第 5 章　模数转换 ADC

今天你是否成功取决于你昨天的态度，今天的态度决定了你明天是否成功。

——莎士比亚

微控制器或微处理器所处理和传送的都是不连续的数字信号，而实际中遇到的大都是连续变化的模拟量。模拟量经传感器转换成电信号的模拟量后，需经模数 AD 转换变成数字信号才可输入到数字系统中进行处理和控制，因而作为把模拟电量转换成数字量输出的接口电路——AD 转换器，是现实世界中模拟信号转向数字信号的桥梁，是电子技术发展的关键和瓶颈所在。

5.1　ADC 总体特性

为了能够使用数字系统（如 MCU）处理模拟信号，必须把模拟信号转换成相应的数字信号。能够实现这种转换的电路称为模数转换器（Analog-to-Digital Converter，ADC）。ADC 能够将连续变化的模拟电压转换成离散的数字量。

Stellaris 系列 ARM 集成有一个 10 位的 ADC 模块，一般支持 8 个输入通道，以及一个内部温度传感器。ADC 模块含有一个可编程的序列发生器，可在无须控制器干涉的情况下对多个模拟输入源进行采样。每个采样序列均对完全可配置的输入源、触发事件、中断的产生和序列优先级提供灵活的编程。

Stellaris 系列 ARM 的 ADC 模块提供以下一系列的特性。

- 最多 8 个模拟输入通道，LM3S8962 为 4 个模拟通道；
- 单端和差分输入配置；
- 内部温度传感器，占用 1 个通道；
- 高达 1Ms/s（每秒采样一百万次）的采样率。LM3S8962 采样率为 500ks/s；
- 4 个可编程的采样转换序列，入口长度为 1～8，每个序列均带有相应的转换结果 FIFO；
- 灵活的触发控制：处理器（软件）、定时器、模拟比较器、PWM、GPIO；
- 硬件可对多达 64 个采样值进行平均计算（牺牲速度换取精度）；
- 转换器采用内部的 3V 参考电压；
- 分开的模拟电源和模拟地，跟数字电源和数字地分离。

图 5-1 从右往左看，模拟信号从外部输入，经过 10 位 AD 转换后，采样结果经过硬件平均电路（64 次），然后送到 FIFO 块，FIFO 块内部有 4 个 FIFO，每个 FIFO 对应不同的采样通道，每个 FIFO 块对应一个采样序列发生器，不同采样序列发生器对应一个中断输出。采样控制对应 4 个选择器，每个选择器对应的 4 个触发控制事件（比较器、GPIO（PB4）、定时器、PWM）控制采样。

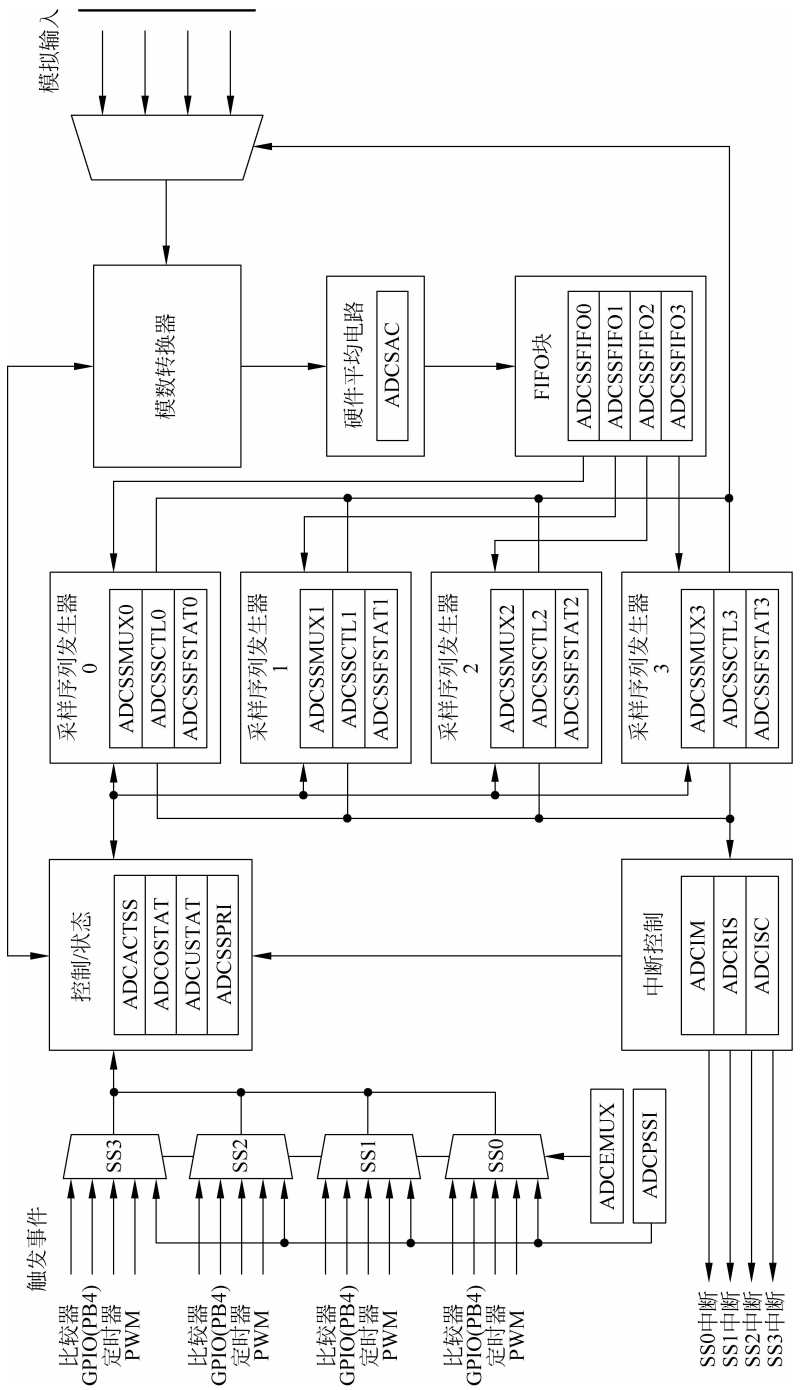
模拟输入
模数转换器
硬件平均电路
ADCSAC
FIFO块
ADCSSFIFO0
ADCSSFIFO1
ADCSSFIFO2
ADCSSFIFO3
采样序列发生器 0
ADCSSMUX0
ADCSSCTL0
ADCSSFSTAT0
采样序列发生器 1
ADCSSMUX1
ADCSSCTL1
ADCSSFSTAT1
采样序列发生器 2
ADCSSMUX2
ADCSSCTL2
ADCSSFSTAT2
采样序列发生器 3
ADCSSMUX3
ADCSSCTL3
ADCSSFSTAT3
控制/状态
ADCACTSS
ADCOSTAT
ADCUSTAT
ADCSSPRI
中断控制
ADCIM
ADCRIS
ADCISC
SS3
SS2
SS1
SS0
ADCEMUX
ADCPSSI
触发事件
比较器
GPIO(PB4)
定时器
PWM
SS0中断
SS1中断
SS2中断
SS3中断

图 5-1　ADC 结构框图

5.2　ADC 功能描述

Stellaris 系 ARM 的 ADC 通过使用一种基于序列(Sequence-based)的可编程方法来收集采样数据，取代了传统 ADC 模块使用的单次采样或双采样的方法。每个采样序列均为一系列程序化的连续(back-to-back)采样，使得 ADC 可以从多个输入源中收集数据，而无须控制器对其进行重新配置或处理。对采样序列内的每个采样进行编程，包括对某些参数进行编程，如输入源和输入模式(差分输入还是单端输入)，采样结束时的中断产生，以及指示序列最后一个采样的指示符。

1. 采样序列发生器

采样控制和数据捕获由采样序列发生器进行处理。所有序列发生器的实现方法都相同，不同的只是各自可以捕获的采样数目和 FIFO 深度。表 5-1 给出了每个序列发生器可捕获的最大采样数及相应的 FIFO 深度。在本实现方案中，每个 FIFO 入口均为 32 位(1 个字)，低 10 位包含的是转换结果。

表 5-1　ADC 序列发生器的采样数和 FIFO 深度

序列发生器	采样数	FIFO 深度	序列发生器	采样数	FIFO 深度
SS0	8	8 个字	SS2	4	4 个字
SS1	4	4 个字	SS3	1	1 个字

对于一个指定的采样序列，每个采样均可以选择对应的输入引脚、温度传感器、中断使能、序列末端和差分输入模式。

当配置一个采样序列时，控制采样的方法是灵活的。每个采样的中断均可使能，这使得在必要时可在采样序列的任意位置产生中断。同样地，也可以在采样序列的任何位置结束采样。例如，如果使用序列发生器 0，那么可以在第 5 个采样后结束并产生中断，中断也可以在第 3 个采样后产生。

在一个采样序列执行完后，可以利用函数 ADCSequenceDataGet()从 ADC 采样序列 FIFO 里读取结果。上溢和下溢可以通过函数 ADCSequenceOverflow()和 ADCSequenceUnderflow()进行控制。

上面是在说采样序列发生器的工作情况，大家看半天，不知道被绕晕了没有。理解如下：采样序列发生器就是可以多次采样数据的一个控制器，可灵活配置采样的次数。通过配置采样序列发生器，可以在采样动作完成后，产生中断标志，可以从寄存器中查询采样结果数据。因此采样序列发生器就是一个缓冲寄存器，目的是确保驱动采样数据。

2. 模块控制

在采样序列发生器的外面，控制逻辑的剩余部分负责对中断产生、序列优先级设置和触发配置等任务。大多数的 ADC 控制逻辑都是在 14～18MHz 的 ADC 时钟速率下运行。当选择了系统 XTAL 后，内部的 ADC 分频器通过硬件自动配置。自动时钟分频器的配置对所有 Stellaris 系列 ARM 均以 16.667MHz 操作频率为目标。

3. 中断

采样序列发生器虽然会对引起中断的事件进行检测，但它们不控制中断是否真正被发送到中断控制器。ADC 模块的中断信号由相应的状态位来控制。ADC 中断状态分为原始的中断状态和屏蔽的中断状态，这可以通过函数 ADCIntStatus()来查知。函数 ADCIntClear()可以清除中断状态。

4. 优先级设置

当同时出现采样事件(触发)时，可以将为这些事件设置优先级，安排它们的处理顺序。优先级值的有效范围是 0～3，其中 0 代表优先级最高，而 3 代表优先级最低。优先级相同的多个激活采样序列发生器单元不会提供一致的结果，因此软件必须确保所有激活采样序列发生器单元的优先级是唯一的。

5. 采样事件

采样序列发生器可以通过多种方式激活，如处理器(软件)、定时器、模拟比较器、PWM、GPIO(PB4)。对于某些型号如 LM3S8938 并不存在专门的硬件 PWM 模块，因此也不会存在 PWM 触发方式。外部的外设触发源随着 Stellaris 家族成员的变化而改变，但所有器件都公用“控制器”和“一直(Always)”触发器。软件可通过函数 ADCProcessorTrigger()来启动采样。在使用“一直”触发器时必须非常小心。如果一个序列的优先级太高，那么可能会忽略(Starve)其他低优先级序列。

6. 硬件采样平均电路

使用硬件平均电路可产生具有更高精度的结果，然而结果的改善是以吞吐量的减小为代价的。硬件平均电路可累积高达 64 个采样值并进行平均，从而在序列发生器 FIFO 中形成一个数据入口。吞吐量根据平均计算中的采样数而相应地减小。例如，如果将平均电路配置为对 16 个采样值进行平均，则吞吐量也减小了 16 因子(Factor)。

平均电路默认是关闭的，因此，转换器的所有数据直接传送到序列发生器 FIFO 中。进行平均计算的硬件由 ADC 采样平均控制(ADCSAC)寄存器进行控制。ADC 中只有一个平均电路，所有输入通道(不管是单端输入还是差分输入)都是接收相同数量的平均值。

7. 模数转换器

转换器本身会为所选模拟输入产生 10 位输出值。通过某些特定的模拟端口，输入的失真可以降到最低。转换器必须工作在 16MHz 左右，如果时钟偏差太多，则会给转换结果带来很大误差。

8. 差分采样

除了传统的单端采样外，ADC 模块还支持两个模拟输入通道的差分采样。

当队列步(Sequence Step)被配置为差分采样后，会形成 4 个差分对之一，编号 0～3。差分对 0 采样模拟输入 0 和 1，差分对 1 采样模拟输入 2 和 3，以此类推。ADC 不会支持其他差分对形式，比如模拟输入 0 跟模拟输入 3。差分对所支持的编号有赖于模拟输入的编号(详见表 5-2)。

表 5-2　差分采样对

差分对	模拟输入	差分对	模拟输入
0	0 和 1	2	4 和 5
1	2 和 3	3	6 和 7

在差分模式下被采样的电压是奇数和偶数通道的差值，即：

$$\Delta V = V_{IN\_ENEN} - V_{IN\_ODD}$$

其中，ΔV 是差分电压；V_{IN_EVEN} 是偶数通道；V_{IN_ODD} 是奇数通道。因此：

- 如果 $\Delta V = 0$，则转换结果 = 0x1FF；
- 如果 $\Delta V > 0$，则转换结果 > 0x1FF（范围在 0x1FF～0x3FF）；
- 如果 $\Delta V < 0$，则转换结果 < 0x1FF（范围在 0～0x1FF）。

