AN12217

S32K1xx and S32M24x ADC Guidelines, Spec and Configuration

Rev. 2.0 — 5 September 2024

Application note

Document information

Information	Content
Keywords	S32K1xx, S32M24x, ADC
Abstract	This application note presents information to understand ADC terminology, best practices, and configuration examples to get the most benefit from using the ADC module.

1 Introduction

NXP S32K1xx and S32M24x automotive microcontroller devices 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 and S32M24x family.

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 and S32M24x family datasheet.

2.1 ADC basic concepts

Resolution: The number of bits in the ADC digital output representing an analog input signal. For S32K1xx and S32M24x 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.

$$VREF = VREFH - VREFL \tag{1}$$

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.

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

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:

$$LSB = \frac{VREF}{2^N} \tag{3}$$

N = ADC resolution. For S32K1xx and S32M24x 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.

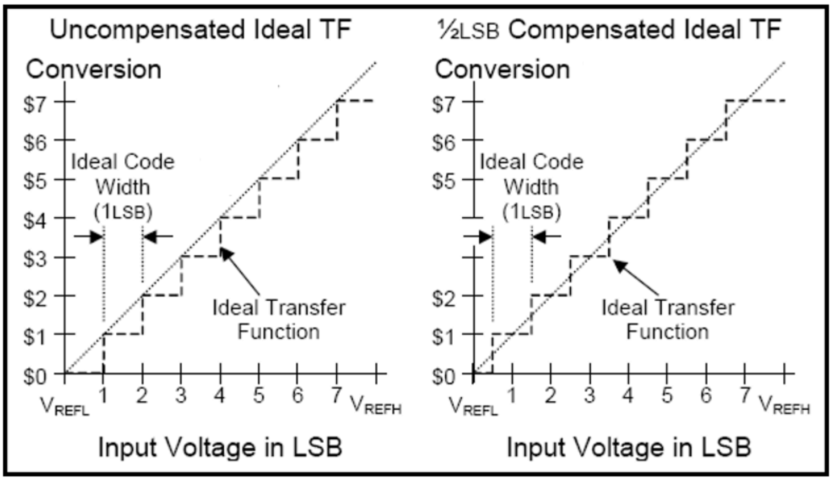

Figure 1. Ideal transfer functions

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 2 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.

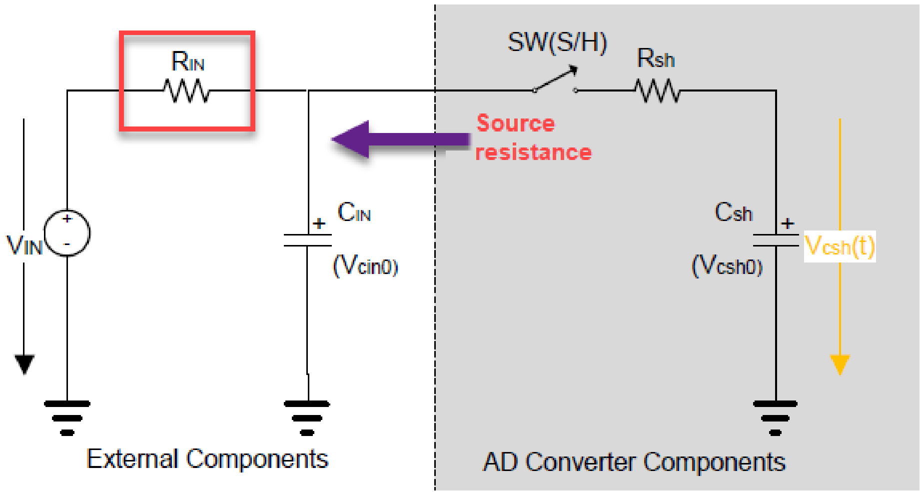

Figure 2. Analog signal source resistance

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 and S32M 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.

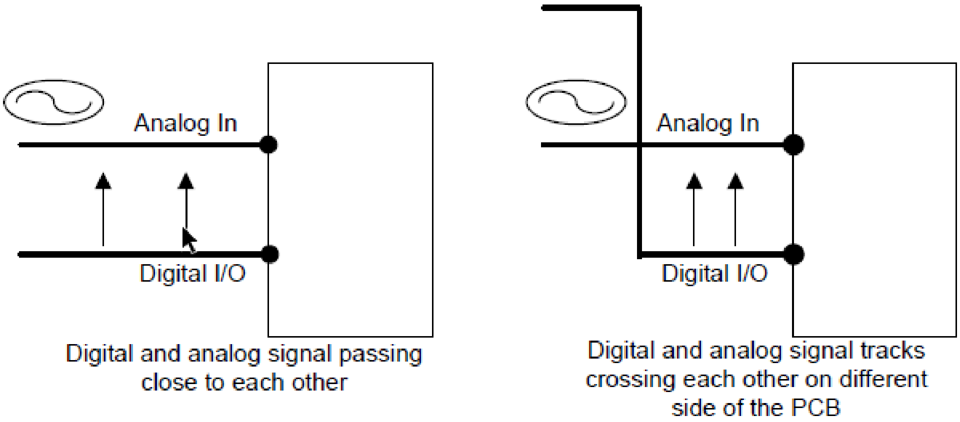

Figure 3. I/O pin crosstalk

2.3 S32K1xx and S32M24x ADC specifications

This section explains the parameters that integrate the specification of SAR ADC found in datasheet for S32K1xx and S32M24x 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.

