ADC Guidelines Spec and Configuration

by: NXP Semiconductor

1. Introduction

NXP S32K3xx automotive microcontroller devices feature a 15-bit resolution 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 to get the maximum benefits 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 S32K3xx family.

Special Features for SARADC S32K3xx:

- 15-bit resolution. Conversion data is always 15 bits wide, regardless of the conversion resolution selected (14, 12, 10, 8 significant bits per conversion data, reducing this number speeds up the conversion because the SAR algorithm executes fewer steps).
- Conversion data captured in a separate register for each input channel.
- Option to improve accuracy via averaging, which calculates conversion data by averaging the data of up to 32 conversions.
- Conversion Options:

Contents

1. Introduction ................................................................. 1
2. ADC concepts, error sources and specification .......... 2
   2.1. ADC basic concepts ........................................... 2
   2.2. Sources of error in ADC measurements ............... 4
   2.3. S32K3xx ADC specifications .............................. 6
3. Best practices to increase accuracy .............................. 11
   3.1. ADC calibration ................................................. 11
   3.2. Reference voltage and power supply .................... 12
   3.3. Using bandgap to monitor reference voltage .......... 12
   3.4. Non Linearity correction ..................................... 13
   3.5. Analog source resistance match .......................... 14
   3.6. Minimizing I/O pin crosstalk ............................. 15
4. ADC triggering ............................................................ 15
   4.1. Software trigger ................................................. 15
   4.2. Hardware triggers ............................................... 18
5. SAR ADC use cases .................................................... 20
   5.1. SAR ADC- ADC calibration use case ................. 20
   5.2. SAR ADC Clock configuration use case ............ 23
   5.3. SAR ADC periodic chain conversions use case ... 24
   5.4. SAR ADC injected conversions use case .......... 26
Appendix A. Why to perform ADC calibration ............... 28
Appendix B. ADC conversion time ................................. 29
Appendix C. ADC Self-test .............................................. 30
— Normal conversion trigger converts a number of input channels, either once per trigger or continuously.
— Injected conversion trigger interrupts an ongoing normal conversion and converts another set of input channels.

• Hardware Conversion Triggers:
  — BCTU conversion trigger interrupts an ongoing conversion and converts an input channel, in which the input is selected and the conversion is started via the BCTU.
  — Other hardware trigger signals can be provided from TRGMUX outputs.
• An analog watchdog monitors conversion data for each input channel and issues an interrupt if the converted data is below or above configurable limits.
• DMA functionality transfers conversion data to other modules.
• Programmable interrupts optionally issue an interrupt when conversion of one input channel or of a set of input channels is finished.
• Self-test functions validate ADC structural integrity during functional operation and generate events with different severities on any finding.
• Conversion clock (AD_clk) control enables the use of ADC in systems with a higher clock frequency by using internal clock dividers.
• ADC is always in Functional mode. No other mode selection exists, Auto turn off of the conversion clock when ADC is idle.
  — Put ADC into Power Down state by writing 1 to MCR[PWDN] to reduce power consumption.
  — Gate the clock signal by writing 1 to MCR[ACKO] when ADC is in Idle state.

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. It also provides information on the specification parameters present in the S32K3xx family datasheet.

2.1. ADC basic concepts

Resolution: The number of bits in the ADC digital output representing an analog input signal. For S32K3xx devices the resolution can be configured to 14, 12, 10 or 8 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.

\[ V_{REF} = V_{REFH} - V_{REFL} \]

Eqn. 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. Refer to the device Data Sheet for the Absolute maximum ratings for VREFH, VRELF [Table 2. Absolute maximum ratings]

\[
ADC_{result} = \frac{(2^N)(Vin)}{VREF}
\]

Eqn. 2 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:

\[
LSB = \frac{VREF}{2^N}
\]

Eqn. 3 LSB equation

N = ADC resolution. For S32K3xx this can be 14, 12, 10 or 8 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:

1. Uncompensated Quantization: The first step is taken at 1 LSB, with each successive step taken at 1 LSB intervals and the last step taken at VREFH – 1 LSB.
2. **1/2 LSB Compensated Quantization**: The first step is taken at ½ LSB, with each successive step taken at 1 LSB intervals and the last step taken at VREFH – 1½ LSB.

