AN3137
Application note

Analog-to-digital converter on STM8L and STM8AL devices: description and precision improvement techniques

Introduction

This application note describes the 12-bit analog-to-digital converter (ADC) embedded in the STM8L and STM8AL family microcontrollers. It offers a selection of hardware and software methods for improving the precision of this ADC. It is divided into the following sections:

■ The first section explains the internal design principle of the STM8L/STM8AL ADC.

■ The second section describes the ADC operation from the programmer's point of view.

■ The final section lists the main types of errors that occur in ADCs and their sources. It describes some hardware and software methods for minimizing these errors.

Firmware examples (source codes) are provided with this application note showing how to implement STM8L/STM8AL ADC routines for minimizing measurement errors.

This document applies to the products listed in *Table 1*.

Table 1.   Applicable products

Product family	Part numbers
Microcontrollers	– STM8L05xxx – STM8L151C2/K2/G2/F2, STM8L151C3/K3/G3/F3 – STM8L151x4, STM8L151x6, STM8L151x8 – STM8L152x4, STM8L152x6, STM8L152x8 – STM8L162R8, STM8L162M8 – STM8AL313x, STM8AL314x, STM8AL316x – STM8AL3L4x, STM8AL3L6x

Contents

List of tables

List of figures

1 STM8L/STM8AL ADC internal hardware

STM8L/STM8AL microcontrollers can include an ADC. This ADC peripheral is used to measure analog signals from external signal sources.

1.1 ADC principle

The ADC in STM8L/STM8AL microcontrollers is a switched-capacitor type. This ADC type uses the SAR (successive approximation register) principle, by which the conversion is performed in several steps. The number of conversion steps is equal to the number of bits in the ADC converter.

The following figures (*Figure 1* to *Figure 6*) explain the principle of ADC operation. They are based on a 10-bit ADC example (whereas STM8L/STM8AL devices contain a 12-bit ADC). The example given below shows only the first steps of approximation but the process continues till the LSB is reached.

Figure 1. Basic schematic of switched-capacitor 10-bit ADC
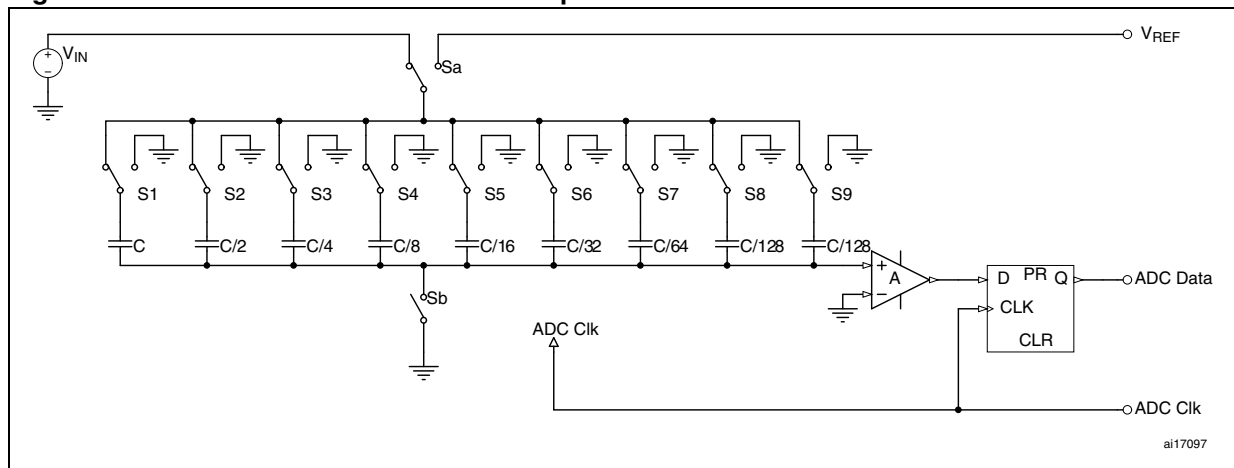
1. Basic ADC schematic with digital output.

Figure 2. **Sample mode**
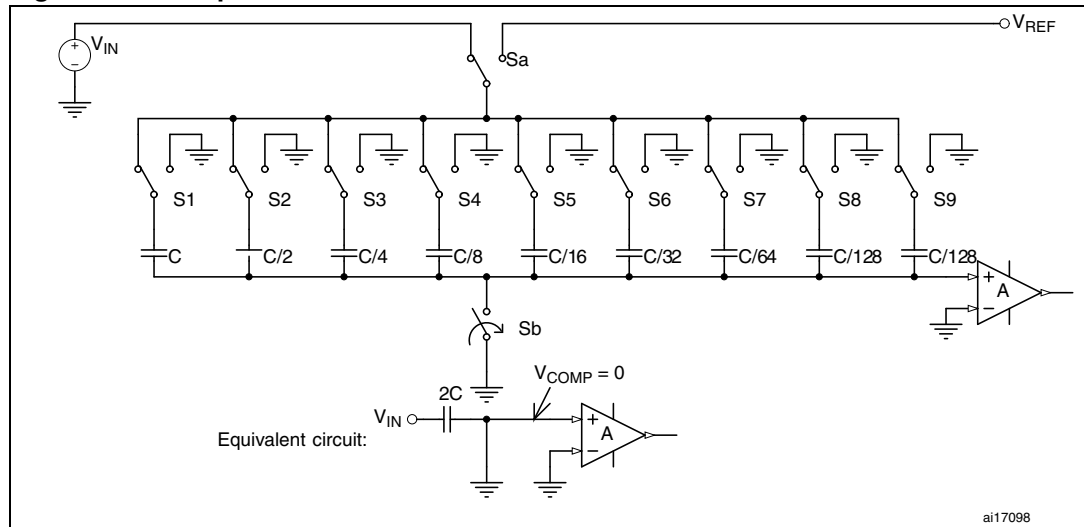
1. Sample state - capacitors are charging to V_{IN} voltage. Sa switch to V_{IN}, Sb switch closed.

Figure 3. **Hold mode**
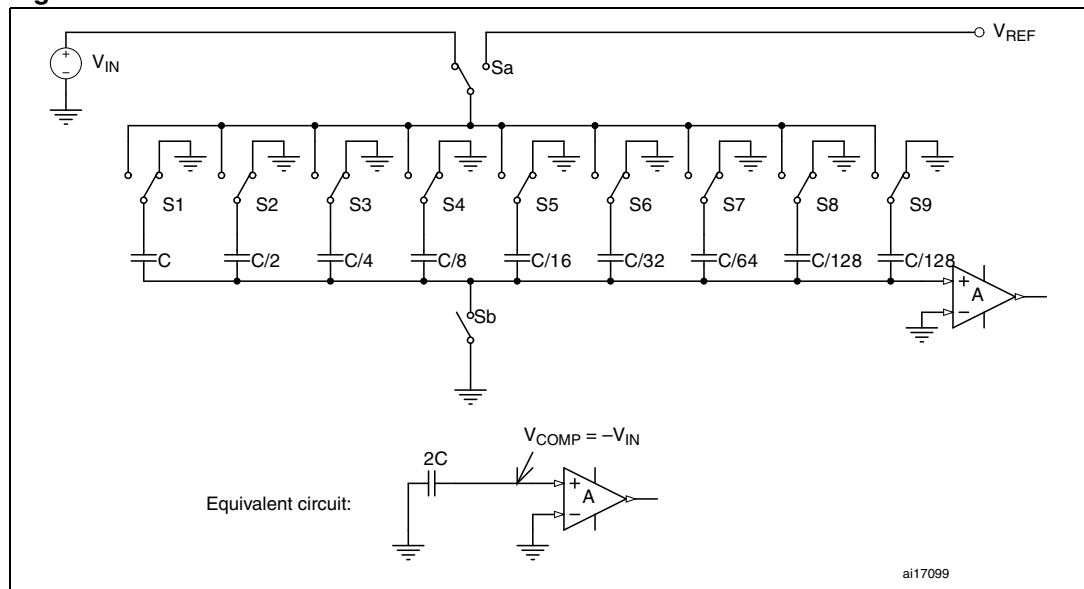
1. Hold mode - input is disconnected - capacitors hold input voltage. Sb switch is open, then S1-S9 switched to ground.

