恩智浦半导体
应用笔记

文档编号：AN12217
第0版，2018年8月

S32K1xx ADC 指南、规格和配置

作者：恩智浦半导体

1. 简介

NXP S32K1xx 汽车微控制器具有 12 位逐次逼近模数转换器（SAR ADC），可用于模拟输入信号的采集和数字化。

本应用笔记提供了有关以下基本主题的信息，便于从ADC模块的使用中获得最大收益：

- 理解 ADC 常用术语、误差来源和规格。
- 提高测量精度的最佳做法。
- S32K1xx 系列的常见触发配置示例。

目录

2. ADC 概念、误差源和规格

本节解释了用于表征 ADC 的概念和术语以及潜在的误差源，并提供了S32K1xx 系列数据表中的规范参数。

2.1. ADC 基本概念

分辨率：ADC 数字输出中代表模拟输入信号的位数。对于 S32K1xx 系列，分辨率可配置为 8、10 或 12 位。

参考电压：ADC需要一个参考电压，用于与模拟输入进行逐次近似比较，以产生数字输出。数字输出是模拟输入相对于该参考电压的比率。

VREF = VREFH – VREFL

其中：

VREFH = 高参考电压

VREFL = 低参考电压

ADC 输出公式：ADC 的转换公式用于计算特定模拟输入电压对应的数字输出。该等式假设没有引入错误的理想 A/D 转换。

$$ADCresult = \frac{(2^N)(Vin)}{VREF}$$

公式 1 – ADC 转换器公式

其中：

ADC 结果 = 转换产生的数字输出值

N = ADC 分辨率

VREF = 参考电压

Vin = 模拟输入电压

最低有效位（LSB）：最低有效位（LSB）是等于 ADC 最小分辨率的电压单位，即导致数字输出变化的最小增量电压。

LSB 等于参考电压除以 ADC 的最大计数：

$$LSB = \frac{VREF}{2^N}$$

公式 2 – LSB 公式

N = ADC 分辨率。对于S32K1xx，这可以是8/10/12位。

VREF = 模拟参考电压。

ADC 实际传递函数：ADC 将输入电压转换为相应的数字编码。描述此行为的曲线是实际传递函数，包括 ADC 模块本身固有的所有误差。

ADC 理想传递函数：*理想的传递函数*代表 ADC 的行为，假设它是完全线性的，或者不管输入的初始电平如何，输入电压的给定变化将在转换代码中产生相同的变化。理想传递函数的分级方式取决于 ADC 使用的量化方法。两种可能的方法是：

- **无补偿量化：**第一步在 1LSB 处执行，每个连续步骤以 1LSB 间隔执行，最后一步在 VREFH – 1LSB 处执行。
- **½LSB 补偿量化：**第一步在 ½LSB 处执行，每个连续步骤以 1LSB 间隔执行，最后一步在 VREFH – 1½LSB 处执行。

下图显示了无补偿量化和 ½LSB 补偿量化方法的理想传递函数图，分辨率为 3 位，VREF = 8V。

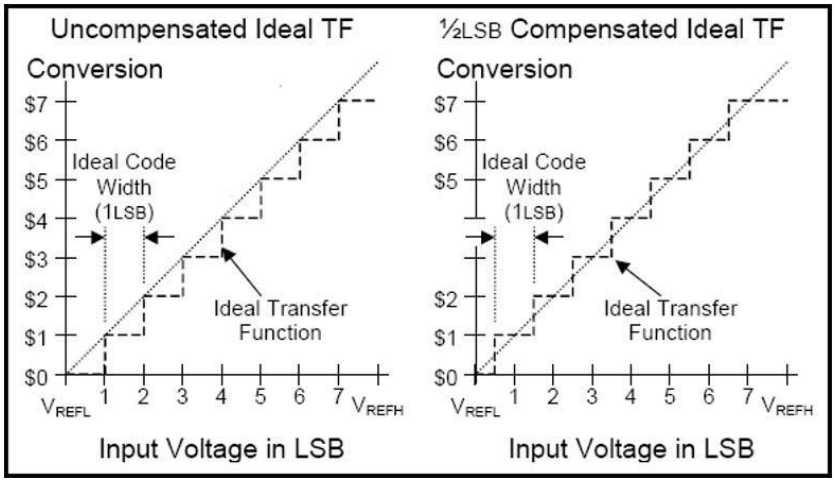

图1 - 理想的传递函数

2.2. ADC 测量中的误差来源

本节介绍了一些影响 ADC 准确执行 A/D 测量的典型因素。

参考电压噪声

ADC 输出与模拟输入电压和参考电压成正比。不稳定的参考电压（例如由电源轨中的噪声引起）会导致转换后的数字输出发生变化。

例子：

- 对于 5 V的 参考电压和 1V 的输入电压，12位分辨率的 ADC 使用公式 1结果为 819。
- 随着绝对参考电压增加 50mV（即 VREF = 5.05V），相同 1V 输入电压的转换值变为 811。
- 产生的参考电压噪声误差为 811 - 819 = - 8LSB。

模拟输入信号噪声

模拟输入信号中微小的高频变化可能会在 ADC 采样过程中导致较大的转换误差。周围电气设备的电磁辐射（EMI 噪声）同样会引起噪声，从而降低转换精度。

如果输入信号中存在的噪声高于 1LSB，这会大幅减少转换结果中可靠位的数量，因为最低有效位会因信号变化而不断变化。

模拟信号源电阻

由于电流流入引脚，模拟信号源的阻抗或信号源与输入引脚之间的串联电阻（RIN）会导致其两端出现压降。可以将其理解为从 ADC “向外看”到驱动要采样的输入信号的源中观察到的电阻。

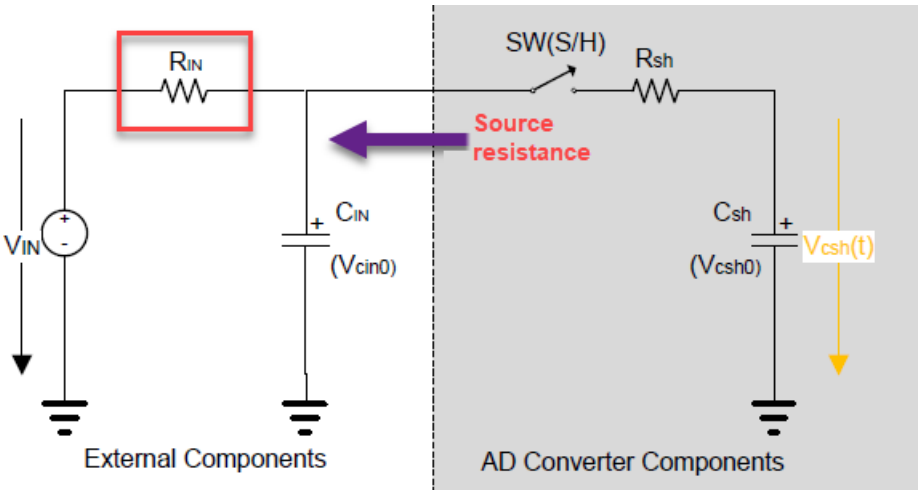

图2 - 模拟信号源电阻

如 图 2 所示，输入信号的采样是通过用电阻 Rsh 控制开关对内部电容器（Csh）充电来实现的。随着源电阻（R_{IN}）的增加，保持电容完全充电所需的时间将会增加。 如果采样时间小于电容器充电至稳定所需的时间，则 ADC 转换的数字值将小于实际值。