The following figure shows the ideal transfer function graphs for uncompensated and ½ LSB compensated methods. In a seek of simplicity and concept enforcement, this is an hypothetical example considering a 3 bit ADC resolution within a VREF = 8 V.

![Ideal transfer functions](image)

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.

#### 2.2.1. 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.
2.2.2. 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.

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.

![Analog signal source resistance](image)

**Figure 2. Analog signal source resistance**

As shown in the above figure, the sampling of the input signal is achieved by charging an internal capacitor (C_{sh}), controlling a switch with resistance (R_{sh}). 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 S32Kxx devices [Table 29. SAR ADC], this parameter can be found as Source Impedance (R_S).

**NOTE**

TUE is degrading when increasing source impedance

2.2.3. I/O pin currents

The ADC behaves as a capacitor, which during conversion time (i.e. ADC clock / (number of samples + a few cycles)) is disconnected from the input pin and charged internally to VREF/2, and then when conversion finishes, connected to the input pin for the duration of sampling period through a few tens of ohms switch. So, from a high impedance sampling source, with zero voltage, in the I/O pin occur pulses at the rate of the sampling, due current flowing out of the pin. This manifests as voltage "surges" on the ADC channel.
NOTE
Consider for precision channels a capacitance of 0.57pF (ADC0), 1.34pF (ADC1), and 1.32pF (ADC2). And the standard channels 4.18 pF (ADC0), /4.13 pF (ADC1), and 4.03pF (ADC2).

On the other hand if the sampling source is at VREF, current should flow into the pin. For the case when the sampling source is approximately at VREF/2, current pulses should diminish. For these cases no voltage “surges” are observed in the ADC channel.

2.2.4. 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.

![I/O pin crosstalk](image)

Figure 3. I/O pin crosstalk

2.2.5. 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.

2.3. S32K3xx ADC specifications

This section explains the parameters that integrate the specification of SAR ADC present in S32K3xx devices datasheet.
ADC concepts, error sources and specification

**ADC clock frequency** (fADCK): 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** (fCONV): 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 following factors:
- ADC clock frequency
- Hardware averaging enabled or disabled
- Number of samples
- Configuration (single or continuous conversions)

**Differential Non-Linearity (DNL):** The differential non-linearity error is a “code width error”, where code width is the range of input voltages, VADIN, 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.

![Differential Non-Linearity (DNL)](image)

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 Differential Non-Linearity](image)

**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 calculated by the following equation.

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

*Eqn. 4 INL equation*

The next figure shows a representation of INL based on the cumulative effect of the individual DNLs.
Figure 6. **Integral Non-Linearity**

**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.

**ADC Offset error:** is defined as the deviation between the first ideal code transition and the first actual code transition. The first ideal code transition takes place at 0.5 LSB. If the output code is greater than zero when the input voltage is less than 0.5 LSB, the ADC has a positive offset error. ADC has a negative offset error if the first output code transition occurs when the input voltage is greater than 0.5 LSB. Both positive and negative offset errors limit the available range of the ADC. A large positive offset error causes the ADC to saturate before the input voltage reaches maximum. A large negative offset error results in zero ADC output code for small input voltages.

### 2.3.1. ADC Gain error

It is defined as the deviation of the midpoint of the last step of the ideal ADC transfer from the midpoint of the last step of the actual ADC, after the offset error is compensated. If the transfer function of the actual ADC results in ADC saturation before the input voltage reaches maximum, a positive gain error is produced. If the transfer function of the actual ADC is such that the ADC does not reach full-scale value when the input voltage is at maximum, a negative gain error is produced. Gain error can also be represented as the full-scale error minus the offset error.

