第10章 综合应用实例

前面我们介绍了 SPCE061A 的结构、C 语言的基本语法以及使用 C 语言开发一些简单的单片机应用程序。本章将结合一个用 SPCE061A 实现简易录音笔的软硬件设计实例，详细介绍开发单片机应用系统程序的过程。

10.1 开发背景

数码录音笔适用于外语学习、课堂记录、会议记录、谈判、研讨、访问、市场调查、速记、采访、备忘的诸多领域。

目前，数码录音笔的功能种类百花齐放，百家争鸣。它们具有的功能一般有：闹钟功能、定时录音功能、重复播放功能、播放速度控制功能、覆盖和附加录音功能，资料的上传和下载功能。

此外，有些数码录音笔还具有其他特殊功能，如视像功能，MP3 功能，录音监听功能，添加目录功能等，不一而足。

因为本章的目的是介绍开发单片机应用系统程序的过程，不是真正的开发一个产品，所以，本章实现的录音笔的功能只有：录音、播放、循环播放、重复播放、选择上一段播放、选择下一段播放、上传数据到 PC、从 PC 下载数据的功能。

10.2 硬件设计

从硬件上来说，本系统涉及到电源管理、音频录入、键盘管理，外扩存储器、与 PC 通信的接口、音频输出几部分。结构框图见图 10.1。

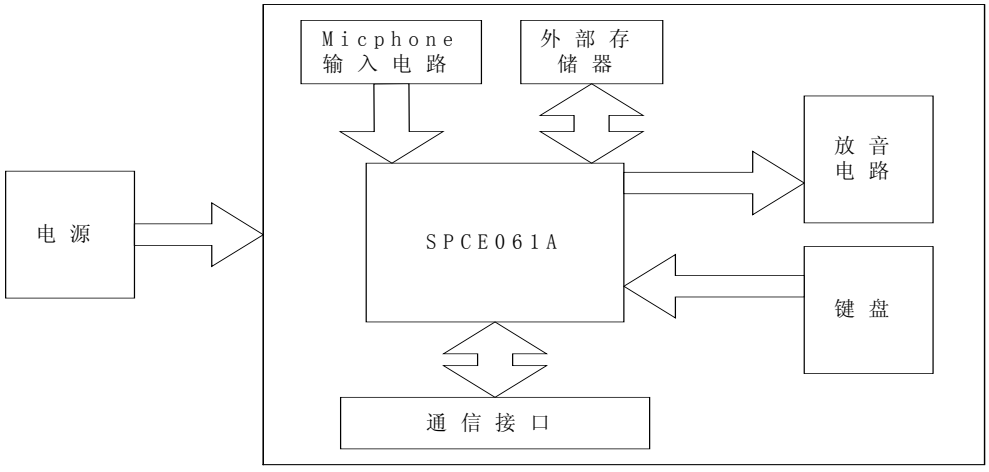

图10.1 结构框图

SPCE061A 是该系统的核心，主要作用是运算和控制周边电路。以下我对各部分电路作个简要介绍。

10.2.2 电源部分

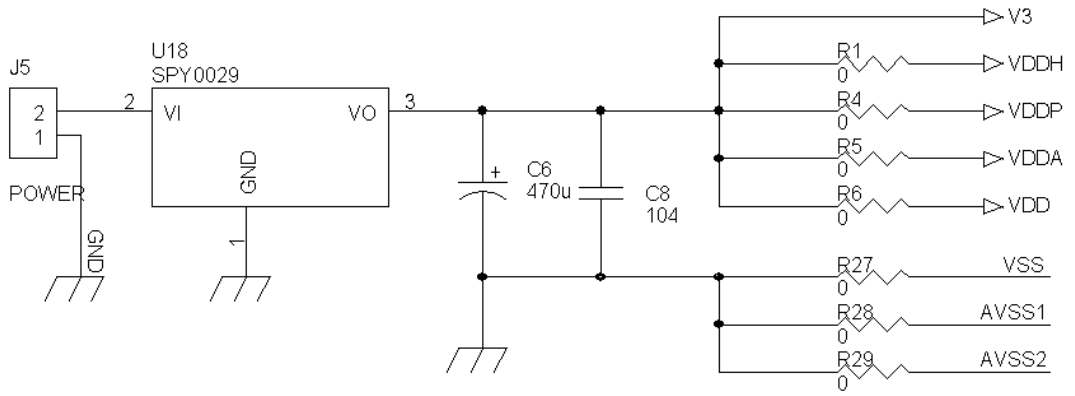

图10.2 电源电路

图 10.2 是电源部分的电路，5V 直流电压经过 SPY0029 后给整个系统供电。

SPY0029 是凌阳公司设计的电压调整 IC，采用 CMOS 工艺。SPY0029 具有静态电流低、驱动能力强、线性调整出色等特点。

图中的 VDDH 为 SPCE061A 的 I/O 电平参考，接 SPCE061A 的 51 脚，这种接法使得 I/O 输入输出高电平为 3.3V ;VDDP 为 PLL 锁相环电源，接 SPCE061A 的 7 脚；VDD 和 VDDA 为数字电源，分别接 SPCE061A 的 15 脚和 36 脚；AVSS1 是模拟地，接 SPCE061A 的 24 脚；VSS 是数字地，接 SPCE061A 的 38 脚； AVSS2 接音频输出电路的 AVSS2。

10.2.3 音频录入部分

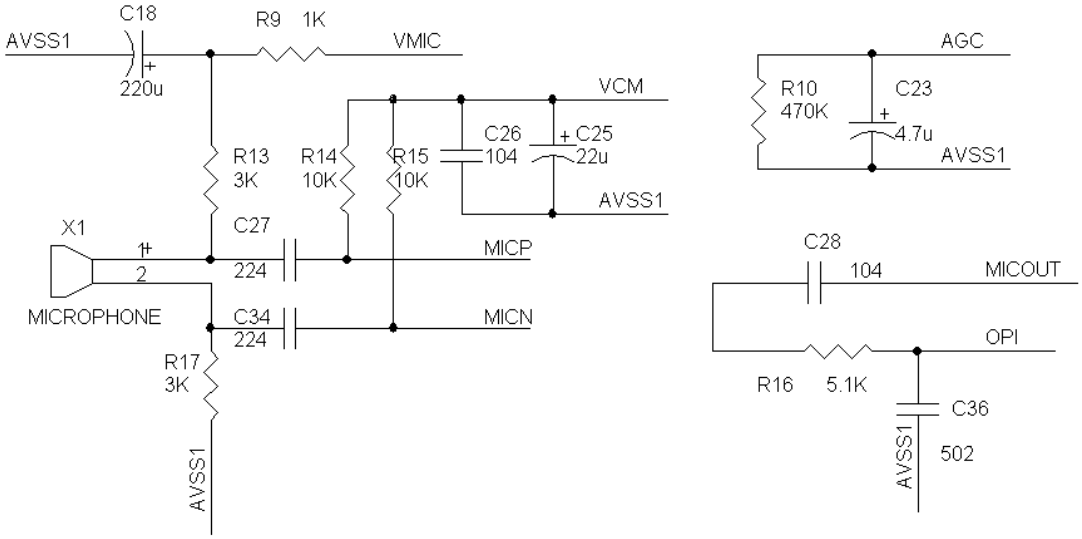

图10.3 音频录入电路

音频录入部分：主要由 Microphone、AGC 电路、ADC 电路构成。

图 10.3 是音频录入电路。因为 SPCE061A 内置了 AGC 电路和 ADC 电路，所以实现音频录入的外围电路变得如此简单。这部分电路与 SPCE061A 的连接是这样的：AGC

