LD3320

开发手册

ICRoute 用声音去沟通
VUI (Voice User Interface)

Web : www.icroute.com
Tel : 021-68546025
Mail: info@icroute.com

目录

一. 简介

LD3320 芯片是一款“**语音识别**”专用芯片。该芯片集成了语音识别处理器和一些外部电路，包括 AD、DA 转换器、麦克风接口、声音输出接口等。本芯片不需要外接任何的辅助芯片如 Flash、RAM 等，直接集成在现有的产品中即可以实现语音识别/声控/人机对话功能。并且，识别的关键词语列表是可以任意动态编辑的。本文档介绍如何编写程序实现芯片的功能。为更好地理解本文档内容，建议用户先仔细阅读《LD3320 数据手册》。

二. 寄存器操作

本芯片的各种操作，都必须通过寄存器的操作来完成。比如设置标志位、读取状态、向 FIFO 写入数据等。寄存器读写操作有 2 种方式，即标准并行方式和串行 SPI 方式。可参考**附录 B** 中的代码。

1. 并行方式

第 46 脚（MD）接低电平时按照此方式工作。

写和读的时序图如下：

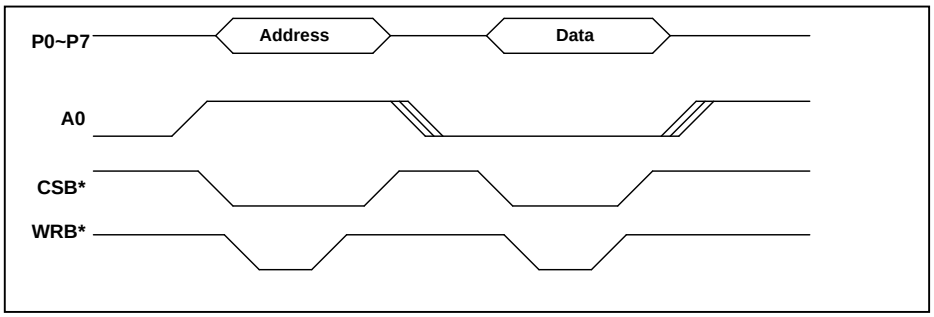


图 1　并行方式写时序

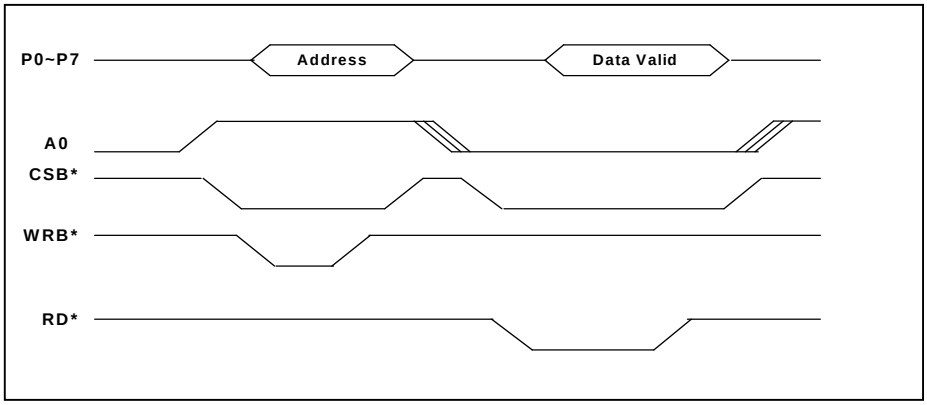


图 2　并行方式读时序

由时序图可以看到，A0 负责通知芯片是数据段还是地址段。A0 为高时是地址，而 A0 为低时是数据。发送地址时 CSB*和 WRB*必须有效，写数据时同样 CSB*和 WRB*必须有效，而读数据时 CSB*和 RDB*必须有效。

2. 串行 SPI 方式

第 46 脚（MD）接高电平，且第 42 腿（SPIS*）接地时按照此方式工作。写和读的时序图如下：

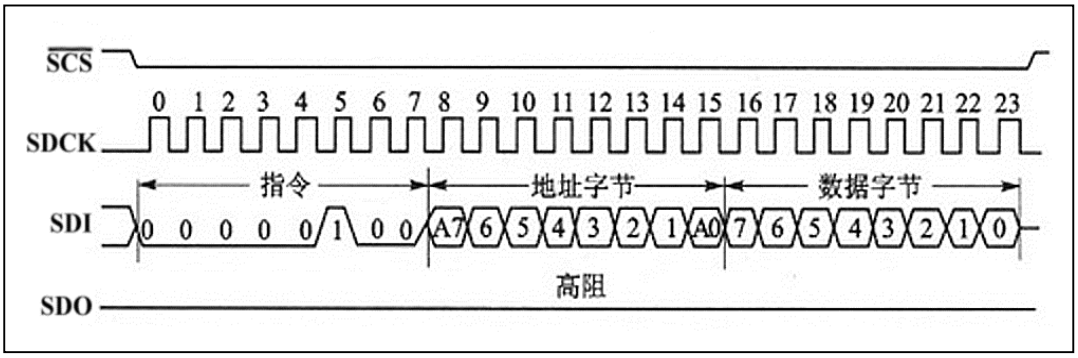


图 3　SPI 方式写时序

写的时候要先给 SDI 发送一个 “写”指令（04H），然后给 SDI 发送 8 位寄存器地址，再给 SDI 发送 8 位数据。在这期间，SCS*必须保持在有效（低电平）。

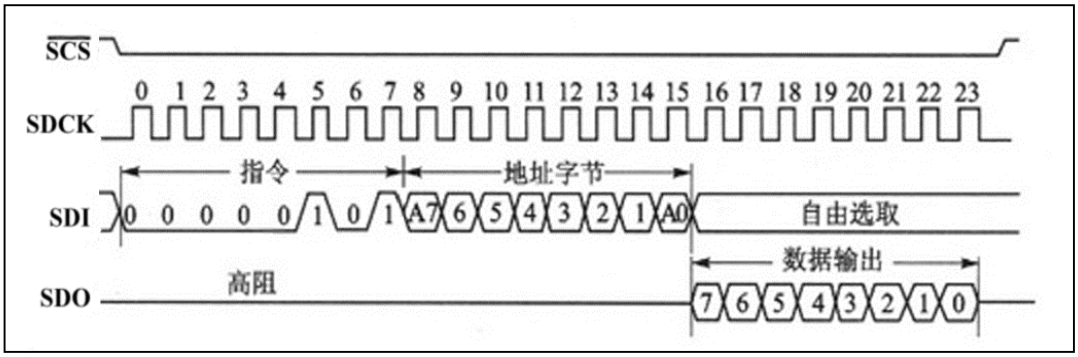


图 4　SPI 方式读时序

写的时候要先给 SDI 发送一个 “读”指令（05H），然后给 SDI 发送 8 位寄存器地址，再从 SDO 接受 8 位数据。在这期间，SCS*必须保持在有效（低电平）。