Gain slope for the transfer function can be easily calculated with a two point calibration method (rise/run). If thinking of the slope of a line with a X-Y graph it is simply:

\[ Y = mX + b \]

*Eqn. 5 Straight line equation*

where in the case of the ADC the analog input voltage value is X, the gain error slope is "m" and the offset error is "b". The following figure shows the transfer function for a two point calibration.
For calibration, it is required to take two ADC measurements at different conditions and calculate the offsets based on those two points, this is the so-called 2-point calibration method.

For the offset relative to what the ADC is seeing from a shorted input case. If the ADC is set to have a zero offset (0 input results in 0 output), then the reading can be above or below the expected due to gain or linearity errors. Then it is also important to take that ADC’s behavior into account.

### 3. Best practices to increase accuracy

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

#### 3.1. ADC calibration

The SAR ADC in S32K3xx 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 subsequent 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.
Best practices to increase accuracy

- 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 refer to the document which can be downloaded from the following link. Although the document refers to a 16-bit SAR ADC and other NXP microcontroller families such as Kinetis, the ADC module in S32Kx3x shares the same basic architecture, so the theory is applicable.

3.2. 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.

3.3. 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 steps are as follows:
1. Trigger an ADC conversion for the bandgap (channel 48).
2. Calculate the actual VREF using the following equation:

$$V_{REF} (mV) = \frac{(1000)(2^N)}{BG\_ADC\_result}$$

Eqn. 6 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 the equation for any voltage calculations in the application.

### 3.4. Non Linearity correction

An important consideration for the two point method, described in the ADC Gain error section, is to consider just as a broad approach for ADC calibration calculation.

It is ok if the measurement remains perfectly linear. The following figures shows what happens with a two point measurements technique, if the response of the System/Application, everything together, is non-linear, that is the actually case. Figure 9 would be similar to a system calibration where one point with a 0 or the shorted input case, and the other point is with a full-scale input. The figure shows an error in the middle.

![Non Linearity Error due calibration points in the outer limits](image)

In the following figure it can be noticed what happens if the calibration is performed in the middle.
Figure 10. **Non Linearity Error due calibration points in the middle**

Now the error in the measurement is present on either side of the calibrated region. What are left with is to calibrate using more regions (more points to perform more partitions) to create a piece-wise linear correction. The more partitions the calibration consider, the closer it will represent the actual non-linear curve response for the system within the ADC. Obviously there is a practical limit to perform for partitions.

### 3.5. 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 component 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 (fADCK)
- Longer sample times.

The sampling time in S32K3xx devices can be scaled via MCR[ADCLKSEL]. ADCLKSEL can only be written in Power Down state (MCR[PWDN] = 1).
Depending on the frequency of the module clock, there might be different settings necessary for functional conversion and for calibration. See the chip data sheet section ADC electrical characteristics for more information.

For a more information on how to design the external RC acquisition circuit and selection of components, refer to 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 S32Kx3x shares the same basic architecture, so the theory is also applicable.

3.6. **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 following figure, shows a representation of such shielding approach:

![Recommended grounding between signals](image)

**Figure 11. Recommended grounding between signals**

4. **ADC triggering**

4.1. **Software trigger**

Software trigger is the simplest of the trigger modes. It simply starts a normal or injected conversions by using the CPU and DMA.

Each ADC supports a single CPU interruption request within a single DMA trigger signal. The interruption can be requested after the conversion of every channel. When using DMA, the conversion result is handled and stored into DMA registers, for the data to be transferred through DMA or host access. The CPU directly links to the DMA via the ADC host interface.

**Working flow**
4.1.1. **ADC DMA operation**

Conversion Data of any channel can be transferred from register to system memory via (DMA) Once enabled, by writing 1 to DMAE[DMAEN].

The on-chip DMA controller can get DMA request after the conversion of every channel by setting the respective bit in the DMAR0, DMAR1, and DMAR2 registers.