接音频录入 AGC 引脚（25 脚），OPI 接 Microphone 的第二运放输入脚（26 脚），MICOUT 接 Microphone 的第一运放输出脚（27 脚），MICN 接 Microphone 的负向输入脚（28 脚），MICP 接 Microphone 正向输入脚（33 脚），VCM 接 ADC 参考电压输出脚（34 脚），VMIC 接 Microphone 电源（37 脚）。

语音信号经 Microphone 转换成电信号，由隔直电容隔掉直流成分，然后输入至 SPCE061A 内部前置放大器。SPCE061A 内部自动增益控制电路 AGC 能随时跟踪、监视前置放大器输出的音频信号电平，当输入信号增大时，AGC 电路自动减小放大器的增益；当输入信号减小时，AGC 电路自动增大放大器的增益，以便使进入 A/D 的信号保持在最佳电平，又可使削波减至最小。

10.2.4 键盘部分

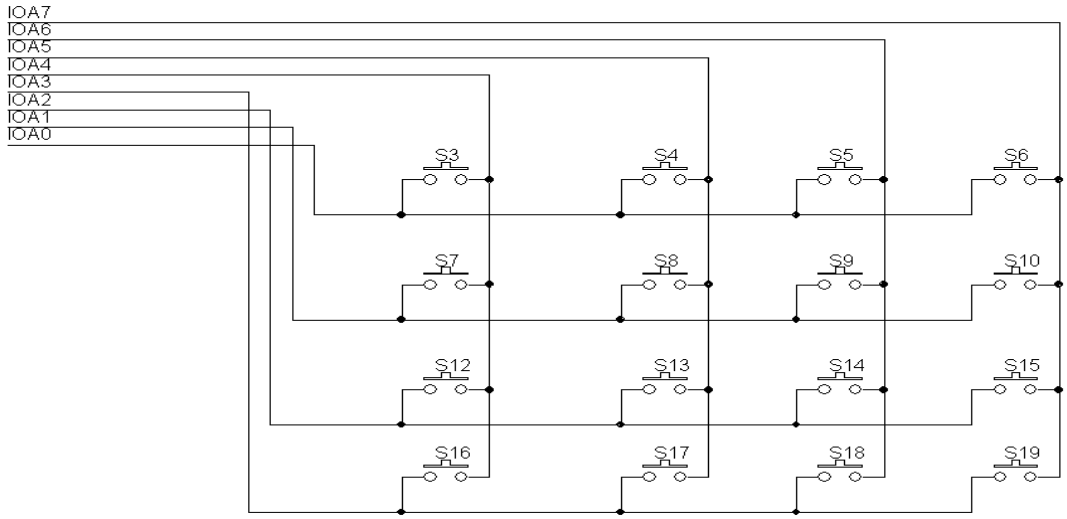

图10.4 4×4 键盘接口电路

图 10.4 是 4×4 键盘的电路。这里用了 A 口的低八位，扩展了 16 个按键，按键的键值分布和对应功能如图 10.5 所示。实际上，我们用了 11 个按键，剩余 5 个可用于其它扩展。

0　播放	4　循环播放	8	12
1　播放前段/上传	5　重复播放	9	13
2　播放后段/下载	6	10	14
3　停止/格式化	7　录音	11	15　开/关 UART

图10.5 键值分布及其意义

10.2.5 外扩存储器部分

因为 SPCE061A 的 FLASH 只有 32Kwords，要存放大量的语音资源，就要外扩存储器。本实例介绍一种用 SIO 扩展串行存储器的方法。本实例采用凌阳公司的 SPR4096 芯片。限于篇幅以及本书的重点所在，这里仅选中与我们设计程序有关的部分进行介绍，详细说明可以参见凌阳公司的 SPR4096 data sheet。

SPR4096 是一个高性能的 4M-bit（512K×8-bit）总线 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。在进行 FLASH 的编程/擦除的时候，可以并发执行 SRAM 的读/写。

SPR4096 内置了一个总线存储器接口和一个串行接口，它允许单片机通过 8-bit 并行模式或者 1-bit 的串行模式访问 FLASH/SRAM 存储区。本例使用串行模式，串行接口的工作频率为 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。最大读电流为 2mA，最大编程/擦除电流为 6mA。

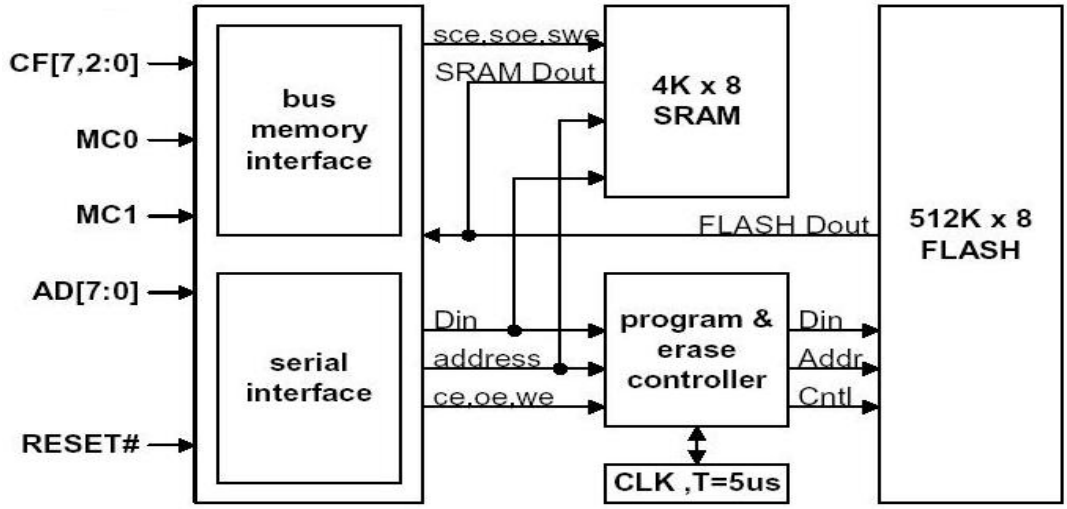

图10.6 SPR4096 模块结构图

图 10.6 是 SPR4096 的模块结构图。它包含了几个部分：总线存储器接口，串行接口，SRAM，编程和擦除控制器和一个 4M 的 FLASH。选择串行接口时，SCLK 作为时钟信号线，SDA 作为 1-bit 的数据线。如果接收到 FLASH 的读指令或者 SRAM 的读/写指令，串行接口可以直接从内存读写数据。但是，如果接收到 FLASH 的编程或擦除指令，串行接口会把这些指令传给编程和擦除控制器，让编程和擦除控制器去完成相应操作。

