# KC3X 软件用户手册

本手册适用于 KC32C、KC35H 等 KC3X 系列模块。使用高速 I2C 从机接口（兼容常规速度 I2C），I2C 接口可以与其他 I2C 设备并联使用。完全相同 24C01 等 I2C 芯片，提供独立的 64 字节的记忆体空间，可以省略例如 24C01、93C46 等记忆芯片。I2C 通讯带有 INT 中断输出端口，用户主机可以在 INT 变化时才读取相应的数据。全部寄存器带有掉电记忆，用户主机写入的数值都可以读取寄存器后还原。非常容易进行开发，可以参考后面章节 I2C 总线标准协议开发详解。

**软件用户手册的一般规则说明:**

如果是两个字节组成 16 位的参数，则第 1 个字节为低位，第 2 个字节为高位。

如果是 4 个字节组成 32 位的参数，则第 1 个字节为低位，第 4 个字节为高位。

0xnn 表示所描述的值不确定，可能为任意值。但其值为原先约定的范围，例如指令长度为 2～137。

B7 表示位于字节的第 7 位，B6 表示位于字节的第 6 位，以此类推；B7:4 使用第 7 至第 4 位。

用户主机写入模块的 $I^2C$ 地址为 0xcc 即 11001100，读取的 $I^2C$ 地址为 0xcd 即 11001101，标准的 $I^2C$ 地址叫法为 0x66，实际是左移了一位。

寄存器长度一般为 8 位，用户主机只需要一个字节的读写即可。另外标注字节长度的寄存器，则需要多个字节读写的，应根据需要进行多字节的读写。

**KC3X 模块写寄存器示意图:**

| 开始 | 器件地址（写） | 寄存器地址 | 写数据 0 | 写数据 X（多字节时） | 结束 |
|---|---|---|---|---|---|

**KC3X 模块读寄存器示意图:**

| 开始 | 器件地址（写） | 待读取的寄存器地址 | 结束 |
|---|---|---|---|

| 开始 | 器件地址（读） | 读数据 0 | 读数据 X（多字节时） | 结束 |
|---|---|---|---|---|

先使用写的器件地址写入待读取的寄存器地址，再使用读的器件地址读入相应的数据。

在对 $I^2C$ 写入每个字节包括数据及地址时，需要接收第 9 位 ACK 位，ACK 位由 KC3X 模块输出 0。用户主机依靠 ACK 可以获得 KC3X 模块是否正常工作的信息。

在对 $I^2C$ 读取时每个字节时，需要发送第 9 位 ACK 位，ACK 位由用户主机输出 0。但最后一个字节则需要发送第 9 位 NAK 位，NAK 位由用户主机输出 1。

## ◆ SDK 开发下载地址

使用标准 8051 的 SDK，带 Windows 电脑必须编译器、编辑器、下载器，只需要一个串口就可以在开发板上直接调试程序:

http://www.hsav.com/download/kc3xm51.zip

https://gitee.com/hsav20/kc3xm51.git

https://github.com/hsav20/kc3xm51.git

深圳市酷唱科技有限公司
Hard & Soft Technology Co., LTD.
地址:深圳市宝安区西乡共乐城 F 栋 2210　　技术支持：support@HSAV.com
电话:0755-27950879　　QQ：1005231106　　业务联系：sales@HSAV.com　　2022 年 3 月 1 日

## ◆ I²C 通讯用户主机寄存器地址简表

| 地址 | 名称 | 描述 |
|---|---|---|
| 0x01 | KCM_READ_IRQ | 读中断请求寄存器，16位寄存器 |
| 0x03 | KCM_CLEAR_IRQ | 清除中断请求寄存器，16位寄存器 |
| 0x05 | KCM_POWER_ON | 用户主机上电寄存器 |
| 0x08 | KCM_VOLUME_MUTE | 音频静音及音量加减控制 |
| 0x09 | KCM_TEST_TONE | 噪音测试控制 |
| 0x0a | KCM_SRC_DETECT | 检测所有有效的音源一次 |
| 0x0d | KCM_ERROR_FLAG | 获取模块错误标志，16 位寄存器 |
| 0x18 | KCM_SRC_FORMAT | 数码信号输入格式及通道信息指示 |
| 0x1a | KCM_SRC_FREQ | 采样频率及码流率指示 |
| 0x1c | KCM_SRC_VALID | 有效的音源输入改变，16 位寄存器 |
| 0x1f | KCM_WORK_STATUS | 模块工作/运行状态指示 |

以下读写寄存器带上电记忆，每次上电会自动恢复上次关机的数值

| | | |
|---|---|---|
| 0x20 | KCM_INPUT_SOURCE | 输入音源选择 |
| 0x21 | KCM_INPUT_VIDEO | 输入视频源选择 |
| 0x23 | KCM_DYN_COMPRES | 杜比数码动态压缩，1 为打开(夜间模式)；0 为正常模式 |
| 0x24 | KCM_SPK_CONFIG | 喇叭设置 |
| 0x25 | KCM_LPF_FREQ | 超低音通道 LPF 低通滤波器频率 |
| 0x26 | KCM_HPF_FREQ | 主声道小喇叭 HPF 高通滤波器频率 |
| 0x28 | KCM_LIP_SYNC_SET | 齿音同步延迟时间，修正对画面与声音不同步 |
| 0x29 | KCM_LIP_SYNC_MAX | 齿音同步最大的延迟时间 |
| 0x2b | KCM_LISTEN_MODE | 聆听模式选择 |
| 0x2c | KCM_EQ_SELECT | 多段 EQ 均衡音效处理选择 |
| 0x2e | KCM_VOLUME_MAX | 设置音量最大值 |
| 0x2f | KCM_VOLUME_CTRL | 音量值设置 |
| 0x30 | KCM_FL_TRIM | 前置左声道微调 |
| 0x31 | KCM_FR_TRIM | 前置右声道微调 |
| 0x32 | KCM_CE_TRIM | 中置声道微调 |
| 0x33 | KCM_SW_TRIM | 超低音声道微调 |
| 0x34 | KCM_SL_TRIM | 环绕左声道微调 |
| 0x35 | KCM_SR_TRIM | 环绕右声道微调 |
| 0x36 | KCM_BL_TRIM | 后置左声道微调 |
| 0x37 | KCM_BR_TRIM | 后置右声道微调 |
| 0x38 | KCM_MIC_MIXER | 话筒声音与主声道合成比例 |
| 0x39 | KCM_MIC_VOLUME | 话筒 1 及话筒 2 音量比例 |
| 0x3a | KCM_MIC_ECHO | 话筒直达声及回声比例 |
| 0x3b | KCM_MIC_DELAY | 话筒延迟时间及重复比例 |
| 0x3c | KCM_MIC_REVERB | 话筒混响 1 及话筒混响 2 比例 |
| 0x3d | KCM_MIC_WHISTLE | 话筒啸叫声音反馈模式 |
| 0x40 | KCM_EXTR_MEMORY | 扩展给用户主机的掉电记忆空间，0x40-0x7f 共 64 字节 |

以下为多字节读写寄存器，这些寄存器不会自动增加寄存器的索引值

| | | |
|---|---|---|
| 0x80 | KCM_CUSTOM_CODE | 设置用户自定义功能寄存器 |
| 0x81 | KCM_RD_INFO | 读取模块信息寄存器 |
| 0x82 | KCM_FW_UPGRADE | 升级模块固件寄存器 |
| 0x83 | KCM_RD_RAM | 读取指定地址的 RAM 内容 |
| 0x86 | KCM_MAX_DELAY | 读取所有声道最大可用的延迟时间 |

| | | |
|---|---|---|
| 0x87 | KCM_DELAY_TIME | 设置所有声道的延迟时间 |
| 0x88 | KCM_PROGUCE_SIGNAL | 模块内部产生的信号配置 |
| 0x8b | KCM_EQ_SETUP | 多段 EQ 均衡音效处理设置 |
| 0x8c | KCM_EQ_VALUE | 多段 EQ 均衡音效处理数值 |
| 0x8d | KCM_MIC_ADJ_MAX | 话筒各种参数最大值设置 |
| 0x90 | KCM_WR_SPECTRUM | 设置频谱模式 |
| 0x91 | KCM_RD_SPECTRUM | 频谱数值读取 |
| 0x94 | KCM_WR_FLASH | 写入 512 字节 FLASH 掉电记忆空间，带偏移量及长度 |
| 0x95 | KCM_WR_GET_FLASH | 准备读取 512 字节 FLASH 掉电记忆空间，写入偏移量 |
| 0x96 | KCM_RD_FLASH | 读取 512 字节 FLASH 掉电记忆空间 |
| 0x98 | KCM_APP_COMMAND | 读取手机/远程 APP 控制指令，多字节 |
| 0xa0 | KCM_PLAY_SD_QTY | 读取 SD 卡文件总数量，共 2 字节 |
| 0xa1 | KCM_PLAY_UDISK_QTY | 读取 U 盘文件总数量，共 2 字节 |
| 0xa2 | KCM_PLAY_FILE_TIME | 读取正在播放文件的总时间，共 2 字节单位秒 |
| 0xa3 | KCM_PLAY_TIME | 读取正在播放的实际时间，共 2 字节单位秒 |
| 0xa4 | KCM_PLAY_INDEX | 读取/写入文件播放编号，共 2 字节 |
| 0xa5 | KCM_PLAY_STATE | 读取/写入文件播放状态，共 1 字节 |
| 0xa6 | KCM_PLAY_OPERATE | 读取/写入文件播放控制，共 1 字节 |
| 0xa7 | KCM_PLAY_FILE_NAME | 读取当前多媒体文件名/歌曲名，最多 32 字节 |

在 SDK 之中，定义了 typedef enum KC3X_REGISTER，可以直接#include "Kc3xType.h"使用。

**注意:（开发前应注意的事项）**

**1、在用户主机上电后，必须写入 0x01 到 KCM_POWER_ON 寄存器，然后等待 KCM_IRQ_SYSTEM_INIT 中断请求（KCM_READ_IRQ 寄存器的 B0）。**

**2、初始化动作，必须写入用户码到 KCM_CUSTOM_CODE 寄存器。**

**3、其余所有寄存器都带上电记忆自动恢复到上次关电的状态。**

**4、掉电记忆的寄存器可以随时读写，没有写入次数的限制。**

**5、与话筒有关的寄存器包括：KCM_CUSTOM_CODE 对应的标志、KCM_MIC_MIXER、KCM_MIC_VOLUME、KCM_MIC_ECHO、KCM_MIC_DELAY、KCM_MIC_REVERB、KCM_MIC_WHISTLE、KCM_EQ_SETUP、KCM_EQ_VALUE、KCM_MIC_ADJ_MAX 等多个寄存器，各种参数调节基本都是基于与最大值的比例。**

