应用报告
ZHCA511A–June 2009–Revised January 2013

针对 Stellaris® 系列微控制器的 ADC 过采样技术

Eric Ocasio

摘要

Stellaris 微控制器系列的某些成品成员具有一个模数转换器 (ADC) 模块。 ADC 的硬件分辨率为 10 位；然而，由于噪声和其它精度减少因素，真正的精度少于 10 位。 这份应用报告提供了一个基于软件的过采样技术，从而提高转换结果中有效位的数量 (ENOB)。 这份文档描述了过采样一个输入信号的方法以及对于精度和总体系统性能的影响。

内容

图片列表

1 过采样

正如名称所示，过采样从输入信号中采集额外的转换数据。 针对采样一个模拟信号的标准法则表明采样频率f_s应该至少是输入信号最高频率分量f_H的采样频率的两倍。 这被称为那奎斯特 (Nyquist) 定理（请见公式 1）。

Nyquist 定理 $f_S = 2\,f_H$ (1)

任何所选的高于f_s的采样频率被视为过采样，并且当与均衡技术组合在一起时，将改进 ENOB。 这是可能的，原因是平均过采样结果也会均分量化噪声，因此，提升了信噪比 (SNR)，这会对 ENOB 产生直接影响。

为了改进每个位的精度，信号必须由一个值为 4 的因子进行过采样，这意味着过采样频率f_{OS}和采样频率f_s间的关系如公式 2中所示。

过采样频率 $f_{OS} = 4^x * f_S$ (2)

在这里，x为需要增加的 ENOB（例如，要提升两位，x=2）。

图 1显示了过采样是如何提升转换结果的准确度的。 在这个线图中，输入信号被 4 倍过采样（样本组显示为绿色和紫色）并被平均。 显示的样本点图示了原始、嘈杂信号和平均值之间的差异；这个示例中的噪声对一个单一信号准确度的影响为 ±3 位。 请注意，平均值（橙色点）比大多数单一样本更加靠近理想值。

