



# LD3320

单芯片/非特定人/动态编辑识别列表  
语音识别芯片

## 简明调试步骤

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

Web : [www.icroute.com](http://www.icroute.com)  
Tel : 021-68546025  
Mail: [info@icroute.com](mailto:info@icroute.com)

LD3320 芯片或者 M-LD3320 模块正常工作的状态是：

1) 识别。

采用给定的参考程序代码不作设置的修改，运行完 RunASR() 启动识别流程后，说出一个关键词语，在说完后等待大概 600 毫秒左右，芯片会送出一个中断提示一次识别流程结束。此时参考程序中的中断处理函数会被调用并去寄存器中读取识别结果。（关于 600 毫秒的结束时间请阅读“语音识别芯片 LD3320 高阶秘籍.pdf”第八节）

如果用户不说话或者不插上麦克风，那么在默认设置下，等待 60 秒后芯片会送出一个中断提示一次识别流程结束。请阅读“开发手册.pdf”中关于 B8 寄存器的说明，这个时间可以最短设置到 1 秒。

2) 播放。

能正确播放送入的 MP3 数据。

用户应该首先调试电路以可以正确读写寄存器，和 LD3320 芯片进行通信。如果用户无法正确地进行读写寄存器的操作，请自行对照我们提供的原理图仔细检查硬件连接，包括焊接，电阻，晶振等方面。

在调试时，一般检查顺序为：

测量检查提供给 LD3320 芯片的电压是否正确：VDD，VDDIO，VDDA 三路电源管脚都需要接入 3.3v。两路地管脚需要正确接地。根据使用的传输方式（并口或者 SPI），测量 MD，SPIS 管脚的电压是否正确。

## 1. 上电调试

LD 芯片重启后，LD\_Reset，就是把(RSTB)管脚的电平拉低拉高后，管脚 29，30 会稳定地送出低电平，这个可以作为一个上电是否正常的检测。

## 2. 读写寄存器调试

检查寄存器的读写是否正确：

向可读写的寄存器写入某个数值，再读出来，用来检查寄存器读写是否正常。

由于曾经有客户出现过每次读写都是在自己的总线上进行，没有真正 touch 到 LD3320 芯片。

所以他每次先向一个寄存器写，再读出来，内容是完全正确，但实际上没有真正地读写进寄存器，而只是把刚刚写的数值再显示了出来。

所以我们建议读写寄存器的序列如下：

```
LD_reset();
LD_ReadReg(0x6);
LD_WriteReg(0x35, 0x33);
LD_WriteReg(0x1b, 0x55);
LD_WriteReg(0xb3, 0xaa);
打印并比较(LD_ReadReg(0x35));
打印并比较(LD_ReadReg(0x1b));
打印并比较(LD_ReadReg(0xb3));
```

就是向 3 个寄存器先依次写数据，再依次读数据出来。作为比较。  
这样可以有效地验证读写寄存器是否正常。

**重要说明：**有一种调试方法是在单片机的 IO 管脚上连接 LED，通过 flashLED 方式来提示信息，但是在实际中发现在一些型号的单片机上，这样的操作会导致单片机的 IO 管脚上的电压出现不稳定和波动，从而导致对 LD3320 的读写不正常。引起各种异常。

所以建议开发者，最好在对 LD3320 进行调试期间，尽量避免使用 FlashLED 的方式进行调试，避免干扰。

### 3. 检查寄存器初始值

在 LD\_Reset 后，写如下代码：

```
LD_reset();

打印(LD_ReadReg(0x06));
打印(LD_ReadReg(0x06));
打印(LD_ReadReg(0x35));
打印(LD_ReadReg(0xb3));
```

在 Reset 芯片后，直接读取 4 个寄存器。

如果是这个顺序打印的值应该是如下的 4 个值

00 87 80 FF

其中第一次读寄存器 06，只是相当于激活了芯片。值为 00，或者为 87，

其中第二次读寄存器 06，值为 0x87，

读寄存器 35，值为 0x80，

读寄存器 b3，值为 0xff。

检查分为两个步骤：

- 用户可以上电后作一次 LD\_reset()，检查寄存器初值。
- 随后在不掉电的情况下，再一次调用 LD\_reset()，再检查一次寄存器初值。

如果系统稳定，那么无论何时调用 LD\_reset()，都会得到正确的寄存器初始值。

如果发现第二次 LD\_reset() 后的寄存器初始值不正确，那么说明系统的硬件连接/电源/主控单片机的 I/O 电压等等硬件方面存在不稳定，导致 LD3320 的状态不稳定。