串行接口模式的选中是通过设置 CF2～CF0 来实现的。当 CF2～CF0 均接高电平时，选中的就是串行接口模式。在串行接口模式下，CF7 为低电平时选中 FLASH，高电平时选中 SRAM。

图 10.7 是 SPR4096 和 SPCE061A 的连接图。SPR4096 的 SCLK 接 SPCE061A 的

IOB0，SDA 接 SPCE061A 的 IOB1。CF0～CF2 接高电平，选择串行接口模式；CF7 接低电平，选中 FLASH。

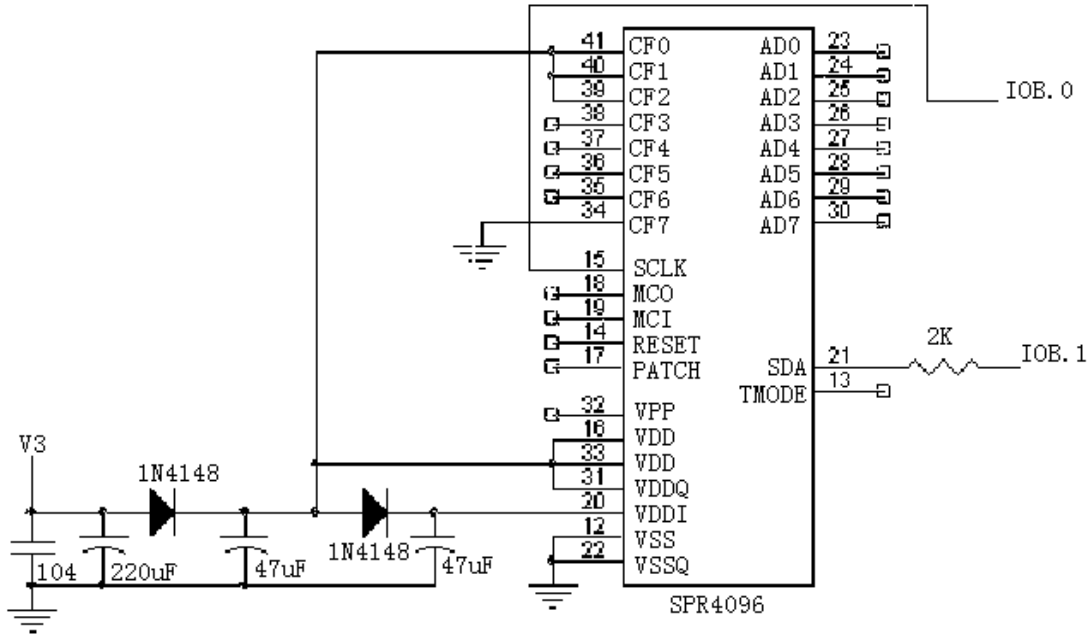

图10.7 外扩存储器电路

10.2.6 通信接口部分

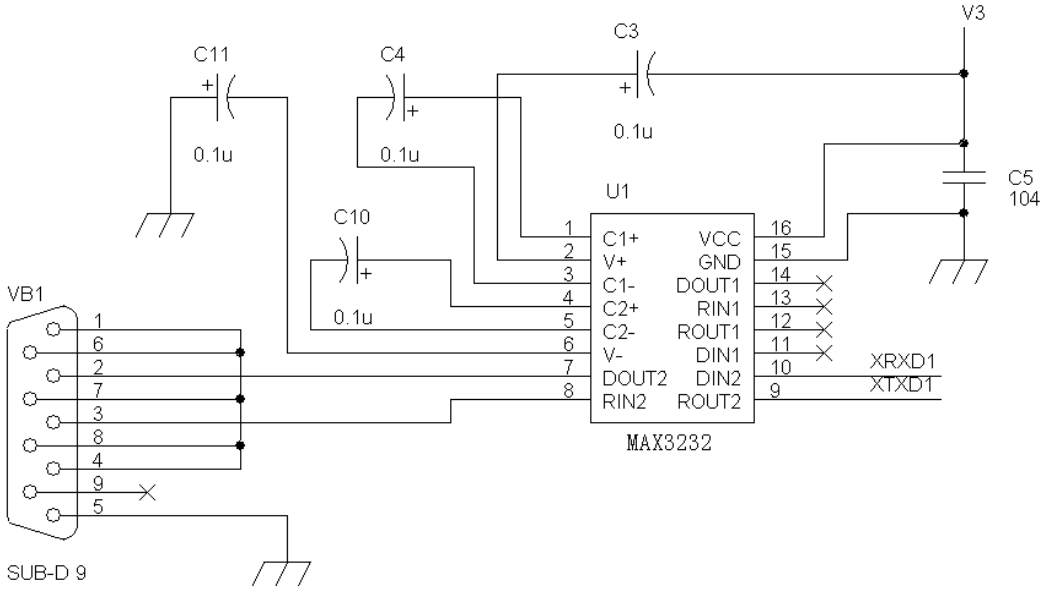

图10.8 和 PC 通信接口电路

本例使用 SPCE061A 的异步串行接口 UART 来实现与 PC 的通信。SPCE061A 的 I/O 电平和 PC 不一致，需要进行电平转换，这里用了一片 HIN232 来解决这个问题。电路图如图 10.8 所示。HIN232 的 XRXD1 和 XTXD1 分别接 SPCE061A 的 IOB7 和

IOB10。

10.2.7 音频输出部分

SPCE061A 内置 2 路 10 位精度的 DAC，只需要外接功放电路即可完成语音的播放。图 10.9 是音频输出电路图。可以接耳机，也可以直接听喇叭输出的声音。

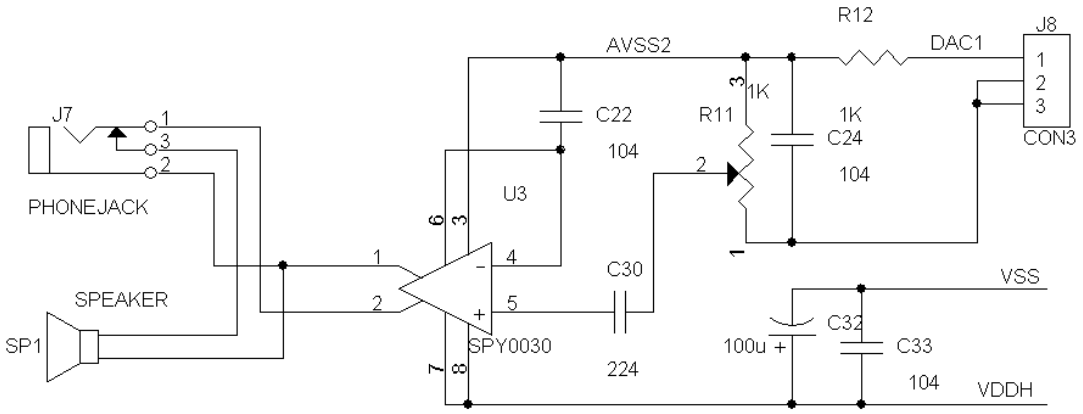

图10.9 音频输出电路

图中的 SPY0030 也是凌阳公司的产品。它的详细说明可以参见凌阳公司的 SPY0030 data sheet。和 LM386 相比，SPY0030 还是比较有优势的，比如 LM386 工作电压需在 4V 以上，SPY0030 仅需 2.4V 即可工作(两颗电池即可工作)；LM386 输出功率 100mW 以下，SPY0030 约 700mW。其他特性请参考 data sheet。

