网名“鱼树”的学员聂龙浩，

学习“韦东山 Linux 视频第 2 期”时所写的笔记很详细，供大家参考。

也许有错漏，请自行分辨。

# 目录

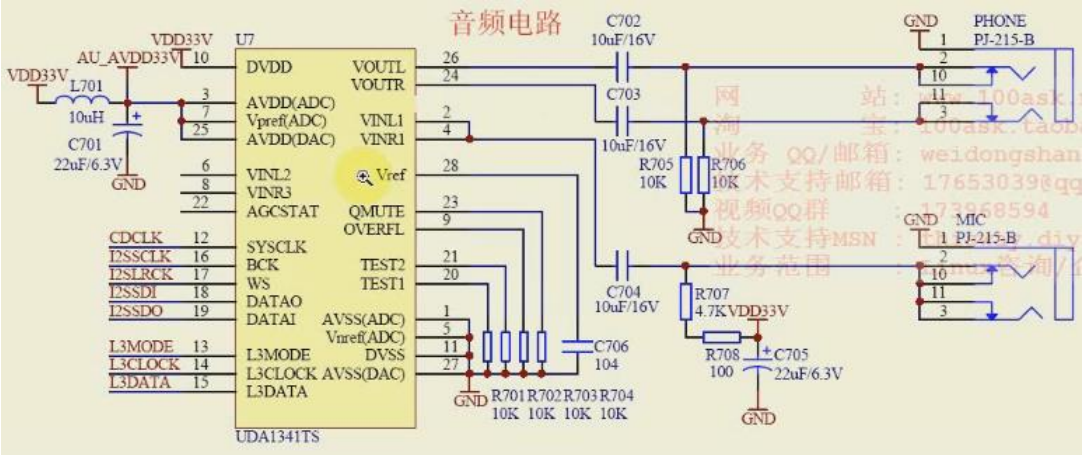


100 百问网 www.100ask.org

100 首间网
www.100ask.org

# 一、 IIS 接口：

## 1. 原理图



## 2. 硬件：

IIS 接口只传输声音数据。
GPIO 管脚发时序控制“控制接口”。（如设置音量）

### 1) 声音是如何录制：

① 采样频率：

一秒钟采集声音多少次。
以固定的时间来采集，采集就是把声音转换成电压信号并记录。
所以采集的时间点越密集，则之后恢复出来的声音会越逼真。对于人耳采集频率不需要太大。

**Table 21-1. CODEC clock (CODECLK = 256 or 384fs)**

| IISLRCK (fs) | 8.000 kHz | 11.025 kHz | 16.000 kHz | 22.050 kHz | 32.000 kHz | 44.100 kHz | 48.000 kHz | 64.000 kHz | 88.200 kHz | 96.000 kHz |
|---|---|---|---|---|---|---|---|---|---|---|
| CODECLK (MHz) | 256fs | | | | | | | | | |
| | 2.0480 | 2.8224 | 4.0960 | 5.6448 | 8.1920 | 11.2896 | 12.2880 | 16.3840 | 22.5792 | 24.5760 |
| | 384fs | | | | | | | | | |
| | 3.0720 | 4.2336 | 6.1440 | 8.4672 | 12.2880 | 16.9344 | 18.4320 | 24.5760 | 33.8688 | 36.8640 |

采样率从 8KHz - 96KHz，到 96KHz 声音已经很饱满了，人耳已经分辨不出来了。若是 4KHz（1 秒采集 4000 次）的采样率人耳听起来就很差了。

② 采集：

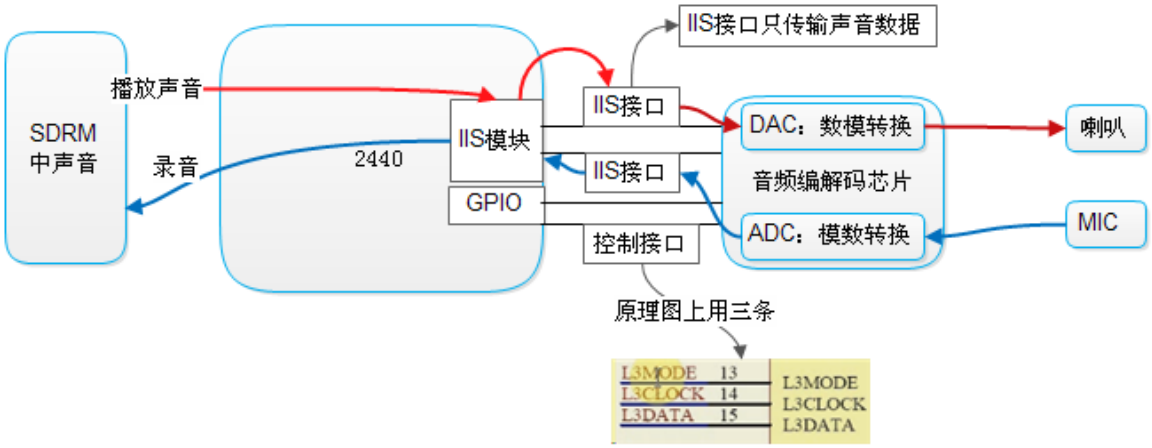
是指把声音这种模拟信号转成数字信号（ADC）存下来。有左声道和右声道，一次采集可能只是采集其一声道，也可能两个声道一起采集。

③ ADC 精度：

精度用“位宽”来表示。采集是“模数转换”，ADC 有 16 位，24 位等，位数越高声音会越精细。

### 2) 声音播放：

把采集到的声音数据按照采集的速度播放出来。



### 3) IIS：如何发数据。

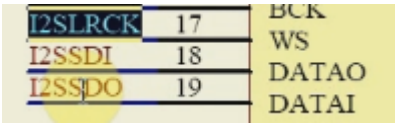
硬件接口，用来传输声音数据。

① LRCK：

表示当前传的声音的声道。

LRCK 是低电平时表示传的是“左声道”数据，LRCK 是高电平时传的是“右声道”数据。

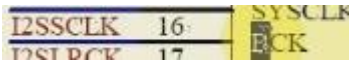
一个声道里 2440 最多传 16 个时钟（16 个 SCLK）。




原理图上“I2SLRCK”确定下面“I2SSDI、I2SSDO”传输的是左声道还是右声道的数据。

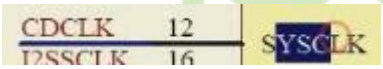
② SCLK:位时钟(BCK)。

每个数据都有很多位。每个时钟传输一个数据。



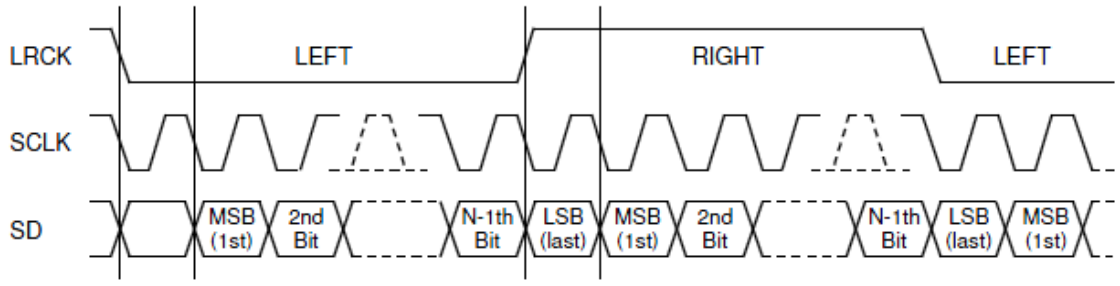

③ SDCLK：芯片的系统时钟。

是 2440 提供给解码芯片使用。UDA1341TS 解码芯片也要工作在一定的频率下。这个是提供系统时钟。




2440 控制输入两种格式数据：可以控制 2440 输出两种格式的数据“IIS、MSB”。

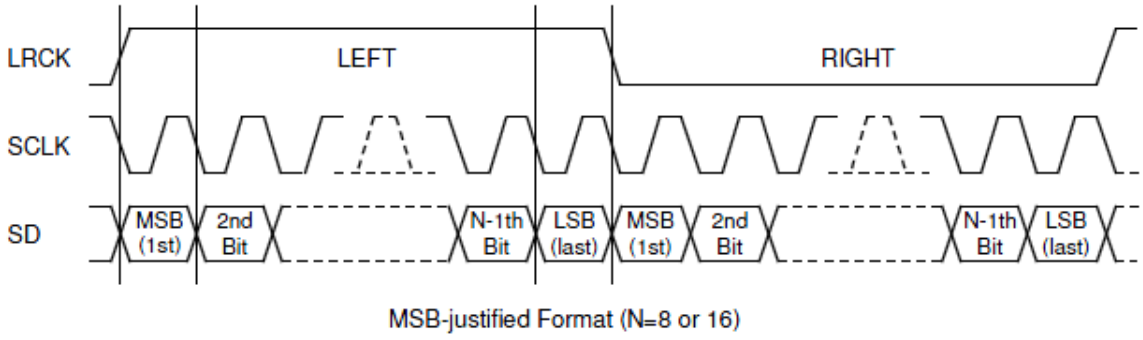
2440 只能外接最高 16 位的“编解码芯片”。




IIS-bus Format (N=8 or 16)

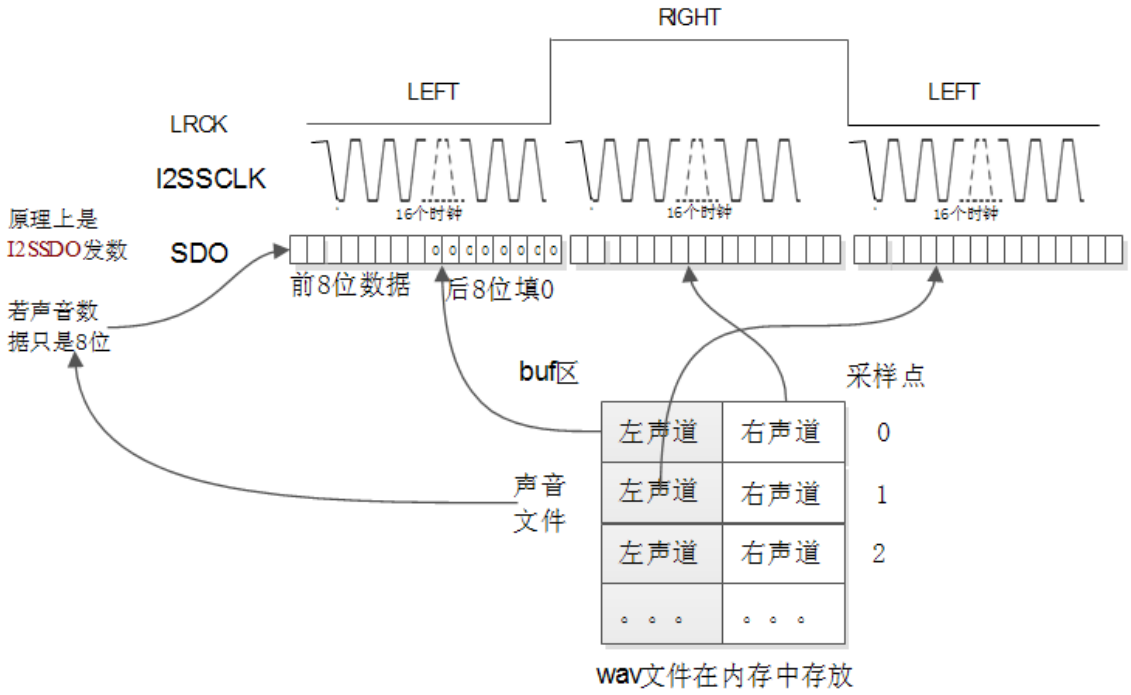
IIS 格式数据是经过一个 SCLK 位时钟后，才输入第一个数据"MSB"（SD 上空了一个 SCLK 时钟）。最后也是以“MSB”结束。

100 百问网 www.100ask.org




MSB-justified Format (N=8 or 16)

MSB 格式是在第一个 SCLK 时钟就立刻输出了第一个数据。




wav文件在内存中存放

上面是声音文件在内存中的存放，传输出来时，LRCLK 低电平期间表示左声道。一个声道 2440 传 16 个 I2SSCLK 时钟。通过“I2SSDO”每一位每一位传输，要是数据只是 8 位的，则 I2SSDO 发送完某声道（如左声道）的前 8 位后，因为是 16 个时钟，则后面 8 位填 0.