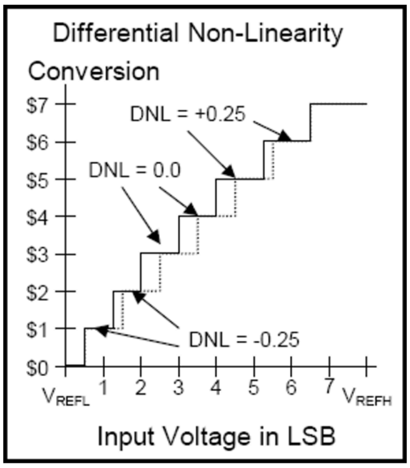

Figure 4. Differential Non-Linearity (DNL)

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.

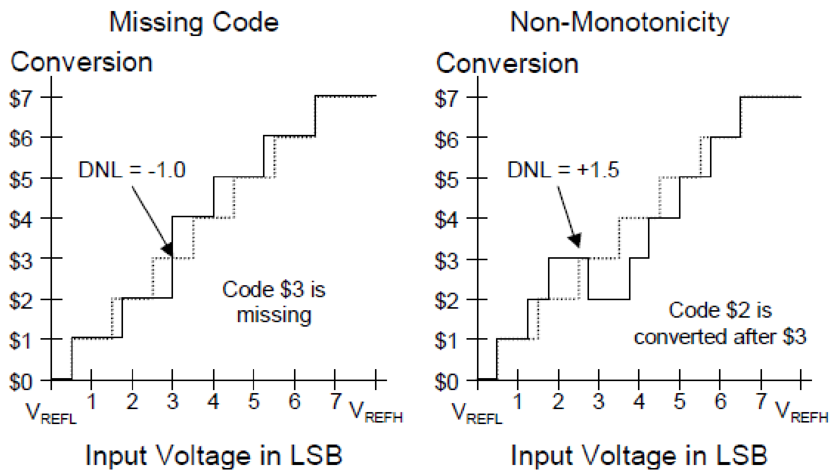

Figure 5. Missing codes and Non-Monotonicity

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 4.

$$INL(x) = \sum_{i=1}^{(x-1)} DNL(i) \quad (4)$$

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

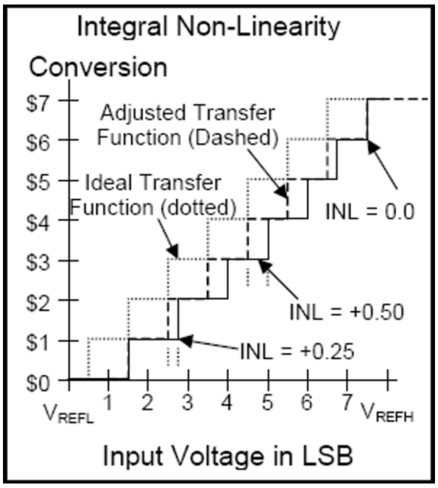

Figure 6. Integral Non-Linearity (INL)

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.

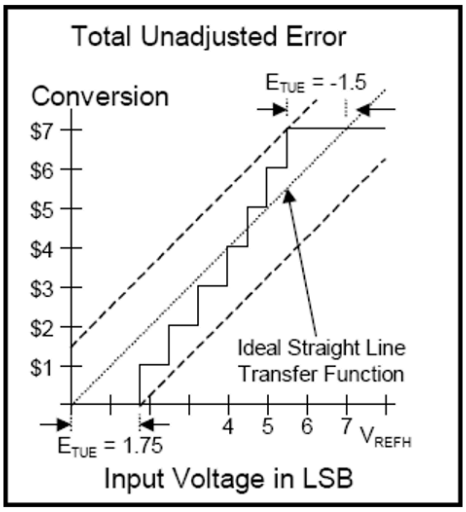

Figure 7. Total Unadjusted Error (TUE)

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 and S32M24x families have 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:

$$VREF\left(mV\right) = \frac{\left(1000\right)\left(2^{N}\right)}{BG\_ADCresult} \tag{5}$$

Where:

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

BG_ADCresult = The ADC conversion result for the bandgap channel

1. Consider the resulting VREF in Equation 5 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 and S32M24x 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 and S32M24x 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:

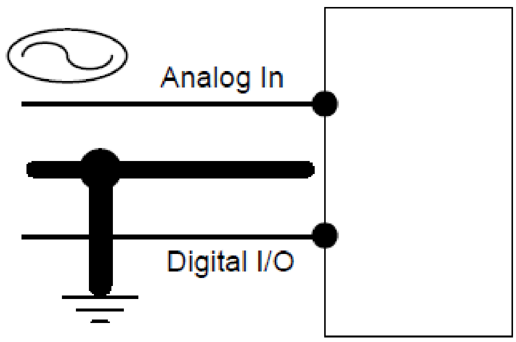

Figure 8. Recommended grounding between signals

4 ADC triggering mode examples

The ADC in S32K1xx and S32M24x families provide 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 and the S32M244 EVB boards, using the potentiometer as the source of the analog signal and the user switch 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 and S32M24x 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

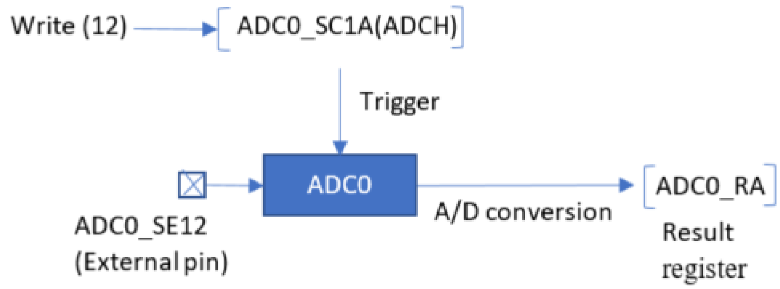

Figure 9. ADC software trigger working flow

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.

