Copyright

这里主要是来自`安徽芒课教育科技有限公司的单片机与嵌入式系统竞赛实训平台`官方提供的资料, 部分模块没来得及自己封装, 便将可能用到资料整合(部分有做修改)在此文件

目录

SPI

```c
// spi.h
#ifndef SPI_H
#define SPI_H

#include "__type__.h"

void SPI_Init(void); // SPI初始化
u8 SPI_SendByte(u8 byte); // SPI为全双工通讯
void SPI_Speed(u8 speed); // SPI速度调节

#endif // SPI_H
```

```c
// spi.c
#include "__config__.h"
#include "spi.h"

// SPI初始化: 主要将SPI总线 从默认P1口上调制P4口, 然后初始化SPI总线
void SPI_Init(void)
{
#if 0
    P4M0 &= 0xFD; // 配置 P4.1 MISO口 仅为输入功能
    P4M1 |= 0x02;
#endif
    AUXR1 |= 0X08; //将 SPI 调整到 P4.0 P4.1 P4.3
    SPDAT = 0;
    SPSTAT = 0xc0; //SPDAT.7和SPDAT.6写11, 可以将中断标志清零。注意是写1才清零
    /**
     * SSIG 1 开启主机模式
     * SPEN 1 SPI使能
     * DORD 0 先发最高位
     * MSTR 1 主机模式
     * CPOL 0 SPICLK空闲时为低
     * CPHA 0 数据在SPICLK的前时钟沿驱动  时钟CPU_CLK/4
    */
    SPCTL = 0xd0;
}

// SPI为全双工通讯  所以在发送的同时可以接收到数据
u8 SPI_SendByte(u8 byte)
{
    SPDAT = byte; // 将数据 写入

    while ((SPSTAT & 0x80) == 0)
        ; //等待写入完成

    SPSTAT = 0xc0; // 清除中断标志, 和写冲突标志, 注意是对应位写1才能清零
    return SPDAT; // 返回得到的数据
}

// SPI时钟速率设置
void SPI_Speed(u8 speed)
{
    // 每一次降速 都要先清为最高 在进行降速
    SPCTL &= 0xFC;
    SPCTL |= speed & 0x03;
    /**
     * SysClk/4,   SPR1=0, SPR0=0
     * SysClk/16,  SPR1=0, SPR0=1
     * SysClk/64,  SPR1=1, SPR0=0
     * SysClk/128, SPR1=1, SPR0=1
    */
}
```

SD

```
// sd.h
#ifndef SD_H
#define SD_H

#include "__config__.h"

// 错误码定义
// ------------------------------------------------------------
#define INIT_CMD0_ERROR   0x01 // CMD0错误
#define INIT_CMD1_ERROR   0x02 // CMD1错误
#define WRITE_BLOCK_ERROR 0x03 // 写块错误
#define READ_BLOCK_ERROR  0x04 // 读块错误
// ------------------------------------------------------------

// SD卡类型定义
#define SD_TYPE_ERR  0X00
#define SD_TYPE_V2   0X04
#define SD_TYPE_V2HC 0X06

// SD传输数据结束后是否释放总线宏定义
#define NO_RELEASE 0
#define RELEASE    1

// SD卡回应标记字
#define MSD_RESPONSE_NO_ERROR    0x00
#define MSD_IN_IDLE_STATE        0x01
#define MSD_ERASE_RESET          0x02
#define MSD_ILLEGAL_COMMAND      0x04
#define MSD_COM_CRC_ERROR        0x08
#define MSD_ERASE_SEQUENCE_ERROR 0x10
#define MSD_ADDRESS_ERROR        0x20
#define MSD_PARAMETER_ERROR      0x40
#define MSD_RESPONSE_FAILURE     0xFF

// SD卡指令表
#define CMD0  0 // 卡复位
#define CMD1  1
#define CMD8  8  // 命令8 , SEND_IF_COND
#define CMD9  9  // 命令9 , 读CSD数据
#define CMD10 10 // 命令10, 读CID数据
#define CMD12 12 // 命令12, 停止数据传输
#define CMD16 16 // 命令16, 设置SectorSize 应返回0x00
#define CMD17 17 // 命令17, 读sector
#define CMD18 18 // 命令18, 读Multi sector
#define CMD23 23 // 命令23, 设置多sector写入前预先擦除N个block
#define CMD24 24 // 命令24, 写sector
#define CMD25 25 // 命令25, 写Multi sector
#define CMD41 41 // 命令41, 应返回0x00
```

```c
#define CMD55 55 // 命令55，应返回0x01
#define CMD58 58 // 命令58，读OCR信息
#define CMD59 59 // 命令59，使能/禁止CRC，应返回0x00

u8 SD_WaitReady(void);                  // 等待SD卡准备
u8 SD_Init(void);                       // 初始化
u8 SD_SendCmd(u8 cmd, u32 arg, u8 crc); // 写命令
void SD_DisSelect(void);                // 释放片选
u32 SD_GetCapacity(void);               // 获取SD卡的容量（字节）

// 向SD卡写入命令之后，读取SD卡的回应次数，
// 即读TRY_TIME次，如果在TRY_TIME次中读不到回应，产生超时错误，命令写入失败
#define TRY_TIME 10

sbit SD_CS = P4 ^ 6; // TF卡使能IO
#endif
```

```c
// sd.c
#include "sd.h"
#include "spi.h"
#include "stc15f2k60s2.h"

u8 SD_Type = 0; // SD卡的类型

//////////////////////////////////////////////////////////////////////////////////

// 等待卡准备好
// 返回值:0,准备好了;其他,错误代码
u8 SD_WaitReady(void)
{
    u32 t = 0;
    do
    {
        if (SPI_SendByte(0XFF) == 0XFF)
            return 0; // OK
        t++;
    } while (t < 0XFFFFFF); // 等待
    return 1;
}

// 取消选择,释放SPI总线
void SD_DisSelect(void)
{
    SD_CS = 1;
    SPI_SendByte(0xff); // 提供额外的8个时钟