但采样频率“LRCK”是可以设置的，即若声音数据是 8 位，那么就可以设置采样频率为 8 位。采样频率是指 1

秒钟采集多少次，可以设置 2440 中的 IIS 控制器输入适合的采样频率。

## 3. WAV 声音文件:

1，头部:

a，采集频率。

b，粗度（位宽）。

c，左右声道。

2，数据

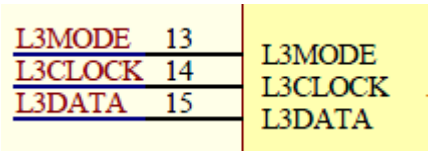
# 一、 控制信号：

## 1. 控制接口规范各不相同：

UDA1341：用的是 L3 接口。
WM8976:用 I2C 接口或 3 线接口

## 2. 读写寄存器：

控制信息的实质。
原理图上 UDA1341 用到的“控制接口”：L3 接口




### 1) L3 接口用到三条条

① L3MODE（L3 模式）：

查看"UDA1341TS"芯片手册，有“Address mode”和“Data transfer mode”。
置 0 时，地址模式
置 1 时，数据模式

100 百问网 www.100ask.org

② L3CLOCK：

每个时钟传输一位。

③ L3DATA：

当“L3MODE”是“Address mode”时，L3DTA 线上是“地址”。
当“L3MODE”是“Data transfer mode”时，L3DTA 线上是“数据”。

## 7.19 Address mode

The address mode is used to select a device for subsequent data transfer and to define the destination registers. The address mode is characterized by L3MODE being LOW and a burst of 8 pulses on L3CLOCK, accompanied by 8 data bits. The fundamental timing is shown in Fig.5.

Data bits 7 to 2 represent a 6-bit device address, with bit 7 being the MSB and bit 2 the LSB. The address of the UDA1341TS is 000101.

Data bits 0 to 1 indicate the type of the subsequent data transfer as shown in Table 4.

In the event that the UDA1341TS receives a different address, it will deselect its microcontroller interface logic.

数据有 8 位，上面说：
“bit7~2”表示一个 6bit 的地址。设置地址是“000 101”.
“bit1~0”表示传输类型。

④ 传输类型：

**Table 4** Selection of data transfer

| BIT 1 | BIT 0 | MODE | TRANSFER |
|---|---|---|---|
| 0 | 0 | DATA0 | direct addressing registers: volume, bass boost, treble, peak detection position, de-emphasis, mute and mode |
| | | | extended addressing registers: digital mixer control, AGC control, MIC sensitivity control, input gain, AGC time constant and AGC output level |
| 0 | 1 | DATA1 | peak level value read-out (information from UDA1341TS to microcontroller) |
| 1 | 0 | STATUS | reset, system clock frequency, data input format, DC-filter, input gain switch, output gain switch, polarity control, double speed and power control |
| 1 | 1 | not used | |

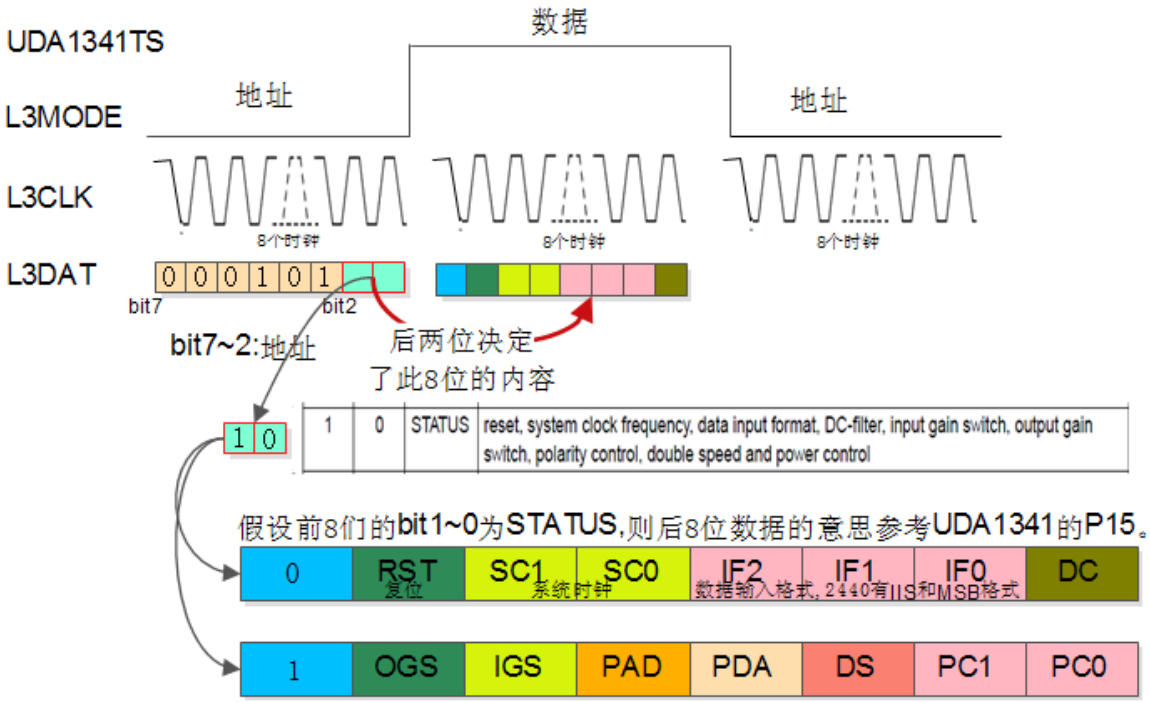
00：DATA0 控制音量。或访问扩展的地址。自动放大控制（AGC），MIC 的灵敏度控制等。
01：DATA1 读回一些信息。（即 L3 接口可以发出一些数据也可以读回一些数据）。
10：STATUS 状态信息，复位、时钟、数据输入格式（数据位宽）等。
I2S 接口上传输的是多少位的数据。2440 的 I2S 是左右声道都发出 16 个时钟，可以设置这 16 位中有多少位有效的数据。IIS 控制器发出 16 位数据，还要告诉“编解码芯片”这 16 位中有多少个数据。就用“STATUS”状态信息。

**Table 6** Data transfer of type ‘STATUS’

| BIT 7 | BIT 6 | BIT 5 | BIT 4 | BIT 3 | BIT 2 | BIT 1 | BIT 0 | REGISTER SELECTED |
|---|---|---|---|---|---|---|---|---|
| 0 | RST | SC1 | SC0 | IF2 | IF1 | IF0 | DC | RST = reset |
| | | | | | | | | SC = system clock frequency (2 bits) |
| | | | | | | | | IF = data input format (3 bits) |
| | | | | | | | | DC = DC-filter |
| 1 | OGS | IGS | PAD | PDA | DS | PC1 | PC0 | OGS = output gain (6 dB) switch |
| | | | | | | | | IGS = input gain (6 dB) switch |
| | | | | | | | | PAD = polarity of ADC |
| | | | | | | | | PDA = polarity of DAC |
| | | | | | | | | DS = double speed |
| | | | | | | | | PC = power control (2 bits) |

11：not used

⑤ 当为“STATUS”时：




当为“DATA0”时，表示是数据 0 传输：

7.21.2 DATA0 DIRECT CONTROL

Table 17 Data transfer of type 'DATA0'

| BIT 7 | BIT 6 | BIT 5 | BIT 4 | BIT 3 | BIT 2 | BIT 1 | BIT 0 | REGISTER SELECTED |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | VC5 | VC4 | VC3 | VC2 | VC1 | VC0 | VC = volume control (6 bits) 音量 |
| 0 | 1 | BB3 | BB2 | BB1 | BB0 | TR1 | TR0 | BB = bass boost (4 bits) |
| | | | | | | | | TR = treble (2 bits) |
| 1 | 0 | PP | DE1 | DE0 | MT | M1 | M0 | PP = peak detection position |
| | | | | | | | | DE = de-emphasis (2 bits) |
| | | | | | | | | MT = mute |
| | | | | | | | | M = mode switch (2 bits) |
| 1 | 1 | 0 | 0 | 0 | EA2 | EA1 | EA0 | EA = extended address (3 bits) 扩展地址 |
| 1 | 1 | 1 | ED4 | ED3 | ED2 | ED1 | ED0 | ED = extended data (5 bits) 扩展数据 |

发出哪个扩展数据后，再发出扩展数据

以上从原理图和芯片手册上大概明白了 L3 接口的意思。

# 二、 分析内核 UDA1341 驱动

看内核中自带的“UDA1341”的驱动：

```
linux-2.6.22.6\sound\soc\s3c24xx\S3c2410-uda1341.c
```

```
int __init s3c2410_uda1341_init(void)
```

```
-->driver_register(&s3c2410iis_driver); 注册平台驱动
```

```
static struct device_driver s3c2410iis_driver = {
    .name = "s3c2410-iis",
    .bus = &platform_bus_type,
    .probe = s3c2410iis_probe,
    .remove = s3c2410iis_remove,
};
```

假如内核中有同名“s3c2410-iis”的“平台设备”时，就调用这个“平台驱动”中的 probe 函数：s3c2410iis_probe

```
int s3c2410iis_probe(struct device *dev)
-->iis_base = (void *)S3C24XX_VA_IIS ; IIS模块的虚拟地址。
-->iis_clock = clk_get(dev, "iis");    使能IIS模块时钟。
   clk_enable(iis_clock);
```

