基于LGTSDK Builder

LGT8F690A 快速开发系列教程

第九篇: ADC的使用



本篇为系列教程的第九篇。如果需要了解教程相关的软件硬件环境,请参考本系列教程的第一篇:《LGT8F690A快速开发系列教程第一篇_急速上手》

LGT8F690A内部集成一个12位的逐次逼近型模数转换器(ADC)。支持最高2MHz采样速率。模拟输入通道支持来自外部的AN0~AN7,以及来自内部其他外设的模拟输出。 ADC的参考电压也有多种选择,可以选择内部参考电压,也可以选择来自系统电源或者外部参考输入(RA7)。

下面为LGT8F690A内置ADC的功能以及相关参数:

ADC功能/参数	功能描述		
工作电压	2.5V ~ 5.5V		
参考电压源	 来自系统电源电压(VCC) 来自外部参考电压(AVREF/RA7) 来自内部1.5V±1%基准电压源 来自内部2.56V±1%基准电压源,需要工作电压在3.0V以上 		
输入通道	8路外部输入通道(ANO~AN7) 内部通道支持: 1. 来自内部6位DAC的输出 2. 来自内部差分放大器(DAP)的输出 3. 来自内部1/5分压电路的输出 4. 来自内部参考电压源(1.5V/2.56V) 5. 来自系统地(GND)		
采样速率	最高2MHz ADC的转换速率选择与当前ADC输入通道的输入驱动电流有关。对应高阻抗的输入源,由于输入电流较小,请选择较低的采样速率。		
失调校准	LGT8F690A内部ADC支持失调校准。 校准后,可在全量程内达到5mV内的测量误差精度。		
自动触发	ADC支持来自内部、外部事件的自动触发转换: 1. 来自TIMER1的通道3比较匹配事件 2. 来自TIMER2的通道4比较匹配事件 3. 来自外部RA1引脚的上升沿 4. 来自外部RB3引脚的上升沿		
自动通道监控	LGT8F690A内置ADC可以自动实时监控一个通道的电压,当输入电压低于或者高于预设值的阀值时,将会触发溢出事件。		

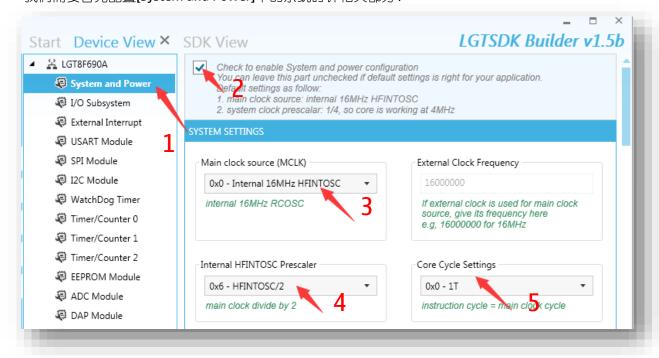
ADC的转换精度除了依赖ADC本身的性能外,还依赖于参考电压的精度。LGT8F690A内置1%可校准参考电压源,可以配置为1.50V/2.56V两档电压输出。

由于早期提供的工程片参考电压没有校准,因此可能会带来比较大的测试误差。为了能够更好的完成本篇校准,我们会先用测试电源电压的方式,校准内部参考。校准的原理也非常简单,我们动态的改变内部参考的校准,然后通过串口输出ADC测量得到的当前电源电压。根据已知供电电压的情况下,我们从串口的打印信息,可以找到内部参考的校准值。

下面我们就以LGTSDK Builder为例, 使用ADC来校准内部参考,这里我们选择1.5V的内部参考。 首先,启动LGTSDK Builder, 新建一个工程, 选择目标芯片,工程名称:lgt8f690a_adc

接下来是配置我们将要用到的外设资源:

因为我们要使用串口,我们需要告知SDK当前的时钟配置,以便SDK可以准确的计算波特率。 因此我们需要首先配置[System and Power]中的系统时钟相关部分:

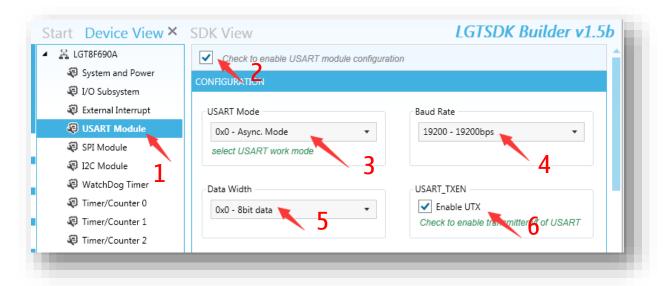


配置说明:

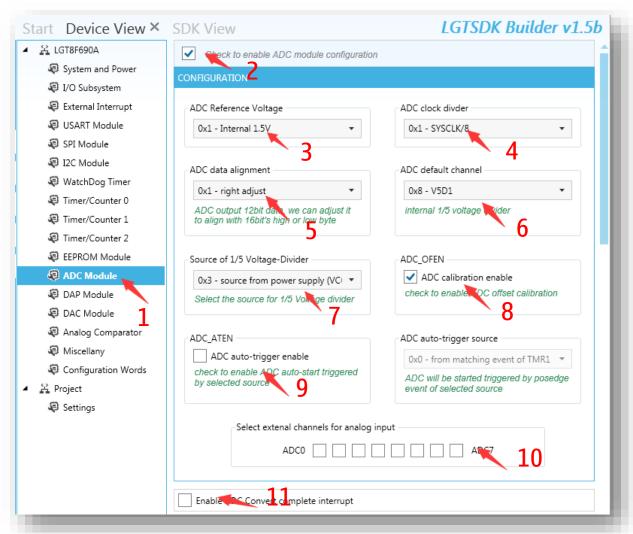
- 3: 主时钟源选择使用内部16MHz RC振荡器;
- 4: 这里我们选择时钟源的2分频,得到最终8MHz的系统运行时钟;
- 5: 因为后面有些电压转换的计算, 我们这里让内核跑1T;

主时钟的配置没有固定要求,可以根据自己的应用情况灵活调整!

下面是串口相关的配置。 这里只用于和**PC**通讯。 我们选择比较常规的配置:



最后是ADC相关的配置,这里我们将会详细说明下:



ADC相关配置说明:

- 3: 选择ADC的参考电压源, 这里我们选择使用1.5V内部参考, 后面我们会先校准它;
- 4: ADC的采样时钟,使用系统时钟的分频产生。我们已经配置了8MHz的系统时钟,这里我们选择使用系统时钟的8分频(SYSCLK/8),也就是ADC将用作于1MHz的采样速率;
- 5: ADC转换数据的对齐方式, 这里我们选择的是常用的右对齐, 低位在低字节;
- 6: ADC的默认选择的通道,这里选择ADC初始化是的选择通道。大部分情况下,我们是使用SDK中ADC转换接□函数的参数指定转换通道,因此这里不是必须的。 但是,我们需要使用内部内部1/5分压电路测量内部电源,这里需要选择内部1/5分压通道,然后我们才能够选择1/5分压电路的输入源。
- 7: 上一步选择内部1/5分压通道后, 这里被使能。 我们选择1/5分压源为芯片的电源(VCC);
- 8: 使能芯片的系统校准。 勾选此项后, ADC将在初始化时, 执行失调校准。
- 9: ADC自动触发转换功能,我们这里不用。
- 10: 这里是选择将要作为模拟输入的外部通道, 这里的选择是让SDK标注我们的引脚使用情况, 并不会影响最终ADC的输入功能, 因此这里是可选的。通常可以不用选择。
- 11: ADC的转换完成中断, 我们不使用中断功能, 此处不用勾选。

ADC的配置完成。 最后, 通常我们也勾选下[Miscellany]配置页, 把一些辅助函数包含进来!

所有的配置完成后,点[Device View] 下的 [Build], 生产SDK工程。

然后是代码实现部分,我们首先实现的代码是要完成内部参考的校准。代码逻辑如下:内部参考可以通过VRTUNE寄存器设置,我们首先将VRTUNE设置到一个最低值,然后在这个参考校准值下,使用ADC转换测量电源电压的1/5通道,然后转换成电源电压,通过串口打印到PC;然后逐次增加VRTUNE的值,同样进行ADC测量和打印。这样在PC端的串口接收程序里,我们就可以看到每个VRTUNE下,ADC测量到的电源电压值。我们从中选择最合适。

下面先简单介绍我们将要使用的SDK函数接□。 这里我们将会用的ADC转换的函数。 在SDK代码编辑器内,输入adc,或者analog,也可以在日志窗□中,看到所有ADC相关的接□函数:

函数名称	功能描述	使用方法
adcEnable()	使能ADC模块	adcEnable()
adcDisable()	禁用ADC模块	adcDisable()
adcSetReference()	设置ADC的参考电压源	adcSetReference(ADC_REFS_1D50) 设置参考电压为内部1.5V adcSetReference(ADC_REFS_2D56) 设置参考电压位内部2.56V
adcSetChannel()	设置ADC当前的转换通道	adcSetChannel(ADC_CHMUX_AN0) 设置当前转换通道为外部输入AN0
adcRead()	启动ADC转换当前通道	Value = adcRead()
adcReadChannel()	启动ADC转换指定通道	Value = adcReadChannel(ADC_CHMUX_V5D) 启动ADC转换内部1/5分压通道的输入电压

