NXP Semiconductors
Application Notes

Document Number: AN12217
Rev. 0, 08/2018

S32K1xx ADC guidelines, spec and configuration

by: NXP Semiconductors

1. Introduction

NXP S32K1xx automotive microcontrollers feature a 12-bit successive approximation Analog-to-Digital converter (SAR ADC) to be used in the acquisition and digitalization of analog input signals.

This application note presents information on the next basic topics to get the most benefit from the use of the ADC module:

- Understanding the ADC common terminology, sources of error and specification.
- Best practices to increase measurement's accuracy.
- Common triggering configuration examples for the S32K1xx family.

Contents

2. ADC concepts, error sources and specification

This section provides an explanation of the concepts and terminology used to characterize an ADC and the potential sources of error, as well as the specification parameters found in the S32K1xx family datasheet.

2.1. ADC basic concepts

Resolution: The number of bits in the ADC digital output representing an analog input signal. For S32K1xx devices the resolution can be configured to 8, 10 or 12 bits.

Reference Voltage: The ADC requires a reference voltage used to create a successive approximation comparison with the analog input is compared to produce a digital output. The digital output is the ratio of the analog input with respect to this reference voltage.

VREF = VREFH – VREFL

Where:

VREFH = High reference voltage

VREFL = Low reference voltage

ADC output formula: The conversion equation of ADC is used to calculate the digital output corresponding to a particular analog input voltage. This equation assumes an ideal A/D conversion with no introduced errors.

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

Equation 1 - ADC converter equation

Where:

ADC result = The digital output value resulting from the conversion

N = ADC resolution

VREF = Reference voltage

Vin = Analog input voltage

Least Significant Bits (LSB): A least significant bit (LSB) is a unit of voltage equal to the smallest resolution of the ADC, i.e. the smallest incremental voltage that causes a change in the digital output.

The LSB is equal to the reference voltage divided by the maximum count of the ADC:

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

Equation 2 - LSB equation

N = ADC resolution. For S32K1xx this can be 8/10/12 bits.

VREF = Analog reference voltage.

ADC Actual Transfer Function: The ADC converts an input voltage to a corresponding digital code. The curve describing this behavior is the *actual transfer function* and includes all the errors inherent to the ADC module itself.

ADC Ideal Transfer Function: The *ideal transfer function* represents the behavior of the ADC assuming it is perfectly linear, or that a given change in input voltage will create the same change in conversion code regardless of the input's initial level. The way the ideal transfer function is divided into steps depends on the method of quantization the ADC uses. The two possible methods are:

- **Uncompensated Quantization:** The first step is taken at 1 LSB, with each successive step taken at 1LSB intervals and the last step taken at VREFH – 1LSB.
- **½LSB Compensated Quantization:** The first step is taken at ½LSB, with each successive step taken at 1LSB intervals and the last step taken at VREFH – 1½LSB.

The figure below shows the ideal transfer function graphs for uncompensated and ½LSB compensated methods, for a 3 bit resolution and VREF = 8 V.

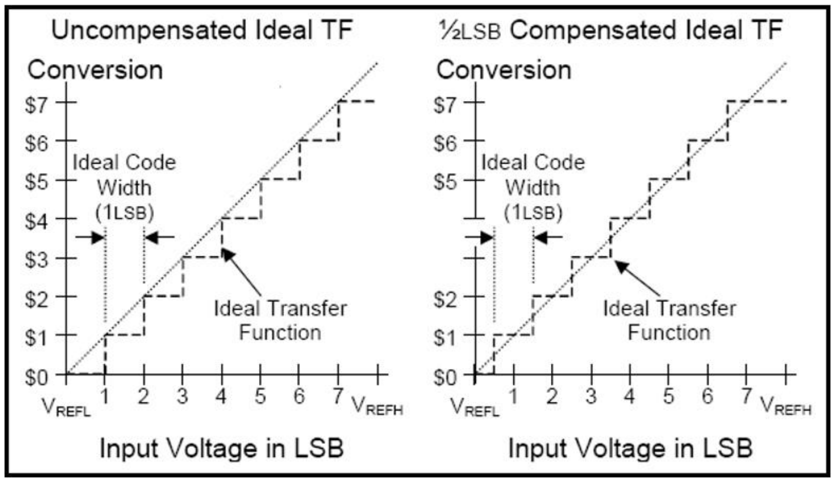

Figure 1 - Ideal transfer functions

2.2. Sources of error in ADC measurements

This section presents some typical factors that prevent the ADC from performing accurate A/D measurements.

Reference voltage noise

The ADC output is directly proportional to the analog input voltage and the reference voltage. An unstable reference voltage (e.g. caused by noise in the supply rail) will cause changes in the converted digital outputs.

Example:

- For a reference voltage of 5 V and a 1 V input voltage, using Equation 1 the ADC result for a 12-bit resolution is 819.
- With a 50 mV increase in the absolute reference voltage (i.e. VREF = 5.05 V), the new converted value for the same 1 V input voltage is now 811.
- The resulting reference voltage noise error is 811-819 = - 8 LSB.

