**AD9854详解与驱动程序**

# 一、目的

本文主要介绍AD9854的工作原理，驱动方式以及驱动代码编写，不是为了别的，是为了以后用到的时候能够不用浪费时间重新学习，同时把驱动代码好好完善一下。

# 二、AD9854手册详解

## 1，AD9854特点

1，最高300MHz内部时钟频率。

2，可以工作在FSK，BPSK，PSK，Chirp，AM模式。

3，两个集成12bitDAC

4，4x到20x可编程时钟倍频器

5，两个48bit可编程频率寄存器

6，两个14bit相移寄存器

7，12bit可编程幅度调制器和开关输出形键控功能

8，但因叫FSK和BPSK数据接口

9，输入输出接口提供PSK功能

10，10MHz串行2线或3线SPI，100MHz并行8位编程。

11，3.3V工作电压

12，输出频率最高150MHz

13，双通道I和Q输出，但是频率一定是一样的。

## 2，引脚

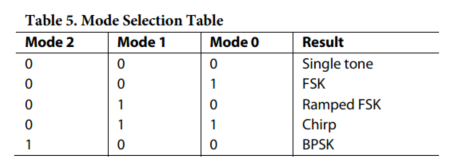
14~19， 如果是并行通信，那么就是地址引脚，如果是串行通信，那么17脚是IO RESET脚，18脚SDO，19脚是SDIO，用于SPI通信的。

48，单极I通道电流输出，或者DAC的余弦

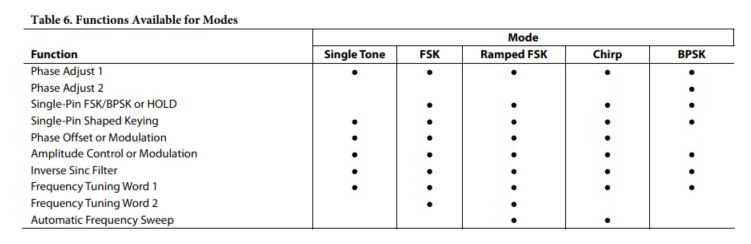
49，互补的I通道单极电流输出，或者DAC的余弦，为什么叫互补呢，因为相位差了180°。

50,51，这两路输出的是正弦信号，也叫Q路信号，互补的两路仍然相位差了180度，但是和I路相位差90°或270°。

## 3，工作方法



AD9854主要有以上几种工作模式，可以通过对1F地址进行写入来控制工作模式。但是不同的模式有一些功能是不能工作的，如下



下面一个一个进行讲解。

**3.1 Single tone模式**

也就是单正弦模式，也是复位之后的默认模式。频率是由Frequency Tuning Word 1 registers的48bit值进行控制。这个值默认是0。复位后默认输出0Hz0相位的信号，I和Q输出信号是满电流一般的信号。下图就是频率从0Hz到自定义频率F1Hz的转变过程。其中Frequency tuning word 可以表示为

其中N就是相位累加器的分辨率（不是相位控制器，因为AD9854是DDS，是依靠相位输出的，通过对频率这种信号进行积分，结果存在相位累加器中，然后再输出），这里N=48，单位是Hz。

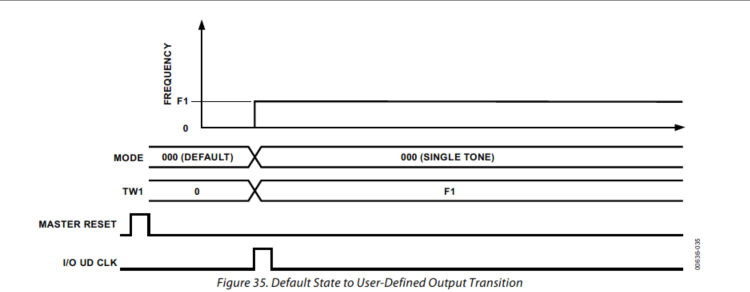
计算出来之后，必须要取整到整数，然后转换成二进制格式，也就是48个1或0,的组合。这个DAC输出的频率可以从直流到1/2 SYSCLK，一般是150MHz。