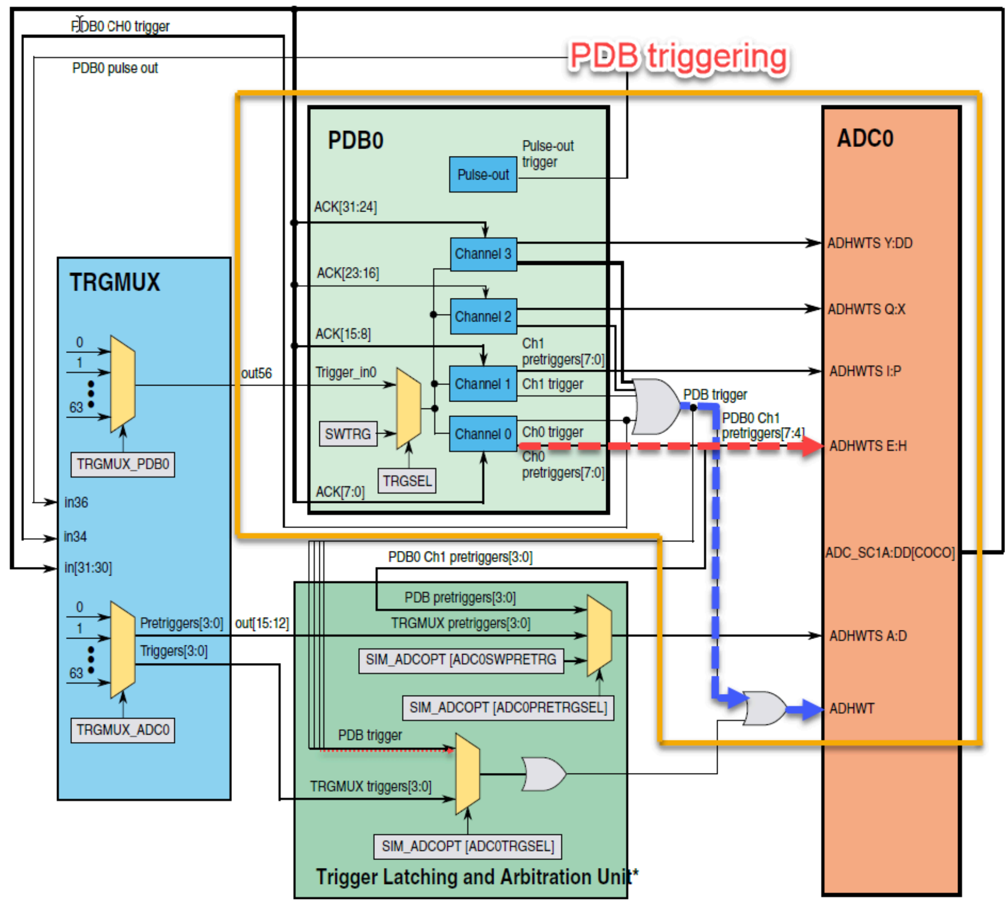

Figure 10. PDB trigger in direct path mode

Example working flow

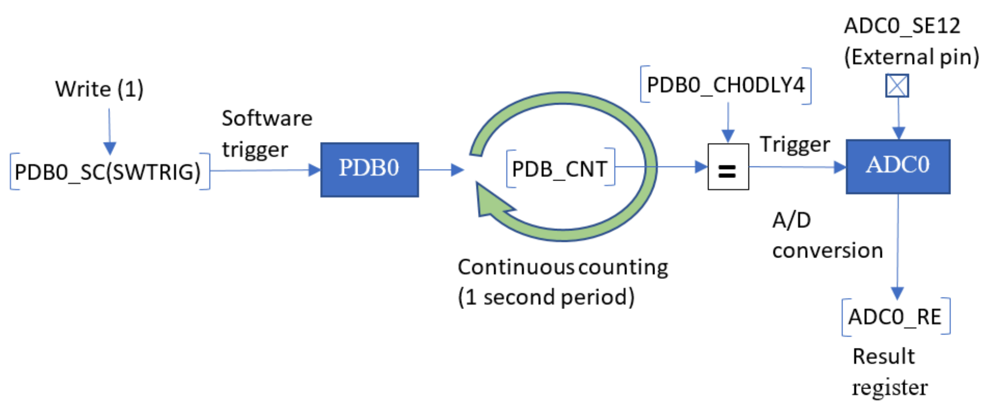

Figure 11. ADC with PDB trigger example working flow

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

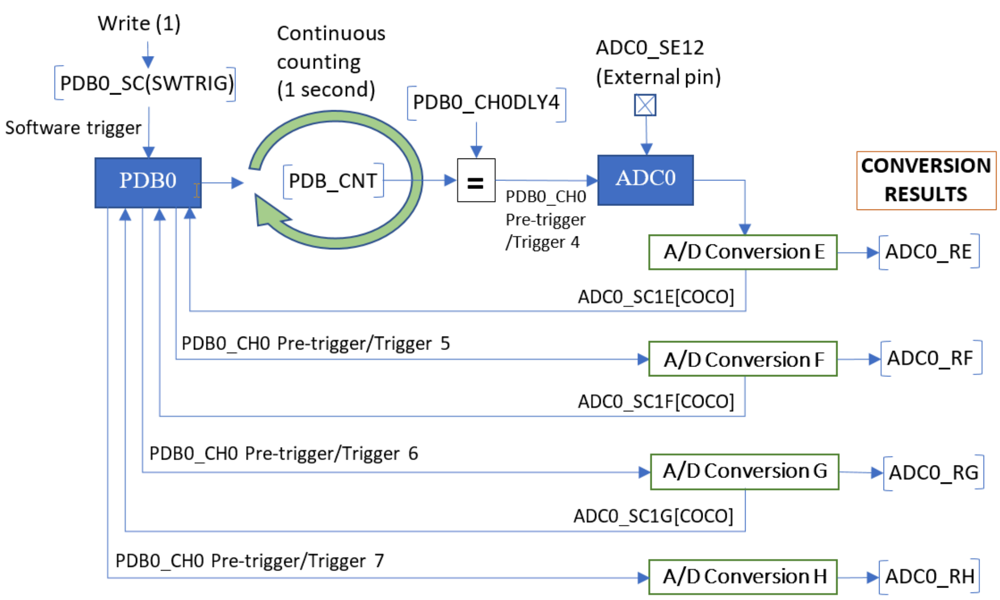

Figure 12. ADC with PDB trigger in back-to-back mode

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 and S32M24x, 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:

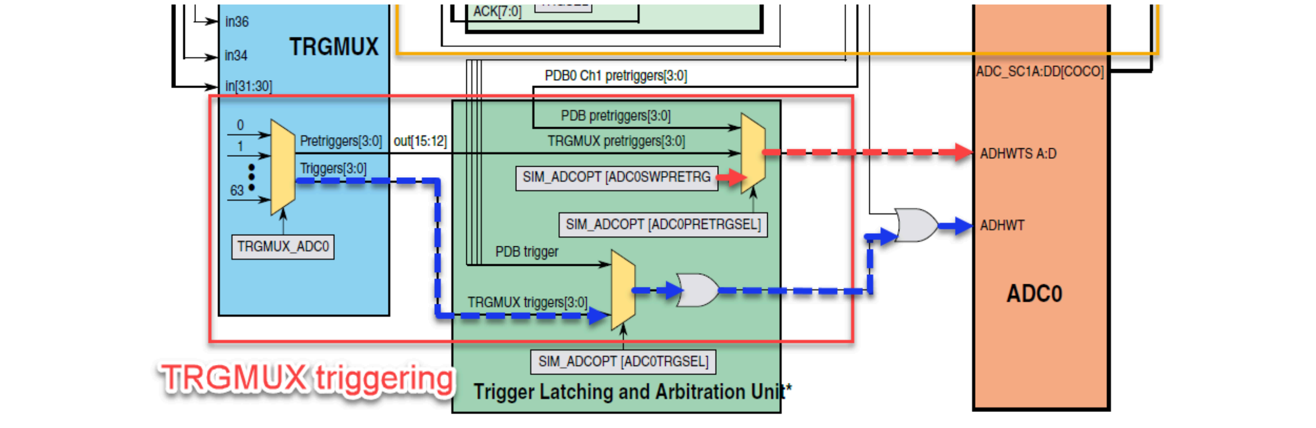

Figure 13. ADC triggering via TRGMUX

Example working flow

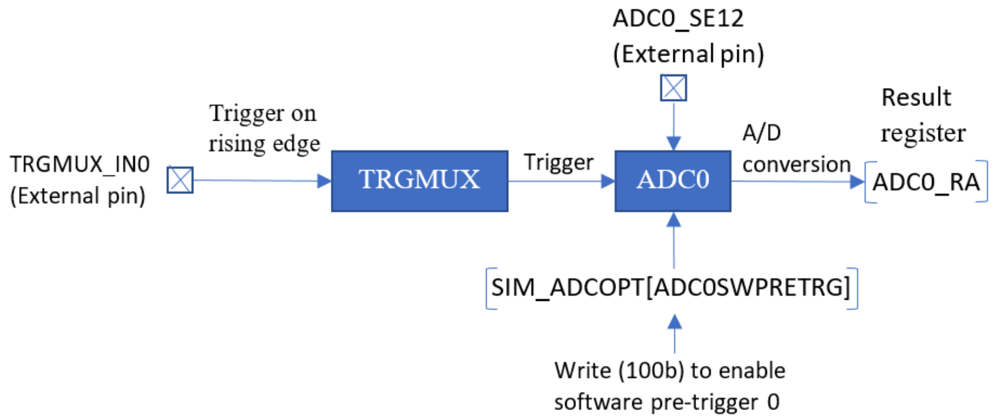

Figure 14. ADC with TRGMUX trigger example working flow

5 References

- S32K1xx Datasheet
- S32M2xx Datasheet
- S32K1xx Reference Manual
- S32M24x Reference Manual
- AN5426 Hardware Design Guidelines for S32K1xx
- AN4373 Cookbook for SAR ADC Measurements
- ADC Calibration document

6 Appendix

6.1 Example code: ADC software triggering with the S32K144 device

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

uint32_t ADC_RawResult;
```



```
uint16_t ADC_mVResult;

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

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

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

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

    IP_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(((IP_ADC0->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);
    /*******************************************************
     * Initialize ADC0:
     * External channel 12, software trigger,
     * single conversion, 12-bit resolution
     *******************************************************/
    IP_ADC0->SC1[0] = ADC_SC1_ADCH_MASK; /* ADCH: Module disabled for
 conversions */

    IP_ADC0->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1);   /* ADIV = 0: Divide
 ratio = 1 */
            /* MODE = 1: 12-bit conversion */

    IP_ADC0->CFG2 = ADC_CFG2_SMPLTS(12); /* SMPLTS = 12: sample time is 13 ADC
 clks */