Figure 4.   Step 1: if MSB = 0, compare with $V_{REF}/2$
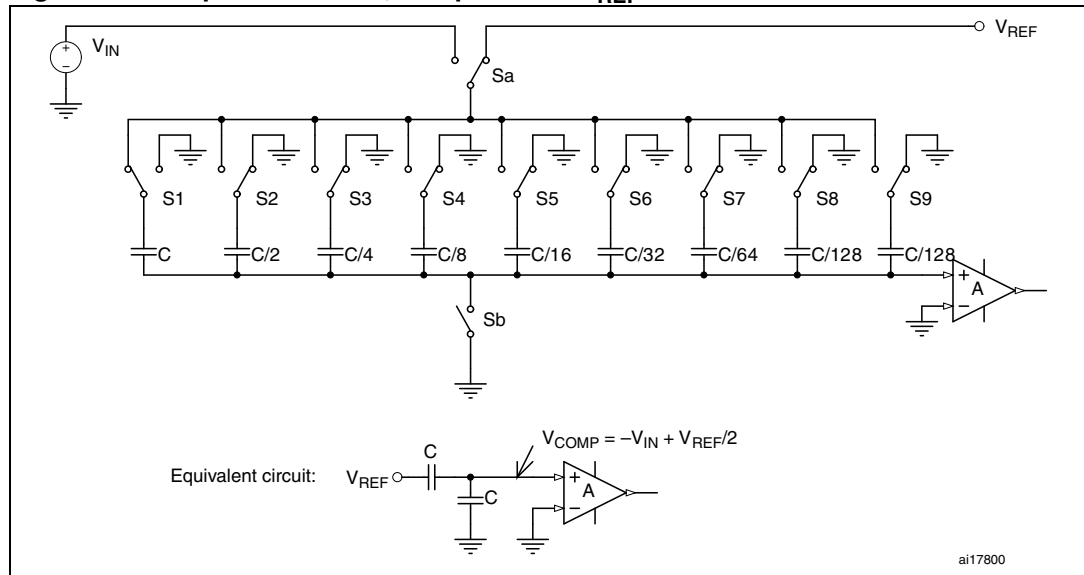
1.   First approximation step. Sa switched to V_{REF} and S1 switched to V_{REF}.

Figure 5.   Step 2: if MSB = 1, then compare with ¾ V_{REF}
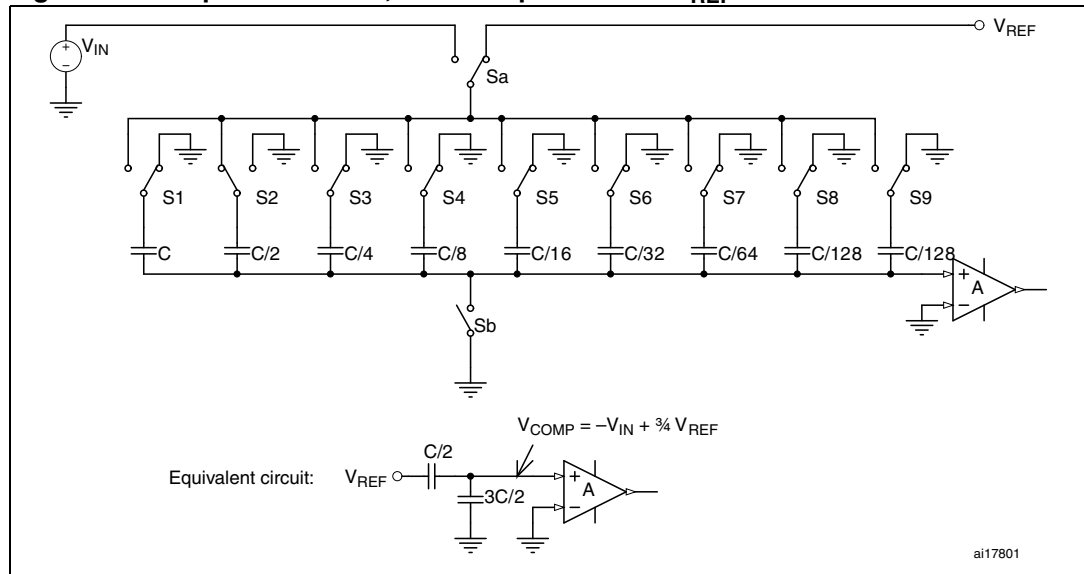
1.   Compare with ¾V_{REF}; if MSB =1. S1 switched back to ground. S2 switched to V_{REF}.

Figure 6. Step 2: if MSB = 0, then compare with ¼ V$_{REF}$
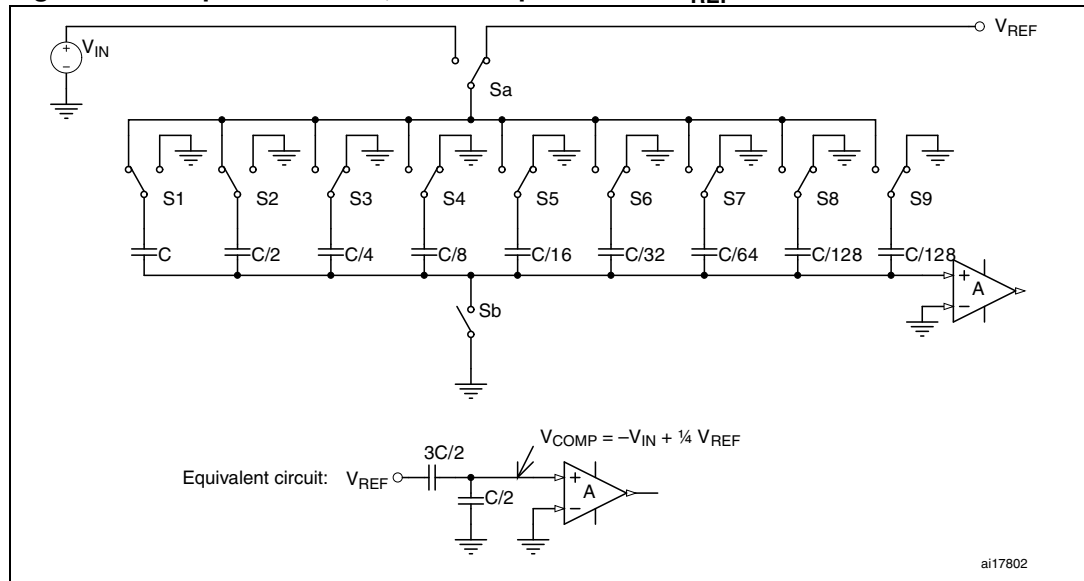
1. Compare with ¼ V$_{REF}$; if MSB =0. S1 remained switched to ground. S2 switched to V$_{REF}$.