这种频率上的转变是相位连续的，也就是说新的信号和上一个信号的相位是相关的。

I通道和Q通道的DAC输出永远是有90度的相位差的，14bit的香味寄存器并不能独立地调整各个DAC的相位，相反，两个DACC同时收到相位偏移的改变的影响。

在单频模式下，可以控制一下几个信号属性和质量

1. 输出频率，到48个bit
2. 输出幅度，到12个bit
3. 固定的，用户自定的幅度控制
4. 可变的，可编程的幅度控制
5. 自动的，可编程的，单引脚控制的开关输出形键控。
6. 输出相位，到14bit

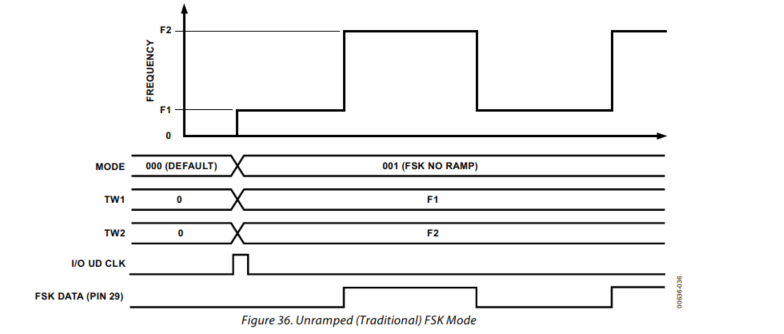


**3.2 FSK，其实是Unramped FSK，就是无斜率FSK 001**

此时DDS输出跟Frequency Tuning Word Register 1和Frequency Tuning Word Register 2有关，也和Pin29有关(FSK/BPSK/Hold)，当29脚低电平输出频率F1，高电平输出频率F2.

频率改变时，它的相位是连续的，只不过有FSK数据到DAC输出有一个延迟。

FSK模式工作模式如下图所示。



**3.3 Rampled FSK， 也叫斜率FSK，频率不是瞬变的，模式010**

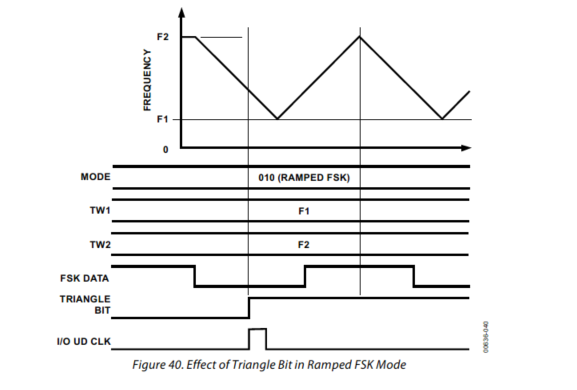
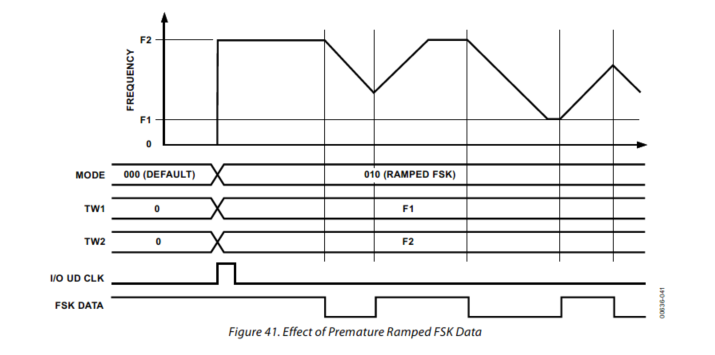


图40是把1F寄存器中的triangle bit置1，然后模式010，此时输出信号就和29脚毫无关系了，DDS自动开始ramp，最终频率做三角形变化。可以通过调整ramp rate clock time period和step size of delta frequency word来调制斜率的变化。