Analog input signal noise

Small but high-frequency variations in the analog input signal can potentially cause big conversion errors during ADC sampling time. Noise can be induced by electromagnetic emissions from surrounding electrical devices (EMI noise). Therefore, the conversion accuracy is negatively impacted.

If the noise present in the input signal is higher than 1LSB, this effectively reduces the number of reliable bits in the conversion result, since the least significant bits are constantly changing due to the signal variations.

Analog-signal source resistance

The impedance of the analog signal source or series resistance (R_{IN}) between the source and the input pin causes a voltage drop across it because of the current flowing into the pin. It can be understood as the resistance observed "looking out" of the ADC into the source driving the input signal to be sampled.

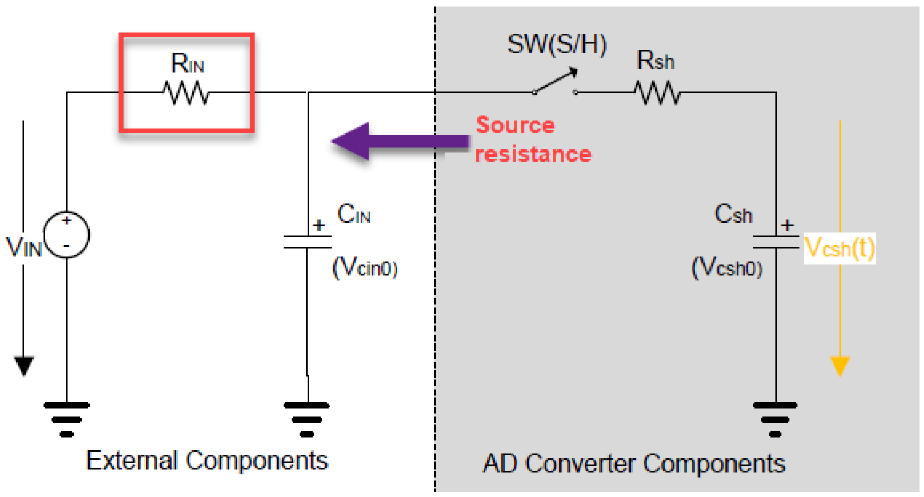

Figure 2 - Analog signal source resistance

As shown in Figure 2, the sampling of the input signal is achieved by charging an internal capacitor (Csh), controlling a switch with resistance Rsh. With the addition of source resistance (R_{IN}), the time required to fully charge the hold capacitor increases. If the sampling time is less than the time required for the capacitor charging to settle, then the digital value converted by the ADC is less than the real value.

For this reason, precautions must be taken to ensure that the analog input signal source resistance is within ADC specification. In the datasheet for S32K devices, this parameter can be found as Source Impedance (R_S).

Temperature influence

The temperature of the system can have a major influence on ADC accuracy, mainly causing offset error drift and gain error drift. The ADC reference voltage also changes with temperature change. These errors can be compensated with adjustments to the microcontroller firmware, such as monitoring the internal bandgap voltage to verify that the reference voltage has not changed or characterizing the system over the application’s temperature range to account for the errors.

I/O pin crosstalk

Switching of I/Os in the vicinity of the analog input pin currently being sampled by the ADC will introduce noise to the conversion due to the capacitive coupling between pins. Crosstalk is caused by PCB tracks that run close to each other or that cross each other. Internally switching digital signals and I/Os introduces high frequency noise.

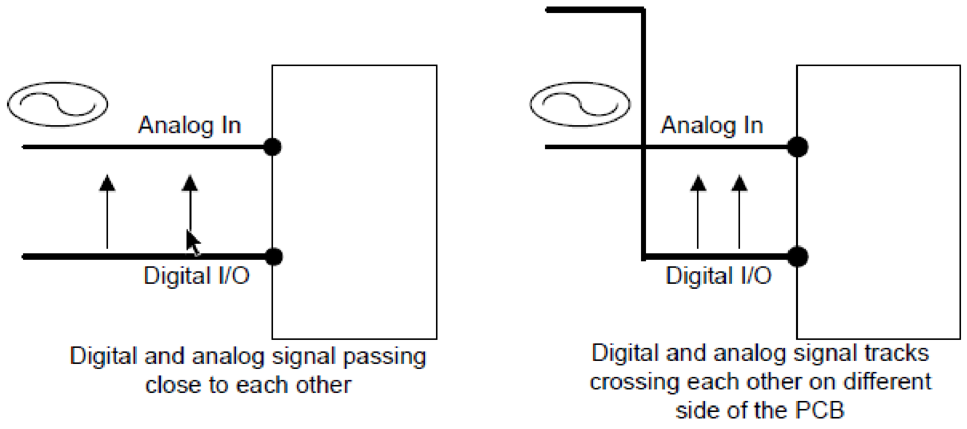
Analog In
Digital I/O
Digital and analog signal passing close to each other
Analog In
Digital I/O
Digital and analog signal tracks crossing each other on different side of the PCB

Figure 3 - I/O pin crosstalk

2.3. S32K1xx ADC specifications

This section explains the parameters that integrate the specification of SAR ADC found in datasheet for S32K1xx devices.