因此，必须采取预防措施以确保模拟输入信号源电阻在 ADC 规格范围内。在 S32K 器件的数据表中，该参数可以在源阻抗 (R_S) 找到。

温度影响

系统温度会对 ADC 精度产生很大影响，主要可以导致失调误差漂移和增益误差漂移。ADC参考电压也会随温度变化而变化。这些误差可以通过调整微控制器固件来补偿，例如监控内部带隙电压以验证参考电压未改变，或者在应用的温度范围内对系统进行描述以解释误差。

I/O 引脚串扰

由于引脚之间存在电容耦合，在当前进行 ADC 采样的模拟输入引脚附近切换 I/O 将给转换引入噪声，该串扰是由彼此靠近或交叉的 PCB 走线引起的。内部对数字信号和 I/O 进行切换同样会引入高频噪声。

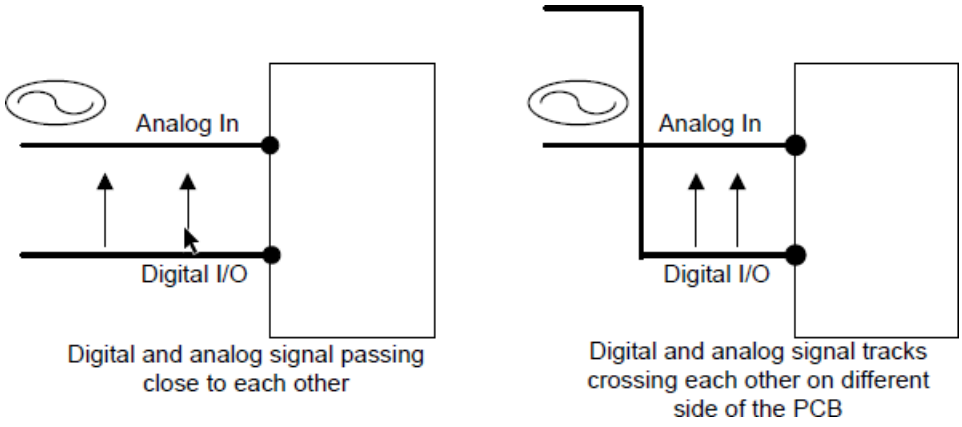

图3 - I/O 引脚串扰

2.3. S32K1xx ADC 规格

本节介绍了集成 S32K1xx 器件数据表中 SAR ADC 规范的参数。

ADC 时钟频率（fADCK）： SAR ADC 模块的输入转换时钟频率。该频率是决定 A/D 转换时间的主要因素。内部 ADC 逼近机制使用该时钟作为转换状态机中不同转换的基准时间。

ADC转换频率（fCONV）： 也称为“转换率”或“采样率”，这是将模拟信号转换为数字信号速度的衡量指标。对于更高的转换频率，可以在确定的时间窗口内采集更多的样本，而较低的转换频率意味着在相同的时间段内将获得更少的样本。

转化率主要取决于以下因素：

- ADC 时钟频率
- 是否启用硬件平均

- 样本数量
- 配置情况（单次或连续转换）

有关如何计算总转换时间的信息，请参阅设备参考手册。

微分非线性（DNL）

差分非线性误差是“编码宽度误差”，其中编码宽度是产生给定ADC转换值的输入电压范围V_{ADIN}。理想情况下，1LSB 的模拟输入电压变化应该会导致数字编码发生变化。因此，DNL 是实际编码宽度与 1LSB 的理想转换电压之间的差值。

请注意，DNL 是针对每个 ADC 转换编码单独测量的，与其他编码无关。

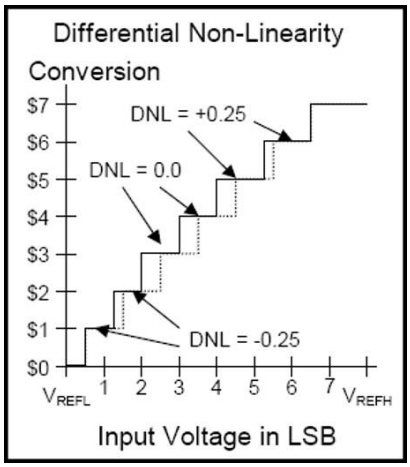

图4 - 微分非线性（DNL）

DNL误差有两个关键的品质因数：

- **缺失编码：**如果无限小的电压变化导致两个数字计数的结果发生变化，并且中间编码从未设置，则 ADC 会丢失编码。DNL 为 -1.0LSB 表示 ADC 缺少代码。
- **单调性：**如果 ADC 随着电压的增加不断增加转换结果（反之亦然），则它是单调的。对于较高的输入电压，非单调 ADC 可能会给出较低的转换结果，这也可能意味着相同的转换可能来自两个不同的电压范围。大于 1.0LSB 的 DNL 表示非单调性。

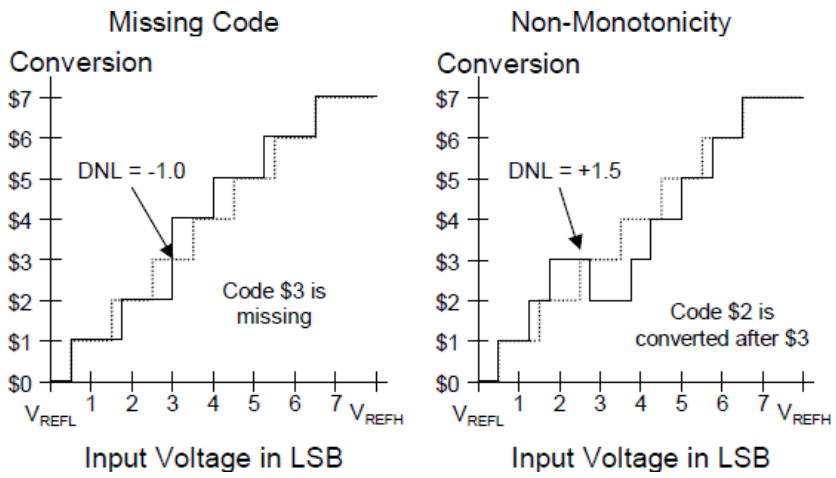

图5 - 缺失编码和非单调性

积分非线性（INL）

虽然与理想情况相比，任何给定的 ADC 编码都给出了 DNL，但积分非线性（INL）是从转换编码 1 到相关编码的所有 DNL 误差的累积效应。那么基本上 INL 是 DNL 的总和，可以用公式 3表示。

$$INL(x) = \sum_{i=1}^{(x-1)} DNL(i)$$

公式 3 – INL 方程

下图是基于单个 DNL 累积效应的 INL 表示。

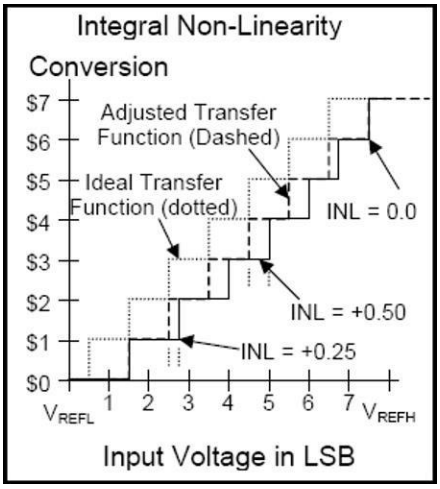

图6 - 积分非线性（INL）

总未调整的总误差（TUE）

TUE 是偏移、增益、线性度和量化误差的总和。这是一个关键参数，因为它决定了 ADC 的真实预期精度。对于任何给定的输入电压 V_ADIN，TUE 是实际获得的转换值与理想条件下的期望值之间的差值，用 LSB 表示。

