StartX,StartY
功能说明: 输出字符在 (StartX,StartY).
Mode: DG_CHAR_COVER(default)
　　DG_CHAR_INVERSE

DG_CHAR_XOR

Memory Modified: None
用　法:　FG_PutChar(char Char,short Font,short StartX,short StartY)
示　　例:
FG_PutChar(R_CharCode,Small,10,10)
FG_PutChar('A',Large,R_Var1,R_Var2)
FG_PutChar(14,Tiny)

;===

Function Name: FG_PutStr
参数: StrPtr,Font,StartX,StartY
功能说明: 输出字符串从 (StartX,StartY).
Mode:
DG_CHAR_COVER(default)
DG_CHAR_INVERSE
DG_CHAR_XOR

Memory Modified: None
用　法:　FG_PutStr(char *StrPtr,short Font,short StartX,short StartY)
示　　例:
FG_PutStr(StringPointer,Tiny,5,6)
FG_PutStr("Hello",Large)

;===

Function Name: FG_GetStrWidth
参数: StrPtr,Font
功能说明: 得到字符串宽度存入 r1.

Memory Modified: None
用　法:
FG_GetStrWidth(char *StrPtr,short Font)
示　　例:
FG_GetStrWidth(StringPointer,Tiny)

;===

Function Name: FG_PutPixel
功能说明: 画一个像素（R_GraphicMode.）
参数: PlotX,PlotY,Mode
Mode:
DG_PIXEL_COVER(default)
DG_PIXEL_ERASE

DG_PIXEL_XOR

Memory Modified: None
用　法: FG_PutPixel(short PlotX,short PlotY,short Mode)
示　　例:
FG_PutPixel(10,25)
FG_PutPixel(R_Var1,R_Var2,DG_PIXEL_ERASE)

;===

Function Name: FG_GetPixel
参数: PlotX,PlotY
功能说明:得到像素的值存入 R1

Memory Modified: None
用　法:　FG_GetPixel(short PlotX,short PlotY)
示　　例:
FG_GetPixel()
FG_GetPixel(10,20)

;===

Function Name: FG_Bar
参数: StartX,StartY,EndX,EndY,Mode
功能说明: 画一个栅条用（R_GraphicMode）.
Mode: DG_BAR_FILL(default),DG_BAR_ERASE,DG_BAR_INVERSE

Memory Modified: None
用　法:　FG_Bar(short StartX,short StartY,short EndX,short EndY,short Mode)
示　　例:
FG_Bar(0,0,20,20,DG_BAR_FILL)
FG_Bar(0,0,R_Var1,R_Var2)
FG_Bar(20,20,DG_BAR_ERASE)
FG_Bar(R_Var1,R_Var2)

;===

2、SPR4096 库函数使用说明

SPR4096 的操作函数在 SP_SerialFlashV1.asm 文件中，主要完成对 flash 的擦除和写，在这个 DVR 程序中所有数据都存入 SPR4096 中

实验十　带有背景音乐的动态图片

【实验目的】

1、 巩固 SPLC501 的使用

2、 学习图片显示与音乐的结合

【实验设备】

1)装有 u'nsp IDE 仿真环境的 PC 机一台

2)μ'nSP™十六位单片机实验箱一个

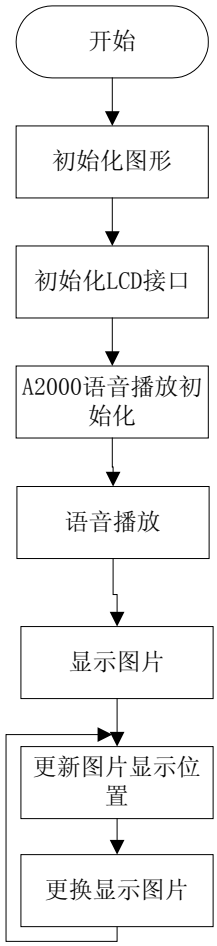
【实验原理】

采用 SPLC501 显示动态图片。

【硬件连接图】

同分立实验中，实验一。

【程序流程图】



【程序及其特殊函数说明】

FG_ClearScreen

调用方式：FG_ClearScreen ()；

功能说明：清除或填充全屏。

参数： DG_CLS_ERASE(default), DG_CLS_FILL

示　　例

FG_ClearScreen (DG_CLS_FILL)

FG_PutBitmap

调用方式： FG_PutBitmap ()；

功能说明：画 bmp 图。

参数： BmpX,BmpY,BmpNum,Mode

功能说明：画位图从 (BmpX,BmpY) 点，用指定模式

```
Mode:
    DG_BMP_COVER(default)
    DG_BMP_INVERSE
    DG_BMP_XOR
Memory Modified: None
用　法: FG_PutBitmap(short BmpNum,short BmpX,short BmpY,short Mode)
示　　例:
    FG_PutBitmap(0,10,20,DG_BMP_COVER)
    FG_PutBitmap(1,10,20)
    FG_PutBitmap(2,DG_BMP_XOR)
    FG_PutBitmap(3)
```

实验十一　UART 控制液晶显示

【实验目的】

1) 巩固 SIO 的基本使用方法。
2) 巩固 UART 的基本使用方法。
3）巩固 LCD 的基本使用方法。

【实验设备】

1)装有 WINDOWS 系统和 μ'nSP™ IDE 仿真环境的 PC 机一台。
2) μ'nSP™ 十六位单片机实验箱一个。
3)串口线一根
4)串口工具，如“串口调试助手 V2.1.exe”

【实验原理】

前面的实验已经分别讲过 SIO、UART 和 LCD 单独使用的情况，本实验是三者一起工作时的情况，读者会发现这跟搭积木一样简单。

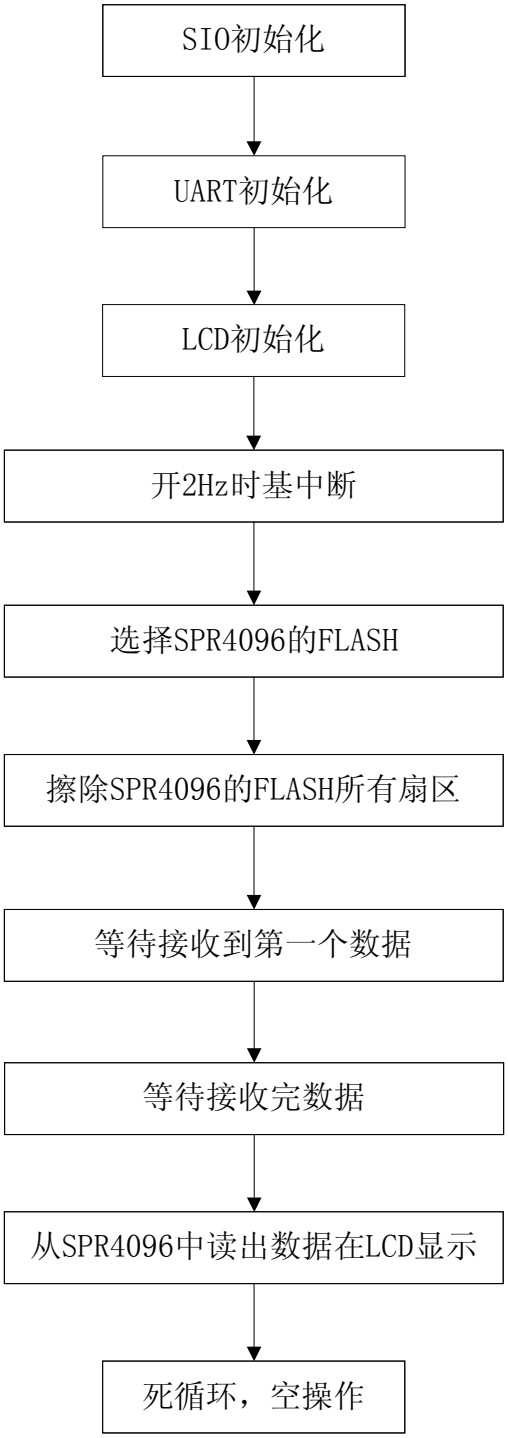
【实验步骤】

1）把 JP10 的三个短路线接好，注意 CF7 选择 B11。
2）JP6 全部接上，注意 CS1 和 B9 相连。
3）下载实验程序，观察现象。
4）当 LCD 显示“UART Ready”时，把串口工具的波特率设置为 9600，无检验位，8 位数据位，1 位结束位。
5）发送一个文件，注意文件只能包含字母、数字和标点符号。
6）LCD 显示“Receiving…”。
7）LCD 显示“Complete”。
6）LCD 显示接收到的内容。