ADC clock frequency (f_{ADCK}): The frequency of the input conversion clock for the SAR ADC module. This frequency is the main factor to determine conversion time for a given A/D conversion. The internal ADC approximation mechanism uses this clock as the base time for the different transitions in the conversion state machine.

ADC conversion frequency (f_{CONV}): Also known as “conversion rate” or “sampling rate”, this is a measure of the speed to convert an analog signal to a digital result. For a higher conversion frequency, more samples can be taken in a determined time window, while a lower conversion frequency means that less samples will be acquired for the same period of time.

The conversion rate mainly depends on the next factors:

- ADC clock frequency
- Hardware averaging enabled or disabled

- Number of samples
- Configuration (single or continuous conversions)

Refer to the device Reference Manual on how to calculate total conversion times.

Differential Non-Linearity (DNL)

The differential non-linearity error is a “code width error”, where code width is the range of input voltages, V_{ADIN}, that result in a given ADC conversion value. Ideally, an analog input voltage change of 1LSB should cause a change in the digital code. Hence, DNL is the difference between the actual code width and the ideal transition voltage of 1LSB.

Please notice that DNL is measured individually for each ADC conversion code independent of other codes.

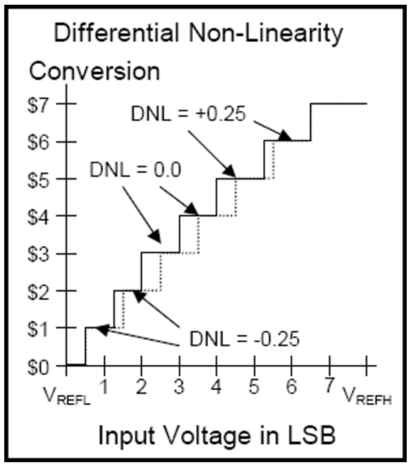

Figure 4 - Differential Non-Linearity (DNL)

There are two critical figures of merit derived from the DNL error:

- **Missing codes:** The ADC has missing codes if an infinitesimally small change in voltage causes a change in result of two digital counts, with the intermediate code never being set. A DNL of -1.0 LSB indicates the ADC has missing codes.
- **Monotonicity:** An ADC is monotonic if it continually increases conversion result with an increasing voltage (and vice versa). A non-monotonic ADC may give a lower conversion result for a higher input voltage, which may also mean that the same conversion may result from two separate voltage ranges. A DNL greater than 1.0 LSB indicates non-monotonicity.

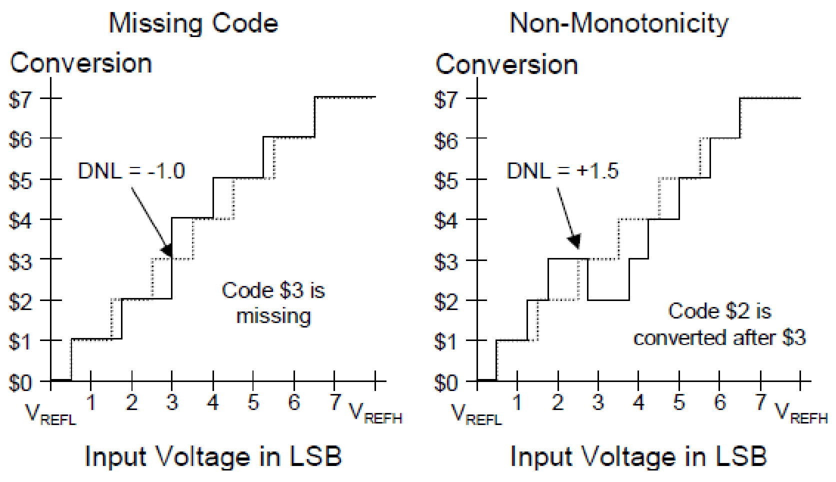

Figure 5 - Missing codes and Non-Monotonicity

Integral Non-Linearity(INL)

While DNL is given for any given ADC code compared to ideal, Integral Non-Linearity (INL) is the cumulative effect of all the DNL errors from conversion code 1 up to the code of interest. Then basically INL is a sum of DNLs which can be expressed by Equation 3.

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

Equation 3 - INL equation

The figure below shows a representation of INL based on the cumulative effect of the individual DNLs.

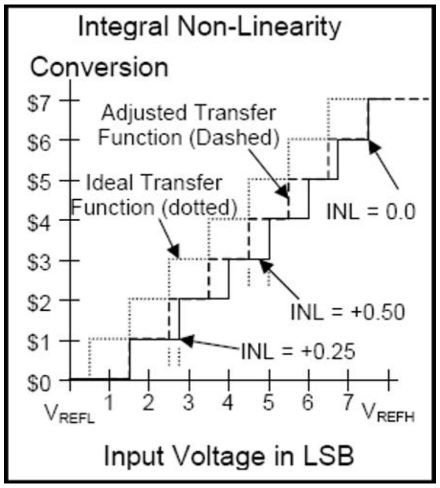

Figure 6 - Integral Non-Linearity (INL)

Total Unadjusted Error (TUE)

TUE is the summation of offset, gain, linearity, and quantization errors. This is a key parameter since it provides the real expected accuracy of the ADC. For any given input voltage, V_{ADIN}, TUE is the difference in the conversion value obtained compared to the ideal expectation, expressed in LSBs.