专有名词“未调整”意味着 TUE 是通过原始转换数据测量的，未经过任何方式标准化来消除 ADC 固有误差。

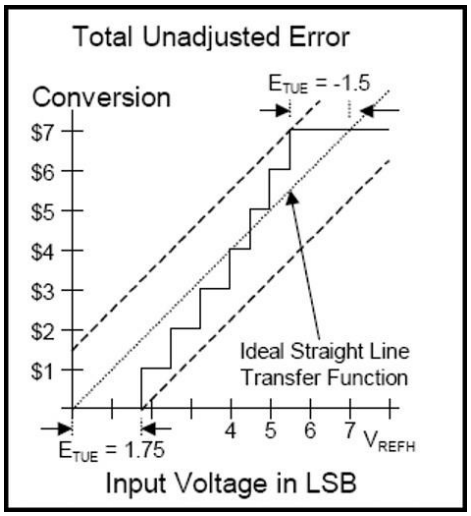

图7 - 总未调整的总误差（TUE）

DNL、INL和 TUE 表示在静态/直流输入上转换时的 ADC 误差。因此，这些误差代表 ADC 在静态/直流下的性能。

3. 提高准确性的最佳措施

本节介绍了提高 ADC 测量精度的通用建议和良好做法。

ADC 校准

S32K1xx 系列中的 SAR ADC 具有自校准机制，可调整内部采样电容器组，以补偿出厂时每个 IC 单元的电容变化。用户必须在每次上电复位后启动 ADC 的自校准，以获得数据表中规定的 ADC 精度。

校准可以运行一次，然后将校准寄存器的值保存在非易失性存储器中，从而可以在复位后恢复它们来避免后续校准。

以下是获得最佳校准的一些建议：

- 所有数字 IO 都应该是不工作的，不必要的模块须禁用。
- VREFH 应该在规定的范围内尽可能高和稳定，因为更高的 VREFH 意味着更大的 ADC 编码宽度。
- 有一个独立的VREFH引脚是最理想的。

- 如果应用中的 ADC 时钟将会快于 25MHz，ADC 自校准应在等于或小于 25MHz 的 ADC 时钟运行。或者，当应用中的 ADC 时钟设置为 25MHz 或更低时，建议在运行校准时使用相同的 ADC 频率。
- 硬件平均应设置为最多32个样本。
- POR 后应在室温下进行一次校准。

有关内部校准机制的更详细说明，请查阅链接中的文档。

参考电压和电源

由于 ADC 使用 VREF 或 VDDA 作为模拟参考电压，电源应该具有良好的线性、负载调节和温度漂移。因此，VREF 在不同负载下保持稳定是非常重要的。每当通过接通一部分电路来增加负载时，电流的增加不应导致电压降低。

如果电压在很宽的电流范围内保持稳定，则电源具有良好的负载调节能力。线性调节值越低，调节效果越好。

同样，负载调节值越低，电压输出的调节度和稳定性越好。也可以使用有高精度稳压器的VREF参考电压。

温度漂移是另一个需要考虑参考电压的重要因素，特别是在某些应用中，ADC 精度需要在全温度范围内规定。

利用带隙电压监测参考电压

要监控 VREF 变化，一个方法是使用内部带隙电压的 ADC 通道。带隙电压通道提供独立于参考电压或模拟电源电压的固定 1V 电压。

监测流程如下：

1- 触发带隙电压的 ADC 转换（通道 27）。
2- 使用以下等式计算实际 VREF：

$$VREF\,(mV) = \frac{(1000)(2^N)}{BG\_ADCresult}$$

公式 4 – VREF 计算

其中：

N = 以位为单位的 ADC 分辨率（8/10/12 位）

BG_ADCresult = 带隙电压通道的 ADC 转换结果

3- 对于应用中任意电压的计算，请参考公式 1 中的结果 VREF。

模拟源电阻匹配

正如ADC 概念、误差源和规格中所述，模拟源的阻抗在ADC精度中起着重要作用。考虑到该因素，希望具有尽可能低的源阻抗。用户应始终确保模拟信号源的阻抗在数据表中的ADC规格范围内。

阻抗匹配的常用方法是在模拟信号源和 ADC 输入引脚之间放置一个外部运算放大器。然而，增加外部运算放大器意味着设计 BOM 成本的增加。

如果测量源阻抗较高的信号，则在配置 ADC 时可能需要考虑以下因素：

- 较低的 ADC 时钟频率（f_{ADCK}）
- 更长的采样时间。S32K1xx 器件中的采样时间会随着 ADC 配置寄存器 2 (CFG2) 中 SMPLTS 字段的值的增加而增加。

有关如何设计外部 RC 采集电路和选择组件的更深入说明，请参阅应用笔记 AN4373。尽管该文档主要针对16位 SAR ADC 和其他 NXP 微控制器系列（例如 Kinetis），但 S32K1xx 中的 ADC 模块与其有着相同的基本架构，因此该理论也适用。

最小化 I/O 引脚串扰

通过在中间放置通畅的模拟接地线来屏蔽模拟信号，可以减少相邻 PCB 走线或 MCU 引脚之间串扰所产生的噪声。下图是这种屏蔽方法的典型用例：

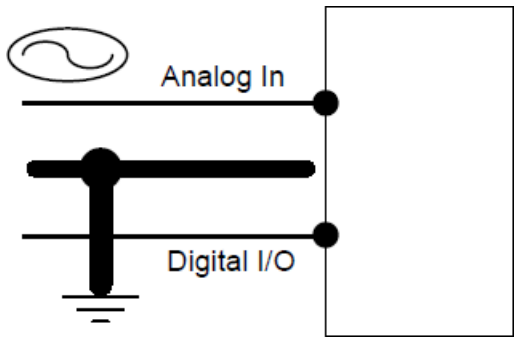

图8 - 推荐的信号间接地方法

4. ADC 触发模式示例

S32K1xx 系列中的 ADC 给可以启动转换的触发源提供了灵活的配置。本节介绍了为典型触发配置创建的示例中的 ADC 工作流程。

示例代码是基于 S32K144 EVB 板创建的，使用电位计作为模拟信号源，使用 PTB5（J2 接头中的引脚 5）作为 TRGMUX 示例的触发输入。

注意

本文档仅提供触发示例的高级概述。有关引脚、时钟、SIM、ADC、PDB 和 TRGMUX 模块的功能和配置设置的详细信息，请参阅参考手册。

4.1. 软件触发

示例代码参见附录 A。

软件触发是最简单的触发模式。它只是在写入 ADCx_SC1A 寄存器中的 ADCH 字段写入后启动单次或连续转换。需要注意的是，S32K1xx 中的 ADC 提供了多个状态和配置 1 寄存器（SC1A 到 SC1AF），但只有 SC1A 可用于软件触发模式。