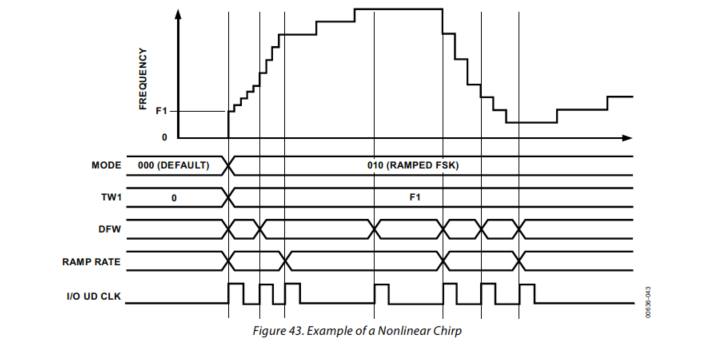
不过要求低频率放在频率控制字1中，高频率放在2中。



FSK引脚是第29脚。

如果1F寄存器triangle bit设置为0， 那么FSK DATA是下降沿，那么就从F2变到F1。如果上升沿就是F1变到F2。

关于扫频斜率调整的话，可以编程48bit的delta frequency word和20bit的ramp rate counter。



**3.4 Chirp模式 011**

FM Chirp的编程步骤

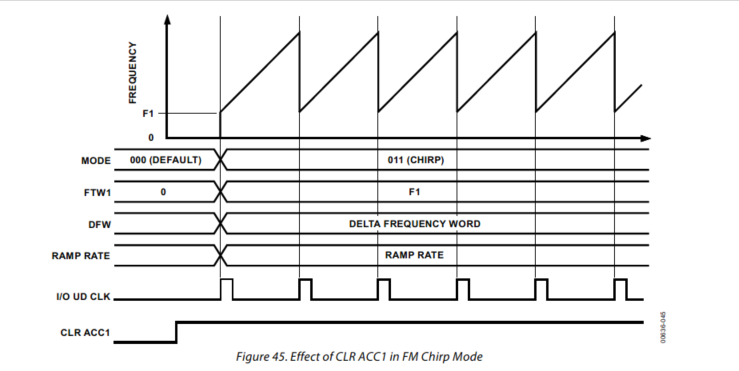
1. 配置起始频率FTW1，寄存器地址从4~9hex 48bit
2. 配置频率步进分辨率，是二进制补码的形式，从10到15hex 48bit
3. 编程频率步进时间，是20bit， 寄存器地址从1A到1Chex
4. 变成玩之后，可以使用一个I/O update脉冲。

二进制补码频率步进分辨率用来决定FM chirp的方向，如果是负数(MSB 是 高)，那么频率想着FTW1变化，如果是整数(MSB 是 低)，那么频率从FTW1向高频率变化。

其中一点很重要的是，FTW1仅仅是FM Chirp的起始点，并没有要求频率返回FTW1。

当FM Chirp信号产生了，他可以随意朝着一个方向移动，只要程序控制好即可。如果

让信号瞬时回到FTW1，只需要控制CLR ACC1和CLR ACC2。当CLR ACC1维持高电平，然后I/O update产生一个脉冲，就能把频率变到FTW1，然后维持原来的Chirp信号变化方式和方向。



其它的感觉不是很有用，就不说明了，用的时候查手册吧。

**3.5 BPSK模式 100**

也就是说信号可以从14bit的两个相移寄存器内键控。键控就是Pin29，BPSK。

如果Pin29是低电平，信号相位选择Phase Adjust Register1，如果Pin29是高，那么选择信号选择相位Phase Adjust Register 2。

总结：基本的BPSK编程步骤

1. 编程载波频率，在FTW1中
2. 编程14bit的香味，Phase Adjust Register1和2都要编程
3. 要在Pin29上进行键控
4. 当准备好之后要激活一下 I/O update

## 4，配置详解

**4.1 内部和外部更新时钟**

更新时钟是由双向I/O引脚，第20脚和一个可编程的32bit下降计数器组成。20脚必须接在一个时钟驱动上（或者IO口模拟时钟），或者由32bit的更新内部产生。

默认模式就是内部时钟（内部更新时钟控制寄存器位是高电平），要想转到外部更新时钟，这个位必须是逻辑低的。内部时钟可以用户自己设置时间间隔，然后自动产生更新脉冲。

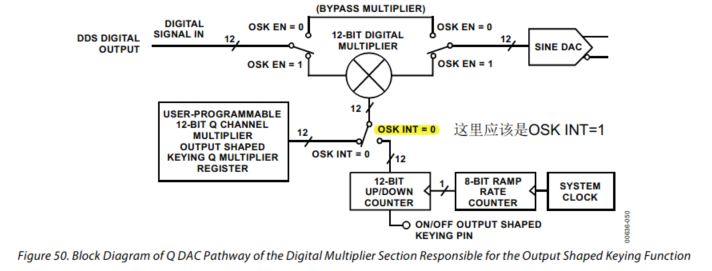
内部更新时钟可以由32bit的更新时钟寄存器(16hex~19hex)，然后设置内部更新时钟控制寄存器位（1F），设为高电平。更新时钟是到1/2系统时钟，这里是150MHz。然后从32bit的二进制值往下技术，当计数器到0了，更新脉冲就会自己产生。这个更新时钟从内部和外部都连到了Pin20上，所以用户可以同步更新信息。脉冲更新脉冲间隔表示为