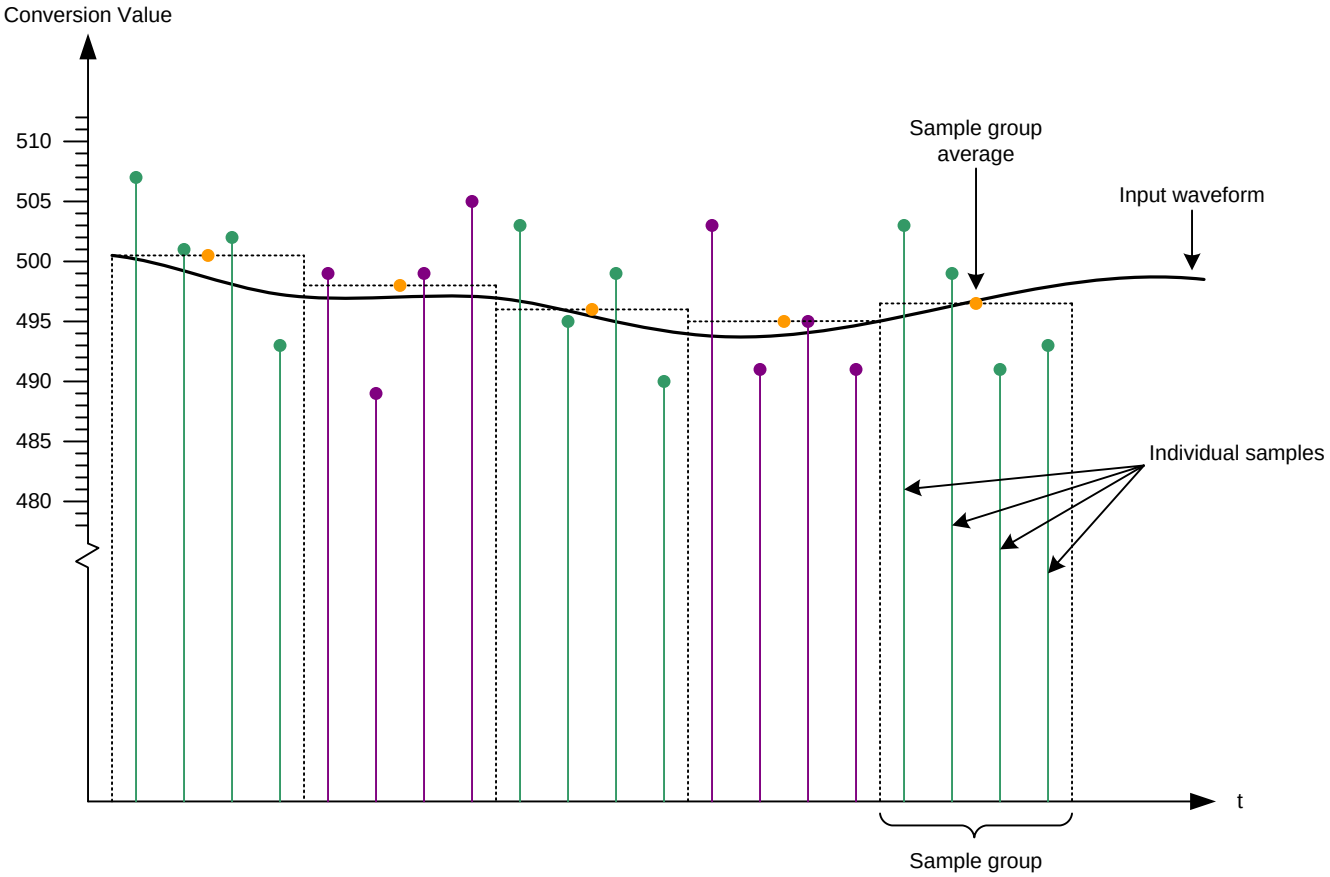

图 **1.** 平均转换结果

2 取平均值

取平均值作为输入信号上的低通滤波器，滤波器的通带随着样本尺寸的增加而变窄。 当对转换结果取平均值时，可采取两个方法：*正常平均*或*移动平均*。

2.1 正常平均

获得*n*个样本，将它们相加，并将相加的结果除以*n*，这被称为*正常平均*，如图 1所示。 当在过采样情况下使用正常平均时，在此技术被使用后，计算中使用的样本数据被丢弃。 这个过程在每次应用程序需要一个新的转换结果时被重复进行。

在一个应用程序中，正常取平均值的方法非常适用于采样频率低于 ADC 采样率的情况。

注: **请注意：**当在正常情况下进行*n*倍过采样时，有效 ADC 采样速率将减少同样的倍数。 例如，对一个输入信号进行 4 倍过采样将使最大有效 ADC 采样速率除以 4，这意味着一个每秒 250K 的 ADC 有效采样速率将变成每秒 62.5K 有效采样速率。

图 2显示了一个正常取平均值被用来对输入源进行 4 倍过采样的情况。对于这个示例，此应用程序需要在 t（*t0*，*t1*，*t3*等）的每一步上准备一个新的转换值（取平均值完成）。

当使用取平均值技术时，由于它与最后 n 个采样的平均值相对应，所以会有一个与计算得出的转换结果相关的轻微延迟。 可使用公式 3中显示的公式来计算此延迟。

已平均的采样延迟$t_{delay} = (t_{sn} - t_{s0})/2 + t_{process}$ (3)

在这里，t_{S0}是首次发生的时间，而t_{Sn}是最后一个采样发生的时间。 在等式中，中断处理程序所需的用来处理采样数据以及计算提供给应用程序的平均$t_{处理}$也被计算在内。 在图 2中，延时的转换结果用橙色标出。