一定需要把这个问题解决后才可能正确地运行整个识别流程。

**重要说明：**在完成第 2 步和第 3 步的工作后，一定要把所有向寄存器写数值的操作都去掉，再去从头完整地运行提供的语音识别实例源程序。因为这些对寄存器进行写操作的语句，是会影响到 LD3320 的状态，不应该出现在正确的流程中。

#### 4. 在运行过程中检查寄存器的状态

如果前三步调试通过，但是运行程序不能完成识别功能，需要在运行过程中检查寄存器的状态。

1) 在完成 LD\_AsrAddFixed() 函数后，检测 0xBF 寄存器的数值，看是否是 0x31

2) 在 LD\_AsrRun() 函数中，在  
LD\_WriteReg(0x37, 0x06);  
delay( 5 );

后，读取 0xBF 寄存器的数值是否不再是 0x31, 而是 0x32~0x3a 之间的某一个数值。

如果这两个步骤检查不正确，可以先适当延长 初始化函数中 和 LD\_AsrRun() 函数中各个 delay 的时间，看是否可以调整好。

如果依然出现错误，一般还是读写寄存器不稳定造成寄存器设置不正确。

可以从 init 函数开始，在每一句设置寄存器的语句后面，都去读取寄存器的数值，看是否设置正确。

**重要说明：**以下三个寄存器是在向其写入数据后，再去读取回来的数值并不是写入的数据。0x17, 0x87, 0x89。

**重要说明：**并口连接 LD3320 芯片的可能错误：在语音识别流程中，读写 LD3320 芯片的寄存器都正确，添加词条等工作时都显得很正常，但后面是不停地出中断，0x2B 寄存器的 bit[3] 始终为 1 报告芯片内部错误，无法正常识别。请作如下检查，设置完 LD\_WriteReg(0x37, 0x06); Delay(100); 然后读取的 0xBF 寄存器是 0x31。对这种错误，请仔细阅读本文档的第 6 节：“并口连接对于 0x37 寄存器的读写时序”。

## 5. 重要提示

目前一般主要碰到的硬件问题有：

- a. 采用的主控单片机选型不当，不是工作在 3.3v 的单片机。

LD3320 芯片需要的工作电压是 3.3v。

要求控制 LD3320 芯片的单片机也工作在同一电压 3.3v。

I/O 管脚上，2.31v~3.3v 代表逻辑“1”，一定要注意保证单片机的 I/O 管脚输出到 LD3320 的管脚的电压**不要超过 3.3V!!!**（包括所有 LD3320 与单片机进行连接的管脚：P0~P7, MD, A0, WRB/SPIS, RDB, CSB/SCS, RSTB, INTB）

- b. 没有在 p0~p7 以及 MD, RDB, CSB, INTB, A0, RSTB, WRB 等控制端口上加上拉电阻，（参考“LD3320 测试板原理图.pdf”中的设置）。对于某些主控单片机会导致寄存器读写正常而芯片工作状态不正常。
- c. 晶振强度不够，用户换用有源晶振，或者缩短晶体到芯片管脚的连线
- d. 焊接不可靠。
- e. 电源设计不合理，诸如：没有采用 LD0 等稳压芯片保证稳定的 3.3v 工作环境；采用了诸如 PC 机 USB 端口直接取电导致电压会出现过低；稳压电路设计不合理导致单片机和 LD3320 工作后电压/电流出现比较大的变化。
- f. 有开发者使用某型号单片机的 I/O 端口输出给 MD 管脚提供高电平来进入 SPI 模式。但是实际测量后发现电压只有 1.7~1.9v，会导致芯片在 SPI 模式下读写寄存器不正常。于是直接把 MD 管脚接入 3.3v，芯片工作正常。

目前碰到的软件时序问题有：

### 1. 并口时序

用户使用的主控 MCU 没有独立的硬件并行读写端口 (WR, RD) 信号，是自己在 I/O 口上编写程序来实现并口读写的时序。 用户需要调试并行的时序，来保证读写寄存器的稳定。