在示例中，外部引脚 ADC0_SE12 用作于ADC的 输入。每次写入 ADC0_SC1A[ADCH] 都会触发一次新的转换。转换完成后，结果可在ADC0_RA寄存器中获得。

示例工作流程

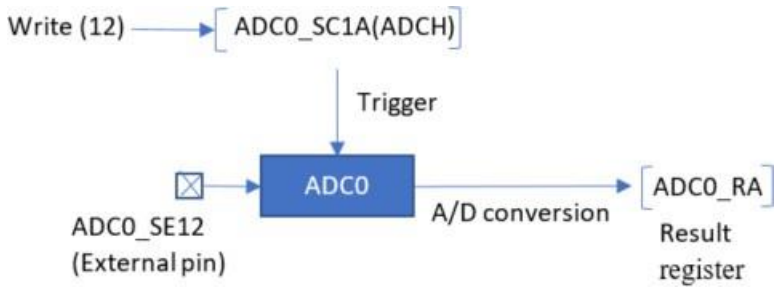

图9 – ADC 软件触发工作流程

4.2. PDB 触发器

示例代码参见附录 B。

PDB 触发方案是 ADC 的默认和推荐的硬件触发方法。该方法使用 PDB 定时器模块从同一通道或不同通道定期触发一个或多个 ADC 转换。当使用 PDB 作为触发器时，PDB 触发器可以通过两条路径到达 ADC 模块：

1) 直接路径：当触发 SC1n 寄存器编号 4 以后（对应寄存器 SC1E 到 SC1AF）的 ADC 转换时遵循此路径。
2) PDB/TRGMUX 多路复用触发路径：当触发 SC1n 寄存器 0 到 3（对应寄存器 SC1A、SC1B、SC1C 和 SC1D）的转换时，触发器通过触发器锁存垫。锁存垫提供锁定 ADC 触发请求的能力，然后由 ADC 处理，一次一个。

在该示例中，PDB 每秒触发一次 ADC0 的外部通道 12（ADC0_SE12 引脚）的新转换。PDB0/Channel 0 中的预触发 4 用于触发基于 ADC0_SC1E 寄存器的转换，因此使用 “直接路径” 模式。PDB 定时器本身最初由软件触发，然后持续运行。