【硬件连接图】

见前面的 SIO 实验、UART 实验、LCD 实验。

【程序流程图】



【程序及其特殊函数说明】

无。

实验十二　0---3V 电压测量表

【实验目的】

1)通过实验了解实验箱资源的基本使用方法。

2)了解凌阳音频、LCD 函数的调用以及 SPCE061A 内部 AD 的使用方法。

【实验设备】

1)装有 WINDOWS 系统和 μ'nSP™ IDE 仿真环境的 PC 机一台。

2) μ'nSP™ 十六位单片机实验箱一个。

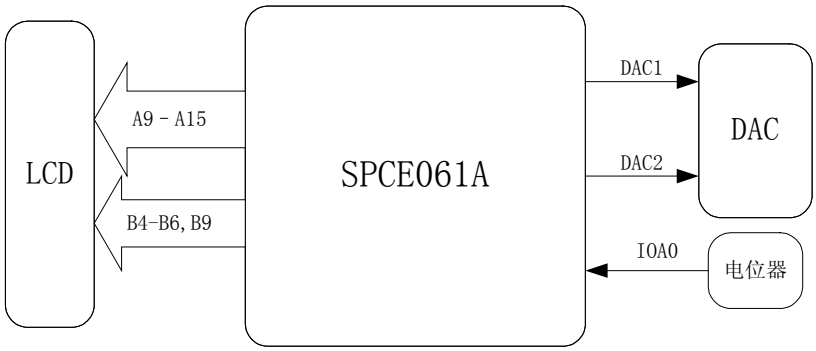
【实验原理】

通过 SPCE061A 内部 ADC 采集数据，把电压在 LCD 上显示出来，并通过 DAC 通道来提示当前电压。

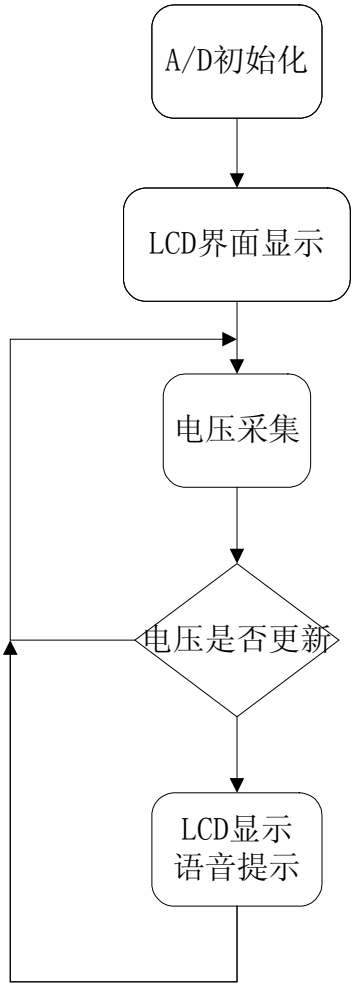
【实验步骤】

1) 把 LCD 的短结子连接好，注意片选的短结子接 B9 端。
2) J17 接 IOA1
3) 下载程序。
4) 调节电位器观察状态。

【硬件连接图】



【程序流程图】



【程序及其特殊函数说明】

本程序主要结合实验箱资源来完成一个语音电压表，无需连线，都采用实验箱上分配给各模块的 IO 资源。

实验十三　录音笔

【实验目的】

1) 巩固 SIO 的基本使用方法。
2) 巩固 LCD 的基本使用方法。

【实验设备】

1)装有 WINDOWS 系统和 μ'nSP™ IDE 仿真环境的 PC 机一台。
2) μ'nSP™ 十六位单片机实验箱一个。

【实验原理】

本实验实现了这样一个录音笔，其功能是用键盘控制语音的录放、存储和擦除。

1、录音键按下，开始录音
2、放音键按下，开始放音
3、停止键按下，停止录音或放音
4、下一段键按下，开始播放下一段
5、擦除键按下，开全部擦除

6、整个操作过程用 lcd 显示状态

【实验步骤】

1）把 JP10 的三个短路线接好，注意 CF7 选择 B11。

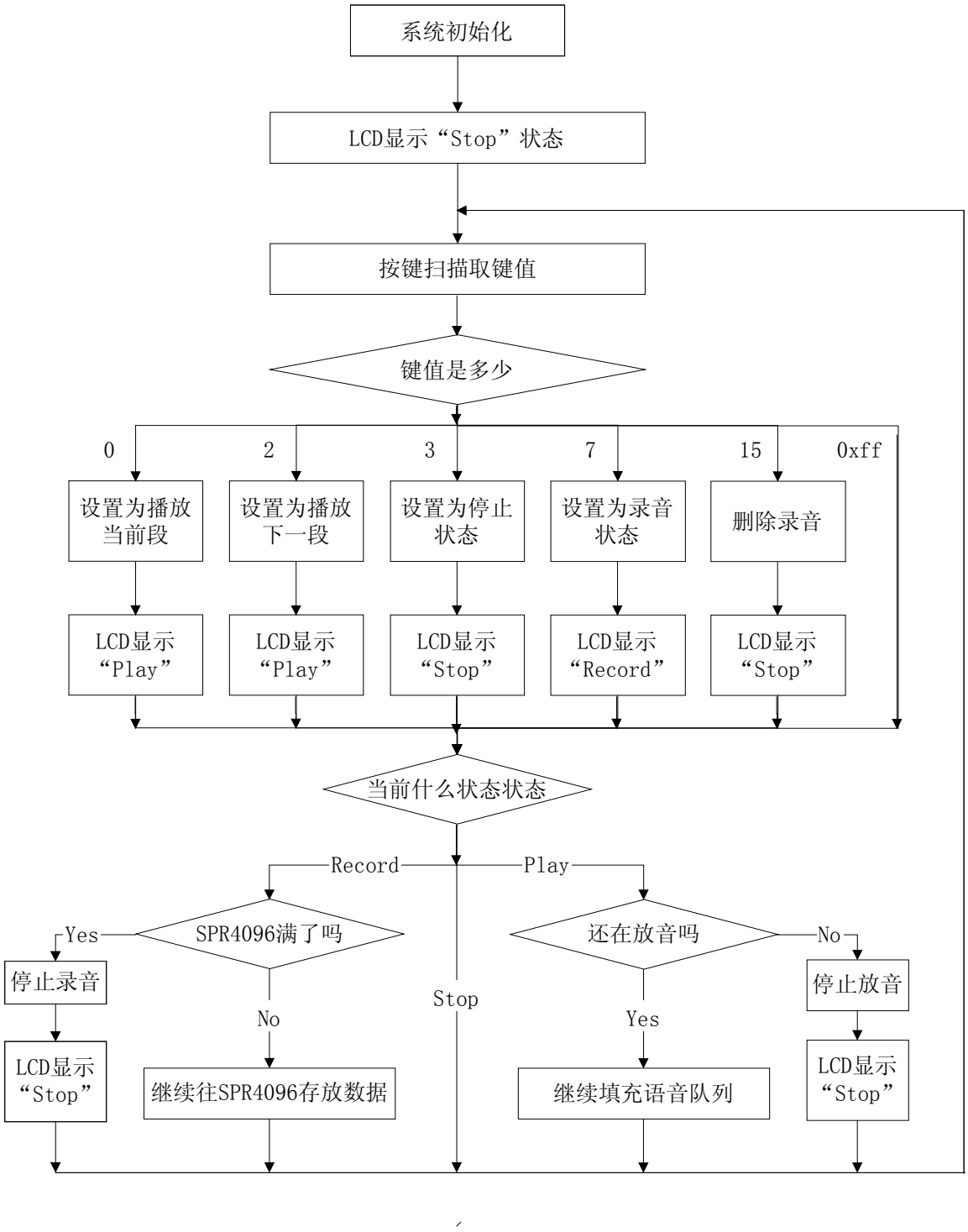
2）JP6 全部接上，注意 CS1 和 B9 相连。

3）下载实验程序，观察现象。

【硬件连接图】

见前面的 SIO 实验、LCD 实验。

【程序流程图】