三．寄存器介绍

寄存器大部分都是有读和写的功能，有的是接受数据的，有的是设置开关和状态的。寄存器的地址空间为 8 位，可能的值为 00H 到 FFH。但是除了在本文档里介绍的寄存器，其他大部分为测试或保留功能的寄存器，请用户参考本文档的用法。

先介绍一些术语：

ASR：自动语音识别技术（Automatic Speech Recognition）。

FIFO:英文 First In First Out 的缩写，是一种先进先出的数据缓存器，它与普通存储器的区别是没有外部读写地址线，这样使用起来非常简单。

(*) LD3320 芯片内部有 2 个 FIFO，分别是：

FIFO_EXTFIFO_DATA　　FIFO_DATA　主数据处理 FIFO 缓存器，ASR 或者 MP3 的主数据区

FIFO_EXT　语音识别添加关键词用 FIFO 缓存器

MCU：本文档中专指外部电路板的主控芯片，对 LD3320 芯片进行控制的微处理器。

DSP：本文档中专指本芯片 LD3320 内部的专用 DSP，实现语音识别和语音播放的算法。

寄存器的详细说明如下：

编号 (16 进制)	说明
01	FIFO_DATA 数据口
02	FIFO 中断允许 第 0 位：允许 FIFO_DATA 中断； 第 2 位：允许 FIFO_EXT 中断；
05	FIFO_EXT 数据口
06	(只读)FIFO 状态 第 6 位：1 表示忙，不能写所有 FIFO。 第 3 位：1 表示 FIFO_DATA 已满，不能写。
08	清除 FIFO 内容（清除指定 FIFO 后再写入一次 00H） 第 0 位：写入 1→清除 FIFO_DATA 第 2 位：写入 1→清除 FIFO_EXT
11	时钟频率设置 1
17	写 48H 可以激活 DSP； 写 4CH 可以使 DSP 休眠，比较省电。
19	时钟频率设置 2
1B	时钟频率设置 3
1C	ADC 开关控制 写 00H ADC 不可用 写 0BH 麦克风输入 ADC 通道可用
1D	时钟频率设置 4
1E	ADC 专用控制，应初始化为 00H
1F	软复位（Soft Reset） 先写入 01H，再写入 00H
20	FIFO_DATA 上限低 8 位（UpperBoundary L）
21	FIFO_DATA 上限高 8 位（UpperBoundary H）
22	FIFO_DATA 下限低 8 位（LowerBoundary L）
23	FIFO_DATA 下限高 8 位（LowerBoundary H）
24	FIFO_DATA MCU 水线低 8 位（MCU water mark L）
25	FIFO_DATA MCU 水线高 8 位（MCU water mark H）
26	FIFO_DATA DSP 水线低 8 位（DSP water mark L）
27	FIFO_DATA DSP 水线高 8 位（DSP water mark H）
29	中断允许（可读写） 第 2 位：FIFO 中断允许，1 表示允许；0 表示不允许。 第 4 位：识别中断允许，1 表示允许；0 表示不允许。
2B	中断请求编号（只读） 第 4 位： 1 表示语音识别有结果产生。

33	MP3 播放用设置 开始播放时写入 01H, 播放完写入 00H。
35	ADC 增益，或可以理解为麦克风（MIC）音量。 测试版工作时使用的设置是 43H。可以设置为 00H-7FH。 建议设置值为 40H-6FH：值越大代表 MIC 音量越大，识别启动越敏感，但可能带来更多误识别；值越小代表 MIC 音量越小，需要近距离说话才能启动识别功能，好处是对远处的干扰语音没有反应。
37	语音识别控制命令下发寄存器 写 04H：通知 DSP 要添加一项识别句。 写 06H：通知 DSP 开始识别语音。 在下发命令前，需要检查 B2 寄存器的状态。
38	FIFO_EXT 上限低 8 位（UpperBoundary L）
3A	FIFO_EXT 上限高 8 位（UpperBoundary H）
3C	FIFO_EXT 下限低 8 位（LowerBoundary L）
3E	FIFO_EXT 下限高 8 位（LowerBoundary H）
40	FIFO_EXT MCU 水线低 8 位（MCU water mark L）
42	FIFO_EXT MCU 水线高 8 位（MCU water mark H）
44	FIFO_EXT DSP 水线低 8 位（DSP water mark L）
46	FIFO_EXT DSP 水线高 8 位（DSP water mark H）
79	时钟频率设置 5
81	耳机左音量 Bit7,6,0: Reserved; Bit[5-1]: 音量大小：数值越小，代表声音越大；数值越大，代表声音越小；本寄存器设置为 00H 为最大音量。 调节本寄存器后，设置 寄存器 87H.Bit1 = 1，可以使调节音量有效。
83	耳机右音量 Bit7,6,0: Reserved; Bit[5-1]: 音量大小：数值越小，代表声音越大；数值越大，代表声音越小；本寄存器设置为 00H 为最大音量。 调节本寄存器后，设置 寄存器 87H.Bit0 = 1，可以使调节音量有效。
85	内部反馈设置 初始化时写入 52H 播放 MP3 时写入 5AH （改变内部增益） 其中 Bit[1,0]为混音器反馈电阻设置 00 ： 60kohm 01 ： 45kohm 10 ： 30kohm 11 ： 15kohm 目前程序中设为 30Kohm

87	模拟电路控制 MP3 播放初始化时写 FFH Bit3：喇叭音量调节激发(见 8E 寄存器) Bit1：耳机左音量调节激发(见 81 寄存器) Bit0：耳机右音量调节激发(见 83 寄存器)
89	模拟电路控制 初始化时写 03H MP3 播放时写 FFH
8D	内部增益控制 初始化时写入 FFH
8E	喇叭输出音量 Bit7,6,1,0: Reserved; Bit[5-2]: 音量大小，共 16 等级：数值越小，代表声音越大；数值越大，代表声音越小； 本寄存器设置为 00H 为最大音量。 调节本寄存器后，设置 寄存器 87H.Bit3 = 1，可以使调节音量有效。
8F	LineOut 选择 初始化时写入 00H
B2	ASR：DSP 忙闲状态 0x21 表示闲，查询到为闲状态可以进行下一步 ASR 动作
B8	ASR：识别时间长度 最长识别时间长度设置为多少秒；缺省值是 60 秒。 在本时间长度内，如果检测到说话声音，语音识别模块将会给出识别结果；如果始终没有说话声音，将会返回 0 识别，见 BA 寄存器，并发出中断。
B9	ASR：当前添加识别句的字符串长度（拼音字符串） 初始化时写入 00H 每添加一条识别句后要设定一次。
BA	中断辅助信息，(读或设为 00) MP3：播放中断时， 第 5 位=1 表示播放器已发现 MP3 的结尾。 ASR：中断时，是语音识别有几个识别候选 Value: 1 - 4: N 个识别候选 0 或者大于 4：没有识别候选
BC	ASR：识别过程强制结束，在 ASR 进行过程中，可以设置本寄存器提前结束本次 ASR 过程； 写 07H，停止录音，但对已有声音进行识别运算，可能会有最优识别候选，返回 BA=0 - 4 写 08H，强制停止 ASR 运算，返回 BA=51H。 这两种设置都会使 DSP 送出中断，如同正常的识别结束