## ◆ 寄存器说明及应用例子

**※KCM_READ_IRQ 读中断请求控制，0x01 及 0x02（16 位）读写寄存器**

**※KCM_CLEAR_IRQ 清除中断请求控制，0x03 及 0x04（16 位）读写寄存器**

当用户检测到 INT 端口变低后，需要读取"KCM_READ_IRQ"寄存器用于判断所产生中断的类型。而且需要写入相应的中断类型到"KCM_CLEAR_IRQ"寄存器以清除对应的中断。

寄存器中断位说明:

| 中断位 | 名称 | 描述 |
|---|---|---|
| B0 | KCM_IRQ_SYSTEM_INIT | 模式初始化完成中断，需要写入"KCM_POWER_ON"寄存器 |
| B1 | KCM_IRQ_FORMAT_INFO | 数码信号输入格式改变中断，需要读取"KCM_SRC_FORMAT"寄存器 |
| B2 | KCM_IRQ_VOLUME | 音量调节改变中断，需要读取" KCM_VOLUME_CTRL"寄存器获取更新的音量值 |
| B3 | KCM_IRQ_SRC_VALID | 有效的音源输入改变中断，需要读取"KCM_SRC_VALID"寄存器 |
| B4 | KCM_IRQ_FIRMWARE | 固件更新，需要读取"KCM_RD_INFO"寄存器 |
| B5 | KCM_IRQ_PLAY_STATE | 多媒体文件播放改变，需要读取"KCM_PLAY_STATE"寄存器 |
| B6 | KCM_IRQ_PLAY_TIME | 多媒体播放时间改变，需要读取"KCM_PLAY_TIME"寄存器 |

| | | |
|---|---|---|
| B7 | KCM_IRQ_APP_COMMAND | 收到手机/远程APP控制指令，需要读取"KCM_APP_COMMAND"寄存器 |

Kc3xType.h 定义了 typedef enum KC3X_IRQ_TYPE。

例子:

```
if (!HAL_KCM_I2C_INT()){                                    // INT 端口变低
    BYTE gLocal_1 = MKCM_ReadRegister(KCM_READ_IRQ) ;
    MKCM_WriteRegister(KCM_CLEAR_IRQ, gLocal_1);
    if (gLocal_1 & KCM_IRQ_SYSTEM_INIT){
        // 模块上电，需要读取相应的寄存器以恢复记忆
        // 写入 KCM_CUSTOM_CODE 的值
    }
    if (gLocal_1 & KCM_IRQ_FORMAT_INFO){
        BYTE gLocal_2 = MKCM_ReadRegister(KCM_SRC_FORMAT) ;
        // 数码信号输入格式或者通道信息改变，相应的显示
    }
}
```

**※KCM_POWER_ON 用户主机上电及模块重新启动，0x05 只写寄存器**

在用户主机上电后，需要写入 0x01 到这个寄存器，然后在收到 KCM_IRQ_SYSTEM_INIT 中断请求（KCM_READ_IRQ 寄存器的 B0）后，再执行相应的初始化动作，这样可以有效地防止通讯双方不同步引起的问题。

写入 0x55 时，模块会重新启动。一般在升级模块固件后需要重新启动。

**※KCM_VOLUME_MUTE 音频静音及音量加减控制，0x08 写寄存器**

B5 为控制音量的加减；只有在 B5 为 1 时 B4 才有效；

B4 为 1 表示音量值加 1，B4 为 0 表示音量值减 1；

B1 为控制音频的静音；只有在 B1 为 1 时 B0 才有效；

B0 为控制整机音频的静音，B0=1 静音打开，这时模块的 MUTE 脚也相应变高；B0=0 静音关闭，这时模块的 MUTE 脚也相应变低。

**※KCM_TEST_TONE 噪音测试控制，0x09 写寄存器**

B4 为打开噪音测试，B2:0 为对应的通道输出，0-7 依次是 FL、 FR、 CN、 SW、 SL、 SR、 BL、 BR 通道。

例子:

```
MKCM_WriteRegister(KCM_TEST_TONE,  0x12);  // 中置声道噪音测试
MKCM_WriteRegister(KCM_TEST_TONE,  0x00);  // 关闭噪音测试，返回正常的播音模式
```

**※KCM_SRC_DETECT 检测所有有效的音源一次，0x0a 写寄存器**

寄存器写入检测的指定时间，单位为 100 毫秒。会检测所有的输入音源一次，直到指定的时间到达。完成后会收到 KCM_IRQ_SRC_VALID 中断，读取 KCM_SRC_VALID 寄存器可以获取所有有效的音源。

**※KCM_ERROR_FLAG 获取模块错误标志，0x0d 及 0x0e 读寄存器**

如果控制没有达到预期效果，或者模块声音不正常，可以查询错误标志了解出问题的原因。

B8 音频模块 B 芯片通讯不正常;

B7 音频模块 WIFI 蓝牙芯片通讯不正常;

B6 多段 EQ 均衡音效初始化不正常，可能超出最大的预设计段数;

B5 话筒音量芯片硬件初始化不正常;

B4 模块控制音量芯片硬件初始化不正常;

B3 模块 ADC/DAC 硬件初始化不正常;

B2 模块 SPDIF 接口芯片、时钟不正常;

B1:0 模块温度范围，00 正常温度，01 超级低温，10 温度偏高，11 温度太高;

**※KCM_SRC_FORMAT 数码信号输入格式及通道信息指示，0x18 只读寄存器**

| B7:4 为音源通道信息指示 | |
|---|---|
| 数值 | 说明 |
| 0x00 | 2/0 Lt/Rt Dolby Surround compatible |
| 0x01 | 1/0 C |
| 0x02 | 2/0 L/R |
| 0x03 | 3/0 L/C/R |
| 0x04 | 2/1 L/R/S |
| 0x05 | 3/1 L/C/R/S |
| 0x06 | 2/2 L/R/SL/SR |
| 0x07 | 3/2 L/C/R/ SL/SR |
| 0x08 | 3/3 L/C/R/ SL/SR /CS |
| 0x09 | 3/4 L/C/R/LS/RS/BL/BR |
| 0x0a | 2/3 L/R/LS/RS/CS |
| 0x0b | 2/4 L/R/LS/RS/BL/BR |

| B3:0 为数码音频格式 | |
|---|---|
| 数值 | 说明 |
| 0x00 | 没有信号输入 |
| 0x01 | 模拟信号输入 |
| 0x02 | PCM 信号输入 |
| 0x03 | AC3 信号输入 |
| 0x04 | AC3 HD 信号输入 |
| 0x05 | DTS 信号输入 |
| 0x06 | DTS HD 信号输入 |
| 0x07 | AAC 信号输入 |
| 0x08 | LPCM 信号输入 |
| 0x09 | HDCD 信号输入 |
| 0x0a | MP3 信号输入 |
| 其余 | 保留 |

Kc3xType.h 定义了 typedef enum KC3X_SRC_TYPE。

**※KCM_SRC_FREQ 采样频率及码流率指示，0x1a 只读寄存器**

B7:3 为音源的码流率

| 数值 | 说明 | | 数值 | 说明 | | 数值 | 说明 | | 数值 | 说明 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | 保留 | | 0x08 | 112K bps | | 0x10 | 448K bps | | 0x18 | 2560K bps |
| 0x01 | 32K bps | | 0x09 | 128K bps | | 0x11 | 512K bps | | 0x19 | 保留 |
| 0x02 | 40K bps | | 0x0a | 160K bps | | 0x12 | 640K bps | | 0x1a | 保留 |
| 0x03 | 48K bps | | 0x0b | 192K bps | | 0x13 | 896K bps | | 0x1b | 保留 |
| 0x04 | 56K bps | | 0x0c | 224K bps | | 0x14 | 1024K bps | | 0x1c | 保留 |
| 0x05 | 64K bps | | 0x0d | 256K bps | | 0x15 | 1280K bps | | 0x1d | 保留 |
| 0x06 | 80K bps | | 0x0e | 320K bps | | 0x16 | 13792K bps | | 0x1e | 保留 |
| 0x07 | 96K bps | | 0x0f | 384K bps | | 0x17 | 20480K bps | | 0x1f | 保留 |

B2:0 为音源的采样频率，000=保留; 001=192 KHz; 010=176.4KHz; 011=96KHz; 100=88.2 KHz; 101=48KHz; 110=44.1KHz; 111=其他频率;

**※KCM_SRC_VALID 有效的音源输入改变， 0x1c 及 0x1d（16 位）读寄存器**

| 位 | 名称 | 描述 |
|---|---|---|
| B0 | KCM_SRC_VALID_ANALOG | 有信号的音源输入: 模拟输入 |
| B1 | KCM_SRC_VALID_RX1 | 有信号的音源输入: 数码 1 |
| B2 | KCM_SRC_VALID_RX2 | 有信号的音源输入: 数码2 |
| B3 | KCM_SRC_VALID_RX3 | 有信号的音源输入: 数码 3 |
| B5 | KCM_SRC_VALID_SD | 有文件的音源输入: SD插入 |
| B6 | KCM_SRC_VALID_UDISK | 有文件的音源输入: U盘插入 |
| B7 | KCM_SRC_VALID_MIC | 有信号的音源输入: 话筒插入 |
| B8 | KCM_SRC_VALID_HDMI1 | 有信号的音源输入: HDMI1 |
| B9 | KCM_SRC_VALID_HDMI2 | 有信号的音源输入: HDMI2 |
| B10 | KCM_SRC_VALID_HDMI3 | 有信号的音源输入: HDMI3 |
| B12 | KCM_SRC_VALID_UCARD | 有信号的音源输入: USB 声卡 |
| B13 | KCM_SRC_VALID_E8CH | 有信号的音源输入: 外置 7.1 声道 |
| B14 | KCM_SRC_VALID_BT | 有信号的音源输入: 蓝牙音频 |
| B15 | KCM_SRC_VALID_WIFI | 有信号的音源输入: WIFI 音频 |