The term "unadjusted" means TUE is measured via raw conversion data, not normalized in any way to remove ADC inherent errors.

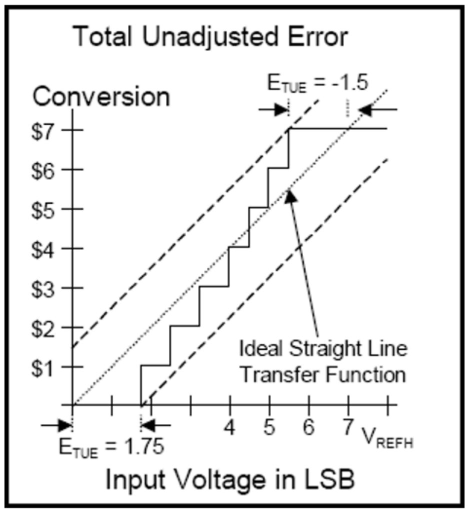

Figure 7 - Total Unadjusted Error (TUE)

DNL, INL and TUE represent the ADC errors when converting on a static/DC input. Hence these errors represent the ADC Static/DC performance.

3. Best practices to increase accuracy

This section includes general recommendations and good practices to increase the accuracy of ADC measurements.

ADC calibration

The SAR ADC in S32K1xx family has a self-calibration mechanism which adjusts the internal sampling capacitor banks aiming to compensate for capacitance variations that come out of the factory for each IC unit. It is mandatory for the user to launch the self-calibration of the ADC after each Power On Reset to obtain the ADC accuracy specified in the datasheet.

Calibration can be run once, then save the calibration registers values in non-volatile memory to restore them after reset, hence avoiding sub-sequent calibrations.

Below are some recommendations to obtain the best possible calibration:

- All digital IO should be silent and unnecessary modules should be disabled.
- VREFH should be as stable and as high as possible within spec, since higher VREFH means larger ADC code widths.
- An isolated VREFH pin would be ideal.

- When the ADC clock in the application will be faster than 25 MHz, the ADC self-calibration should be run with an ADC clock equal or less than 25 MHz otherwise, when the ADC clock in the application is set to 25 MHz or less, it is recommended to use the same ADC frequency when running the calibration.
- Hardware averaging should be set to the maximum 32 samples.
- Calibration should be done once at room temperature after POR.

For a more detailed description of the internal calibration mechanism please revise the document in the link.

Reference voltage and power supply

The power supply should have a good line, load regulation, and temperature drift since the ADC uses VREF or VDDA as the analog reference. Thus, it is essential for VREF to remain stable at different loads. Whenever the load is increased by switching on a part of the circuit, the increase in current should not cause the voltage to decrease.

If the voltage remains stable over a wide current range, the power supply has good load regulation. The lower the line regulation value, the better the regulation.

Similarly, the lower the load regulation value, the better the regulation and the stability of the voltage output. It is also possible to use a reference voltage for VREF with a high precision regulator.

Temperature drift is another important factor to consider voltage reference, especially in some applications, the ADC accuracy is specified within full temp range.

Using bandgap to monitor reference voltage

To monitor VREF changes an option is to use the internal bandgap ADC channel. The bandgap channel delivers a fixed 1 V voltage independently from the reference voltage or analog supply voltage.

The procedure is as follows:

1- Trigger an ADC conversion for the bandgap (channel 27).

2- Calculate the actual VREF using the following equation:

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

Equation 4 - VREF calculation

Where:

N = ADC resolution in bits (8/10/12 bits)

BG_ADCresult = The ADC conversion result for the bandgap channel

3- Consider the resulting VREF in Equation 1 for any voltage calculations in the application.

Analog source resistance match

As described in ADC concepts, error sources and specification, the analog source resistance plays an important role in ADC accuracy. For this reason, it is desirable to have a source resistance as low as possible. User should always ensure that the analog signal source resistance is within ADC specification, expressed in the datasheet.

A common approach for impedance matching is to place an external operational amplifier between the analog signal source and the ADC input pin. However, the added external Op-Amp means an increase in the cost of the design BOM.

If measuring a signal with high source resistance the next considerations might be taken when configuring the ADC:

- Lower ADC clock frequencies (f_{ADCK})
- Longer sample times. The sampling time in S32K1xx devices can be increased with a higher value of the SMPLTS field in the ADC Configuration Register 2 (CFG2).

For a deeper explanation on how to design the external RC acquisition circuit and selection of components, see application note AN4373. Although the document refers to a 16-bit SAR ADC and other NXP microcontroller families such as Kinetis, the ADC module in S32K1xx shares the same basic architecture, so the theory is also applicable.