BD	初始化控制寄存器 写入 02H；然后启动；为 MP3 模块； 写入 00H；然后启动；为 ASR 模块；
BF	ASR：ASR 状态报告寄存器 读到数值为 0x35，可以确定是一次语音识别流程正常结束，可与（0xb2）寄存器的 0x21 值配合使用。
C1	ASR：识别字 Index（00H—FFH）
C3	ASR：识别字添加时写入 00
C5	ASR：读取 ASR 结果（最佳）
C7	ASR：读取 ASR 结果（候补 2）
C9	ASR：读取 ASR 结果（候补 3）
CB	ASR：读取 ASR 结果（候补 4）
CD	DSP 休眠设置 初始化时写入 04H 允许 DSP 休眠。
CF	内部省电模式设置 初始化时 写入 43H MP3 初始化和 ASR 初始化时写入 4FH

四．驱动程序

1. 芯片复位

就是对芯片的第 47 腿（RSTB*）发送低电平。可按照以下顺序：

```
void LD_reset()
{
  RSTB =1;
  delay(1);
  RSTB =0;
  delay(1);
  RSTB =1;
}
```

delay(1) 是延迟 1 毫秒的意思，为了更稳定地工作。

芯片初始化一般在程序的开始进行，如果有时芯片的反应不太正常，也可用这个方法恢复芯片的初始状态。

2. 语音识别

语音识别的操作顺序是：
通用初始化→语音识别用初始化→写入识别列表→开始识别，
并准备好中断响应函数，打开中断允许位。
这里需要说明一下，如果不用中断方式，也可以通过查询方式工作。在“开始识别”后，读取寄存器 B2H 的值，如果为 21H 就表示有识别结果产生。在此之后读取候选项等操作与中断方式相同。

（1） 通用初始化

按照以下序列设置寄存器。

```
void LD_Init_Common()
{
    bMp3Play = 0;

    LD_ReadReg(0x06);
    /* soft reset. */
    LD_WriteReg(0x1F, 0x1);
    delay( 10 );
    LD_WriteReg(0x1F, 0x0);
    delay( 10 );

    LD_WriteReg(0x89, 0x03);
    LD_WriteReg(0xCF, 0x43);
    LD_WriteReg(0xCB, 0x02);

    /*PLL setting*/
    LD_WriteReg(0x11, LD_PLL_11);
    LD_WriteReg(0x19, LD_PLL_19);
    LD_WriteReg(0x1B, LD_PLL_1B);
    LD_WriteReg(0x1D, LD_PLL_1D);
    LD_WriteReg(0x79, LD_LEDMTR_FREQ);
    LD_WriteReg(0xCD, 0x04);
    LD_WriteReg(0x17, 0x4c);
```

```
    LD_WriteReg(0xB9, 0x00);
    LD_WriteReg(0xCF, 0x4f);
}
```

（2） 语音识别用初始化

按照以下序列设置寄存器。

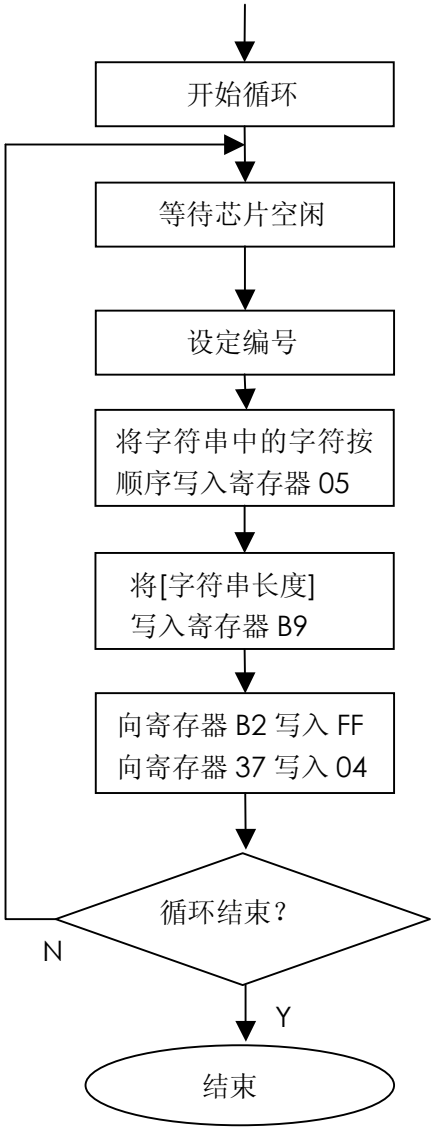
```
void LD_Init_ASR()
{
    nLD_Mode=LD_MODE_ASR_RUN;
    LD_WriteReg(0xBD,0x00);
    LD_WriteReg(0x17,0x48);
    delay( 10 );

    //!!注意，下面四个寄存器，会随晶振频率变化而设置
    LD_WriteReg(0x1E,0x00);
    LD_WriteReg(0x19,0x3f);
    LD_WriteReg(0x1D,0x1f);
    LD_WriteReg(0x1B,0x08);

    LD_WriteReg(0x3C,0xf0);
    LD_WriteReg(0x3E,0x07);
    LD_WriteReg(0x38,0xff);
    LD_WriteReg(0x3A,0x07);

    LD_WriteReg(0x40,0x08);
    LD_WriteReg(0x42,0x00);
    LD_WriteReg(0x44,0x08);
    LD_WriteReg(0x46,0x00);
    delay( 1 );
}
```

（3） 写入识别列表

列表的规则是，每个识别条目对应一个特定的编号（1 个字节），不同的识别条目的编号可以相同，而且不用连续。本芯片最多支持 50 个识别条目，每个识别条目是标准普通话的汉语拼音（小写），每 2 个字（汉语拼音）之间用一个空格间隔。下面是一个简单的例子：

编号	字符串
1	bei jing
1	shou du
3	shang hai
7	tian jin
8	chong qing

编号可以相同，可以不连续，但是数值要小于 256（00H~FFH）。例子中的“北京”和“首都”对应同一编号，说这两个词会有相同的结果返回。简单流程图如下：



代码如下：

```
#define CODE_DEFAULT    0
```

```
#define CODE_BEIJING    1
#define CODE_SHANGHAI   3
#define CODE_TIANJIN    7
#define CODE_CHONGQING 8
```