【程序及其特殊函数说明】

无。

实验十四　USB 实现语音录放及其上传下载

【实验目的】

1、通过实验了解实验箱 USB 模块的硬件连接。

2、简单的 USB 通讯：实现文件的上传和下载,结合语音录放（DVR 实验）功能实现压缩后的语音通过 PC 机解码播放。
3、掌握编程中常用函数的使用。

【实验设备】

1）装有 WINDOWS 系统和 μ'nSP™ IDE 仿真环境的 PC 机一台。
2）μ'nSP™ 十六位单片机实验箱一个。

【实验原理】

1、用 USB 模组和 SPCE061A 最小系统实现 USB 简单通讯，并实现两点功能：
2、通过 USB 通讯，将 PC 端的二进制文件下载到外扩 flash 中，同时也可以实现文件的上传。
3、结合 DVR 将压缩后的文件通过 PC 端解码播放。

【实验步骤】

1. 接好硬件，包括三个按键（IOA0,IOA1,IOA2）、SPR4096 FLASH、USB 模组等
2. 将程序 example2 下载到 SPCE061A 中，如果未装驱动，按照提示安装驱动。
3. 等 USB 通讯指示灯亮后，运行 P C 端应用软件路径在：USB-EXAMPLE2－PC－PC－Release－Testfirm.exe，如下图所示：

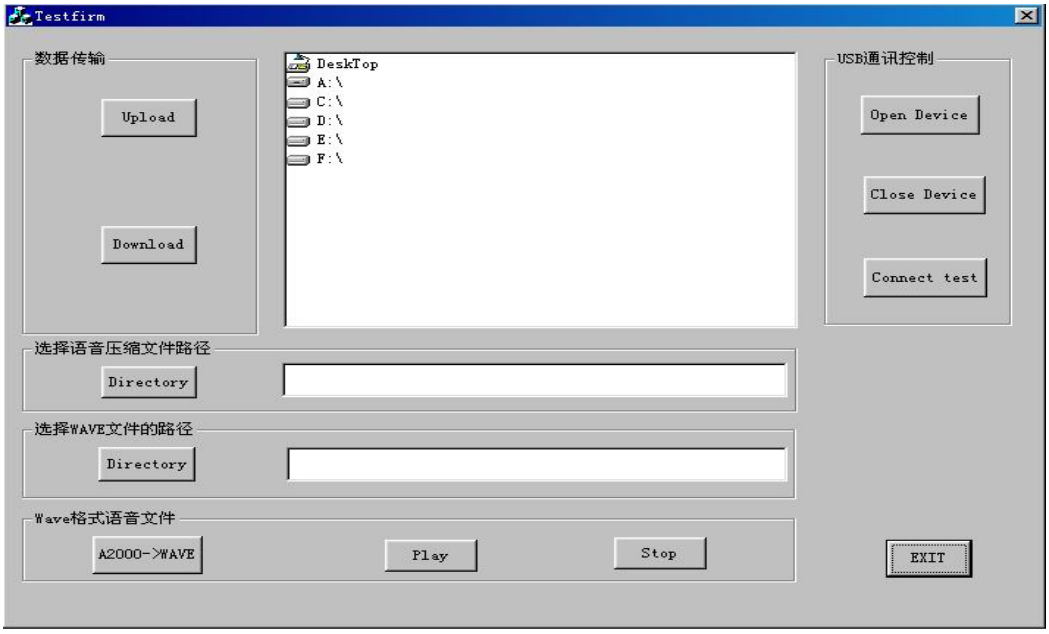


图2 PC 端应用软件的界面

4. 接着点击“Open Device”按钮，如果出现如下提示框，执行第 5 步操作，否则点击“确定”然后执行第 2 步操作，重新开始。

5. 点击“连接测试”按钮，如果出现如下提示框，执行第 6 步操作，否则点击“确定”然后执行第 2 步操作，重新开始。
6. 录、放音处理
   1） 按下“Record”键(IOA0)，进行录音
   2） 按下“Stop”键(IOA1)，停止录音
   3） 按下“Play”键(IOA2)，播放语音

7. 录音完音后，将语音数据上传到 PC 机
   1） 点击组合框“选择语音压缩文件路径”里的“Directory”按钮
   2） 点击组合框“数据传输”里的“Upload”按钮，开始从 FLASH 传输语音数据到 PC 机。当传输完毕，出现下图对话框。在数据传输的过程中，USB 模组上的 LED 灯会不停地闪烁。