Minimizing I/O pin crosstalk

The noise generated by crosstalk between adjacent PCB tracks or MCU pins can be reduced by shielding the analog signal by placing clear analog ground tracks in the middle. The figure below is a representation of such shielding approach:

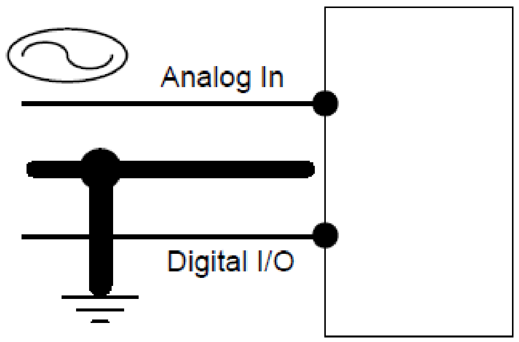

Figure 8 - Recommended grounding between signals

4. ADC triggering mode examples

The ADC in S32K1xx family provides a flexible configuration in terms of the possible trigger sources to initiate conversions. This section provides a description of the ADC working flow in examples created for the typical triggering configurations.

The example codes were created based on the S32K144 EVB board, using the potentiometer as the source of the analog signal and the PTB5 (pin 5 in J2 header) as trigger input for the TRGMUX example.

NOTE

This document only presents a high-level overview of the triggering examples. For detailed information of the functionality and configuration settings for the Pins, Clocks, SIM, ADC, PDB and TRGMUX modules, please refer to the Reference Manual.

4.1. Software trigger

For the example code refer to Appendix A.

Software trigger is the simplest of the trigger modes. It simply starts a single or continuous conversions after a write to the ADCH field in the ADCx_SC1A register. It is important to notice that the ADC in S32K1xx provides several Status and Configuration 1 registers (SC1A up to SC1AF), but only SC1A can be used for software trigger mode.

In the example, external pin ADC0_SE12 is used as the ADC input. A new conversion is triggered with each write to ADC0_SC1A[ADCH]. The result is available in the ADC0_RA register once the conversion is complete.

Example working flow

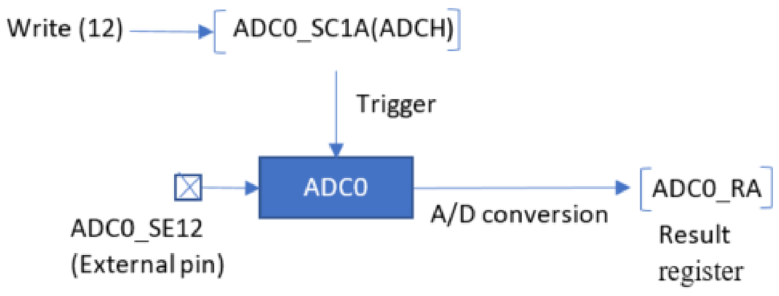

Figure 9 - ADC software trigger working flow

4.2. PDB trigger

For the example code refer to Appendix B.

PDB triggering scheme is the default and suggested hardware trigger method for the ADC. This method uses the PDB timer module to trigger one or more ADC conversions periodically, either from the same channel or from different channels. When using the PDB as trigger, there are two paths that can be followed by the PDB trigger to reach the ADC module:

1) Direct path: This path is followed when triggering ADC conversions for SC1n register number 4 onward (corresponding to registers SC1E up to SC1AF).
2) PDB/TRGMUX multiplexed triggering path: When triggering conversions for SC1n registers 0 to 3 (corresponding to registers SC1A, SC1B, SC1C and SC1D), the trigger goes through the trigger latching gasket. The latching gasket provides the capability to latch ADC trigger requests, which are then processed by the ADC one at a time.

In the example, a new conversion for external channel 12 of ADC0 (ADC0_SE12 pin) is triggered each second by the PDB. The pre-trigger 4 in PDB0/Channel 0 is used to trigger conversions based on ADC0_SC1E register, thus using the “Direct path” mode. The PDB timer itself is initially triggered by software and then runs continually.

The figure below shows the trigger (blue dotted line) and pre-trigger (red dotted line) paths.