10.3 软件设计

同硬件设计一样，软件设计也是分块进行的。主要包括以下部分的程序：系统初始化程序、SPR4096 的读写程序、内部 FLASH 的读写程序、键盘扫描程序、录音放音程序、UART 通讯程序。各部分程序由主程序（main.c）调用，组成一个整体。

10.3.1 主程序

我们首先来看看主程序，图 10.10 是它的流程图。

系统初始化之后，就是一个死循环。在这个循环里的运作可以分为三部分：键盘扫描、根据键值设置子状态、根据子状态作相应的操作。

在程序中定义了两个状态：基状态（Status_Base）和子状态（Status_Sub）。基状态包括 UART 开启和 UART 关闭两种情况；子状态包括录音、播放、循环播放、重复播放、下载、上传、停止 7 种情况。

系统初始化部分包括硬件初始化和给变量的初始化，以下为系统初始化部分的代码。

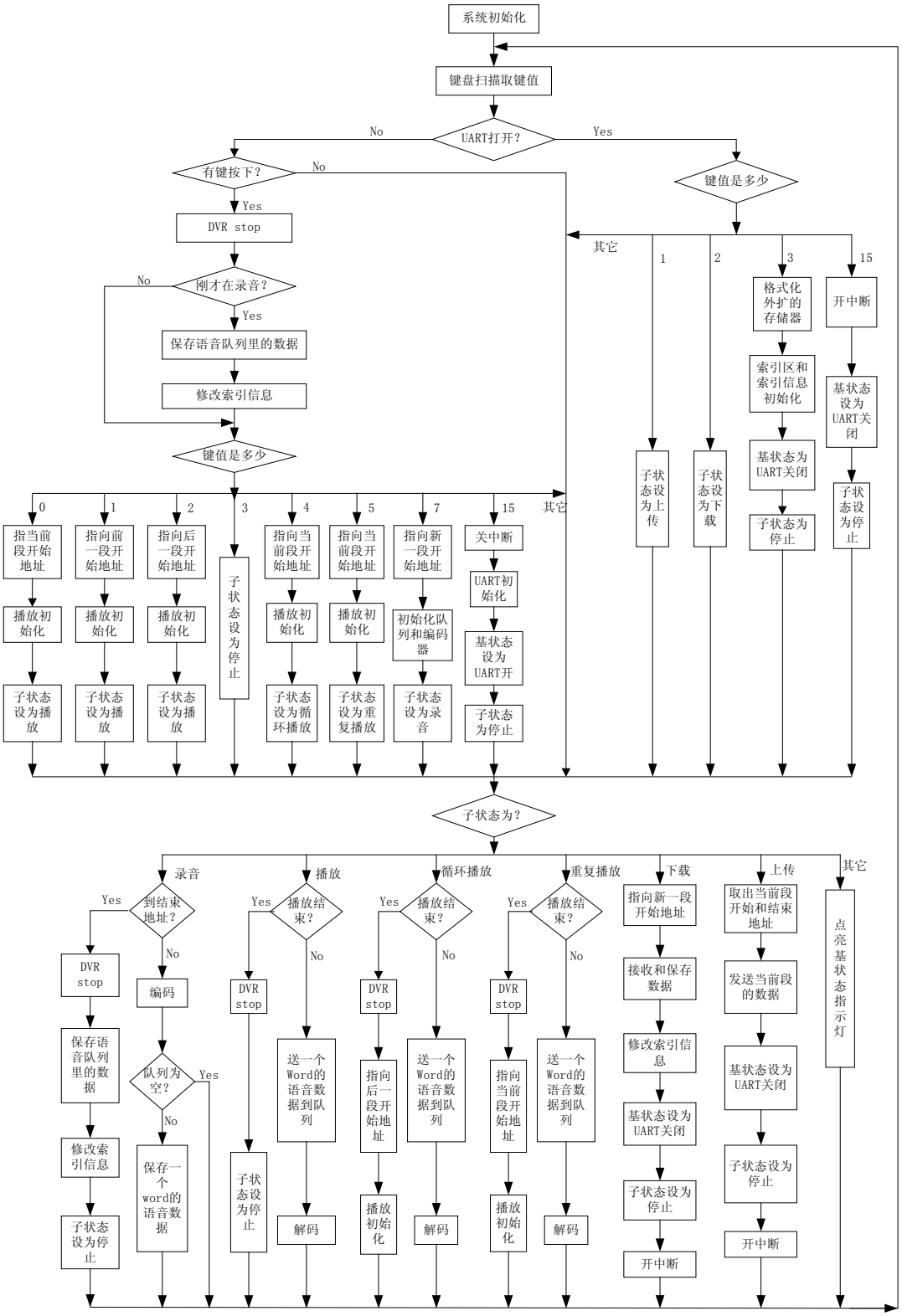

图10.10 主程序流程图

```
*************************************************************************
系统初始化部分代码
－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－
        BaseType = UartClose;                   //variable intitial
        SubStatus = Stop;
        Segment = 1;
        Addr = 0;
        Keycode = 0xff;                         //no key
        for(i=0,internal_addr=0xf000;i<21;i++,internal_addr++)
            Buffer[i] = SP_ReadWord(internal_addr);//read information from index
        System_Initial();                       //System Initial（setup IO and interrupt）
        SACM_DVR_Initial(Manual);               //DVR Initial as manual mode
*************************************************************************
```

索引区是 SPCE061A 内部 FLASH 从 0xf000 开始的 21 个 Word。0xf000 存放的是 SegFlag，记录的是总共有多少段数据；以下每两个 Word 记录某段的开始地址和结束地址，比如 0xf001 和 0xf002 存放的是第一段的开始和结束地址，0xf003 和 0xf004 存放的是第二段的开始和结束地址。这里的索引区只有 21 个 Word，所以这能保存 10 段语音数据，如果你要记录 100 段数据的话，把索引区改为 201 个 Word 就可以了。这里用到了函数 SP_ReadWord()，它的功能是从读内部 FLASH 中读出一个 Word 的数据，具体代码见 10.3.3 内部 FLASH 的读写程序部分。