先介绍一个读取 0xB2 寄存器的函数，如果在以后的 ASR 命令函数前不能够读取到正确 Idle 状态，说明芯片内部可能出错了。经拷机测试，当使用的电源电压/电流出现不稳定有较大波动时，有小概率会出现这种情况。出现这种情况时，建议 Reset LD3320 芯片，重新启动设置芯片。

```
// Return 1: success.
uint8 LD_Check_ASRBusyFlag_b2()
{
    uint8 j;
    uint8 flag = 0;
    for (j=0; j<10; j++)
    {
        if (LD_ReadReg(0xb2) == 0x21)
        {
            flag = 1;
            break;
        }
        delay(10);
    }
    return flag;
}

// Return 1: success.
uint8 LD_AsrAddFixed()
{
    uint8 k, flag;
    uint8 nAsrAddLength;
    const char sRecog[5][13] = {"bei jing", "shou du",
        "shang hai", "tian jin", "chong qing"};
    const uint8 pCode[5] = {CODE_BEIJING, CODE_BEIJING,
        CODE_SHANGHAI, CODE_TIANJIN, CODE_CHONGQING};
```

```
    flag = 1;
    for (k=0; k<5; k++)
    {

        if(LD_Check_ASRBusyFlag_b2() == 0)
        {
            flag = 0;
            break;
        }

        LD_WriteReg(0xc1, pCode[k] );
        LD_WriteReg(0xc3, 0 );
        LD_WriteReg(0x08, 0x04);
        Delay(1);
        LD_WriteReg(0x08, 0x00);
        Delay(1);

        for (nAsrAddLength=0; nAsrAddLength<20;
            nAsrAddLength++)
        {
            if (sRecog[k][nAsrAddLength] == 0)
                break;
            LD_WriteReg(0x5, sRecog[k][nAsrAddLength]);
        }
        LD_WriteReg(0xb9, nAsrAddLength);
        LD_WriteReg(0xb2, 0xff);
        LD_WriteReg(0x37, 0x04);
    }
    return flag;
}
```

（4） **开始识别**

按照本节的说明，设置几个相关的寄存器，就可以控制 LD3320 芯片开始语音识别。

值得注意：单片机程序中，一般会用一个全局变量记录和控制当前状态(例如：LD_ASR_RUNING 状态或者 LD_ASR_FOUNDOK 状态)，

在编程时一定要把对该状态的设置语句放在 LD3320 芯片正式开始识别以前，例如下面例程中的语句

```
nAsrStatus=LD_ASR_RUNING;
```

便是设置全局变量。需要把这句语句放置在 LD3320 正式开始识别之前调用：

```
{      。。。
   nAsrStatus=LD_ASR_RUNING;
   LD_AsrRun();
}
```

参考代码如下：

```
// Return 1: success.
uint8 LD_AsrRun()
{
   nAsrStatus=LD_ASR_RUNING;
   LD_WriteReg(0x35,MIC_VOL);
   LD_WriteReg(0x1C,0x09);
   LD_WriteReg(0xBD,0x20);
   LD_WriteReg(0x08,0x01);
   delay( 1 );
   LD_WriteReg(0x08,0x00);
   delay( 1 );

      if(LD_Check_ASRBusyFlag_b2() == 0)
      {
         return 0;
      }

   LD_WriteReg(0xb2, 0xff);
   LD_WriteReg(0x37,0x06);
   delay(5);
   LD_WriteReg(0x1C,0x0b);
   LD_WriteReg(0x29,0x10);
   EX0=1;
   return 1;
```

```
}
```

综上所述：语音识别的流程可以总结成如下的参考代码，用户可以以此为参考，根据自己产品的使用流程来进行合理的改动。

```
uint8 RunASR()
{
    uint8 i=0;
    uint8 asrflag=0;
    for (i=0; i<5; i++)
    {
        LD_Init_Common();
        LD_Init_ASR();
        delay(100);
        if (LD_AsrAddFixed()==0)
        {
            LD_reset();
            delay(100);
            continue;
        }
        delay(10);
        if (LD_AsrRun() == 0)
        {
            LD_reset();
            delay(100);
            continue;
        }

        asrflag=1;
        break;
    }

    return asrflag;
}
```

对上述函数的调用参考代码：

文件 main.c 函数 main()

```
            case LD_ASR_NONE:
```

```
{
    nAsrStatus=LD_ASR_RUNING;
    if (RunASR()==0)
    {
        nAsrStatus = LD_ASR_ERROR;
        LED1=0;
        LED2=0;
    }
    break;
}
```