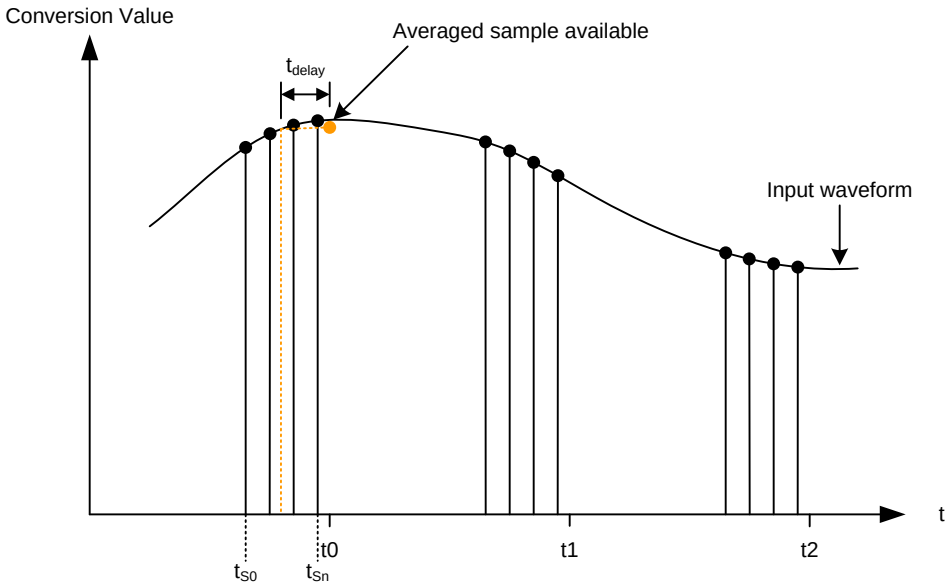

图 **2.** 正常取平均值

2.2 继动平均

一个*继动平均*在取平均数计算中使用 n 个最近样本的采样缓冲器，从而使得 ADC 能够以其*最大速率*进行采样（ADC 采样速率不会如正常取平均数中那样被减少*n*倍），这使得它非常适合于要求过采样和更高采样速率的应用。 根据应用的不同，可用有效采样数据将采样缓冲器预先填满（在第一个“真实”数据点前获得*n-1* 个样本），或者可被保留在未知状态。 不将缓冲器预先填满的缺点是*n-1*个样本包含无效数据并会对继动平均计算产生负面影响。 如果应用程序可接受的话，可在软件负责解决最初*n-1*个采样偏离可能性的情况下不进行缓冲器填充。

图 3显示了使用继动平均进行过采样的示例。 此线图显示了输入信号被 4 被过采样的情况，这意味着采样缓冲器使用四个最近的采样来计算平均值。 在这个示例中，应用程序在*t*的每一步需要一个新样本。 第一个过采样结果在*t0*上被计算得出之前，采样缓冲器搜集三个样本，这样提供给应用程序的第一个数据是有效的。

当使用一个继动平均时，采用公式 3来计算同样的延迟。 在图 3中，针对*t0*，*t1*，*t2*的延迟值（分别显示为 d0，d1 和 d2）用橙色标出。

请注意：由于在每个中断期间必须执行采样缓冲器操作，所以继动平均的使用会增加额外的处理开销。

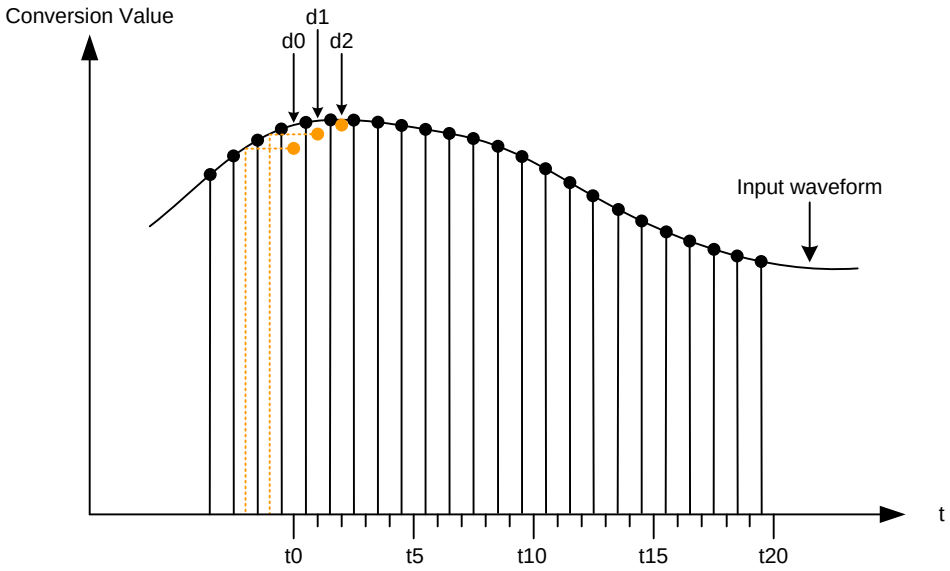

图 **3.** 继动平均

3 执行

ADC 中的采样序列发生器架构通过使用一个单一触发来采集多达 17 个单独采样（来自任一模拟通道）来简化过采样。 这样，通过为一个应用程序提供在任一指定时间过采样一定数量通道的方法来实现软件的灵活性。