1.2 ADC resolution

ADC resolution can be set in 4 steps: 6-bit, 8-bit, 10-bit or 12-bit resolution. Decreasing the resolution speeds up the conversion time.

1.3 ADC clock, sampling time, speed

The ADC is driven by a clock derived from the MCU system clock through a divider. The divider divides the master clock by a factor 1 or 2. The maximum ADC clock is 16 MHz and the minimum is 320 kHz.

Each conversion step (which consists of switching the capacitor network, comparing results, and storing them to a register bit, as described in *Figure 4* to *Figure 6*) is performed in one ADC clock cycle. Consequently, a 12-bit conversion takes 12 cycles. The sampling period (*Figure 2*) has a programmable range from 4 to 384 clock cycles (charging input sampling capacitors).

The shortest conversion time for a 12-bit resolution is 1 μs (4 sampling clocks + 12 approximation clocks on a 16 MHz ADC clock). It is possible to further reduce the conversion time by decreasing the ADC resolution (to 6 or 8 or 10 bits).

Once enabled, the ADC analog peripheral needs a t$_{WKUP}$ stabilization time before starting the conversion. Refer to the datasheet of your device to know the exact t$_{WKUP}$ value (usually 3 μs).

The user must implement this stabilization delay in the STM8L/STM8AL firmware.

1.4 Power supply range

The ADC is powered from V_{DDA}/V_{SSA} pins (usually connected to V_{DD}/V_{SS} - depending on packages). The voltage range for a proper ADC operation is from 1.8 V to 3.6 V (the digital part operates from 1.65 V). See the electrical characteristics in the STM8L/STM8AL device datasheets for further details.

1.5 ADC reference voltage

The ADC reference voltage is either internally connected to the analog power supply pins or connected to the V_{REF+} external pin, where an external reference voltage source can be connected. This reference voltage connection option depends on the STM8L/STM8AL package and device types:

● In case V_{DDA} supply voltage is greater than 2.4 V, the V_{REF+} pin can be connected to a voltage different from V_{DDA}. V_{REF+} must then be greater or equal to 2.4 V and less or equal to V_{DDA} ($V_{DDA} \geq V_{REF+} \geq 2.4$ V).

● In case V_{DDA} supply voltage is less than 2.4 V, V_{REF+} must have the same voltage as V_{DDA}.

As the ADC reference voltage highly influences ADC precision, these voltage properties are an important characteristic to be taken into account in the application design (stability, noise,...).

The ADC can convert the V_{REFINT} internal reference voltage (on ADC channel 25) to perform calibration (especially when V_{DD}/V_{SS} are used as reference voltage). See *Section 2.3.2: Correction to VDD/VREF stability* for more information.

1.6 Input analog multiplexer

The ADC has an input multiplexer which is used to select one of the STM8L/STM8AL input pins as the analog input to the ADC. 25 external channels (on I/O pins) and 2 internal channels (internal reference voltage and temperature sensor) are available.

The Schmitt triggers can be disabled on selected ADC input pins (mainly used in digital I/O pin configuration). The STM8L/STM8AL power consumption can then decrease but the digital inputs on the corresponding I/O pins will not work.

1.6.1 Internal channels

Internal analog channels (internal reference voltage and temperature sensor) can be used for calibration purposes (or for temperature measurement). These analog sources must be enabled before using them (they are disabled by default so as to save power).

1.6.2 Fast/slow channels

Analog channels are divided into two groups: a fast channel sequence and a slow channel sequence.

Each channel sequence has an own individual setting for sampling time (two different sampling times can be set: one for each channel sequence).This feature is useful when some measured voltage sources require a longer sampling time (e.g. source with a high internal impedance). Refer to *Section 2.3.3: Measurement of sources with high internal impedance*).

1.7　Conversion triggers

ADC conversion can start when a trigger occurs. Three main sources of trigger can be selected:

● software trigger (writing to one register bit)

● timer trigger (if a timer event occurs)

● external trigger (change of the state of a dedicated I/O pin)

1.8　Analog watchdog feature

This function is used to monitor converted values on selected channels. If the converted value on a given ADC input channel is outside the reference voltage threshold, the analog watchdog flag (AWD) bit is set and can generate an interrupt.

1.9　Interrupt generation

The ADC can generate an interrupt if some events occur. This simplifies the programming and speeds up the ADC service routines.

The interrupt sources are the following ones:

● end of conversion (when an ADC data register is filled with a new value)

● analog watchdog event (if the ADC converted value is outside a given voltage range)

● overrun event (when, after a conversion, a second conversion is complete and the DMA has not read the first conversion value yet)

1.10　DMA transfer

The ADC can be programmed to use the built-in DMA to transfer converted ADC data to the memory. This speeds up the data collection (instead of ADC data register polling by the CPU, the DMA transfer runs in the background) when the CPU is performing another operation. Refer to the STM8L05xx, STM8L15xx, STM8L162x, STM8AL31xx and STM8AL3Lxxx microcontroller family reference manual (RM0031) for a description of DMA operation. Using the DMA transfer is especially useful in long data array collection (voice, signal processing,...).

1.11　Sampling modes

The ADC can operate in three modes which mainly differ in the way converted input channels are managed:

● single mode (one conversion and then ADC stops)

● continuous mode (ADC continues conversion to the next channel)

● scan mode (ADC converts a set of selected channels to the memory)

These modes are described in *Section 2: STM8L/STM8AL ADC usage/programming*.

2 STM8L/STM8AL ADC usage/programming

2.1 Sampling mode overview and usage

2.1.1 Single mode

In this mode, only one input channel must be selected in the ADC_SQRx registers (if more channels are selected, the highest selected channel is measured). In addition, the DMAOFF bit must be set in the ADC_SQR1 register (to disable DMA transfer). The input channel is then converted and the ADC conversion stops (one simple conversion). The converted value is stored into the ADC_DR data register. An interrupt (EOC) can be generated after the end of conversion.

Note: If the DMAOFF bit in the ADC_SQR1 register is reset (and if DMA is properly programmed) the conversion is then performed in scan mode. Refer to Section 2.1.3: Scan mode.

2.1.2 Continuous mode

In this mode, the ADC does not stop after conversion but continues to the next channel in the selected channel sequence. Conversions continue until the CONT bit and the ADON bits are set and the converted values are transferred through the DMA to the RAM or EEPROM memory buffer. As the EOC interrupt is generated only after the end of the conversion of the selected channel sequence, the result of each channel conversion cannot be read from the ADC_DR register.

To save the conversion of each channel in a memory (RAM or EEPROM), the DMA must be used in peripheral to memory mode. If the CONT bit is reset during a conversion, the current selected channel sequence conversions end with the last selected channel) and then the ADC stops.

Note: The CONT bit must not be set again during a conversion.

2.1.3 Scan mode

This mode works automatically in continuous mode. Refer to *Section 2.1.2*. It can also be used in single mode by resetting the DMAOFF bit in the ADC_SQR1 register. Refer to *Section 2.1.1*.

In scan mode, the selected channel sequence is taken into account (like in single mode) and the selected channels are converted and transferred through DMA to memory (DMA should be properly programmed).

If the single mode is selected, the ADC converts the selected channel sequence and it is stopped after the last channel conversion. For the next selected channel sequence, a scan conversion is necessary to restart the ADC.

An interrupt (EOC) can be generated after the end of the channel sequence conversion.