The DMA request can be cleared at different times in two modes:

- **MODE-1**: Clearing of DMA request on ACK from DMA controller (DMAE[DCLR] = 0)
- **MODE-2**: Clearing of DMA request on Read to data registers (DMAE[DCLR] = 1)

The following figures show the operation of DMA in two modes (cycle counts are typical values).
Figure 13. **DMA operation Mode-1** (DMAE[DCLR] = 0)

Figure 14. **DMA operation Mode-2** (DMAE[DCLR] = 1)
4.2. **Hardware triggers**

In this chip, hardware trigger signals can be provided from BCTU and TRGMUX outputs. This feature enables synchronous conversion of two independent ADC instances in parallel.

4.2.1. **BCTU triggering**

BCTU triggering scheme is the default and suggested hardware trigger method for the ADC. All the ADC instances are triggered from the BCTU. The BCTU provides channel conversion commands to the ADC, including the channel information. In addition, the ADC provides the conversion result back to the BCTU, including all the channel information.

The BCTU gives trigger pulse to all the 72 channels of ADC to initiate a conversion based on BCTU channel number.

When using the BCTU as trigger, the BCTU-ADC result register is as follow:

- The ADC result registers (ADCx_PCDRn[CDATA], ADCx_ICDRn[CDATA] and ADCx_ECDRn[CDATA]) store 15 bit conversion data.
- The BCTU ADC registers store 15-bit conversion data (BCTU_ADC0DR[ADC0_DATA], BCTU_ADC1DR[ADC1_DATA] and BCTU_ADC2DR[ADC2_DATA]).

The BCTU interface enhances ADC's injected conversion capability. It contains control inputs to select the channels to be converted from the appropriate event configuration register. The following figure shows the interface.

```
Figure 15. ADC Body Cross-triggering Unit (BCTU) trigger
```

The BCTU generates a trigger (bctu_trigger) and a channel number (bctu_numchannel) to be converted. A single channel is converted for each request. After performing the conversion, ADC returns the result on the (bctu_dataout) bus together with two output signals named (bctu_nextcmd) and (bctu_push). The assertion of signal (bctu_nextcmd) means ADC is ready to accept next trigger from BCTU.

The (bctu_push) signal is asserted at the end of conversion, meaning that conversion is finished and the conversion result available at output (bctu_dataout) is valid.
The conversion result is also saved in the corresponding channel's data register and it is compared with analog watchdog thresholds if requested.

The signals \((bctu\_trigger, \ bctu\_nextcmd)\) and \((bctu\_push)\) are all of type single cycle active high pulse in the ADC clock domain.

The channel number provided from BCTU must be valid when \((bctu\_trigger)\) is active high. The result data from ADC is valid with \((bctu\_push)\) high.

The BCTU interface has two modes of operation:

- Trigger
- Control

To enable the BCTU interface, program \(BCTUEN\). The operating mode (Trigger or Control) can be fixed or programmable. See the device reference manual in chapters “57.3.5.1 BCTU Trigger mode” and “57.3.5.2 BCTU Control mode” for more information.

**Working flow**

![ADC triggering via BCTU working flow](image)

**4.2.2. TRGMUX triggering**

In this chip, hardware trigger signals can be provided from TRGMUX outputs. 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 S32K3xx, 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 TRGMUX output numbers 0, 1, 2 [ADC_0] 4, 5, 6 [ADC_0] and 8, 9, 10 [ADC_2] for normal Conversion, Injected Conversion and Normal Conversion Sync Pulse, respectively. For better understanding, refer to the “Table 266. Hardware triggers” in the device reference manual.

If ADC is in Idle state (that is, no conversion phase ongoing and the MCR[PWDN] and MCR[ACKO] fields are 0) and the MCR[XSTRTEN] is set, an event on the external start signal causes ADC to start either normal or injected conversion operation. The MCR[NSTART]/MCR[JSTART] field is automatically set respectively.

The normal conversion sync pulse is used with normal conversion trigger to synchronize the start timing of normal conversion between multiple ADC instances.