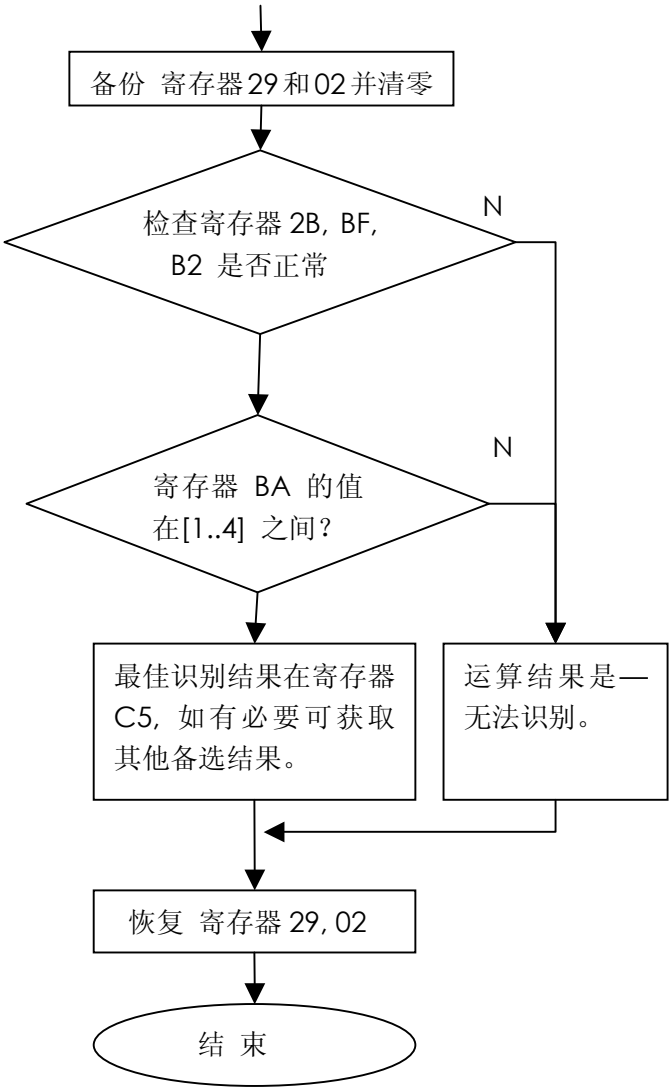
（5） 响应中断

如果麦克风采集到声音，不管是否识别出正常结果，都会产生一个中断信号。而中断程序要根据寄存器的值分析结果。

读取 BA 寄存器的值，可以知道有几个候选答案，而 C5 寄存器里的答案是得分最高、最可能正确的答案。

例如发音为“上海”并被成功识别（无其他候选），那么 BA 寄存器里的数值是 1，而 C5 寄存器里的值是对应的编码 3。

以下为简单流程图：



程序代码如下：

```
void ExtInt0Handler(void) interrupt 0
{
    uint8 nAsrResCount=0;

    EX0=0;
    ET0=0;

    ucRegVal = LD_ReadReg(0x2B);
    ucHighInt=LD_ReadReg(0x29);
```

```
    LD_WriteReg(0x29,0) ;

    ucLowInt=LD_ReadReg(0x02);
    LD_WriteReg(0x02,0) ;
    if(nLD_Mode == LD_MODE_ASR_RUN)
    {
        if( (ucRegVal & 0x10) &&
            LD_ReadReg(0xbf)==0x35 &&
            LD_ReadReg(0xb2)==0x21)
        {
            nAsrResCount = LD_ReadReg(0xba);
            if(nAsrResCount>0 && nAsrResCount<4)
            {
                nAsrStatus=LD_ASR_FOUNDOK;
            }
            else
            {
                nAsrStatus=LD_ASR_FOUNDZERO;
            }
        }
        else
        {
            nAsrStatus=LD_ASR_FOUNDZERO;
        }

        LD_WriteReg(0x2b,0);
        LD_WriteReg(0x1C,0);
        ET0=1;
        return;
    }
}


uint8 LD_GetResult()
{
    return LD_ReadReg(0xc5);
}
```

值得注意：获取识别结果

`LD_ReadReg(0xba);` 多少条候选识别结果，值 1~4 说明是有正确的识别结果。

4 个候选结果的读取：根据 0xba 决定读取几个识别结果。

```
LD_ReadReg(0xc5);
LD_ReadReg(0xc7);
LD_ReadReg(0xc9);
LD_ReadReg(0xcb);
```

在目前的 Demo 程序中，只读取了最优候选。在其他使用场合，如果需要读取其他候选，用户可以自己编程实现。

3. 声音播放

播放声音的操作顺序是：
通用初始化→MP3 播放用初始化→调节播放音量→开始播放声音，
并准备好中断响应函数，打开中断允许位。

（1） **通用初始化**

和语音识别部分一样，按指定序列设置寄存器。

（2） **声音播放用初始化**

请参照源代码设置寄存器。

```
void LD_Init_MP3()
{
  nLD_Mode = LD_MODE_MP3;
  LD_WriteReg(0xBD,0x02);
  LD_WriteReg(0x17,0x48);
  LD_WriteReg(0x85,0x52);
  LD_WriteReg(0x8F,0x00);
  LD_WriteReg(0x81,0x00);
  LD_WriteReg(0x83,0x00);
  LD_WriteReg(0x8E,0xff);
  LD_WriteReg(0x8D,0xff);
  LD_WriteReg(0x87,0xff);
  LD_WriteReg(0x89,0xff);
  LD_WriteReg(0x22,0);
```

```
  LD_WriteReg(0x23,0);
  LD_WriteReg(0x20,(uint8)2031);
  LD_WriteReg(0x21,(uint8)((2031>>8)&0x07));
  LD_WriteReg(0x24,(uint8)1524);
  LD_WriteReg(0x25,(uint8)((1524>>8)&0x07));
  LD_WriteReg(0x26,(uint8)1524);
  LD_WriteReg(0x27,(uint8)((1524>>8)&0x07));
}
```

（3） 调节播放音量

这里需要修改寄存器 8E。

音量分为 16 级，用 4 位二进制表示，范围是 0-15。

设置音量时，将(15-音量值) 设给寄存器 8E 的第 2-5 位。

7	6	5	4	3	2	1	0

高 低

源代码如下：

```
void LD_AdjustMIX2SPVolume(uint8 val)
{
  uint8 ucTmp;
  ucSPVol = val;
  val = ((15-val)&0x0f) << 2;
  ucRegVal = LD_ReadReg(0x8E)&0xc3;
  LD_WriteReg(0x8E, val | ucRegVal);
}
```