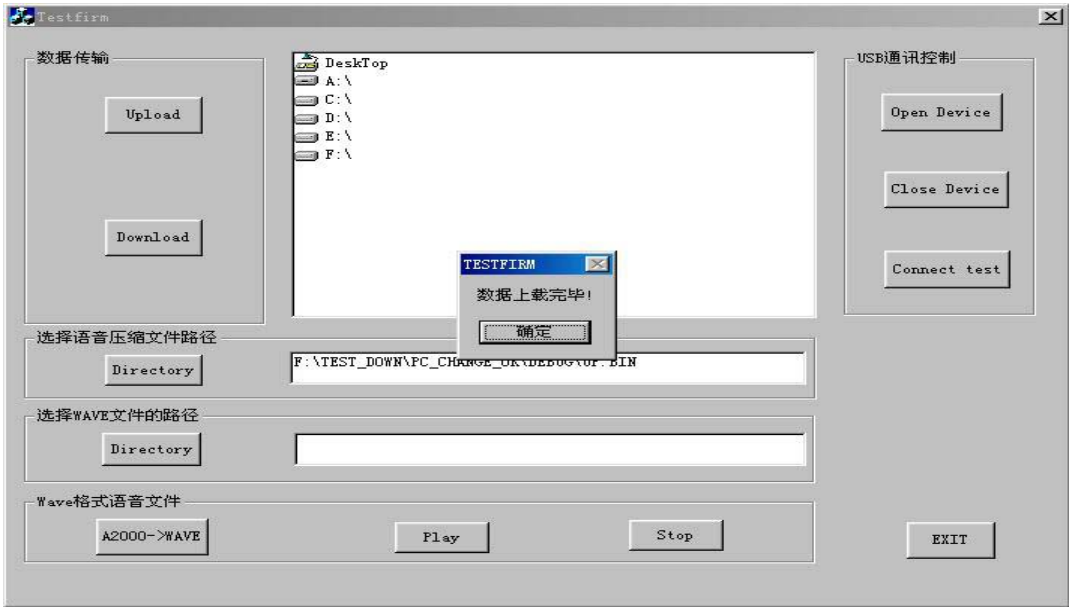


图3 上载数据

3） 点击确定，出现如图 12 中所示提示框。

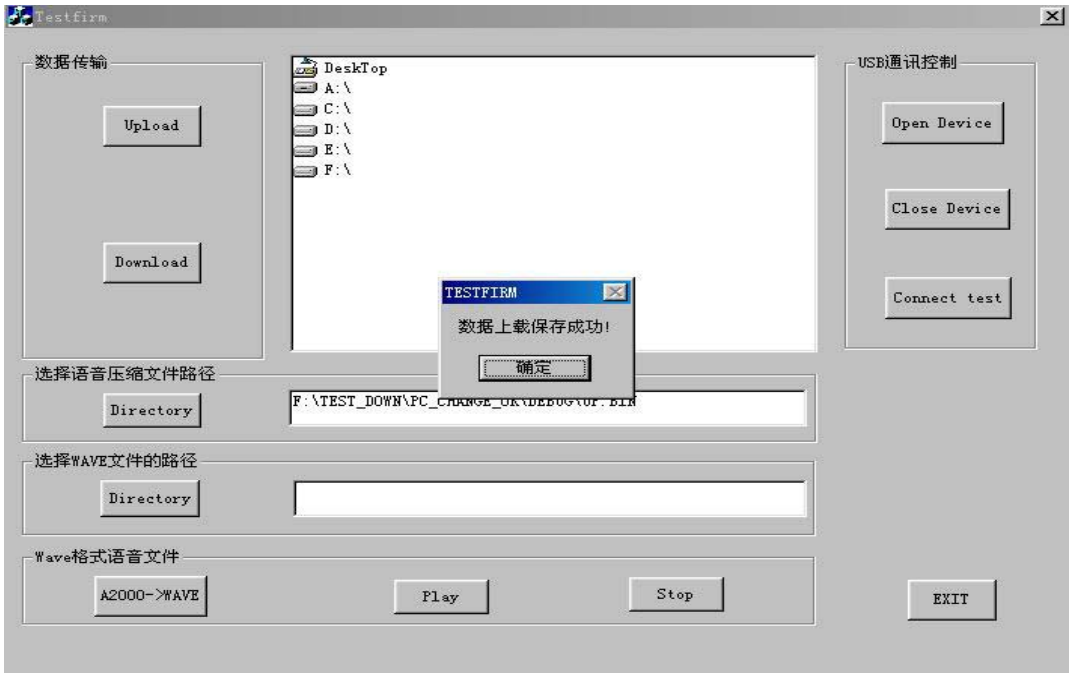


图4 数据上载保存

4） 点击确定，保存数据成功。

8. 将语音压缩后的文件解码转成 wave 格式的文件、播放试听

1） 选择语音压缩文件路径、选择 wave 文件的路径

2） 点击“A2000－>WAVE”的按钮，出现如图 13 中所示的提示框。

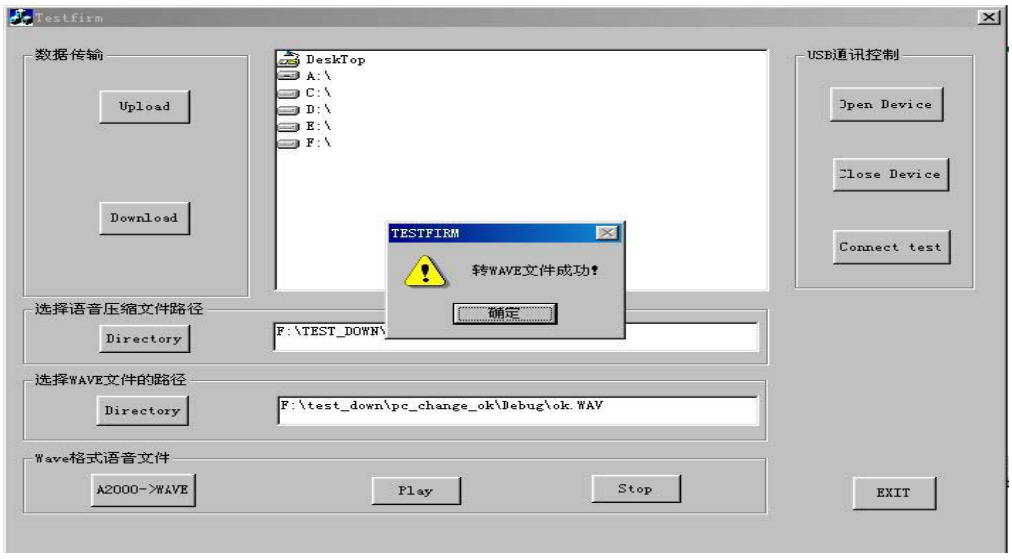


图5 A2000 转 Wave 文件

3） 点击“确定”按钮，再点击“Play”按钮，试听。

4） 可随时点击“Stop”按钮，取消播放。

9. 下载语音数据到 FLASH

1） 选择语音压缩文件路径

2） 点击“Download”按钮，出现如图 14 中所示的提示框，表明读取文件成功。

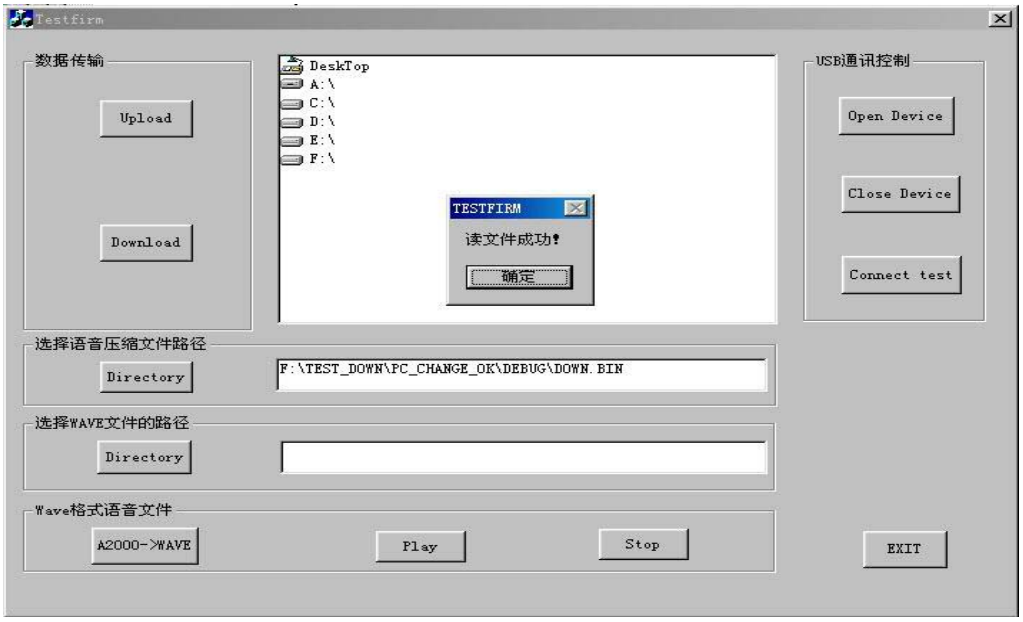


图6 读取文件

3） 点击“确定”按钮，开始下载数据，LED 灯不停闪烁，下载完毕后，出现如图 15 中所示的提示框，表明文件下载成功。

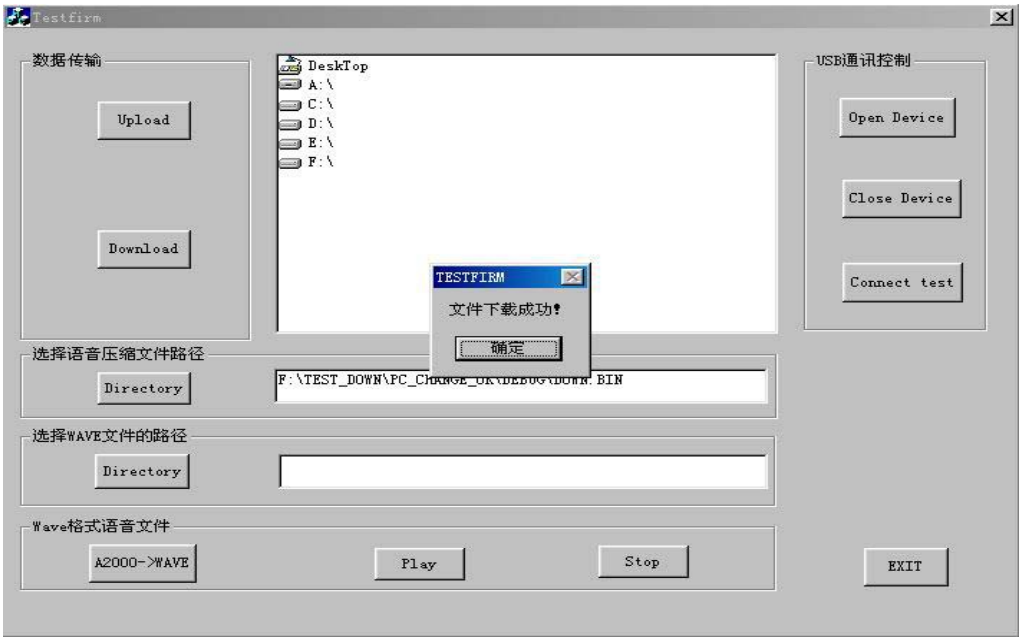


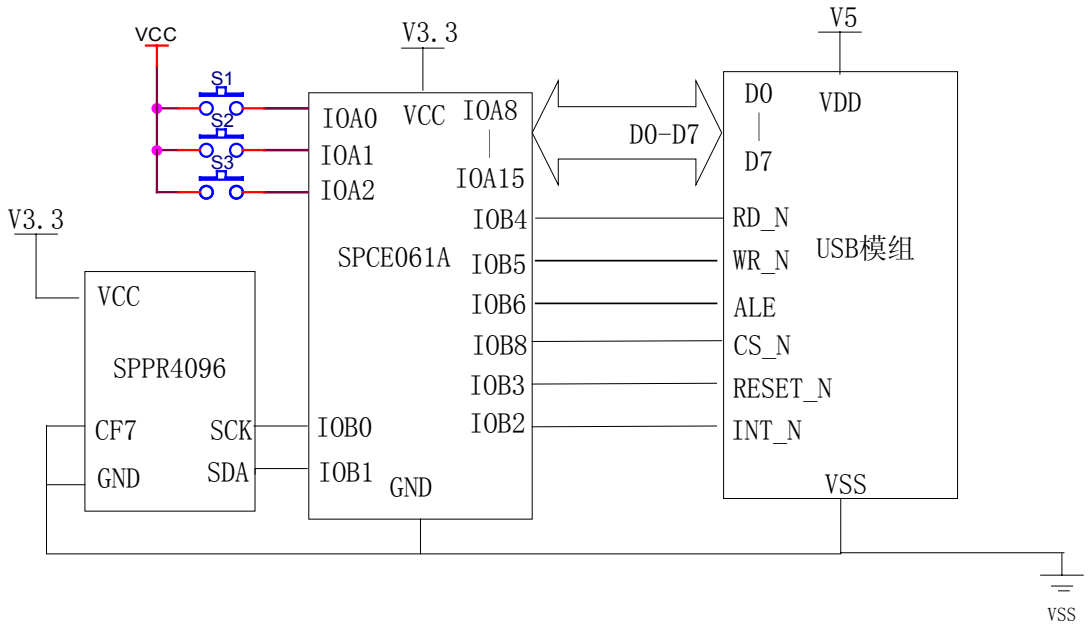
图7 下载语音文件

点击“确定”按钮，然后再按“Play”键进行单片机 SPCE061A 上播放语音

【硬件连接图】

3、 模组原理图（参见分立实验中，USB 通讯）

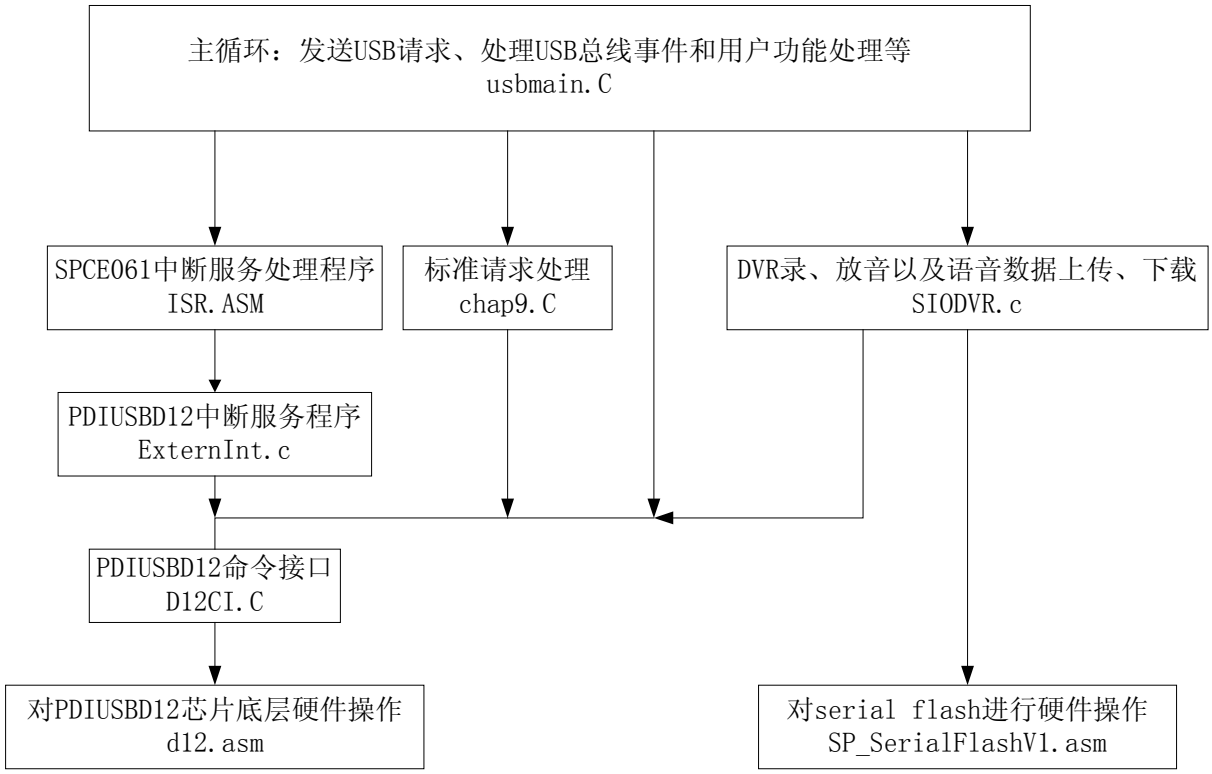
4、 USB、SPR4096 和 SPCE061A 连接图



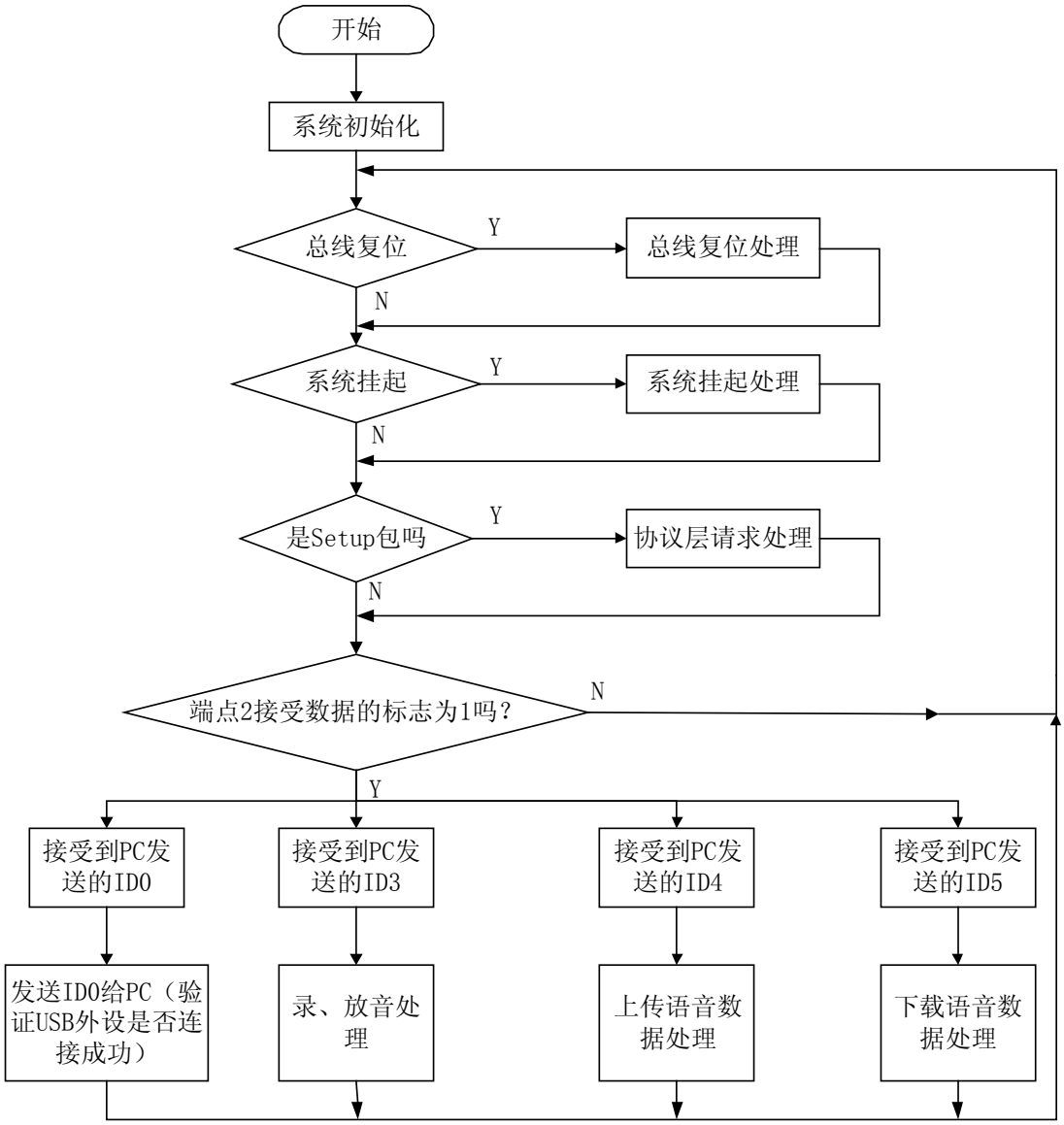
【程序流程图】

固件设计介绍：

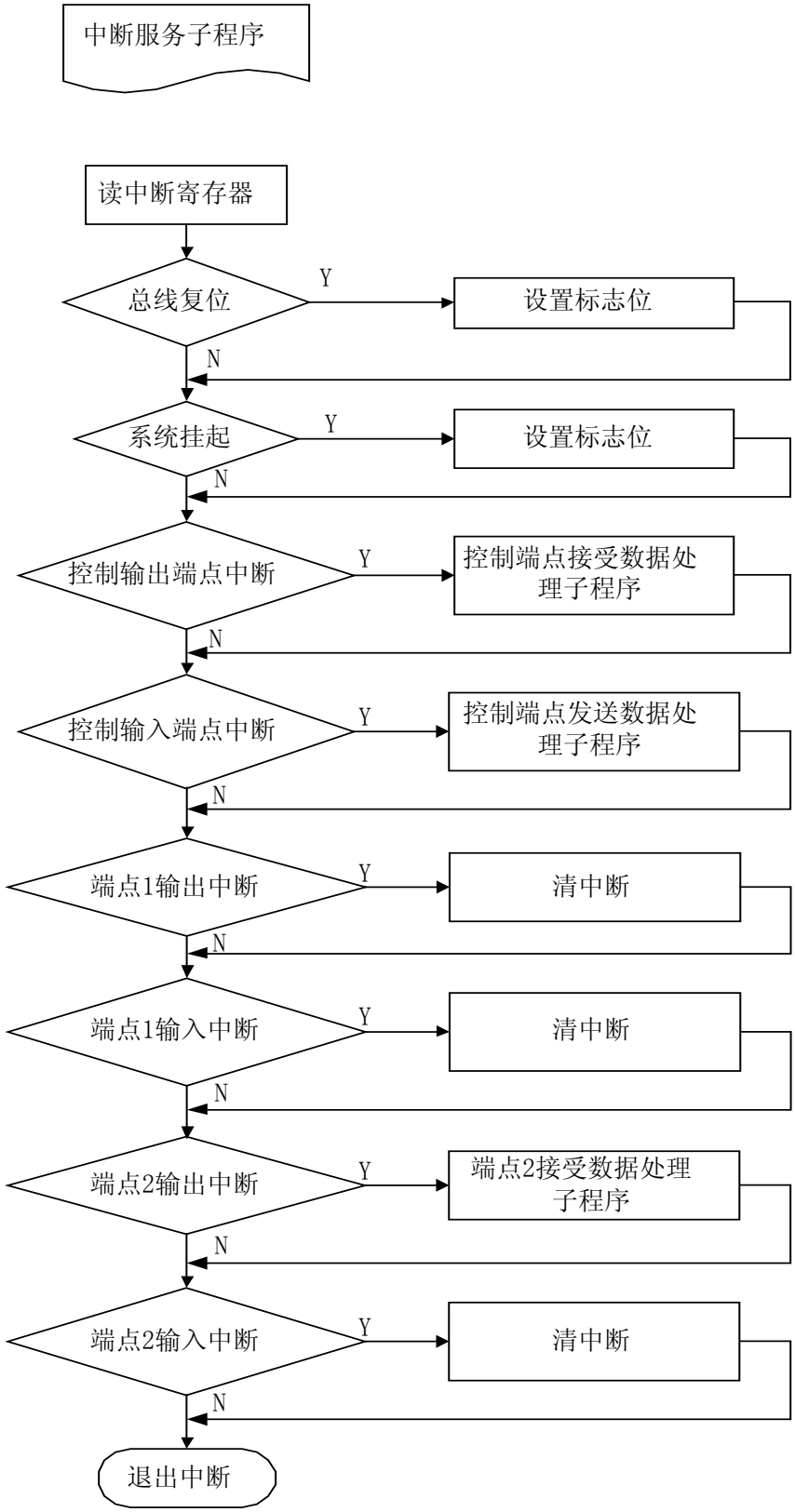
1、固件结构和数据流向



2、主程序流程：


开始
系统初始化
总线复位
Y
总线复位处理
N
系统挂起
Y
系统挂起处理
N
是Setup包吗
Y
协议层请求处理
N
端点2接受数据的标志为1吗?
N
Y
接受到PC发送的ID0
接受到PC发送的ID3
接受到PC发送的ID4
接受到PC发送的ID5
发送ID0给PC（验证USB外设是否连接成功）
录、放音处理
上传语音数据处理
下载语音数据处理

3、中断流程



【程序及其特殊函数说明】

1、源文件程序清单：

chap9.c：处理 USB 协议的相关代码，用于 PC 机枚举 USB 外设时用

D12.asm：处理 D12 芯片的读、写操作
ExternInt.c：处理 D12 的中断
D12CI.c：处理 PDIUSBD12 芯片的控制命令
SIODVR.C 主要处理 DVR 程序，包括录、放音程序、以及语音数据的上传、下载程序。
SP_SerialFlashV1.asm：Flash 读写等操作接口
SystemInitial.asm：系统初始化

2、常用函数介绍：