**Working flow**

![ADC triggering via TRGMUX working flow](image)

**Figure 17.** ADC triggering via TRGMUX working flow

### 5. SAR ADC use cases

#### 5.1. SAR ADC- ADC calibration use case

In this example a calibration process is performed for the Analog-to-digital converter instance 1. A known reference voltage is sampled and converted under controlled conditions to determine the correction values (offset, gain, and capacitor mismatch)

**NOTE**

Calibration should be done after each reset and whenever required in run time operation.
Steps to start a calibration:

1. Configure CALBISTREG (sample time, average enable)
2. Set TEST_EN to start calibration
3. Poll BUSY bit for end of calibration
4. Check status bit for success or fail of calibration
5. A known reference voltage is sampled and converted under controlled conditions to determine the correction values (offset, gain, and capacitor mismatch)
5.1.1. **SAR ADC: Implementation ADC calibration example code**

```c
/* Perform Calibration */

/* 1. Configure the ADC operating clock for 40MHz operation (program ADCLKSEL=0b */
/* for 80 MHz system clock) */
ADC_1->MCR |= ADC_MCR_ADCLKSEL(1); /* AD_clk frequency is half bus clock frequency */
/* 2. Bring ADC from power down state to active conversion (program PWDN =0b) */
ADC_1->MCR &= ~ADC_MCR_PWDN_MASK; /* Resetting this bit will start ADC transition to IDLE mode */
//while ((ADC_0->MSR & ADC_MSR_ADCSTATUS_MASK )!= 0){}
/* 3. Configure the Calibration BIST Control and status register (CALBISTREG) for */
/* TEST conditions. The default values are set for maximum accuracy (recommended). */

ADC_1->CALBISTREG = (ADC_CALBISTREG_RESN(0) |
                   ADC_CALBISTREG_CALSTFUL_MASK | /* RESN=0: 14-bit resolution selected */
                   ADC_CALBISTREG_TSAMP(3) | /* TSAMP=3: 32 cycle of ADC clk for Sample period in Calibration process */
                   ADC_CALBISTREG_NR_SMPL(3))| /* Enable full range (conversions start from bit 15) */
ADC_CALBISTREG_AVG_EN_MASK | /* ADC Calibration Result Average feature enabled */
/* 4. Start calibration (program TEST_EN =1b), calibration start immediately. */
ADC_CALBISTREG_TEST_EN_MASK; /* Enable the Calibration (self clearing) */
/* 5. Poll the status of C_T_BUSY for 0. (wait until it becomes '0') */
while( (ADC_1->CALBISTREG & ADC_CALBISTREG_C_T_BUSY_MASK) != 0 ) {} 
/* 6. Check the TEST_FAIL to know the final status. If '1' then calibration failed. */
if ((ADC_1->CALBISTREG & ADC_CALBISTREG_TEST_FAIL_MASK )!=0 )
{

SIUL2->GPDO50 |= SIUL2_GPDO50_PDO_n_MASK; /* Drive high on PTB 18 to turn-on LED D33 */
   while( 1 ) {} /* Trap CPU due to ADC calibration failure */
}
/* 7. Check the status of CALIBRTD bit. If calibration is successful this bit will be '1'. */
if ((ADC_1->MSR & ADC_MSR_CALIBRTD_MASK ) == 0 )
{
SIUL2->GPDO50 |= SIUL2_GPDO50_PDO_n_MASK; /* Drive high on PTB 18 to turn-on LED D33 */
   while( 1 ) {} /* Trap CPU due to ADC calibration failure */
}
```

---

ADC Guidelines Spec and Configuration, Rev. 0, 11/2021

NXP Semiconductors
5.2. **SAR ADC Clock configuration use case**

In this example a clock configuration is performed for the Analog-to-digital converter instance 0. The ADC_0 MODULE_CLK is from M7_CORE_CLK (i.e. 160MHz).

The prescaler can be bypassed while using FIRC as a source. The prescaler should be controlled such that the clock frequency at analog block is less than or equal to 80 MHz.

The frequency of the conversion clock has to be within the limits defined in the component data sheet. The minimum speed of operation of ADC analog block is 6 MHz (Using FIRC/2 as system clock source, MC_CGM.MUX_0_DC_O[DIV] as 1'b1 and using ADC prescaler. However it is important to consider the ADC results will be degraded.