System_Initial()是硬件部分的初始化函数，10.3.2 是对它的专门介绍。

以下部分是根据键值作相应设置的代码。对照图 10.10，很容易读懂的。

```
*************************************************************************
        if(BaseType == UartClose)                       //BaseType = UartClose
        {
            if(Keycode!=0xff)                           //if it has key pressed
            {
                DVR_Stop();
                if(SubStatus == Record)
                {
                    Send_QueueData_to_Sflash();
                    Modify_Index();
                }

                switch(Keycode)
                {
                    case 0:                             //Play current segment
                        Addr=Buffer[Segment*2-1];
                        Play_Initial();
                        SubStatus = Play;
                    break;
```

```
case 1:                                  //Play previous segment
    if((Segment>1)&&(Segment<=Buffer[0]))
    {
        Segment--;
        Addr=Buffer[Segment*2-1];
    }
    else
    {
        Segment=Buffer[0];
        Addr=Buffer[Buffer[0]*2-1];
    }
    Play_Initial();
    SubStatus = Play;
break;
case 2:                                  //Play next segment
    if(Segment<Buffer[0])
    {
        Segment++;
        Addr=Buffer[Segment*2-1];
    }
    else
    {
        Segment=1;
        Addr=Buffer[1];
    }
    Play_Initial();
    SubStatus = Play;
break;
case 3:                                  //Stop
    SubStatus = Stop;
break;
case 4:                                  //Replay all segments
    Addr=Buffer[Segment*2-1];
    Play_Initial();
    SubStatus = RePlay_All;
break;
case 5:                                  //Replay current segment only
    Addr=Buffer[Segment*2-1];
    Play_Initial();
    SubStatus = RePlay;
break;
case 7:                                  //Record to serial flash
```

```
                    Addr = Buffer[Buffer[0]*2+1];
                    SACM_DVR_InitQueue();
                    SACM_DVR_InitEncoder(RceMonitorOff);
                    SubStatus = Record;
                break;
                case 15:                                    //Open UART
                    Close_Interrupt();
                    SP_UART_Init();
                    BaseType=UartOpen;
                    SubStatus=Stop;
                break;
                default:
                break;
            }
        }
    }

    else                                                    //BaseType = UartCpen
    {
        switch(Keycode)
        {
            case 1:                                         //Upload
                SubStatus=Upload;
            break;
            case 2:                                         //Download
                SubStatus=Download;
            break;
            case 3:                                         //Format serial flash
                SP_SIOMassErase();
                SP_PageErase(0xf000);
                for(i=0;i<21;i++)
                {
                    Buffer[i]=0;
                    SP_WriteWord(0xf000+i,0);
                }
                Segment = 1;                                //variable intitial
                Addr = 0;
                BaseType = UartClose;
                SubStatus = Stop;
            break;
            case 15:                                        //Close UART
                Open_Interrupt();
```

```
                    BaseType=UartClose;
                    SubStatus = Stop;
                break;
                default:
                break;
            }
    }
*************************************************************************
```

以下部分是根据子状态作相应操作的代码。

```
*************************************************************************
        switch(SubStatus)
        {
            case Record:
                if(Addr < C_SflashSize)                 // SPR4096 is not full
                {
                    SACM_DVR_Encode();                  // Get data and Encode
                    if(SACM_DVR_TestQueue() != Empty)   //check queue not empty
                    {
                        Ret = SACM_DVR_FetchQueue();    // Get data from Queue
                        SP_SIOSendAWord(Addr,Ret);      // save to User SPR4096
                        Addr += 2;
                    }
                }
                else                                    // SPR4096 is full
                {
                    DVR_Stop();
                    Send_QueueData_to_Sflash();
                    Modify_Index();
                    SubStatus = Stop;
                }
            break;
            case Play:
                if(SACM_DVR_Status()&0x01)              // Check if it is "Playing"
                {
                    Fill_A_Word_To_Queue();
                    SACM_DVR_Decode();
                }
                else
                {
                    DVR_Stop();
                    SubStatus = Stop;
```

```
        }
    break;

    case RePlay_All:
        if(SACM_DVR_Status()&0x01)              // Check if it is "Playing"
        {
            Fill_A_Word_To_Queue();
            SACM_DVR_Decode();
        }
        else
        {
            DVR_Stop();
            if(Segment<Buffer[0])
            {
                Segment++;
                Addr=Buffer[Segment*2-1];
            }
            else
            {
                Segment=1;
                Addr=Buffer[1];
            }
            Play_Initial();
        }
    break;
    case RePlay:
        if(SACM_DVR_Status()&0x01)              // Check if it is "Playing"
        {
            Fill_A_Word_To_Queue();
            SACM_DVR_Decode();
        }
        else
        {
            DVR_Stop();
            Addr=Buffer[Segment*2-1];
            Play_Initial();
        }
    break;
    case Download:                              //Download
        Addr=Buffer[Buffer[0]*2+1];
        UART_Download();
        Modify_Index();
```

```
                BaseType = UartClose;
                SubStatus = Stop;
                Open_Interrupt();
            break;
            case Upload:                                    //Upload
                Addr0=Buffer[Segment*2-1];
                Addr1=Buffer[Segment*2];
                for(Addr=Addr0;Addr<=Addr1;Addr++)
                {
                    Ret=SP_SIOReadAByte(Addr);
                    SP_UartSentByte(Ret);
                }
                BaseType = UartClose;
                SubStatus = Stop;
                Open_Interrupt();
            break;
            case Stop:
            default:
                i=50;
                while(i--)
                {
                    if(BaseType)
                        SP_Export(P_IOA_Data,0x3f00);     //UART open , light L6L7
                    else
                        SP_Export(P_IOA_Data,0xfc00);     //UART close, light L0L1
                }
            break;
    }
*************************************************************************
```

我们可以看到程序中调用了多次 Modify_Index()，它的功能是什么呢？我们看看它的代码就知道了。以下这段代码包括了一个读、改、写的过程。先把索引区的信息读到 Buffer 数组中保存起来，然后修改，在对索引区进行页擦除之后又把修改后的索引信息写回去。大家可以看到代码中计算下一段开始地址时，地址增量是 2048。为什么是 2048 呢？我们在上文提到，SPR4096 的每个扇区为 2K-byte，也就是 2048-Byte。擦除 SPR4096 的最小单位就是 2048-Byte。为了保证删除其中某段语音数据的时候，不会删除到相邻的其它段，所以把地址增量设为 2048。

```
*************************************************************************
void Modify_Index(void)
{
    for(i=0,internal_addr=0xf000;i<21;i++,internal_addr++)
        Buffer[i]=SP_ReadWord(internal_addr);
    Buffer[0]+=1;                                   //Segment number
```

```
    Ret=Buffer[0]*2;
    Buffer[Ret]=Addr;                          //StopAddress of current segment
    Tem=0;
    for(i=1;(i<256)&&((Tem<Addr)||(Tem==Addr));i++)
        Tem=2048*i;
    Buffer[Ret+1]=Tem;                         //StartAddress of next segment
    SP_PageErase(0xf000);
    for(i=0,internal_addr=0xf000;i<21;i++,internal_addr++)
        SP_WriteWord(internal_addr,Buffer[i]);   //store index in internal flash rom
}
*****************************************************************************
```