比如是否需要增加 NOP，在给 /WR 和 /RD 信号之前留下足够的时间锁存数据。

## 2. SPI 时序

- a. 仔细阅读“LD3320 并行串行读写辅助说明.pdf”中的 SPI 模拟程序，LD3320 芯片要求每次在通过 SPI 方式读写寄存器时，都需要先传送一个命令字。
- b. 如果用户使用硬件 SPI 端口，需要先把 SPI 端口的时钟频率降到最低进行调试，调试通过后，可以逐步提高时钟频率进行尝试。
- c. 如果用户使用硬件 SPI 端口，当需要用户自己通过 I/O 口来进行 SPI 的选中（LD3320 芯片的 SCS 管脚）时，一定要注意每次对 LD3320 芯片进行时，都需要先把 SCS 拉低，再拉高。而不是简单地把 SCS 一致拉低。同时，要注意在拉低 SCS 后，增加几个 nop，再启动单片机的硬件 SPI 端口进行读写，保证时序的稳定性。
- d. 如果用户使用软件模拟 SPI 程序，当读写出现不正常时，可以在模拟程序中有 nop 和 delay 的地方增加 nop 的数目，来保证 SPI 的时序稳定。
- e. 使用软件模拟 SPI 程序时，是在 SDCK 下跳沿之前增加 NOP，这是因为 LD3320 芯片的 SPI 模式是下跳沿有效。开发者在移植这个模拟程序时，要注意不要改变模拟程序的时序。

目前碰到的软件编程问题有：

提供的参考程序源代码，是完全可以稳定运行识别和播放的源代码。相当于 LD3320 的驱动。开发者在没有调试通过以前，请一定**不要修改**提供程序中的任何对于寄存器的读写操作！

如果用户没有搭建读写 flash 的硬件平台，只是想实现语音识别功能，可以把参考程序中对于 PlaySound() 函数的调用注释掉就可以了。

参考程序中各个函数的逻辑功能，在“开发手册.pdf”中都有详细介绍和说明，用户应该先仔细阅读“开发手册”，再来阅读参考程序。

[或者向 info@icroute.com](mailto:info@icroute.com) 发邮件索要当前最新版本的参考程序。

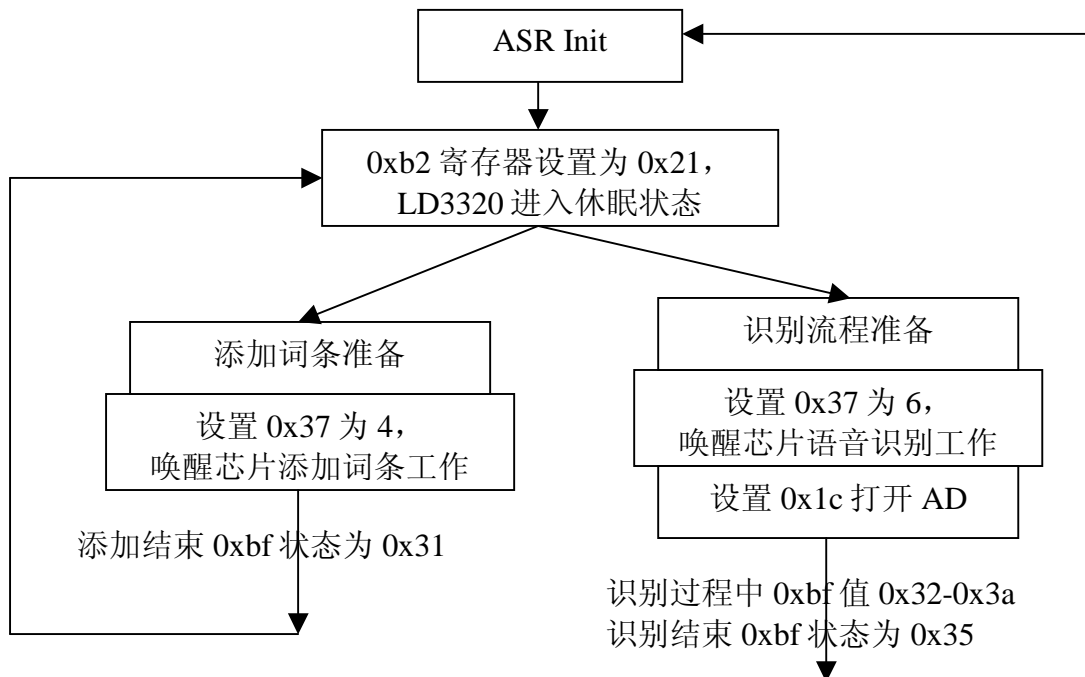
用户选用自己的晶振并连接到 LD3320 芯片后，应该根据提供的参考程序修改 LDChip.h 中“#define CLK\_IN”的数值，将该数值修改为用户使用的晶振频率，比如“8”或者“22.1184”等，以匹配用户的晶振频率。