差分对指定了模拟输入的极性：偶数编号的输入总是正，奇数编号的输入总是负。为得到恰当的有效转换结果，负输入必须在正输入的±1.5V 范围内。如果模拟输入高于 3V 或低于 0V（模拟输入的有效范围），输入电压被截断，即其结果是 3V 或 0V。

在图 5-2 里显示了以 1.5V 为中心的负输入示例。在这个配置中，微分的电压跨度可以从－1.5V 至 1.5V。图 5-3 显示了以－0.75V 为中心的负输入示例，这就意味着在正通道的输入在－0.75V 的微分电压时达到饱和，因为这个输入电压低于 0V。图 5-4 显示了以 2.25V 为中心的负输入，这里，在正通道的输入在 0.75V 的微分电压时达到饱和，因为输入电压有可能大于 3V。

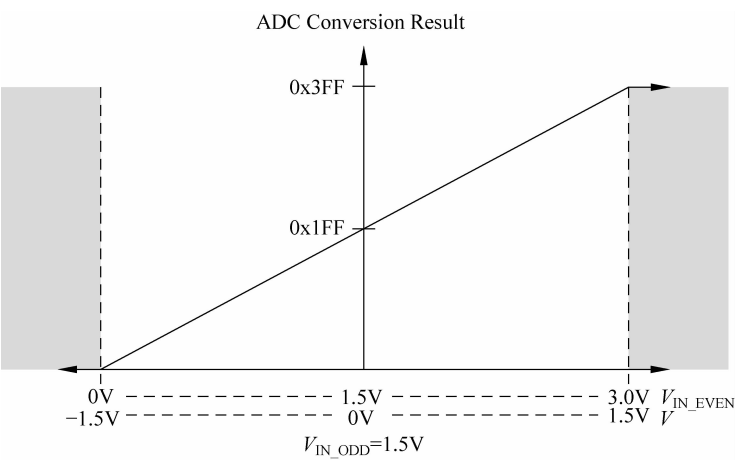

图 5-2　采样范围（V_{IN_ODD} = 1.5V）

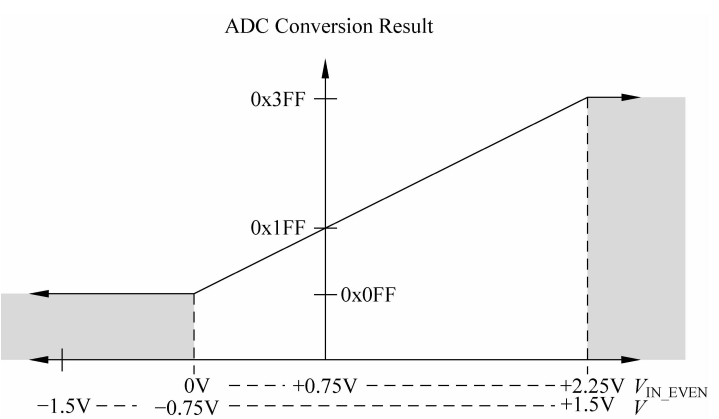

图 5-3　采样范围（V_{IN_ODD} = 0.75V）

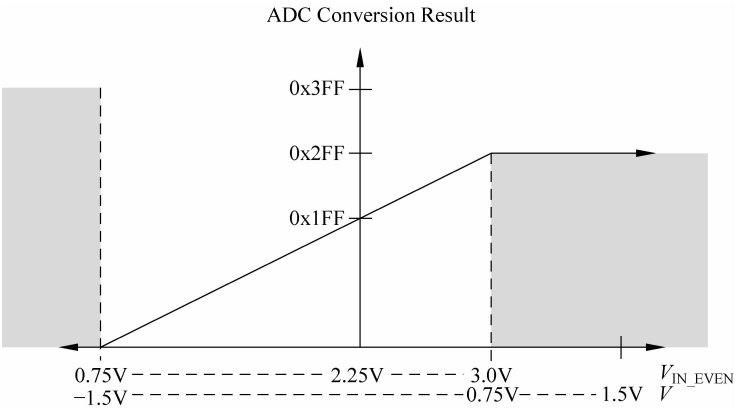

图 5-4　差分采样范围(V_{IN_ODD} = 2.25V)

9. 测试模式

ADC 模块的测试模式是用户可用的测试模式,它允许在 ADC 模块的数字部分内执行回送操作。这在调试软件中非常有用,无须提供真实的模拟激励信号。

10. 内部温度传感器

内部温度传感器提供了模拟温度读取操作和参考电压。输出终端 SENSO 的电压通过以下等式计算得到:

$$SENSO = 2.7 - (T + 55)/75$$

这种关系如图 5-5 所示。

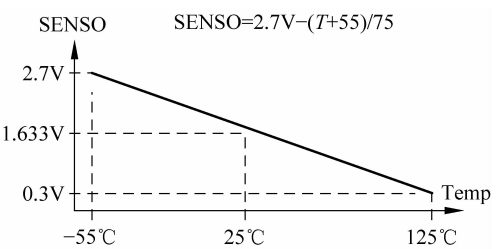

图 5-5　ADC 温度传感器温度-电压关系

下面我们来推导一个实用的 ADC 温度转换公式。假设温度电压 SENSO 对应的 ADC 采样值为 N,2.7V 对应 $N1$,$(T+55)/75$ 对应 $N2$。

已知:

$$N1 \times (3/1024) = 2.7$$

$$N2 \times (3/1024) = (T + 55)/75$$

由此得到:

$$N = N1 - N2 = 2.7/(3/1024) - ((T + 55)/75)/(3/1024)$$

解得:

$$T = (151040 - 225 \times N)/1024$$

结论:ADC 配置为温度传感器模式后,只要得到 10 位采样值 N,就能推算出摄氏温度 T。

5.3　ADC 应用注意事项

在实际应用当中,为了更好地发挥 Stellaris 系列的 10 位 ADC 特性,我们建议用户在设计时注意以下几个要点:

- 供电稳定可靠。Stellaris 系列的 ADC 参考电压是内部的 3.0V,该参考电压的上一

级来源是 VDDA，因此 VDDA 的供电必须要稳定可靠。建议 VDDA 精度要达到 1%。此外，建议 VDD 的供电也要尽可能稳定，以减少对 VDDA 的串扰。

- 模拟电源与数字电源分离。Stellaris 系列芯片都提供有数字电源 VDD/GND 和模拟电源 VDDA/GNDA，在设计时建议采用两路不同的 3.3V 电源稳压器分别进行供电。如果为了节省成本，也可以采用单路 3.3V 电源，但 VDDA/GNDA 要通过电感从 VDD/GND 分离出来。一般 GND 和 GNDA 最终还是要连接在一起的，建议用一个绕线电感连接并且接点尽可能靠近芯片（电感最好放在 PCB 背面），如图 5-6 所示。

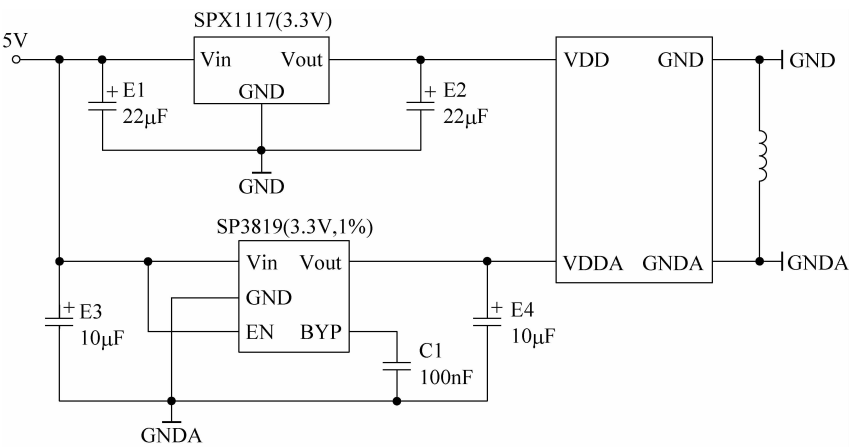

图 5-6　ADC 模拟电源与数字电源分离

采用多层 PCB 布局在成本允许的情况下，最好采用 4 层以上的 PCB 板，这能够带来更加优秀的 EMC 特性，减小对 ADC 采样的串扰，结果更加精确。

- 钳位二极管保护。一个典型的应用：测量电网 AC 220V 变化情况，电网电压经变压器降压到 3V 以符合 ADC 输入不能超过 3V 的要求，然后直接送到 ADC 输入引脚。如果真的这样用了，那么保证你的产品将来会大批坏掉！因为电网是存在波动的，瞬间电压可能大大超过额定的 220V，因此经变压器之后的电压可能远超 3V，自然有可能损坏芯片。正确的做法是必须要有限压保护措施，典型的用法是钳位保护二极管，能够把输入电压限制在 GND－VD2 到 VDD＋VD1 之间，如图 5-7 所示。
- 低通或带通滤波。为了抑制串入 ADC 输入信号上的干扰，一般要进行低通或带通滤波。RC 滤波是最常见也是成本最低的一种选择，并且电阻 R 还起到限流作用，如图 5-7 所示。
- 差分输入信号密近平行布线。如果是 ADC 的差分采样应用，则这一对输入的差分信号在 PCB 板上应当安排成密近的平行线，如果在不同线路板之间传递差分信号则应当采用屏蔽的双绞线。密近的布线会使来自外部的干扰同时作用于两根信号线上，这只会形成共模干扰，而

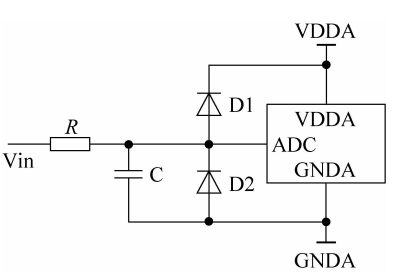

图 5-7　ADC 输入通道低通滤波与钳位保护

最终检测的是两根信号之间的差值，对共模信号不敏感。

- 差分模式也不能支持负的共模电压。Stellaris 系列的 ADC 支持差分采样，采样结果仅决定于两个输入端之间的电压差值。但是输入到每个输入端的共模电压（相对于 GNDA 的电压值）还是不能超过 0～3V 的额定范围。如果超过太多，有可能造成芯片损坏（参见图 5-7 的保护措施）。ADC 工作时钟必须在 16MHz 左右。Stellaris 系列 ADC 模块的内在特性要求工作时钟必须在 16MHz 左右，否则会带来较大的误差甚至是错误的转换结果。
- ADC 模块的时钟在 16MHz 左右。有两种方法可以保证提供。第一种方法是直接提供 16MHz 的外部时钟，可以从 OSC0 输入而 OSC1 悬空。对于 2008 年新推出的 DustDevil 家族能够直接支持 16MHz 的晶振。第二种方法是启用 PLL 单元，根据内部时钟树的结构（详见“系统控制（SysCtl）”这一部分内容），不论由 PLL 分频获得的主时钟频率是多少，提供给 ADC 模块的时钟总能够“自动地”保证在 16MHz 左右。

5.4　ADC 库函数

5.4.1　ADC 采样序列操作

- 函数 ADCSequenceEnable()和 ADCSequenceDisable()用来使能和禁止一个 ADC 采样序列。
- 函数 ADCSequenceConfigure()和 ADCSequenceStepConfigure()是两个至关重要的 ADC 配置函数，决定了 ADC 的全部功能。
- 函数 ADCSequenceDataGet()用来读取 ADC 结果 FIFO 里的数据。
- 函数 ADCSequenceOverflow()和 ADCSequenceOverflowClear()用于处理 ADC 结果 FIFO 出现上溢的情况。
- 函数 ADCSequenceUnderflow()和 ADCSequenceUnderflowClear()用于处理 ADC 结果 FIFO 出现下溢的情况。

表 5-3 所示为 ADC 采样序列的操作函数。

表 5-3　ADC 采样序列操作函数

函 数 名 称	参　　数	功能描述
void ADCSequenceConfigure (unsigned long ulBase, unsigned long ulSequenceNum, unsigned long ulTrigger, unsigned long ulPriority)	ulBase：ADC 模块的基址，取值 ADC_BASE。 ulSequenceNum：ADC 采样序列的编号，取值 0、1、2、3。 ulTrigger：启动采样序列的触发源。 ulPriority：相对于其他采样序列的优先级，取值 0、1、2、3（优先级依次从高到低）	配置 ADC 采样序列的触发事件和优先级原型