2.2 Conversion modes

The ADC can operate in different modes depending on application requirements. The following sections list the most commonly used operating modes.

2.2.1 Single conversion

This conversion mode is usually used when the application needs to read occasionally some analog values (input voltage, temperature...).

2.2.2 Single conversion with trigger

This mode is used for the same applications as for the single conversion described in *Section 2.2.1* but the conversion start is triggered by a periodical interval or by an external event (collection of equidistant samples, external event measurement...).

2.2.3 One-shot channel sequence scan

This mode is a single scan mode (see *Section 2.1.3*). It is useful for collecting values from a sequence of channels in one time, for example, measuring a set of external sensors within equidistant intervals and storing them into a log file (measuring a set of sensors in one time). This mode can be combined with any trigger depending on application needs: software, timer or external trigger.

2.2.4 Continuous channel sequence scan

This mode continuously converts selected channels and stores data into memory. As the DMA is used, the CPU does not need to manage measurements, it only processes the data: it displays them and sends them to the communication line. This scan mode uses continuous scan mode. Refer to *Section 2.1.3*.

2.3 Specific modes using ADC features

2.3.1 Analog watchdog monitoring

By default, the ADC runs in continuous mode and the analog watchdog (enabled on one channel) monitors the converted values that are outside the reference voltage range defined in the ADC_HTRx and ADC_LTRx threshold registers). If the value becomes "insecure", then an interrupt occurs and the AWD flag bit is set.

In this mode, the CPU does not need to perform periodical ADC measurements. It only has to react on analog watchdog interrupts (there is no performance loss due to the ADC operation).

This mode is selected by setting a channel for the analog watchdog monitoring in the ADC_CR3 register (CHSEL bits). The detection of insecure values can be performed by monitoring the AWD bit (in the ADC_SR register) or by using the automatic interrupt through the analog watchdog (AWDIE bit set in ADC_CR1 register to enable interrupts).

2.3.2 Correction to V_{DD}/V_{REF} stability

The ADC usually uses V_{DD}/V_{SS} as reference voltage. As this voltage is not accurate, this method may result in incorrect measured values. By measuring the internal reference voltage, it is possible to perform corrections to measured values and obtain more precise measurements (post-processing of collected ADC data).

Correction to collected data can be done by using the following formula:

$$V_{IN} = \frac{D_{IN}}{D_{REFINT}} \times V_{REFINT}$$

where:

- V_{IN} = input voltage measured after correction
- D_{IN} = ADC data from input voltage channel
- V_{REFINT} = internal reference voltage (around 1.23 V - see device datasheet -)
- D_{REFINT} = ADC data from internal reference voltage channel

2.3.3 Measurement of sources with high internal impedance

In this case, the ADC must be configured for a long sampling time. The internal sampling capacitor is charged to signal source voltage through a high signal source impedance. Sampling time must be selected depending on the source impedance and on the required signal bandwidth (the RC works as lowpass filter and long sampling time limits the bandwidth).

The required sampling time is given by the RC timing constant and the appropriate resolution.

Note: Exact calculations are explained in Section 3.5.1: External resistance design error on page 18. The internal sampling capacitance is defined in the device datasheet.

2.3.4 Low power mode with ADC enabled

To save as much power as possible while the ADC is running, the STM8L/STM8AL microcontroller families can be used in Wait mode.

In this case, make sure that the required minimum ADC clock (320 kHz - see device datasheet) is provided to ADC for a proper operation.

The ADC interrupt can wake up the CPU from wait modes.

Note: The ADC does not work in Halt, Active-halt, Low power run and Low power wait modes: it must be switched off before entering a low power mode not supported by this peripheral.

The ADC uses the internal reference voltage, so the user must make sure this reference voltage is woken up before enabling the ADC after an exit from Halt or Active-halt mode. When enabled, the ADC needs a stabilization time (t_{WKUP}) before starting the first conversion (refer to the device datasheet).

Example: after an RTC wakeup (alarm) from Halt mode, the ADC is first enabled. Then, the firmware waits for a t_{WKUP} time (3 µs) and the conversion can be started by setting the START bit in the ADC_CR1 register.

The ADC consumption (when the ADC is active and is converting) is given in the device datasheet (around 1400 µA). In order to save power, it is recommended to perform the ADC conversion at the highest possible speed, and then switch the ADC off.

3 ADC errors

3.1 Introduction

This section lists the main errors which have an effect on A/D conversion accuracy. These types of error occur in all A/D converters and conversion quality depends on eliminating them. Values for these errors are specified in the ADC characteristics section of the STM8L/STM8AL device datasheets. The datasheets also include sections describing sources of error or methods for minimizing them.

3.2 Linearity errors

3.2.1 Differential nonlinearity

Differential nonlinearity (DNL) shows how far a code is from a neighboring code. The distance is measured as a change in input voltage magnitude and then converted to LSBs. Good ADC performances are specified as "no missing codes". This means that if the input voltage is swept over its range, all output code combinations will appear at the converter output. A DNL error of <±1 LSB guarantees no missing codes. With a DNL equal to -1 LSB, the ADC does not guarantee to have no missing codes. With a DNL greater than -1, the device has missing codes.

Figure 7. DNL: no missing codes
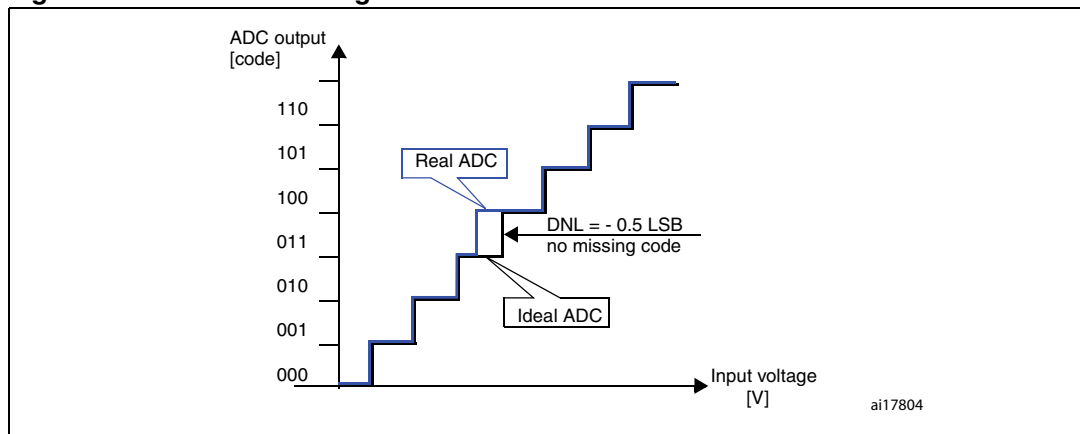
Figure 8.   DNL: missing code
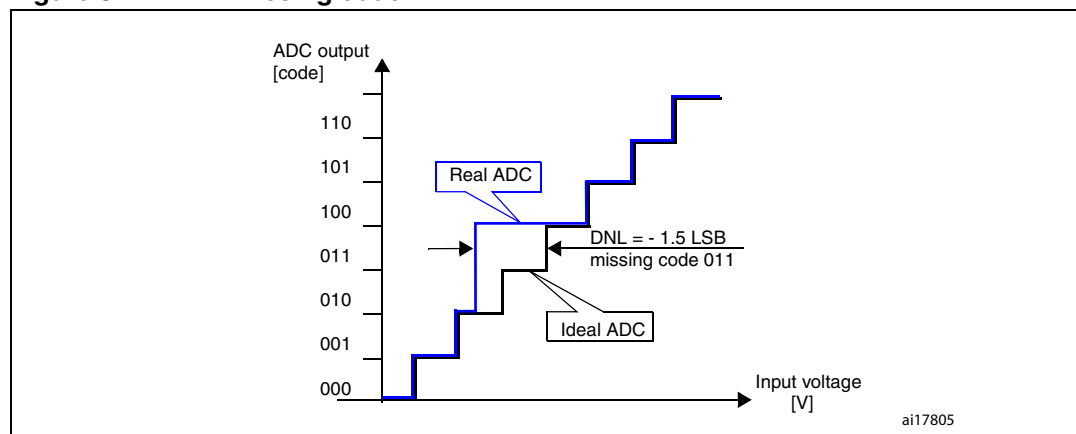
3.2.2   Integral nonlinearity