这个函数只调节喇叭（Speaker）的音量，和耳机等其他输出无关。而且实验板上只有喇叭连接了输出。

（4） 开始播放声音

- 开始播放位置清零（自定义变量 Mp3Pos=0）
- 寄存器 1B 的第 3 位设为 1（按位或 0x08）
- 循环执行：

  while（【播放条件】=true）

  {

  顺序将 MP3 数据放入寄存器 01(每次一个字节)；

  Mp3Pos 增加 1

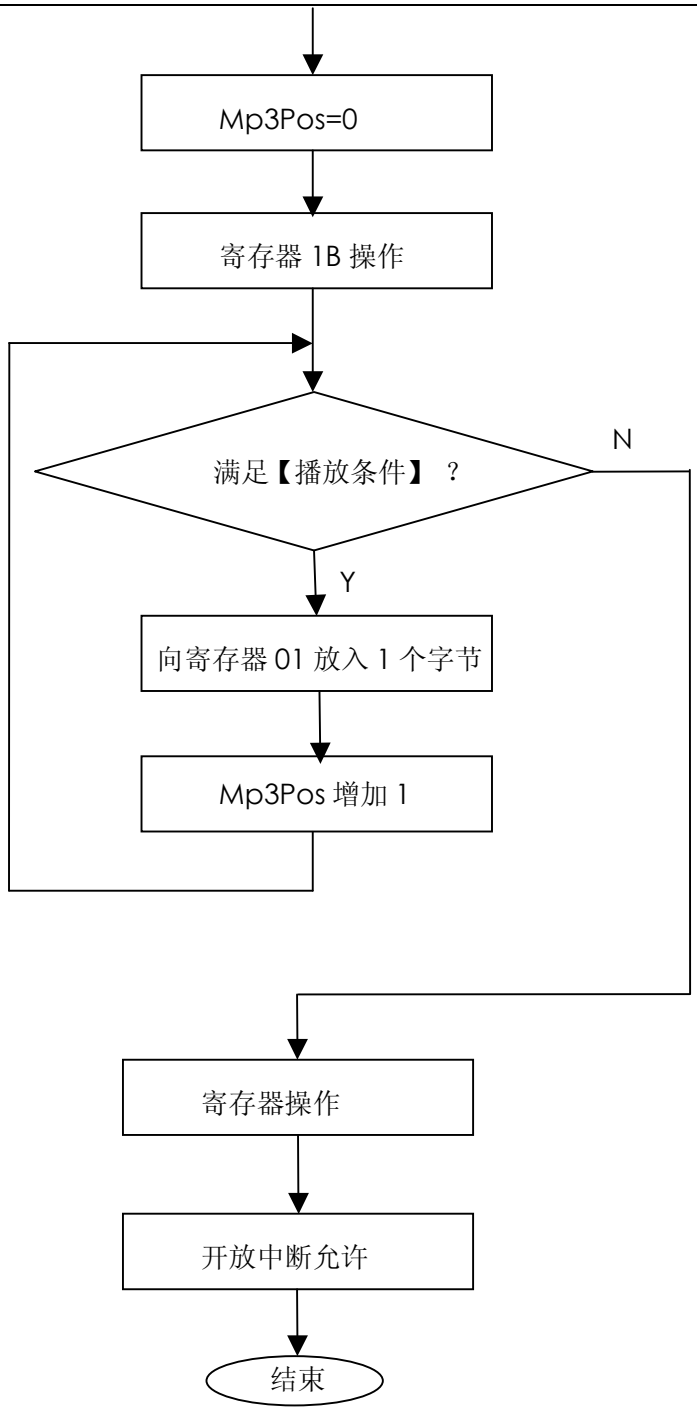
  }

  【播放条件】为下面条件都成立，有一个不满足就跳出循环：

- 读取寄存器 06，第 3 位=0
- Mp3Pos < MP3 文件的总长度。
  在实验板上，MP3 数据是由串行 Flash 里读出。(基于 SPI 协议。)

- 修改 BA 、17 等寄存器。(参照源代码)
- 开放中断允许。例如，EX0=1。

开始播放的简单流程图如下：



源代码如下：

```
void LD_play()
{
 nMp3Pos=0;
 bMp3Play=1;
```

```
 LD_WriteReg(0x1B, LD_ReadReg(0x1B)|0x08);
 if (nMp3Pos >=  nMp3Size)
    return ;

 LD_ReloadMp3Data();

LD_WriteReg(0xBA, 0);
 LD_WriteReg(0x17,0x48);
 LD_WriteReg(0x33, 0x01);
 ucRegVal  =  LD_ReadReg(0x29);
 LD_WriteReg(0x29, ucRegVal|MASK_INT_FIFO);

 ucRegVal  =  LD_ReadReg(0x02);
 LD_WriteReg(0x02, ucRegVal|MASK_AFIFO_INT);
 ucRegVal=LD_ReadReg(0x89);
 LD_WriteReg(0x89, ucRegVal | 0x0c);
 ucRegVal = (2& 0x03)<<2;//
 ucStatus=LD_ReadReg(0x85)&(~0x0c);
 LD_WriteReg(0x85, ucStatus | ucRegVal);

 EX0=1;
}

void LD_ReloadMp3Data()
{
 uint32 nCurMp3Pos;
 uint8 val;
 uint8 k;

 nCurMp3Pos = nMp3StartPos + nMp3Pos;
 FLASH_CS=1;
 FLASH_CLK=0;
 FLASH_CS=0;

 IO_Send_Byte(W25P_FastReadData);   /* read command */
  IO_Send_Byte(((nCurMp3Pos & 0xFFFFFF) >> 16));  /* send 3
address bytes */
```

```
    IO_Send_Byte(((nCurMp3Pos & 0xFFFF) >> 8));
    IO_Send_Byte(nCurMp3Pos & 0xFF);
    IO_Send_Byte(0xFF);

   ucStatus = LD_ReadReg(0x06);
   while ( !(ucStatus&MASK_FIFO_STATUS_AFULL) &&
          (nMp3Pos<nMp3Size) )
   {
      val=0;
      for(k=0;k<8;k++)
      {
         FLASH_CLK=0;
         val<<=1;
         FLASH_CLK=1;
         val|=FLASH_DO;
      }
      LD_WriteReg(0x01,val);
      nMp3Pos++;
      ucStatus = LD_ReadReg(0x06);
    }
    FLASH_CS=1;
    FLASH_CLK=0;
  }
```

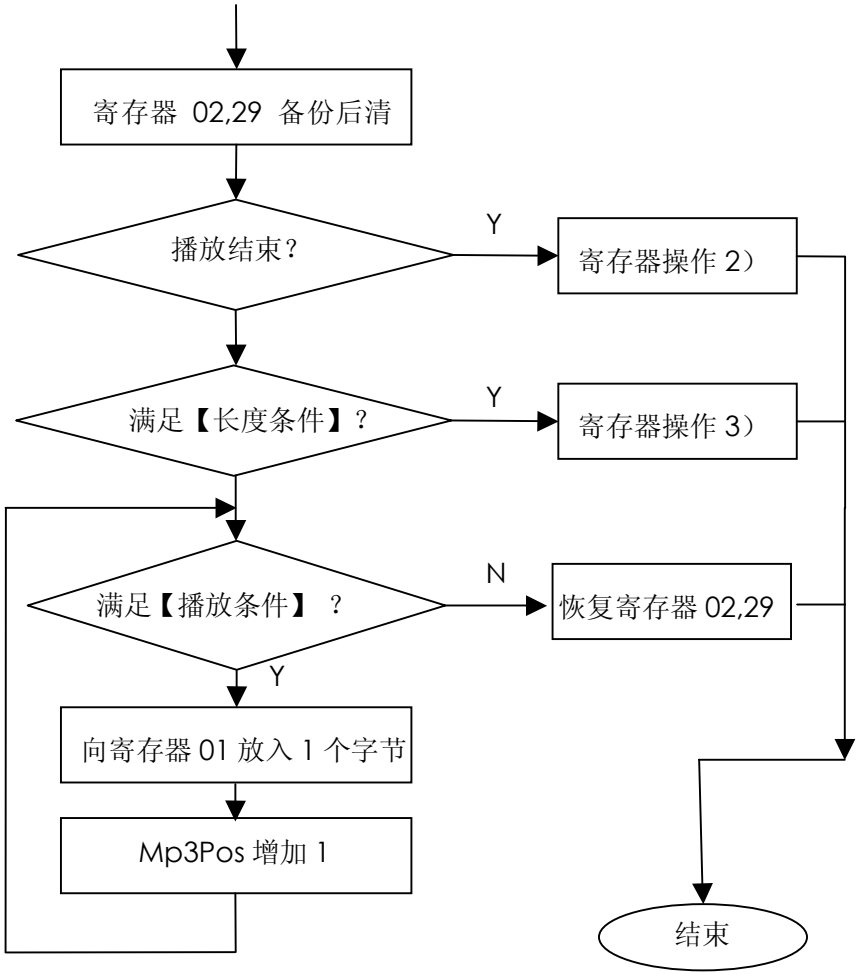
LD_ReloadMp3Data（）函数的功能是送入数据，不同的硬件结构可能需要改写这一部分。例如有的系统可能使用大容量的 RAM，取数据就会很方便。这里是根据串行 FLASH 存储器的接口写的函数，使用的是 SPI 协议。