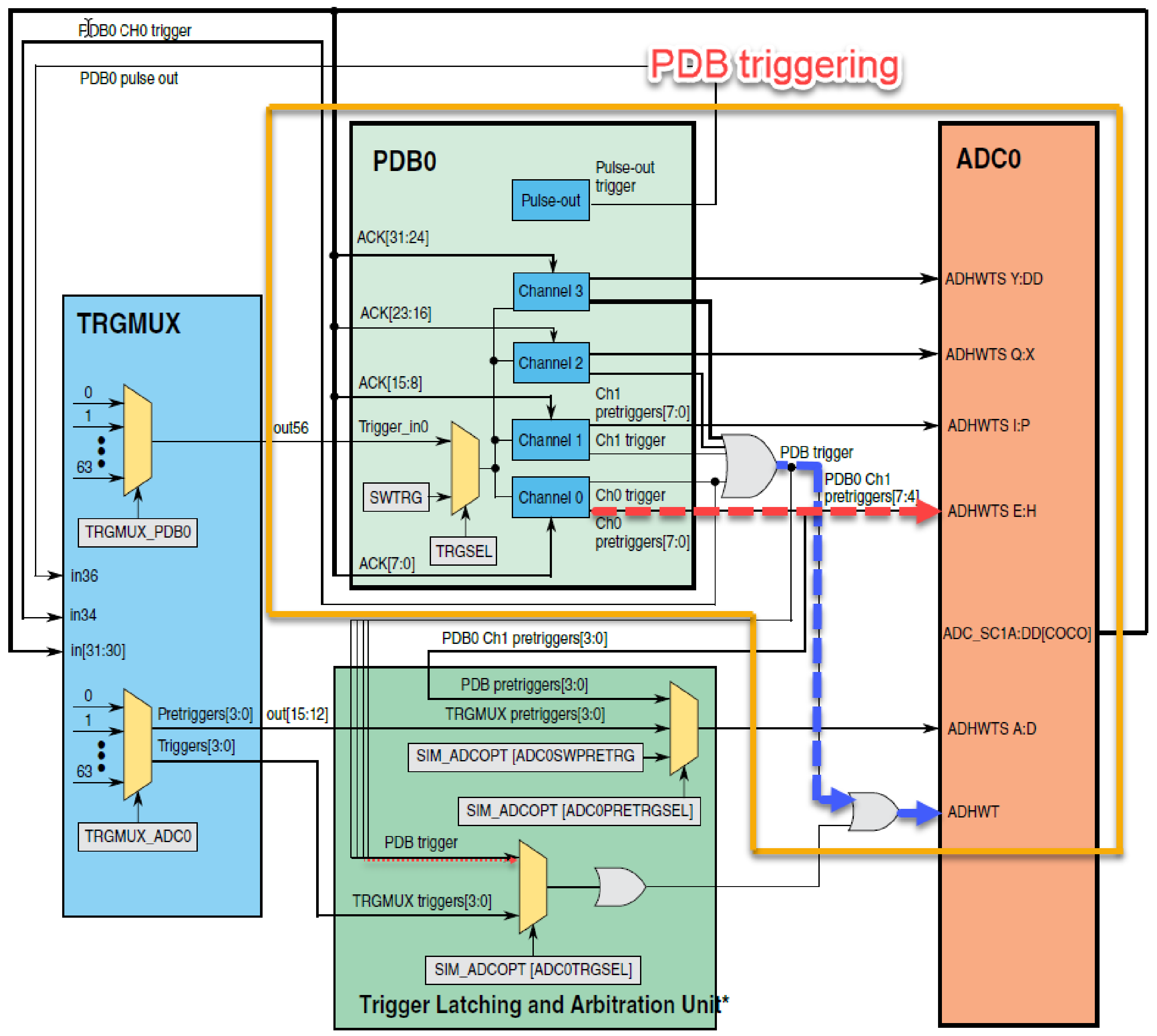

Figure 10 - PDB trigger in direct path mode

Example working flow

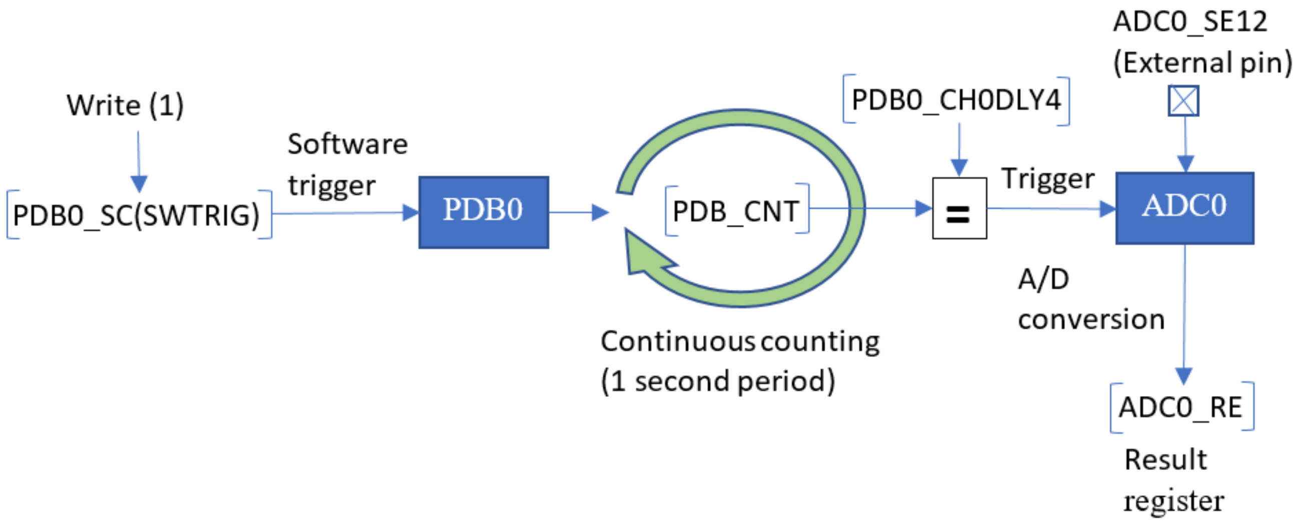

Figure 11 - ADC with PDB trigger example working flow

4.3. PDB trigger in back-to-back mode

For the example code refer to Appendix C.

Back-to-back is a mode of operation in which ADC conversion complete flags trigger the next PDB channels pre-trigger and trigger outputs, one at a time. This is especially useful whenever several ADC channels must be sampled and converted in a row, one after each other. The two paths for PDB trigger (direct and multiplexed) mentioned in previous section still apply for this mode.