续表

函数名称	参　数	功能描述
void ADCSequenceStepConfigure (unsigned long ulBase, unsigned long ulSequenceNum, unsigned long ulStep, unsigned long ulConfig)	ulBase：ADC 模块的基址，取值 ADC_BASE。 ulSequenceNum：ADC 采样序列的编号，取值 0、1、2、3。 ulStep：步值，决定触发产生时 ADC 捕获序列的次序，对于不同的采样序列取值也不相同。 ulConfig：步进的配置	配置 ADC 采样序列发生器的步进原型
void ADCSequenceEnable(unsigned long ulBase, unsigned long ulSequenceNum)	ulBase：ADC 模块的基址，取值 ADC_BASE。 ulSequenceNum：ADC 采样序列的编号，取值 0、1、2、3	使能一个 ADC 采样序列
void ADCSequenceDisable(unsigned long ulBase, unsigned long ulSequenceNum)	ulBase：ADC 模块的基址，取值 ADC_BASE。 ulSequenceNum：ADC 采样序列的编号，取值 0、1、2、3	禁止一个 ADC 采样序列
long ADCSequenceDataGet (unsigned long ulBase, unsigned long ulSequenceNum, unsigned long * pulBuffer)	ulBase：ADC 模块的基址，取值 ADC_BASE。 ulSequenceNum：ADC 采样序列的编号，取值 0、1、2、3。 pulBuffer：无符号长整型指针，指向保存数据的缓冲区	从 ADC 采样序列里获取捕获到的数据原型。 返回：复制到缓冲区的采样数
long ADCSequenceOverflow (unsigned long ulBase, unsigned long ulSequenceNum)	ulBase：ADC 模块的基址，取值 ADC_BASE。 ulSequenceNum：ADC 采样序列的编号，取值 0、1、2、3	确定 ADC 采样序列是否发生了上溢。 返回：溢出返回 0，未溢出返回非 0。 正常操作不会产生上溢，但是如果在下次触发采样前没有及时从 FIFO 里读取捕获的采样值时则可能会发生上溢
long ADCSequenceOverflowClear (unsigned long ulBase, unsigned long ulSequenceNum)	ulBase：ADC 模块的基址，取值 ADC_BASE。 ulSequenceNum：ADC 采样序列的编号，取值 0、1、2、3	清除 ADC 采样序列的上溢条件
long ADCSequenceUnderflow (unsigned long ulBase, unsigned long ulSequenceNum)	ulBase：ADC 模块的基址，取值 ADC_BASE。 ulSequenceNum：ADC 采样序列的编号，取值 0、1、2、3	确定 ADC 采样序列是否发生了下溢。 返回：溢出返回 0，未溢出返回非 0。 正常操作不会产生下溢，但是如果过多地读取在 FIFO 里的采样值时则会发生下溢
void ADCSequenceUnderflowClear (unsigned long ulBase, unsigned long ulSequenceNum)	ulBase：ADC 模块的基址，取值 ADC_BASE。 ulSequenceNum：ADC 采样序列的编号，取值 0、1、2、3	清除 ADC 采样序列的下溢条件

表 5-4 描述了函数 ADCSequenceConfigure()。

表 5-4　函数 ADCSequenceConfigure()

函数名称	函数 ADCSequenceConfigure()
功能	配置 ADC 采样序列的触发事件和优先级原型
原型	void ADCSequenceConfigure(unsigned long ulBase, unsigned long ulSequenceNum, unsigned long ulTrigger, unsigned long ulPriority)
参数	ulBase：ADC 模块的基址，取值 ADC_BASE。 ulSequenceNum：ADC 采样序列的编号，取值 0、1、2、3。 ulTrigger：启动采样序列的触发源，取下列值之一。 ADC_TRIGGER_PROCESSOR　　//处理器事件 ADC_TRIGGER_COMP0　　//模拟比较器 0 事件 ADC_TRIGGER_COMP1　　//模拟比较器 1 事件 ADC_TRIGGER_COMP2　　//模拟比较器 2 事件 ADC_TRIGGER_EXTERNAL　　//外部事件(PB4 中断) ADC_TRIGGER_TIMER　　//定时器事件 ADC_TRIGGER_PWM0　　//PWM0 事件 ADC_TRIGGER_PWM1　　//PWM1 事件 ADC_TRIGGER_PWM2　　//PWM2 事件 ADC_TRIGGER_ALWAYS　　//触发一直有效(用于连续采样) ulPriority：相对于其他采样序列的优先级，取值 0、1、2、3(优先级依次从高到低)
返回	无

函数使用示例如下。

(1) ADC 采样序列配置：ADC 基址，采样序列 0，处理器触发，优先级 0。

```
ADCSequenceConfigure(ADC_BASE, 0, ADC_TRIGGER_PROCESSOR, 0);
```

(2) ADC 采样序列配置：ADC 基址，采样序列 1，定时器触发，优先级 2。

```
ADCSequenceConfigure(ADC_BASE, 1, ADC_TRIGGER_TIMER, 2);
```

(3) ADC 采样序列配置：ADC 基址，采样序列 2，外部事件(PB4 中断)触发，优先级 3。

```
ADCSequenceConfigure(ADC_BASE, 2, ADC_TRIGGER_EXTERNAL, 3);
```

(4) ADC 采样序列配置：ADC 基址，采样序列 3，模拟比较器 0 事件触发，优先级 1。

```
ADCSequenceConfigure(ADC_BASE, 3, ADC_TRIGGER_COMP0, 1);
```

注意：①并非所有 Stellaris 系列的成员都可以使用上述全部的触发源。请查询相关器件的数据手册来确定它们的可用触发源。②在对一系列的采样序列的优先级进行编程时，每个采样序列的优先级必须是唯一的；由调用者来确保优先级的唯一性。

表 5-5 描述了函数 ADCSequenceStepConfigure()。

表 5-5　函数 ADCSequenceStepConfigure()

<table>
<tr><th>函数名称</th><th>ADCSequenceStepConfigure()</th></tr>
<tr><td>功能</td><td>配置 ADC 采样序列发生器的步进原型</td></tr>
<tr><td>原型</td><td>void ADCSequenceStepConfigure(unsigned long ulBase,
unsigned long ulSequenceNum,
unsigned long ulStep,
unsigned long ulConfig)</td></tr>
<tr><td>参数</td><td>ulBase：ADC 模块的基址，取值 ADC_BASE。
ulSequenceNum：ADC 采样序列的编号，取值 0、1、2、3。
ulStep：步值，决定触发产生时 ADC 捕获序列的次序，对于不同的采样序列取值也不相同。
<table>
<tr><th>采样序列编号</th><th>步值范围</th></tr>
<tr><td>0</td><td>0～7</td></tr>
<tr><td>1</td><td>0～3</td></tr>
<tr><td>2</td><td>0～3</td></tr>
<tr><td>3</td><td>0</td></tr>
</table>
ulConfig：步进的配置，取下列值之间的“或运算”组合形式。
• ADC 控制：
ADC_CTL_TS　　//温度传感器选择
ADC_CTL_IE　　//中断使能
ADC_CTL_END　　//队列结束选择
ADC_CTL_D　　//差分选择
• ADC 通道：
ADC_CTL_CH0　　//输入通道 0(对应 ADC0 输入)
ADC_CTL_CH1　　//输入通道 1(对应 ADC1 输入)
ADC_CTL_CH2　　//输入通道 2(对应 ADC2 输入)
ADC_CTL_CH3　　//输入通道 3(对应 ADC3 输入)
ADC_CTL_CH4　　//输入通道 4(对应 ADC4 输入)
ADC_CTL_CH5　　//输入通道 5(对应 ADC5 输入)
ADC_CTL_CH6　　//输入通道 6(对应 ADC6 输入)
ADC_CTL_CH7　　//输入通道 7(对应 ADC7 输入)
注意：ADC 通道每次(即每步)最多只能选择 1 个，如果要选取多通道则要多次调用本函数分别进行配置；如果已经选择了内置的温度传感器(ADC_CTL_TS)则不能再选择 ADC 通道；如果已选择了差分采样模式(ADC_CTL_D)，则 ADC 通道只能选取下列值之一。
ADC_CTL_CH0　　//差分输入通道 0(对应 ADC0 和 ADC1 输入的组合)
ADC_CTL_CH1　　//差分输入通道 1(对应 ADC2 和 ADC3 输入的组合)
ADC_CTL_CH2　　//差分输入通道 2(对应 ADC4 和 ADC5 输入的组合)
ADC_CTL_CH3　　//差分输入通道 3(对应 ADC6 和 ADC7 输入的组合)</td></tr>
<tr><td>返回</td><td>无</td></tr>
</table>

函数使用示例如下。

(1) ADC 采样序列步进配置：ADC 基址，采样序列 2，步值 0，采样 ADC0 输入后结束并申请中断。

```
ADCSequenceStepConfigure(ADC_BASE, 2, 0, ADC_CTL_CH0 | ADC_CTL_END | ADC_CTL_IE);
```

(2) ADC 采样序列步进配置：ADC 基址，采样序列 3，步值 0，采样温度传感器后结束并申请中断。

```
ADCSequenceStepConfigure(ADC_BASE, 3, 0, ADC_CTL_TS | ADC_CTL_END | ADC_CTL_
IE);
```

(3) ADC 采样序列步进配置：ADC 基址，采样序列 0，步值 0，采样 ADC0 输入。

```
ADCSequenceStepConfigure(ADC_BASE, 0, 0, ADC_CTL_CH0);
```

(4) ADC 采样序列步进配置：ADC 基址，采样序列 0，步值 1，采样 ADC1 输入。

```
ADCSequenceStepConfigure(ADC_BASE, 0, 1, ADC_CTL_CH1);
```

(5) ADC 采样序列步进配置：ADC 基址，采样序列 0，步值 2，再次采样 ADC0 输入。

```
ADCSequenceStepConfigure(ADC_BASE, 0, 2, ADC_CTL_CH0);
```

(6) ADC 采样序列步进配置：ADC 基址，采样序列 0，步值 3，采样 ADC3 输入后结束并申请中断。

```
ADCSequenceStepConfigure(ADC_BASE, 0, 3, ADC_CTL_CH3 | ADC_CTL_END | ADC_CTL_
IE);
```

(7) ADC 采样序列步进配置：ADC 基址，采样序列 1，步值 0，差分采样 ADC0/ADC1 输入。

```
ADCSequenceStepConfigure(ADC_BASE, 1, 0, ADC_CTL_D | ADC_CTL_CH0);
```

(8) ADC 采样序列步进配置：ADC 基址，采样序列 1，步值 1，差分采样 ADC2/ADC3 输入后结束并申请中断。

```
ADCSequenceStepConfigure(ADC_BASE, 1, 1, ADC_CTL_D | ADC_CTL_CH1 |
                         ADC_CTL_END | ADC_CTL_IE);
```

5.4.2　ADC 处理器触发

ADC 采样触发方式有许多种选择，其中处理器(软件)触发是最简单的一种情况。在配置好 ADC 模块以后，只要调用函数 ADCProcessorTrigger()就能够引起一次 ADC 采样。参见表 5-6 的描述。

表 5-6　函数 ADCProcessorTrigger()

函数名称	ADCProcessorTrigger()
功能	引起一次处理器触发 ADC 采样
原型	void ADCProcessorTrigger(unsigned long ulBase, unsigned long ulSequenceNum)
参数	ulBase：ADC 模块的基址，取值 ADC_BASE。 ulSequenceNum：ADC 采样序列的编号，取值 0、1、2、3
返回	无

5.4.3　ADC 过采样

ADC 过采样的实质是以牺牲采样速度来换取采样精度。硬件上的自动求平均值电路

能够对多达连续 64 次的采样做出平均计算，有效消除采样结果的不均匀性。对硬件过采样的配置很简单，就是调用函数 ADCHardwareOversampleConfigure()。

在《Stellaris 外设驱动库》里，还额外提供了简易的软件过采样的库函数，能够对多至 8 个采样求取平均值。用户也可以参考其源代码做出更优秀的改进。有关软件过采样的函数请参考表 5-7 的描述。

表 5-7　ADC 硬件/软件过采样相关函数

函 数 名 称	参　　数	功能描述
void ADCHardwareOversampleConfigure (unsigned long ulBase, unsigned long ulFactor)	ulBase：ADC 模块的基址，取值 ADC_BASE。 ulFactor：采样平均数，取值 2、4、8、16、32、64，如果取值 0 则禁止硬件过采样	配置 ADC 硬件过采样的因数
void ADCSoftwareOversampleConfigure (unsigned long ulBase, unsigned long ulSequenceNum, unsigned long ulFactor)	ulBase：ADC 模块的基址，取值 ADC_BASE。 ulSequenceNum：ADC 采样序列的编号，取值 0、1、2(采样序列 3 不支持软件过采样)。 ulFactor：采样平均数，取值 2、4、8。 参数 ulFactor 和 ulSequenceNum 的取值是关联的。在 4 个采样序列当中，只有深度大于 1 的采样序列才支持过采样，因此 ulSequenceNum 不能取值 3。当 ulFactor 取值 2、4 时，ulSequenceNum 可以取值 0、1、2；当 ulFactor 取值 8 时，ulSequenceNum 只能取值 0	配置 ADC 软件过采样的因数原型
void ADCSoftwareOversampleStepConfigure (unsigned long ulBase,unsigned long ulSequenceNum,unsigned long ulStep, unsigned long ulConfig)	ulBase：ADC 模块的基址，取值 ADC_BASE。 ulSequenceNum：ADC 采样序列的编号，取值 0、1、2(采样序列 3 不支持软件过采样)。 ulStep：步值，决定触发产生时 ADC 捕获序列的次序。 ulConfig：步进的配置，取值跟表 5-4 当中的参数 ulConfig 相同	ADC 软件过采样步进配置原型
void ADCSoftwareOversampleDataGet (unsigned long ulBase, unsigned long ulSequenceNum, unsigned long * pulBuffer, unsigned long ulCount)	ulBase：ADC 模块的基址，取值 ADC_BASE。 ulSequenceNum：ADC 采样序列的编号，取值 0、1、2(采样序列 3 不支持软件过采样)。 pulBuffer：长整型指针，指向保存数据的缓冲区。 ulCount：要读取的采样数	从采用软件过采样的一个采样序列获取捕获的数据原型