（5） **中断响应。**

开始播放可以把声音数据的最初部分送入芯片，等到芯片播放这一段后会发出中断请求。而中断函数里会不断的送入数据，直到 FIFO_DATA 装满或声音数据结束。这一段程序和开始播放比较类似，都是通过 LD_ReloadMp3Data()函数送入数据。

由于 LD3320 芯片只有一只管脚负责中断请求输出，所以一般情况下用一个中断响应函数处理 2 种中断。这里为了简明，将中断函数分开书写。

中断处理函数里，播放声音部分流程图如下：



源代码是：

```
void ExtInt0Handler(void) interrupt 0  /*using 1*/
{
  uint8 nAsrResCount=0;
  EX0=0;
  ET0=0;
  ucRegVal = LD_ReadReg(0x2B);
  ucHighInt=LD_ReadReg(0x29);
  LD_WriteReg(0x29,0) ;
  ucLowInt=LD_ReadReg(0x02);
  LD_WriteReg(0x02,0) ;
  if(nLD_Mode == LD_MODE_MP3)
```

```
    {
      if(LD_ReadReg(0xBA)&CAUSE_MP3_SONG_END)
      {
        LD_WriteReg(0x2B,LD_ReadReg(0x2B)
                        &(~MASK_INT_SYNC));
        LD_WriteReg(0xBA, 0);
        LD_WriteReg(0xBC,0x0);
        bMp3Play=0;
        LD_WriteReg(0x08,1);
        LD_WriteReg(0x08,0);
        LD_WriteReg(0x33, 0);
        return ;
       }

     if(nMp3Pos>=nMp3Size)
    {
       LD_WriteReg(0xBC,0x01);
       ucStatus = LD_ReadReg(0x02);
       ucStatus&=(~MASK_AFIFO_INT);
       LD_WriteReg(0x02, ucStatus);
       ucStatus=LD_ReadReg(0x29);
       ucStatus&=(~MASK_INT_FIFO);
       LD_WriteReg(0x29,ucStatus|MASK_INT_SYNC) ;
       EX0=1;
       ET0=1;
       return;
    }

    LD_ReloadMp3Data();

    LD_WriteReg(0x29,ucHighInt);
    LD_WriteReg(0x02,ucLowInt) ;
    EX0=1;
  }
}
```

五．补充说明

- 此芯片的特色是兼有语音识别和 MP3 播放的两项功能，但是由于这两项功能会使用一些公用的资源，所以为了使芯片稳定地工作，在功能切换的时候，必须从最“通用初始化”开始，对芯片进行一系列的设置。

- 当芯片长时间没有响应时，可能是应用程序的设置不合理或者是电源的电压/电流有比较大的波动造成。这时应使用芯片复位的功能（对芯片的 RTSB* 发送低电平），使芯片重新开始工作。

- 演示板中接入芯片的晶振频率是 22.1184。如果用户选用其他频率的晶振，请根据参考代码中的说明在程序中修改相应的参数。

- 附件资料里有基于测试版的简单演示代码（Keil51 工程），可以在语音识别和声音播放之间切换，做到人说一句，机器回答一句。但是因为 LD3320 芯片只提供一个中断源，这时中断函数必须同时兼顾语音识别和声音播放，用户开发的时候一定要注意控制好流程。比如利用一些标志位来合理控制。

- 附件的简单代码中，控制逻辑是反复启动 ASR 语音识别过程。可以把这样的逻辑称之为“循环识别”。

  如果是每次接收到外界的触发，比如用户按键一次，只启动一次 ASR 语音识别过程。获得结果（或者识别没有结果）后，不再循环启动 ASR 识别过程，而是需要外界再提供一次触发，才再启动一次 ASR 语音识别过程，可以把这样的逻辑称之为“触发识别”。

  用户可以根据自己产品的实际应用环境，来采用不同的逻辑。

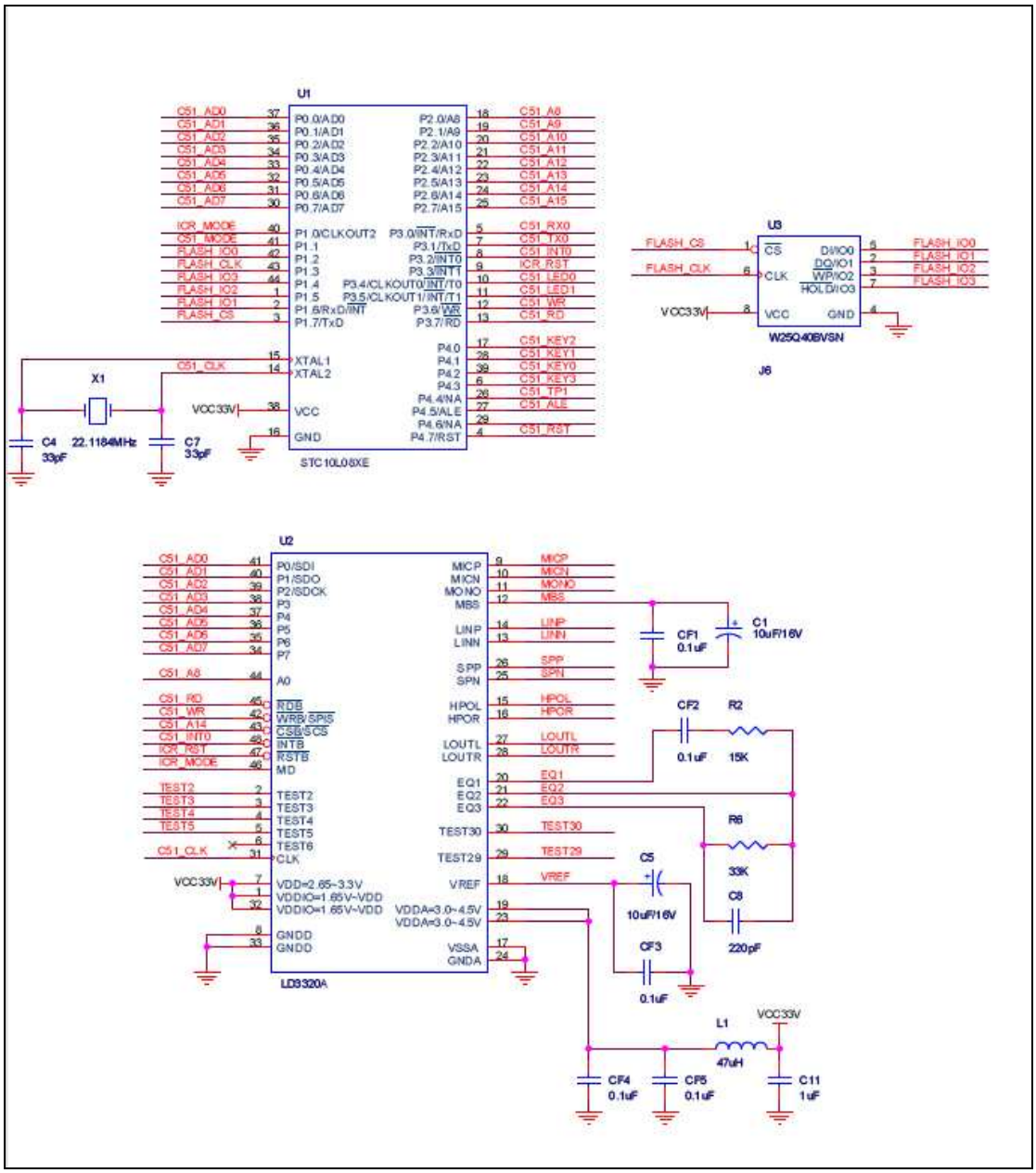
- 一般来说，触发识别适合于识别精度比较高的场合。外界触发后，产品可以播放提示音或者其他方式来提示用户在接下来的几秒钟内说出要识别的内容，这样来引导用户在规定的时间内只说出要识别的内容，从而保证比较高的识别率。