N就是32bit用户编程的值，N可以从，然后内部这个Updata脉冲的高电平是固定的八个系统时钟周期。如果这个updata clock register是一个小于5的数，那么I/O UD CLK就抑制是高电平，

**4.2 开/关输出键控 OSK**

开关OSK，用户必须使能数字乘法器，可以通过设置OSK EN bit（Control Register Address 20hex），设为高电平。如果OSK EN bit是低电平，那么用于幅度控制的数字乘法器就会被旁路，那么I和Q路的DAC输出将会一直是满幅度。

除了OSK EN bit意外，第二个控制位是OSK INT（20 hex），也要被设置为高电平。这样线性内部控制输出器就可以使能，如果OSN INT是低电平，那么数字乘法器的倍数就是用户可编程的12bit幅度寄存器，这12bit寄存器也被成为Output Shape Key I和Output Shape Key Q，地址分别是21hex和24hex。最大输出幅度是电阻的函数，并且当OSK INT使能的时候不可编程改变。下图可以清楚表明



当然，幅度从0幅到满腹的转换时间也需要被编程，这个转换时间和8bit的ramp rate counter有关，这是一个系统时钟驱动的8bit减数器。当减数器减到0后，会产生一个脉冲，这脉冲会传给一个12bit计数器，然后值会慢慢增加，这个值就是幅度控制值，传给12bit的乘法器。假设系统时钟100MHz，8bit的ramp rate counter是3，那么12bit计数时钟就是25Mhz，如果要从0计数到4095，那么从0幅到满幅的所需时间就是4096\*1/25M = 163.8us。最终，只需改变Pin 30(OSK)的逻辑状态，给30脚一个高电平，OSK INT设为1，这样幅度就会从0变到满，然后给30脚一个低电平，这样幅度就会从满变到0。

**4.3 I和Q路DAC**

Sin由Q路输出，Cos由I路输出，(300MSPS 最大，注意是样本数，而不是频率)，输出的最大幅度由Pin56脚的决定。满幅输出电流是20mA，然而一般10mA是能有最好的无杂散动态范围（SFDR）。一般是

是以幅度表示的，不管用什么表示，DAC的电压输出一般是从-0.5V到1V，超过这个范围可能会造成DAC甚至可能损伤。因此要选择合适的负载阻抗，限制一下这个信号摆幅。

两个DAC一般都会用逆sinc滤波器提前处理，（因为幅度是Sinc型）从而获得从DC到Nyquist的平坦幅度特性，两个DAC都可以通过DAC PD位来掉电，（地址F）。

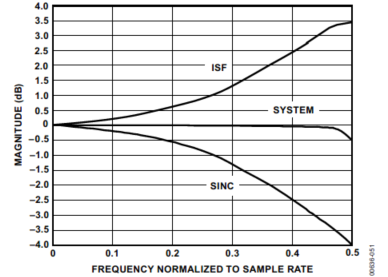
**4.3 控制DAC**

12bit Q路DCA可以被重新配置成控制或辅助DAC。控制DAC可以提供直流电平到外部电路，产生CA信号，或者作为板载比较器做占空比循环。当SRC Q DAC bit（1F hex）为高电平，Q DAC的输入就把内部12bit Q 数据源（默认）切换成外部12bit用户提供的补码，这个数据可以通过串行或并行接口提供，放在一个寄存器里，12bit Q DAC register(26hex和27hex)，最大数据率100MHz，驱动时钟是系统时钟，该DAC和I 路DAC驱动能力一样。电阻同时配置了两个DAC。如果不需要控制DAC，那么可以控制 Q DAC power-down bit(1D hex)，设为高电平来单独把这个DAC掉电。

**4.4 逆Sinc函数**

因为两路的DAC频谱都是以sinx/x滚降的，为了保证大带宽信号，如QPSK不变形，用一个逆sinc滤波器（ISF）来进行频谱补偿。当然这个滤波器可以被旁路掉，来极大地降低功耗，尤其在高时钟频率的情况下。当Q DAC被配置成控制DAC时，逆sinc函数并不起作用。