5.4.4　ADC 中断控制

4 个采样序列 SS0、SS1、SS2、SS3 的中断控制是独立进行的，在中断向量表里分别独享 1 个向量号。

函数 ADCIntEnable()和 ADCIntDisable()用来使能和禁止 ADC 采样序列中断。而函数 ADCIntClear()用来清除其中断状态。

函数 ADCIntRegister()和 ADCIntUnregister()用来注册和注销 ADC 采样序列中断。

表 5-8 描述了 ADC 中断控制的相关函数。

表 5-8　ADC 中断控制的相关函数

函 数 名 称	参　　数	功能描述
void ADCIntEnable(unsigned long ulBase, unsigned long ulSequenceNum)	ulBase：ADC 模块的基址，取值 ADC_BASE	使能 ADC 采样序列的中断
void ADCIntDisable(unsigned long ulBase, unsigned long ulSequenceNum)	ulBase：ADC 模块的基址，取值 ADC_BASE。 ulSequenceNum：ADC 采样序列的编号，取值 0、1、2、3	禁止 ADC 采样序列的中断
unsigned long ADCIntStatus(unsigned long ulBase, unsigned long ulSequenceNum, tBoolean bMasked)	ulBase：ADC 模块的基址，取值 ADC_BASE。 ulSequenceNum：ADC 采样序列的编号，取值 0、1、2、3。 bMasked：如果需要获取原始的中断状态，则取值 false；如果需要获取屏蔽的中断状态，则取值 true	获取 ADC 采样序列的中断状态
void ADCIntClear(unsigned long ulBase, unsigned long ulSequenceNum)	ulBase：ADC 模块的基址，取值 ADC_BASE。 ulSequenceNum：ADC 采样序列的编号，取值 0、1、2、3	清除 ADC 采样序列的中断状态
void ADCIntRegister(unsigned long ulBase, unsigned long ulSequenceNum, void (*pfnHandler)(void))	ulBase：ADC 模块的基址，取值 ADC_BASE。 ulSequenceNum：ADC 采样序列的编号，取值 0、1、2、3。 pfnHandler：函数指针，指向 ADC 中断服务函数	注册一个 ADC 采样序列的中断服务函数
void ADCIntUnregister(unsigned long ulBase, unsigned long ulSequenceNum)	ulBase：ADC 模块的基址，取值 ADC_BASE。 ulSequenceNum：ADC 采样序列的编号，取值 0、1、2、3	注销 ADC 采样序列的中断服务函数

编程范例：下面的示例显示了如何使用 ADC 的 API 来初始化一个处理器触发的采样序列，触发，然后在数据准备就绪后读回数据。

```
unsigned long ulValue;
//当处理器触发出现时，使能第一个采样序列来捕获通道 0 的值
ADCSequenceConfigure(ADC_BASE, 0, ADC_TRIGGER_PROCESSOR, 0);
ADCSequenceStepConfigure(ADC_BASE, 0, 0, ADC_CTL_IE | ADC_CTL_END | ADC_CTL_CH0);
ADCSequenceEnable(ADC_BASE, 0);

//触发采样序列
ADCProcessorTrigger(ADC_BASE, 0);

//等待采样序列完成
while(!ADCIntStatus(ADC_BASE, 0, false))
{
}

//从 ADC 读取值
ADCSequenceDataGet(ADC_BASE, 0, &ulValue);
```

5.5　ADC 模块的应用

5.5.1　ADC 模块初始化

1. 模块初始化配置

ADC 模块的初始化过程很简单，只需几个步骤。主要的步骤包括使能 ADC 时钟和重新配置采样序列发生器的优先级(如有必要)。ADC 初始化的顺序如下：

(1) 通过写 0x0001.0000 的值到 RCGC1 寄存器来使能 ADC 时钟。

(2) 如果应用有要求，那么在 ADCSSPRI 寄存器中重新配置采样序列发生器的优先级。默认配置是采样序列发生器 0 优先级最高，采样序列发生器 3 优先级最低。

2. 采样序列发生器的配置

采样序列发生器的配置要稍微比模块初始化的过程复杂，因为每个采样序列是完全可编程的。每个采样序列发生器的配置如下：

(1) 确保采样序列发生器被禁能，这可以通过写 0 到 ADCACTSS 寄存器中对应的 ASEN 位来实现。采样序列发生器无须使能就可编程。如果在配置过程中发生触发事件，那么在编程期间禁能序列发生器就可以预防错误的执行操作。

(2) 在 ADCEMUX 寄存器中为采样序列发生器配置触发事件。

(3) 在 ADCSSMUXn 寄存器中为采样序列的每个采样配置相应的输入源。

(4) 在 ADCSSCTLn 寄存器中为采样序列的每个采样配置采样控制位。当对最后半个字节进行编程时，确保 END 位已置位。END 位置位失败会导致不可预测的行为。

(5) 如果要使用中断，那么必须写 1 到 ADCIM 寄存器中相应的 MASK 位。

(6) 通过写 1 到 ADCACTSS 寄存器中相应的 ASEN 位来使能采样序列发生器逻辑。

3. 初始化配置例程

```
//设置系统时钟
SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |SYSCTL_
XTAL_8MHz);
//使能 ADC 时钟及 GPIOF
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC);
//设置 GPIOF0 为输出
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_0);

//配置 ADC,通道 1,基准源为控制器产生
ADCSequenceConfigure(ADC_BASE,0,ADC_TRIGGER_PROCESSOR,0);
ADCSequenceStepConfigure(ADC_BASE,0,0,ADC_CTL_IE|ADC_CTL_END|ADC_CTL_CH1);
//ADC 采样使能
ADCSequenceEnable(ADC_BASE,0);
```

5.5.2　ADC 开始采样

```
//开始采样
```

```
ADCProcessorTrigger(ADC_BASE,0);
//等待采样完成
while(!ADCIntStatus(ADC_BASE,0,false)) ;
//读 ADC 寄存器值
ADCSequenceDataGet(ADC_BASE,0,&ulValue);
//判断 ADC 寄存器值是否大于 500(0～1023)
if(ulValue>500)
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, 1);      //LED 灭
    }
else
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, 0);      //LED 亮
    }
```

5.6　ADC 实例分析

5.6.1　处理器触发 ADC 采样实例

目标：采用处理器触发 ADC 采样，完成外部电压采集。

方法：在配置好 ADC 模块之后，就可以随时调用 ADCProcessorTrigger()函数来触发 ADC 采样。

现象：采样结果通过 UART 输出，电压单位是 mV。当 RW0 旋钮停留在某个位置以后，ADC 采样结果的变动一般不会超过±3mV，即±1 个 LSB(精确值是 3V/1024＝2.93mV)。

Stellaris 系列 ADC 模块的采样触发方式有多种，应用非常灵活。其中处理器(软件)是最简单的一种采样触发方式，关键是调用函数 ADCProcessorTrigger()。

图 5-8 所示为 ADC 输入测试电路示意图。Stellaris 系列 MCU 的 ADC 模块采用模拟电源 VDDA/GNDA 供电。RW0 和 RW1(即 *R*20 和 *R*79)是电位器，输出电压在 0～3.3V 之间，并带有手动旋钮，便于操作。*R*77 和 C57 组成简单的 RC 低通滤波电路，能够滤除寄生在由 RW1 产生的在模拟信号上的扰动。以后的每个 ADC 例程都会采用类似的测试电路。

程序清单 5-1 是处理器触发 ADC 采样的例程，注意在函数 adcInit()里对 ADC 模块的基本配置方法。其中对采样速率的设置不应超过实际芯片的最高速率限制。在配置好 ADC 模块之后，就可以随时调用 ADCProcessorTrigger()函数触发 ADC 采样了。在该例程里，采样结果通过 UART 输出，电压单位是 mV。只要电路设计没有问题，当 RW0 旋钮停留在某个位置以后，ADC 采样结果的变动一般不会超过±3mV。

1. ADC *初始化函数*

首先使能 ADC 模块，设置 ADC 采样速率(采样速率为 125ks/s)。采样序列配置：ADC 基址、采样序列编号、触发事件、采样优先级。采样步进设置：ADC 基址、采样序列编号、步值、通道设置。使能 ADC 中断、使能 ADC 采样序列中断、使能处理器中断，如程序清单 5-1 所示。

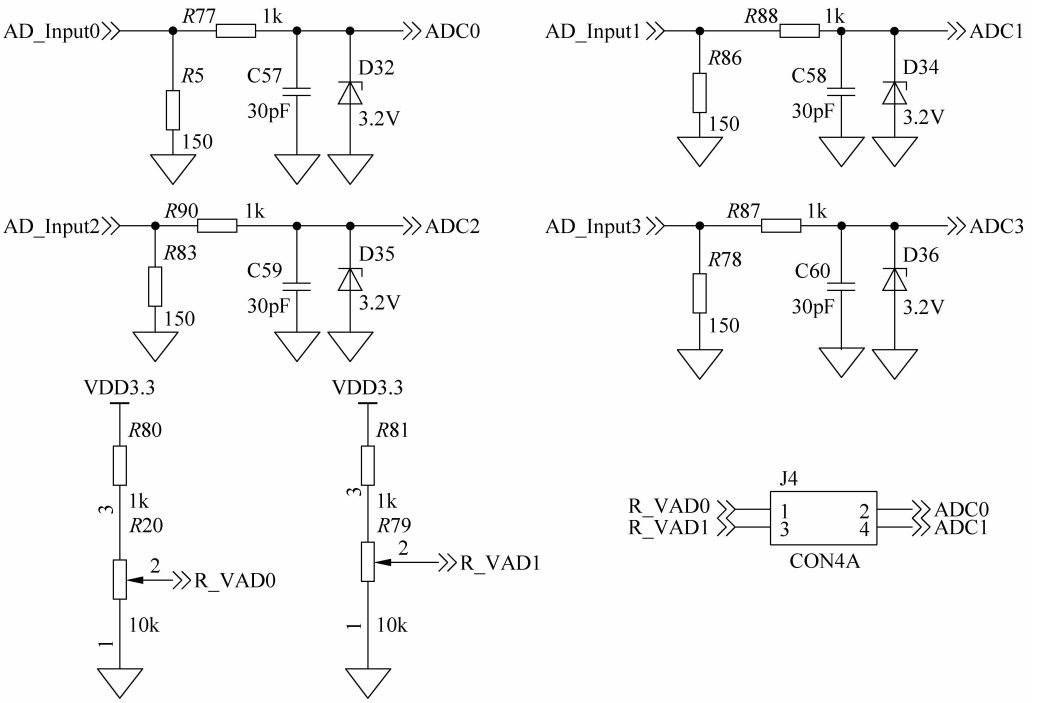

图 5-8　ADC 输入测试电路

程序清单 5-1　ADC 初始化配置部分函数

```
SysCtlPeriEnable(SYSCTL_PERIPH_ADC);                    //使能 ADC 模块
SysCtlADCSpeedSet(SYSCTL_ADCSPEED_125KSPS);             //设置 ADC 采样速率
ADCSequDisable(ADC_BASE, 0);                            //配置前先禁止采样序列
//采样序列配置: ADC 基址、采样序列编号、触发事件、采样优先级
ADCSequConfig(ADC_BASE, 0, ADC_TRIGGER_PROCESSOR, 0);
//采样步进设置: ADC 基址、采样序列编号、步值、通道设置
ADCSequStepConfig(ADC_BASE, 0, 0, ADC_CTL_CH0 |
                  ADC_CTL_END |
                  ADC_CTL_IE);

ADCIntEnable(ADC_BASE, 0);                              //使能 ADC 中断
IntEnable(INT_ADC0);                                    //使能 ADC 采样序列中断
IntMasterEnable();                                      //使能处理器中断
ADCSequEnable(ADC_BASE, 0);                             //使能采样序列
```

2. ADC 采样子函数

ADC 采样主要调用处理器触发采样序列，即调用函数 ADCProcessorTrigger()进行 ADC 触发。使用函数 ADCSequDataGet()获取数据，再判断中断中产生的 ADC 采样结束标志 ADC_EndFlag，若为 1，则表明 ADC 转换完成，使用函数 ADCSequDataGet()读取 ADC 转换数据，如程序清单 5-2 所示。请读者仔细理解“触发采样”和“采样完成”两个概念以及各自的动作。

程序清单 5-2　ADC 采样子函数 adcSample()

```
unsigned long adcSample(void)
{
```

```
    unsigned long ulValue;
    ADCProcessorTrigger(ADC_BASE, 0);                    //处理器触发采样序列
    while (!ADC_EndFlag);                                //等待采样结束
    ADC_EndFlag = false;                                 //清除 ADC 采样结束标志
    ADCSequDataGet(ADC_BASE, 0, &ulValue);               //读取 ADC 转换结果
    return(ulValue);
}
```