下图展示了触发（蓝色虚线）和预触发（红色虚线）路径。

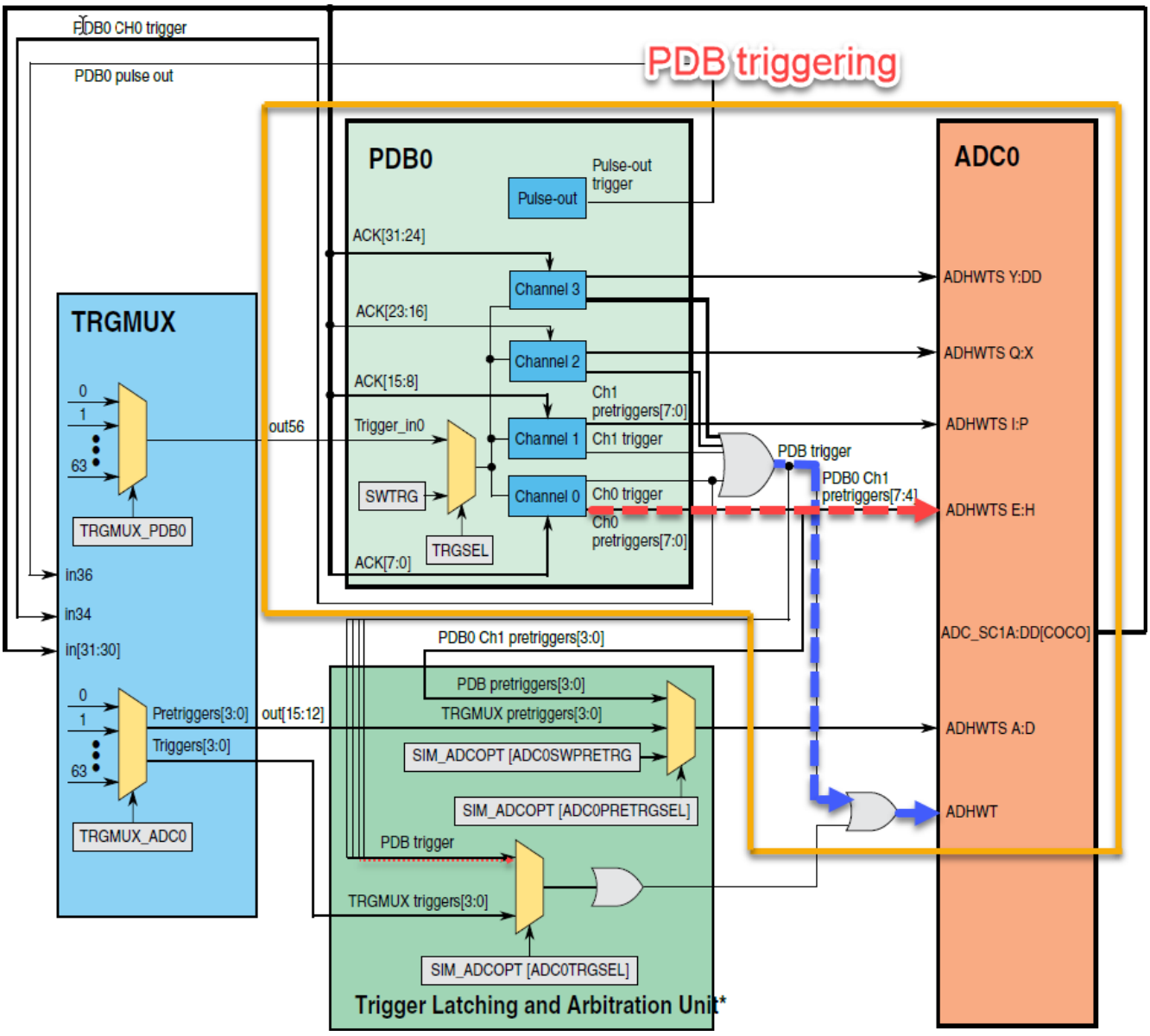

图10 - 直接路径模式下的 PDB 触发器

示例工作流程

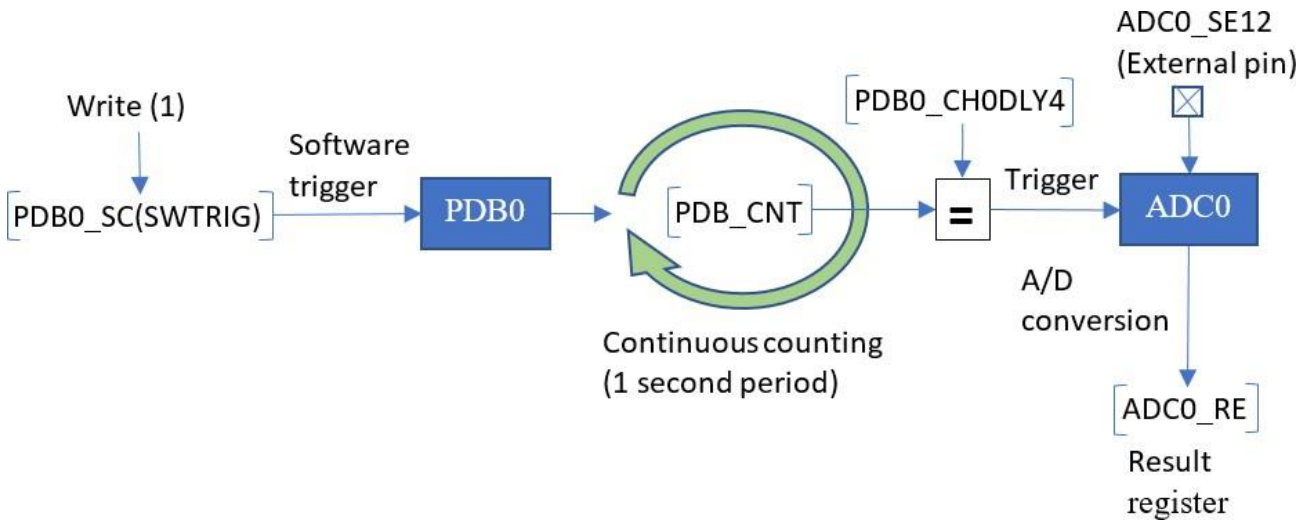

图11 - PDB 触发 ADC 样例的工作流程

4.3. 背靠背模式下的 PDB 触发器

示例代码参见附录 C。

背靠背是一种操作模式，其中 ADC 转换完成标志触发下一个 PDB 通道预触发和触发输出，一次一个。这在必须一个接一个地连续采样和转换多个 ADC 通道时特别有用。上一节中提到的 PDB 触发的两条路径（直接和复用）仍然适用于该模式。

在示例中，使用具有直接路径的 PDB 触发ADC0 的外部通道 12（ADC0_SE12 引脚），每秒连续转换四次。PDB0/Channel 0 的预触发 4 通过计数器匹配其对应的延迟寄存器（PDB0_CH0DLY4）来使能，而预触发 5/6/7 通过背靠背模式下与之相应的 ADC0 COCO 转换标志自动触发。因此使用的 ADC 通道需设置为 ADC0_SC1E 至 ADC0_SC1H。PDB 定时器最初由软件触发。

示例工作流程

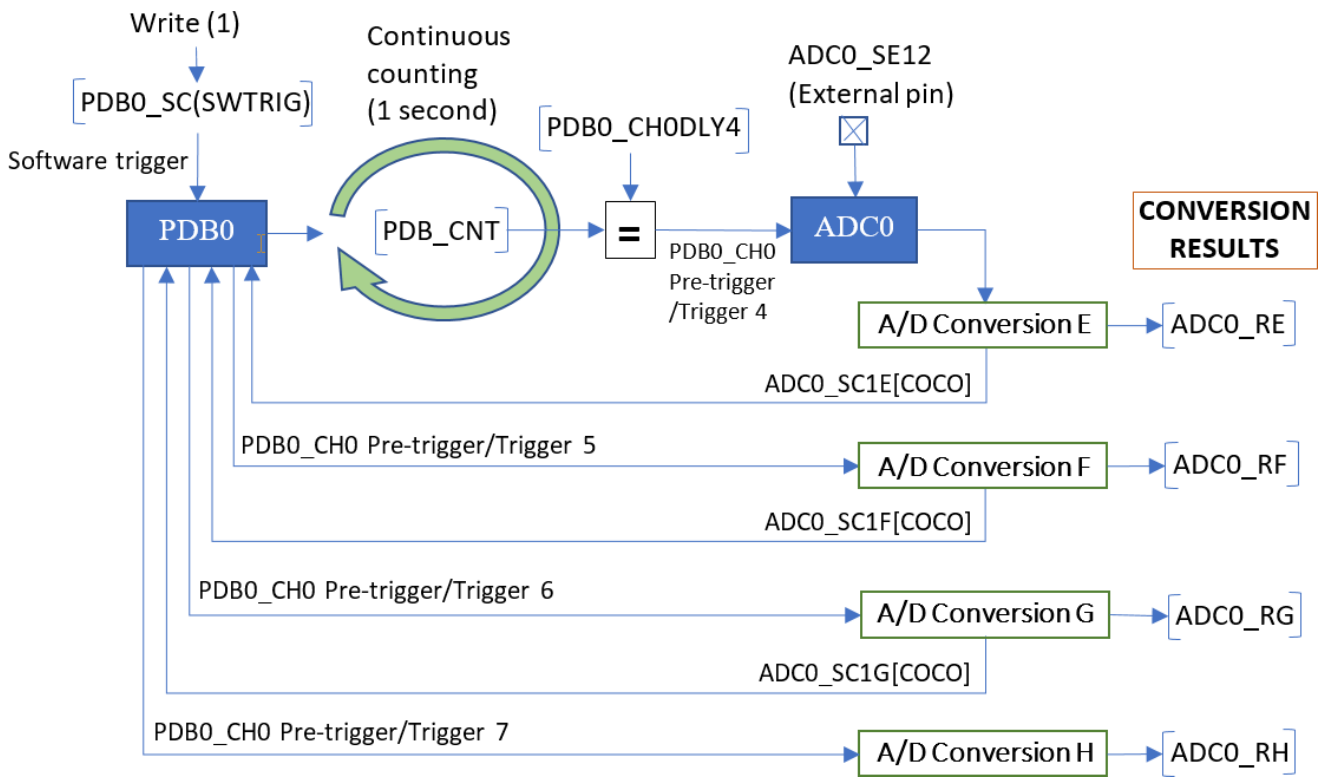

图12 - 背靠背模式下带 PDB 触发器的 ADC

4.4. TRGMUX 触发器

示例代码参见附录 D。

TRGMUX 是一个非常灵活的模块，用于将外设的触发输入与各种内部和/或外部触发信号（定时器模块、模拟模块标志位、外部引脚）互连。特别是对于 S32K1xx 中的 ADC，TRGMUX 可用于将AD转换与任何可用的触发信号同步。并且值得注意的是，在触发SC1n寄存器0到3[寄存器SC1A、SC1B、SC1C和SC1D]的ADC转换时，可以使用TRGMUX机制，且这种触发每次都要经过触发触发器锁存垫。

在该示例中，外部通道 12 (ADC0_SE12) 的单个 ADC0 转换由外部信号（在本例中为信号 TRGMUX_IN0）的每个上升沿触发。对于此用例，必须通过写入SIM_ADCOPT[ADC0SWPRETRG]寄存器字段向 ADC0 提供软件预触发。