这个逆sinc滤波器由控制寄存器(20 hex)控制，把bypass inverse sinc bit设为高的时候，这个滤波器就会被旁路掉。



**4.5 REFCLK倍频**

REFCLK倍频是基于PLL的，这个倍频大小可以从4x到20x。所以输入时钟是20MHz，倍频15倍就达到了最大系统频率300MHz，倍频大小可以通过控制寄存器（1E）来设置、

当然，这个倍频器可以被旁路掉，使得AD9854直接用外部时钟。也就是说REFCLK可以通过倍频器的输出来充当，也可以通过REFCLK输入来充当。REFCLK可以是单端也可以是查分的，只不过是可以设置Pin 64脚，DIFF CLK ENABLE的高低来设置。

**4.6 PLL范围bit**

PLL范围bit用于选择这个REFCLK的频率倍频器范围，如果内部时钟频率是200MHz~300Mhz，那么这个range bit应当设置为逻辑1，如果小于200MHz，那么PLL range bit应该设置为逻辑0，这个PLL range bit主要是调整PLL环路参数的，用来获得最好的相位噪声表现。

**4.7 PLL滤波器**

PLL滤波器脚，Pin61，可以对PLL环路滤波器提供外部零点补偿。一般是接一个1.3K电阻和一个0.1uF电容串联，另一端尽可能离Pin60近一点。如果还想要更好的噪声表现，也可以把1E寄存器里面的旁路PLL位置1。

**4.8 使能差分REFCLK**

给DIFF CLK ENABLE引脚一个高电平就可以使能差分时钟输入，差分引脚分别是Pin69和Pin68，最小的输入信号幅度需要达到400mVpp，差分信号的共模输入电平可以从1.6V到1.9V.

DIFF CLK ENABLE Pin64是低电平时，就是单端时钟模式，这时候Pin68应当维持低电平或者高电平。

**4.9 高速比较器**

比较器适用于高速，并且翻转速度高于300MHz，抖动低，输入敏感并且内置迟滞。输出电平可以在阻抗50欧姆的情况下达到1Vpp或者高阻的状态下达到CMOS电平。比较器可以单独关掉用于节省电力。这个比较器可以用于时钟产生器，用于把DDS输出的正弦波转为方波。

**4.10 断电**

可以控制1D寄存器，把相对应的bit设置为1为断电，设置为0意味着上电。其中最重要的一点就是逆sinc滤波器非常耗电，如果把它关掉将能极大地节省功耗。如果想要把AD9854全部关掉，那么可以操控1D寄存器的四个PD为全部设为高电平，最终功耗可以降低到10mW。

## 5，对AD9854进行编程

**5.1 数据传输过程**

所有的寄存器都可以被读取和写入，无论是并行操控还是SPI操控。

S/P select（Pin 70）用来选择I/O的模式，如果要选择并行I/O，应当把这个引脚接到VDD上，串行接口应该把它接到GND上。

无论是那种模式，这个IO口的数据都是写进一个buffer里面的，只有把buffer数据传到寄存器中，写入数据才能起作用。当然，这个传输过程可以通过一下两个方式完成。

1. 内部，由用户编程的速率
2. 外部，这个IO操作可以在REFCLK缺失的时候进行，然而要想把buffer传到寄存器，需要外部UD做一个时钟脉冲，从而把buffer传到寄存器。