3. ADC 采样序列 0 的中断子函数

ADC 采样序列 0 的中断，首先读取中断状态，然后清除中断状态，判断中断状态是否有效，若有效将置位 ADC 采样结束标志 ADC_EndFlag，从而将转换标志 ADC_EndFlag 用于 ADC 采样子函数中判断 ADC 采样结束的标志，读取 ADC 转换结果（因此注意 ADC_EndFlag 必须声明为全局变量），如程序清单 5-3 所示。

程序清单 5-3　ADC 采样序列 0 的中断函数 ADC_Sequence_0_ISR()

```
void ADC_Sequence_0_ISR(void)
{
    unsigned long ulStatus;
    ulStatus = ADCIntStatus(ADC_BASE, 0, true);          //读取中断状态
    ADCIntClear(ADC_BASE, 0);                            //清除中断状态,重要
    if (ulStatus != 0)                                   //如果中断状态有效
    {
        ADC_EndFlag = true;                              //置位 ADC 采样结束标志
    }
}
```

代码分析：从主函数中可以看出，首先时钟初始化、串口输出初始化、ADC 采样初始化，在循环函数中使用 adcSample()函数实现 ADC 采样，此函数通过判断是否转换完的中断标志，然后通过 ADCSequDataGet()进行数据读取。注意 ADC_Sequence_0_ISR()中断函数，这个函数必须在工程的起始代码程序中定义声明。初始化中必须配置的两个函数为采样配置函数 ADCSequConfig()和采样步进设置函数 ADCSequStepConfig()，剩下的就是为中断开启的函数了。采样配置函数的触发事件定义 ADC_TRIGGER_PROCESSOR。

注：完整的运行工程请参阅本书附光盘文件夹“第 5 章 ADC\ADC_cpu_trigger”下的工程。

5.6.2　ADC 内置的温度传感器实例

在 Stellaris 系列的 ADC 模块里，附带了一个内置的温度传感器，能够随时检测芯片的温度。该温度传感器可以有以下用途。

- 测试用：在单独测试 ADC 模块的功能时使用，而不必提供外部的模拟信号源。
- 测量芯片自身温度，防止可能出现的过温（高温应用场合必备）。
- 估算环境温度：芯片温度总是比环境温度略高，如果通过实验找到这个差值，则可以进行软件修正。
- 随机算法里可以提供随机数种子。

目标：利用 ADC 内置温度传感器测量处理器内部的温度。

方法：利用 ADC 内部的温度传感器(因此不需要提供外部模拟信号到 ADC 引脚)，将一路 ADC 转换设置为内部温度传感器，使用摄氏温度与 ADC 采样结果 N 之间的换算关系，获得最终处理器内部的摄氏温度。

现象：处理器内部的温度值通过 UART 端口输出显示。

在程序当中，还用到了睡眠模式(Sleep-Mode)，尽量降低功耗，使芯片温度更接近于环境温度(如果读者感兴趣可以在软件上做进一步的修正)。

1. *系统节拍定时器初始化子函数*

这主要为使用睡眠模式提供定时器，从而尽量降低功耗，使芯片温度更接近于环境温度，如程序清单为 5-4 所示。

程序清单 5-4　系统节拍定时器初始化函数 SysTickInit()

```
void SysTickInit(void)
{
    SysTickPeriodSet(TheSysClock);          //设置 SysTick 计数器的周期值
    SysTickIntEnable();                     //使能 SysTick 中断
    IntMasterEnable();                      //使能处理器中断
    SysTickEnable();                        //使能 SysTick 计数器
}
```

2. ADC *初始化子函数*

首先使能 ADC 模块，设置 ADC 采样速率(采样速率为 125ks/s)。采样序列配置：ADC 基址、采样序列编号、触发事件、采样优先级。采样步进设置：ADC 基址、采样序列编号、步值，通道设置为 ADC_CTL_TS，即表示内部温度传感器。使能 ADC 中断、使能 ADC 采样序列中断、使能处理器中断。本程序使用的采样序列编号为 3(注意：采样序列 3 的 FIFO 值为 1)，如程序清单 5-5 所示。

程序清单 5-5　ADC 初始化函数 adcInit()

```
void adcInit(void)
{
    SysCtlPeriEnable(SYSCTL_PERIPH_ADC);                //使能 ADC 模块
    SysCtlADCSpeedSet(SYSCTL_ADCSPEED_125KSPS);         //设置 ADC 采样速率
    ADCSequDisable(ADC_BASE, 3);                        //配置前先禁止采样序列

                          //采样序列配置：ADC 基址、采样序列编号、触发事件、采样优先级
    ADCSequConfig(ADC_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);
                          //采样步进设置：ADC 基址、采样序列编号、步值、通道设置
    ADCSequStepConfig(ADC_BASE, 3, 0, ADC_CTL_TS |ADC_CTL_END |ADC_CTL_IE);

    ADCIntEnable(ADC_BASE, 3);                          //使能 ADC 中断
    IntEnable(INT_ADC3);                                //使能 ADC 采样序列中断
    IntMasterEnable();                                  //使能处理器中断

    ADCSequEnable(ADC_BASE, 3);                         //使能采样序列
}
```

3. ADC *采样子函数*

ADC 采样主要调用处理器触发采样序列，即调用 ADCProcessorTrigger()进行触发

ADC。使用函数 ADCSequDataGet()获取数据，再判断中断中产生的 ADC 采样结束标志 ADC_EndFlag，若为 1，则表明 ADC 转换完成，使用函数 ADCSequDataGet()读取 ADC 转换数据，如程序清单 5-6 所示。

程序清单 5-6　ADC 采样子函数 adcSample()

```
unsigned long adcSample(void)
{
与普通的采样完全一致，代码省略
}
```

4. 显示芯片温度值子函数

温度显示值按照 ADC 温度转换公式计算，$T=(151\,040-225\times N)/1024$，其中温度电压 SENSO 对应的 ADC 采样值为 N，如程序清单 5-7 所示。

程序清单 5-7　ADC 显示芯片温度值函数 tmpDisplay()

```
void tmpDisplay(unsigned long ulValue)
{
    unsigned long ulTmp;
    char cBuf[40];
    ulTmp = 151040UL - 225 * ulValue;//计算公式如上
    sprintf(cBuf, "%ld.", ulTmp / 1024);
    uartPuts(cBuf);
    sprintf(cBuf, "%ld", (ulTmp % 1024) / 102);
    uartPuts(cBuf);
    uartPuts("℃\r\n");
}
```

5. ADC 采样序列 3 的中断

首先读取中断状态，然后清除中断状态，判断中断状态是否有效，若有效将置位 ADC 采样结束。用于在主函数中判断 ADC 采样结束标志，从而读取 ADC 转换结果，如程序清单 5-8 所示。

程序清单 5-8　ADC 采样序列 3 的中断函数 ADC_Sequence_3_ISR()

```
void ADC_Sequence_3_ISR(void)
{
代码与程序清单 5-3 基本一致，只需将 ADC 采样序列"0"改为对应的"3"即可
}
```

代码分析：从主函数中可以看出，首先时钟初始化、串口输出初始化、ADC 采样初始化、系统节拍定时器初始化。在循环函数中使用 adcSample()函数实现 ADC 采样，此函数通过判断是否转换完的中断标志，然后通过 ADCSequDataGet()进行数据读取。注意 ADC_Sequence_3_ISR()中断函数，这个函数必须在工程的起始代码程序中定义声明。初始化中必须配置的两个函数为采样配置函数 ADCSequConfig()和采样步进设置函数 ADCSequStepConfig()，剩下的就是为中断开启的函数了。系统节拍定时器初始化的作用是唤醒处理器。

注：完整的运行工程请参阅本书附光盘文件夹"第 5 章 ADC ADC_Temperature_Sensor"下的工程。

5.6.3　处理器触发多通道 ADC 采样实例

目标：处理器触发多通道 ADC 采样。

方法：通过 ADCSequStepConfig()函数设定 ADC 采样序列步进配置，包括 ADC 基址、采样序列 0、步值、采样通道，其中通道就是需要设置的多通道 ADC_CTL_CH0，ADC_CTL_CH1，ADC_CTL_CH2，ADC_CTL_CH3，从而完成多通道采样。

现象：多通道 ADC 采样值通过 UART 串行端口输出。

程序清单 5-9 给出了 ADC 多通道采样的用法。关键是 ADCSequStepConfig()这个函数的用法，每次只能对一个通道的采样进行配置，如果要对多个通道的采样进行配置，就要连续并列多个 ADCSequStepConfig()函数，并在配置最后一个采样时结束并触发中断。采样序列 SS0 可以支持 8 个采样，但具体哪一步对应哪个通道的采样可以由用户做自由安排。读取这一组采样结果时，仍然使用函数 ADCSequDataGet()，它能够一次性读取全部采样结果而不需要分多次调用，这就要求用于保存结果的缓冲区 ulVal[ ]定义的空间足够大，以避免溢出。

处理器触发多通道 ADC 采样的整个过程与处理器触发 ADC 采样基本一致，不同处在于多次通过 ADCSequStepConfig()函数设定 ADC 采样序列步进配置，包括 ADC 基址、采样序列 0、步值、采样通道，其中通道就是需要设置的多通道 ADC_CTL_CH0，ADC_CTL_CH1，ADC_CTL_CH2，ADC_CTL_CH3，如程序清单 5-9 所示。

程序清单 5-9　ADC 初始化函数 adcInit()

```
void adcInit(void)
{
    SysCtlPeriEnable(SYSCTL_PERIPH_ADC);                    //使能 ADC 模块
    SysCtlADCSpeedSet(SYSCTL_ADCSPEED_125KSPS);             //设置 ADC 采样速率
    ADCSequDisable(ADC_BASE, 0);                            //配置前先禁止采样序列
    //采样序列配置: ADC 基址、采样序列编号、触发事件、采样优先级
    ADCSequConfig(ADC_BASE, 0, ADC_TRIGGER_PROCESSOR, 0);
    //ADC 采样序列步进配置: ADC 基址、采样序列 0、步值、采样通道
    ADCSequStepConfig(ADC_BASE, 0, 0, ADC_CTL_CH0);         //第 0 步: 采样 ADC0
    ADCSequStepConfig(ADC_BASE, 0, 1, ADC_CTL_CH1);         //第 1 步: 采样 ADC1
    ADCSequStepConfig(ADC_BASE, 0, 2, ADC_CTL_CH2);         //第 2 步: 采样 ADC2
    ADCSequStepConfig(ADC_BASE, 0, 3, ADC_CTL_CH0);         //第 3 步: 再次采样 ADC0
    ADCSequStepConfig(ADC_BASE, 0, 4, ADC_CTL_CH3 |         //第 4 步: 采样 ADC3 后
                                      ADC_CTL_END |         //结束,并申请中断
                                      ADC_CTL_IE);

    ADCIntEnable(ADC_BASE, 0);                              //使能 ADC 中断
    IntEnable(INT_ADC0);                                    //使能 ADC 采样序列中断
    IntMasterEnable();                                      //使能处理器中断
    ADCSequEnable(ADC_BASE, 0);                             //使能采样序列
}
```

代码分析：从主函数中可以看出，首先时钟初始化、串口输出初始化、ADC 采样初始化，在循环函数中使用 adcSample()函数实现 ADC 采样，此函数通过判断是否转换完的中断标注，然后通过 ADCSequDataGet()进行数据读取。注意 ADC_Sequence_3_ISR()中断

函数，这个函数必须在工程的起始代码程序中定义声明。初始化中必须配置的两个函数为采样配置函数 ADCSequConfig()和采样步进设置函数 ADCSequStepConfig()，剩下的就是为中断开启的函数了。注意这里采用 4 通道进行采样，ADC 采样序列步进配置：ADC 基址、采样序列 0、步值。采样通道有讲究，在采样 ADC3 后结束，并申请中断。

注：完整的运行工程请参阅本书附光盘文件夹“第 5 章 ADC ADC_Multi-Channel_Sample”下的工程。

5.6.4　定时器溢出触发 ADC 采样实例

目标：定时器溢出触发 ADC 采样。

方法：通过设置定时器使能内部触发脉冲的产生，内部触发 ADC 采样。

现象：处理器内部的温度通过 UART 端口输出。

注意：Timer0～Timer2 在溢出时触发 ADC 采样的功能是正常的，但 Timer3 在 32 位定时器模式下可能不会触发 ADC 采样，用户在实际应用时要引起注意。

1. Timer 初始化

主要是使能 Timer 模块、配置 Timer 为 32 位周期定时、使能内部触发脉冲的产生、调试时暂停计数、设置定时器初值等操作。注意此处必须通过函数 TimerControlTrigger()使能内部触发脉冲的产生，如程序清单 5-10 所示。

程序清单 5-10　定时器初始化函数 timerInit()

```
void timerInit(void)
{
    SysCtlPeriEnable(SYSCTL_PERIPH_TIMER2);                    //使能 Timer 模块
    TimerConfigure(TIMER2_BASE, TIMER_CFG_32_BIT_PER); //配置 Timer 为 32 位周期定时
    TimerControlTrigger(TIMER2_BASE, TIMER_A, true);       //使能内部触发脉冲的产生
    TimerControlStall(TIMER2_BASE, TIMER_A, true);         //调试时暂停计数(必要)
    TimerLoadSet(TIMER2_BASE, TIMER_A, 20000000UL);        //设置 Timer 初值
    TimerEnable(TIMER2_BASE, TIMER_A);                     //使能 Timer 计数
}
```