Kc3xType.h 定义了 typedef enum KC3X_SRC_VALID。

**※KCM_WORK_STATUS 模块工作/运行状态指示，0x1f 写寄存器**

| 位 | 名称 | 描述 |
|---|---|---|
| B0 | KCM_SET_STANDBY | 模块运行了待机状态 |
| B1 | KCM_SET_MUTE_ON | 模块打开了静音，没有任何声音输出 |
| B5 | KCM_APP_BLUETOOTH | 模块通过蓝牙与手机/电脑 APP 通讯 |
| B6 | KCM_APP_WIFI | 模块通过 WIFI 内网与手机/电脑 APP 通讯 |
| B7 | KCM_APP_SERVER | 模块通过 WIFI 互联网与服务器连接手机/电脑 APP |

**※KCM_INPUT_SOURCE 输入音源选择，0x20 写寄存器**

B6:4 为输入类型选择，B3:0 为对应的通道:

| 值 | 名称 | 描述 |
|---|---|---|
| 0 | KCM_INPUT_ANALOG | 音源选择模拟输入 |
| 1 | KCM_INPUT_DIGITAL | 音源选择数码输入，B3:0 为对应的通道：0 为 RX1， 1 为 RX2，2 为 RX3。 |
| 2 | KCM_INPUT_HDMI | 音源选择 HDMI 输入，B3:0 为对应的通道：0 为 HDMI1，1 为 HDMI2，2 为 HDMI3，4 为 HDMI-ARC。 |
| 3 | KCM_INPUT_MEDIA | 音源选择多媒体输入，B3:0 为对应输入：0为SD卡，1 为U盘、2为USB声卡，3为外置7.1声道，4为蓝牙输入。 |
| 4 | KCM_INPUT_SIGNAL | 音源选择内部产生信号，B3:0为各种信号类型：1为正弦波配合PROGUCE_SIGNAL寄存器可以产生各种不同的正弦波用于生产测试。 |

Kc3xType.h 定义了 typedef enum KC3X_INPUT_TYPE。

例子:

```
MKCM_WriteRegister(KCM_INPUT_SOURCE,  0x00);   // 选择模拟输入
MKCM_WriteRegister(KCM_INPUT_SOURCE,  0x12);   // 选择为数码 RX2
```

**※KCM_DYN_COMPRES 杜比数码动态压缩，0x23 读写寄存器**

在输入码流为杜比数码时，寄存器值为 0 表示输出没有任何压缩。100 表示打开了最大的压缩。一般应用 50 为夜间模式。

**※KCM_SPK_CONFIG 喇叭设置，0x24 读写寄存器**

B7:6 为后置喇叭，0 为没有使用、1 为小喇叭、2 为大喇叭;

B5:4 为环绕声喇叭，0 为没有使用、1 为小喇叭、2 为大喇叭;

B3:2 为中置喇叭，0 为没有使用、1 为小喇叭、2 为大喇叭;

B1 为前置喇叭，0 为小喇叭、1 为大喇叭;

B0 为超低音喇叭，0 为没有超低音、1 有超低音。

其中小喇叭表示相应的通道带高通滤波器，只输出高频信号大喇叭为全频输出。

例子:

设置前置大喇叭，中置及环绕声小喇叭，有超低音。

```
MKCM_WriteRegister(KCM_SPK_CONFIG,  0x17)
```

**※KCM_LPF_FREQ 超低音通道 LPF 低通滤波器频率，0x25 读写寄存器**

超低音的低通滤波器的高频截止频率，有效数值范围 40Hz 至 250Hz，一般推荐 70Hz。

**※KCM_HPF_FREQ 主声道小喇叭 HPF 高通滤波器频率，0x26 读写寄存器**

当选择小喇叭时，相应的声道就使用本寄存器设置的频率，为高通滤波器的低频截止频率有效数值范围 40Hz 至

250Hz，一般推荐 70Hz。

**※KCM_LIP_SYNC_SET 齿音同步延迟时间，修正画面与声音不同步，0x28 读写寄存器**

用于修正画面与声音不同步的现像，可以将所有声道的声道一起延迟输出，寄存器的值为延迟时间设置，每步为 2ms，最大时间可以从齿音同步最大的延迟时间寄存器获取。

**※KCM_LIP_SYNC_MAX 齿音同步最大的延迟时间，0x29 读写寄存器**

齿音同步最大的延迟时间获取，每步为 2ms。

**※KCM_LISTEN_MODE 聆听模式选择，0x2b 写寄存器**

B5:4 为聆听模式类型选择:

| 值 | 名称 | 描述 |
|---|---|---|
| 00 | KCM_LISTEN_STEREO | 选择为双声道立体声，B0 为 0 关闭超低音；为 1 打开超低音； |
| 01 | KCM_LISTEN_MULTI | 选择多声道源码模式，没有任何多声道算法，B0 为 0 关闭超低音；为 1 打开超低音； |
| 10 | KCM_LISTEN_SURROUND | 选择多声道模式，B1:0 为各种不同算法的多声道模式； |
| 11 | KCM_LISTEN_EFECT | 选择多声道音效，B1:0为各种不同算法的多声道音效； |

例子:

```
MKCM_WriteRegister(KCM_LISTEN_MODE,  0x01);    // 双声道立体声，打开超低音
MKCM_WriteRegister(KCM_ LISTEN_MODE,  0x10);    //多声道源码模式，没有任何多声道算法
```

**※KCM_EQ_SELECT 音效高低音音调或多段 EQ 均衡器通道选择，0x2c 读写寄存器**

0 为停止使用音效，1 至 4 分别为 4 组预置音效高低音音调或多段 EQ 均衡器。需要初始化设置 KCM_EQ_SETUP 及 KCM_EQ_VALUE 寄存器。

注意，如果话筒声道使用 EQ，则第 4 组固定用于话筒，这时 KCM_EQ_SELECT 选择为 4 会无效。

**※KCM_VOLUME_MAX 设置音量最大值，0x2e 读写寄存器**

使用指定的音量芯片节，如果不使用音量芯片则寄存器无效，音量总步数设置，推荐使用 80，表示总音量最大为 80 步。

**※KCM_VOLUME_CTRL 音量值设置，0x2f 读写寄存器**

总音量数值写入或者读出。0x00 为最小音量，最大音量与 VOLUME_MAX 对应；一般应用建议使用 VOLUME_MUTE 控制整机音量。

因为 dsp 数码音效处理时的任何值都不能大于 0dB，如果需要做增益，必须将整体的作相同的衰减，这样才能不产生过载失真。使用 VOLUME_MUTE 可以自动利用音量的芯片作增益平衡以取得最好的重放效果。

如果选择每步为 0.5dB 的 DSP 数码音量或者音量芯片每步为 0.5dB 的，需要计算步数与实际音量及各声道微调的差异，例如最大音量选择这 80，则 80 为 0dB，79 为-0.5dB，78 为-1dB，77 为-1.5dB。以此类推。

**※KCM_FL_TRIM 前置左声道微调，0x30 读写寄存器**
**※KCM_FR_TRIM 前置右声道微调，0x31 读写寄存器**
**※KCM_CE_TRIM 中置声道微调，0x32 读写寄存器**
**※KCM_SW_TRIM 超低音声道微调，0x33 读写寄存器**
**※KCM_SL_TRIM 环绕左声道微调，0x34 读写寄存器**
**※KCM_SR_TRIM 环绕右声道微调，0x35 读写寄存器**
**※KCM_BL_TRIM 后置左声道微调，0x36 读写寄存器**
**※KCM_BR_TRIM 后置右声道微调，0x37 读写寄存器**

所有声道的音量微调，在音量芯片之中调节，如果不使用音量芯片则寄存器无效。

B4 为 0 时 B3:0 的值为增益+0 至+15dB，为 1 时 B3:0 的值为衰减-0 至-15dB。

B3:0 为 dB 的数值不包括符号，0 为增益或衰减 0dB，15 为增益或衰减 15dB。

**※KCM_MIC_MIXER 话筒声音与主声道合成比例，0x38 读写寄存器**

B7:4 为话筒声音主音源合成到主声道，与 0x0f 为最大值的比例。

B3:0 为主音源合成到主声道，与 0x0f 为最大值的比例。

注意,使用话筒必须打开 KCM_CUSTOM_CODE 对应的标志;话筒 EQ 或高低音控制参见 KCM_EQ_SETUP 及 KCM_EQ_VALUE 寄存器；各种比例最大值参见 KCM_MIC_ADJ_MAX 寄存器。

**※KCM_MIC_VOLUME 话筒 1 及话筒 2 音量比例，0x39 读写寄存器**

B7:4 为话筒 1 音量，与 KCM_MIC_ADJ_MAX 对应的最大值比例，最小为关闭音量;

B3:0 为话筒 2 音量，与 KCM_MIC_ADJ_MAX 对应的最大值比例，最小为关闭音量。

注意，当话筒 1 及 2 音量比例都为 0，话筒合成到主声道自动关闭。

**※KCM_MIC_ECHO 话筒直达声及回声比例，0x3a 读写寄存器**

B7:4 为话筒直达声合成比例，与 KCM_MIC_ADJ_MAX 对应的最大值比例，最小为 0 关闭直达声的合成;

B3:0 为话筒回声合成比例，与 KCM_MIC_ADJ_MAX 对应的最大值比例，最小为 0 关闭回声的合成。

**※KCM_MIC_DELAY 话筒延迟时间及重复比例，0x3b 读写寄存器**

B7:4 为话筒延迟时间，与 KCM_MIC_ADJ_MAX 对应的最大值比例。最大实际延迟时间为 300 毫秒;

B3:0 为话筒重复从延时返回输入合成，与 KCM_MIC_ADJ_MAX 对应的最大值比例，最小为 0 关闭重复的合成。

**※KCM_MIC_REVERB 话筒混响 1 及混响 2 比例，0x3c 读写寄存器**

B7:4 为话筒混响 1 合成的比例，最大为 100%，最小为关闭混响 1 的合成;

B3:0 为话筒混响 2 合成的比例，最大为 100%，最小为关闭混响 2 的合成。

**※KCM_MIC_WHISTLE 话筒啸叫声音反馈模式，0x3d 读写寄存器**

0 为话筒啸叫声音反馈模式关闭。

**※KCM_CUSTOM_CODE 设置用户自定义功能寄存器，0x80 读写寄存器**

一般应用，当有些用户自定义的功能时使用:

共 4 个字节，每个客户型号都不相同，演示版本为 0x12 0x12 0x00 0x00。读取时，4 个字节与写入的完全相同。

字节 0 及字节 1 为客户型号，每个客户型号都不相同，没有对应的型号可以打不开后面的设置。

| | | |
|---|---|---|
| 字节 2 | B2:0 | 互换输出声道; |
| | B3 | 5.1 的系统之中使用 7.1 功能，额外多了后置的左右声道; |
| | B6:4 | 为设置音量芯片类型。0 为不使用模块内部的音量; 1 为使用 DSP 数码音量，每步 1dB; 2 为使用 DSP 数码音量，每步 0.5dB; 3 为使用 PT2258 + PT2259 或者兼容的音量芯片;4 为使用 M62446 或者兼容的音量芯片;5 为使用 CS3318 或者兼容的音量芯片; |
| | B7 | 每个输入通道单独记忆聆听模式及多段 EQ 均衡音效选择; |
| 字节 3 | B2:0 | 保留为 0 |
| | B3 | 话筒 MIC 与模拟输入交换; |
| | B4 | 允许话筒声音混合到主声道，必须打开这个标志才能使用话筒功能; |
| | B5 | 数码输入输出时钟输入输出选择: 0 为 BCK 及 WCK 为输出; 1 为 BCK 及 WCK 为输入; |
| | B7:6 | 数码输入输出格式数据选择: 0 为标准 I2S，数据延迟 1 位;1 为左对齐; 2 为右对齐; 3 为 PCM 格式; |

**※KCM_RD_INFO 读取模块信息寄存器，0x81 只读寄存器**

读取 10 字节的模块信息，可以在固件升级时确认升级成功。

字节 0，升级百分比，小于 100 为忙，100 为成功，104 为出错。

字节 1，模块型号：0x31 为 KC3X 模块；0x53 为 KC35H。

字节 2 至 3 为 16 位的固件时间 time：时=time/1800；分=(time%1800)/30；秒=(time%1800) %30。

字节 4 至 5 为 16 位的固件日期 date：年=(time/372)+2010；月=(time%372)/31+1；日=(time%372)%31+1。

字节 6 至 9 为 32 位的版本号 version(a.b.c)：a=version/1000000；b=version%1000000/1000；c= version%1000000%1000。

**※KCM_FW_UPGRADE 升级模块固件寄存器，0x82 读写寄存器**

高级应用，当需要通过主板升级模块的固件时使用：

步骤 1、本寄存器写入升级文件的前面 16 字节。

步骤 2、循环读取本寄存器 1 字节，返回 6(正确，继续下一步)、7(文件类型出错)。

步骤 3、本寄存器写入整个升级文件的所有字节。

步骤 4、循环读取本寄存器 1 字节，返回 0(正确) 、1(文件校验出错) 、2(文件长度出错)。

步骤 5、KCM_POWER_ON 寄存器写入 0x55 重启模块。

步骤 6、收到中断 KCM_IRQ_SYSTEM_INIT 后读取 KCM_RD_INFO，对比字节 1 至 8 与文件的字节 12 至 19 相同表示已经使用了新的固件。

**※KCM_RD_RAM 读取指定地址的 RAM 内容，0x83 只读寄存器**

写入 4 字节指定的 RAM 地址后，可以读取任意长度从指定地址开始的 RAM 内容。可以用于升级固件时校验。

**※KCM_MAX_DELAY 读取所有声道最大可用的延迟时间，0x86 只读寄存器**

字节 0 至字节 7 分别是前置左、前置右、中置、超低音、环绕左、环绕右、后置左、后置右声道声道的最大可以延迟时间，单位毫秒。

**※KCM_DELAY_TIME 设置所有声道的延迟时间，0x87 读写寄存器**

字节 0 至字节 7 分别是前置左、前置右、中置、超低音、环绕左、环绕右、后置左、后置右声道声道的最大可以延迟时间，单位毫秒。

**※KCM_PROGUCE_SIGNAL 模块内部产生的信号配置，0x88 读写寄存器**

字节 0 频率。

**※KCM_EQ_SETUP 多路均衡 EQ 音效处理设置，0x8b 读写寄存器**

内置 4 组预置均衡 EQ 音效音效记忆，每组可以分别调节不同的音效值，而且独立记忆，使用者只需要选择 KCM_EQ_SELECT 寄存器，即调用相应已记忆的均衡 EQ 音效。用户整机只需要一个简单的音效选择按键，就可以将原来调节的均衡 EQ 音效调出来了，无需写大量的寄存器。

| 字节 | 作用 | 说明 |
|---|---|---|
| 字节 0 | 选择 4 组预置音效位置 | 1 至 4 分别为 4 组预置音效。如果话筒声道使用 EQ，则第 4 组固定用于话筒，这时 KCM_EQ_SELECT 选择为 4 会无效。 |
| 字节 1 | 使用 EQ 的声道标志 | 对应的通道标志为 1 则使用均衡，0 则停止使用均衡<br>B7 后置右声道使用 EQ;<br>B6 后置左声道使用 EQ;<br>B5 环绕声右声道使用 EQ;<br>B4 环绕声左声道使用 EQ;<br>B3 保留为 0; |

| | | |
|---|---|---|
| | | B2 中置声道使用 EQ;<br>B1 前置右声道使用 EQ;<br>B0 前置左声道使用 EQ; |
| 字节 2 | 主声道滤波器模式及各个通道段数量 | B7: 6 为主声道滤波器模式;<br>B5 为不使用自动的衰减值配合音量芯片;<br>B4: 0 为主声道 EQ 段数量, |
| 字节 3 | 话筒声道滤波器模式及 EQ 段数量 | B7: 6 为话筒声道滤波器模式;<br>B5 为不使用自动的衰减值配合音量芯片;<br>B4: 0 为话筒 EQ 段数量; |
| 字节 4 | 滤波器 Q 值，数值越大滤波特性越尖 | B7: 4 为话筒声道滤波器 Q 值, 0-15;<br>B3: 0 为主声道滤波器 Q 值, 0-15; |
| 字节 5 | EQ 预衰减值设置 | B7: 6 为话筒声道预衰减值;<br>B3: 0 为主声道预衰减值;<br>值 0 为不衰减。其余值为对应衰减的 dB 值。 |

注意，每个主声道数*EQ 段数+话筒不能超出 EQ 总段数。例如前置左右声道分别用 7 段的 EQ，就已经用掉了 14 段 EQ，如果话筒想再使用 3 段 EQ，一共就是 17，已经超出总段数是 16，所以只能用两段 EQ 了。

因为数码音效处理时的任何值都不能大于 0dB，如果需要做增益，必须将整体的作相同的衰减，这样才能不产生过载失真。设置的衰减值应该为所有段的最大增益值。

当选择为两段均衡 EQ 音效时，自动转换为音调调节，只调节低音及高音。

当滤波器模式选择 B7: 6 为 00 时所有段滤波器相同，B7 则最高段使用 HSF 滤波器，能通过所有高于频率设定值的信号，与音调的高音作用一样；B6 为 1 则最低段使用 LSF 滤波器，能通过所有低于频率设定值的信号，与音调的低音作用一样。

**※KCM_EQ_VALUE 多路段均衡 EQ 音效处理数值，0x8c 读写寄存器**

字节 0 选择预置音效的位置，1 至 4 分别为 4 组预置音效位置。如果打开话筒声道 EQ 音效则第 4 组为话筒的音效。1 至 4 分别为 4 组预置音效。注意这里只是修改对应的数值，未必会修改即时音效，需要 KCM_EQ_SELECT 选中的音效位置才会改变即时音效;

字节 1 开始每 3 个字节为两段 12 位的 EQ 设置值，其中第一字节为第 1 段低 8 位，第二字节为第 2 段低 8 位，第三字节的 B3: 0 为第 1 段高 4 位，B7: 4 为第 2 段高 4 位;

主声道每 3 个字节为两段 12 位的 EQ 设置值，紧跟着是 MIC 声道同样的格式。

每段 12 位的格式如下:

B11: 8 为 dB 的数值不包括符号，0 为增益或衰减 0dB，15 为增益或衰减 15dB。

B7 为 0 时 B11: 8 的值为增益+0 至+15dB，为 1 时 B11: 8 的值为衰减-0 至-15dB。

B6: 5 为频率的单位范围，B4: 0 为频率计算值

| B6 至 B5 | 说明 | B4 至 B0 |
|---|---|---|
| 0 | 20 至 175Hz 频段<br>每步 5Hz | 00000 = 0 * 5 + 20 = 20Hz 至<br>11111 = 31 * 5 + 20 = 175Hz |
| 1 | 150 至 1700Hz 频段<br>每步 50Hz | 00000 = 0 * 50 + 150 = 150Hz 至<br>11111 = 31 * 50 + 150 = 1700Hz |
| 2 | 1500 至 4600Hz 频段<br>每步 100Hz | 00000 = 0 * 100 + 1500 = 1500Hz 至<br>11111 = 31 * 100 + 1500 = 4600Hz |
| 3 | 4.5 至 20KHz 频段<br>每步 500Hz | 00000 = 0 * 500 + 4500 = 4500Hz 至<br>11111 = 31 * 500 + 4500 = 20000Hz |

有效范围为 20Hz 至 20000Hz。

例子:

选择及预置音效第 1 组，前置左、前置右及中置加入 5 段均衡 EQ 音效，频率分别为 50Hz, 300 Hz, 1000Hz, 5000Hz, 15000 Hz; 增益衰减分别为+15dB, -2dB, -12dB, +5dB, +10dB; Q 值为 128; 所有段滤波器相同;

```
BYTE set[] = {
```

```
    1, 0x03, 5, 128, 0,
};
BYTE value[] = {
    1,
    0x00|0x00|6, 0x80|0x20|3, 2<<4|15,      // 50Hz, 300Hz  +15dB, -2dB
    0x80|0x20|6, 0x00|0x40|17, 5<<4|12,         // 1000Hz, 5000Hz  -12dB, +5dB,
    0x00|0xc0|21, 0x00|0x20|3, 10,          // 15000Hz,    +10dB,
};
  MKCM_WriteNByte (KCM_EQ_SETUP,  5,  set);
  MKCM_WriteNByte(KCM_EQ_VALUE,  10,  value);
  MKCM_WriteRegister(KCM_EQ_SELECT,  1)。
```