这一部分显示了使用 Stellaris 微控制器所实现的多种过采样。 有多种方法可将采样序列发生器配置，ADC 触发器和中断组合在一起使用，然而，下面显示的示例着重于最有可能使用的技术。

所有示例代码使用 Stellaris 外设驱动程序库 ADC 函数。 针对驱动程序库和软件示例的源代码显示在http://www.ti.com网页上的应用注解中。

3.1 使用驱动程序库函数进行高达 8 倍的过采样

Stellaris 外设驱动程序库具有可实现高达 8 倍过采样的内置函数。 对于大多数应用程序，由于将 ENOB 提升了大约 1.4 位，所以这个水平上的过采样已经足够满足需要。

使用驱动程序库过采样函数是过采样输入信号的最简便的方法。 配置一个“典型”ADC 转换与一个过采样转换间主要的差别是函数调用。 过采样函数具有一个*ADCSoftwareOversample*前缀，可轻易地与标准 ADC 函数区分开来。

一旦确定了 ADC 转换过程所需的参数（采样频率、触发源、通道等），就直接写入代码。 例如，建立一个 10ms 8 倍过采样的周期转换（由一个定时器触发）的代码由Example 1中所示的代码段组成。

3.2 使用驱动程序库函数的 8 倍过采样

Example 1. 代码段 **1.a. ADC** 配置 **-** 驱动程序库函数

```
//
// Initialize the ADC to oversample channel 1 by 8x using sequencer 0.
// Sequencer will be triggered by one of the general-purpose timers.
//
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));
//
// Initialize Timer 0 to trigger an ADC conversion once every 10 milliseconds
//
TimerConfigure(TIMER0_BASE, TIMER_CFG_32_BIT_PER);
TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet() / 100);
TimerControlTrigger(TIMER0_BASE, TIMER_A, true);
```

代码段 1.a中显示的 ADC 配置在采样完成时命令一个中断发生，这意味着必须执行一个中断处理程序（请见 代码段 1.b）。 由于驱动程序库过采样函数自动平均采样的数据，中断处理程序函数相对简单。 请牢记，在每次中断期间计算平均值会增加中断处理程序的计算开销。

Example 2. 代码段 **1.b. ADC** 中断处理程序

```
void
ADCIntHandler(void)
{
long lStatus;
//
// Clear the ADC interrupt
//
ADCIntClear(ADC_BASE, 0);
//
// Get averaged data from the ADC
//
lStatus = ADCSoftwareOversampleDataGet(ADC_BASE, 0,&g_ulAverage);
//
// Placeholder for ADC processing code
//
}
```

当配置步骤和中断处理程序准备就绪时，转换过程被启动。 在定时器被打开前（开始计数），ADC 序列发生器和中断必须被启用（请进啊 代码段 1.c）。

Example 3. 代码段 1.c. 启用 ADC 和中断

```
//
// Enable ADC sequencer 0 and its interrupt (in both the ADC and NVIC)
//
ADCSequenceEnable(ADC_BASE, 0);
ADCIntEnable(ADC_BASE, 0);
IntEnable(INT_ADC0);
//
// Enable the timer and start conversion process
//
TimerEnable(TIMER0_BASE, TIMER_A);
```

4 使用多个序列发生器或一个定时器进行的超过 8 倍的过采样

驱动程序库过采样函数的过采样倍数限制为 8 倍（基于采样序列发生器的硬件限制），这意味着要求更大过采样因子的应用程序必须使用一个另外的执行方法。 这一部分显示了如何使用两种方法来处理这一情况：多采样序列发生器和一个运行频率为过采样频率的定时器。

4.1 使用多个采样序列发生器实现的 16 倍过采样

采样序列发生器的灵活性可实现多种配置。 为了进行 16 倍过采样，可使用采样序列发生器 0-2，这是因为它们总容量为 16 个样本 (8+4+4)。 为了使这个水平上的过采样能够运行，序列发生器内的所有步骤必须被设定为采样同一个模拟输入，这意味着无法使用一个单一序列发生器来采样多个输入。