2. ADC 初始化

对于定时器溢出触发 ADC 采样，主要在 ADCSequConfig()函数中设置 ADC_TRIGGER_TIMER 为定时器触发，如程序清单 5-11 所示。

程序清单 5-11　ADC 初始化 adcInit()

```
void adcInit(void)
{
    SysCtlPeriEnable(SYSCTL_PERIPH_ADC);                       //使能 ADC 模块
    SysCtlADCSpeedSet(SYSCTL_ADCSPEED_125ksps);            //设置 ADC 采样率
    ADCSequDisable(ADC_BASE, 0);                               //禁止采样序列
        //采样序列配置：ADC 基址、采样序列 0、定时器触发、采样优先级 0
    ADCSequConfig(ADC_BASE, 0, ADC_TRIGGER_TIMER, 0);
        //采样步进设置：ADC 基址、采样序列 0、步值 0、采样 ADC0 后停止，并申请中断
    ADCSequStepConfig(ADC_BASE, 0, 0, ADC_CTL_CH0 |
                                      ADC_CTL_END |
                                      ADC_CTL_IE);
```

```
    ADCIntEnable(ADC_BASE, 0);                    //使能 ADC 中断
    IntEnable(INT_ADC0);                          //使能 ADC 采样序列中断
    IntMasterEnable();                            //使能处理器中断
    ADCSequEnable(ADC_BASE, 0);                   //使能采样序列
}
```

代码分析：从主函数中可以看出，首先时钟初始化、串口输出初始化、定时器初始化、ADC 采样初始化。在循环函数中使用 adcSample()函数实现 ADC 采样，此函数通过判断是否转换完的中断标志，然后通过ADCSequDataGet()进行数据读取。注意 ADC_Sequence_0_ISR()中断函数，这个函数必须在工程的起始代码程序中定义声明。初始化中必须配置的两个函数为采样配置函数ADCSequConfig()和采样步进设置函数 ADCSequStepConfig()，剩下的就是为中断开启的函数了，采样配置函数的触发事件定义 ADC_TRIGGER_TIMER。

注：完整的运行工程请参阅本书附光盘文件夹“第 5 章 ADC ADC_Timer_Trigger”下的工程。

5.6.5　差分输入 ADC 采样实例

目标：差分输入 ADC 采样。

方法：通过设置定时器使能内部触发脉冲的产生，内部触发 ADC 采样。

现象：ADC 采样值通过 UART 端口输出。

能够支持差分输入采样是 Stellaris 系列 ADC 模块的一个特色，采样结果仅取决于两个输入引脚之间的电压差值，而与它们相对于 GNDA 的共模电压无关。在差分模式下 ADC0～ADC7 这 8 个输入引脚可以组成 4 个差分输入对，差分对 0 对应 ADC0 和 ADC1，差分对 1 对应 ADC2 和 ADC3，以此类推。差分采样和单端采样是可以并存的，例如当 ADC2 和 ADC3 配置为差分模式时，其他 6 个输入通道仍然可以配置为独立的单端模式。

图 5-9 所示为 ADC 差分输入采样的测试电路示意图。ADC2 和 ADC3 组成差分输入对，电位器输出电压送到 ADC2，在这里我们把 ADC3 作为一个参考，用分压电阻 $R2$ 和 $R3$ 提供固定的 1.5V 电压输入。说明：在实际应用当中，ADC3 当然也可以像 ADC2 那样是变动的，并非一定要固定。

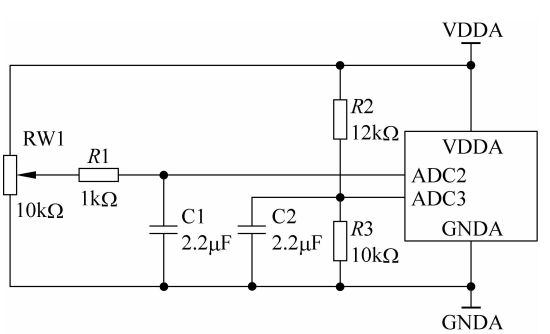

图 5-9　ADC 差分采样测试电路

程序清单 5-12 演示了 ADC 差分输入采样的用法。跟单端应用不同的是，要把 10 位采样结果看成是有符号数，如果 ADC2 的电压高于 ADC3 则结果是正值，低于则是负值，相等

就是 0。程序的运行结果仍然通过 UART 输出，当旋动电位器 RW1 时，会显示出正的电压或负的电压差值。读者可以拿一块万用表切换到 DC 电压挡，红黑表笔分别点在 ADC2 和 ADC3 上，与 UART 输出的结果进行对照。

1. 差分 ADC 采样初始化

差分输入 ADC 采样与普通的单端采样不同的地方在于，对于差分输入 ADC 采样，ADCSequStepConfig(ADC_BASE, 0, 0, ADC_CTL_D | ADC_CTL_CH1 | ADC_CTL_END | ADC_CTL_IE)设定 ADC_CTL_D 为差分输入采样，差分通道 1：ADC2 和 ADC3，如程序清单 5-12 所示。

程序清单 5-12　ADC 初始化函数 adcInit()

```
void adcInit(void)
{
    SysCtlPeriEnable(SYSCTL_PERIPH_ADC);                    //使能 ADC 模块
    SysCtlADCSpeedSet(SYSCTL_ADCSPEED_125KSPS);             //设置 ADC 采样速率
    ADCSequDisable(ADC_BASE, 0);                            //配置前先禁止采样序列
                  //采样序列配置：ADC 基址、采样序列编号、触发事件、采样优先级
    ADCSequConfig(ADC_BASE, 0, ADC_TRIGGER_PROCESSOR, 0);
                      //采样步进设置：ADC 基址、采样序列编号、步值、通道设置
    ADCSequStepConfig(ADC_BASE, 0, 0, ADC_CTL_D |           //差分输入采样
                                      ADC_CTL_CH1 |         //差分通道 1：ADC2 和 ADC3
                                      ADC_CTL_END |
                                      ADC_CTL_IE);
    ADCIntEnable(ADC_BASE, 0);                              //使能 ADC 中断
    IntEnable(INT_ADC0);                                    //使能 ADC 采样序列中断
    IntMasterEnable();                                      //使能处理器中断
    ADCSequEnable(ADC_BASE, 0);                             //使能采样序列
}
```

2. ADC 采样值主函数

主函数中，必须通过转换将 ADC 采样值转换为差分数值，如程序清单 5-13 所示。

程序清单 5-13　差分输入 ADC 采样主函数

```
int main(void)
{
...
        iVal = adcSample();                                 //ADC 采样
        iVal = iVal - 0x1FF;                                //转换成差分数值
        iVal = (iVal * 3000) / 1024;                        //转换成电压值
        sprintf(cBuf, "ADC0 = %d(mV)\r\n", iVal);           //输出格式化
        uartPuts(cBuf);                                     //通过 UART 显示结果
        SysCtlDelay(1500 * (TheSysClock / 3000));           //延时约 1500ms
...
}
```

代码分析：在 ADCSequStepConfig()函数中设置差分采样。其他地方与处理器触发采样一致。

注：完整的运行工程请参阅本书附光盘文件夹“第 5 章 ADC ADC_Difference_Sample”下的工程。

5.6.6　ADC 硬件过采样实例

目标：ADC 硬件过采样。

方法：通过配置 ADCHardwareOversampleConfigure()来设置硬件过采样。

现象：ADC 采样值通过 UART 端口输出，显示结果值较无过采样时稳定。

ADC 过采样是一种以牺牲速度换取精度的方法，可以用在对转换速度要求不太高的场合。程序清单 5-14 演示了 ADC 硬件过采样的用法。实际上利用的是 ADC 模块的硬件平均电路，能够对多达 64 个采样求取平均值。该例程是从 ADC 内置的温度传感器例程（程序清单 5-5）改进而来，增加了对函数 ADCHardwareOversampleConfigure()的调用。温度结果仍然从 UART 输出，我们会明显感到温度结果值的显示稳定多了。

硬件过采样主要使用了 ADCHardwareOversampleConfigure()函数配置。其他与采用定时器触发 ADC 采样完全一致。

程序清单 5-14　硬件过采样 ADC 初始化函数 adcInit()

```
void adcInit(void)
{
    SysCtlPeriEnable(SYSCTL_PERIPH_ADC);                    //使能 ADC 模块
    SysCtlADCSpeedSet(SYSCTL_ADCSPEED_125KSPS);             //设置 ADC 采样速率
    ADCSequDisable(ADC_BASE, 3);                            //配置前先禁止采样序列
    //采样序列配置：ADC 基址、采样序列编号、触发事件、采样优先级
    ADCSequConfig(ADC_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);
    ADCHardwareOversampleConfigure(ADC_BASE, 16);           //硬件过采样配置
    //采样步进设置：ADC 基址、采样序列编号、步值、通道设置
    ADCSequStepConfig(ADC_BASE, 3, 0, ADC_CTL_CH0 |
                                      ADC_CTL_END |
                                      ADC_CTL_IE);
    ADCIntEnable(ADC_BASE, 3);                              //使能 ADC 中断
    IntEnable(INT_ADC3);                                    //使能 ADC 采样序列中断
    IntMasterEnable();                                      //使能处理器中断

    ADCSequEnable(ADC_BASE, 3);                             //使能采样序列
}
```

注：完整的运行工程请参阅本书附光盘文件夹“第 5 章 ADC ADC_hardware_oversample”下的工程。

5.6.7　ADC 软件过采样实例

目标：ADC 软件过采样。

方法：通过配置 ADCSoftwareOversampleConfigure()设置软件过采样，通过函数 ADCSoftwareOversampleStepConfigure()设置软件过采样的步进值。通过 ADCSoftwareOversampleDataGet()函数读取 ADC 软件过采样的值。

现象：ADC 采样值通过 UART 端口输出，显示结果值较无过采样时稳定。

ADC 过采样是一种以牺牲速度换取精度的方法，可以用在对转换速度要求不太高的场合。程序清单 5-15 演示了 ADC 软件过采样的用法。利用库函数实现的软件过采样仅能够

对 8 个采样求取平均值，读者如果感兴趣，可以参照其源代码进行改进。该例程是从 ADC 内置的温度传感器例程（程序清单 5-5）改进而来，温度结果仍然从 UART 输出，我们会明显感到温度结果值的显示稳定多了。

1. 软件过采样 ADC 初始化子函数

软件过采样主要使用了 ADCSoftwareOversampleConfigure()函数配置。其中使用函数 ADCSoftwareOversampleStepConfigure()进行软件过采样步进设置，包括 ADC 基址、采样序列编号、步值、通道设置。其他设置与普通的处理器或者定时器触发采样一致，如程序清单 5-15 所示。

程序清单 5-15　软件过采样 ADC 初始化函数 adcInit()

```
void adcInit(void)
{
    SysCtlPeriEnable(SYSCTL_PERIPH_ADC);                    //使能 ADC 模块
    SysCtlADCSpeedSet(SYSCTL_ADCSPEED_125KSPS);             //设置 ADC 采样速率
    ADCSequDisable(ADC_BASE, 0);                            //配置前先禁止采样序列
        //采样序列配置：ADC 基址、采样序列编号、触发事件、采样优先级
    ADCSequConfig(ADC_BASE, 0, ADC_TRIGGER_PROCESSOR, 0);
    ADCSoftwareOversampleConfigure(ADC_BASE, 0, 8);         //软件过采样配置
        //软件过采样步进设置：ADC 基址、采样序列编号、步值、通道设置
    ADCSoftwareOversampleStepConfigure(ADC_BASE, 0, 0, ADC_CTL_CH0 |
                                                       ADC_CTL_END |
                                                       ADC_CTL_IE);

    ADCIntEnable(ADC_BASE, 0);                              //使能 ADC 中断
    IntEnable(INT_ADC0);                                    //使能 ADC 采样序列中断
    IntMasterEnable();                                      //使能处理器中断

    ADCSequEnable(ADC_BASE, 0);                             //使能采样序列
}
```

2. 软件过采样子函数

ADC 采样中，若使用软件过采样，则数据读取函数也应该变为 ADCSoftwareOversampleDataGet()，从而通过软件过采样读取数据转换结果，如程序清单 5-16 所示。

程序清单 5-16　软件过采样 ADC 采样函数 adcSample()

```
unsigned long adcSample(void)
{
    unsigned long ulValue;

    ADCProcessorTrigger(ADC_BASE, 0);                       //处理器触发采样序列
    while (!ADC_EndFlag);                                   //等待采样结束
    ADC_EndFlag = false;                                    //清除 ADC 采样结束标志
    ADCSoftwareOversampleDataGet(ADC_BASE, 0, &ulValue, 1); //软件过采样读取转换结果

    return(ulValue);
}
```

Stellaris 系列微控制器的部分产品中提供了模数转换器（ADC）模块。ADC 的硬件分

辨率为 10 位，但由于噪音和其他使精度变小的因素的影响，实际的精度小于 10 位。本应用文档提供了一个基于软件的过采样技术，从而使转换结果的有效位数（ENOB）得到了改善。文档中描述了对输入信号执行过采样的方法，以及在精度和整个系统性能上的影响。