**※KCM_MIC_ADJ_MAX 话筒各种参数控制最大值设置，0x8d 读写寄存器**

字节 0 为合成及音量方式设置，B7 为加倍延迟时间选择，使用 16BIT 整数保存延迟时间；B6 保留；B5:4 话筒合成到主通道选择，0=只合成前置左右声道；1=FL+FR+CE；2=FL+FR+CE+SL+SR；3=FL+FR+CE+SL+SR+BL+BR；B3 话筒声道支持 EQ 或高低音音调控制；B2 保留；B1:0 为话筒 1 及 2 音量合成芯片类型，0=话筒 1 及 2 分别从 MICL 及 MICR 输入，全部使用数码合成，这个方式节省成本但话筒的信噪比较差；1=使用 PT2259 音量芯片合成到 MICL 输入，话筒的信噪比及动态范围比较好推荐使用。

字节 1 为话筒 1 及 2 音量比例 KCM_MIC_VOLUME 控制最大值。B7:4 为话筒 1 音量最大值，有效范围 0x01 至 0x0f。B3:0 为话筒 2 音量最大值，有效范围 0x01 至 0x0f。

字节 2 为话筒直达声及回声比例 KCM_MIC_ECHO 控制最大值，B7:4 直达声最大值，有效范围 0x01 至 0x0f，B3:0 为回声音量最大值，有效范围 0x01 至 0x0f。

字节 3 为话筒延迟时间及重复比例 KCM_MIC_DELAY 控制最大值，B7:4 为话筒延迟时间最大值，有效范围 0x01 至 0x0f，B3:0 为话筒重复比例最大值，有效范围 0x01 至 0x0f。

字节 4 为话筒混响 1 及混响 2 比例 KCM_MIC_REVERB 控制最大值，B7:4 为话筒混响 1 合成最大值，有效范围 0x01 至 0x0f。B3:0 为话筒混响 2 合成最大值，有效范围 0x01 至 0x0f。

注意，设置控制最大值用于对应的参数使用时方便以百分比表示，并不会改变实际的参数范围。只是方便根据显示的方式用百分比表示。例如当设置最大值为 9 时，表示参数可以为 0 至 9，参数为 4 表示控制的量为 50%，9 为 100%最大。

**※KCM_WR_SPECTRUM 设置频谱模式，0x90 读写寄存器**

字节 0 选择频谱方式，0 为停止使用频谱；1 为频率电平方式；2 为低速取样点方式；3 为高速取样点方式；

字节 1 至字节 3 为组成两个 12 位的显示缓冲宽度及高度像素，字节 1 的为宽度低 8 位；字节 2 的为高度低 8 位；字节 3 的 B3:0 为宽度高 4 位；字节 3 的 B7:4 为高度高 4 位。显示缓冲高度像素，只支持 8、16、32、64、128、256;，宽度可以是任意值。

如果选择频率电平方式，则从字节 4 开始每两个字节组成 16 位的的频率点，字节数量为宽度像素*2。

**※KCM_RD_SPECTRUM 频谱数值读取，0x91 读寄存器**

使用频谱后，用户主机定时（一般 50 至 100 毫秒）像素缓冲用于显示就可以了。

| 高度 | 范围 | 说明 |
|---|---|---|
| 8 | 每 3 个字节为一组，共 8 列，每组可以装载 8x8 像素点 | 字节 0 的 B2:0 为第 1 列，B5:3 为第 2 列，字节 1 的 B2:0 为第 3 列，B5:3 为第 4 列，字节 2 的 B2:0 为第 5 列，B5:3 为第 6 列，字节 0、1、2 的 B6 为第 7 列，B7 为第 8 列。 |
| 16 | 每个字节装载两列 | B3:0 为第 1 列，B7:4 为第 2 列。 |

| | | |
|---|---|---|
| 32 | 每 5 个字节为 1 组，共 8 列，每组可以装载 5x32 像素点 | 字节 0 至字节 4 的 B4:0 分别为第 1 至第 5 列，B5 为第 6 列，B6 为第 7 列，B7 为第 8 列。 |
| 64 | 每 6 个字节为一组，共 8 列，每组可以装载 6x64 像素点 | 字节 0 至字节 5 的 B5:0 分别为第 1 列至第 6 列，B6 为第 7 列，B7 为第 8 列。 |
| 128 | 每 7 个字节为一组，共 8 列，每组可以装载 7x128 像素点。 | 字节 0 至字节 6 的 B6:0 分别为第 1 列至第 7 列，B7 为第 8 列。 |
| 256 | 8 个字节装载 8 个点 | 字节与像素点完全对应 |

**※KCM_WR_FLASH 写入 512 字节 FLASH 掉电记忆空间，带偏移量及长度，0x94 写寄存器**

字节 0 至字节 1 为偏移量，字节 2 至字节 3 为写入的长度，字节 4 开始为数据。

**※KCM_WR_GET_FLASH 准备读取 512 字节 FLASH 掉电记忆空间，写入偏移量，0x95 写寄存器**

字节 0 至字节 1 为偏移量。

**※KCM_RD_FLASH 读取 512 字节 FLASH 掉电记忆空间，0x96 读寄存器**

读取由准备读取 KCM_WR_GET_FLASH 指定偏移量开始的数据。

**※KCM_APP_COMMAND 读取手机/远程 APP 控制指令，0x98 读寄存器**

读取手机/远程 APP 控制指令，多字节。

字节 0 为长度，表示后面还需要读取的长度，不包括自己。

字节 1 为寄存器索引，与 KC3X_REGISTER 定义相一致。

字节 2 为字节 n 为寄存器的实际数值，一般直接用于主机的显示。

模块内部已经实现具体控制，主机收到本寄存器用于显示就可以了。

**※KCM_PLAY_SD_QTY 读取 SD 卡多媒体文件总数量，0xa0 读寄存器**

**※KCM_PLAY_UDISK_QTY 读取 U 盘多媒体文件总数量，0xa1 读寄存器**

读取 SD 卡/U 盘多媒体文件总数量，共 2 字节。

**※KCM_PLAY_FILE_TIME 读取正在播放文件的总时间，共 2 字节单位秒，0xa2 读寄存器**

**※KCM_PLAY_TIME 读取正在播放的实际时间，共 2 字节单位秒，0xa3 读寄存器**

读取正在播放文件的总时间及实际时间，共 2 字节单位秒。

**※KCM_PLAY_INDEX 读取/写入文件播放编号，共 2 字节，0xa4 读寄存器**

读取/写入文件播放编号，共 2 字节。从 0 开始。

**※KCM_PLAY_STATUS 多媒体文件播放状态，0xa5 读写寄存器**

B2:0 播放暂停标志，写入时 001 暂停，010 播放，011 停止播放；读取时 000 已经停止，001 正在暂停，010 正在播放， 100 播放停止，需要用户发送命令播放下一个文件；

B5:4 重复播放标志，00 没有重复，01 重复当前文件，10 重复当前文件夹，11 重复所有文件；

B7:6 随机播放标志，00 没有随机播放；01 随机播放当前文件夹；10 随机播放所有文件；

注意：

1、 当收到 KCM_IRQ_PLAY_STATUS 中断读取到本寄存器的标志为停止时，需要控制播放下一个文件，模块没有自动播放下一个文件的动作。

2、 本寄存器分为几种类型的标志，相互之间有影响的，可能需要处理位才写入。一般建议在读取的数据基础上

与或对应的位后写入。

Kc3xType.h 定义了 KC3X_STATUS_ RD_STOP、KC3X_STATUS_REPEAT_ALL 等常量。

**※KCM_PLAY_ OPERATE 多媒体文件播放控制，0xa6 只写寄存器**

0x01 播放前一个文件;

0x02 播放后一个文件;

0x03 快进播放;

0x04 快退播放;

Kc3xType.h 定义了 typedef enum KC3X_PLAY_OPERATE。

**※KCM_PLAY_FILE_NAME 读取当前多媒体文件名/歌曲名，最多 32 字节，0xa7 只写寄存器**

读取当前多媒体文件名/歌曲名，最多 32 字节。

**※调节与寄存器对应关系**

一般来说，用户主机在收到上电中断后，从 I2C 相应的寄存器读取上次修改的数值，因为寄存器的数值可能不是连贯的，这时通常写两个函数与之对应，可以参考 SDK 的 kcm_sub.c 之中的方法，用查表法做转换。

1、从寄存器读取的值，调用 MSUB_FromRegister 后，转换到本机处理的值，一般只是上电做一次。

```
BYTE MSUB_FromRegister(BYTE index, BYTE value);
```

例如本机有模拟输入，数码 1 输入及数码 2 输入，对应的寄存器值为 0x30，0x00，0x01，修改表格如下所示

```
CONST_CHAR Tab_InputSwitch[] = {
 0x30, 0x01, 0x02,
}
BYTE gLocal_1 = MKCM_ReadRegister(KCM_INPUT_SRC);
gDIP_InputSource = MSUB_FromRegister(KCM_INPUT_SRC, gLocal_1);
```

gDIP_InputSource 的值就是 0 对应 0x30，1 对应 0x00，2 对应 0x01 了。连贯操作 gDIP_InputSource 就可以轻松做显示了。

2、当需要将修改的值写入寄存器时，调用 MSUB_ToRegister 转换后再写入。

```
BYTE MSUB_ToRegister(BYTE index, BYTE counter);
```

在按键处理处修改 gDIP_InputSource 的值后，如下调用转换函数

```
BYTE gLocal_1 = MSUB_ToRegister (KCM_INPUT_SRC,  gDIP_InputSource);
MKCM_WriteRegister(KCM_INPUT_SRC,  gLocal_1);
```

gLocal_1 的值返回为 0x30，  0x00 或者 0x01，符合寄存器的要求。

3、通过配合整机身功能修改 MSUB_FromRegister 及 MSUB_ToRegister 这两个转换函数，就可以在记忆与本机处理的值之间轻松转换了。

4、如果在整机设计时有些功能是不能调节的，就需要在初始化时设置为固定的，以避免不同模块时参数被改变的情况。例如，整机需要固定时所有喇叭都为小喇叭，有超低音的，可以在初始化时

```
BYTE gLocal_1 = MKCM_ReadRegister(KCM_SPK_CONFIG);
if (gLocal !=0x15) { // 如果模块的值与设置的不同
     MKCM_WriteRegister(KCM_SPIC_CONFIG, 0x15);
}
```

## ◆ I2C 总线标准协议开发详解

采用I2C总线技术可以使系统的硬件设计大大简化、系统的体积减小、可靠性提高；同时，系统的更改和扩充极为容易。

I2C标准协议之中允许有双I2C主机，但大多数应用并没有使用。为了简单起见，我们认为一个I2C总线之中只有一个I2C主机的方式来使用I2C接口。以下函数是基于这样的应用而编写的。