-->配置 L3 接口： GPB2~4 配置成“输出引脚”。

```
/* GPB 4: L3CLOCK, OUTPUT */
s3c2410_gpio_cfgpin(S3C2410_GPB4, S3C2410_GPB4_OUTP);
s3c2410_gpio_pullup(S3C2410_GPB4,1);
/* GPB 3: L3DATA, OUTPUT */
s3c2410_gpio_cfgpin(S3C2410_GPB3,S3C2410_GPB3_OUTP);
/* GPB 2: L3MODE, OUTPUT */
s3c2410_gpio_cfgpin(S3C2410_GPB2,S3C2410_GPB2_OUTP);
s3c2410_gpio_pullup(S3C2410_GPB2,1);
```

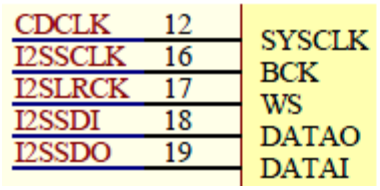

## 1. 从原理图上看 L3 引脚的连接：

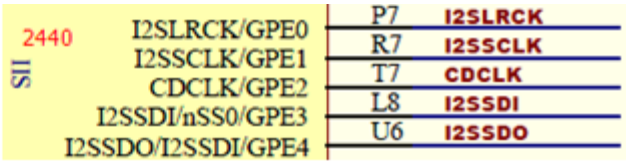
上面是 UDA1341 芯片管脚连接，和 2440 上的管脚连接

### 1) --> GPE0~4 配置成用于 IIS：

```
/* GPE 3: I2SSDI */
s3c2410_gpio_cfgpin(S3C2410_GPE3,S3C2410_GPE3_I2SSDI);
s3c2410_gpio_pullup(S3C2410_GPE3,0);
/* GPE 0: I2SLRCK */
s3c2410_gpio_cfgpin(S3C2410_GPE0,S3C2410_GPE0_I2SLRCK);
s3c2410_gpio_pullup(S3C2410_GPE0,0);
/* GPE 1: I2SSCLK */
s3c2410_gpio_cfgpin(S3C2410_GPE1,S3C2410_GPE1_I2SSCLK);
s3c2410_gpio_pullup(S3C2410_GPE1,0);
/* GPE 2: CDCLK */
```

```
s3c2410_gpio_cfgpin(S3C2410_GPE2,S3C2410_GPE2_CDCLK);
s3c2410_gpio_pullup(S3C2410_GPE2,0);
/* GPE 4: I2SSDO */
s3c2410_gpio_cfgpin(S3C2410_GPE4,S3C2410_GPE4_I2SSDO);
s3c2410_gpio_pullup(S3C2410_GPE4,0);
```




```
-->init_s3c2410_iis_bus(); 初始化总线。初始 2400 中的IIS控制器。
  -->__raw_writel(0, iis_base + S3C2410_IISPSR);
__raw_writel(0, iis_base + S3C2410_IISCON);
__raw_writel(0, iis_base + S3C2410_IISMOD);
__raw_writel(0, iis_base + S3C2410_IISFCON);
clk_disable(iis_clock);
-->init_uda1341(); 使用L3接口初始化uda1341芯片
   -->uda1341_l3_address(UDA1341_REG_STATUS); /* 地址模式 */
```

```
/* UDA1341 Register bits */
#define UDA1341_ADDR 0x14 设备地址0x14即 10100

#define UDA1341_REG_DATA0 (UDA1341_ADDR + 0)
#define UDA1341_REG_STATUS (UDA1341_ADDR + 2)
```

假如要把 UDA1341_REG_DATA0 发送出去，则是发送 0x14(0001 0100),从低到高位是“00101000”则低 2 位是数据类型 DATA0"00",设置地址是“101000”，这里是状态。

```
void uda1341_l3_address(u8 data)
     -->// write_gpio_bit(GPIO_L3MODE, 0);
    s3c2410_gpio_setpin(S3C2410_GPB2,0); 把GPB2即L3MODE引脚设置为0低电平。
 -->// write_gpio_bit(GPIO_L3CLOCK, 1);
s3c2410_gpio_setpin(S3C2410_GPB4,1); 把GPB4即L3CLOCK设置为1高电平
```

-->将形参“u8 data”数据的每一位发送出去。

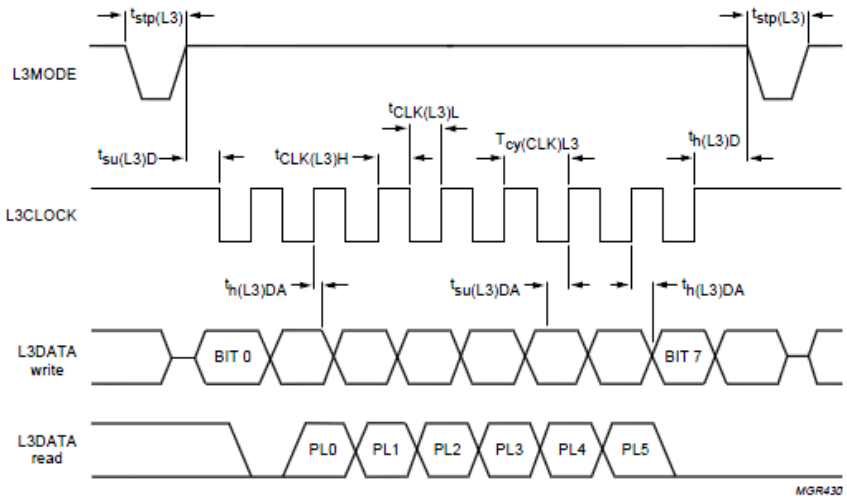



Fig.6 Timing for data transfer mode.

```
for (i = 0; i < 8; i++) {
if (data & 0x1) { 若最低位是1(与上1应该是先发出最低位)
s3c2410_gpio_setpin(S3C2410_GPB4,0); 先让时钟为低电平.
s3c2410_gpio_setpin(S3C2410_GPB3,1); 然后让数据变为1.
udelay(1);
s3c2410_gpio_setpin(S3C2410_GPB4,1);
} else {
s3c2410_gpio_setpin(S3C2410_GPB4,0); 否则时钟变为低电平
s3c2410_gpio_setpin(S3C2410_GPB3,0); 数据也变成成低电平
udelay(1);
s3c2410_gpio_setpin(S3C2410_GPB4,1); 拉高时钟线
}
data >>= 1; 接着再把数据右移一位。
}
   -->uda1341_l3_data(0x40 | STA0_SC_384FS | STA0_IF_MSB|STA0_DC_FILTER);
  发完数据类型为“STATUS”后，就开始发数据“STA0_SC_384FS - 时钟频率”、“STA0_IF_MSB--接口模式”。
```

## 2. 地址模式：

**L3MODE 低电平，L3CLK 发出 8 个时钟，L3DAT 在 8 个时钟里发出数据。**

**-->发完地址后，再发出数据：**

```
uda1341_l3_data(DATA0 |DATA0_VOLUME(0x0)); // maximum volume设置音量
uda1341_l3_data(DATA1 |DATA1_BASS(uda1341_boost)| DATA1_TREBLE(0));
uda1341_l3_data((DATA2 |DATA2_DEEMP_NONE) &~(DATA2_MUTE));
uda1341_l3_data(EXTADDR(EXT2));
uda1341_l3_data(EXTDATA(EXT2_MIC_GAIN(0x6)) | EXT2_MIXMODE_CH1);
```

1) -->设置两个 DMA 通道:

一个用来播放，一个用来录音。

```
output_stream.dma_ch = DMACH_I2S_OUT;
if (audio_init_dma(&output_stream, "UDA1341 out")) {
audio_clear_dma(&output_stream,&s3c2410iis_dma_out);
printk( KERN_WARNING AUDIO_NAME_VERBOSE
": unable to get DMA channels\n" );
return -EBUSY;
}

input_stream.dma_ch = DMACH_I2S_IN;
if (audio_init_dma(&input_stream, "UDA1341 in")) {
audio_clear_dma(&input_stream,&s3c2410iis_dma_in);
printk( KERN_WARNING AUDIO_NAME_VERBOSE
": unable to get DMA channels\n" );
return -EBUSY;
}
```

当把内存中的音频数据传到 2440 IIS 控制器时，用 DMA 来做。DMA 就是不用 CPU 来操作，在 DMA 中设置源、目的和长度后就自动把数据传给 IIS 控制器。当有数据来时，DMA 又从 IIS 控制器取出数据放到内存。

```
-->audio_dev_dsp = register_sound_dsp(&smdk2410_audio_fops, -1);
-->audio_dev_mixer = register_sound_mixer(&smdk2410_mixer_fops, -1);
```

www.100ask.org

2) 分析：audio_dev_dsp = register_sound_dsp(&smdk2410_audio_fops, -1);

```
audio_dev_dsp = register_sound_dsp(&smdk2410_audio_fops, -1);
-->sound_insert_unit()在sound_core.c核心层
     sound_insert_unit(&chains[3], fops, dev, 3, 131, "dsp", S_IWUSR |
S_IRUSR, NULL);
```

这里是把“&smdk2410_audio_fops”放到“&chains[3]”里（把 smdk2410_audio_fops 放到了 chains 链表的第三项里）

3) 分析 sound_core.c：

```
int __init init_soundcore(void)
-->register_chrdev(SOUND_MAJOR, "sound", &soundcore_fops)
```

```
static const struct file_operations soundcore_fops=
{
     /* We must have an owner or the module locking fails */
     .owner   = THIS_MODULE,
     .open    = soundcore_open,
};
```

在注册的'file_operations'结构中只有一个".open",而读写函数没有,则 open 只是跳转作用。

-->s = __look_for_unit(chain, unit); unit 是次设备号。是从某个“chains[chain]”得到结构体 sound_unit
-->然后从这个 s 结构体中得到一个新的 file_operations 结构体：其中有 read,write 等，和 LCD 的框架差不多。

```
   new_fops = fops_get(s->unit_fops);
-->device_create(sound_class, dev, MKDEV(SOUND_MAJOR, s->unit_minor), s->name+6);
```

最终会创建设备：/dev/dsp

这个设备节点做什么用，最终要看“&smdk2410_audio_fops”。因为 open 时会从链表 chain[3]中找到 smdk2410_audio_fops。把它赋值给“file->f_op”。

```
static struct file_operations smdk2410_audio_fops = {
     llseek: smdk2410_audio_llseek,
     write: smdk2410_audio_write,
     read: smdk2410_audio_read,
     poll: smdk2410_audio_poll,
     ioctl: smdk2410_audio_ioctl,
     open: smdk2410_audio_open,
     release: smdk2410_audio_release
};
```

1. 主设备号
2. file_operations
3. register_chrdev

app: open () // 假设主设备号为 14

---

soundcore_open()从这个.open 函数最后找到新的 file_operations 结构的读写。
int unit = iminor(inode); 以次设备号为下标。得到一个 sound_unit 结构.
s = __look_for_unit(chain, unit);以次设备号为下标得到一个 sound_unit 声间单元.