下图显示了触发（蓝色虚线）和预触发（红色虚线）信号的整体路径：

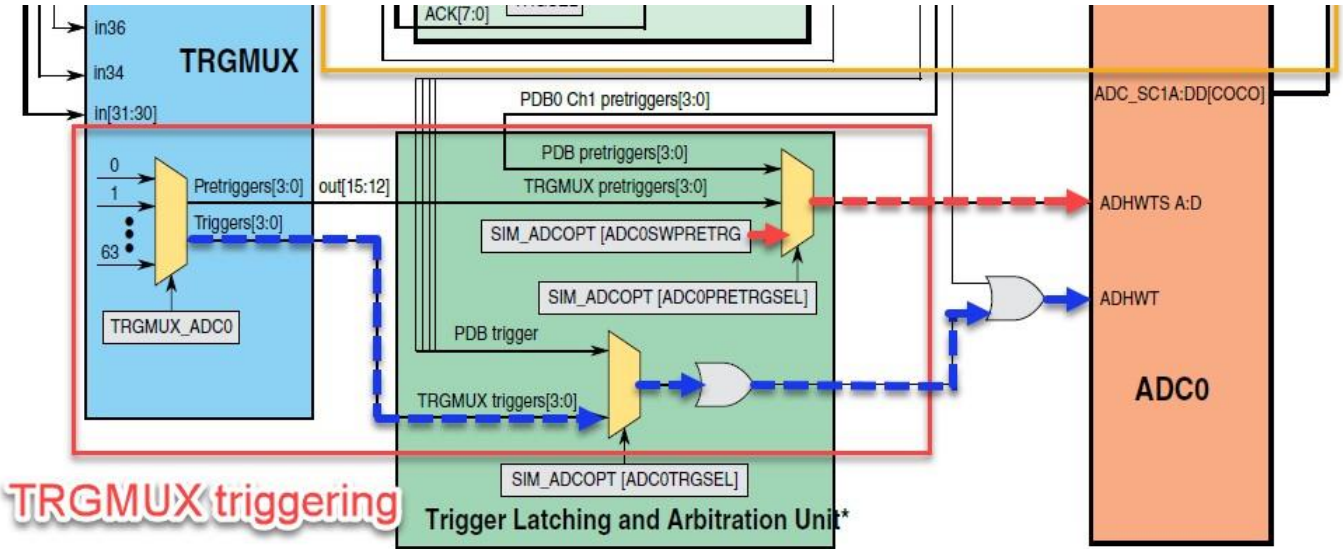

图13 - 通过 TRGMUX 触发 ADC

示例工作流程

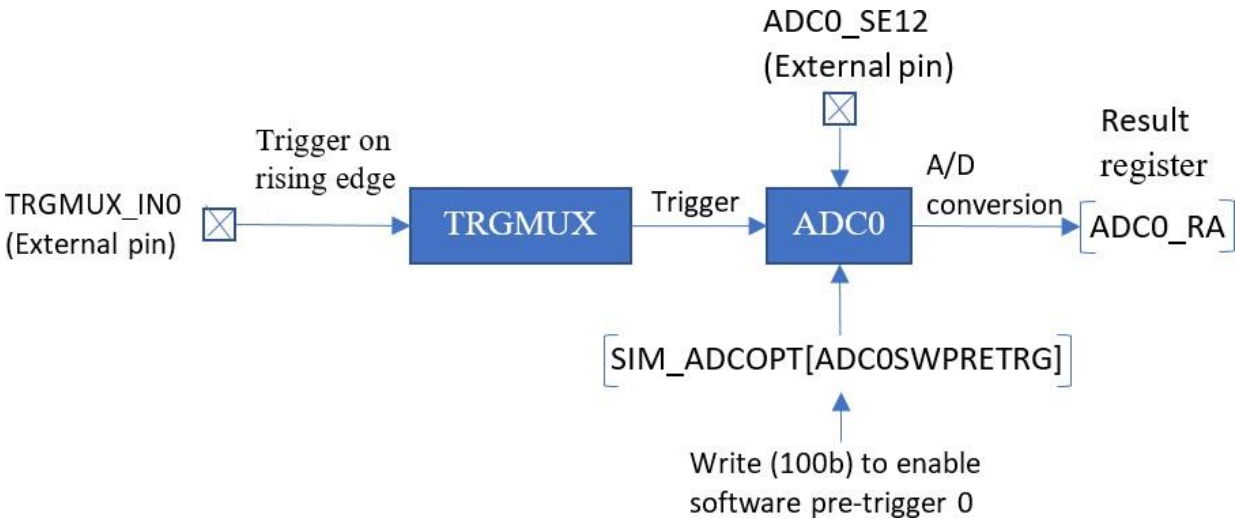

图14 - 带有 TRGMUX 触发示例工作流程的 ADC

5. 参考资料

- S32K1xx 数据表
- S32K1xx 参考手册
- S32K1xx 的 AN5426 硬件设计指南
- AN4373 SAR ADC 测量手册
- ADC 校准文件

附录 A. 示例代码：ADC 软件触发

```
#include "S32K144.h" /* include peripheral declarations S32K144 */

uint32_t ADC_RawResult;
uint16_t ADC_mVResult;

void WDOG_disable (void)
{
    WDOG->CNT=0xD928C520;        /* Unlock watchdog */
    WDOG->TOVAL=0x0000FFFF;      /* Maximum timeout value */
    WDOG->CS = 0x00002100;       /* Disable watchdog */
}

int main(void)
{
    WDOG_disable();       /* Disable Watchdog */

    SCG->FIRCDIV =  SCG_FIRCDIV_FIRCDIV2(4);  /* FIRCDIV2 = 4: FIRCDIV2 divide by 8 */

    /***** Calibrate ADC0 *****/
    PCC->PCCn[PCC_ADC0_INDEX] &=~ PCC_PCCn_CGC_MASK;    /* Disable clock to change PCS */
    PCC->PCCn[PCC_ADC0_INDEX] |= PCC_PCCn_PCS(3);       /* PCS = 3: Select FIRCDIV2 */
    PCC->PCCn[PCC_ADC0_INDEX] |= PCC_PCCn_CGC_MASK;     /* Enable bus clock in ADC */

    ADC0->SC3 = ADC_SC3_CAL_MASK       /* CAL = 1: Start calibration sequence */
              | ADC_SC3_AVGE_MASK      /* AVGE = 1: Enable hardware average */
              | ADC_SC3_AVGS(3);       /* AVGS = 11b: 32 samples averaged */

    /* Wait for completion */
    while(((ADC0->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);
```

```
    /*****************************************************
     * Initialize ADC0:
     * External channel 12, software trigger,
     * single conversion, 12-bit resolution
     *****************************************************/
    ADC0->SC1[0] = ADC_SC1_ADCH_MASK;         /* ADCH: Module disabled for conversions */

    ADC0->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1);    /* ADIV = 0: Divide ratio = 1 */
                                                         /* MODE = 1: 12-bit conversion */

    ADC0->CFG2 = ADC_CFG2_SMPLTS(12);  /* SMPLTS = 12: sample time is 13 ADC clks */

    ADC0->SC2 = ADC_SC2_ADTRG(0);      /* ADTRG = 0: SW trigger */

    ADC0->SC3 = 0x00000000;      /* ADCO = 0: One conversion performed */
                                 /* AVGE,AVGS = 0: HW average function disabled */
    for(;;)
    {
        /* Initiate new conversion by writing to ADC0_SC1A(ADCH) */
        ADC0->SC1[0] = ADC_SC1_ADCH(12);    /* ADCH = 12: External channel 12 as input */

        /* Wait for latest conversion to complete */
        while(((ADC0->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);
        ADC_RawResult = ADC0->R[0];    /* Read ADC Data Result A (ADC0_RA) */
        ADC_mVResult = (ADC_RawResult * 5000) / (1<<12);  /* Convert to mV (@VREFH = 5V) */
    }
    return 0;
}
```