I2C通讯由用户主机即用户单片机系统发起，分为开始、写字节或读字节及停止等几种状态，使用普通的IO口即可以有效地模拟出I2C总线的时序，获得较好的通讯行性能。我们设计了4个函数，分别对应I2C的开始、写字节、读字节及停止等4个状态，这4个函数已经过多种不同类型的单片机系统应用，驱动了无数种不同类型的I2C设备，可靠性、稳定性及兼容性都较高，用户可以简单地将其移植到自己的单片机系统之中。

- **void MI2C_Bus_Start();**

I2C总线开始，初始化总线及产生I2C的开始信号。

- **BOOL MI2C_Bus_Write(BYTE gLocal_1);**

I2C总线写字节。将传入的8位参数输出到I2C设备，而且接收ACK信号，调用时读取返回值即可以了解I2C设备应答的是ACK或者NAK了。一般如果返回NAK，即表示对I2C设备写入失败，应该重试或作相应的提示。

- **BYTE MI2C_Bus_Read(BOOL FLocal_NAK);**

I2C读字节从I2C设备读回一个字节，而且输出ACK/NAK应答信号。参数FLocal_NAK只有最后一个字节的读取才需要设置为1，即NAK读取；其余字节都为0。I2C字节读函数需要特别注意这个应答位，因为用错应答位将不能传输有效的停止位，导致整个传输失败。

- **void MI2C_Bus_Stop();**

I2C总线停止。产生相应的停止信号及恢复I2C总线，使I2C总线进入空闲状态。

在I2C写入数据的函数之中有返回值，这个值为读取从I2C设备返回的ACK/NAK的值。在大多数的应用之中可以忽略这个返回值。事实上在我们的应用之中，从来没有理会过这个返回值。对此，在有人机对话的产品之中一般不会有问题的。如果写入到I2C设备出错，一般使用者都会再动作一次，这样就人为地重发了一次，确保写入相应的数据到I2C设备。但是，如果是机器自动控制的或者对写入数据要求较严格的，一般调用时判断返回值。如果返回NAK的则重发一次，多次重发后还是返回NAK的做错误提示。

**写I2C设备的寄存器的方法:**

```
MI2C_Bus_Start();
MI2C_Bus_Write(I2C设备写地址);
MI2C_Bus_Write(寄存器索引值);          // 可能是多个字节
MI2C_Bus_Write(寄存器内容);            // 可能是多个字节
MI2C_Bus_Stop();
```

在I2C总线开始后，先写一个字节的I2C设备地址，这个地址一般可以在I2C设备的数据手册之中可以找到；紧跟着为一个字节或多个字节的寄存器索引；最后为一个或多个字节的数值。在OTG15E之中，寄存器索引值为1个字节，数值可以从一个字节到多个字节不等，具体可参阅《OTG15X系列I2C软件用户手册》。最后要调用停止，这样就可以写入数据到I2C设备了。

**从I2C设备读取数据一般分为两种:**

一种为寄存器型读取，例如24C01、OTG15X、DA32UD等，这种设备的特点是先做一个假写的动作再读取数据；另一种为通讯型读取，例如IP72C等，这种设备的特点是用户主机无法直接指定需要读取数据的顺序，I2C设备会将数据以

队列的形式准备好，在用户主机读完一组后再传输下一组直到所有的数据被读完。

在I2C系统之中，因为很多时候I2C设备内的状态是随时变化的，所以一般需要另外增加一个INT脚，从I2C设备输出到用户主机，用户主机可以在INT变低后读取数据，这样可以减少用户主机查询的时间。这样的系统会额外需要用户主机多一个输入脚，而且一般都是在主循环之中不停检测这个INT输入脚，当发现INT变低后开始调用读取函数进行读取。

**寄存器型读取例如OTG15X的读取方法:**

```
if (!HAL_I2C_INT()){                    // 检测到I2C从机产生中断,INT为低
MI2C_Bus_Start();
MI2C_Bus_Write(I2C设备写地址索引值);
MI2C_Bus_Write(中断寄存器);          // 可能是多个字节
MI2C_Bus_Stop();
MI2C_Bus_Start();
MI2C_Bus_Write(I2C设备读地址);
gLocal_1 = MI2C_Bus_Read(0);          // 读取数据，可能是多个字节
gLocal_1 = MI2C_Bus_Read(1);          // 最后一次读取数据输入参数必须要为1
MI2C_Bus_Stop();
}
```

当用户主机检测到INT为低时，先读取中断寄存器以了解产生中断的原因；然后根据中断的标志查找对应的中断位的寄存器索引值；最后通过索引值再读取相应的寄存器数据。寄存器读取方式通常是需读取的数据是相对确定的，而且可以通过中断寄存器来指示正在变化的寄存器。这样先写入寄存器索引值后读取的方式，在一些手册中叫做假写；这时并没有任何数据写入I2C设备，只是提醒I2C设备准备相应的数据供下一步读取。

**通讯型读取例如IP72X的读取方法:**

```
if (!HAL_I2C_INT()){                    // 检测到I2C从机产生中断,INT为低
    MI2C_Bus_Start();
    MI2C_Bus_Write(I2C设备读地址);
    gLocal_1 = MI2C_Bus_Read(0);        // 读取寄存器索引值，可能是多个字节
    switch(gLocal_1){                   // 从读取到的索引值作相应的数据读取
    case 0x01:
        gLocal_1 = MI2C_Bus_Read(0); // 读取数据，可能是多个字节
        break;
    case:                               // 其他需要处理的case
    break;
    }
    MI2C_Bus_Read(1);                 // 最后一次读取数据输入参数必须要为1
    MI2C_Bus_Stop();
}
```

当用户检测到INT为低时，调用I2C总线开始，写入I2C设备的读地址，之后即可先读取寄存器索引值，再根据索引值来读取对应的数据。这样不需要再使用写地址作假写以让I2C设备根据这个假写的索引值准备数据，因而在读取时的效率较高。

在读取I2C设备数据时，开发时需要注意的就是传入到函数的参数，这个参数一般读取都为0，即应答ACK，表示接下来还有读的动作；但是，最后一个字节读取时必须为1，即应答NAK，表示这个字节为读的最后字节，跟随的将会是I2C总线停止信号。

可以下载I2C_bus.c及d文件直接编译使用。在我们的SDK软件包之中可以找到这2个文件，其中的#ifo之中为示范

的定义方式，用户可以直接在自己的头文件中定义使用。

用好这4个函数即可以方便地开发出控制I2C设备的产品。

定义三个宏的注意事项：

```
#define HAL_I2C_SCL(b) {P10=b;} //定义SCL
#define HAL_I2C_SDA(b) {P11=b;} //定义SDA输出
#define HAL_I2C_IN_SDA() (P11) //定义SDA输入
```

如果为标准8051的IO口类型，因为8051的IO口其实与I2C是相同的，即谁也没有输出高电平，高电平是依靠上拉电阻实现的，这样直接按照电平来定义即可；另外一种是带输出控制的真双向口IO口，例如PIC系列单片机，这时当需要输出低电平时将输出寄存器设置为输出，同时将输出电平置为低，当需要高电平时，仅需要将输出寄存器变为输入即可，这样就可以与标准的I2C电平相兼容，如下面的宏所示。

```
#define HAL_I2C_SCL (b) if(b){OUTA0(0);}else{OUTA0(1);OPA0(0);}}
#define HAL_I2C_SDA (b) if(b){OUTA1(0);}else{OUTA1(1);OPA0(0);}}
#define HAL_I2C_IN_SDA() (PA)
```

以上宏假设调用OUTA0(0)时，将PA0设置为输入；调用OUTA0(1)时，将PA0设置为输出。OPA0（0）为将PA0输出低电平，实际上需要SCL输出高时将PA0变为输入；在SCL输出低时将PA0变为输出，而且输出低电平。以上处理可以获得与I2C电气规格相同的效果。

I2C的函数之中调用了MI2C_100K_DELAY及MI2C_400K_DELAY两个延时函数，这两个必须在用户程式之中定义。其中100K为低速的，400K为高速的。一般的I2C设备分为高速及低速设备，主要的分别是字节之间处理的速度有差异。因为完成一个字节后可能需要时间准备下一个字节，所以需要较长的延时，但字节之间的位延时可以短一些，这样可以有效提高传输的速度。当然，如果对传输的速度没有要求，这两个函数可以定义成相同的。

请注意：延时时间可以比I2C规格长，即变慢，但不能变快。所以在刚调试系统时是先使用较大的值获得较长的延时时间，这样免除因为延时不够造成的控制失灵的情况出现。

在系统正常工作及稳定之后逐渐减少延时的值以获得更好的性能。

```
void MI2C_100K_DELAY();                 // 按照100Kbps的标准延时，以确保状态稳定
    BYTE gLocal_1;
    gLocal_1 = 10;
    while (--gLocal_1 != 0);            //这里只是复制给一个局部变量自减以达到延时的目的
    return;
}
void MI2C_400K_DELAY(){                 // 按照400Kbps的标准延时，可以有效地提高通讯的速度
    return;             // 一般的单片机这里直接返回就可以了。如果太快则需要按上面的方法延时
}
```

请注意I2C设备地址的表示方法，一般数据手册有两种写法：一种是标准的地址，使用时需要向左移一位；另外一种是已经左移了一位，这时将设备地址分为写地址及读地址，直接写入即可。OTG15X及IP72X都是采用后者，例如IP72X的地址为0x72,则在使用到读数据时地址为0x73，不需再移位。