The figure below shows the ADC clock configuration most important paths.

![Fig 18. ADC clock configuration working flow](image)

<table>
<thead>
<tr>
<th>Module</th>
<th>MC_ME PRTNx</th>
<th>MC_ME COF by</th>
<th>Slot #</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_0</td>
<td>PRTN0</td>
<td>COFB1</td>
<td>58</td>
</tr>
<tr>
<td>ADC_1</td>
<td>PRTN0</td>
<td>COFB1</td>
<td>59</td>
</tr>
<tr>
<td>ADC_2</td>
<td>PRTN1</td>
<td>COFB1</td>
<td>58</td>
</tr>
</tbody>
</table>

5.2.1. **SAR ADC implementation Clock configuration example code for ADC_0**

```c
/* ADC0 Clocking */
/* Configure Clock for Analog-to-digital converter 0 */
/* ADC_0 MODULE_CLK is from M7_CORE_CLK (i.e. 160MHz) */
MC_ME->PRTN0_COFB1_CLKEN |= MC_ME_PRTN0_COFB1_CLKEN_REQ58(1); /* REQ58: Analog-to-digital converter 0 */
MC_ME.PRTN0_PCONF.B.PCE = 1; /* PCE=1: Enable the clock to Partition #0 */
MC_ME->PRTN0_PUPD |= MC_ME_PRTN0_PUPD_PCU_D_MASK; /* PCUD=1: Trigger the hardware process */
MC_ME->CTL KEY = 0x5AF0; /* Enter key */
MC_ME->CTL KEY = 0x555F; /* Enter inverted key */
while(!(MC_ME->PRTN0_COFB1_STAT & MC_ME_PRTN0_COFB1_STAT_BLOCK58_MASK)) { }
/* Wait until ADC_0 clock is running */
/* ADC0 Set-up */
ADC_0->MCR |= ADC_MCR_PWDN_MASK; /* Power down before initialization */
/* the analog module is requested to enter Power Down mode */
power-down mode */
ADC_0->MCR |= ADC_MCR_OWREN MASK; /* Enable overwriting older conversion */
```

NXP Semiconductors

ADC Guidelines Spec and Configuration, Rev. 0, 11/2021
5.3. **SAR ADC periodic chain conversions use case**

In this example an periodic chain conversion example is configured and performed for the Analog-to-digital converter instance 0.

In the code example the Sequentially Channel 54 and Channel 55 are enabled to perform for normal chain conversion at every 10ms triggered by PIT. In One-Shot operation mode.

Take in consideration, for this usage case, the conversion data is written right aligned (i.e. bits 14 to 0) and the configuration setup has been enabled to selects for the rising edge for the external trigger to start the ADC normal chain conversion.

The following figure shows the most important paths and register configurations for the ADC periodic chain conversion usage case.

![Diagram of ADC periodic chain conversions](image)