  而循环识别比较适合于识别精度不高的场合，在循环识别的过程中，用户的其他说话声音，或者外界的其他声音，都有可能被识别引擎误识别出错误的结果，需要产品的控制逻辑都作相应的处理。

- 根据现有设置，每次语音识别的时间限制是设定开始后的 1 分钟。在这个期间内，芯片接收到声音，无论识别是否成功，都会发送中断信号。如果到了 1 分钟还是没有接收到声音，也会发送一个中断，而 BA 寄存器里的数值是 0，表示识别失败。改动 B8 寄存器可以改变这个长度。

附录 A　测试版电路原理图

附件（LD3320 测试版原理图.pdf）里有全部的电路图，这里只画出核心 3 个芯片的连接关系。



这个电路中，MCU 的 P0 端口的 8 根线和 LD3320 并行方式连接，控制线也分别连接。此外还连接了复位信号和中断信号。对 LD3320 来说，复位信号（RSTB*）由 MCU 发出，而中断信号由 LD3320 发出，MCU 负责接收。

附录 B　寄存器操作介绍

在实验板上，利用 MCU 自己的读写时序特征，可以很方便地读写 LD 芯片的寄存器。在现有连接中，MCU 的 A14 连接到 LD 芯片的 CSB*，而 MCU 的 A8 连接到 LD 芯片的 A0。

并行方式的读写函数为：

```
#define LD_INDEX_PORT (*((volatile uint8 xdata*)(0x8100)))
#define LD_DATA_PORT  (*((volatile uint8 xdata*)(0x8000)))
void LD_WriteReg( uint8 ulAddr, uint8 ucVal )
{
  LD_INDEX_PORT  = ulAddr;
  LD_DATA_PORT = ucVal;
}
uint8 LD_ReadReg( uint8 ulAddr )
{
  LD_INDEX_PORT = ulAddr;
  return (uint8)LD_DATA_PORT;
}
```

8100：第 14 位=1　第 8 位=1

8000：第 14 位=1　第 8 位=0

于是 MCU 向 LD 芯片的 CSB*和 A0 发出信号，配合此时的数据线，可以对 LD 芯片的寄存器进行读写操作，十分方便。第 15 位=1 是为了避开低端地址。

在 SPI 方式下，读写寄存器操作比较麻烦，速度也较慢。在实验板上，SPI 方式下语音识别没有问题，但是声音播放可能会不流畅。 这是由于采用的低端 51 处理器作系统主控 MCU，系统运行频率低。如果用户的系统中采用高主频的主控 MCU，就会解决这一问题。写和读的代码分别如下：

```
sbit SCS=P2^6;    //芯片片选信号
sbit SDCK=P0^2;   //SPI 时钟信号
sbit SDI=P0^0;    //SPI 数据输入
sbit SDO=P0^1;    //SPI 数据输出
sbit SPIS=P3^6;   //SPI 模式设置：低有效。
```

```
/*****************************************************************
函数名： LD_SPI_WriteReg
功能：根据 SPI 接口写数据
参数：  address,地址
    dataout，写入数据
*****************************************************************/
void LD_SPI_WriteReg(unsigned char address,unsigned char dataout)
{
    unsigned char i = 0;
    unsigned char command=0x04;
    SPIS =0;
    SCS = 0;
    //write command
    for (i=0;i < 8; i++)
    {
        if ((command & 0x80) == 0x80)
            SDI = 1;
        else
            SDI = 0;
        delay(1);
        SDCK = 0;
        command = (command << 1);
        SDCK = 1;
    }
    //write address
    for (i=0;i < 8; i++)
    {
        if ((address & 0x80) == 0x80)
            SDI = 1;
        else
            SDI = 0;
        delay(1);
        SDCK = 0;
        address = (address << 1);
        SDCK = 1;
    }
    //write data
    for (i=0;i < 8; i++)
    {
        if ((dataout & 0x80) == 0x80)
            SDI = 1;
        else
            SDI = 0;
        delay(1);
```

```
        SDCK = 0;
        dataout = (dataout << 1);
        SDCK = 1;
    }
    SCS = 1;
}

/*****************************************************************
函数名： LD_SPI_ReadReg
功能：根据 SPI 接口写数据
参数：  address,地址
返回值：读取数据
*****************************************************************/
unsigned char LD_SPI_ReadReg(unsigned char address)
{
    unsigned char i = 0;
    unsigned char in =0 ;
    unsigned char temp = 0;
    unsigned char command=0x05;
    SPIS =0;
    SCS = 0;
    //write command
    for (i=0;i < 8; i++)
    {
        if ((command & 0x80) == 0x80)
            SDI = 1;
        else
            SDI = 0;
        delay(1);
        SDCK = 0;
        command = (command << 1);
        SDCK = 1;
    }
    //write address
    for (i=0;i < 8; i++)
    {
        if ((address & 0x80) == 0x80)
            SDI = 1;
        else
            SDI = 0;
        delay(1);
        SDCK = 0;
        address = (address << 1);
```

```
        SDCK = 1;
    }
    //Read data
    for (i=0;i < 8; i++)
    {
        in = (in << 1);
        temp = SDO;
        SDCK = 0;
        if (temp == 1)
            in |= 0x01;
        SDCK = 1;
    }
    SCS = 1;
    return in;
}
```

2010. 01.