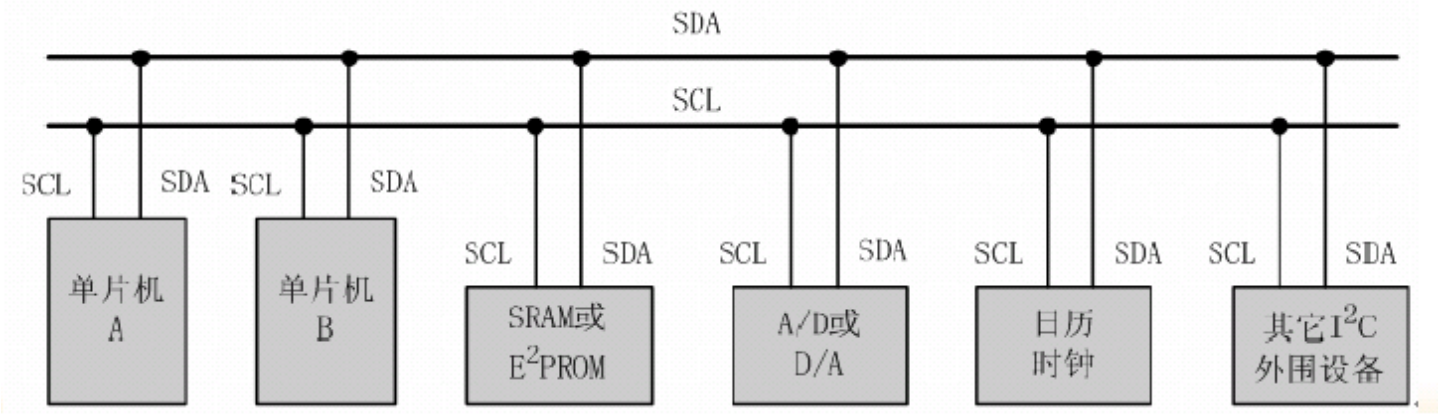
**总括上述如下：**

1. 定义MI2C_100k_DELAY及400K延时函数；
2. 定义HAL_I2C_SCL及HAL_I2C_SDA等3个宏；
3. 使用开始、读字节、 写字节及停止4个函数作I2C的读写。这样你的的系统之中即拥有了2个I2C设备控制的能力。如果需要深入了解I2C，请参阅下一节具体说明。

## ◆ I²C总线硬件的详细说明

I2C总线一般应用于单片机系统中对多个设备进行控制，最大特点是所有的I2C设备并联在一起，通讯速度为100Kbps或400Kbps。

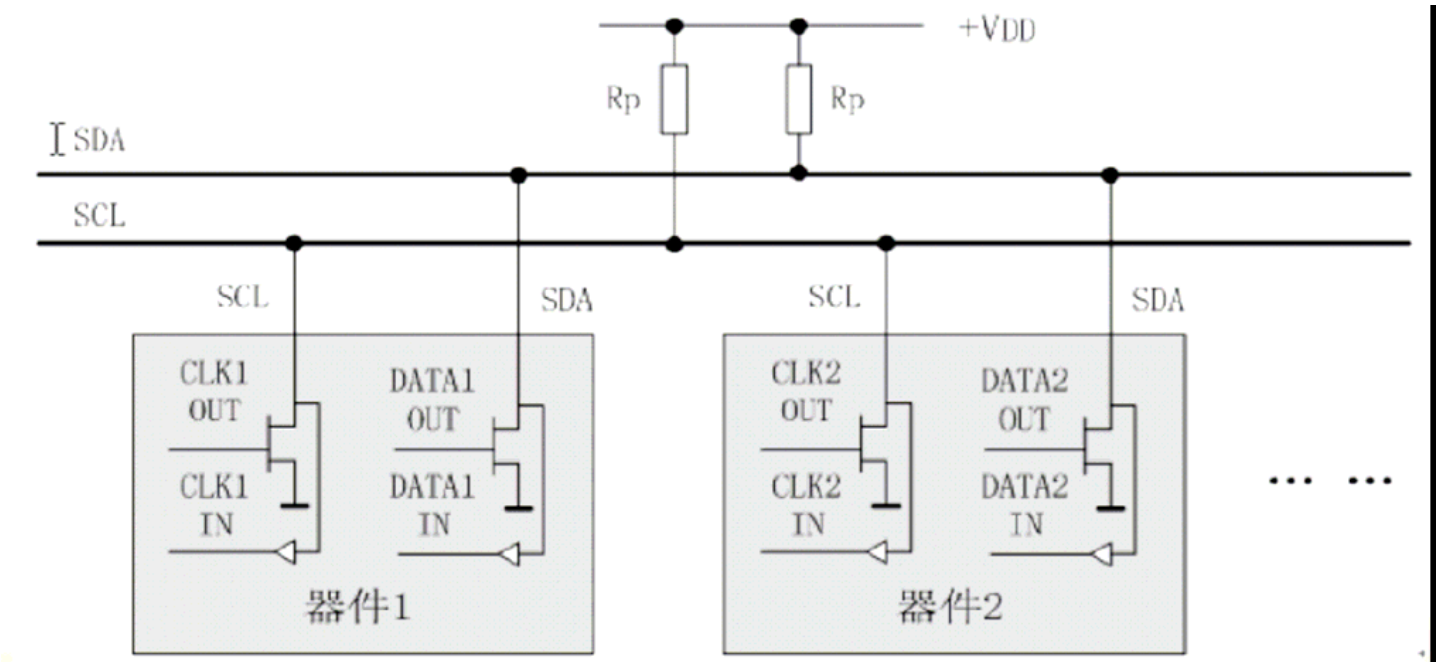
I2C总线只有两根双向信号线。一根是数据线SDA，另一根是时钟线SCL。




I2C总线通过上拉电阻接正电源。当总线空闲时，两根线均为高电平。连到总线上的任一器件输出的低电平，都将使总线的信号变低，即各器件的SDA及SCL都是线“与”关系。

每个接到I2C总线上的器件都是唯一的地址。主机与其他器件间的数据传送可以是由主机发送数据到其他器件，这时主机即为发送器。由总线上接收数据的器件则为接收器。

在多主机系统中，可能是同时有几个主机企图启动总线传送数据。为了避免混乱，I2C总线要通过总线仲裁，以决定由哪一台主机控制总线。



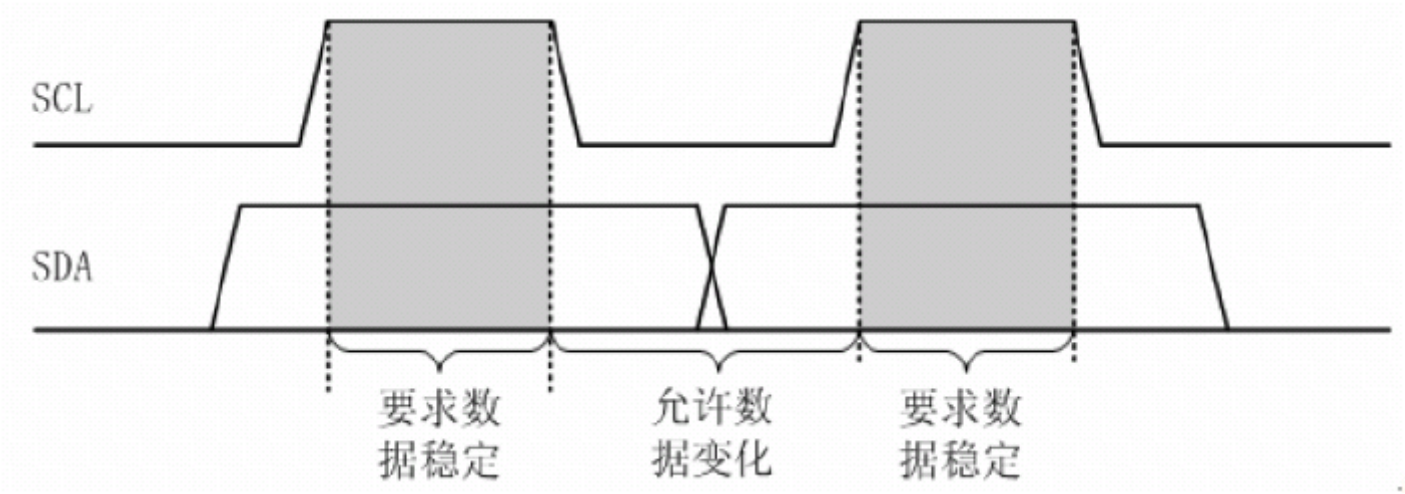

每个接到I2C总线上的器件都是唯一的地址。主机与其他器件间的数据传送可以是由主机发送数据到其他器件，这时主机即为发送器。由总线上接收数据的器件则为接收器。

在多主机系统中，可能是同时有几个主机企图启动总线传送数据。为了避免混乱，I2C总线要通过总线仲裁，以决定由哪一台主机控制总线。

## ◆ I²C总线的数据传送

### 一、数据位的有效性规定

I2C总线进行数据传送时，时钟信号为高电平期间，数据线上的数据必须保持稳定，只有在时钟线上的信号为低电平期间，数据线上的高电平或低电平状态才允许变化。




### 二、起始和终止信号

SCL线为高电平期间，SDA线由高电平向低电平的变化表示起始信号；SCL线为高电平期间，SDA线由低电平向高电平的变化表示终止信号。

起始和终止信号都是由主机发出的，在起始信号产生后，总线就处于被占用的状态；在终止信号产生后，总线就处于空闲状态。

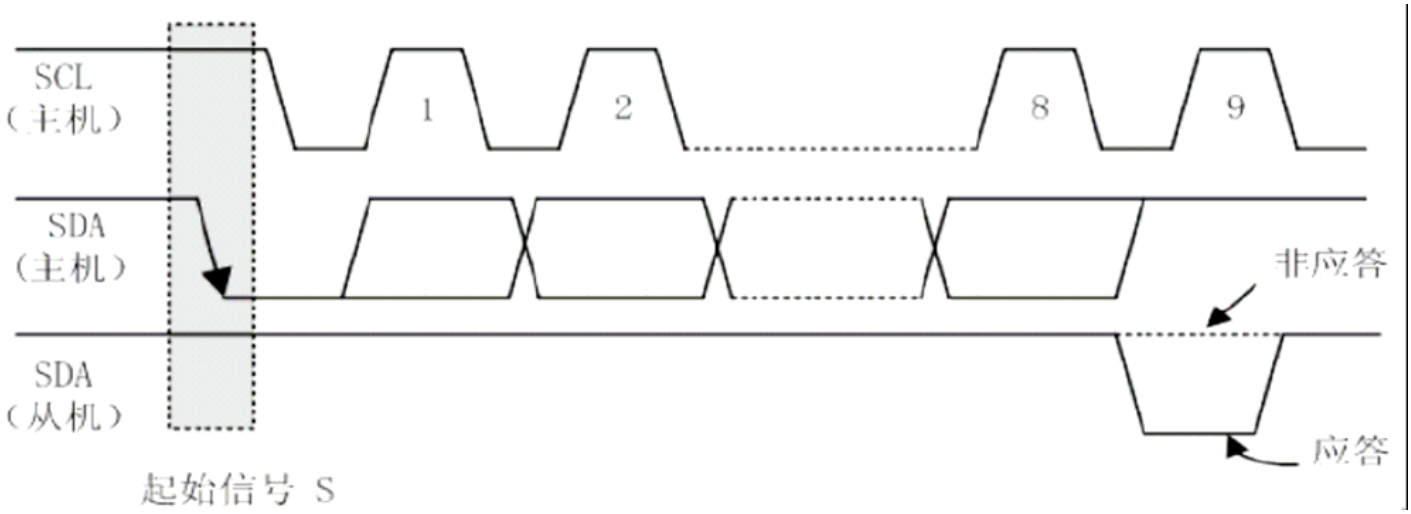
链接到I2C总线上的器件，若具有I2C总线的硬件接口，则很容易检测到起始和终止信号。对于不具备I2C总线硬件接口的有些单片机来说，为了检测起始和终止信号，必须保证在每个时钟周期内对数据线SDA采样两次。