}

// 选择sd卡,并且等待卡准备OK
// 返回值:0,成功;1,失败;
u8 SD_Select(void)
{
    SD_CS = 0;
    if (SD_WaitReady() == 0)
        return 0; // 等待成功
    SD_DisSelect();
    return 1; // 等待失败
}

// 向SD卡发送一个命令
// 输入: u8 cmd   命令
//       u32 arg  命令参数
//       u8 crc   crc校验值
// 返回值:SD卡返回的响应
u8 SD_SendCmd(u8 cmd, u32 arg, u8 crc)
{
    u8 r1;
    u8 Retry = 0;

    SD_DisSelect(); // 取消上次片选
```

```
    if (SD_Select())
        return 0XFF; // 片选失效

    // 发送
    SD_CS = 1; // 片选拉高
    SPI_SendByte(0xff);
    SPI_SendByte(0xff);
    SPI_SendByte(0xff);
    SD_CS = 0;

    SPI_SendByte(cmd | 0x40); // 分别写入命令      最高2位固定为1
    SPI_SendByte(arg >> 24);  // 命令参数 2-5字节   4个字节 32位
    SPI_SendByte(arg >> 16);
    SPI_SendByte(arg >> 8);
    SPI_SendByte(arg);
    SPI_SendByte(crc);
    // 停止传数据命令
    if (cmd == CMD12)
        SPI_SendByte(0xff); // Skip a stuff byte when stop reading
                            // 等待响应，或超时退出
    Retry = 0X1F;           // 循环32次
    do
    {
        r1 = SPI_SendByte(0xFF);
    } while ((r1 & 0X80) && Retry--);
    // 返回状态值
    return r1;
}

// 初始化SD卡
u8 SD_Init(void)
{
    u8 r1;      // 存放SD卡的返回值
    u16 retry; // 用来进行超时计数
    u8 buf[4];
    u8 i;

    SPI_Speed(3); // 设置到低速模式

    // 发送最少74个脉冲 等待SD卡内部供电电压上升时间       进入SPI模式
    for (i = 0; i < 10; i++)
        SPI_SendByte(0XFF);

    retry = 20;
    do
    {
        r1 = SD_SendCmd(CMD0, 0, 0x95); // 进入IDLE状态    即复位sd卡 空闲状态
    } while ((r1 != 0X01) && retry--);

    SD_Type = 0; // 默认无卡

    if (r1 == 0X01) // 获取版本信息
```

```
    {
        if (SD_SendCmd(CMD8, 0x1AA, 0x87) == 1) // SD V2.0    发送接口状态命令
        { // 如果返回值为1 则是 SDV2.0版本
            for (i = 0; i < 4; i++)
                buf[i] =
                    SPI_SendByte(0XFF); // Get trailing return value of R7 resp
                                        // 提取返回值R7数据
            if (buf[2] == 0X01 && buf[3] == 0XAA) // 卡是否支持2.7~3.6V
            {
                retry = 0XFFFE;
                do
                {
                    SD_SendCmd(CMD55, 0, 0X01);               // 发送CMD55
                    r1 = SD_SendCmd(CMD41, 0x40000000, 0X01); // 发送CMD41
                } while (r1 && retry--);
                if (retry &&
                    SD_SendCmd(CMD58, 0, 0X01) == 0) // 鉴别SD2.0卡版本开始
                {
                    for (i = 0; i < 4; i++)
                        buf[i] = SPI_SendByte(0XFF); // 得到OCR值
                    if (buf[0] & 0x40)
                        SD_Type = SD_TYPE_V2HC; // 检查CCS
                    else
                        SD_Type = SD_TYPE_V2;
                }
                else
                    SD_Type = SD_TYPE_ERR; // 错误的卡
            }
        }
    }

    SD_DisSelect(); // 取消片选
    SPI_Speed(0);   // 高速
    if (SD_Type)
        return 0; // 如果没有采集到SD卡版本 则跳出函数
    else if (r1)
        return r1;
    return 0xaa; // 其他错误
}
```

VS1053

```
// vs1053.h
#ifndef VS1053_H
#define VS1053_H

#include "__config__.h"

//////////////////////////////////////////////////////////////
// 与外部的接口
// #define VS_DQ       PCin(13)  // DREQ
// #define VS_RST      PEout(6)  // RST
// #define VS_XCS      PFout(7)  // XCS
// #define VS_XDCS     PFout(6)  // XDCS

sbit VS_RST  = P1 ^ 3; // RST
sbit VS_DQ   = P1 ^ 1; // DREQ
sbit VS_XDCS = P1 ^ 0; // XDCS
sbit VS_XCS  = P3 ^ 4; // XCS

//////////////////////////////////////////////////////////////

typedef struct
{
    u8 mvol;    // 主音量,范围:0~254
    u8 bflimit; // 低音频率限定,范围:2~15(单位:10Hz)
    u8 bass;    // 低音,范围:0~15.0表示关闭.(单位:1dB)
    u8 tflimit; // 高音频率限定,范围:1~15(单位:Khz)
    u8 treble; // 高音,范围:0~15(单位:1.5dB)(原本范围是:-8~7,通过函数修改了);
    u8 effect;    // 空间效果设置.0,关闭;1,最小;2,中等;3,最大.