从 chains 数组里得到 s，谁来设置这个数组?

```
new_fops = fops_get(s->unit_fops); 若声音单元s存在得从中得到一个fops。
file->f_op = new_fops;
err = file->f_op->open(inode,file);若此新的file_operations结构有open则调用它的open。
```

录音:
app:  read
---

file->f_op->read 调用到新的 file_operations 中的读

播放:
app: write
---

file->f_op->write 调用到新的 flie_operations 中的写。
“write: smdk2410_audio_write,”：写时，将数据通过 2440 的 IIS 控制器发给音频编解码芯片。
ssize_t smdk2410_audio_write(struct file *file, const char *buffer, size_t count, loff_t * ppos)
把应用程序的数据拷贝到“const char *buffer”，最后“s3c2410_dma_enqueue（）--把 dma 放到队列”，dma 会自动把内存里的数据取出来发送给 IIS 控制器。IIS 控制器再通过 IIS 接口发给“编解码芯片”，“编解码芯片”启动 DAC 转换将数字信号转成模拟信号发送给耳机等。

4) 分析：audio_dev_mixer = register_sound_mixer(&smdk2410_mixer_fops, -1);

```
int register_sound_mixer(const struct file_operations *fops, int dev)
-->sound_insert_unit(&chains[0], fops, dev, 0, 128, "mixer", S_IRUSR | S_IWUSR, NULL);
```

将"smdk2410_mixer_fops"结构放到“chains”的第 0 项。

```
-->r = __sound_insert_unit(s, list, fops, index, low, top);
-->device_create(sound_class, dev, MKDEV(SOUND_MAJOR, s->unit_minor),s->name+6);
```

创建设备:/dev/mixer
这个设备节点做什么用得看：smdk2410_mixer_fops

```
static struct file_operations smdk2410_mixer_fops = {
    ioctl: smdk2410_mixer_ioctl,
    open: smdk2410_mixer_open,
    release: smdk2410_mixer_release
};
```

```
ioctl: smdk2410_mixer_ioctl,
int smdk2410_mixer_ioctl(struct inode *inode, struct file *file,unsigned int cmd, unsigned long arg)
-->SOUND_MIXER_WRITE_VOLUME 调整音量
-->SOUND_MIXER_READ_VOLUME 读取音量
```

等等。。

从以上的分析看，声卡驱动是字符驱动，分析比较困难是因为对声卡硬件不熟悉。

LINUX 中有两套声卡驱动程序：
ALSA（改进的 LINUX 声音架构）、OOS

```
< > Sound card support
    Advanced Linux Sound Architecture  --->
    Open Sound System  --->
```

ALSA 非常复杂。

## 3. 测试:

```
:/work/system/linux-2.6.22.6$ vi sound/soc/s3c24xx/Makefile
```

```
# S3c24XX Platform Support
snd-soc-s3c24xx-objs := s3c24xx-pcm.o
snd-soc-s3c24xx-i2s-objs := s3c24xx-i2s.o

obj-$(CONFIG_SND_S3C24XX_SOC) += snd-soc-s3c24xx.o
obj-$(CONFIG_SND_S3C24XX_SOC_I2S) += snd-soc-s3c24xx-i2s.o
obj-y += s3c2410-uda1341.o
```

1) 在 make menuconfig 中搜索：

```
Search Configuration Parameter
Enter CONFIG_ (sub)string to search for (omit CONFIG_)
SND_S3C24XX_SOC
< Ok >  < Help >
```

```
Search Results
Symbol: SND_S3C24XX_SOC [=n]
Prompt: SoC Audio for the Samsung S3C24XX chips
  Defined at sound/soc/s3c24xx/Kconfig:1
  Depends on: HAS_IOMEM && !M68K && SOUND!=n && SND!=n && ARCH_S3C2410 && SND_SOC
  Location:
    -> Device Drivers
      -> Sound
        -> Advanced Linux Sound Architecture
          -> Advanced Linux Sound Architecture (SND [=y])
            -> System on Chip audio support


Symbol: SND_S3C24XX_SOC_I2S [=y]
Prompt: I2S of the Samsung S3C24XX chips
  Defined at sound/soc/s3c24xx/Kconfig:9
  Depends on: HAS_IOMEM && !M68K && SOUND!=n && SND!=n && ARCH_S3C2410 && SND_SOC
  Location:
    -> Device Drivers
```

2) 1. 确定内核里已经配置了 sound\soc\s3c24xx\s3c2410-uda1341.c

```
-> Device Drivers
  -> Sound
    -> Advanced Linux Sound Architecture
      -> Advanced Linux Sound Architecture
        -> System on Chip audio support
          <*> I2S of the Samsung S3C24XX chips
```

3) 2. make uImage

使用新内核启动

4) 3. ls -l /dev/dsp /dev/mixer

```
# ls -l /dev/dsp /dev/mixer
crw-rw----    1 0        0          14,   3 Jan 13 16:32 /dev/dsp
crw-rw----    1 0        0          14,   0 Jan 13 16:32 /dev/mixer
#
```

5) 4. 播放:

在 WINDOWS PC 里找一个 wav 文件,放到开发板根文件系统里

```
cat Windows.wav > /dev/dsp
```

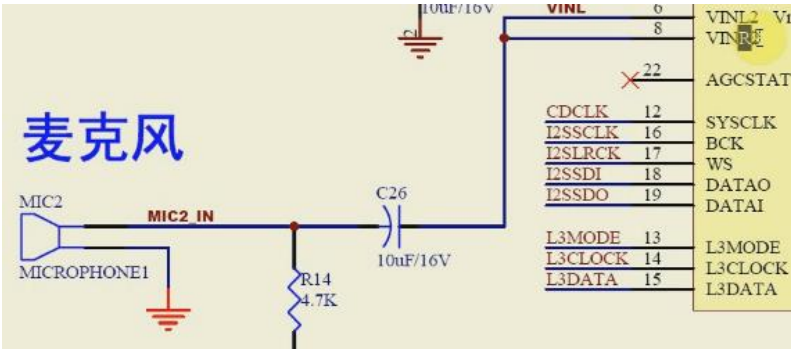
```
# cat Windows.wav > /dev/dsp
s3c2410-uda1341-superlp: audio_set_dsp_speed:44100 prescaler:66
```

6) 5. 录音:

cat /dev/dsp > sound.bin
然后对着麦克风说话
ctrl+c 退出
cat sound.bin > /dev/dsp // 就可以听到录下的声音

linux-2.6.22.6\sound\soc\s3c24xx\S3c2410-uda1341.c

mini2440 的麦克风是从通道 2 进来的（原理图）：



```
/* 对于MINI2440,  */
uda1341_l3_data(EXTDATA(EXT2_MIC_GAIN(0x6)) | EXT2_MIXMODE_CH2);//input channel 1 select(input channel 2 off)
```

EXT2_MIXMODE_CH1 是指麦克风在原理图上接的通道

1，天嵌 2440 接通道

1，而 mini2440 要改成通道 2 EXT2_MIXMODE_CH2

mini2440 修改如下：

```
#if 1
    /* 对于MINI2440,  */
    uda1341_l3_data(EXTDATA(EXT2_MIC_GAIN(0x6)) | EXT2_MIXMODE_CH2);//input channel 2 select
#else
    /* 对于TQ2440 */
    uda1341_l3_data(EXTDATA(EXT2_MIC_GAIN(0x6)) | EXT2_MIXMODE_CH1);//input channel 1 select(input channe
#endif
```

## 4. 分析 声卡 中的 DMA：

```
output_stream.dma_ch = DMACH_I2S_OUT;
if (audio_init_dma(&output_stream, "UDA1341 out")) {
    audio_clear_dma(&output_stream,&s3c2410iis_dma_out);
    printk( KERN_WARNING AUDIO_NAME_VERBOSE
            ": unable to get DMA channels\n" );
    return -EBUSY;
}
```

输出 stream 中的 DMA（是播放的意思）。

audio_init_dma（）音频初始化 DMA 函数。

```
if(s->dma_ch == DMACH_I2S_OUT){
    channel = DMACH_I2S_OUT;
    source  = S3C2410_DMASRC_MEM;
    hwcfg   = BUF_ON_APB;
    devaddr = 0x55000010;
    dcon    = S3C2410_DCON_HANDSHAKE|S3C2410_DCON_SYNC_PCLK|S3C2410_DCON_INTREQ|\
              S3C2410_DCON_TSZUNIT|S3C2410_DCON_SSERVE|S3C2410_DCON_CH2_I2SSDO|S3C2410_DCON_HWTRIG; // VAL: 0xa0800000;
    flags   = S3C2410_DMAF_AUTOSTART;

    ret = s3c2410_dma_request(s->dma_ch, &s3c2410iis_dma_out, NULL);
    s3c2410_dma_devconfig(channel, source, hwcfg, devaddr);
    s3c2410_dma_config(channel, 2, dcon);
    s3c2410_dma_set_buffdone_fn(channel, audio_dmaout_done_callback);
    s3c2410_dma_setflags(channel, flags);
    s->dma_ok = 1;
    return ret;
}
```

```
int __init audio_init_dma(audio_stream_t * s, char *desc)
s->dma_ch == DMACH_I2S_OUT
-->ret = s3c2410_dma_request(s->dma_ch, &s3c2410iis_dma_out, NULL); 2410DMA请求。
```

| | Source0 | Source1 | Source2 | Source3 | Source4 | Source5 | Source6 |
|---|---|---|---|---|---|---|---|
| Ch-0 | nXDREQ0 | UART0 | SDI | Timer | USB device EP1 | I2SSDO | PCMIN |
| Ch-1 | nXDREQ1 | UART1 | I2SSDI | SPI0 | USB device EP2 | PCMOUT | SDI |
| Ch-2 | I2SSDO | I2SSDI | SDI | Timer | USB device EP3 | PCMIN | MICIN |
| Ch-3 | UART2 | SDI | SPI1 | Timer | USB device EP4 | MICIN | PCMOUT |

从代码中可知“s3c2410_dma_request()”函数形参 1 为“s->dma_ch”即“DMACH_I2S_OUT”，从 2440 DMA 控制器中可知“I2S OUT”只有上表中的“Ch-0 I2SSDO”和“Ch-2 I2SSDO”。则就是说“播放”时只有“通道 0”或“通道 2”可以选择。“s3c2410_dma_request()”函数里，要是“Ch-0”被占用，就用“Ch-2”。若是两者都被占用则返回失败。

```
int s3c2410_dma_request(unsigned int channel,struct s3c2410_dma_client *client,void *dev)
-->chan = s3c2410_dma_map_channel(channel); DMA映射通道
-->s3c2410_dma_devconfig(channel, source, hwcfg, devaddr); 配置 DMA 。
```