**Figure 19.** Periodic Chain Conversions working flow
5.3.1. SAR ADC implementation periodic chain conversions example Code for ADC_0

```c
/* Configure normal chain conversion for ch54 and ch55 by hardware trigger */
ADC_0->CALBISTREG |= ADC_CALBISTREG_RESN(1); /* RESN=1: 12-bit resolution selected */
ADC_0->CTR0 |= ADC_CTR0_INPSAMP(0x20); /* Set the sampling phase duration */
ADC_0->MCR |= ADC_MCR_MODE(0); /* MODE=0: One-Shot operation mode */
ADC_0->MCR |= ADC_MCR_WLSIDE(0); /* WLSIDE=0: The conversion data is written right aligned (bits 14 to 0) */
ADC_0->MCR |= ADC_MCR_TRGEN_MASK; /* TRGEN=1: Enables the external trigger to start a conversion */
ADC_0->MCR |= ADC_MCR_EDGE(0); /* EDGE=1: selects the rising edge for the external trigger */
ADC_0->IMR |= ADC_IMR_MSKECH_MASK; /* Enable end of Normal Chain conversion interrupt */
/* Enable ch54 and ch55 for normal chain conversion */
ADC_0->NOMR1 |= ADC_NOMR1_CH54_MASK;
ADC_0->NOMR1 |= ADC_NOMR1_CH55_MASK;
Enable_Interrupt( ADC_0_IRQHandler );
ADC_0->MCR &= ~ADC_MCR_PWDN_MASK; /* Resetting this bit will start ADC transition to IDLE mode */
```

```c
void ADC_0_Handler(void)
{
    if(( ADC_1->ISR & ADC_ISR_ECH_MASK ) != 0 ) /* Interrupt by ECH? */
    {
        EMIOS_1->CH.UC[19].B = ( uint16_t)( ADC_1->ICDR[54] & 0x7FFF ); /* PWM trailing edge count value */
        EMIOS_1->CH.UC[27].B = ( uint16_t)( ADC_1->ICDR[55] & 0x7FFF ); /* PWM trailing edge count value */
        ADC_1->ISR |= ADC_ISR_ECH_MASK; /* Write 1 to clear ECH flag */
    }
}
```

**NOTE**

Results in ADC modules are 15-bit width regardless of “RESN” configuration.
5.4. **SAR ADC injected conversions use case**

In this example an injected conversion example is configured and performed for the Analog-to-digital converter instance 0.

In the code example the Channel 55 is converted in the middle of an a consecutive Channel 54 normal chain conversion.

The following figure shows the most important paths and register configurations for the ADC injected conversion usage case.

**Working flow**

![Injected Conversions working flow](image)

*Figure 20. Injected Conversions working flow*
5.4.1. **SAR ADC implementation injected conversions example code**

```c
/* Configure normal chain conversion for ch54 and ch55 by CPU software trigger */
ADC_0->CALBISTREG |= ADC_CALBISTREG_RESN(1); /* RESN=1: 12-bit resolution selected */
ADC_0->CTR0 |= ADC_CTR0_INPSAMP(0x20); /* Set the sampling phase duration */
ADC_0->MCR  |= ADC_MCR_MODE(1);    /* MODE=1: Scan operation mode */
ADC_0->MCR  |= ADC_MCR_WLSIDE(0);  /* WLSIDE=0: The conversion data is written right aligned (bits 14 to 0) */
ADC_0->MCR  |= ADC_MCR_TRGEN_MASK(0); /* TRGEN=0: The External trigger is Disabled*/

ADC_0->IMR |= ADC_IMR_MSKECH_MASK; /* Enable end of Normal Chain conversion interrupt */
ADC_0->IMR |= ADC_IMR_MSKJECH_MASK; /* Enable end of Injected Chain conversion interrupt */
/* Enable ch54 for normal chain conversion */
ADC_0->NCMR1 |= ADC_NCMR1_CH54_MASK; // CH54 ADC0_//ADC1_S10
/* Enable ch55 for injected chain conversion */
ADC_0->NCMR1 |= ADC_JCMR1_CH55_MASK; // CH55 ADC0_//ADC1_S10
ADC_0->MCR &= ~ADC_MCR_PWDN_MASK; /* Resetting this bit will start ADC transition to IDLE mode */

void ADC_0_Handler(void)
{
    if(( ADC_1->ISR & ADC_ISR_ECH_MASK) != 0) /* Interrupt by ECH? */
    {
        EMIOS_1->CH.UC[19].B = ( uint16_t)( ADC_1->ICDR[54] & 0x7FFF ); /* PWM trailing edge count value */
        ADC_1->ISR |= ADC_ISR_ECH_MASK; /* Write 1 to clear ECH flag */
    }
    if(( ADC_1->ISR & ADC_ISR_ECH_MASK) != 0) /* Interrupt by ECH? */
    {
        EMIOS_1->CH.UC[27].B = ( uint16_t)( ADC_1->ICDR[55] & 0x7FFF ); /* PWM trailing edge count value */
        ADC_1->ISR |= ADC_ISR_ECH_MASK; /* Write 1 to clear ECH flag */
    }

    NOTE

    Results in ADC modules are 15-bit width regardless of “RESN” configuration.
```
Appendix A. Why to perform ADC calibration

The capacitor size matching and amplifier gain will vary during manufacturing and various run-time environmental influences, and the conversion result contains errors.