    u8 speakersw; // 板载喇叭开关,0,关闭;1,打开 // 增加
    u8 saveflag;  // 保存标志,0X0A,保存过了;其他,还从未保存
} _vs10xx_obj;

extern _vs10xx_obj vsset; // VS10XX设置

#define VS_WRITE_COMMAND 0x02
#define VS_READ_COMMAND  0x03
// VS10XX寄存器定义
#define SPI_MODE        0x00
#define SPI_STATUS      0x01
#define SPI_BASS        0x02
#define SPI_CLOCKF      0x03
#define SPI_DECODE_TIME 0x04
#define SPI_AUDATA      0x05
#define SPI_WRAM        0x06
#define SPI_WRAMADDR    0x07
#define SPI_HDAT0       0x08
#define SPI_HDAT1       0x09
```

```c
#define SPI_AIADDR  0x0a
#define SPI_VOL     0x0b
#define SPI_AICTRL0 0x0c
#define SPI_AICTRL1 0x0d
#define SPI_AICTRL2 0x0e
#define SPI_AICTRL3 0x0f
#define SM_DIFF     0x01
#define SM_JUMP     0x02
#define SM_RESET    0x04
#define SM_OUTOFWAV 0x08
#define SM_PDOWN    0x10
#define SM_TESTS    0x20
#define SM_STREAM   0x40
#define SM_PLUSV    0x80
#define SM_DACT     0x100
#define SM_SDIORD   0x200
#define SM_SDISHARE 0x400
#define SM_SDINEW   0x800
#define SM_ADPCM    0x1000
#define SM_ADPCM_HP 0x2000

#define I2S_CONFIG 0XC040
#define GPIO_DDR   0XC017
#define GPIO_IDATA 0XC018
#define GPIO_ODATA 0XC019 // 新增实现对功放芯片控制

u16 VS_RD_Reg(u8 address);           // 读寄存器
u16 VS_WRAM_Read(u16 addr);          // 读RAM
void VS_WR_Data(u8 dat);             // 写数据
void VS_WR_Cmd(u8 address, u16 dat); // 写命令
u8 VS_HD_Reset(void);                // 硬复位
void VS_Soft_Reset(void);            // 软复位
u16 VS_Ram_Test(void);               // RAM测试
void VS_Sine_Test(void);             // 正弦测试

u8 VS_SPI_ReadWriteByte(u8 dat);
void VS_SPI_SpeedLow(void);
void VS_SPI_SpeedHigh(void);
void VS_Init(void);             // 初始化VS10XX
void VS_Set_Speed(u8 t);        // 设置播放速度
u16 VS_Get_HeadInfo(void);      // 得到比特率
u32 VS_Get_ByteRate(void);      // 得到字节速率
u16 VS_Get_EndFillByte(void);   // 得到填充字节
u8 VS_Send_MusicData(u8* buf);  // 向VS10XX发送32字节
void VS_Restart_Play(void);     // 重新开始下一首歌播放
void VS_Reset_DecodeTime(void); // 重设解码时间
u16 VS_Get_DecodeTime(void);    // 得到解码时间

// void VS_Load_Patch(u16* patch, u16 len);                // 加载用户patch
u8 VS_Get_Spec(u16* p);                                 // 得到分析数据
void VS_Set_Bands(u16* buf, u8 bands);                  // 设置中心频率
void VS_Set_Vol(u8 volx);                               // 设置主音量
```

```
void VS_Set_Bass(u8 bfreq, u8 bass, u8 tfreq, u8 treble); // 设置高低音
void VS_Set_Effect(u8 eft);                               // 设置音效
void VS_Set_All(void);

void vs10xx_read_para(_vs10xx_obj* vs10xxdev);
void vs10xx_save_para(_vs10xx_obj* vs10xxdev);

void VS_WRAM_Write(u16 addr, u16 val); // 新增 // 写VS10xx的RAM
void VS_SPK_Set(u8 sw); // 新增 板载喇叭开/关设置函数.