附录 B. 示例代码：带有 PDB 触发器的 ADC

```
#include "S32K144.h" /* include peripheral declarations S32K144 */

uint32_t ADC_RawResult;
uint16_t ADC_mVResult;

void WDOG_disable (void)
{
    WDOG->CNT=0xD928C520;        /* Unlock watchdog */
    WDOG->TOVAL=0x0000FFFF;      /* Maximum timeout value */
    WDOG->CS = 0x00002100;       /* Disable watchdog */
}

int main(void)
{
    WDOG_disable();                    /* Disable Watchdog */

    SCG->FIRCDIV = SCG_FIRCDIV_FIRCDIV2(4);   /* FIRCDIV2 = 4: FIRCDIV2 divide by 8 */

    /**********************************************
     * Calibrate ADC0
     **********************************************/
    PCC->PCCn[PCC_ADC0_INDEX] &=~ PCC_PCCn_CGC_MASK;     /* Disable clock to change PCS */
    PCC->PCCn[PCC_ADC0_INDEX] |= PCC_PCCn_PCS(3);        /* PCS = 3: Select FIRCDIV2 */
    PCC->PCCn[PCC_ADC0_INDEX] |= PCC_PCCn_CGC_MASK;      /* Enable bus clock in ADC */

    ADC0->SC3 = ADC_SC3_CAL_MASK       /* CAL = 1: Start calibration sequence */
              | ADC_SC3_AVGE_MASK      /* AVGE = 1: Enable hardware average */
              | ADC_SC3_AVGS(3);       /* AVGS = 11b: 32 samples averaged */

    /* Wait for completion */
    while(((ADC0->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);

    /*****************************************************
     * Initialize ADC0:
     * External channel 12, hardware trigger,
     * single conversion, 12-bit resolution
     *
     * NOTE: ADC0->SC1[4] corresponds to ADC0_SC1E register
     *****************************************************/
    ADC0->SC1[4] = ADC_SC1_ADCH_MASK;         /* ADCH: Module disabled for conversions */

    ADC0->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1);   /* ADIV = 0: Divide ratio = 1 */
                                                        /* MODE = 1: 12-bit conversion */

    ADC0->CFG2 = ADC_CFG2_SMPLTS(12);  /* SMPLTS = 12: sample time is 13 ADC clks */

    ADC0->SC2 = ADC_SC2_ADTRG(1);      /* ADTRG = 1: HW trigger */
```

```
    ADC0->SC1[4] = ADC_SC1_ADCH(12);    /* ADCH = 12: External channel 12 as ADC0 input */

    ADC0->SC3 = 0x00000000;             /* ADCO = 0: One conversion performed */
                                        /* AVGE,AVGS = 0: HW average function disabled */

    /*********************************************
     * Initialize PDB0:
     * 1 second period, continuous mode
     * PDB0_CH0 pre-trigger 4 output enabled
     *********************************************/

     PCC->PCCn[PCC_PDB0_INDEX] |= PCC_PCCn_CGC_MASK;       /* Enable bus clock in PDB */

     PDB0->SC = PDB_SC_PRESCALER(6) /* PRESCALER = 6: clk divided by (64 x Mult factor) */
              | PDB_SC_TRGSEL(15)   /* TRGSEL = 15: Software trigger selected */
              | PDB_SC_MULT(3)     /* MULT = 3: Multiplication factor is 40 */
              | PDB_SC_CONT_MASK;   /* CONT = 1: Enable operation in continuous mode */

    /* PDB Period = (System Clock / (Prescaler x Mult factor)) / Modulus */
    /* PDB Period = (48 MHz / (64 x 40)) / 18750 */
    /* PDB Period = (18750 Hz) / (18750) = 1 Hz */
    PDB0->MOD = 18750;

    PDB0->CH[0].C1 = (PDB_C1_TOS(0x10) /* TOS = 10h: Pre-trigger 4 asserts with DLY match */
                   | PDB_C1_EN(0x10)); /* EN = 10h: Pre-trigger 4 enabled */

    PDB0->CH[0].DLY[4] = 9375;   /* Delay set to half the PDB period = 9375 */

    PDB0->SC |= PDB_SC_PDBEN_MASK | PDB_SC_LDOK_MASK;    /* Enable PDB. Load MOD and DLY */

    PDB0->SC |= PDB_SC_SWTRIG_MASK;                      /* Single initial PDB trigger */

    for(;;)
    {
        /* Wait for latest conversion to complete */
        while(((ADC0->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);

        ADC_RawResult = ADC0->R[4];    /* Read ADC Data Result E (ADC0_RE) */
        ADC_mVResult = (ADC_RawResult * 5000) / (1<<12);   /* Convert to mV (@VREFH = 5V) */
    }
    return 0;
}
```