下面是代码部分, 根据目前芯片的特性, 内部1.5V内部参考, 只需要在 $0x18 \sim 0x38$ 之间即可。 因此我们定义VRTUNE变化的范围是在 $0x18 \sim 0x38$ 之间。

为了方便打印输出, 我们定义了两个辅助的打印函数printInt/printIntLn,两个函数都是将一个整形数据转换成字符后,送串□输出;区别是后者可以会打印换行符。

```
LGTSDK Builder v1.5b
SDK View X
main.c X
6
7
    // Import external definitions
    extern void init_modules(void);
8
9
10
    #define VRTUNE_MIN 0x18
                             参考校准值的范围
    #define VRTUNE MAX 0x38
11
12
13
    void printInt(char *prefix, u16 data, u8 dec)
14 🗏 {
15
        u8 caTmp[8];
16
                                      将data转换为字符串,然后通过串口打印
17
        axu utoa(caTmp, data, dec);
                                      Prefix为打data字符串前,先输出的前导提示信息
18
        usartWriteString(prefix);
                                      Dec为指定data转换字符的进制,可以16机制,10进制等
        usartWriteString(caTmp);
19
   -}
20
21
    void printIntLn(char *prefix, u16 data, u8 dec)
22
23 🗏 {
24
        printInt(prefix, data, dec);
25
                                           与printInt相同, 但会打印换行符
        usartWriteString((char *)"\r\n");
26
```

下面是主程序部分:

```
LGTSDK Builder v1.5b
                          SDK View X
Start Device View

    Project 'lg8f660a_adc'

                          main.c X
                          26 L

    Environment

                          27
     Includes
                          28
                              int main(void)
       allinone.h
                          29 🗏 {
                          30
                                  u8 vrcal = VRTUNE MIN;
                                                          初始化校准值
       FSYSCLK.h
                          31
                          32
                                  u16 adcVal = 0; -

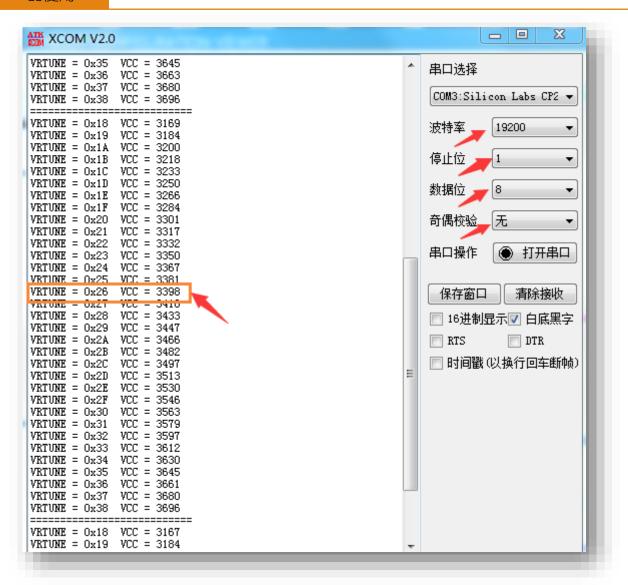
    存储ADC转换结果

       🗓 global.h
                          33
       🗓 lgt8f690a.h
                          34
                                   // Device initialization
                          35
                                  init_modules();
       Igt8f690a_adc.h
                          36
                          37
                                  usartWriteString((char *)"Hello from LGT8F690A!\r\n");
       Igt8f690a_misc.h
                          38
       Igt8f690a_sys.h
                          39
                                  // Add your code from here
                          40
                                  while(1)
       Igt8f690a_usart.h
                          41 🖨
                                      adcVal = 0;
                          42
                                                            - 清零上次转换结果
       macros_auto.h
                          43
                                      // 8x samples
     Sources
                          44
                                      for(u8 i = 0; i < 8; i++)
                          45
                                          adcVal += adcReadChannel(ADC_CHMUX_V5D);
                                                                                  ADC转换8次,求均值
       Igt8f690a_adc.c
                                      // average
                          46
                          47
                                      adcVal >>= 3;
       Igt8f690a_misc.c
                          48
       Igt8f690a_sys.c
                                      // mV = adcVal*1500*5/4096 => adcVal*1.83
                          49
                                                                                转换为mV单位电压值
                          50
                                      adcVal = (u16)((float)adcVal * 1.83);
       🖆 lgt8f690a_usart.c
                          51
                                                                                注意要放大5倍!
    init_auto.c
                          52
                                      printInt((char *)"VRTUNE = 0x", VRTUNE, 16);
                          53
                                      printIntLn((char *)" VCC = ", adcVal, 10);
                                                                                  串□打印参考校准值
    nain.c
                          54
                                                                                  和对应的ADC结果
                                      // VRTUNE step
                          55
                          56
                                      if(++vrcal > VRTUNE MAX) {
                          57
                                          vrcal = VRTUNE_MIN;
                          58
                                          usartWriteString((char *)"===
                          59
                                                      - 逐次增加校准值, 当达到最大时,打印分隔符
                          60
                          61
                                      VRTUNE = vrcal; ← 写入校准寄存器,完成参考的校准
                          62
                          63
                                      delayms(100); ◆ 参考稳定, 并间隔两次测量
                          64
```