#endif
```

```c
// vs1053.c
#include "vs1053.h"
#include "spi.h"
#include "stc15f2k60s2.h"

// VS10XX默认设置参数
_vs10xx_obj vsset = {
    220, // 音量:220
    6,   // 低音上线 60Hz
    15,  // 低音提升 15dB
    10,  // 高音下限 10Khz
    15,  // 高音提升 10.5dB
    0,   // 空间效果
    1,   // 板载喇叭默认打开.
};

//////////////////////////////////////////////////////////////////////////////
// 移植时候的接口
// dat:要写入的数据
// 返回值:读到的数据
u8 VS_SPI_ReadWriteByte(u8 dat)
{
    return SPI_SendByte(dat);
}

// SPI总线低速
void VS_SPI_SpeedLow(void)
{
    SPI_Speed(3); // 设置到低速模式
}

// SPI总线高速
void VS_SPI_SpeedHigh(void)
{
    SPI_Speed(0); // 设置到高速模式
}

//////////////////////////////////////////////////////////////////////////////
// 软复位VS1053
void VS_Soft_Reset(void)
{
    u8 retry = 0;
    while (VS_DQ == 0)
        ;                          // 等待软件复位结束
    VS_SPI_ReadWriteByte(0Xff); // 启动传输
    retry = 0;
    while (VS_RD_Reg(SPI_MODE) != 0x0800) // 软件复位,新模式
    {
        VS_WR_Cmd(
            SPI_MODE, 0x0804
        ); // 软件复位,新模式 软复位成功后 第三位会从1跳0 这时说明复位成功
        delay_1ms(3); // 等待至少1.35ms
```

```
        if (retry++ > 100)
            break;
    }
    while (VS_DQ == 0)
        ; // 等待软件复位结束
    retry = 0;
    while (VS_RD_Reg(SPI_CLOCKF) != 0X9800) // 设置VS10XX的时钟,3倍频 ,1.5xADD
    {
        VS_WR_Cmd(SPI_CLOCKF, 0X9800); // 设置VS10XX的时钟,3倍频 ,1.5xADD
        if (retry++ > 100)
            break;
    }
    delay_1ms(20);
}

// 硬复位MP3
// 返回1:复位失败;0:复位成功
u8 VS_HD_Reset(void)
{
    u8 retry = 0;
    VS_RST   = 0;
    delay_1ms(20);
    VS_XDCS = 1; // 取消数据传输
    VS_XCS  = 1; // 取消数据传输
    VS_RST  = 1;
    while (VS_DQ == 0 && retry < 200) // 等待DREQ为高
    {
        retry++;
        delay_5us(10);
    };
    delay_1ms(20);
    if (retry >= 200)
        return 1;
    else
        return 0;
}

// 正弦测试
void VS_Sine_Test(void)
{
    VS_HD_Reset();
    VS_WR_Cmd(0x0b, 0X2020);     // 设置音量
    VS_WR_Cmd(SPI_MODE, 0x0820); // 进入VS1053的测试模式
    while (VS_DQ == 0)
        ;   // 等待DREQ为高
            // printf("mode sin:%x\n",VS_RD_Reg(SPI_MODE));
    // 向VS10XX发送正弦测试命令：0x53 0xef 0x6e n 0x00 0x00 0x00 0x00
    // 其中n = 0x24，设定VS10XX所产生的正弦波的频率值，具体计算方法见VS10XX的datasheet
    VS_SPI_SpeedLow(); // 低速
    VS_XDCS = 0;       // 选中数据传输
    VS_SPI_ReadWriteByte(0x53);
    VS_SPI_ReadWriteByte(0xef);
```

```
    VS_SPI_ReadWriteByte(0x6e);
    VS_SPI_ReadWriteByte(0x24);
    VS_SPI_ReadWriteByte(0x00);
    VS_SPI_ReadWriteByte(0x00);
    VS_SPI_ReadWriteByte(0x00);
    VS_SPI_ReadWriteByte(0x00);
    delay_1ms(100);
    VS_XDCS = 1;
    // 退出正弦测试
    VS_XDCS = 0; // 选中数据传输
    VS_SPI_ReadWriteByte(0x45);
    VS_SPI_ReadWriteByte(0x78);
    VS_SPI_ReadWriteByte(0x69);
    VS_SPI_ReadWriteByte(0x74);
    VS_SPI_ReadWriteByte(0x00);
    VS_SPI_ReadWriteByte(0x00);
    VS_SPI_ReadWriteByte(0x00);
    VS_SPI_ReadWriteByte(0x00);
    delay_1ms(100);
    VS_XDCS = 1;

    // 再次进入正弦测试并设置n值为0x44，即将正弦波的频率设置为另外的值
    VS_XDCS = 0; // 选中数据传输
    VS_SPI_ReadWriteByte(0x53);
    VS_SPI_ReadWriteByte(0xef);
    VS_SPI_ReadWriteByte(0x6e);
    VS_SPI_ReadWriteByte(0x44);
    VS_SPI_ReadWriteByte(0x00);
    VS_SPI_ReadWriteByte(0x00);
    VS_SPI_ReadWriteByte(0x00);
    VS_SPI_ReadWriteByte(0x00);
    delay_1ms(100);
    VS_XDCS = 1;
    // 退出正弦测试
    VS_XDCS = 0; // 选中数据传输
    VS_SPI_ReadWriteByte(0x45);
    VS_SPI_ReadWriteByte(0x78);
    VS_SPI_ReadWriteByte(0x69);
    VS_SPI_ReadWriteByte(0x74);
    VS_SPI_ReadWriteByte(0x00);
    VS_SPI_ReadWriteByte(0x00);
    VS_SPI_ReadWriteByte(0x00);
    VS_SPI_ReadWriteByte(0x00);
    delay_1ms(100);
    VS_XDCS = 1;
}

// ram 测试
// 返回值:RAM测试结果
// VS1003如果得到的值为0x807F，则表明完好;VS1053为0X83FF.