    IP_ADC0->SC2 = ADC_SC2_ADTRG(0); /* ADTRG = 0: SW trigger */

    IP_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) */
        IP_ADC0->SC1[0] = ADC_SC1_ADCH(12);   /* ADCH = 12: External channel 12
 as input */

        /* Wait for latest conversion to complete */
        while(((IP_ADC0->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);
        ADC_RawResult = IP_ADC0->R[0]; /* Read ADC Data Result A (ADC0_RA) */
        ADC_mVResult = (ADC_RawResult * 5000) / (1<<12);  /* Convert to mV
 (@VREFH = 5V) */
```

```
    }
    return 0;
}
```

6.2 Example code: ADC with PDB trigger with the S32K144 device

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

uint32_t ADC_RawResult;
uint16_t ADC_mVResult;

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

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

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

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

    IP_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(((IP_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
     ******************************************************/
    IP_ADC0->SC1[4] = ADC_SC1_ADCH_MASK; /* ADCH: Module disabled for
 conversions */

    IP_ADC0->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1);   /* ADIV = 0: Divide
 ratio = 1 */
            /* MODE = 1: 12-bit conversion */

    IP_ADC0->CFG2 = ADC_CFG2_SMPLTS(12); /* SMPLTS = 12: sample time is 13 ADC
 clks */
```



```
    IP_ADC0->SC2 = ADC_SC2_ADTRG(1); /* ADTRG = 1: HW trigger */

    IP_ADC0->SC1[4] = ADC_SC1_ADCH(12); /* ADCH = 12: External channel 12 as
 ADC0 input */

    IP_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
     ************************************************/

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

     IP_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 */
    IP_PDB0->MOD = 18750;

  IP_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 */

    IP_PDB0->CH[0].DLY[4] = 9375; /* Delay set to half the PDB period = 9375 */

    IP_PDB0->SC |= PDB_SC_PDBEN_MASK | PDB_SC_LDOK_MASK;  /* Enable PDB. Load
 MOD and DLY */

    IP_PDB0->SC |= PDB_SC_SWTRIG_MASK;        /* Single initial PDB trigger */

    for(;;)
    {
        /* Wait for latest conversion to complete */
        while(((IP_ADC0->SC1[4] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);

        ADC_RawResult = IP_ADC0->R[4]; /* Read ADC Data Result E (ADC0_RE) */
        ADC_mVResult = (ADC_RawResult * 5000) / (1<<12); /* Convert to mV
 (@VREFH = 5V) */
    }
    return 0;
}
```

6.3 Example code: ADC with PDB and back-to-back triggers with the S32K144 device

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

uint32_t ADC_Results[4];

void WDOG_disable (void)
{
```

```
    IP_WDOG->CNT=0xD928C520;      /* Unlock watchdog */
    IP_WDOG->TOVAL=0x0000FFFF;    /* Maximum timeout value */
    IP_WDOG->CS = 0x00002100;     /* Disable watchdog */
}

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

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

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

    IP_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(((IP_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
     *************************************************/
    IP_ADC0->SC1[4] = ADC_SC1_ADCH_MASK;/* ADCH = 1F: Module is disabled for
 conversions*/
      /* AIEN = 0: Interrupts are disabled */
    IP_ADC0->SC1[5] = ADC_SC1_ADCH_MASK;
    IP_ADC0->SC1[6] = ADC_SC1_ADCH_MASK;
    IP_ADC0->SC1[7] = ADC_SC1_ADCH_MASK;

    IP_ADC0->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1);  /* ADIV = 0: Divide
 ratio = 1 */
            /* MODE = 1: 12-bit conversion */

    IP_ADC0->CFG2 = ADC_CFG2_SMPLTS(12); /* SMPLTS = 12: sample time is 13 ADC
 clks */

    IP_ADC0->SC2 = ADC_SC2_ADTRG(1); /* ADTRG = 1: HW trigger */

    IP_ADC0->SC1[4] = ADC_SC1_ADCH(12); /* SC1E[ADCH] = 12: External channel 12
 as input */
    IP_ADC0->SC1[5] = ADC_SC1_ADCH(12); /* SC1F[ADCH] = 12: External channel 12
 as input */
    IP_ADC0->SC1[6] = ADC_SC1_ADCH(12); /* SC1G[ADCH] = 12: External channel 12
 as input */
    IP_ADC0->SC1[7] = ADC_SC1_ADCH(12); /* SC1H[ADCH] = 12: External channel 12
 as input */
```

```
    IP_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
     **************************************************/

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

    IP_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 */
    IP_PDB0->MOD = 18750;

    IP_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 */

    IP_PDB0->CH[0].DLY[4] = 9375; /* Delay set to half the PDB period = 9375 */

    IP_PDB0->SC |= PDB_SC_PDBEN_MASK | PDB_SC_LDOK_MASK;  /* Enable PDB. Load
 MOD and DLY */

    IP_PDB0->SC |= PDB_SC_SWTRIG_MASK;       /* Single initial PDB trigger */

    for(;;)
    {
        /* Wait for last conversion in the sequence to complete (ADC0_SC1H) */
        while(((IP_ADC0->SC1[7] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);

        ADC_Results[0] = IP_ADC0->R[4]; /* Read ADC Data Results 4-7 (ADC0_RE to
 ADC0_H) */
        ADC_Results[1] = IP_ADC0->R[5];
        ADC_Results[2] = IP_ADC0->R[6];
        ADC_Results[3] = IP_ADC0->R[7];
    }

    return 0;
}
```

6.4 Example code: ADC with TRGMUX trigger with the S32K144 device

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

uint32_t ADC_RawResult;
uint16_t ADC_mVResult;
```