等待，这些函数都是对 2440 的 DMA 配置封装好的，其中就是操作些寄存器。

最后，要 DMA 传输数据时：

```
static struct file_operations smdk2410_audio_fops = {
    llseek: smdk2410_audio_llseek,
    write: smdk2410_audio_write,
    read: smdk2410_audio_read,
    poll: smdk2410_audio_poll,
    ioctl: smdk2410_audio_ioctl,
    open: smdk2410_audio_open,
    release: smdk2410_audio_release
};
```

```
size_t smdk2410_audio_write(struct file *file, const char *buffer, size_t count, loff_t *
ppos)
-->copy_from_user(b->start + b->size, buffer, chunksize) 将数据从用户空间拷贝进来。
-->ret = s3c2410_dma_enqueue(s->dma_ch, (void *) b, b->dma_addr, b->size) “enqueue”即入队列。启动DMA操作
  -->s3c2410_dma_ctrl(chan->number | DMACH_LOW_LEVEL, S3C2410_DMAOP_START); 启动 DMA 。
    -->启动DMA
case S3C2410_DMAOP_START:
   return s3c2410_dma_start(chan);
   -->tmp |= S3C2410_DMASKTRIG_ON;  "#define S3C2410_DMASKTRIG_ON     (1<<1)"
   -->dma_wrreg(chan, S3C2410_DMA_DMASKTRIG, tmp);
```

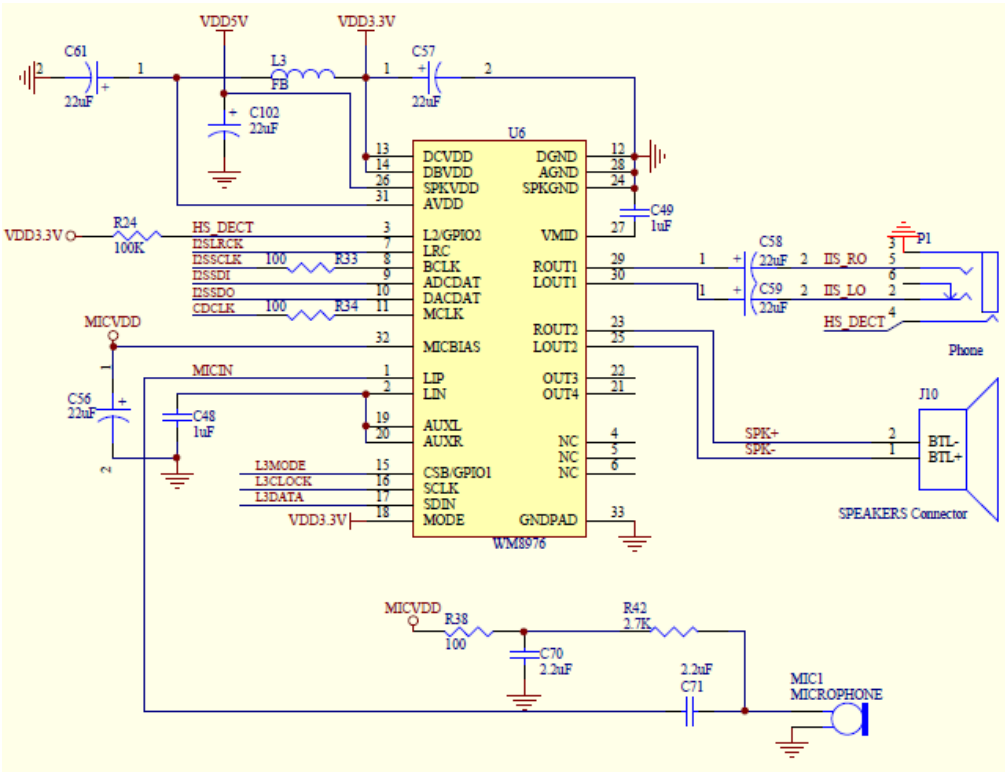
| | | | |
|---|---|---|---|
| ON_OFF | [1] | DMA channel on/off bit.<br>0: DMA channel is turned off.<br>(DMA request to this channel is ignored.)<br>1: DMA channel is turned on and the DMA request is handled. This bit is automatically set to off if we set the DCONn[22] bit to “no auto reload” and/or STOP bit of DMASKTRIGn to “stop”. Note that when DCON[22] bit is "no auto reload", this bit becomes 0 when CURR_TC reaches 0. If the STOP bit is 1, this bit becomes 0 as soon as the current atomic transfer is completed.<br>**Note:** This bit should not be changed manually during DMA operations (i.e. this has to be changed only by using DCON[22] or STOP bit). | 0 |

# 三、 WM8976 硬件设置：

内核中并没有 WM8976 的驱动。
传输声音数据：IIS 是标准规范，不管使用什么声卡芯片，都可以用 IIS。

## 1. 传输控制信息：

“控制接口”
芯片寄存器




## 2. 一，控制接口：

### 1) 两种模式：

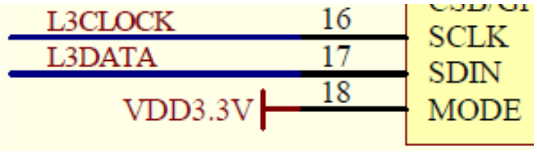
两线模式 与 三线模式：

| MODE | INTERFACE FORMAT |
|---|---|
| Low | 2 wire |
| High | 3 wire |

**Table 64 Control Interface Mode Selection**

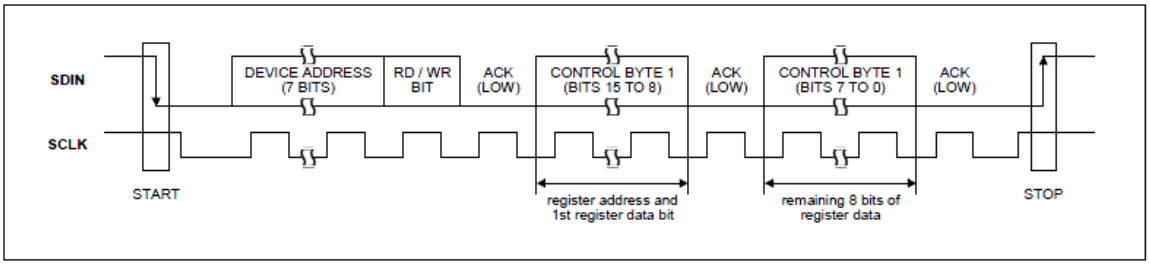
① 两线模式：

I2C 接口。



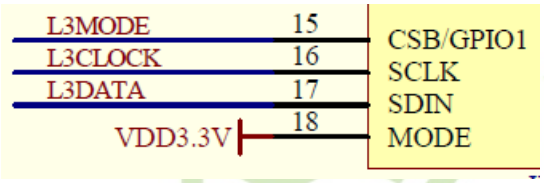

（MODE 引脚是高电平）

若“MODE”引脚是低电平，就只用到“SCLK”和“SDIN”引脚。这就是 I2C 的引脚。

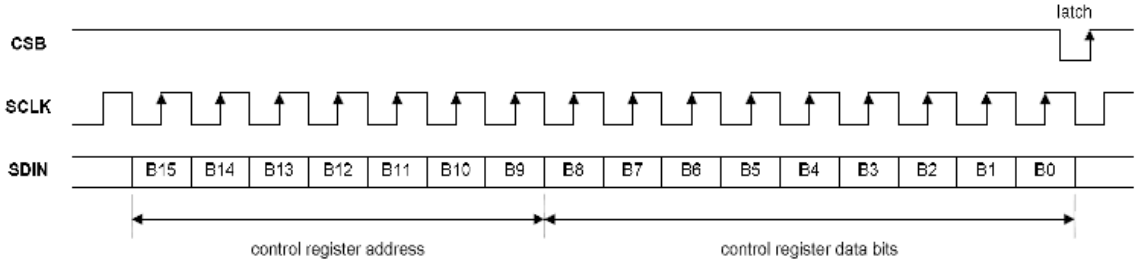



两线模式是 I2C 接口

② 三线模式：




“MODE”引脚是高电平，就成为“三线模式”：




先传 16 位数据（如上 SCLK16 个有标箭头的脉冲），先传的 16 位数据是先传最高位。CSB 是来一个低脉冲时把前面的 16 位数据取进来。

CSB 平时高电平，每一个 SCLK 脉冲有一个 SDIN 数据。传完 16 位数据后，CSB 来一个低脉冲，芯片就要把这 16 位数据来操作。

16 位数据中，前 7 位是“寄存器地址”，后 9 位是“数据”。

从上面分析，IIS 控制部分不用修改，只是要针对具体的芯片写“控制接口”，如上面的 WM8976 的控制接口就不是"UDA1341"的 L3 控制接口。还有就是“芯片寄存器”不一样。

2) 写“WM8976”驱动：

```
01285:      //init_uda1341();
01286:      init_wm8976();
01287:
```

3) 设计代码：

## 3. 一，写寄存器函数： 按 WM8976 芯片手册的时序写。

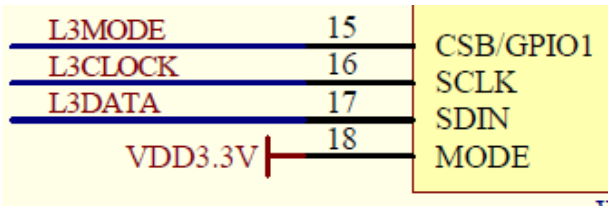
1) 数据为 16 位，前 7 位为地址，后 9 位为数据。

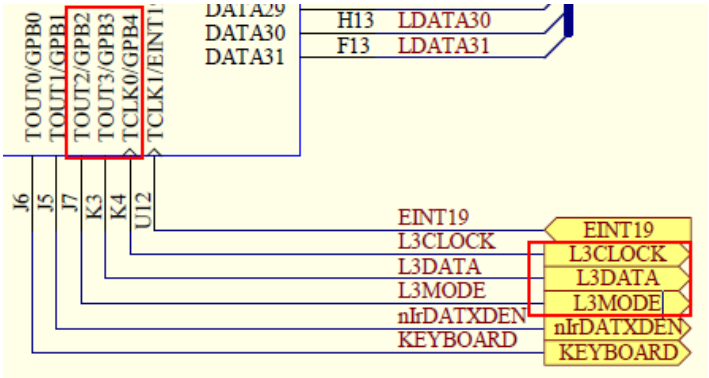
```
/* 写操作寄存器函数:按wm8976芯片手册规定,寄存器为7位，数据为9。一共16位。 */
static void wm8976_write_reg(unsigned char reg, unsigned int data)
{
    int i;
    unsigned long flags;
    /* 1，先关中断，最后关中断 */
    local_irq_save(flags);

    local_irq_restore(flags);
}
```