u16 VS_Ram_Test(void)
{
```

```
    VS_HD_Reset();

    VS_WR_Cmd(SPI_MODE, 0x0820); // 进入VS10XX的测试模式
    while (VS_DQ == 0)
        ;               // 等待DREQ为高
    VS_SPI_SpeedLow(); // 低速

    VS_XDCS = 0; // xDCS = 1, 选择VS10XX的数据接口
    VS_SPI_ReadWriteByte(0x4d);
    VS_SPI_ReadWriteByte(0xea);
    VS_SPI_ReadWriteByte(0x6d);
    VS_SPI_ReadWriteByte(0x54);
    VS_SPI_ReadWriteByte(0x00);
    VS_SPI_ReadWriteByte(0x00);
    VS_SPI_ReadWriteByte(0x00);
    VS_SPI_ReadWriteByte(0x00);
    delay_1ms(200);
    VS_XDCS = 1;
    return VS_RD_Reg(SPI_HDAT0
    ); // VS1003如果得到的值为0x807F, 则表明完好;VS1053为0X83FF.;
}

// 向VS10XX写命令
// address:命令地址
// dat:命令数据
void VS_WR_Cmd(u8 address, u16 dat)
{
    while (VS_DQ == 0)
        ;               // 等待空闲
    VS_SPI_SpeedLow(); // 低速
    VS_XDCS = 1;
    VS_XCS  = 0;
    VS_SPI_ReadWriteByte(VS_WRITE_COMMAND); // 发送VS10XX的写命令
    VS_SPI_ReadWriteByte(address);          // 地址
    VS_SPI_ReadWriteByte(dat >> 8);         // 发送高八位
    VS_SPI_ReadWriteByte(dat);              // 第八位
    VS_XCS = 1;
    VS_SPI_SpeedHigh(); // 高速
}

// 向VS10XX写数据
// dat:要写入的数据
void VS_WR_Data(u8 dat)
{
    VS_SPI_SpeedHigh(); // 高速,对VS1003B,最大值不能超过36.864/4Mhz
    VS_XDCS = 0;
    VS_SPI_ReadWriteByte(dat);
    VS_XDCS = 1;
}

// 读VS10XX的寄存器
// address: 寄存器地址
```

```c
// 返回值：读到的值
// 注意不要用倍速读取,会出错
u16 VS_RD_Reg(u8 address)
{
    u16 temp = 0;
    while (VS_DQ == 0)
        ;              // 非等待空闲状态
    VS_SPI_SpeedLow(); // 低速    SPI降速
    VS_XDCS = 1;
    VS_XCS  = 0;
    VS_SPI_ReadWriteByte(VS_READ_COMMAND); // 发送VS10XX的读命令
    VS_SPI_ReadWriteByte(address);         // 地址
    temp = VS_SPI_ReadWriteByte(0xff);     // 读取高字节
    temp = temp << 8;
    temp += VS_SPI_ReadWriteByte(0xff); // 读取低字节
    VS_XCS = 1;
    VS_SPI_SpeedHigh(); // 高速
    return temp;
}

// 读取VS1053的RAM
// addr：RAM地址
// 返回值：读到的值
u16 VS_WRAM_Read(u16 addr)
{
    u16 res;
    VS_WR_Cmd(SPI_WRAMADDR, addr);
    res = VS_RD_Reg(SPI_WRAM);
    return res;
}

// 写VS10xx的RAM  // 移植正点原子
// addr：RAM地址
// val:要写入的值
void VS_WRAM_Write(u16 addr, u16 val)
{
    VS_WR_Cmd(SPI_WRAMADDR, addr); // 写RAM地址
    while (VS_DQ == 0)
        ;                      // 等待空闲
    VS_WR_Cmd(SPI_WRAM, val); // 写RAM值
}

// 设置播放速度（仅VS1053有效）
// t:0,1,正常速度;2,2倍速度;3,3倍速度;4,4倍速;以此类推
void VS_Set_Speed(u8 t)
{
    VS_WR_Cmd(SPI_WRAMADDR, 0X1E04); // 速度控制地址
    while (VS_DQ == 0)
        ;                    // 等待空闲
    VS_WR_Cmd(SPI_WRAM, t); // 写入播放速度
}
```

```
// FOR WAV HEAD0 :0X7761 HEAD1:0X7665
// FOR MIDI HEAD0 :other info HEAD1:0X4D54
// FOR WMA HEAD0 :data speed HEAD1:0X574D
// FOR MP3 HEAD0 :data speed HEAD1:ID
// 比特率预定值,阶层III
const u16 bitrate[2][16] = {
    {0, 8, 16, 24, 32, 40, 48, 56, 64, 80, 96, 112, 128, 144, 160, 0},
    {0, 32, 40, 48, 56, 64, 80, 96, 112, 128, 160, 192, 224, 256, 320, 0}
};
// 返回Kbps的大小
// 返回值: 得到的码率
u16 VS_Get_HeadInfo(void)
{
    unsigned int HEAD0;
    unsigned int HEAD1;
    HEAD0 = VS_RD_Reg(SPI_HDAT0);