注：完整的运行工程请参阅本书附光盘文件夹“第 5 章 ADC_software_oversample”下的工程。

5.7　过采样原理与实现

过采样，顾名思义就是从输入信号中采集额外的转换数据。模拟信号采样的标准约定指出：采样频率 f_S 至少是输入信号的最高频率成分 f_H 的两倍。这被称做奈奎斯特采样定理（Nyquist Theorem）。

等式 1　奈奎斯特采样定理：

$$f_S = 2f_H$$

只要所选的采样频率高于 f_S 就被看做是过采样。当过采样与平均技术相结合时，可改善 ENOB。这是可以实现的，因为在将过采样的结果进行平均的同时也将量化噪音进行了平均，这样就提高了信噪比（SNR），信噪比的提高会在 ENOB 上产生一个直接的影响，从而改善 ENOB。

精度上每提高一位，必须对信号进行 4 倍的过采样，即过采样频率 f_{OS} 与采样频率 f_S 的关系如等式 2 所示：

等式 2　过采样频率：

$$f_{OS} = 4^x \times f_S$$

x 为 ENOB 上需改进的位数（例如，需要改进 2 位，则 $x=2$）。

过采样对转换结果的精度的改善如图 5-10 所示。图 5-10 中，对输入信号进行 4 倍过采样（样品组用绿色和紫色表示）并进行平均。图 5-10 中显示的采样点阐明了原始的、带噪音信号与平均值之间的差值。该例中的噪音在单个采样值上对精度的影响达到±3 位。我们注意到平均值（橙色圆点）比大多数单个的采样值更接近于理想值，并且精确得多。

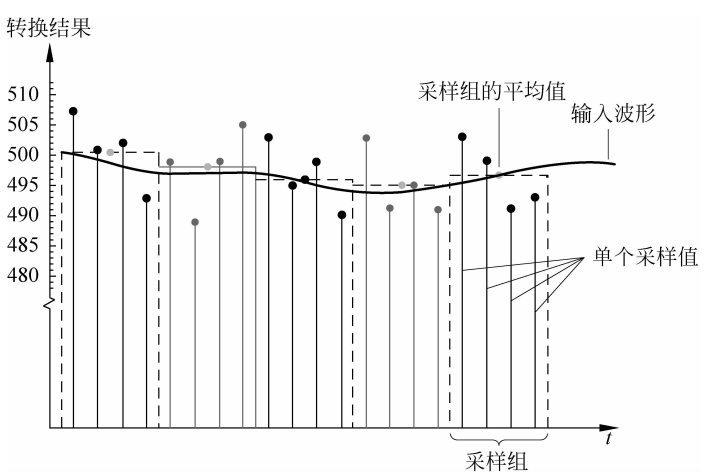

图 5-10　平均后的转换结果

5.7.1　平均

平均操作可看做是输入信号上的一个低通滤波器，当采样数据宽度(Simple Size)增加时滤波器的通带变窄。有两种方法可对转换结果进行平均：常规平均和滑动平均(Rolling Average)。

1. 常规平均

对输入信号进行 n 次采样，将采样值相加并将结果除以 n，这是常规平均。图 5-11 所示即为常规平均。当在过采样方案中使用常规平均时，使用该技术之后，用于计算平均值的采样数据被丢弃。每次应用程序需要一个新的转换结果时，重复该处理。

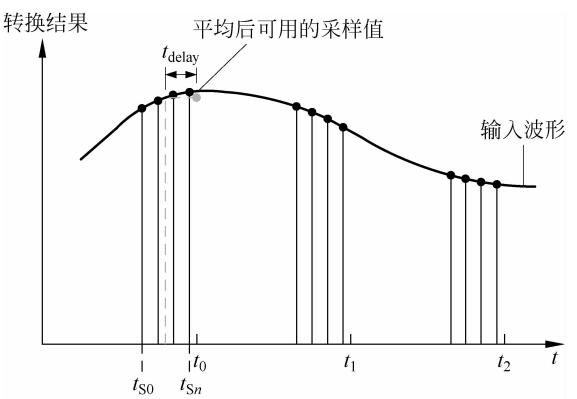

图 5-11　常规平均

在应用中，常规平均方案可理想地用于采样频率与 ADC 的采样率相比较小的情况。

要点：当在常规平均方案中执行 n 倍过采样时，有效的 ADC 采样率将按照相同的因子降低。例如，在对输入信号进行 4 倍过采样时，最大的有效 ADC 采样率降低为原来的 1/4，即采样率为 250ks/s 的 ADC 有效地变为 62.5ks/s 的 ADC。

图 5-11 显示的解决方案使用常规平均对输入源进行 4 倍过采样。在该例中，应用要求在每个 t 阶段(t_0、t_1、t_2 等)准备好一个新值(平均操作完成)。

在使用平均技术时，因为计算后的转换结果要与上面的 n 个采样点对应，因此稍微有一点延迟。延迟时间使用等式 3 中的公式来计算。

等式 3　平均后的采样延迟：

$$t_{delay} = (t_{Sn} - t_{S0})/2 + t_{process}$$

t_{S0} 为进行平均时第一个采样点出现的时间，t_{Sn} 为最后一个采样点出现的时间。中断处理程序处理采样数据所需的时间，并将供应用使用的平均 t_{proces} 分解到等式中。在图 5-11 中，延迟后的转换结构用橙色来突出表示。

2. 滑动平均

滑动平均在平均计算中使用存放 n 个最近采样值的采样缓冲区，允许 ADC 在其最大采样率时采样(ADC 采样率并不像常规平均那样减小为原来的 $1/n$)，这样它可理想地用于要求过采样和更高采样率的应用中。在未知状态中，采样缓冲区能够用有效的采样数据预先填充(通过捕获第一个“实际”数据点之前的 $n-1$ 个采样点)，也可保持为空，由应用来决

定。不预先填充缓冲区的危害是前面的 $n-1$ 个采样点包含无效的数据，并在滑动平均计算中产生不利的影响。如果这些影响可被应用所接受，并且如果软件能够解决前面的 $n-1$ 个偏移的采样点的可能性，则可去除缓冲区填充操作。

图 5-12 显示了采用滑动平均的过采样实例。图 5-12 中显示的情况为：输入信号进行 4 倍过采样，即采样缓冲区使用 4 个最近的采样值来计算平均值。在该例中，应用要求在每个 t 时刻有一个新的采样值。在 t_0 时刻计算第一个过采样的结果之前，采样缓冲区收集了 3 个采样值，这样提供给应用的第一个数据有效。

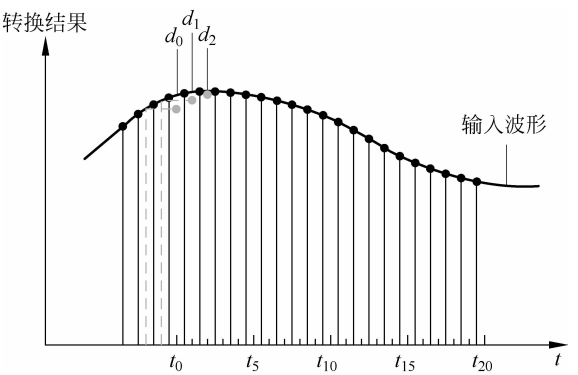

图 5-12　滑动平均

在使用滑动平均时，等式 3 中计算得来的采样延迟也同样适用。在图 5-12 中，t_0、t_1、t_2 时刻延迟后的值用橙色突出显示（分别显示为 d_0、d_1、d_2）。

要点：因为必须在每次中断过程中执行采样缓冲区处理，因此使用滑动平均增加了额外的处理开销。

5.7.2　过采样实现

TI Micro 在 ADC 中使用采样定序器结构，它使用一次触发就可采集到高达 17 个不同的采样值（来自任意的模拟通道），这样过采样的实现就变得非常简单。而通过向应用提供在任意给定的时刻对多个通道进行过采样的方法，使得软件的实现也具有极大的灵活性。

下面将给出使用 Stellaris 微控制器的多种过采样实现。有许多方法是将采样定序器的配置、ADC 触发和中断相结合来工作的。这里所举的例子焦点都集中在最常使用的技术上。

1. *使用驱动库函数的 8 倍过采样*

Stellaris 驱动库具有内置的允许进行高达 8 倍过采样的函数。该级别的过采样能够使 ENOB 改进大约 1.4 位，因此在大多数应用中已足够了。

使用驱动库的过采样函数是对输入信号进行过采样的最简单的方法。配置“典型”ADC 转换和过采样转换的主要不同在于函数调用。过采样函数有一个 ADCSoftwareOversample 前缀，很容易从标准 ADC 函数中识别出来。

一旦确定好 ADC 转换处理的参数（采样频率、触发源、通道等），写代码是非常简单的。例 5-1 中的代码段即为建立一个 8 倍过采样的 10ms 周期转换（由定时器触发）的代码。

例 5-1　利用驱动库函数的 8x 过采样。

程序清单 5-17.a　ADC 配置——驱动库函数

```
//初始化 ADC,使用定序器 0 对通道 1 进行 8x 过采样
//定序器将被其中一个通用定时器触发
ADCSequenceConfigure(ADC_BASE, 0, ADC_TRIGGER_TIMER, 0);
ADCSoftwareOversampleConfigure(ADC_BASE, 0, 8);
ADCSoftwareOversampleStepConfigure(ADC_BASE, 0, 0, (ADC_CTL_CH1 \
| ADC_CTL_IE | ADC_CTL_END));

//初始化定时器 0,每隔 10ms 触发一次 ADC 转换
TimerConfigure(TIMER0_BASE, TIMER_CFG_32_BIT_PER);
TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet() / 100);
TimerControlTrigger(TIMER0_BASE, TIMER_A, true);
```

程序清单 5-17.a 的 ADC 配置表示在采样完成时产生一个中断,这样就必须具有中断处理程序(见程序清单 5-17.b)。驱动库的过采样函数自动将采样的数据进行平均,因此,中断处理函数相对来说也是很基础的。但要记住:要将每次中断中计算的平均值和计算的开销提供给中断处理程序。

程序清单 5-17.b　ADC 中断处理程序

```
void
ADCIntHandler(void)
{
long lStatus;
//
//清除 ADC 中断
ADCIntClear(ADC_BASE, 0);
//
//获得 ADC 的平均数据
lStatus = ADCSoftwareOversampleDataGet(ADC_BASE, 0, &g_ulAverage);
//
//占位符,供 ADC 处理数据
}
```

在将配置步骤和中断处理程序放在适当位置后,启动转换处理。定时器打开(开始计数)之前,ADC 定序器和中断必须使能(见程序清单 5-17.c)。

程序清单 5-17.c　使能 ADC 和中断

```
//
//使能 ADC 定序器 0 及其中断 (在 ADC 和 NVIC 中)
ADCSequenceEnable(ADC_BASE, 0);
ADCIntEnable(ADC_BASE, 0);
IntEnable(INT_ADC0);
//
//使能定时器并启动转换处理
TimerEnable(TIMER0_BASE, TIMER_A);
```

2. 使用多个定序器或一个定时器实现大于 8 倍的过采样

驱动库的过采样函数最大只能进行 8 倍过采样(根据采样定序器的硬件限制),因此需

要更大的过采样因子的应用必须使用其他方法实现。本节将描述如何使用下面的两种方法来解决这个问题：在过采样频率下运行的多个采样定序器和一个定时器。

例 5-2　使用多个采样定序器的 16x 过采样。

采样定序器的灵活性允许对其进行多种配置。将采样定序器 0～采样定序器 2 累积起来可获得 16 个采样(8＋4＋4)，因此使用采样定序器 0～采样定序器 2 可实现 16 倍过采样，如图 5-13 所示。为使该级别的过采样能够工作，定序器中的所有阶段必须设置为对相同的模拟输入进行采样，这意味着丢弃了使用一个定序器采样多个输入的功能。

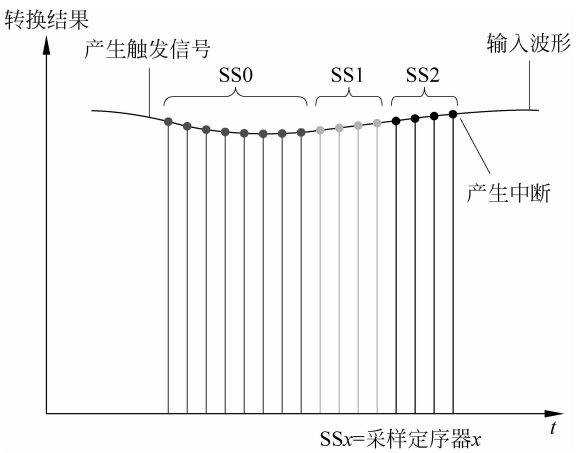

图 5-13　使用多个采样定序器的过采样

程序清单 5-18.a 使用定序器 0～定序器 2 配置一个 10ms 的周期转换。使用一个定时器触发就可启动所有的 3 个定序器的采样操作，而无需复杂的触发配置。为获得所需的结果，要对采样定序器的优先级进行配置。采样定序器 2 的优先级最低(即它最后采样)，并且在采样定序器 2 的最后一步之后，配置为发出一个“转换结束”中断。