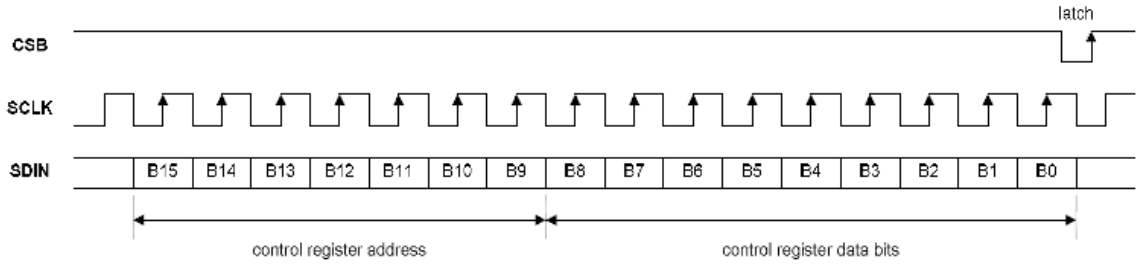
过程先关中断，最后操作完后开中断。

www.100ask.org

① 原理图和时序图如下：






② 2，设置时序：

```
/* 3，寄存器reg和数据data共16位，reg为高7位，reg左移9位；data为低9位，0x1ff为‘111111111’ */
unsigned short val = (reg << 9) | (data & 0x1ff); /* 将此值写入 */

/* 2，将CSB、SCLK、SDIN设置成高电平 */
s3c2410_gpio_setpin(S3C2410_GPB2, 1); /* CSB */
s3c2410_gpio_setpin(S3C2410_GPB3, 1); /* SCLK */
s3c2410_gpio_setpin(S3C2410_GPB4, 1); /* SDIN */

/* 1，先关中断，最后关中断 */
local_irq_save(flags);
/* 4，WM8976手册P78，三线控制接口的时序操作：CSB高电平期间，SCLK时钟低电平期间，SDIN发出一位数据，16位发完后，CSB最后有一个低电平。 */
/* 4.1，bit15若为高电平时，数据线上就输出1；若为低电平时，数据线上输出0 */
for(i = 0; i<16; i++)
{
    if(val & (1<<15)) {/* 先写最高位，一共16位，最高位即1左移15位处。&操作后，即val的第16位与上1 */
        s3c2410_gpio_setpin(S3C2410_GPB4, 0); /* 先让时钟为低电平 */
        s3c2410_gpio_setpin(S3C2410_GPB3, 1); /* 再让数据线上输出1 */
        udelay(1); /* 写之中加点延迟 */
        s3c2410_gpio_setpin(S3C2410_GPB4, 1); /* 再让时钟为高电平，即1个周期时钟内的操作 */
    }
    else
    {
        s3c2410_gpio_setpin(S3C2410_GPB4, 0); /* 先让时钟为低电平 */
        s3c2410_gpio_setpin(S3C2410_GPB3, 0); /* 再让数据线上输出1 */
        udelay(1); /* 写之中加点延迟 */
        s3c2410_gpio_setpin(S3C2410_GPB4, 1); /* 再让时钟为高电平，即1个周期时钟内的操作 */
    } /* 以上便处理好了最高位 */
    val = val << 1; /* 高位处理完后，再左移一位处理 */
}
/* 4，2，16位寄存器地址和数据发完后，CSB引脚有一个低电平脉冲 */
s3c2410_gpio_setpin(S3C2410_GPB2, 0); /* 写完16位后，最后需要一个CSB低脉冲 */
udelay(1);
s3c2410_gpio_setpin(S3C2410_GPB2, 1); /* 低电平后过一会子再变成高电平。完全按照时序上的操作。 */
/* 4.3.再将其他引脚也恢复成高电平*/
s3c2410_gpio_setpin(S3C2410_GPB3, 1); /* SCLK */
s3c2410_gpio_setpin(S3C2410_GPB4, 1); /* SDIN */
```

写哪些寄存器：这个需要从头慢慢看芯片手册 WM8976。

1) p80：

Power-up when NOT using the output 1.5x boost stage:

1. Turn on external power supplies. Wait for supply voltage to settle.
2. Mute all analogue outputs.
3. Set L/RMIXEN = 1 and DACENL/R = 1 in register R3.
4. Set BUFIOEN = 1 and VMIDSEL[1:0] to required value in register R1. Wait for the VMID supply to settle. *Refer notes 1 and 2.
5. Set BIASEN = 1 in register R1.
6. Set L/ROUT1EN = 1 in register R2.
7. Enable other mixers as required.
8. Enable other outputs as required.
9. Set remaining registers.

1，打开电源，等待电源稳定。（硬件上）
2，关闭所有的模拟输出。
3，在寄存器 R3 里设置 L/RMIXEN=1；DACENL/R=1。等待。

这一步骤需要很久才能调试好。

**REGISTER MAP**

| ADDR B[15:9] | | REGISTER NAME | B8 | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | DEF'T VAL |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DEC | HEX | | | | | | | | | | | (HEX) |
| 0 | 00 | Software Reset | Software reset | | | | | | | | | |
| 1 | 01 | Power manage't 1 | BUFDCOPEN | OUT4MIXEN | OUT3MIXEN | PLLEN | MICBEN | BIASEN | BUFIOEN | VMIDSEL | | 000 |
| 2 | 02 | Power manage't 2 | ROUT1EN | LOUT1EN | SLEEP | 0 | BOOSTENL | 0 | INPPGAENL | 0 | ADCENL | 000 |
| 3 | 03 | Power manage't 3 | OUT4EN | OUT3EN | LOUT2EN | ROUT2EN | 0 | RMIXEN | LMIXEN | DACENR | DACENL | 000 |
| 4 | 04 | Audio Interface | BCP | LRP | WL | | FMT | | DAC LRSWAP | ADC LRSWAP | DAC MONO | 050 |
| 5 | 05 | Companding ctrl | 0 | 0 | 0 | WL8 | DAC_COMP | | ADC_COMP | | LOOPBACK | 000 |
| 6 | 06 | Clock Gen ctrl | CLKSEL | MCLKDIV | | | BCLKDIV | | | 0 | MS | 140 |
| 7 | 07 | Additional ctrl | 0 | 0 | 0 | 0 | 0 | SR | | | SLOWCLKEN | 000 |
| 8 | 08 | GPIO Stuff | 0 | 0 | 0 | OPCLKDIV | | GPIO1POL | GPIO1SEL[2:0] | | | 000 |
| 9 | 09 | Jack detect control | JD_VMID | | JD_EN | 0 | JD_SEL | 0 | 0 | 0 | 0 | 000 |
| 10 | 0A | DAC Control | 0 | 0 | SOFTMUTE | 0 | 0 | DACOSR128 | AMUTE | DACPOLR | DACPOLL | 000 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 11 | 0B | Left DAC digital Vol | DACVU | DACVOLL | | | | | | | | 0FF |
| 12 | 0C | Right DAC dig'l Vol | DACVU | DACVOLR | | | | | | | | 0FF |
| 13 | 0D | Jack Detect Control | | JD_EN1 | | | | JD_EN0 | | | | 000 |
| 14 | 0E | ADC Control | HPFEN | HPFAPP | HPFCUT | | | ADCOSR 128 | 0 | 0 | ADCLPOL | 100 |
| 15 | 0F | ADC Digital Vol | ADCVU | ADCVOLL | | | | | | | | 0FF |
| 18 | 12 | EQ1 – low shelf | EQ3DMODE | 0 | EQ1C | | | EQ1G | | | | 12C |
| 19 | 13 | EQ2 – peak 1 | EQ2BW | 0 | EQ2C | | | EQ2G | | | | 02C |
| 20 | 14 | EQ3 – peak 2 | EQ3BW | 0 | EQ3C | | | EQ3G | | | | 02C |
| 21 | 15 | EQ4 – peak 3 | EQ4BW | 0 | EQ4C | | | EQ4G | | | | 02C |
| 22 | 16 | EQ5 – high shelf | 0 | 0 | EQ5C | | | EQ5G | | | | 02C |
| 24 | 18 | DAC Limiter 1 | LIMEN | LIMDCY | | | | LIMATK | | | | 032 |
| 25 | 19 | DAC Limiter 2 | 0 | 0 | LIMLVL | | | LIMBOOST | | | | 000 |
| 27 | 1B | Notch Filter 1 | NFU | NFEN | NFA0[13:7] | | | | | | | 000 |
| 28 | 1C | Notch Filter 2 | NFU | 0 | NFA0[6:0] | | | | | | | 000 |
| 29 | 1D | Notch Filter 3 | NFU | 0 | NFA1[13:7] | | | | | | | 000 |
| 30 | 1E | Notch Filter 4 | NFU | 0 | NFA1[6:0] | | | | | | | 000 |
| 32 | 20 | ALC control 1 | ALCSEL | 0 | 0 | ALCMAXGAIN | | | ALCMINGAIN | | | 038 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 33 | 21 | ALC control 2 | 0 | ALCHLD | | | | | ALCLVL | | | 00B |
| 34 | 22 | ALC control 3 | ALCMODE | ALCDCY | | | | | ALCATK | | | 032 |
| 35 | 23 | Noise Gate | 0 | 0 | 0 | 0 | 0 | NGEN | NGTH | | | 000 |
| 36 | 24 | PLL N | 0 | 0 | 0 | 0 | PLLPRE SCALE | PLLN[3:0] | | | | 008 |
| 37 | 25 | PLL K 1 | 0 | 0 | 0 | PLLK[23:18] | | | | | | 00C |
| 38 | 26 | PLL K 2 | PLLK[17:9] | | | | | | | | | 093 |
| 39 | 27 | PLL K 3 | PLLK[8:0] | | | | | | | | | 0E9 |
| 41 | 29 | 3D control | 0 | 0 | 0 | 0 | 0 | DEPTH3D | | | | 000 |
| 43 | 2B | Beep control | 0 | 0 | 0 | MUTER PGA2INV | INVROUT2 | BEEPVOL | | | BEEPEN | 000 |
| 44 | 2C | Input ctrl | MBVSEL | 0 | 0 | 0 | 0 | 0 | L2_2 INPPGA | LIN2 INPPGA | LIP2 INPPGA | 033 |
| 45 | 2D | INP PGA gain ctrl | INPPGA UPDATE | NPPGAZCL | INPPGA MUTEL | INPPGAVOLL | | | | | | 010 |
| 47 | 2F | ADC Boost ctrl | PGABOOSTL | 0 | L2_2BOOSTVOL | | | 0 | AUXL2BOOSTVOL | | | 100 |
| 49 | 31 | Output ctrl | 0 | 0 | DACL2 | DACR2 | OUT4 | OUT3 | SPK | TSDEN | VROI | 002 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | RMIX | LMIX | BOOST | BOOST | BOOST | | | |
| 50 | 32 | Left mixer ctrl | AUXLMIXVOL | | | AUXL2LMIX | BYPLMIXVOL | | | BYPL2LMIX | DACL2LMIX | 001 |
| 51 | 33 | Right mixer ctrl | AUXRMIXVOL | | | AUXR2RMIX | 0 | | | 0 | DACR2RMIX | 001 |
| 52 | 34 | LOUT1 (HP) volume ctrl | HPVU | LOUT1ZC | LOUT1 MUTE | LOUT1VOL | | | | | | 039 |
| 53 | 35 | ROUT1 (HP) volume ctrl | HPVU | ROUT1ZC | ROUT1 MUTE | ROUT1VOL | | | | | | 039 |
| 54 | 36 | LOUT2 (SPK) volume ctrl | SPKVU | LOUT2ZC | LOUT2 MUTE | LOUT2VOL | | | | | | 039 |
| 55 | 37 | ROUT2 (SPK) volume ctrl | SPKVU | ROUT2ZC | ROUT2 MUTE | ROUT2VOL | | | | | | 039 |
| 56 | 38 | OUT3 mixer ctrl | 0 | 0 | OUT3 MUTE | 0 | 0 | OUT4_ 2OUT3 | BYPL2 OUT3 | LMIX2 OUT3 | LDAC2 OUT3 | 001 |
| 57 | 39 | OUT4 (MONO) mixer ctrl | 0 | 0 | OUT4 MUTE | HALFSIG | LMIX2 OUT4 | LDAC2 OUT4 | 0 | RMIX2 OUT4 | RDAC2 OUT4 | 001 |