相关的更新时钟部分可以参照4.1章节。

**5.2 MASTER RESET**

MASTER RESET引脚必须维持高电平，至少要10个系统时钟周期。这会初始化通信总线并且初始化寄存器。

**5.3 并行IO**

6个地址线，8个双向数据线，分离的读写控制线。可以每10.5ns进行一次写操作，但是读操作在100MHz下不能保证。

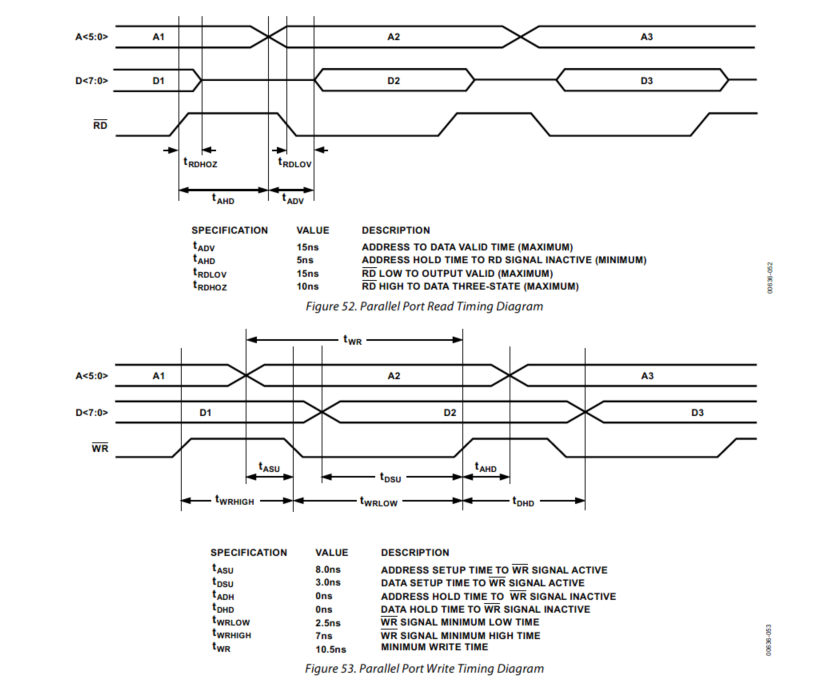
**5.4 串行IO**

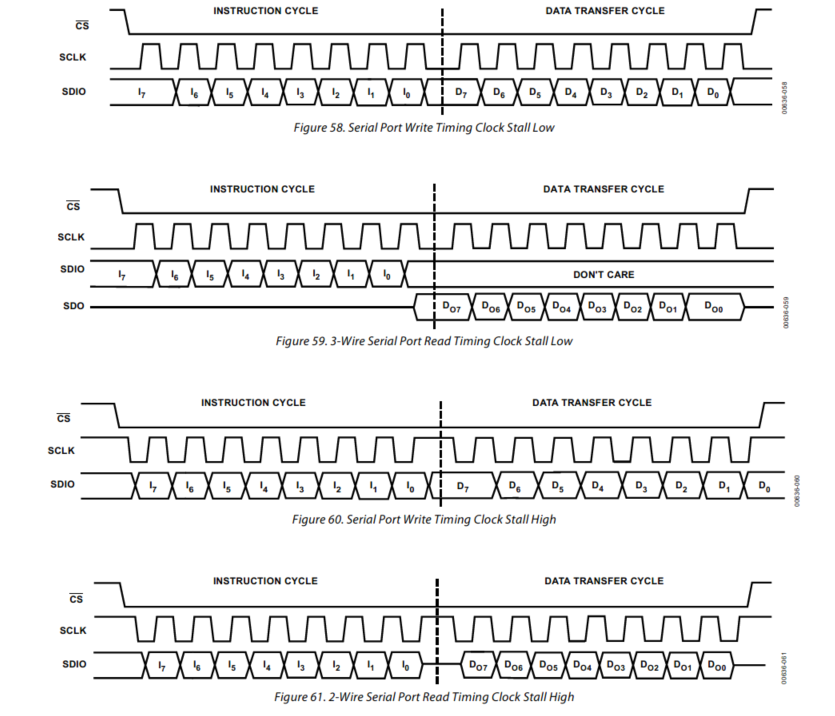
串行IO非常灵活，兼容很多传输格式，如Motorola 6905/11 SPI和Intel 8051 SSR协议。可以通过一个引脚(SDIO)或两个单向引脚，输入SDIO和输出SDO。传输支持MSB在前或者LSB在前，最高10MHz。

在用串行IO时，最好用外部的I/O Update时钟模式，防止更新事件在串行通信过程的时候发生。

要想退出默认的内部更新时间，应当在上电的时候，在开始REFCLK信号开始之前但是在Master reset之后，编程AD9854到外部更新时钟模式。然后再开始REFCLK，从而将数据传输到寄存器，迫使设备到外部更新模式。

其它的就不看了，主要就是如何如何操作，但是AD9854的特性以及结构已经基本的说清楚了；。





# 三、驱动程序调试过程

## 1，数据解析

蓝牙的通信协议我已经放在文件夹里了，最重要的是如何根据所设计的蓝牙通信协议解析出数据。有这样几个问题，我是这样解决的。

（1） 如何把串口接收到的数据按位提取出来

答：因为串口数据都是通过8bit数组传输的，所以通过定义一个当前读取数据的位置即可，这里是current\_index，这样每读取一次，current\_index自增一次即可。