以上涉及的函数，SP_ReadWord()、SP_WriteWord()和 SP_PageErase()的代码在 10.3.3 小节，SP_SIOMassErase()、SP_PageErase()、SP_SIOSendAWord()和 SP_SIOReadAByte()的代码在 10.3.4 小节，SP_UART_Init()、SP_UartSentByte()、UART_Download()的代码在 10.3.5 小节，其它函数的代码请到 www.unsp.com.cn 下载。

10.3.2 硬件系统初始化程序

硬件系统初始化包括中断的设置，IO 的设置。这里分别给出了纯汇编格式和调用库函数实现的两种方式。

以下为是调用库函数实现系统初始化的代码。

```
*****************************************************************************
    void   System_Initial()
    {
        SP_Init_IOA(0xfff0, 0xfff0, 0xfff0);
        SP_Init_IOB(0xff7f, 0xffff, 0x0003);
        Set_SIO_Ctrl(0x00C3)
        R_InterruptStatus = C_FIQ_TMA;
        Set_INT_Ctrl(C_FIQ_TMA);
    }
*****************************************************************************
```

以下为是用汇编语言实现系统初始化的代码。

```
*****************************************************************************
    .PUBLIC _System_Initial;
    _System_Initial: .PROC
        CALL    F_User_Init_IO;            //IO initial
        R1=C_FIQ_TMA                       //Set FIQ_TMA interrupt
        [_R_InterruptStatus] = R1
        [P_INT_Ctrl] = R1
        RETF;
        .ENDP;
-----------------------------------------------------------------------------
```

```
F_User_Init_IO:
    R1 = 0xfff0;                          //high-8-bit for Led, low-8-bit for key
    [P_IOA_Attrib] = R1;
    [P_IOA_Dir] = R1;
    [P_IOA_Data] = R1;
    R1 = 0xFF7F;      //Set IOB7 as Input Floating, IOB10 as Output Buffer Low
    [P_IOB_Dir] = R1;
    R1 = 0xFFFF;
    [P_IOB_Attrib] = R1;
    R1 = 0x0003;
    [P_IOB_Data] = R1;
    R1 = 0x00C3;                          //Set SIO status
    [P_SIO_Ctrl] = R1;
    RETF;
********************************************************************************
```

10.3.3 内部 FLASH 的读写程序

32K 字的内部 FLASH 被划分为 128 个 PAGE(每个 PAGE 存储容量为 256 字)，第一页 [0x8000—0x80ff]最后一页为[0xff00—0xffff]。全部 32K 字闪存均可被编程写入或被擦除。

上电以后，芯片就处于读存储单元状态，读存储单元的操作与 SRAM 相同。函数 SP_ReadWord()实现的功能是从内部 FLASH 中读出一个 Word，参数为 FLASH 地址，返回值为该地址存放的数据。以下分别给出了用 C 语言实现和用汇编语言实现的代码。顺便说一下，本小节的函数均给出了用 C 语言实现和用汇编语言实现的代码。

SP_ReadWord ()用 C 语言实现的代码如下：

```
********************************************************************************
unsigned int   SP_ReadWord (unsigned int addr)
{
    unsigned int *ADDR,data;
    ADDR=(unsigned int*)addr;
    data = *ADDR;
    return data;
}
********************************************************************************
```

SP_ReadWord()用汇编语言实现的代码如下：

```
********************************************************************************
//-- Procedure:     _SP_ReadWord
//-- Syntax:        SP_ReadWord(page addr)
//-- Parameter:     R1 = page addr
//-- Return:        R1 = data
--------------------------------------------------------------------------------
```

```
.PUBLIC _SP_ReadWord
_SP_ReadWord: .PROC
      PUSH BP TO [SP];
      BP = SP + 1;
      R1 = [BP+3];
      CALL F_ReadWord;
      POP BP FROM [SP];
      RETF
      .ENDP
－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－
.PUBLIC F_ReadWord
F_ReadWord: .PROC
      PUSH R2,R2 to [SP];
      R2 = [R1];
      R1 = R2;
      POP R2,R2 from [SP];
      RETF;
      .ENDP
***********************************************************************
```

和读 FLASH 相比，擦写就麻烦一点了。图 10.11 说明了擦写的过程。需要指出一点，图中所提到延时等待是由硬件完成不需要软件延时。

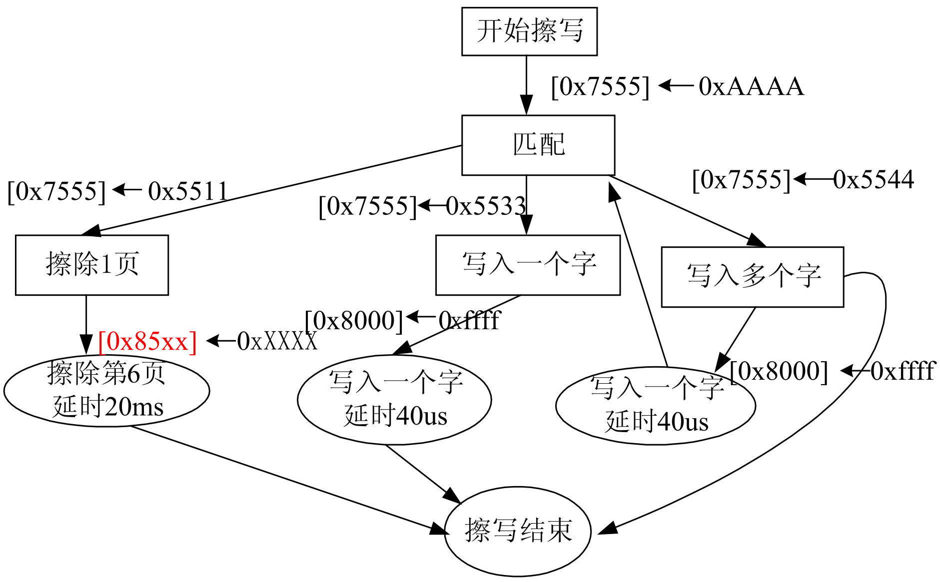

图10.11 内部 FLASH 的擦写过程

SP_PageErase()为页擦除函数。

用 C 语言实现 SP_PageErase()的代码如下：

```
*************************************************************************
    void   SP_PageErase(unsigned int addr)
    {
        unsigned int *ADDR;
        ADDR=(unsigned int*)addr;
        *P_Flash_Ctrl = 0xAAAA;
        *P_Flash_Ctrl = 0x5511;
        *ADDR=1;
    }
*************************************************************************
```