代码段 2.a使用序列发生器 0-2 来配置一个 10ms 周期转换。 一个单一定时器触发器被用来启动所有 3 个序列发生器上的采样，这样就免除了对于复杂触发器配置的需要。 为了获得所需的结果，配置了采样序列发生器的优先级，这样，采样序列发生器 2 的优先级最低（意味着其最后采样），而“转换终止”中断被配置成在采样序列发生器 2 的最后一步后被置为有效（如图 4中所示）。

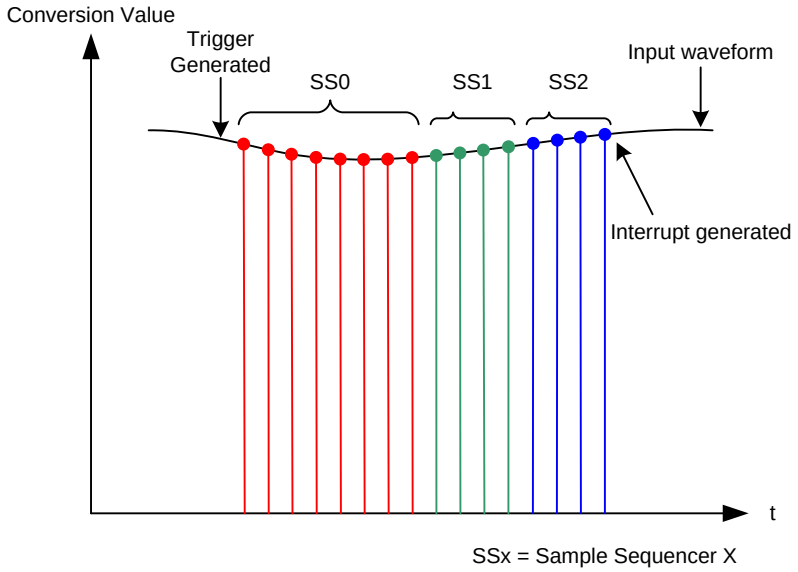

图 **4. 16** 倍过采样

***Example 4.** 代码段 **2.a. ADC** 配置 **-** 多个采样序列发生器*

```
//
// Initialize the ADC for 16x oversampling on channel 1 using sequencers
// 0-2. The conversion is triggered by a GPTM.
//
ADCSequenceConfigure(ADC_BASE, 0, ADC_TRIGGER_TIMER, 0);
ADCSequenceConfigure(ADC_BASE, 1, ADC_TRIGGER_TIMER, 1);
ADCSequenceConfigure(ADC_BASE, 2, ADC_TRIGGER_TIMER, 2);
//
// Configure sequence steps for sequencer 0
//
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));
//
// Configure sequence steps for sequencer 1
//
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));
//
// Configure sequence steps for sequencer 2
//
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));
//
// Initialize Timer 0 to trigger an ADC conversion once every 10 milliseconds
//
TimerConfigure(TIMER0_BASE, TIMER_CFG_32_BIT_PER);
TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet() / 100);
TimerControlTrigger(TIMER0_BASE, TIMER_A, true);
```

在 代码段 2.b中，中断处理程序必须从先进先出 (FIFO) 中采集数据并执行取平均值计算。*ADCSequenceDataGet*函数并未在此处使用，这是因为所需结果的获得不涉及函数开销，所以直接寄存器读取被用来清空序列发生器 FIFO。 *ADCSequenceDataGet*的使用要求函数定义一个额外的 8 入口采样缓冲器。 即使使用直接寄存器读取，仍然会在中断处理程序内产生由执行总和以及取平均值计算所导致的计算开销。

***Example 5.** 代码段 **2.b. ADC** 中断处理程序*

```
void
ADCIntHandler(void)
{
unsigned long ulIdx;
unsigned long ulSum = 0;
//
// Clear the interrupt
//
ADCIntClear(ADC_BASE, 2);
//
```

Example 5. 代码段 2.b. ADC 中断处理程序 (continued)

```
// Get the data from sequencer 0
//
for(ulIdx = 8; ulIdx; ulIdx--)
{
ulSum += HWREG(ADC_BASE + ADC_O_SSFIFO0);
}
//
// Get the data from sequencers 1 and 2
//
for(ulIdx = 4; ulIdx; ulIdx--)
{
ulSum += HWREG(ADC_BASE + ADC_O_SSFIFO1);
ulSum += HWREG(ADC_BASE + ADC_O_SSFIFO2);
}
//
// Average the oversampled data
//
g_ulAverage = ulSum >> 4;
//
// Placeholder for ADC processing code
//
}
```