Integral nonlinearity (INL) is defined as the integral of the DNL errors. So, good INL guarantees good DNL. The INL error shows how far from the ideal transfer function value the measured converter result is. For example, an INL error of ±2LSB in a 12-bit system means the maximum nonlinearity error may be off by 2/4096 or 0.05%. Note that neither INL nor DNL errors can be calibrated or corrected easily.

3.3   Offset error

Offset and gain errors can easily be calibrated by the application firmware.

First, connect ADC input to V_{REF-}/V_{SS} (low potential of reference voltage) and perform a conversion. Then the conversion result represents the zero offset error. Then a gain adjustment is performed. A subsequent offset error calibration may be required.

3.4   Gain error

Gain error is defined as the full-scale error minus the offset error. Full-scale error is measured at the last ADC transition on the transfer-function curve and compared to the ideal ADC transfer function.

Gain error is easily corrected in firmware with this linear function:

$y = (k1/k2).x$

where k1 is the slope of the ideal transfer function and k2 is the slope of the measured transfer function.

Offset error and gain error can decrease dynamic range. For example this is observed if a full-scale input voltage is applied and the code obtained is 4092 instead of the ideal 4095 (for a 12-bit converter), or if the full-scale code 4095 appears with an input voltage lower than the full-scale voltage.

Figure 9. Offset and gain error
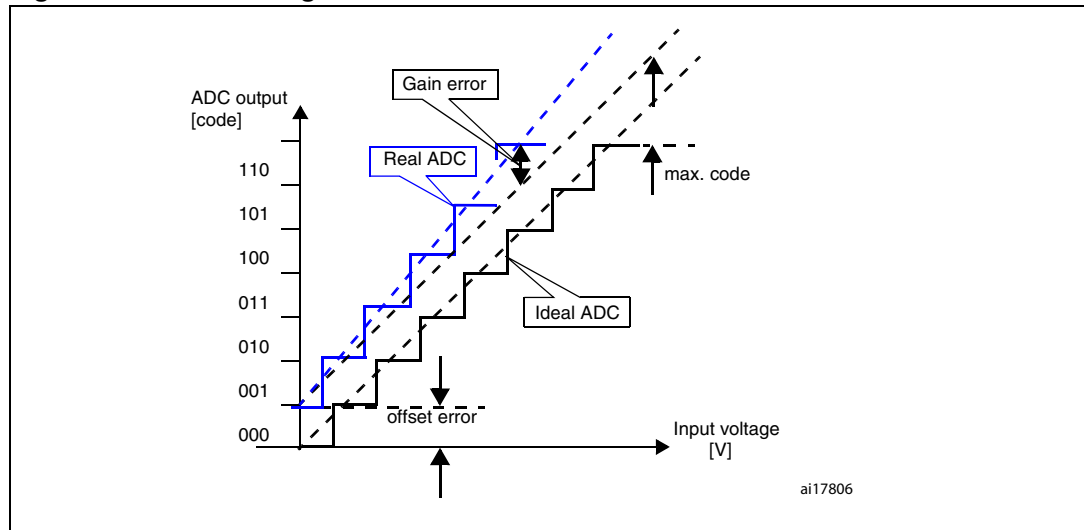
3.5 Hardware design errors

3.5.1 External resistance design error

The input multiplexer has a nonzero impedance (R_{mpx}). Sampling is done by switch which has also a nonzero impedance (R_{sw}). Both impedances, together with the sampling capacitance (C_{samp} = typ. 16 pF; see datasheet) and external signal source resistance (R_{ext}), create a low pass filter. Therefore the external signal source impedance (R_{ext}) must be designed so that this low pass filter cut-off frequency will not be too low in relation to the ADC sampling time (t_S = 4 ... 384 ADC clocks).

$$(R_{sw} + R_{mpx} + R_{ext}) \cdot C_{samp} \ll t_s$$

Figure 10. Analog input circuit
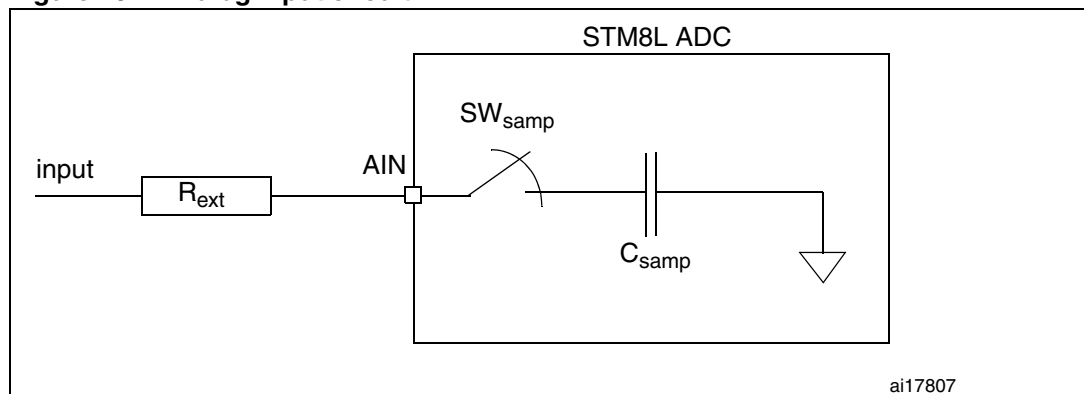
If the ADC converter has a 12-bit resolution (n=12) then the maximum precision is 1/2 LSB level (0.5/4096 from full range). So the acceptable error caused by charging C_{samp} through all resistors is 0.5/4096 (during t_S sampling time = 4 to 384 cycles of f_{ADC} clock). Then the maximum resistor value $R_{max}=R_{ext}+R_{sw}+R_{mpx}$ is obtained from the following equation:

$$\text{error} = \frac{(U_c - U_i)}{U_i} = e^{-\frac{\frac{t_S}{f_{ADC}}}{R_{max} \cdot C_{samp}}} = \frac{0.5}{n}$$

$$R_{max} = -\frac{t_S}{f_{ADC} \cdot C_{samp} \cdot \ln\frac{0.5}{n}}$$

For worst case: f_{ADC} = 16 MHz, 12-bit resolution (n = 12) required and sampling time = 4 clocks (t_S = 4), the maximum serial resistor is:

$$R_{max} = -\frac{4}{(16 \cdot 10^6) \cdot (16 \cdot 10^{-12}) \cdot \ln\frac{0.5}{4096}} \cong 1.7k\Omega$$

and the maximum external resistor is:

$$R_{ext} = R_{max} - (R_{sw} + R_{mpx}) \cong 1.7k\Omega - (cca.1.2k\Omega) \cong 500\Omega$$

The external parasitic PCB and package capacitance must also be taken into account. Therefore the maximum resistance of external signal source R_{ext} is lower, and in practice R_{ext} < 500 Ω is suggested.