WM8976 只有 58 个寄存器。

## 4. 1，先复位：

| | | | | |
|---|---|---|---|---|
| 0 | 00 | Software Reset | Software reset | |

下面是寄存器说明：P87 开始是 58 个寄存器的取值说明。

| REGISTER ADDRESS | BIT | LABEL | DEFAULT | DESCRIPTION | REFER TO |
|---|---|---|---|---|---|
| 0 (00h) | [8:0] | RESET | N/A | Software reset | Resetting the Chip |

R0（寄存器 0）：software reset，在 P78 说明是“写任何一个值到 software reset”寄存器即可复位芯片。是电时是复位的（里面有上电复位电路）。

```
/* software reset */
wm8976_write_reg(0, 0);
```

## 5.2，往寄存器 3 写入一个值：

| 3 | 03 | Power manage't 3 | OUT4EN | OUT3EN | LOUT2EN | ROUT2EN | 0 | RMIXEN | LMIXEN | DACENR | DACENL | 000 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| 3 (03h) | 8 | OUT4EN | 0 | OUT4 enable<br>0 = disabled<br>1 = enabled | Power Management |
| | 7 | OUT3EN<br>从原理图上看OUT3没接 | 0 | OUT3 enable<br>0 = disabled<br>1 = enabled | Power Management |
| | 6 | LOUT2EN<br>输出2的左声道使能 | 0 | LOUT2 enable<br>0 = disabled<br>1 = enabled | Power Management |
| | 5 | ROUT2EN<br>输出2的右声道使能 | 0 | ROUT2 enable<br>0 = disabled<br>1 = enabled | Power Management |
| | 3 | RMIXEN<br>右输出通道混音使能 | 0 | Right output channel mixer enable:<br>0 = disabled<br>1 = enabled | Analogue Outputs |
| | 2 | LMIXEN<br>左输出通道混音使能 | 0 | Left output channel mixer enable:<br>0 = disabled<br>1 = enabled | Analogue Outputs |
| | 1 | DACENR<br>右通道DAC使能 | 0 | Right channel DAC enable<br>0 = DAC disabled<br>1 = DAC enabled | Analogue Outputs |
| | 0 | DACENL<br>左通道DAC使能 | 0 | Left channel DAC enable<br>0 = DAC disabled<br>1 = DAC enabled | Analogue Outputs |

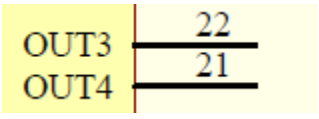
这里先写 R3 而不是按顺序先写 R2，是因为芯片上的操作步骤中推荐：

3. Set L/RMIXEN = 1 and DACENL/R = 1 in register R3.
4. Set BUFIOEN = 1 and VMIDSEL[1:0] to required value in register R1. Wait for the VMID supply to settle. *Refer notes 1 and 2.

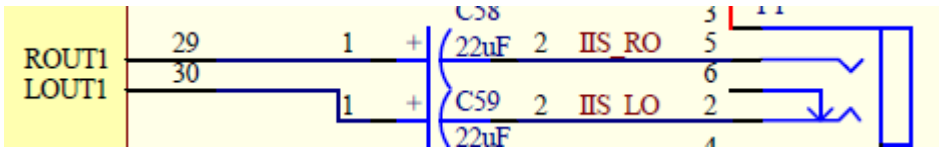
从上面看到这个“寄存器 3”的设置有 9 位。每一位的功能也如上说明了。这些位在原理图上也会有相应的引脚，是否设置它们也要看原理图是否有接线：

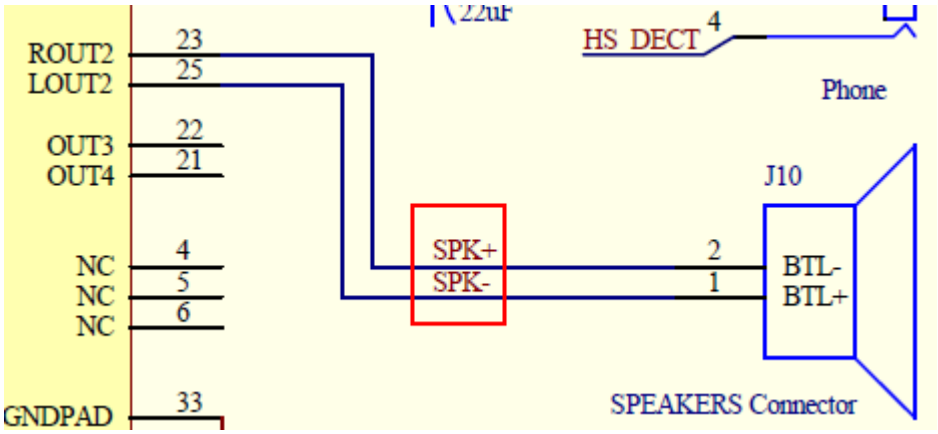
① 声道使能：bit5 ~ bit8

OUT3,OUT4 并没有接线，就不用设置。LOUT2EN ROUT2EN



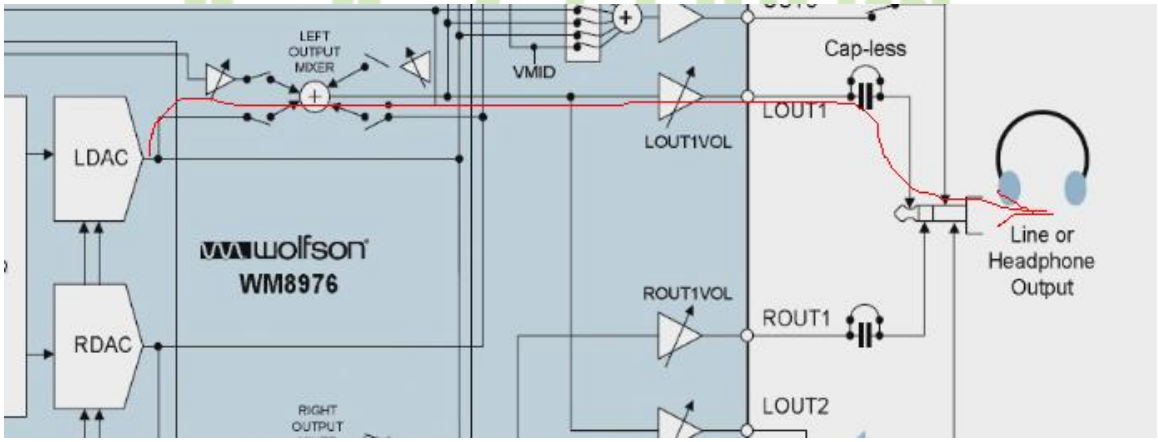

只需要“OUT1”的左声道和右声道。还有“OUT2”的 SPK+\SPK-：




② ，混音使能 和 “ADC 使能”: bit3 ~ bit0

多路叠加在一起，再输出。




从上面的框架图来年地，有通过“混音器”，所以使能它。而且“DAC”也要使能。

```
/* OUT2的左/右声道打开
 * 左/右通道输出混音打开
 * 左/右DAC打开
 */
wm8976_write_reg(0x3, 0x6f);
```

③ 3，往寄存器 1 中写值：

| | | | | | |
|---|---|---|---|---|---|
| 1 (01h) | 8 | BUFDCOPEN | 0 | Dedicated buffer for DC level shifting output stages when in 1.5x gain boost configuration.<br>0=Buffer disabled<br>1=Buffer enabled (required for 1.5x gain boost) | Analogue Outputs |
| | 7 | OUT4MIXEN | 0 | OUT4 mixer enable<br>0=disabled<br>1=enabled | Power Management |
| | 6 | OUT3MIXEN | 0 | OUT3 mixer enable<br>0=disabled<br>1=enabled | Power Management |
| | 5 | PLLEN | 0 | PLL enable<br>0=PLL off<br>1=PLL on | Master Clock and Phase Locked Loop (PLL) |
| | 4 | MICBEN | 0 | Microphone Bias Enable<br>0 = OFF (high impedance output)<br>1 = ON | Input Signal Path |
| | 3 | BIASEN | 0 | Analogue amplifier bias control<br>0=disabled<br>1=enabled | Power Management |
| | 2 | BUFIOEN | 0 | Unused input/output tie off buffer enable<br>0=disabled<br>1=enabled | Power Management |
| | 1:0 | VMIDSEL | 00 | Reference string impedance to VMID pin<br>00=off (open circuit)<br>01=75kΩ<br>10=300kΩ<br>11=5kΩ | Power Management |

这里为了简化设置，全部都设置为“1”，都使能起来。

Bit4"MICBEN：麦克风偏至使能。

麦克风有电源，平时为了省电，“MICBIAS”引脚不输出，要用麦克风时一定要他使能。

```
wm8976_write_reg(0x1, 0x1f);//biasen,BUFIOEN,VMIDSEL=11b
```

④ 4，还有很多寄存器要设置，但对于硬件来说不熟悉时，很多可以参考网上别人写的代码。这里其他寄存器的设置也类似。结果如下：

```
wm8976_write_reg(0x2, 0x185);//ROUT1EN LOUT1EN, inpu PGA enable ,ADC enable

wm8976_write_reg(0x6, 0x0);//SYSCLK=MCLK
wm8976_write_reg(0x4, 0x10);//16bit
wm8976_write_reg(0x2B,0x10);//BTL OUTPUT
wm8976_write_reg(0x9, 0x50);//Jack detect enable
wm8976_write_reg(0xD, 0x21);//Jack detect
wm8976_write_reg(0x7, 0x01);//Jack detect
```

设置也这些寄存器后，就操作完了 WM8976 芯片。

## 2) 三，其他设置：

1，调整音量 和 混音器的增益信号：

```
uda1341_volume = 63 - (((val & 0xff) + 1) * 63) / 100;
//uda1341_l3_address(UDA1341_REG_DATA0);
//uda1341_l3_data(uda1341_volume);
```