在启动转换过程前，采样序列发生器和中断被启用（请见 代码段 2.c）。

Example 6. 代码段 2.c. 启用 ADC 和中断

```

//
// Enable the sequencers and interrupt
//
ADCSequenceEnable(ADC_BASE, 0);
ADCSequenceEnable(ADC_BASE, 1);
ADCSequenceEnable(ADC_BASE, 2);
ADCIntEnable(ADC_BASE, 2);
IntEnable(INT_ADC2);
//
// Enable the timer and start conversion process
//
TimerEnable(TIMER0_BASE, TIMER_A);
```

4.2 使用运行频率为 f_{OS} 的定时器进行 16 倍过采样

进行过采样的另外方法（不会消耗大部分 ADC 序列发生器资源）是使用运行频率为过采样频率的周期定时器。 例如，如果一个转换必须在每 10ms 返回到主应用程序并且被 16 倍过采样，一个定时器可被配置成每 625µs 获取一个单一样本。 由于定时器在过采样频率上触发一个转换会明显地生成额外的 ADC 中断通信量，必须在应用中将其考虑在内。

代码段 3.a中显示了将 ADC 和定时器配置成此种运行方式的代码。

Example 7. 代码段 3.a. ADC 配置 - 运行频率为 f_{OS} 的定时器

```
//
// Initialize the ADC to take a single sample on channel 1, sequencer 3
// when a trigger is detected.
//
ADCSequenceConfigure(ADC_BASE, 3, ADC_TRIGGER_TIMER, 0);
ADCSequenceStepConfigure(ADC_BASE, 3, 0, (ADC_CTL_CH1 | ADC_CTL_IE \
                         | ADC_CTL_END));
//
// Initialize Timer 0 to trigger an ADC conversion once every 625 microseconds
//
TimerConfigure(TIMER0_BASE, TIMER_CFG_32_BIT_PER);
TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet() / 1600);
TimerControlTrigger(TIMER0_BASE, TIMER_A, true);
```

现在，ADC 在过采样频率上进行采样，中断处理程序必须掌握已获取的样本数量和总和（请见 代码段 3.b）。 当已经收集了 16 个转换时，执行取平均值计算，而全局样本数量和总和变量被清零。

Example 8. 代码段 3.b. ADC 中断处理程序

```
{
//
// Clear the interrupt
//
ADCIntClear(ADC_BASE, 3);
//
// Add the new sample to the global sum
//
g_ulSum += HWREG(ADC_BASE + ADC_O_SSFIFO3);
//
// Increment g_ucOversampleCnt
//
g_ucOversampleCnt++;
//
// If 16 samples have accumulated, average them and reset globals
//
if(g_ucOversampleCnt == 16)
{
    g_ulAverage = g_ulSum >> 4;
    g_ucOversampleCnt = 0;
    g_ulSum = 0;
}
//
// Placeholder for ADC processing code
//
}
```

最后，在启用定时器之前，序列发生器 3 和其中断被启用，而全局计数器和总和变量被清零（请见 代码段 3.c）。

Example 9. 代码段 3.c. 启用 ADC，中断和全局变量

```
//
// Enable the sequencer and interrupt
//
ADCSequenceEnable(ADC_BASE, 3);
ADCIntEnable(ADC_BASE, 3);
IntEnable(INT_ADC3);
//
// Zero the oversample counter and the sum
//
g_ucOversampleCnt = 0;
g_ulSum = 0;
//
// Enable the timer and start conversion process
//
TimerEnable(TIMER0_BASE, TIMER_A);
```

5 使用继动平均进行过采样

继动平均方法在采样频率接近 ADC 的最大采样速率是比较有用。 继动平均应用的主要组件是采样缓冲器，此缓冲器在每次转换完成时减少和增加数据。

在5.1 节中，ADC 被配置成每 100µs 采样一次，而采样缓冲器包含 16 个条目。 请注意，应用不会用有效数据预填充采样缓冲器，所以最初 16 个样本必须由软件进行相应的处理。 ADC 被配置成在一个定时器触发时采样并在每次转换后中断处理器。