In the example, external channel 12 of ADC0 (ADC0_SE12 pin) is converted four times in a row each second, using PDB with direct path. The pre-trigger 4 of PDB0/Channel 0 is enabled with a counter match to its corresponding delay register (PDB0_CH0DLY4), while the pre-triggers 5/6/7 are automatically triggered in back-to-back mode with the corresponding ADC0 COCO conversion flags. The ADC channel settings used are therefore ADC0_SC1E to ADC0_SC1H. The PDB timer is initially triggered by software.

Example working flow

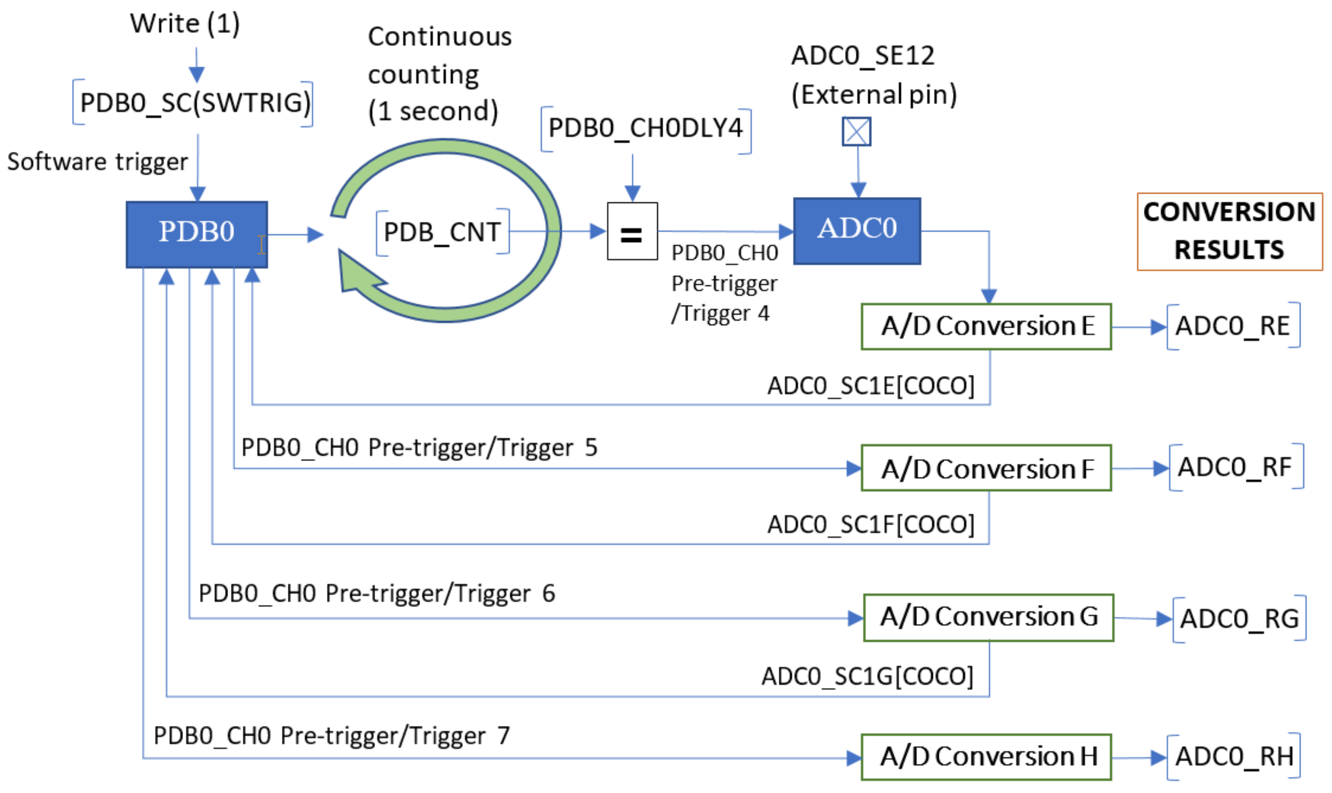

Figure 12 – ADC with PDB trigger in back-to-back mode

4.4. TRGMUX trigger

For the example code refer to Appendix D.

The TRGMUX is a very flexible module for interconnecting the trigger inputs of peripherals to a wide variety of internal and/or external trigger signals (timer modules, analog modules flags, external pins). In particular for ADC in S32K1xx, the TRGMUX can be used to synchronize conversions with any of the available trigger signals. It is worth mentioning that the TRGMUX mechanism can be used when triggering ADC conversions for SC1n registers 0 to 3 [registers SC1A, SC1B, SC1C and SC1D], and this kind of trigger always goes through the trigger latching gasket.

In the example, a single ADC0 conversion of external channel 12 (ADC0_SE12) is triggered with each rising edge of an external signal, in this case the signal TRGMUX_IN0. For this use case, a software pre-trigger must be provided to ADC0 by writing to the SIM_ADCOPT[ADC0SWPRETRG] register field.