In case of longer sampling times and reduced number of ADC clocks, better results can be obtained: for instance, sampling time can be extended to 384 cycles (instead of 4 cycles), which offers an allowed resistance higher than 50 kΩ (@ 16 MHz ADC clock). It is possible to further increase the allowed external resistance by decreasing the ADC clock frequency or selecting a lower resolution. Refer to the datasheet of your device to obtain the exact values of the RC parameters.

3.5.2   ADC reference voltage source

One of the biggest potential sources of errors in an ADC is the reference voltage. It is important to look at three reference voltage specifications: temperature drift, voltage noise, long term stability.

3.5.3   Temperature influence

Temperature has a big effect on ADC precision. Mainly two specifications are important: offset drift and gain drift. These errors can be compensated in microcontroller firmware. One method is to fully characterize the offset and gain drift and provide a lookup table in memory to correct measurement due to temperature change. The ADC in each MCU device must then be compensated individually - therefore this calibration takes additional cost and time. The second method is to recalibrate the ADC when the temperature change reaches given values.

3.5.4 AC performance

Usually ADC performance is good only when the spectrum of the input signal is limited from zero to half of the sampling frequency (Nyquist frequency). This is because an ADC design contains not only resistors but also capacitors (and parasitic RLC elements). Therefore frequency response degrades ADC performance if frequency increases. In case of AC measurement, the ADC frequency parameters must be studied; mainly signal-to-noise ratio (SNR), total harmonic distortion (THD) and spurious-free dynamic range (SFDR).

4 Methods for precision improvement

4.1 Introduction

ADC converter precision is fixed by its principle, design and implementation but it can also be improved by several hardware and/or software methods. Hardware methods improve ADC results by changing the environment around the ADC (PCB design, RC filters...) to obtain more accurate ADC output. Software methods try to improve the raw ADC results by post-processing methods. Some CPU computation power and time are required to perform these methods, which can be implemented in the microcontroller firmware.

4.2 Hardware methods

● Analog zooming (use appropriate V_{REF} voltage):
   – select reference voltage between input signal ranges
   – gives full ADC range - minimum voltage per bit
● White noise added to measured signal:
   – wobbling of input signal over several bits gives the opportunity to use averaging (if input signal is very stable)
   – white noise gives independence from sampling frequency
● Good hardware design:
   – grounding
   – reference voltage filtering
   – supply filtering
   – preamplifier usage
   – frequency independence
   – …

4.2.1 Analog zooming

Analog zooming is a method which improves precision by properly selecting the reference voltage. The reference voltage is selected in the expected range of the signal to be measured.

If the measured signal has an offset, then the reference voltage should also have a similar offset. If the measured signal has defined maximum amplitude, then the reference voltage should also have a similar maximum value. By matching this reference voltage to the measurement signal range, we obtain the maximum possible resolution using the full A/D converter output range.

An analog zooming can be performed between V_{SS} and V_{REF+} if $V_{DDA} > V_{REF+} \geq 2.4$ V.

4.2.2 Adding white noise or triangular sweep

This method combines hardware and software techniques to improve precision. From a software point of view, this method uses averaging and from a hardware point of view, it uses signal modification/spreading/dithering.

Averaging can be used in cases where the input signal is noisy (some signal change is necessary in order to be able to calculate an average) and the requirement is to obtain the mean value of a signal. A problem appears when the input signal is a very stable voltage without noise. In this case, when the input signal is measured, each data sample is the same. This is because the input signal level is somewhere between two ADC word levels (e.g. between 0x14A and 0x14B). Therefore it is not possible to determine the input voltage level more precisely (e.g. if the level is near to 0x14A or near to 0x14B level).

The solution is to add noise or some signal change (with uniform signal distribution e.g. triangular sweep) to the input signal which pushes its level across 1-bit ADC level (so that the signal level changes below 0x14A and above 0x14B level). This causes the ADC results to vary. Applying software averaging to the different ADC results, produces the mean value of the original input signal.

Example of implementation of this method is done by using a triangular generator with RC coupling to the input signal (noise signal generation is more complicated). Care must be taken not to modify the mean value of the original input signal (so, capacitive coupling must be used).

A very simple implementation is to design the white noise source as a quasi-triangular source which is generated directly by the STM8L/STM8AL microcontroller (*Figure 11*).

Figure 11. Simple quasi-triangular source using a microcontroller output
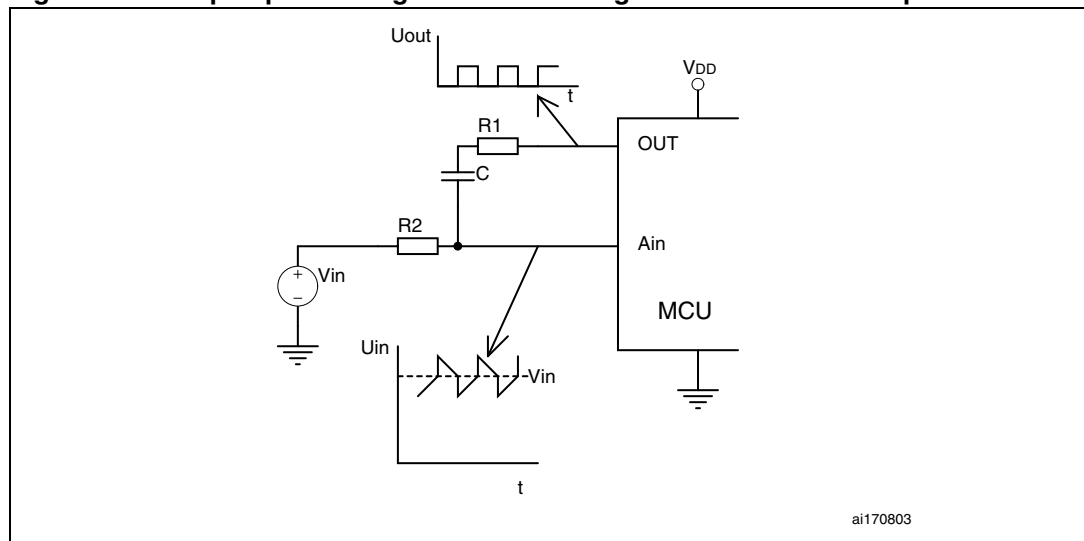
4.2.3 Hardware design rules

Good hardware design can minimize many external influences on the measured signal and measuring device. Therefore good design is also a method of improving precision, or rather a method of minimizing additional errors caused by the external hardware. The following list gives the main rules for minimizing external influences.

Grounding:
Care must be taken in the design of the PCB grounding. Ground paths on a PCB have nonzero resistance (and impedance) therefore current flowing through these paths generates a voltage drop. This voltage drop changes the voltages at points that are important for the ADC. For example it changes the reference voltage on the microcontroller pins slightly or influences the input signal on the microcontroller analog input pins. Good design minimizes this voltage drop. Methods of doing this are: minimizing the ground path impedance and minimizing the current through important ground paths (topology).
To minimize ground impedance: use massive ground paths. To minimize current through sensitive connections: use star topology, use independent PCB lines from input signal pins to microcontroller analog inputs, use independent PCB lines from the reference voltage source to microcontroller reference voltage inputs. Simply: supply current should not flow through signal ground - use signal ground and power ground in design.