    HEAD1 = VS_RD_Reg(SPI_HDAT1);
    // printf("(H0,H1):%x,%x\n",HEAD0,HEAD1);
    switch (HEAD1)
    {
    case 0x7665: // WAV格式
    case 0X4D54: // MIDI格式
    case 0X4154: // AAC_ADTS
    case 0X4144: // AAC_ADIF
    case 0X4D34: // AAC_MP4/M4A
    case 0X4F67: // OGG
    case 0X574D: // WMA格式
    case 0X664C: // FLAC格式
    {
        //// printf("HEAD0:%d\n",HEAD0);
        HEAD1 = HEAD0 * 2 / 25; // 相当于*8/100
        if ((HEAD1 % 10) > 5)
            return HEAD1 / 10 + 1; // 对小数点第一位四舍五入
        else
            return HEAD1 / 10;
    }
    default: // MP3格式,仅做了阶层III的表
    {
        HEAD1 >>= 3;
        HEAD1 = HEAD1 & 0x03;
        if (HEAD1 == 3)
            HEAD1 = 1;
        else
            HEAD1 = 0;
        return bitrate[HEAD1][HEAD0 >> 12];
    }
    }
}
// 得到平均字节数
// 返回值: 平均字节数速度
u32 VS_Get_ByteRate(void)
{
```

```
    return VS_WRAM_Read(0X1E05); // 平均位速
}

// 得到需要填充的数字
// 返回值:需要填充的数字
u16 VS_Get_EndFillByte(void)
{
    return VS_WRAM_Read(0X1E06); // 填充字节
}

// 发送一次音频数据
// 固定为32字节
// 返回值:0,发送成功
//    1,VS10xx不缺数据,本次数据未成功发送
u8 VS_Send_MusicData(u8* buf)
{
    u8 n;
    if (VS_DQ != 0) // 送数据给VS10XX
    {
        VS_XDCS = 0;
        for (n = 0; n < 32; n++)
        {
            VS_SPI_ReadWriteByte(buf[n]);
        }
        VS_XDCS = 1;
    }
    else
        return 1;
    return 0; // 成功发送了
}

// 切歌
// 通过此函数切歌，不会出现切换“噪声”
// 主要目的是取消当前解码过程  当解码取消后才能达到一种所谓的切歌的效果
void VS_Restart_Play(void)
{
    u16 temp;
    u16 i;
    u8 n;
    u8 vsbuf[32];

    for (n = 0; n < 32; n++)
        vsbuf[n] = 0; // 清零

    temp = VS_RD_Reg(SPI_MODE); // 读取SPI_MODE的内容
    temp |= 1 << 3;             // 设置SM_CANCEL位
    temp |= 1 << 2;             // 设置SM_LAYER12位,允许播放MP1,MP2
    VS_WR_Cmd(SPI_MODE, temp);  // 设置取消当前解码指令

    for (i = 0;
         i <
         2048;) // 发送2048个0,期间读取SM_CANCEL位.如果为0,则表示已经取消了当前解码
```

```
    {
        if (VS_Send_MusicData(vsbuf) == 0) // 每发送32个字节后检测一次
        {
            i += 32;                    // 发送了32个字节
            temp = VS_RD_Reg(SPI_MODE); // 读取SPI_MODE的内容
            if ((temp & (1 << 3)) == 0)
                break; // 成功取消了
        }
    }

    if (i < 2048) // SM_CANCEL正常
    {
        temp = VS_Get_EndFillByte() & 0xff; // 读取填充字节
        for (n = 0; n < 32; n++)
            vsbuf[n] = temp; // 填充字节放入数组
        for (i = 0; i < 2052;)
        {
            if (VS_Send_MusicData(vsbuf) == 0)
                i += 32; // 填充
        }
    }
    else
        VS_Soft_Reset(); // SM_CANCEL不成功,坏情况,需要软复位

    temp = VS_RD_Reg(SPI_HDAT0);
    temp += VS_RD_Reg(SPI_HDAT1);

    if (temp) // 软复位,还是没有成功取消,放杀手锏,硬复位
    {
        VS_HD_Reset();   // 硬复位
        VS_Soft_Reset(); // 软复位
    }
}

// 重设解码时间
void VS_Reset_DecodeTime(void)
{
    VS_WR_Cmd(SPI_DECODE_TIME, 0x0000);
    VS_WR_Cmd(SPI_DECODE_TIME, 0x0000); // 操作两次
}

// 得到mp3的播放时间n sec
// 返回值：解码时长
u16 VS_Get_DecodeTime(void)
{
    u16 dt = 0;
    dt     = VS_RD_Reg(SPI_DECODE_TIME);
    return dt;
}

// vs10xx装载patch.