5.1 每 100 微妙使用继动平均进行过采样

Example 10. 代码段 4.a. ADC 配置 - 继动平均

```
//
// Initialize the ADC to take a single sample on channel 1, sequencer 3
// when a trigger is detected.
//
ADCSequenceConfigure(ADC_BASE, 3, ADC_TRIGGER_TIMER, 0);
ADCSequenceStepConfigure(ADC_BASE, 3, 0, (ADC_CTL_CH1 | ADC_CTL_IE \
                         | ADC_CTL_END));
//
// Initialize Timer 0 to trigger an ADC conversion once every 100 microseconds
//
TimerConfigure(TIMER0_BASE, TIMER_CFG_32_BIT_PER);
TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet() / 10000);
TimerControlTrigger(TIMER0_BASE, TIMER_A, true);
```

中断处理程序负责更新采样缓冲器并执行取平均值计算（请见 代码段 4.b）。 对于每个 ADC 中断，采样缓冲器中的最后一个元素被丢弃，而剩余的数据在缓冲器内部被移动一个位置。 然后，在取平均值被计算前，新的转换结果被放置在采样缓冲器的开头。 此外，必须将中断处理程序内执行的额外计算所增加的开销考虑在内。

Example 11. 代码段 4.b. ADC 中断处理程序

```
void
ADCIntHandler(void)
{
//
```

Example 11. 代码段 4.b. ADC 中断处理程序 (continued)

```
// Clear the interrupt
//
ADCIntClear(ADC_BASE, 3);
//
// Check g_ucOversampleIdx to make sure that it is within range
//
if(g_ucOversampleIdx == 16)
{
   g_ucOversampleIdx = 0;
}
   //
   // Subtract the oldest value from the global sum
   //
g_ulSum -= g_ulSampleBuffer[g_ucOversampleIdx];
//
// Replace the oldest value with the new sample value
//
g_ulSampleBuffer[g_ucOversampleIdx] = HWREG(ADC_BASE + ADC_O_SSFIFO3);
//
// Add the new sample to the overall sum
//
g_ulSum += g_ulSampleBuffer[g_ucOversampleIdx];
//
// Increment g_ucOversampleIdx
//
g_ucOversampleIdx++;
//
// Get the averaged value from the sample buffer data
//
g_ulAverage = g_ulSum >> 4;
//
// Placeholder for ADC processing code
//
}
```

此外，在定时器被启动前，序列发生器和其中断被启用（请见 代码段 4.c）。

Example 12. 代码段 4.c. 启用 ADC 和中断

```

//
// Enable the sequencer and the interrupt
//
ADCSequenceEnable(ADC_BASE, 3);
ADCIntEnable(ADC_BASE, 3);
IntEnable(INT_ADC3);

//
// Enable the timer and start conversion process
//
TimerEnable(TIMER0_BASE, TIMER_A);
```

6 需要考虑的问题

本文档中所描述的过采样技术对于总体系统性能会产生明显的影响，这是因为它们需要执行取平均值计算的额外代码、额外中断和大多数采样序列发生器资源。 为应用选择最合适的技术需要分析已增加的流量和更大中断处理程序之间的平衡。

7 结论

采样序列发生器架构为执行过采样技术提供广泛选项。 当与软件取平均值计算组合在一起时，此架构使得系统设计人员能够有效地实现采样频率、系统性能和采样分辨率之间的工程平衡。

8 参考

以下相关文档和软件可从Stellaris 网址为：http://www.ti.com/stellaris：

- Stellaris LM3S 微控制器数据表（单独器件文档在产品选择工具中提供）。
- Stellaris 外设驱动程序库。 可从www.ti.com/tool/sw-drl内下载。
- StellarisWare 驱动程序库用户手册，版号 SW-DRL-UG (SPMU019)

重要声明

德州仪器(TI) 及其下属子公司有权根据 JESD46 最新标准, 对所提供的产品和服务进行更正、修改、增强、改进或其它更改， 并有权根据 JESD48 最新标准中止提供任何产品和服务。客户在下订单前应获取最新的相关信息, 并验证这些信息是否完整且是最新的。所有产品的销售都遵循在订单确认时所提供的TI 销售条款与条件。

TI 保证其所销售的组件的性能符合产品销售时 TI 半导体产品销售条件与条款的适用规范。仅在 TI 保证的范围内，且 TI 认为 有必要时才会使用测试或其它质量控制技术。除非适用法律做出了硬性规定，否则没有必要对每种组件的所有参数进行测试。