2.1Void F_USB_Isr（void）

该函数为处理分析 D12 芯片的中断源，主要是设置相应 D12 的中断源标志，用户只要知道有以下四种中断源标志位即可。

Ep1_ReceiveDataFlag：该标志位为 1，表示 PC 主机向 MCU 发送数据，即 pc 数据已经发送到端点 1 的 buffer，等待 MCU 读取。

Ep1_SendDataFlag： 该标志位为 1，表示 PC 主机请求 MCU 发送数据到 D12 端点 1 的 Buffer。

Ep2_ReceiveDataFlag：该标志位为 1，表示 PC 主机向 MCU 发送数据，即 pc 数据已经发送到端点 2 的 buffer，等待 MCU 读取。

Ep2_SendDataFlag： 该标志位为 1，表示 PC 主机请求 MCU 发送数据到 D12 端点 1 的 Buffer。

2.2 用户对 D12 的操作主要有以下四个函数：

1） unsigned int F_D12_ReadLastTransactionStatus(unsigned int bEndp);
【参　数】端点号，取值范围为 0－5
【返回值】参见 PDIUSBD12 用户手册，来源：www.unsp.com.cn
【功　能】清 D12 的端点中断源

2） unsigned int F_D12_ReadEndpoint(unsigned int endp, unsigned int len, unsigned int * buf);
【参　数】1.端点号 2.数据长度 3. 数据缓冲区的地址
【返回值】读取到的数据实际长度。
【功　能】读 D12 中 Buffer 的数据，该函数要与 F_D12_ReadLastTransactionStatus（）函数配合使用，建议使用 F_D12_ReadEndpointAndClrD12Int（）函数。

3） unsigned int F_D12_WriteEndpoint(unsigned int endp, unsigned int len, unsigned int * buf);
【参　数】1.端点号 2.数据长度 3. 数据缓冲区的地址
【返回值】写入缓冲区的实际数据长度。
【功　能】写数据到 D12 的 Buffer

4） unsigned int F_D12_ReadEndpointAndClrD12Int(unsigned int endp, unsigned int len, unsigned int * buf);
【参　数】1.端点号 2.数据长度 3. 数据缓冲区的地址
【返回值】读取到的数据实际长度。
【功　能】读 D12 中 Buffer 的数据，该函数比 F_D12_ReadEndpoint（）多一个清中断的操作。

3、注意事项

当 pc 端程序执行 Readfile（）的时候，MCU 只有使能 D12 的端点 Buffer 的时候，才会产生中断。当标志位 Ep1_SendDataFlag 或 Ep2_SendDataFlag 为 1 时， EasyUSB11.lib 中已经清中断了，用户不需再清中断。

主程序：

```
main()
{