// patch：patch首地址
```

```
// len: patch长度
void VS_Load_Patch(u16* patch, u16 len)
{
    u16 i;
    u16 addr, n, val;
    for (i = 0; i < len;)
    {
        addr = patch[i++];
        n    = patch[i++];
        if (n & 0x8000U) // RLE run, replicate n samples
        {
            n &= 0x7FFF;
            val = patch[i++];
            while (n--)
                VS_WR_Cmd(addr, val);
        }
        else // copy run, copy n sample
        {
            while (n--)
            {
                val = patch[i++];
                VS_WR_Cmd(addr, val);
            }
        }
    }
}

// 设定VS10XX播放的音量和高低音
// volx:音量大小(0~254)
void VS_Set_Vol(u8 volx)
{
    u16 volt = 0;           // 暂存音量值
    volt     = 254 - volx; // 取反一下,得到最大值,表示最大的表示
    volt <<= 8;
    volt += 254 - volx;       // 得到音量设置后大小
    VS_WR_Cmd(SPI_VOL, volt); // 设音量
}

// 设定高低音控制
// bfreq:低频上限频率    2~15(单位:10Hz)
// bass:低频增益         0~15(单位:1dB)
// tfreq:高频下限频率    1~15(单位:Khz)
// treble:高频增益       0~15(单位:1.5dB,小于9的时候为负数)
void VS_Set_Bass(u8 bfreq, u8 bass, u8 tfreq, u8 treble)
{
    u16 bass_set     = 0; // 暂存音调寄存器值
    signed char temp = 0;
    if (treble == 0)
        temp = 0; // 变换
    else if (treble > 8)
        temp = treble - 8;
    else
```

```
        temp = treble - 9;
    bass_set = temp & 0X0F; // 高音设定
    bass_set <<= 4;
    bass_set += tfreq & 0xf; // 高音下限频率
    bass_set <<= 4;
    bass_set += bass & 0xf; // 低音设定
    bass_set <<= 4;
    bass_set += bfreq & 0xf;       // 低音上限
    VS_WR_Cmd(SPI_BASS, bass_set); // BASS
}

// 设定音效
// eft:0,关闭;1,最小;2,中等;3,最大.
void VS_Set_Effect(u8 eft)
{
    u16 temp;
    temp = VS_RD_Reg(SPI_MODE); // 读取SPI_MODE的内容
    if (eft & 0X01)
        temp |= 1 << 4; // 设定LO
    else
        temp &= ~(1 << 5); // 取消LO
    if (eft & 0X02)
        temp |= 1 << 7; // 设定HO
    else
        temp &= ~(1 << 7);     // 取消HO
    VS_WR_Cmd(SPI_MODE, temp); // 设定模式
}

void VS_SPK_Set(u8 sw)
{
    VS_WRAM_Write(GPIO_DDR, 1 << 4);    // VS1053的GPIO4设置成输出
    VS_WRAM_Write(GPIO_ODATA, sw << 4); // 控制VS1053的GPIO4输出值(0/1)
}

//////////////////////////////////////////////////////////////////////////////
// 设置音量,音效等.