The figure below shows the overall path of the trigger (blue dotted line) and pre-trigger (red dotted line) signals:

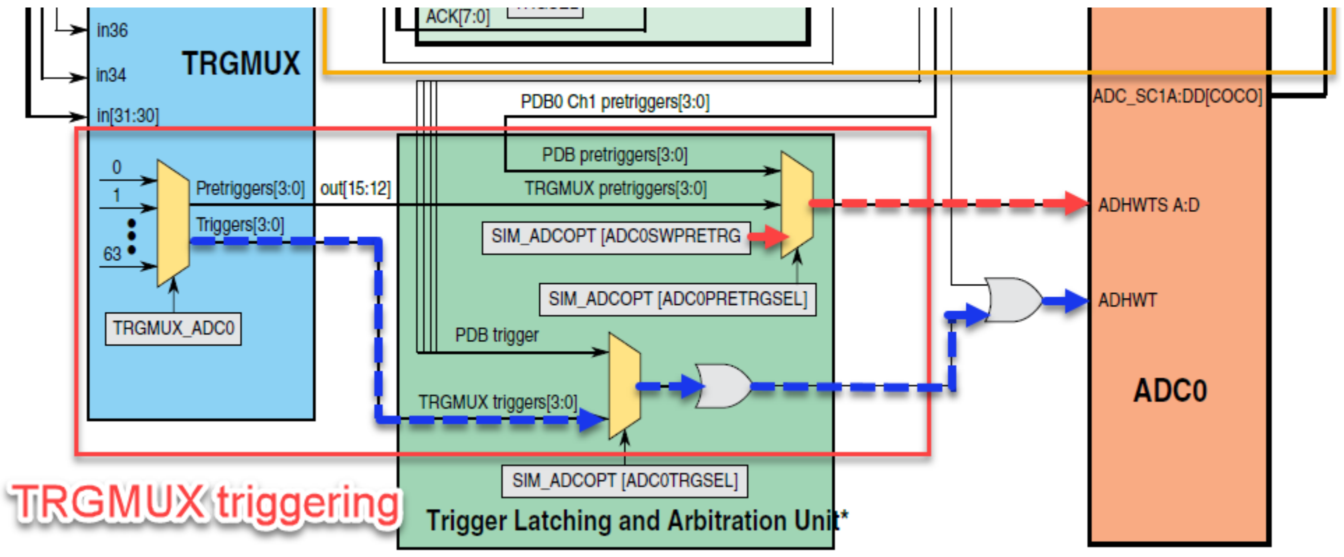

Figure 13 – ADC triggering via TRGMUX

Example working flow

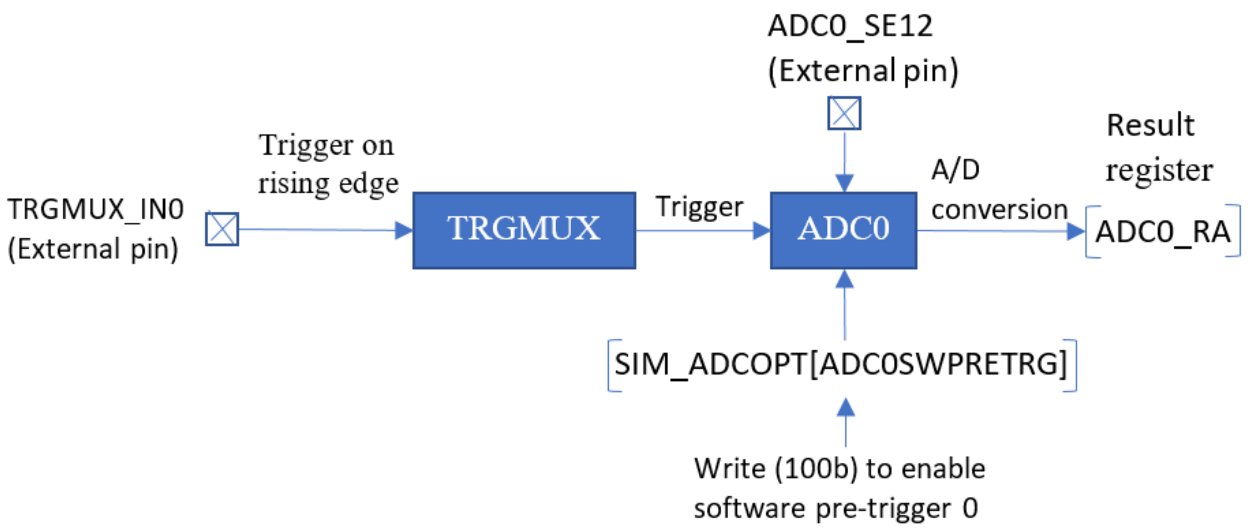

Figure 14 - ADC with TRGMUX trigger example working flow

5. References

- S32K1xx Datasheet
- S32K1xx Reference Manual
- AN5426 Hardware Design Guidelines for S32K1xx
- AN4373 Cookbook for SAR ADC Measurements
- ADC Calibration document

Appendix A. Example code: ADC software triggering

```
#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;
}
```

Appendix B. Example code: ADC with PDB trigger

```
#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;
}
```

Appendix C. Example code: ADC with PDB and back-to-back triggers

```
#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;
}
```

Appendix D. Example code: ADC with TRGMUX trigger

```
#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;
}
```

Document Number: AN12217
Rev. 0
08/2018