（2） 虽然能读取数据了，但是不同的命令其对应的数据长度也是不定的，这怎么解决？

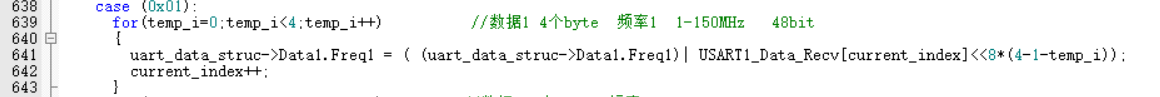
答：cmd = 0x01的Data2，是2个byte，那么我就在命令解析的case (0x01)中把current\_index自增两次即可全部提取出来，也就是一个for循环的事。而且为了保证数据空间不浪费，我用了Union来节省空间，即Data1一个Union，Data2一个Union，Data3和Data4因为都只有一个，所以就不用Union了。

（3）解析出的数据怎么提取出来呢？

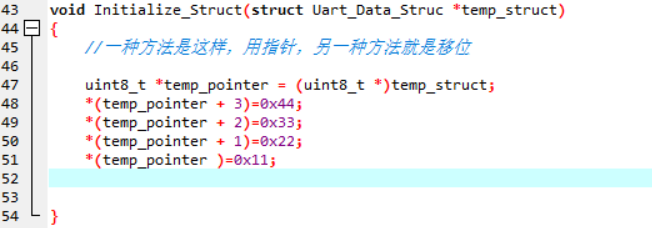
我构建了一个名为Uart\_Data\_Struc的结构体，包括了cmd，Data1，Data2，StepFreq，Clock\_Freq\_Control，其中Data1和Data2是Union。然后把结构体指针作为参数传递到函数里面，然后在函数中提取对应的参数对结构体赋值。

（4）串口传输的数据是8bit，我怎么样把串口发送的8bit数据赋值给4字节的uint32\_t数据呢？

假设我设置频率为1MHz，那么转化为十六进制就是00 0f 42 40。那么接收到的数据也是00 0f 42 40，因为一次只能去8bit，可以先取00左移24位，然后并上0f左移16位，然后并上42左移8位，最后并上40即可。具体实现是这样的。



也可以利用8bit指针指向结构体，然后利用指针赋值。

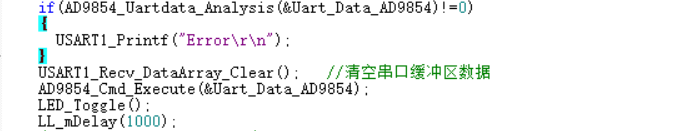


不过因为是结构体，指针指向的位置就是结构体第一个成员的指针，这里是cmd。而要想指到Data1，就需要计算cmd所占空间，然后把每个成员的起始指针求出来，然后再分别利用上述方法赋值，这样确实有点麻烦了。

（5）解析出数据之后，怎么执行相对应的命令。

解析出数据之后，需要立刻清零串口接收区，防止下一次数据不能从头存放。然后把结构体作为指针传递到执行函数中，通过判断结构体的cmd，执行对应的命令即可。

不过需要注意的一点就是，串口接收数据，数据解析，串口缓冲区清零还有命令执行的顺序应当仔细设计。



我这里是最后有个延时1s，反正有一点延时就行。这个延时占据了绝大多数时间，一般串口接收数据时，程序一般在延时，然后中断后立刻读取数据，解析数据。解析数据后立刻清零，然后执行相对应的命令，很少出错。如果想稳一点，也可以在解析数据到执行命令之间进行原子操作，也就是说关掉中断，保证这三条语句连续。等待语句执行结束后进行中断。

## 2，连上AD9854的排线之后，蓝牙无法工作

实际上，AD9854模块的电压是通过LM317而产生的，而LM317并不是稳压芯片，而是电压调整芯片，输出电压和输入电压强相关。由于我输入电压设置的是5V，导致LM317的输出电压是2.7V，这样原本3.3V的输出引脚实际只有2.7V左右。然后连上排线之后，单片机的3.3V输出和2.7V接在了一起，导致实际3.3V的电压最终被拉到了3V左右，然后蓝牙就无法工作了，但是单片机仍然工作正常。

最后我把我设计的PCB上的3.3V给掰掉了，这样就是两个模块共地，但是分别供电，于是问题解决。