void VS_Set_All(void)
{
    VS_Set_Vol(vsset.mvol);
    VS_Set_Bass(vsset.bflimit, vsset.bass, vsset.tflimit, vsset.treble);
    VS_Set_Effect(vsset.effect);
    VS_SPK_Set(vsset.speakersw); // 控制板载喇叭状态
}
```

MP3_Player

```h
// mp3_player.h
#ifndef   MP3PLAYER_H
#define   MP3PLAYER_H

#include "__type__.h"

u8 Mp3Player_Init(void);

#endif
```

```c
// mp3_player.c
#include "mp3player.h"
#include "stc15f2k60s2.h"
#include "vs1053.h"

// ==========================================================================
// 名称: Mp3Player_Init()
// 功能: 播放器初始化
// 入口参数: 无
// 出口参数: 1:初始化成功, 0:初始化失败
// 说明:
// ==========================================================================
u8 Mp3Player_Init(void)
{
    u16 ret;

    VS_HD_Reset();       // 硬复位        ok
    ret = VS_Ram_Test(); // 存储器测试
    if (ret != 0X83FF && ret != 0x807F)
        return 0;
    VS_Set_Vol(120); // 设置音量
    VS_Sine_Test();  // 正弦波测试
    VS_HD_Reset();   // 硬复位
    VS_Soft_Reset(); // 软复位
    return 1;
}
```

音乐播放

```c
#include "mp3player.h"
#include "pff.h"
#include "sd.h"
#include "spi.h"
#include "vs1053.h"

FATFS fatfs;  // 文件系统结构体定义
u8 tbuf[512]; // 512字节 SD卡数据缓存空间

void main(void)
{
    FRESULT res;
    u16 br;
    u32 cnt = 0; // 音乐进度递增变量
    u8 mp3in;

    SP = 0X80; // 调整堆栈指向

    // init();
    io_init();
    lcd_init();
    spi_init();
    SD_Init(); // SD卡初始化

    pf_mount(&fatfs); // 初始化petit FATFS文件系统  并提取tf卡相应数据
        // 这句非常重要，它是使用所有Petit Fatfs文件系统功能的前提

    lcd_cmd(0x80);
    lcd_printf("VS1053--TEST");

    mp3in = Mp3Player_Init();

    lcd_cmd(0x90);
    if (mp3in == 1)
        lcd_printf("MP3 Init OK");
    else
        lcd_printf("MP3 Init Error");

    res = pf_open("/MUSIC/练习.mp3"); // 打开指定路径下的音乐文件名

    lcd_cmd(0x88);
    if (res == FR_OK)
        lcd_printf("Already open");

    VS_Restart_Play();      // 重启播放
    VS_Set_All();           // 设置音量等信息
    VS_Reset_DecodeTime(); // 复位解码时间
    VS_Set_Vol(220);        // 设置音量
```

```
    VS_SPI_SpeedHigh();    // SPI设置为高速

    lcd_cmd(0x98);
    lcd_printf("Play music...");

    led = 1;

    while (1)
    {
        res =
            pf_read(tbuf, 512, &br); //通过文件系统读取指定文件夹下的一音乐数据

        if ((res != FR_OK))
        {
            led = 0;
            while (1)
                ;
        }
        cnt = 0;

        do
        {
            if (VS_Send_MusicData(tbuf + cnt) == 0) //一次送32个字节音乐数据
            {
                cnt += 32;
            }
            else
            {
                led = 0;
            }
        } while (cnt < 512);
    }
    if (br != 512) // 文件结束
    {
        while (1)
            ;
    }
}
```

智慧农业使用 VS1053

```
/*
为了在“智能农业”场景下使用vs1053模块，使用杜邦线按下文的方式两两连接插针：
VS_RST <----> RL-IO
VS_REQ <----> BEEP-IO
VS_DCS <----> DS-CE
VS_CS  <----> LED-IO

引脚替换 连接
RL-IO      P52
LED-IO     P53
DS-CE      P54
BEEP-IO    P55
*/

sbit VS_RST = P5^2;         // RST
sbit VS_DQ  = P5^5;         // REQ  要读取
sbit VS_XDCS= P5^4;         // DCS
sbit VS_XCS = P5^3;         // CS

void Mp3PlayAlarm(u8 num) //播放音频文件
{
    FRESULT res;
    u16 br;
    u32 cnt = 0; //音乐进度递增变量
    u8 mp3in;

    Init_SPI();       //SPI初始化
    SD_Init();        //SD卡初始化
    pf_mount(&fatfs); //初始化petit FATFS文件系统  并提取tf卡相应数据
        //这句非常重要，它是使用所有Petit Fatfs文件系统功能的前提

    mp3in = Mp3Player_Init();
    if (mp3in == 1)
        LCD12864_WriteData('3');
    else
        LCD12864_WriteData('E');

    if (num == 1)
        res = pf_open("/MUSIC/1.mp3"); //打开指定路径下的音乐文件名
    else
        res = pf_open("/MUSIC/2.mp3");

    if (res == FR_OK)
        LCD12864_WriteData('2');

    VS_Restart_Play();      // 重启播放
    VS_Set_All();           // 设置音量等信息
    VS_Reset_DecodeTime(); // 复位解码时间
```

```
    VS_Set_Vol(220);        // 设置音量
    VS_SPI_SpeedHigh();     // SPI设置为高速

    LCD12864_WriteData('1');

    while (1)
    {
        res =
            pf_read(tbuf, 512, &br); //通过文件系统读取指定文件夹下的一音乐数据

        if ((res != FR_OK))
        {
            while (1)
                ;
        }
        cnt = 0;

        do
        {
            if (VS_Send_MusicData(tbuf + cnt) == 0) //一次送32个字节音乐数据
            {
                cnt += 32;
            }
            else
            {
            }
        } while (cnt < 512);

        if ((br != 512)) // 文件结束
        {
            break;
        }
    }
}
```