程序清单 5-18.a　ADC 配置——多个采样定序器

```
//初始化 ADC,以便使用定序器 0～定序器 2 对通道 1 进行 16x 过采样；转换操作通过 GPTM 触发
ADCSequenceConfigure(ADC_BASE, 0, ADC_TRIGGER_TIMER, 0);
ADCSequenceConfigure(ADC_BASE, 1, ADC_TRIGGER_TIMER, 1);
ADCSequenceConfigure(ADC_BASE, 2, ADC_TRIGGER_TIMER, 2);
//配置定序器 0 的序列步骤
ADCSequenceStepConfigure(ADC_BASE, 0, 0, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 0, 1, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 0, 2, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 0, 3, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 0, 4, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 0, 5, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 0, 6, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 0, 7, (ADC_CTL_CH1 | ADC_CTL_END));

//配置定序器 1 的序列步骤
ADCSequenceStepConfigure(ADC_BASE, 1, 0, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 1, 1, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 1, 2, ADC_CTL_CH1);
```

```
ADCSequenceStepConfigure(ADC_BASE, 1, 3, (ADC_CTL_CH1 | ADC_CTL_END));

//配置定序器 2 的序列步骤
ADCSequenceStepConfigure(ADC_BASE, 2, 0, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 2, 1, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 2, 2, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 2, 3, (ADC_CTL_CH1 | ADC_CTL_IE | ADC_CTL_END));

//初始化定序器 0,每隔 10ms 触发一次 ADC 转换
TimerConfigure(TIMER0_BASE, TIMER_CFG_32_BIT_PER);
TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet() / 100);
TimerControlTrigger(TIMER0_BASE, TIMER_A, true);
```

在程序清单 5-18.b 中,中断处理程序必须收集 FIFO 的数据并进行平均计算。因为不需要处理函数开销就可以获得所需的结果,所以不使用 ADCSequenceDataGet()函数,并且使用直接的寄存器读操作来清空定序器的 FIFO。而使用 ADCSequenceDataGet()时,要求函数定义一个额外的 8 入口采样缓冲区,即使使用直接的寄存器读操作,中断处理程序中执行的总和计算和平均计算仍然会有可计算的开销。

程序清单 5-18.b　ADC 中断处理程序

```
void ADCIntHandler(void)
{
    unsigned long ulIdx;
    unsigned long ulSum = 0;
    ADCIntClear(ADC_BASE, 2);                                    //清除中断
//获得来自定序器 0 的数据
    for(ulIdx = 8; ulIdx; ulIdx--)
        {
            ulSum += HWREG(ADC_BASE + ADC_O_SSFIFO0);
        }

//获得来自定序器 1 和定序器 2 的数据
    for(ulIdx = 4; ulIdx; ulIdx--)
    {
        ulSum += HWREG(ADC_BASE + ADC_O_SSFIFO1);
        ulSum += HWREG(ADC_BASE + ADC_O_SSFIFO2);
    }
//将过采样的数据进行平均
    g_ulAverage = ulSum >> 4;
//占位符,以便 ADC 处理代码
}
```

在启动转换处理之前,将采样定序器和中断使能(如程序清单 5-18.c 所示)。

程序清单 5-18.c　使能 ADC 和中断

```
//使能定序器和中断
ADCSequenceEnable(ADC_BASE, 0);
ADCSequenceEnable(ADC_BASE, 1);
ADCSequenceEnable(ADC_BASE, 2);
ADCIntEnable(ADC_BASE, 2);
```

```
IntEnable(INT_ADC2);
TimerEnable(TIMER0_BASE, TIMER_A);                    //使能定序器并启动转换处理
```

例 5-3　使用在 f_{OS} 下运行的定时器进行 16x 过采样。

另一个实现 16x 过采样的方法(无须消耗 ADC 定序器的大部分资源)是使用一个在过采样频率下运行的周期定时器。例如，如果转换处理每 10ms 必须返回到主应用程序并且即将进行 16 倍过采样，则能够将定时器配置为每 625μs 获得一个采样值。让定时器在过采样频率下触发一次转换明显地产生了额外的 ADC 中断，这必须在应用程序中说明。

将 ADC 和定时器配置为执行上述操作的代码见程序清单 5-19.a。

程序清单 5-19.a　ADC 配置——在 f_{OS} 下运行的定时器

```
//初始化 ADC,以便在检测到一次触发时在通道 1、定序器 3 上获得一个采样值
ADCSequenceConfigure(ADC_BASE, 3, ADC_TRIGGER_TIMER, 0);
ADCSequenceStepConfigure(ADC_BASE, 3, 0, (ADC_CTL_CH1 | ADC_CTL_IE | ADC_CTL_
END));

//初始化定时器 0,每 625μs 触发一次 ADC 转换
TimerConfigure(TIMER0_BASE, TIMER_CFG_32_BIT_PER);
TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet() / 1600);
TimerControlTrigger(TIMER0_BASE, TIMER_A, true);
```

既然 ADC 在过采样频率下进行采样操作，中断处理程序必须知道已获得的采样数以及总和(见程序清单 5-19.b)。在累积了 16 次转换后，将这 16 个采样值进行平均，并清除全局采样计数变量和总和变量。

程序清单 5-19.b　ADC 中断处理程序

```
void ADCIntHandler(void)
{
ADCIntClear(ADC_BASE, 3);                             //清除中断
g_ulSum += HWREG(ADC_BASE + ADC_O_SSFIFO3);           //将新的采样值加到全局总和中
g_ucOversampleCnt++;                                  //g_ucOversampleCnt 加 1
//如果累积了 16 个采样值,则将它们平均并将全局变量复位
if(g_ucOversampleCnt == 16)
    {
        g_ulAverage = g_ulSum >> 4;
        g_ucOversampleCnt = 0;
        g_ulSum = 0;
    }

//占位符,以便 ADC 处理代码
}
```

最后，在使能定时器之前，将定序器 3 及其中断使能，并清除全局计数器和总和变量(见程序清单 5-19.c)。

程序清单 5-19.c　使能 ADC、中断并清除全局变量

```
//使能定序器和中断
ADCSequenceEnable(ADC_BASE, 3);
ADCIntEnable(ADC_BASE, 3);
```

```
IntEnable(INT_ADC3);
g_ucOversampleCnt = 0;                                    //将过采样计数器和总和变量
                                                            清零
g_ulSum = 0;
TimerEnable(TIMER0_BASE, TIMER_A);                        //使能定时器并启动转换处理
```

3. 使用滑动平均进行过采样

当采样频率接近 ADC 的最大采样率时，滑动平均非常有用。滑动平均应用中的主要元件是采样缓冲区，它在每次转换完成时减去/加上数据。

例 5-4 将 ADC 配置为每隔 100μs 进行一次采样，采样缓冲区含有 16 个入口。注意：应用程序不向采样缓冲区预先填充有效的数据，这样，前 16 个采样值必须相应地由软件来处理。ADC 配置为在定时器触发时采样，并在每次转换之后将处理器中断。

例 5-4　使用滑动平均每 100μs 过采样一次。

程序清单 5-20.a 所示为 ADC 配置——滑动平均。

程序清单 5-20.a　ADC 配置——滑动平均

```
//初始化 ADC，以便在检测到触发时在通道 1、定时器 3 上获得一个采样值
ADCSequenceConfigure(ADC_BASE, 3, ADC_TRIGGER_TIMER, 0);
ADCSequenceStepConfigure(ADC_BASE, 3, 0, (ADC_CTL_CH1 | ADC_CTL_IE | ADC_CTL_END));

//初始化定时器 0，每 100μs 触发一次 ADC 转换
TimerConfigure(TIMER0_BASE, TIMER_CFG_32_BIT_PER);
TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet() / 10000);
TimerControlTrigger(TIMER0_BASE, TIMER_A, true);
```

中断处理程序必须更新采样缓冲区并进行平均计算（见程序清单 5-20.b）。在每次 ADC 中断时，去掉采样缓冲区中的最后一个元素，缓冲区中剩下的数据移动一个位置。然后，在计算平均值之前将新的转换结果放在采样缓冲区的开始处。中断处理程序中执行的额外计算又一次增加了开销，这一点必须要考虑。

程序清单 5-20.b　ADC 中断处理程序

```
void ADCIntHandler(void)
{
ADCIntClear(ADC_BASE, 3);                                 //清除中断

if(g_ucOversampleIdx == 16)                               //检查 g_ucOversampleIdx，确保它的值在范围内
    {
        g_ucOversampleIdx = 0;
    }
g_ulSum -= g_ulSampleBuffer[g_ucOversampleIdx];           //从全局总和中减去最早的值
//用新的采样值代替最早的值
g_ulSampleBuffer[g_ucOversampleIdx] = HWREG(ADC_BASE + ADC_O_SSFIFO3);
g_ulSum += g_ulSampleBuffer[g_ucOversampleIdx];           //将新的采样值加到总和中
g_ucOversampleIdx++;                                      //g_ucOversampleIdx 加 1
g_ulAverage = g_ulSum >> 4;                               //从采样缓冲区的数据中获得平均值
//占位符，供 ADC 处理代码
}
```

在启动定时器之前，使能定时器及其中断（见程序清单 5-20.c）。

程序清单 5-20.c　使能 ADC 和中断

```
ADCSequenceEnable(ADC_BASE, 3);                    //使能定序器和中断
ADCIntEnable(ADC_BASE, 3);
IntEnable(INT_ADC3);
TimerEnable(TIMER0_BASE, TIMER_A);                 //使能定时器并启动转换处理
```

注：完整的运行过采样工程请参阅本书附光盘文件夹第 5 章 ADC_16x_oversample、ADC_hardware_oversampleCH0、ADC_software_oversampleCH0 下的工程。

5.8 小　　结

本章介绍了嵌入式技术中模拟量的采集和 AD 采样，首先介绍了 Cortex-M3 内核集成的 AD 的特性、功能，然后介绍了 ADC 使用过程中注意的事项，接下来列出了 ADC 的常用库函数，列举了几个 ADC 采样的例子，包括 ADC 单通道触发采样、多通道采样、内部温度传感器采样、定时器触发采样、外部触发采样等，最后分析了 ADC 的软硬件过采样及实现技术。

5.9 思　考　题

一、填空题

1. LM3S 的 ADC 模块是属于_____ AD 转换器。最大转换频率为_____。工作时采用_____方式使一次可以完成多路模拟量的采集。

2. 转换器采用内部的参考电压为_____。

3. LM3S 的 ADC 模块灵活的触发控制可以是_____、_____、_____、_____、_____等。

4. Stellaris 系 ARM 的 ADC 通过使用一种_____方法来收集采样数据，取代了传统 ADC 模块使用的单次采样或双采样的方法。

5. 采样控制和数据捕获由采样序列发生器进行处理。所有序列发生器的实现方法都相同，不同的只是各自可以_____和_____。

二、简答题

1. 简述 Stellaris 系列 ARM 的 ADC 转换器的主要指标及特性。

2. 利用库函数编写处理器触发单路 ADC 采样程序。

3. 利用寄存器定义读取的方式编写处理器触发单路 ADC 采样程序。

4. 简述 ADC 采样电路电源设计中应注意的问题。

5. 简述 ADC 采样电路中输入端电路设计应注意的问题。

6. 简述 ADC 过采样的原理及实现方法。

7. 利用 ADC 过采样技术编写 12 位精度的 ADC 采样程序。

8. 为了更好地发挥 Stellaris 系列的 10 位 ADC 特性，一般在电路中应做哪些处理？

9. 采用函数 ADCSequStepConfig()对 ADC 采样序列步进进行配置，要求：ADC 基址、采样序列 2、步值 0、采样 ADC0 输入后结束并申请中断。

10. 采用函数 ADCSequStepConfig()对 ADC 采样序列步进进行配置，要求：ADC 基址、采样序列 3、步值 0、采样温度传感器后结束并申请中断。

11. 采用函数 ADCSequStepConfig()对 ADC 采样序列步进进行配置，要求：ADC 基址、采样序列 1、步值 1、差分采样 ADC2/ADC3 输入后结束并申请中断。

12. Stellaris 系列 ARM 的 ADC 通过使用一种基于序列的可编程方法来收集采样数据，请解析何为基于序列的采集方法？

13. 简述 ADC 采样优先级设置方法。

14. 简述硬件过采样和软件过采样的区别，并指出其调用的函数。

15. 采用函数 ADCSequenceConfigure()对 ADC 采样序列进行配置：ADC 基址、采样序列 0、处理器触发、优先级 0。

16. 采用函数 ADCSequenceConfigure()对 ADC 采样序列进行配置：ADC 基址、采样序列 1、定时器触发、优先级 2。

17. 采用函数 ADCSequenceConfigure()对 ADC 采样序列进行配置：ADC 基址、采样序列 2、外部事件(PB5 中断)触发、优先级 3。

18. 采用函数 ADCSequenceConfigure()对 ADC 采样序列进行配置：ADC 基址、采样序列 3、模拟比较器 0、事件触发、优先级 1。

19. 编写程序实现将开发板上的 AD 模块采集到的电压数据用串行通信口送到 PC，PC 通过串口调试精灵或超级终端观看数据。

20. 简述采样序列发生器的功能。