    int i=0;
```

```
unsigned int Ret=0;
System_Initial0();
for(i=0;i<64;i++)
{
   MainEpBuf[i]=0;                          // 数组清 0
};

reconnect_USB();                            // 断开－延时－连接
SP_InitWriteD12();                          // D12 写初始化
Interrupt_On();                             // 开中断
while( TRUE )
{

   if (bEPPflags.bits.bus_reset)            //总线复位处理
   {
        bEPPflags.bits.bus_reset = 0;       //清标志
   }
   if (bEPPflags.bits.suspend)              //总线挂起处理
   {
        bEPPflags.bits.suspend= 0;          //清标志
   }
   if (bEPPflags.bits.setup_packet)         //协议处理
   {
        bEPPflags.bits.setup_packet = 0;    //清标志
        control_handler();
   }

   Key = SP_GetCh();
   if(Key==1||Key==2||Key==4||KeyStatus!=Stop)
   {
        Ret=0;
        IsoDvrHandle();
        Ret=0;
   }
   if(bEPPflags.bits.ep2_rxdone==1)
   {
   bEPPflags.bits.ep2_rxdone = 0;
      Ret= D12_ReadEndpoint(4,64,MainEpBuf);
      //FileLen_Block 为缓冲区中有效数据的长度
   if(MainEpBuf[0]==ID0)                    //ID0=1    for test
   {
        D12_WriteEndpoint(5,1,MainEpBuf);  //将 ID0 送到 PC
   }
```

```
        if(MainEpBuf[0]==ID3)//ID3=4    for dvr
        {

        }
        if(MainEpBuf[0]==ID4)                    //ID4=5     UPLOAD
        {
            if(MainEpBuf[1]==1)
            {
                Interrupt_Off_True();
                FileLength[0]=5;//APPLY ID