Voltage filtering:
To minimize any external influences on the precise voltage levels used by the ADC, it is good to use filters on these voltage sources. Good filters must be applied on the reference voltage and on the supply voltage. Filter design depends on the noise amplitude present on the voltage sources, the external noise disturbance and the required precision.

Preamplifier usage:
If the measured signal is too small (in comparison to the ADC range) then it is good to use external preamplifier. This amplifier can fit the input signal range to the ADC range and can also insert offsets between input signal and ADC input. Care must be taken in the preamplifier design in order not to generate additional errors (for example additional offset, amplifier gain stability, linearity, frequency response...).

Frequency independence:
In cases where an AC signal is measured, frequency is an important characteristic of the input signal path. When measuring fast changes in the input signal then the frequency response of the signal path must be flat, even at high frequencies. The parasitic resistance and parasitic capacitance on PCB signal paths must be minimized. Otherwise, when measuring DC signal levels, adding an RC filter to the input signal data path improves robustness against external noise.

4.3 Software methods

- Averaging samples:
    - averaging decreases speed but can give improved accuracy
- Digital filtering (50/60 Hz suppression from DC value)
    - set proper sampling frequency (the trigger from timer is useful in this case)
    - perform software post-processing on sampled data (e.g.: comb filter - for 50 Hz noise and its harmonics suppression)
- Fast Fourier Transform (FFT) for AC measurements:
    - to show harmonic parts in measured signal
    - slower, due to the use of more computation power
- Calibration of ADC: offset, gain, bit weight calibration
    - decreases internal ADC errors
    - internal ADC structure must be known
- Minimizing internal noise generated by CPU
    - design firmware to use minimum disturbance from microcontroller during ADC conversion - digital silence.

4.3.1 Averaging samples

The principle of this method is to increase ADC precision but decrease ADC conversion speed. If the measured analog signal produces unstable ADC values, then the mean value of the given input signal can be obtained by averaging a set of values. Variation can be caused by signal noise or noise generated by the microcontroller itself (high speed digital signals capacitively coupled to the analog input signal).

Averaging is performed by choosing an appropriate number of samples to be averaged. This number depends on the required precision, minimum conversion speed and the level of other ADC errors (if another error has a greater influence on ADC precision, then increasing the number of averaging values has no effect on total measurement precision).

The advantage of this method is improving precision without any hardware changes. The disadvantage is lower conversion speed and therefore also lower frequency response (it is equivalent to decreasing effective sampling frequency).

4.3.2 Digital signal filtering

This is a modern method which uses digital signal processing techniques. In principle, averaging is also a simple digital filter with a specific frequency response. But if the noise frequency spectrum is known, a digital filter can be designed which minimizes noise influence and maximizes ADC frequency response. For example, if the noise in the measured signal is coming from the 50 Hz power lines, then an appropriate digital filter suppresses the 50 Hz frequency and delivers a precise 10 Hz input signal response.

The disadvantage of this method is that it needs sufficient microcontroller processing power and resources: CPU speed, data/program memory usage.

4.3.3 FFT for AC measurement

In some specific cases the application needs to know the amplitude of an AC signal with a given frequency. In this case the effective value of an AC signal can also be obtained by using a relatively slow sampling speed (in comparison to the measured signal frequency). For example, when measuring an AC mains signal (which is near-to-sinusoidal and has relatively low harmonics content), it is sufficient to choose a sampling frequency 32 times greater than the mains frequency (50 Hz). In this case you can obtain harmonics of up to the 15th order. The amplitude of 15th harmonics in the main signal is very small (the next order harmonics can be neglected). The calculated effective value of the mains signal is obtained with high precision because the effective values of harmonics are added to the total AC harmonic value as:

$$U_{ef} = \sqrt{U_1^2 + U_2^2 + \ldots + U_n^2}$$

So if the 15th harmonics amplitude is only 1% (0.01) from 1st harmonics (50 Hz) then its contribution to the total effective value will be only 0.01% (because of above equation - square addition: $0.01^2 = 0.0001$).

The principle of this method is therefore to sample the AC signal with a known frequency and then perform FFT post-processing on the data of each measured period. Because the number of sampling points per measured signal period is small (32 points for example) then the performance needed for FFT processing is not so high (only 32-point FFT for example). If there is no requirement for real-time processing, for example, with a stable input signal shape, and measuring only one period per second, as in the case of the mains signal, then FFT can be calculated even by an 8-bit microcontroller.

Advantages: this method is good for AC measurement of a stable input signal. The disadvantage is the requirement for precise signal sampling. The frequency of the measured signal must be known and the ADC sampling frequency must be set exactly as a 2^n multiplier of the measured frequency. The input signal frequency is measured by another method. The ADC sampling frequency is tuned by programming the prescaler and MCU master clock or interpolation can be used to insert sample points at the required frequency if sampling is performed with an inaccurate clock.

4.3.4 ADC calibration

This method requires knowledge of internal ADC structure and how the ADC converter is implemented inside the microcontroller. This knowledge is necessary in order to design a physical/mathematical model of the ADC implementation.

A proper physical model (which is usually a schematic diagram) is used as the base for describing it mathematically. From the mathematical model each element in the model can be obtained by set of equations (for example, resistor/capacitor values which represent bit weights). To solve these equations, it is necessary to perform a set of practical measurements and obtain a set of solvable equations (for example, measurement: input signal versus the proper ADC digital output words).

From the measured values and mathematical computation of the model, all known values of model elements (resistors, voltages, capacitors,...) can be put into the schematic diagram.

So instead of the ADC schematic with the designed values you obtain an ADC schematic with the real values for a given microcontroller.

Computed model parameters is stored in the microcontroller memory after calibration and used in post-processing to correct ADC values.

4.3.5 Minimizing internal CPU noise

When it operates, the CPU generates a lot of internal and external signal changes which are transferred into the ADC peripheral through capacitive coupling. This disturbance influences ADC precision (unpredictable noise due to various possible microcontroller operations).

To minimize influences of the CPU (and of other peripherals) on ADC, it is necessary to minimize digital signal changes during sampling and conversion time (digital silence). This is done using one of the following methods (applied during sampling and conversion time):

● minimizing I/O pin changes

● minimizing internal CPU changes (CPU stop, wait mode)

● stopping clock for unnecessary peripherals (timers, communications...)

4.4 High impedance source measurement

This section describes the ADC measurement behavior of STM8L/STM8AL ADC when a signal source with high internal impedance is used. It explains how to design an application to reach the requested precision and provides workarounds.

4.4.1 ADC input stage problem

The ADC in STM8L/STM8AL devices is a switched-capacitor ADC type. Switched capacitors work also as sampling capacitors (see *Section 1* for a detailed explanation).