用汇编语言实现 SP_PageErase()的代码如下：

```
*************************************************************************
//-- Procedure:     _SP_PageErase
//-- Decription:    Erase a page in flash
//-- Syntax:        SP_PageErase(page addr)
//-- Parameter:     R1 = page addr
//-- Return:        None
－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－
.PUBLIC F_PageErase

F_PAGEERASE: .PROC

  PUSH R2 to [SP];
  R2 = R1;
  R1 = 0xAAAA;                    // Enable Write/Erase
  [P_Flash_Ctrl] = R1;
  R1 = 0x5511;                    // Erase instruction
  [P_Flash_Ctrl] = R1;
  [R2] = R1;                      // Write any value to erase page
  POP R2 from [SP];
  RETF;
  .ENDP
*************************************************************************
```

SP_WriteWord()的功能是往 FLASH 写一个 Word 的数据，该函数有两个参数，第一个是 FLASH 地址，第二个是要写入的数据。

用 C 语言实现 SP_WriteWord()的代码如下：

```
*************************************************************************
    void   SP_WriteWord (unsigned int addr,unsigned int data)
    {
        unsigned int *ADDR;
        ADDR=(unsigned int*)addr;
        *P_Flash_Ctrl = 0xAAAA;
        *P_Flash_Ctrl = 0x5533;
```

```
        *ADDR=data;
    }
*********************************************************************
```

用汇编语言实现 SP_WriteWord()的代码如下：

```
*********************************************************************
//-- Procedure:     _SP_WriteWord
//-- Syntax:        SP_WriteWord(page addr,data)
//-- Parameter:     R1 = page addr，      R2 = data
//-- Return:        None.PUBLIC _SP_WriteWord
－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－
.PUBLIC _SP_WriteWord
_SP_WriteWord: .PROC
    PUSH BP,BP TO [SP];
    BP = SP + 1;
    R1 = [BP+3];
    R2 = [BP+4];
    CALL F_WriteWord;
    POP BP,BP FROM [SP];
    RETF
    .ENDP
－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－
.PUBLIC F_WriteWord
F_WriteWord: .PROC
    PUSH R3,R3 to [SP];
    R3 = R1;
    R1 = C_EnableFlashAccess;          // Enable Write/Erase
    [P_Flash_Ctrl] = R1;
    R1 = C_ProgramFlash;               // Program instruction
    [P_Flash_Ctrl] = R1;
    [R3] = R2;                         // Write Flash
    POP R3,R3 from [SP];
    RETF;
    .ENDP
*********************************************************************
```

10.3.4 串行 FLASH 的读写程序

串行 FLASH 读写从原理上来说，和 SPCE061A 内部的 FLASH 读写没有什么区别。大家已经看过 SPCE061A 内部的 FLASH 读写代码，所以理解这个小节的代码就是轻而易举的事情了。本小节跟上个小节一样，用到的函数均给出了用 C 语言实现和用汇编语言实现的代码。

SP_SIOSendAByte()实现的功能是送一个 Byte 的数据到串行 FLASH，送一个 Word 的数据到串行 FLASH 的代码可以到 www.unsp.com.cn 下载。

用 C 语言实现 SP_SIOSendAByte()的代码如下：

```
*************************************************************************
    void   SP_SIOSendAByte (unsigned long int addr,unsigned int data)
    {
        int a;
        *P_SIO_Addr_Low=addr;                  // input SFLASH low address
        addr>>=8;                              // right shift 8
        *P_SIO_Addr_Mid=addr;                  // input SFLASH mid address
        addr>>=8;                              // right shift 8
        addr&=0x0007;                          // input SFLASH hi address
        *P_SIO_Addr_High=addr;
        *P_SIO_Ctrl=0x00C3+C_SIOCLOCK;   // clk=CPUclk/8, 24 bit address    ;write
        *P_SIO_Start=1;                        // enable write mode
        a=0x2ff;                               // delay
        while(a--);
        *P_SIO_Data=data;                      // state to transmit data
        Wait_FreeFlag();                       // Wait, until SIO is not busy
        *P_SIO_Stop=1;                         // disable write mode
    }
*************************************************************************
```

这里调用了一个函数 Wait_FreeFlag()，它的功能是查询 P_SIO_Start 单元的第七位是否为“1”，等待 SIO 到 non_busy 状态。以下是该函数的代码：

```
*************************************************************************
    void Wait_FreeFlag(void)
    {
        unsigned int a;
        a=*P_SIO_Start;
        a&=0x0080;
        while(a)
        {
            a=*P_SIO_Start;
            a&=0x0080;
        }
    }
*************************************************************************
```

用汇编语言实现 SP_SIOSendAByte()的代码如下：

```
*************************************************************************
    // Function:        Send A Byte to Serial Flash
    // Syntax:          SP_SIOSendAByte(AddressLow,AddressHigh,   data)
    // Used register:   R1,R2,R3
－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－
```

```
    .PUBLIC _SP_SIOSendAByte;
    _SP_SIOSendAByte: .PROC
        PUSH BP,BP TO [SP];
        BP = SP + 1;
        R1 = [BP+3];
        [P_SIO_Addr_Low]=R1;            // input SFLASH low address
        R1=R1 LSR 4;                    // right shift 8
        R1=R1 LSR 4;
        [P_SIO_Addr_Mid]=R1;            // input SFLASH mid address
        R1 = [BP+4];                    // Port direction
        R1=R1&0x0007;                   // input SFLASH hi address
        [P_SIO_Addr_High]=R1;
        R1=0x00C3+C_SIOCLOCK;           //C_SIOCLOCK=0x0010
        [P_SIO_Ctrl]=R1;                // clk=CPUclk/8, 24 bit address ;write
        [P_SIO_Start]=R1;               // enable write mode
        R1 = [BP+5];
        [P_SIO_Data]=R1;                // state to transmit data
    L_WaitSIOSendReady:
        R1=[P_SIO_Start];
        TEST      R1,0x0080
        JNZ       L_WaitSIOSendReady
        [P_SIO_Stop]=R1;                //disable write mode
        POP BP,BP FROM [SP];
        RETF;
        .ENDP;
*************************************************************************
```

SP_SIOReadAByte ()实现的功能是从串行 FLASH 读出一个 Byte 的数据，从串行 FLASH 读出一个 Word 的数据的代码可以到 www.unsp.com.cn 下载。

用 C 语言实现 SP_SIOReadAByte ()的代码如下：

```
*************************************************************************
    unsigned int SP_SIOReadAByte (unsigned long int addr)
    {
        unsigned int a,data;
        *P_SIO_Addr_Low=addr;               // input SFLASH low address
        addr>>=8;                           // right shift 8
        *P_SIO_Addr_Mid=addr;               // input SFLASH mid address
        addr>>=8;                           // right shift 8
        addr&=0x0007;                       // input SFLASH hi address
        *P_SIO_Addr_High=addr;
        *P_SIO_Ctrl=0x0083+C_SIOCLOCK;      // clk=CPUclk/16, 24 bit address    ;read
        *P_SIO_Start=1;                     // enable read mode
```

```
        a=*P_SIO_Data;                    // Clear SFLASH buffer
        Wait_FreeFlag();                  // Wait, until SIO is not busy
        data=*P_SIO_Data;                 // Read exact Data
        Wait_FreeFlag();                  // Wait, until SIO is not busy
        *P_SIO_Stop=1;                    // disable read mode
        return(data);
    }
*************************************************************************
```