## 6. 并口连接情况下对于 0x37 寄存器的读写时序

0x37 寄存器是一个特殊的寄存器，通过设置 0x37 寄存器，来启动芯片内部的语音识别运算模块。

当用户使用并口方式和芯片连接，并且不是使用主控 MCU 的硬件并口设置，而是使用自己写的并口读写时序时，一定要注意对于 0x37 寄存器进行写数据设置时，需要保证写的速度足够快。否则可能会出现时序错误的风险。

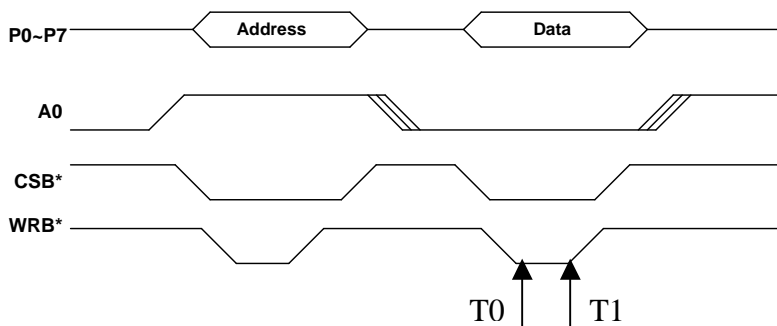
在语音识别流程中，0x37 寄存器是控制命令入口，0xb2 寄存器是内部忙闲的状态寄存器，一些内部状态将由 0xbf 寄存器报告出来



在这个流程中，0x37 寄存器实际上有二重功能，写该寄存器，1). 唤醒芯片内部工作，2). 将值送进去选择哪项工作。

当调用 `LD_WriteReg(0x37, 0x06);` 时，先把 0x37 这个寄存器地址写入了芯片，此时已经唤醒了芯片内部的语音识别模块，需要立即把命令字 (0x06) 传入芯片，否则就会造成语音识别模块接受不到该命令字，造成该语句设置失效。一般的表现就是在 `LD_WriteReg(0x37, 0x06);` 后读取 0xBF 的数值没有按照预期进行变化。

重温一下 LD3320 的并口写时序图：



在图中，标出了“T0”和“T1”二个时间点。对于写 0x37 寄存器，在 T0 时间点的时候，由于已经确定了是写 0x37 寄存器，所以就将芯片内部唤醒并开始语音识别流程；而在 T1 时间点，数据才会随同上跳沿拿进来。对其他寄存器，只是为了设置寄存器的值，这个长度没有什么关系。而对于 0x37 寄存器的双重功能来说，这个时间长度太长就会带来时序上的错误。

请注意：SPI 方式是异步写寄存器的，不会有这个时序错误的风险。

在软件模拟并口的时序中，此时就需要尽可能地避免 delay，在给出的参考程序中，在把 CS 拉低后，到把 CS 再拉高完成一次写，最多只放置了 3 个 nop。用户如果出现 `LD_WriteReg(0x37, 0x06); Delay(100);` 延时一段时间后 0xBF 寄存器没有按照预期进行变化，就需要在并口读写的时序中把这里的写时序尽可能地拉快。

芯片内部为 0x37 寄存器准备了一个 byte 的缓冲用来接受命令字，但是当外部提供的时序过于缓慢时，就是指 [T0, T1] 脉冲太宽时，该缓冲无法正常接受到命令字。

所以用户也可以连续调用两次该设置语句，看是否可以起作用。

（以上的解释也包括对 `LD_WriteReg(0x37, 0x04);` 的说明）

综上，用户如果发现前面的流程正常，仅仅是最后的识别无法正常进行，`LD_WriteReg(0x37, 0x06);` 后 0xBF 寄存器始终没有变化，芯片不停地送出中断，0x2B 寄存器的 bit[3] 始终为 1。可能是因为没有正确地设置 0x37 寄存器导致语音识别模块没有启动而造成。

#### 解决办法：

1. 调整并口的读写时序，保证写寄存器的速度足够快。
2. 权宜之计：尝试连续调用二次 `LD_WriteReg(0x37, 0x04);` 或者二次 `LD_WriteReg(0x37, 0x06);` 看是否可以解决。如果可以解决，还是希望用户可以通过调整并口的写时序，来从根本上解决。

2011 年 3 月 17 日星期四更新