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

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

    IP_SCG->FIRCDIV =  SCG_FIRCDIV_FIRCDIV2(4); /* FIRCDIV2 = 4: FIRCDIV2 divide
 by 8 */
    /**************************************************
     * Configure pin PTB5 as TRGMUX_IN0
     **************************************************/
    IP_PCC->PCCn[PCC_PORTB_INDEX] = PCC_PCCn_CGC_MASK;/* Enable clock gate for
 PORTB */
    IP_PORTB->PCR[5] = PORT_PCR_MUX(6);   /* Mux = 6: PTB5 as TRGMUX_IN0 */

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

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

    IP_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(((IP_ADC0->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);

    /**************************************************
     * Initialize ADC0:
     * External channel 12, hardware trigger,
     * single conversion, 12-bit resolution
     *************************************************/
    IP_ADC0->SC1[0] = ADC_SC1_ADCH_MASK; /* ADCH: Module disabled for
 conversions */

    IP_ADC0->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1);/* ADIV = 0: Divide
 ratio = 1 */
           /* MODE = 1: 12-bit conversion */
    IP_ADC0->CFG2 = ADC_CFG2_SMPLTS(12); /* SMPLTS = 12: sample time is 13 ADC
 clks */

    IP_ADC0->SC2 = ADC_SC2_ADTRG(1); /* ADTRG = 1: HW trigger */

    IP_ADC0->SC1[0] = ADC_SC1_ADCH(12); /* ADCH = 12: External channel 12 as
 input */
```

```
    IP_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
     *************************************************/
 IP_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(((IP_ADC0->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);
        ADC_RawResult = IP_ADC0->R[0];  /* Read ADC Data Result 0 */
        ADC_mVResult = (ADC_RawResult * 5000) / (1<<12);  /* Convert to mV
 (@VREFH = 5V) */
 }
 return 0;
}
```

6.5 Example code: ADC software triggering with the S32M244 device

```
#include "S32M244.h"

uint32_t ADC_RawResult;
uint16_t ADC_mVResult;

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

int main(void) {

 WDOG_disable();  /* Disable Watchdog */

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

 /***** Calibrate ADC1 *****/
 IP_PCC->PCCn[PCC_ADC1_INDEX] &=~ PCC_PCCn_CGC_MASK;     /* Disable clock to
 change PCS */
 IP_PCC->PCCn[PCC_ADC1_INDEX] |= PCC_PCCn_PCS(3);     /* PCS = 3: Select
 FIRCDIV2 */
 IP_PCC->PCCn[PCC_ADC1_INDEX] |= PCC_PCCn_CGC_MASK;     /* Enable bus clock in
 ADC */

 IP_ADC1->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(((IP_ADC1->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);
```

```
 /**************************************************
  * Initialize ADC1:
  * External channel 11, software trigger,
  * single conversion, 12-bit resolution
  **************************************************/
 IP_ADC1->SC1[0] = ADC_SC1_ADCH_MASK;  /* ADCH: Module disabled for conversions
 */

 IP_ADC1->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1);    /* ADIV = 0: Divide
 ratio = 1 */
              /* MODE = 1: 12-bit conversion */

 IP_ADC1->CFG2 = ADC_CFG2_SMPLTS(12); /* SMPLTS = 12: sample time is 13 ADC clks
 */

 IP_ADC1->SC2 = ADC_SC2_ADTRG(0);  /* ADTRG = 0: SW trigger */

 IP_ADC1->SC3 = 0x00000000; /* ADCO = 0: One conversion performed */
     /* AVGE,AVGS = 0: HW average function disabled */

 for(;;)
 {
  /* Initiate new conversion by writing to ADC1_SC1A(ADCH) */
  IP_ADC1->SC1[0] = ADC_SC1_ADCH(11);  /* ADCH = 11: External channel 11 as
 input */

  /* Wait for latest conversion to complete */
  while(((IP_ADC1->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);
  ADC_RawResult = IP_ADC1->R[0]; /* Read ADC Data Result A (ADC1_RA) */
  ADC_mVResult = (ADC_RawResult * 5000) >> 12;
         /* Convert to mV (@VREFH = 5V) */
 }
 return 0;
}
```

6.6 Example code: ADC with PDB trigger with the S32M244 device

```
#include "S32M244.h"

uint32_t ADC_RawResult;
uint16_t ADC_mVResult;

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

int main(void) {

 WDOG_disable();    /* Disable Watchdog */

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

 /*********************************************
  * Calibrate ADC1
  *********************************************/
```

```
  IP_PCC->PCCn[PCC_ADC1_INDEX] &=~ PCC_PCCn_CGC_MASK;
                                        /* Disable clock to change PCS */
  IP_PCC->PCCn[PCC_ADC1_INDEX] |= PCC_PCCn_PCS(3);
                                        /* PCS = 3: Select FIRCDIV2 */
  IP_PCC->PCCn[PCC_ADC1_INDEX] |= PCC_PCCn_CGC_MASK;
                                        /* Enable bus clock in ADC */

  IP_ADC1->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(((IP_ADC1->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);
        /*******************************************************
   * Initialize ADC1:
   * External channel 11, hardware trigger,
   * single conversion, 12-bit resolution
   *
   * NOTE: ADC1->SC1[0] corresponds to ADC0_SC1A register
   *******************************************************/
  IP_ADC1->SC1[0] = ADC_SC1_ADCH_MASK;
                                        /* ADCH: Module disabled for conversions
  */