接收器件收到一个完整的数据字节后，有可能需要完成一些其他工作，如处理内部中断服务等，可能无法立刻接收下一个字节，这时接收器件可以将SCL线拉成低电平，从而使主机处于等待状态。直到接收器件准备好接收下一个字节时，再释放SCL线使之为高电平，从而使数据传送可以继续进行。

### 三、数据传送格式

#### （1）字节传送与应答

每一个字节必须保证8位长度。数据传送时，先传送最高位（MSB），每一个被传送的字节后面都必须跟随一位应答位（即一帧共有9位）。




由于某种原因从机不对主机寻址信号应答时（如从机正在进行实时性的处理工作而无法接收总线上的数据），它必须将数据线置于高电平，而由主机产生一个终止信号以结束总线的数据传送。

如果从机对主机进行了应答，但在数据传送一段时间后无法继续接收更多的数据时，从机可以通过对无法接收的第一个数据字节的"非应答"通知主机，主机则应发出终止信号以结束数据的继续传送。

当主机接收数据时，它收到最后一个数据字节后，必须向从机发出一个结束传送的信号。这个信号是由对从机的"非应答"来实现的。然后，从机释放 SDA 线，以允许主机产生终止信号。

**（2）数据帧格式**

I2C总线上传送的数据信号是广义的，既包括地址信号，又包括真正的数据信号。

在起始信号后必须传送一个从机的地址（7位），第8位是数据的传送方向位（R/），用"0"表示主机发送数据（），"1"表示主机接收数据（R）。每次数据传送总是由主机产生的终止信号结束。但是，若主机希望继续占用总线进行新的数据传送，则可以不产生终止信号，马上再次发出起始信号对另一从机进行寻址。

在总线的一次数据传送过程中，可以有以下几种组合方式:

A， 主机向从机发送数据，数据传送方向在整个传送过程中不变:

| S | 从机地址 | 0 | A | 数据 | A | 数据 | A/$\overline{A}$ | P |
|---|---|---|---|---|---|---|---|---|

注: 有阴影部分表示数据由主机向从机传送，无阴影部分则表示数据由从机向主机传送。A表示应答， $\overline{A}$表示非应答（高电平）。S表示起始信号，P表示终止信号。

B， 主机在第一个字节后，立即由从机读数据。

| S | 从机地址 | 1 | A | 数据 | A | 数据 | $\overline{A}$ | P |
|---|---|---|---|---|---|---|---|---|

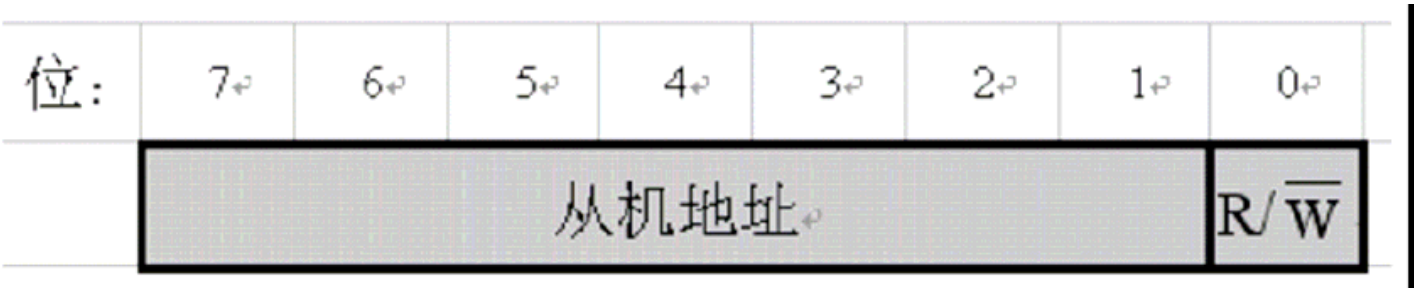
C， 在传送过程中，当需要改变传送方向时，起始信号和从机地址都被重复产生一次，但两次读/写方向正好相反。

| S | 从机地址 | 0 | A | 数据 | A/$\overline{A}$ | S | 从机地址 | 1 | A | 数据 | $\overline{A}$ | P |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

## 四、总线的寻址

I2C总线协议有明确的规定：采用7位的寻址字节（寻址字节是起始信号后的第一个字节）。

**（1）寻址字节的位定义**




D7～D1位组成从机的地址。D0位是数据传送方向位，为“0”时表示主机向从机写数据，为“1”时表示主机由从机读数据。

主机发送地址时，总线上的每个从机都将这7位地址码与自己的地址进行比较，如果相同，则认为自己正被主机寻址，根据R/位将自己确定为发送器或接收器。

从机的地址由固定部分和可编程部分组成。在一个系统中可能希望接入多个相同的从机，从机地址中可编程部分决定了可接入总线该类器件的最大数目。如一个从机的7位寻址位有4位是固定位，3位是可编程

位，这时仅能寻址8个同样的器件，即可以有8个同样的器件接入到该I2C总线系统中。

**（2）寻址字节中的特殊地址**

固定地址编号0000和1111已被保留位特殊用途。

| 地 址 位 | | | | | | | R/W | 意 义 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 通用呼叫地址 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 起始字节 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | X | CBUS 地址 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | X | 为不同总线的保留地址 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | X | 保留 |
| 0 | 0 | 0 | 0 | 1 | X | X | X | |
| 1 | 1 | 1 | 1 | 1 | X | X | X | |
| 1 | 1 | 1 | 1 | 0 | X | X | X | 十位从机地址 |

I2C总线特殊地址表

起始信号后的第一字节的8位为“0000 0000”时，称为通用呼叫地址。通用呼叫地址的用意在第二字节中加以说明。格式为：

| 第一字节（通用呼叫地址） | | | | | | | | | 第二字节 | | | | | | | LSB | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | A | X | X | X | X | X | X | X | B | A |

第二字节为06H时，所有能响应通用呼叫地址的从机器件复位，并由硬件装入从机地址的可编程部分。能响应命令的从机器件复位时不拉低SDA和SCL线，以免堵塞总线。

第二字节为04H时，所有能响应通用呼叫地址并通过硬件来定义其可编程地址的从机器件将锁定地址中的可编程位，但不进行复位。

如果第二字节的方向位B为“1”，则这两个字节命令称为硬件通用呼叫命令。

在这第二字节的高7位说明自己的地址。接在总线上的智能器件，如单片机或其他微处理器能识别这个地址，并与之传送数据。硬件主器件作为从机使用时，也用这个地址作为从机地址。格式为：

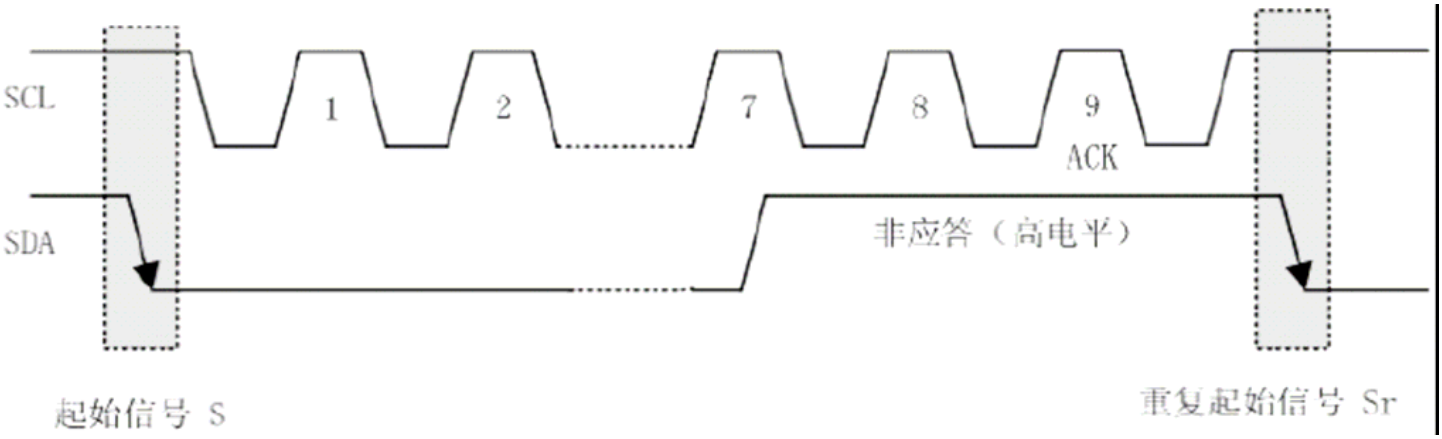
| S | 0000 0000 | A | 主机地址 | 1 | A | 数据 | A | 数据 | A | P |
|---|---|---|---|---|---|---|---|---|---|---|

在系统中另一种选择可能是系统复位时硬件主机器件工作在从机接收器方式，这时由系统中的主机先告诉硬件主机器件数据应送往的从机器件地址，当硬件主机器件要发送数据时就可以直接向指定从机器件发送数据了。

**（3）起始字节**

起始字节是提供给没有I2C总线接口的单片机查询I2C总线时使用的特殊字节。

不具备I2C总线接口的单片机，则必须通过软件不断地检测总线，以便及时地响应总线的请求。单片机的速度与硬件接口器件的速度就出现了较大的差别，为此，I2C总线上的数据传送要由一个较长的起始过程加以引导。




引导过程由起始信号、起始字节、应答位、重复起始信号（Sr）组成。

请求访问总线的主机发出起始信号后，发送起始字节（0000 0001），另一单片机可以用一个比较低的速率采样SDA线，直到检测到起始字节中的7个“0”中的一个为止。在检测到SDA线上的高电平后，单片机就可以用较高的采样速率，以便寻找作为同步信号使用的第二个起始信号Sr。

在起始信号后的应答时钟脉冲仅仅是为了和总线所使用的格式一致，并不要求器件在这个脉冲期间作应答。