UDA1341 的音量是先发出“DATA0”后再发出音量。

**Table 17** Data transfer of type 'DATA0'

| BIT 7 | BIT 6 | BIT 5 | BIT 4 | BIT 3 | BIT 2 | BIT 1 | BIT 0 | REGISTER SELECTED |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | VC5 | VC4 | VC3 | VC2 | VC1 | VC0 | VC = volume control (6 bits) 音量 |
| 0 | 1 | BB3 | BB2 | BB1 | BB0 | TR1 | TR0 | BB = bass boost (4 bits) |
| | | | | | | | | TR = treble (2 bits) |
| 1 | 0 | PP | DE1 | DE0 | MT | M1 | M0 | PP = peak detection position |
| | | | | | | | | DE = de-emphasis (2 bits) |
| | | | | | | | | MT = mute |
| | | | | | | | | M = mode switch (2 bits) |
| 1 | 1 | 0 | 0 | 0 | EA2 | EA1 | EA0 | EA = extended address (3 bits) 扩展地址 |
| 1 | 1 | 1 | ED4 | ED3 | ED2 | ED1 | ED0 | ED = extended data (5 bits) 扩展数据 |

发出哪个扩展数据 后，再发出扩展数据

**Table 18** Volume settings

| VC5 | VC4 | VC3 | VC2 | VC1 | VC0 | VOLUME (dB) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | –1 |
| 0 | 0 | 0 | 0 | 1 | 1 | –2 |
| : | : | : | : | : | : | : |
| 1 | 1 | 1 | 0 | 1 | 1 | –58 |
| 1 | 1 | 1 | 1 | 0 | 0 | –59 |
| 1 | 1 | 1 | 1 | 0 | 1 | –60 |
| 1 | 1 | 1 | 1 | 1 | 0 | –∞ |
| 1 | 1 | 1 | 1 | 1 | 1 | –∞ |

0ask.org

这 6 位（VC5~VC0）值越小，音量越大。6 个“1”即“111111”为十进制“63”。

```
mixer_igain = 31 - (val * 31 / 100);
/* use mixer gain channel 1*/
//uda1341_l3_address(UDA1341_REG_DATA0);
//uda1341_l3_data(EXTADDR(EXT0));
//uda1341_l3_data(EXTDATA(EXT0_CH1_GAIN(mixer_igain)));
break;
```

初步设置芯片时，可以先去掉这些设置，调整好芯片后再开启这些功能。

# 6. 四，编译测试：

## 1) 怎么写 WM8976 驱动程序?

1. IIS 部分一样，保持不变
2. 控制部分不同，重写

① 测试 WM8976:

1. 确定内核里已经配置了 sound\soc\s3c24xx\s3c2410-uda1341.c

```
-> Device Drivers
  -> Sound
    -> Advanced Linux Sound Architecture  // 兼容OSS
      -> Advanced Linux Sound Architecture
        -> System on Chip audio support
        <*> I2S of the Samsung S3C24XX chips
```

② 2. 修改 sound/soc/s3c24xx/Makefile

```
obj-y += s3c2410-uda1341.o
改为:
obj-y += s3c-wm8976.o
```

```
# S3c24XX Platform Support
snd-soc-s3c24xx-objs := s3c24xx-pcm.o
snd-soc-s3c24xx-i2s-objs := s3c24xx-i2s.o

obj-$(CONFIG_SND_S3C24XX_SOC) += snd-soc-s3c24xx.o
obj-$(CONFIG_SND_S3C24XX_SOC_I2S) += snd-soc-s3c24xx-i2s.o
#obj-y += s3c2410-uda1341.o
obj-y += s3c-wm897 .o
```

100 百问网 www.100ask.org

③ 3. make uImage

使用新内核启动

④ 4. ls -l /dev/dsp /dev/mixer

```
# ls -l /dev/dsp /dev/mixer
crw-rw---- 1 0        0          14,   3 Jan  1 00:00 /dev/dsp
crw-rw---- 1 0        0          14,   0 Jan  1 00:00 /dev/mixer
# c
```

⑤ 5. 播放:

在 WINDOWS PC 里找一个 wav 文件,放到开发板根文件系统里

```
cat Windows.wav > /dev/dsp
```

⑥ 6. 录音:

cat /dev/dsp > sound.bin

然后对着麦克风说话，若是麦克风的“增益”没设置，输入的声音的音量会很少。

ctrl+c 退出

cat sound.bin > /dev/dsp  // 就可以听到录下的声音

修改音量和混音增益：

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 52 | 34 | LOUT1 (HP) volume ctrl | HPVU | LOUT1ZC | LOUT1 MUTE | LOUT1VOL | | | | | | 039 |
| 53 | 35 | ROUT1 (HP) volume ctrl | HPVU | ROUT1ZC | ROUT1 MUTE | ROUT1VOL | | | | | | 039 |
| 54 | 36 | LOUT2 (SPK) volume ctrl | SPKVU | LOUT2ZC | LOUT2 MUTE | LOUT2VOL | | | | | | 039 |
| 55 | 37 | ROUT2 (SPK) volume ctrl | SPKVU | ROUT2ZC | ROUT2 MUTE | ROUT2VOL | | | | | | 039 |
| 56 | 38 | OUT3 mixer ctrl | 0 | 0 | OUT3 MUTE | 0 | 0 | OUT4_ 2OUT3 | BYPL2 OUT3 | LMIX2 OUT3 | LDAC2 OUT3 | 001 |

从上面的提示看“52”和“53”号寄存器：

| | | | | | |
|---|---|---|---|---|---|
| 52 (34h) | 8 | HPVU<br>这位必须写为1 | N/A | LOUT1 and ROUT1 volumes do not update until a 1 is written to HPVU (in reg 52 or 53) | Analogue Outputs |
| | 7 | LOUT1ZC | 0 | Headphone volume zero cross enable:<br>1 = Change gain on zero cross only<br>0 = Change gain immediately | Analogue Outputs |
| | 6 | LOUT1 MUTE | 0 | Left headphone output mute:<br>0 = Normal operation<br>1 = Mute | Analogue Outputs |
| | 5:0 | LOUT1VOL | 111001 | Left headphone output volume:<br>000000 = -57dB<br>...<br>111001 = 0dB<br>...<br>111111 = +6dB | Analogue Outputs |
| 53 (35h) | 8 | HPVU | N/A | LOUT1 and ROUT1 volumes do not update until a 1 is written to HPVU (in reg 52 or 53) | Analogue Outputs |
| | 7 | ROUT1ZC | 0 | Headphone volume zero cross enable:<br>1 = Change gain on zero cross only<br>0 = Change gain immediately | Analogue Outputs |
| | 6 | ROUT1 MUTE | 0 | Right headphone output mute:<br>0 = Normal operation<br>1 = Mute | Analogue Outputs |
| | 5:0 | ROUT1VOL | 111001 | Right headphone output volume:<br>000000 = -57dB<br>...<br>111001 = 0dB<br>...<br>111111 = +6dB | Analogue Outputs |

52 寄存器的“HPVU”必须写为“1”，音量才会更新。

从“bit[5~0]”看是值越大，音量越大。默认值是“111001”，十进制为“57”。

Uda1341_volume = 57,名字没变，这里是设置 WM8976 的默认音量。

```
ioctl: val越大表示音量越大，0-最小，100-最大
```

应用程序得到的值是 100，而 WM8976 的音量是“0-63”：

```
WM8976: 52,53号寄存器bit[5:0]表示音量，值越大音量越大，0-63
```

这要扩大一下。应用程序传入“100”时，这个 52 寄存器的 bit[5:0]是为 63.

应用程序传入“0”时，这个 52 寄存器的 bit[5:0]是为 0.

可以算出音量值：音量 = val*63/100

```
/* ioctl: val越大表示音量越大，0-最小，100-最大
 * UDA1341: 寄存器的值越小音量越大
 * WM8976: 52,53号寄存器bit[5:0]表示音量，值越大音量越大，0-63
 */

uda1341_volume = (((val & 0xff) + 1) * 63) / 100;
wm8976_write_reg(52, (1<<8)|uda1341_volume);
wm8976_write_reg(53 (1<<8)|uda1341_volume);
```

设置 WM8976 的 52 号寄存器，先让 bit8 为 1，再让 bit5-0 设置上值，所以是(1<<8)|音量值。

2) 下面是读音量：

```
case SOUND_MIXER_READ_VOLUME:
    val = (uda1341_volume * 100) / 63;
    return put_user(val, (long *) arg);
```

以上只是自己通过寄存器设置这个声卡芯片，要是想很清楚的设置此芯片，还是要看厂家提供的驱动来修改，厂家对芯片最熟悉。要是厂家没有提供驱动，那就没办法只能自己慢慢摸索。

# 四、 播放 MP3:

MP3 是压缩文件，要先解压它，再发给 2440 的 IIS 控制器处理后，再发给“编解码芯片”。

100 百问网 www.100ask.org

## 1. 使用 madplay 测试声卡：

1. 解压:

```
tar xzf libid3tag-0.15.1b.tar.gz  // 库
tar xzf libmad-0.15.1b.tar.gz     // 库
tar xzf madplay-0.15.2b.tar.gz    // APP 依赖上面两个库文件
```

## 2. 先编译库文件，编译 libid3tag-0.15.1b

mkdir tmp

```
nd/app$ mkdir tmp
nd/app$ cd -
```

```
cd libid3tag-0.15.1b
./configure --host=arm-linux --prefix=/work/drivers_and_test/21th_sound/app/tmp
make
make install
```

## 3. 编译 libmad-0.15.1b

```
cd libmad-0.15.1b
./configure --host=arm-linux --prefix=/work/drivers_and_test/21th_sound/app/tmp
make
make install
```

## 4. 编译 madplay

```
CFLAGS      C compiler flags
LDFLAGS     linker flags, e.g. -L<lib dir> if you have libraries in a
            nonstandard directory <lib dir>
```

```
cd madplay-0.15.2b/
./configure --host=arm-linux --prefix=/work/drivers_and_test/21th_sound/app/tmp LDFLAGS="-
L/work/drivers_and_test/21th_sound/app/tmp/lib" CFLAGS="-I
/work/drivers_and_test/21th_sound/app/tmp/include"
```

上面配置时指定了链接的库和库的头文件。

```
make

make install
```

## 5. 把 tmp/bin/*  tmp/lib/*so* 复制到根文件系统：

```
/tmp$ cp lib/*so* /work/nfs_root/first_fs/lib/ -d
```

“-d”保持链接属性.

## 6. 把一个 mp3 文件复制到根文件系统

## 7. madplay --tty-control /1.mp3

```
# madplay --tty-control /1.mp3
MPEG Audio Decoder 0.15.2 (beta) - Copyright (C) 2000-2004 Robert Leslie et al.
s3c2410-uda1341-superlp: audio_set_dsp_speed:44100 prescaler:66
```

播放过程中不断按小键盘的减号("-")会降低音量
不断按小键盘的加号("+")会降低音量