To reduce the error, the ADC provides calibration feature.

A known reference voltage is sampled and converted to determine the calibration values for offset, gain, and capacitor mismatch.

These calibration values (except gain calibration) are used in a result post-processing step to reduce the error. The gain calibration is used during sample phase to define the additional charge to be loaded in order to compensate for gain failure.

Calibration needs to be run after every power-up reset and whenever required in run time operation. It is also recommended to run calibration if the operating conditions (particularly VrefH) changes.
Appendix B. ADC conversion time

The ADC total conversion time, in terms of the module clock cycles, is calculated using the following equation for normal and injected conversions:

\[
\text{Total\_conversion\_time} = \left\{ \left( \text{PST} + \text{ST} + \text{CT} + \text{DP} \right) \times \text{chain\_length} \right\} + \text{TPT} \times \text{TAD\_clk}
\]

\textit{Eqn. 7 Total ADC conversion time}

Where:

- TCT: Total ADC Conversion Time
- PST: Pre-Sample phase time (configurable)
- ST: Sample phase time (configurable)
- CT: Compare phase Time (fixed – 4 AD_CLK cycle per bit)
- DP: Data Processing Time (fixed – 2 AD_CLK cycle)
- TPT: Trigger Processing time (fixed – 2 IPG_CLK cycle, AD_CLK=IPG_CLK/2)
- TAD_clk: ADC operating clock

B.1 ADC Conversion Time Example

The SAR ADC conditions are as follows:

- No pre-sampling is selected
- Sample time kept default (22 cycles)
- Conversion time (4 cycles per bit)
- ADC resolution 12 bit + 1 bit for a special capacitor (CS)
- 3 channels are programmed in NCMRn register (chain=3)
- SAR-controller clock is equal to ipg_clock (80 MHz, T = 12.5 ns)

The total time taken for the 3 conversions is:

\[
\text{Total\_conversion\_time} = \left( \left( 0 + 22 + (4 \times 13) + 2 \right) \times 3 \right) + 1 \times \text{TAD\_clk}
\]

\[
= 229 \text{ cycles} \times 12.5 \text{ ns}
\]

\[
\approx 2.862 \mu\text{s}
\]

\textit{Eqn. 8 Total ADC conversion time example}
Appendix C. ADC Self-test

For safety applications, it is important to check at regular intervals if the ADC is operating correctly. The following test algorithms have been implemented:

**Supply Self-Test (Algorithm S):** It includes the conversion of the bandgap and VREF voltages. The supply test conversions must be an atomic operation (that is, all supply algorithm conversions must be performed one after another with no functional conversions in between).

- Step 0 - Supply self-test for band gap voltage
- Step 1 - Supply self-test for VRH
- Step 2 - Supply self-test for VRH

**Capacitive Self-Test (Algorithm C):** It includes a sequence of test steps per definition of Algorithm C which executes the capacitive matrix of the CDAC used for sampling and conversion.

Step 0–11 – Capacitive self-test.

![Self-Test Configuration Register](image)

Figure 22. **Self-Test Configuration Register**
Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any warranty under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFAREMOBILE, MIGLO, NTAG, ROADLINK, SMARTTX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltVec, C 5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C Ware, the Energy Efficient Solutions logo, Kinetics, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Hybrid, Airlast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. Arm, AMBA, Arm Powered, Artisan, Cortex, Jazelle, Keil, SecurCore, Thumb, TrustZone, and μVision are registered trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. Arm7, Arm9, Arm11, bigLITTLE, CoreLink, CoreSight, DesignStart, Mali, Mbed, NEON, POP, Sensinode, Socrates, ULINK and Versatile are trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2021 NXP B.V.