                FileLength[1]=(unsigned int)(0x000000FF&SpeechFileLength);
                FileLength[2]=(unsigned int)((0x0000FF00&SpeechFileLength)>>8);
                FileLength[3]=(unsigned int)((0x00FF0000&SpeechFileLength)>>16);
                FileLength[4]=(unsigned int)((0xFF000000&SpeechFileLength)>>24);
                Addr_UpLoad=0;
                Up_Flash_Addr_Up=0x0000;
                FlashAdd_Up=0;
                D12_WriteEndpoint(5,5,FileLength);   //将 FileLength 送到 PC
                G_UpLoad_Flag=1;
                Interrupt_On_True();

            }
        }
        if(MainEpBuf[0]==ID5)                    //ID5=6    DOWNLOAD
        {

            if(MainEpBuf[1]==1)
            {
                Interrupt_Off_True();            // 开中断
                SP_SIOMassErase();
                SpeechFileLength=256*MainEpBuf[5]+MainEpBuf[4];
                SpeechFileLength=256*(SpeechFileLength)+MainEpBuf[3];
                SpeechFileLength=256*(SpeechFileLength)+MainEpBuf[2];
                DownLoadFileLen_Up=0x0000;
                Down_Flash_Addr_Up=0x0000;
                FlashAdd_Up=0;
                D12_WriteEndpoint(5,1,MainEpBuf);
                m_CtrlRecFileData=1;
                Interrupt_On_True();             // 关中断
            }
        }    //end if download
      }      //end if ep2_rxdone==1
    } // Main Loop
}
```