  IP_ADC1->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1);
                                                        /* ADIV = 0: Divide
  ratio = 1 */
                /* MODE = 1: 12-bit conversion */

  IP_ADC1->CFG2 = ADC_CFG2_SMPLTS(12);
                                        /* SMPLTS = 12: sample time is 13 ADC
  clks */

  IP_ADC1->SC2 = ADC_SC2_ADTRG(1);  /* ADTRG = 1: HW trigger */

  IP_ADC1->SC1[0] = ADC_SC1_ADCH(11);
 /* ADCH = 11: External channel 11 as ADC1 input */

  IP_ADC1->SC3 = 0x00000000;  /* ADCO = 0: One conversion performed */
       /* AVGE,AVGS = 0: HW average function disabled */
     /**************************************************
      * Initialize PDB1:
      * 1 second period, continuous mode
      * PDB1_CH0 pre-trigger 1 output enabled
      *************************************************/

  IP_PCC->PCCn[PCC_PDB1_INDEX] |= PCC_PCCn_CGC_MASK;
                                                    /* Enable bus clock in PDB
  */

  IP_PDB1->SC = PDB_SC_PRESCALER(6)  /* PRESCALER = 6 */
      | PDB_SC_TRGSEL(15)   /* Software trigger selected */
      | PDB_SC_MULT(3)    /* Multiplication factor is 40 */
      | PDB_SC_CONT_MASK;  /* 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 */
  IP_PDB1->MOD = 18750;
```

```
 IP_PDB1->CH[0].C1 = (PDB_C1_TOS(0x01)  /* Pre-trigger 1 asserts with DLY match
 */
           | PDB_C1_EN(0x01)); /* Pre-trigger 1 enabled */

       IP_PDB1->CH[0].DLY[1] = 9375; /* Delay set to half the PDB period = 9375
 */

 IP_PDB1->SC |= PDB_SC_PDBEN_MASK | PDB_SC_LDOK_MASK;
                                                     /* Enable PDB. Load MOD and
 DLY */

 IP_PDB1->SC |= PDB_SC_SWTRIG_MASK;  /* Single initial PDB trigger */

 for(;;)
 {
  /* Wait for latest conversion to complete */
  while(((IP_ADC1->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);

  ADC_RawResult = IP_ADC1->R[0]; /* Read ADC Data Result E (ADC1_RA) */
  ADC_mVResult = (ADC_RawResult * 5000) >> 12;
                                                 /* Convert to mV (@VREFH = 5V)
 */
 }

    return 0;
}
```

6.7 Example code: ADC with PDB and back-to-back triggers with the S32M244 device

```
#include "S32M244.h"

uint32_t ADC_RawResult;
uint16_t ADC_mVResult;

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

int main(void) {

 WDOG_disable();    /* Disable Watchdog */

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

 /*************************************************
  * Calibrate ADC1
  *************************************************/
 IP_PCC->PCCn[PCC_ADC1_INDEX] &=~ PCC_PCCn_CGC_MASK;
/* Disable clock to change PCS */
 IP_PCC->PCCn[PCC_ADC1_INDEX] |= PCC_PCCn_PCS(3);
/* PCS = 3: Select FIRCDIV2 */
 IP_PCC->PCCn[PCC_ADC1_INDEX] |= PCC_PCCn_CGC_MASK;
/* Enable bus clock in ADC */

 IP_ADC1->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(((IP_ADC1->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);
        /************************************************
   * Initialize ADC1:
   * External channel 12, hardware trigger,
   * single conversion, 12-bit resolution
   *
   * NOTE: ADC1->SC1[4] corresponds to ADC0_SC1E register
   ************************************************/
  IP_ADC1->SC1[4] = ADC_SC1_ADCH_MASK;
                               /* ADCH = 1F: Module is disabled for
  conversions*/
      /* AIEN = 0: Interrupts are disabled */
  IP_ADC1->SC1[5] = ADC_SC1_ADCH_MASK;
  IP_ADC1->SC1[6] = ADC_SC1_ADCH_MASK;
  IP_ADC1->SC1[7] = ADC_SC1_ADCH_MASK;

  IP_ADC1->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1);
                                                    /* ADIV = 0: Divide ratio =
  1 */

              /* MODE = 1: 12-bit conversion */

  IP_ADC1->CFG2 = ADC_CFG2_SMPLTS(12);
                                       /* SMPLTS = 12: sample time is 13 ADC
  clks */

  IP_ADC1->SC2 = ADC_SC2_ADTRG(1);
                                   /* ADTRG = 1: HW trigger */

  IP_ADC1->SC1[4] = ADC_SC1_ADCH(11);
                               /* SC1E[ADCH] = 11: External channel 11 as input
  */
  IP_ADC1->SC1[5] = ADC_SC1_ADCH(11);
                               /* SC1F[ADCH] = 11: External channel 11 as input
  */
  IP_ADC1->SC1[6] = ADC_SC1_ADCH(11);
                               /* SC1G[ADCH] = 11: External channel 11 as input
  */
  IP_ADC1->SC1[7] = ADC_SC1_ADCH(11);
                               /* SC1H[ADCH] = 11: External channel 11 as input
  */

  IP_ADC1->SC3 = 0x00000000;  /* ADCO = 0: One conversion performed */
       /* AVGE,AVGS = 0: HW average function disabled */

        /************************************************
   * Initialize PDB1:
   * 1 second period, continuous mode
   * PDB1_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
   *************************************************/

  IP_PCC->PCCn[PCC_PDB1_INDEX] |= PCC_PCCn_CGC_MASK; /* Enable bus clock in PDB
  */
```