代码完成, 编译成功后, 下载到最小测试。

将最小板上的串口通过串口转换接口连接PC。 在PC上启动串口工具:

以上基准过程,仅仅作为针对内部参考没有校准的工程样片。量产版本的芯片,可以不用执行此过程!



以我的测试环境为例, 芯片工作时, 测量VCC的电压是3.400V。从串口的输出中, 找到与3.400V 最近的电压测量输出。如上图, 此时对应的参考校准值为: 0x26。

这样,我们就找到了参考的校准值!以后,在代码初始化时,执行 VRTUNE = 0x26;即可完成对内部1.5V参考电压的校准。

如果你使用的是2.56V参考来测试,可以根据输出情况,调整VRTUNE_MIN/MAX。

注意事项: 请准确的确认芯片工作时的工作电压, 这将影响校准的结果!

内部参考的校准已经完成,下面我已一个正常使用的应用,示例ADC的使用!

例程将使用内部1.5V参考电压,ADC通过内部1/5分压电路监控芯片的电源电压。同时我们还将用ADC分时采样另外一路外部输入ANO/RBO的输入电压。并将实时采样到的两路电压转换为mV电压值,通过串口发送的PC上显示。

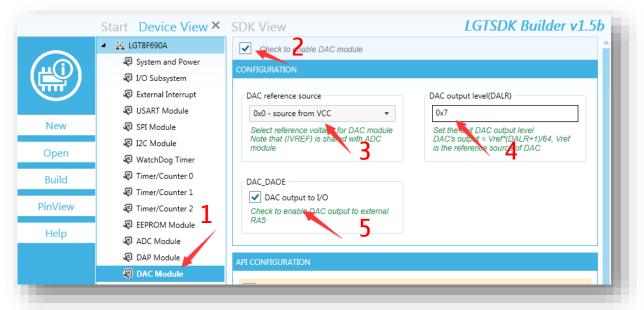
ANO/RBO的外部输入电压,可以来自外部的其他输入源。为了方便测试,我们将使用LGT8F690A内部的6位的DAC产生的电压输出,作为ANO/RBO的输入。LGT8F690A内部的DAC的电压,可以输出到芯片的RA5引脚上,这个引脚并非ADC的输入通道,但是RA5傍边的RBO是ADC的外部输入通道0,我们可以使用跳线将RA5与RBO短接起来。这样就可以通过调整DAC的输出,给ANO提供一个测试源。

这里,DAC的输出,我们也将做成一个可通过串□调整的。 通过串□想芯片发送一个字节的HEX数据,芯片收到这个数据后, 用这个数据调整DAC的输出电压。

例程中,我们将会使用到一个新的读串口数据的接口函数,usartGetCharAsync(), 这是一个异步实 现。该函数并不会等待串口接收到一个字节。如果函数执行时串口接收到有效数据,函数将数据 存储到参数地址的地址,并返回数据有效状态;如果串口没有数据, 直接范围数据无效状态!