TI 对应用帮助或客户产品设计不承担任何义务。客户应对其使用 TI 组件的产品和应用自行负责。为尽量减小与客户产品和应 用相关的风险，客户应提供充分的设计与操作安全措施。

TI 不对任何 TI 专利权、版权、屏蔽作品权或其它与使用了 TI 组件或服务的组合设备、机器或流程相关的 TI 知识产权中授予 的直接或隐含权限作出任何保证或解释。TI 所发布的与第三方产品或服务有关的信息，不能构成从 TI 获得使用这些产品或服 务的许可、授权、或认可。使用此类信息可能需要获得第三方的专利权或其它知识产权方面的许可，或是 TI 的专利权或其它 知识产权方面的许可。

对于 TI 的产品手册或数据表中 TI 信息的重要部分，仅在没有对内容进行任何篡改且带有相关授权、条件、限制和声明的情况 下才允许进行复制。TI 对此类篡改过的文件不承担任何责任或义务。复制第三方的信息可能需要服从额外的限制条件。

在转售 TI 组件或服务时，如果对该组件或服务参数的陈述与 TI 标明的参数相比存在差异或虚假成分，则会失去相关 TI 组件 或服务的所有明示或暗示授权，且这是不正当的、欺诈性商业行为。TI 对任何此类虚假陈述均不承担任何责任或义务。

客户认可并同意，尽管任何应用相关信息或支持仍可能由 TI 提供，但他们将独力负责满足与其产品及在其应用中使用 TI 产品 相关的所有法律、法规和安全相关要求。客户声明并同意，他们具备制定与实施安全措施所需的全部专业技术和知识，可预见 故障的危险后果、监测故障及其后果、降低有可能造成人身伤害的故障的发生机率并采取适当的补救措施。客户将全额赔偿因 在此类安全关键应用中使用任何 TI 组件而对 TI 及其代理造成的任何损失。

在某些场合中，为了推进安全相关应用有可能对 TI 组件进行特别的促销。TI 的目标是利用此类组件帮助客户设计和创立其特 有的可满足适用的功能安全性标准和要求的终端产品解决方案。尽管如此，此类组件仍然服从这些条款。

TI 组件未获得用于 FDA Class III（或类似的生命攸关医疗设备）的授权许可，除非各方授权官员已经达成了专门管控此类使 用的特别协议。

只有那些 TI 特别注明属于军用等级或“增强型塑料”的 TI 组件才是设计或专门用于军事/航空应用或环境的。购买者认可并同 意，对并非指定面向军事或航空航天用途的 TI 组件进行军事或航空航天方面的应用，其风险由客户单独承担，并且由客户独 力负责满足与此类使用相关的所有法律和法规要求。

TI 已明确指定符合 ISO/TS16949 要求的产品，这些产品主要用于汽车。在任何情况下，因使用非指定产品而无法达到 ISO/TS16949 要求，TI不承担任何责任。

	产品		应用
数字音频	www.ti.com.cn/audio	通信与电信	www.ti.com.cn/telecom
放大器和线性器件	www.ti.com.cn/amplifiers	计算机及周边	www.ti.com.cn/computer
数据转换器	www.ti.com.cn/dataconverters	消费电子	www.ti.com/consumer-apps
DLP® 产品	www.dlp.com	能源	www.ti.com/energy
DSP - 数字信号处理器	www.ti.com.cn/dsp	工业应用	www.ti.com.cn/industrial
时钟和计时器	www.ti.com.cn/clockandtimers	医疗电子	www.ti.com.cn/medical
接口	www.ti.com.cn/interface	安防应用	www.ti.com.cn/security
逻辑	www.ti.com.cn/logic	汽车电子	www.ti.com.cn/automotive
电源管理	www.ti.com.cn/power	视频和影像	www.ti.com.cn/video
微控制器 (MCU)	www.ti.com.cn/microcontrollers		
RFID 系统	www.ti.com.cn/rfidsys		
OMAP应用处理器	www.ti.com/omap		
无线连通性	www.ti.com.cn/wirelessconnectivity	德州仪器在线技术支持社区	www.deyisupport.com

邮寄地址： 上海市浦东新区世纪大道 1568 号，中建大厦 32 楼 邮政编码： 200122