```
 IP_PDB1->SC = PDB_SC_PRESCALER(6)    /* clk divided by (64 x Mult factor) */
    | PDB_SC_TRGSEL(15)  /* Software trigger selected */
    | PDB_SC_MULT(3)  /* Multiplication factor is 40 */
    | PDB_SC_CONT_MASK; /* 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 */
 IP_PDB1->MOD = 18750;

 IP_PDB1->CH[0].C1 = (PDB_C1_BB(0xE0) /* Back-to-back for pre-triggers 5/6/7 */
       | PDB_C1_TOS(0x10)
                                              /* Pre-trigger 4 asserts with DLY
 match */
       | PDB_C1_EN(0xF0)); /* Pre-triggers 4/5/6/7 enabled */

 IP_PDB1->CH[0].DLY[4] = 9375; /* Delay set to half the PDB period = 9375 */

 IP_PDB1->SC |= PDB_SC_PDBEN_MASK | PDB_SC_LDOK_MASK;
                                                /* Enable PDB. Load MOD and DLY
 */

 IP_PDB1->SC |= PDB_SC_SWTRIG_MASK; /* Single initial PDB trigger */

       for(;;)
 {
  /* Wait for last conversion in the sequence to complete (ADC1_SC1H) */
  while(((IP_ADC1->SC1[7] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);

  /* Read ADC Data Results 4-7 (ADC1_RE to ADC1_RH) */
  ADC_Results[0] = IP_ADC1->R[4];
  ADC_Results[1] = IP_ADC1->R[5];
  ADC_Results[2] = IP_ADC1->R[6];
  ADC_Results[3] = IP_ADC1->R[7];
 }

 return 0;
}
```

6.8 Example code: ADC with TRGMUX trigger with the S32M244 device

```
#include "S32M244.h"

uint32_t ADC_RawResult;
uint16_t ADC_mVResult;

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

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

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

 /***********************************************
```

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

 /* Select TRGMUX_IN0 as ADC1 Trigger Mux input source 0 */
 IP_TRGMUX->TRGMUXn[TRGMUX_ADC1_INDEX] = TRGMUX_TRGMUXn_SEL0(2U);

 /***********************************************
  * Calibrate ADC1
  ***********************************************/
 IP_PCC->PCCn[PCC_ADC1_INDEX] &=~ PCC_PCCn_CGC_MASK;
                                                       /* Disable clock to
 change PCS */
 IP_PCC->PCCn[PCC_ADC1_INDEX] |= PCC_PCCn_PCS(3);    /* PCS = 3: Select FIRCDIV2
 */
 IP_PCC->PCCn[PCC_ADC1_INDEX] |= PCC_PCCn_CGC_MASK; /* Enable bus clock in ADC
 */
    IP_ADC1->SC3 = ADC_SC3_CAL_MASK  /* Start calibration sequence */
     | ADC_SC3_AVGE_MASK /* Enable hardware average */
     | ADC_SC3_AVGS(3);  /* 32 samples averaged */

 /* Wait for completion */
 while(((IP_ADC1->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);

 /***********************************************
  * Initialize ADC1:
  * External channel 11, hardware trigger,
  * single conversion, 12-bit resolution
  ***********************************************/
 IP_ADC1->SC1[0] = ADC_SC1_ADCH_MASK; /* ADCH: Module disabled for conversions
 */

 IP_ADC1->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1);   /* Divide ratio = 1 */
                                  /* MODE = 1: 12-bit conversion */

 IP_ADC1->CFG2 = ADC_CFG2_SMPLTS(12); /* Sample time is 13 ADC clks */

 IP_ADC1->SC2 = ADC_SC2_ADTRG(1);  /* HW trigger */

 IP_ADC1->SC1[0] = ADC_SC1_ADCH(11);  /* External channel 11 as input */

 IP_ADC1->SC3 = 0x00000000;  /* 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
  ***********************************************/
 IP_SIM->ADCOPT = SIM_ADCOPT_ADC1PRETRGSEL(2)      /* Software pretrigger */
       | SIM_ADCOPT_ADC1SWPRETRG(4)  /* SW Pre-trigger 0 */
       | SIM_ADCOPT_ADC1TRGSEL(1); /* TRGMUX output as trigger */

 for(;;)
 {
  /* Wait for latest conversion to complete */
  while(((IP_ADC1->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT) == 0);
  ADC_RawResult = IP_ADC1->R[0];     /* Read ADC Data Result A */
  ADC_mVResult = (ADC_RawResult * 5000) >> 12;
```

```
    /* Convert to mV (@VREFH = 5V) */
  }
  return 0;
}
```

7 Revision histroy

Table 1. Revision history

Document ID	Release date	Description
AN12217 v. 2	5 September 2024	Added information for S32M24X devices.
AN12217 v. 1	January 2020	Updated example code in Section 6.2.
AN12217 v. 0	August 2018	Initial release

8 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Suitability for use in automotive applications — This NXP product has been qualified for use in automotive applications. If this product is used by customer in the development of, or for incorporation into, products or services (a) used in safety critical applications or (b) in which failure could lead to death, personal injury, or severe physical or environmental damage (such products and services hereinafter referred to as "Critical Applications"), then customer makes the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, safety, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. As such, customer assumes all risk related to use of any products in Critical Applications and NXP and its suppliers shall not be liable for any such use by customer. Accordingly, customer will indemnify and hold NXP harmless from any claims, liabilities, damages and associated costs and expenses (including attorneys' fees) that NXP may incur related to customer's incorporation of any product in a Critical Application.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

Contents

Date of release: 5 September 2024
Document identifier: AN12217