用汇编语言实现 SP_SIOReadAByte ()的代码如下：

```
*************************************************************************
    // Function: Read A Byte to Serial Flash
    // Syntax: SP_SIOReadAByte(AddressLow, AddressHigh)
    // Used register: R1,R2,R3
    // Return register: R1
－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－
    .PUBLIC _SP_SIOReadAByte;
    _SP_SIOReadAByte: .PROC
            PUSH BP,BP TO [SP];
            BP = SP + 1;
            R1 = [BP+3];
            [P_SIO_Addr_Low]=R1;          // input SFLASH low address
            R1=R1 LSR 4;
            R1=R1 LSR 4;
            [P_SIO_Addr_Mid]=R1;          // input SFLASH mid address
            R1 = [BP+4];                  // Port direction
            R1=R1&0x0007;                 // input SFLASH hi address
            [P_SIO_Addr_High]=R1;
            R1=0x0083+C_SIOCLOCK;
            [P_SIO_Ctrl]=R1;              // clk=CPUclk/16, 24 bit address    ;read
            [P_SIO_Start]=R1;             // enable read mode
            R2=[P_SIO_Data];              // Clear SFLASH buffer
    L_WaitSIOReadReady1:
            R1=[P_SIO_Start];
            TEST      R1,0x0080
            JNZ       L_WaitSIOReadReady1
            R1=[P_SIO_Data];              // Read exact Data
    L_WaitSIOReadReady2:                  // Wait read stop
            R2=[P_SIO_Start];
            TEST      R2,0x0080
            JNZ       L_WaitSIOReadReady2
            [P_SIO_Stop]=R2;              // disable read mode
```

```
            POP BP,BP FROM [SP];
            RETF;
            .ENDP;
*************************************************************************
```

SIOMassErase()实现的功能是对串行 FLASH 进行格式化操作，对串行 FLASH 进行页擦除的代码可以到 www.unsp.com.cn 下载。

用 C 语言实现 SP_SIOMassErase ()的代码如下：

```
*************************************************************************
    void      SP_SIOMassErase (void)
    {
        long int a;
        *P_SIO_Ctrl=0x00C0+C_SIOCLOCK;   // clk=CPUclk/8, 16 bit address    ;write
        *P_SIO_Addr_Low=0x0000;           // input SFLASH low address
        *P_SIO_Addr_Mid=0x00C0;           // input SFLASH mid address
        *P_SIO_Addr_High=0x00C0;          // input SFLASH hi address
        *P_SIO_Start=1;                   // enable write mode
        *P_SIO_Data=0;                    // A7~A0 = 0,state to transmit data
        Wait_FreeFlag();                  // Wait, until SIO is not busy
        *P_SIO_Stop=1;                    // disable read mode
        a=0x7ff;                          // delay
        while(a--);
    }
*************************************************************************
```

用汇编语言实现 SP_SIOMassErase ()的代码如下：

```
*************************************************************************
    //Function : Mass Erase for S_Flash
    // Syntax: SIOMassErase()
    // Used register: R1,R2
————————————————————————————————————————————————————————
    .PUBLIC _SP_SIOMassErase;
    _SP_SIOMassErase: .PROC
            PUSH R1,R2 to [SP];
            R1=0x00C0+C_SIOCLOCK;
            [P_SIO_Ctrl]=R1;            // clk=CPUclk/8, 16 bit address    ;write
            R2=0x0000;
            [P_SIO_Addr_Low]=R2;        // input SFLASH low address
            R2=0x00C0;
            [P_SIO_Addr_Mid]=R2;        // input SFLASH mid address
            R2=0x00C0;
           [P_SIO_Addr_High]=R2;
```

```
            [P_SIO_Start]=R1;                   // enable write mode
            R1=0;                               // A7~A0 = 0
            [P_SIO_Data]=R1;                    // state to transmit data
L_WaitSIOSendReadyMass:
            R1=[P_SIO_Start];
            TEST    R1,0x0080
            JNZ     L_WaitSIOSendReadyMass
            [P_SIO_Stop]=R1;                    //disable write mode
            POP R1,R2 from [SP];
            RETF;
            .ENDP;
*********************************************************************
```

10.3.5 UART 通讯程序

UART 通讯程序包括 UART 的初始化、发送数据、读取数据等。本节给出的函数全部用 C 语言实现。需要用汇编语言实现的可以到 www.unsp.com.cn 下载。

UART 初始化的函数可以这样写：

```
*********************************************************************
void  SP_UART_Init (void)
{
    int a;
    *P_UART_Command1=0x20;              //Uart internal RESET
    *P_UART_Command1=0x00;
    *P_UART_BaudScalarHigh=0x00;        //Set UART baud rate to 115200
    *P_UART_BaudScalarLow=0x6B;
    *P_UART_Command1=0x000C;            //Uart setting
    *P_UART_Command2=0x00C0;
    a=*P_UART_Data;                     // clear receiving buffer
}
*********************************************************************
```

发送一个 Byte 的函数可以这样写：

```
*********************************************************************
void  SP_UartSentByte (unsigned int data)
{
    int a;
    a=*P_UART_Command2;
    a=a&0x0040;
    while(!a)                           //Check bit 6 to see if TxRDY = 1
    {
```

```
            a=*P_UART_Command2;
            a=a&0x0040;
        }
        *P_UART_Data=data;                  // send data
    }
*****************************************************************************
```

接收一个 Byte 的函数可以这样写：

```
*****************************************************************************
    unsigned int    SP_UartReadByte (void)
    {
        unsigned int data,a;
        a=*P_UART_Command2;
        a=a&0x0080;
        while(!a)                           //Check bit 7 to see if RxRDY = 1
        {
            a=*P_UART_Command2;
            a=a&0x0080;
        }
        data=*P_UART_Data;                  // read data
        return(data);
    }
*****************************************************************************
```

10.3.6 键盘扫描程序

F_Key_Scan()为 4×4 键盘扫描函数，返回值和按键的关系见图 10.5。需要说明一点，没有键按下的时候返回值为 0xff。为了节省时间，F_Key_Scan()是用汇编语言编写的。鉴于大家对键盘扫描都比较熟悉，所以这里就不浪费篇幅了。有需要的可以到 www.unsp.com.cn 下载 F_Key_Scan()的代码。

10.4 小结

本章介绍了一个应用 SPCE061A 作为控制核心实现简易录音笔的实例。我们介绍了模块的硬件设计方法以及针对各个硬件进行编程的过程，并给出了将各个硬件功能组合起来形成一个可用的简易录音笔的方法。这种设计方法对于一般单片机应用系统的开发同样适用。