实验十五　原始语音资源的存储和播放

【实验目的】

1、 熟悉原始语音的录放方法

2、 体会凌阳语音的压缩及播放效果

【实验设备】

1)装有 WINDOWS 系统和 μ'nSP™ IDE 仿真环境的 PC 机一台。

2) μ'nSP™ 十六位单片机实验箱一个。

【实验原理】

利用 SPR4096 存储 MIC 采样来的数据，播放时，直接从 SPR4096 中读出，送语音输出通道播放。主要采用两个按键实现语音的播放和录制。

【实验步骤】

1、根据硬件连接图连接

2、SPR4096 为实验箱默认连接

3、开始软件涉及

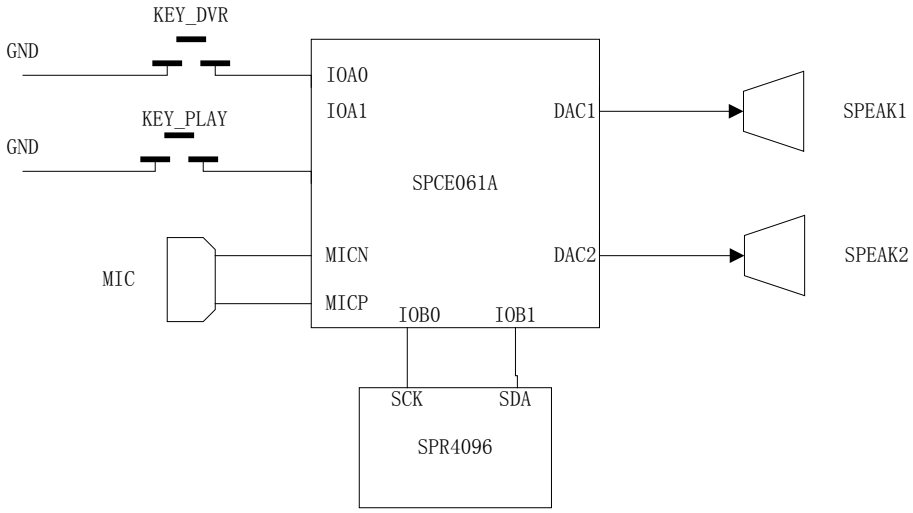
4、联机调试

5、按一键录音，

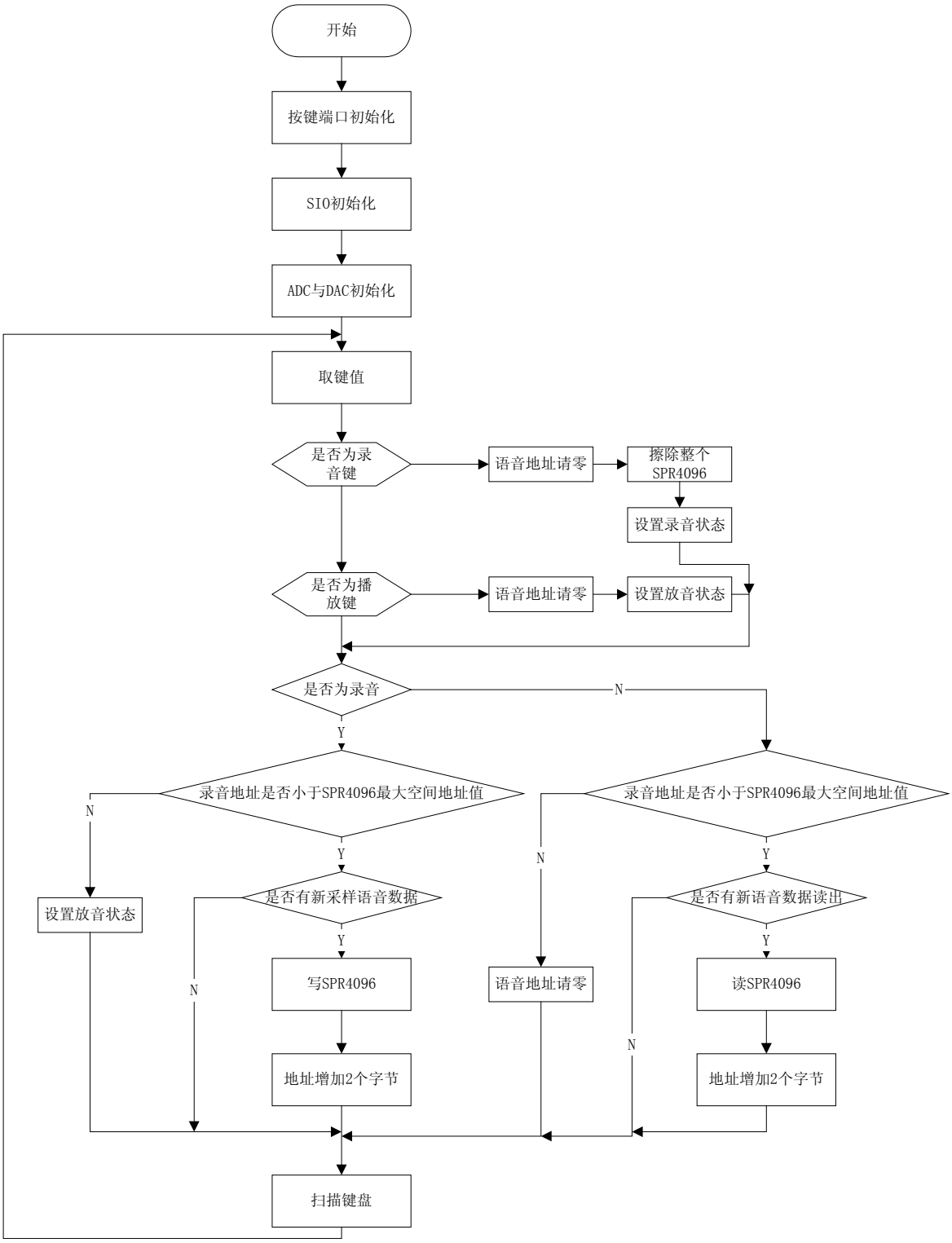
6、按二键播放录入声音

7、仔细听播放的声音效果

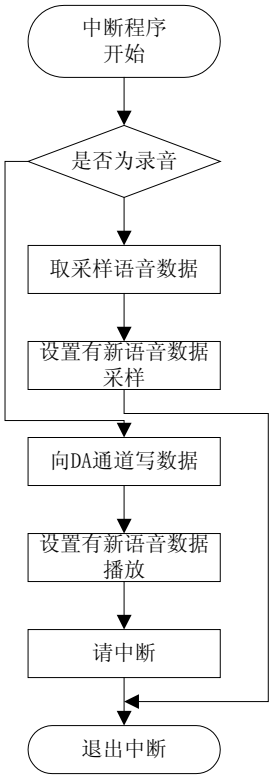
【硬件连接图】



【程序流程图】



中断程序流程图



【程序及其特殊函数说明】

SPR4096 库函数使用说明

SPR4096 的操作函数在 SP_SerialFlashV1.asm 文件中，主要完成对 flash 的擦除和写，在这个 DVR 程序中所有数据都存入 SPR4096 中