函数名称	功能描述	使用方法
usartGetCharAsync()	如果串口有数据,将数据保存到指定的地址,返回1; 否则直接返回0	<pre>if(usartGetCharAsync(&data) == TRUE) { // data in valid now }</pre>
dacSetLevel()	设置DAC的输出电压值	dacSetLevel(0x7) DAC的输出将为: VDAC*(0x7 + 1)/64 其中, VDAC为DAC的参考电源

因为我们要用到DAC, 我们在SDKBuilder上回到[Device View]配置页,来增加DAC相关的设置.下面是LGT8F690A的DAC配置页设置:



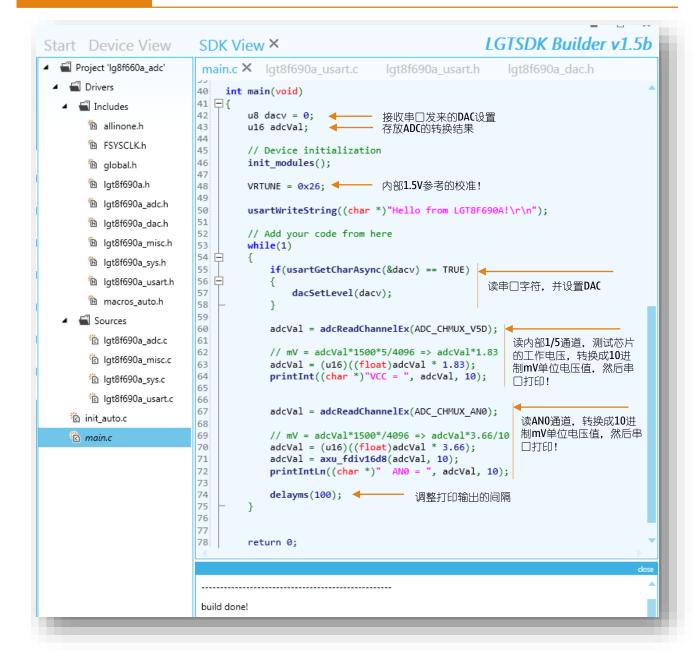
简单介绍下DAC相关配置:

- 3: 选择DAC的参考电压源, 这里我们选择VCC, 即芯片的工作电源电压。
- 4: DAC的电压值设置寄存器,这个值将决定DAC的输出电压值。因为我们将会用串口调整这个值,这里只有随便给一个电压值就可以了。但需要注意,我们的ADC是用1.5V的内部参考,当DAC的输出电压大于1.5V时,ADC的转换结果将会满偏。6位的DAC,这值的范围是0x00~0x3F;
- 5: 勾选此处,DAC的输出与RA5连接。

DAC的配置完成,在[Device View]视图中,点[Build],重新生产SDK代码,回到SDK View中。

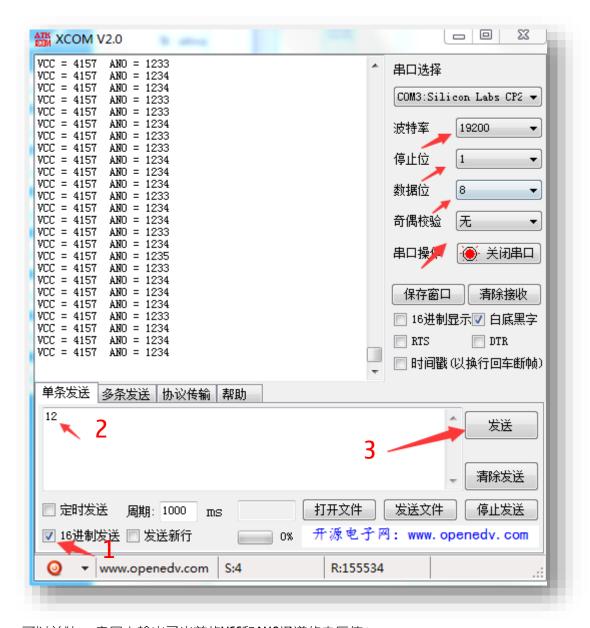
下面是代码部分:





代码中需要注意,采样ANO通道没有经过内部1/5通道,因此转化到输入源时,不用放大5倍。 代码编写完成,编译通过后,启动LGTMix_ISP,将生成的HEX下载到最小板上测试!

测试前,先使用跳线帽将最小开发板上的RBO引脚与其旁边的RA5引脚短接,然后将最小板的串口与PC串口连接,PC上打开串口助手工具。



可以单独,串口上输出了当前的VCC和ANO通道的电压值!

下面说明如何通过串□控制DAC的输出,进而改变ANO的输入电压: 在串□助手上,一般都有HEX模式发送数据的功能,比如上面的XCOM

- 1: 勾选此处的16机制发送功能;
- 2: 这里填发送的数据, 我们填写12, 就是发送一个字节的数据0x12
- 3: 点击这里的"发送"按键,即可将数据发送出去。

数据发送后,然后就可以从串口的输出上看到,ANO的采样数据发送了变化。同时可以用万用表测量RA5或者RBO、这里是当前DAC的输出值,应该可以和串口打印的ANO一致!