附录 C. 示例代码：带有 PDB 和背靠背触发器的 ADC

```
#include "S32K144.h" /* include peripheral declarations S32K144 */

uint32_t ADC_Results[4];

void WDOG_disable (void)
{
    WDOG->CNT=0xD928C520;     /* Unlock watchdog */
    WDOG->TOVAL=0x0000FFFF;   /* Maximum timeout value */
    WDOG->CS = 0x00002100;    /* Disable watchdog */
}

int main(void)
{
    WDOG_disable();              /* Disable Watchdog*/

    SCG->FIRCDIV = SCG_FIRCDIV_FIRCDIV2(4);   /* FIRCDIV2 = 4: FIRCDIV2 divide by 8 */

    /**********************************************
     * Calibrate ADC0
     **********************************************/
    PCC->PCCn[PCC_ADC0_INDEX] &=~ PCC_PCCn_CGC_MASK;     /* Disable clock to change PCS */
    PCC->PCCn[PCC_ADC0_INDEX] |= PCC_PCCn_PCS(3);        /* PCS = 3: Select FIRCDIV2 */
    PCC->PCCn[PCC_ADC0_INDEX] |= PCC_PCCn_CGC_MASK;      /* Enable bus clock in ADC */

    ADC0->SC3 = ADC_SC3_CAL_MASK        /* CAL = 1: Start calibration sequence */
              | ADC_SC3_AVGE_MASK       /* AVGE = 1: Enable hardware average */
              | ADC_SC3_AVGS(3);        /* AVGS = 11b: 32 samples averaged */

    /* Wait for completion */
    while(((ADC0->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);

    /**********************************************
     * Initialize ADC0:
     * External channel 12, hardware trigger,
     * single conversion, 12-bit resolution
     *
     * NOTE: ADC0->SC1[4] corresponds to ADC0_SC1E register
     **********************************************/
    ADC0->SC1[4] = ADC_SC1_ADCH_MASK;  /* ADCH = 1F: Module is disabled for conversions*/
                                       /* AIEN = 0: Interrupts are disabled */
    ADC0->SC1[5] = ADC_SC1_ADCH_MASK;
    ADC0->SC1[6] = ADC_SC1_ADCH_MASK;
    ADC0->SC1[7] = ADC_SC1_ADCH_MASK;

    ADC0->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1);    /* ADIV = 0: Divide ratio = 1 */
                                                         /* MODE = 1: 12-bit conversion */

    ADC0->CFG2 = ADC_CFG2_SMPLTS(12);  /* SMPLTS = 12: sample time is 13 ADC clks */
```

```
    ADC0->SC2 = ADC_SC2_ADTRG(1); /* ADTRG = 1: HW trigger */

    ADC0->SC1[4] = ADC_SC1_ADCH(12); /* SC1E[ADCH] = 12: External channel 12 as input */
    ADC0->SC1[5] = ADC_SC1_ADCH(12); /* SC1F[ADCH] = 12: External channel 12 as input */
    ADC0->SC1[6] = ADC_SC1_ADCH(12); /* SC1G[ADCH] = 12: External channel 12 as input */
    ADC0->SC1[7] = ADC_SC1_ADCH(12); /* SC1H[ADCH] = 12: External channel 12 as input */

    ADC0->SC3 = 0x00000000;            /* ADCO = 0: One conversion performed */
                                       /* AVGE,AVGS = 0: HW average function disabled */

    /*********************************************
     * Initialize PDB0:
     * 1 second period, continuous mode
     * PDB0_CH0 pre-trigger outputs 4/5/6/7 enabled
     * Pre-trigger 4 asserted by channel delay register match
     * Back to back mode enabled for pre-triggers 5/6/7
     *********************************************/

    PCC->PCCn[PCC_PDB0_INDEX] |= PCC_PCCn_CGC_MASK;  /* Enable bus clock in PDB */ PDB0-

    >SC = PDB_SC_PRESCALER(6) /* PRESCALER = 6: clk divided by (64 x Mult factor) */
             | PDB_SC_TRGSEL(15)       /* TRGSEL = 15: Software trigger selected */
             | PDB_SC_MULT(3)          /* MULT = 3: Multiplication factor is 40 */
             | PDB_SC_CONT_MASK;       /* CONT = 1: Enable operation in continuous mode */

    /* PDB Period = (System Clock / (Prescaler x Mult factor)) / Modulus */
    /* PDB Period = (48 MHz / (64 x 40)) / 18750 */
    /* PDB Period = (18750 Hz) / (18750) = 1 Hz */
    PDB0->MOD = 18750;

    PDB0->CH[0].C1 = (PDB_C1_BB(0xE0)  /* BB = E0h: Back-to-back for pre-triggers 5/6/7 */
                   | PDB_C1_TOS(0x10)  /* TOS = 10h: Pre-trigger 4 asserts with DLY match */
                   | PDB_C1_EN(0xF0)); /* EN = F0h: Pre-triggers 4/5/6/7 enabled */

    PDB0->CH[0].DLY[4] = 9375;   /* Delay set to half the PDB period = 9375 */

    PDB0->SC |= PDB_SC_PDBEN_MASK | PDB_SC_LDOK_MASK;    /* Enable PDB. Load MOD and DLY */

    PDB0->SC |= PDB_SC_SWTRIG_MASK;                     /* Single initial PDB trigger */

    for(;;)
    {
        /* Wait for last conversion in the sequence to complete (ADC0_SC1H) */
        while(((ADC0->SC1[7] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);

        ADC_Results[0] = ADC0->R[4];   /* Read ADC Data Results 4-7 (ADC0_RE to ADC0_H) */
        ADC_Results[1] = ADC0->R[5];
        ADC_Results[2] = ADC0->R[6];
        ADC_Results[3] = ADC0->R[7];
    }

    return 0;
}
```

附录 D. 示例代码：带有 TRGMUX 触发器的 ADC

```
#include "S32K144.h" /* include peripheral declarations S32K144 */

uint32_t ADC_RawResult;
uint16_t ADC_mVResult;

void WDOG_disable (void)
{
    WDOG->CNT=0xD928C520;     /* Unlock watchdog */
    WDOG->TOVAL=0x0000FFFF;   /* Maximum timeout value */
    WDOG->CS = 0x00002100;    /* Disable watchdog */
}

int main(void)
{
    WDOG_disable();              /*!Disable Watchdog*/

    SCG->FIRCDIV =  SCG_FIRCDIV_FIRCDIV2(4);  /* FIRCDIV2 = 4: FIRCDIV2 divide by 8 */

    /**********************************************
     * Configure pin PTB5 as TRGMUX_IN0
     **********************************************/
    PCC->PCCn[PCC_PORTB_INDEX] = PCC_PCCn_CGC_MASK; /* Enable clock gate for PORTB */
    PORTB->PCR[5] = PORT_PCR_MUX(6);                /* Mux = 6: PTB5 as TRGMUX_IN0 */

    /* Select TRGMUX_IN0 as ADC0 Trigger Mux input source 0 */
    TRGMUX->TRGMUXn[TRGMUX_ADC0_INDEX] = TRGMUX_TRGMUXn_SEL0(2U);

    /**********************************************
     * Calibrate ADC0
     **********************************************/
    PCC->PCCn[PCC_ADC0_INDEX] &=~ PCC_PCCn_CGC_MASK;/* Disable clock to change PCS */
    PCC->PCCn[PCC_ADC0_INDEX] |= PCC_PCCn_PCS(3);    /* PCS = 3: Select FIRCDIV2 */
    PCC->PCCn[PCC_ADC0_INDEX] |= PCC_PCCn_CGC_MASK; /* Enable bus clock in ADC */

    ADC0->SC3 = ADC_SC3_CAL_MASK        /* CAL = 1: Start calibration sequence */
              | ADC_SC3_AVGE_MASK       /* AVGE = 1: Enable hardware average */
              | ADC_SC3_AVGS(3);        /* AVGS = 11b: 32 samples averaged */

    /* Wait for completion */
    while(((ADC0->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);

    /**********************************************
     * Initialize ADC0:
     * External channel 12, hardware trigger,
     * single conversion, 12-bit resolution
     **********************************************/
    ADC0->SC1[0] = ADC_SC1_ADCH_MASK;  /* ADCH: Module disabled for conversions */

    ADC0->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1);   /* ADIV = 0: Divide ratio = 1 */
                                                        /* MODE = 1: 12-bit conversion */
```

```
    ADC0->CFG2 = ADC_CFG2_SMPLTS(12);   /* SMPLTS = 12: sample time is 13 ADC clks */

    ADC0->SC2 = ADC_SC2_ADTRG(1);       /* ADTRG = 1: HW trigger */

    ADC0->SC1[0] = ADC_SC1_ADCH(12);    /* ADCH = 12: External channel 12 as input */

    ADC0->SC3 = 0x00000000;             /* ADCO = 0: One conversion performed */
                                        /* AVGE,AVGS = 0: HW average function disabled */

    /*********************************************
     * SIM Configurations for ADC triggering:
     * Pre-trigger source: Software pre-trigger
     * Trigger select: TRGMUX output
     *********************************************/
    SIM->ADCOPT = SIM_ADCOPT_ADC0PRETRGSEL(2) /* ADC0PRETRGSEL = 10b: Software pretrigger */
                | SIM_ADCOPT_ADC0SWPRETRG(4) /* ADC0SWPRETRG = 100b: SW Pre-trigger 0 */
                | SIM_ADCOPT_ADC0TRGSEL(1);   /* ADC0TRGSEL = 1: TRGMUX output as trigger */

    for(;;)
    {
        /* Wait for latest conversion to complete */
        while(((ADC0->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);
        ADC_RawResult = ADC0->R[0];           /* Read ADC Data Result 0 */
        ADC_mVResult = (ADC_RawResult * 5000) / (1<<12);  /* Convert to mV (@VREFH = 5V) */
      }
      return 0;
}
```

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: AN12217
Rev. 0
08/2018