When a signal comes from a voltage source with high internal impedance (for instance, 150 kOhm), an added error can be seen in measurement results. Error signals have also been observed on the ADC input pin, as shown in *Figure 14* (if the voltage source has zero voltage: $U_{in} = 0$ V, $R_{in} = 150$ KOhm, $C_{ext} = 0$ pF):

Figure 12. Typical voltage source connection to ADC input
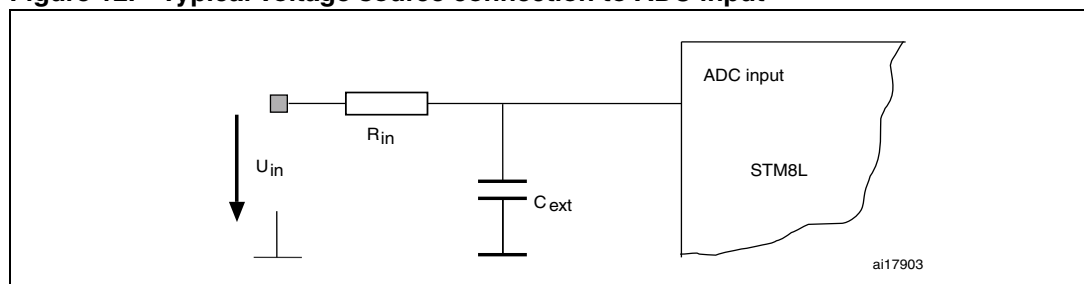
Figure 13.   Noise observed on ADC input pin during ADC conversions

ADC input signal during conversion: an ADC noise is injected to the input.
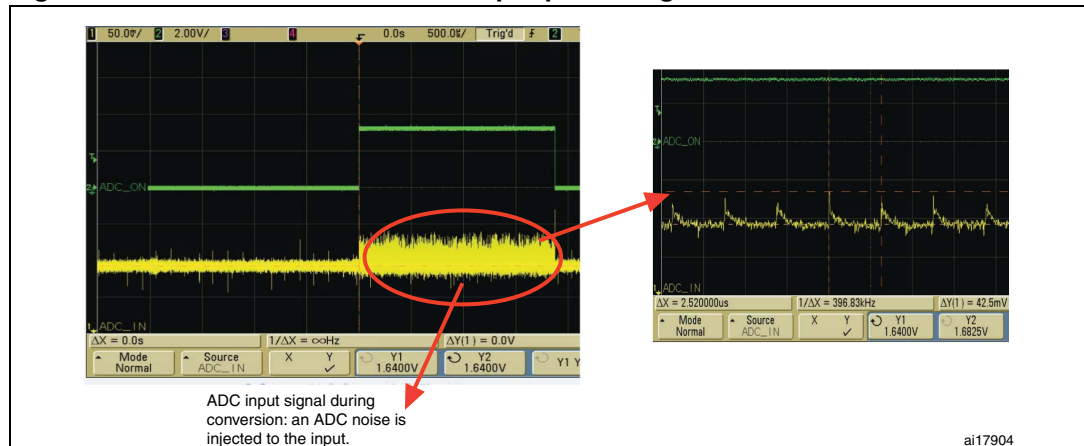
4.4.2   Explanation of the behavior

The explanation of this added pin noise and added measurement error (in case a signal source with high internal impedance is used) comes from the internal ADC structure: its input sampling circuit.

Figure 14 shows a simplified schematic of the input stage (sample and hold circuit).

Figure 14.   ADC simplified schematic of input stage - sample and hold circuit
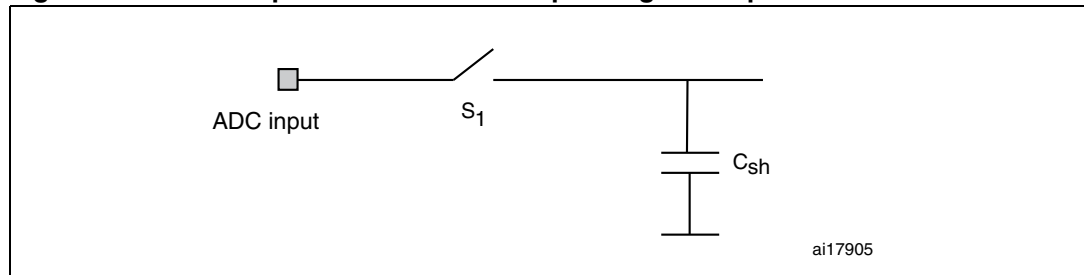
The spikes (noise) present on ADC input pin during conversions are related to the sampling switch (S1). If the switch is closed, some charge (coming from the sample and hold capacitor C_{sh} or caused by another effect) is transferred to the input pin. Then this charge starts discharging through the source impedance (R_{in}). The discharge process ends at the end of the sampling time (t_S) when the switch S1 is opened. The remaining undischarged voltage remains on the capacitor C_{sh} and ADC measures this voltage. If the sampling time (t_S) is too short, the remaining voltage does not drop under 0.5 lsb and ADC measurement shows an added error. *Figure 15* shows this process.

Figure 15. ADC input pin noise spikes from internal charge during sampling process
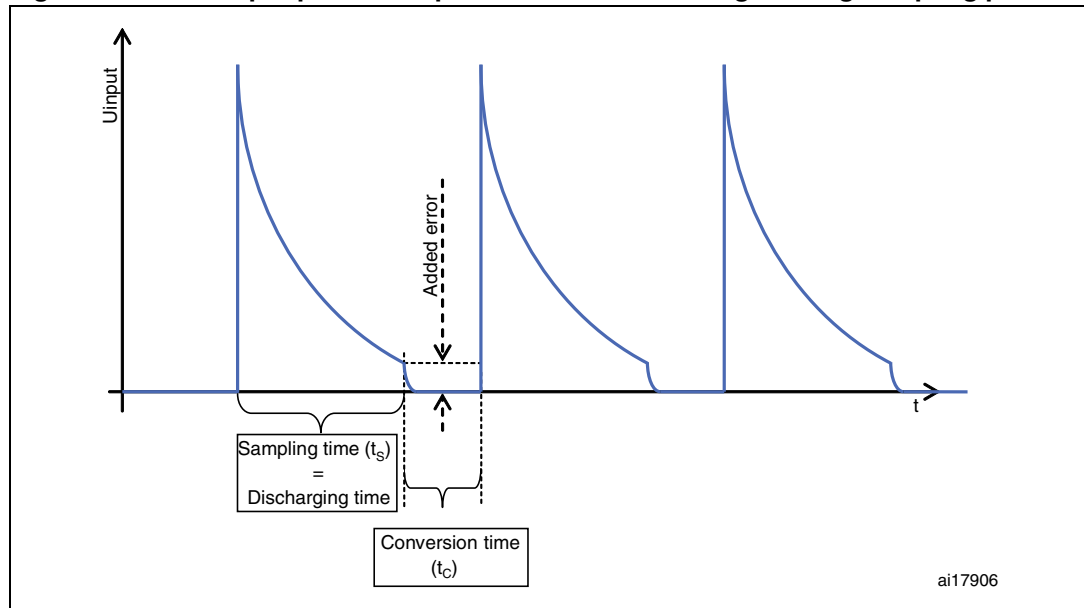
Note that a non-zero external capacitance C_{ext} (parasitic pin capacitance) also exists, so during conversion time the pin capacitance is discharged through source impedance R_{in}.

4.4.3 Minimizing added errors

Workaround for high impedance sources

To solve the added error problem, you can increase the sampling time (t_S) configuring ADC settings in MCU firmware, so that you discharge the C_{sh} charge through the source impedance R_{in}. The time constant ($R_{in} * C_{sh}$) is the reference for choosing the sampling time. To choose the sampling time cycles, use this formula (see also *Section 1*):

$$t_S \geq f_{ADC} \cdot (R_{in} \cdot C_{sh}) \cdot \ln\frac{4096}{0.5} \qquad [cycles]$$

The clock for ADC (fADC) is another important factor; slowing down the ADC clock increases the sampling time.

Figure 16. Effect of sampling time extension
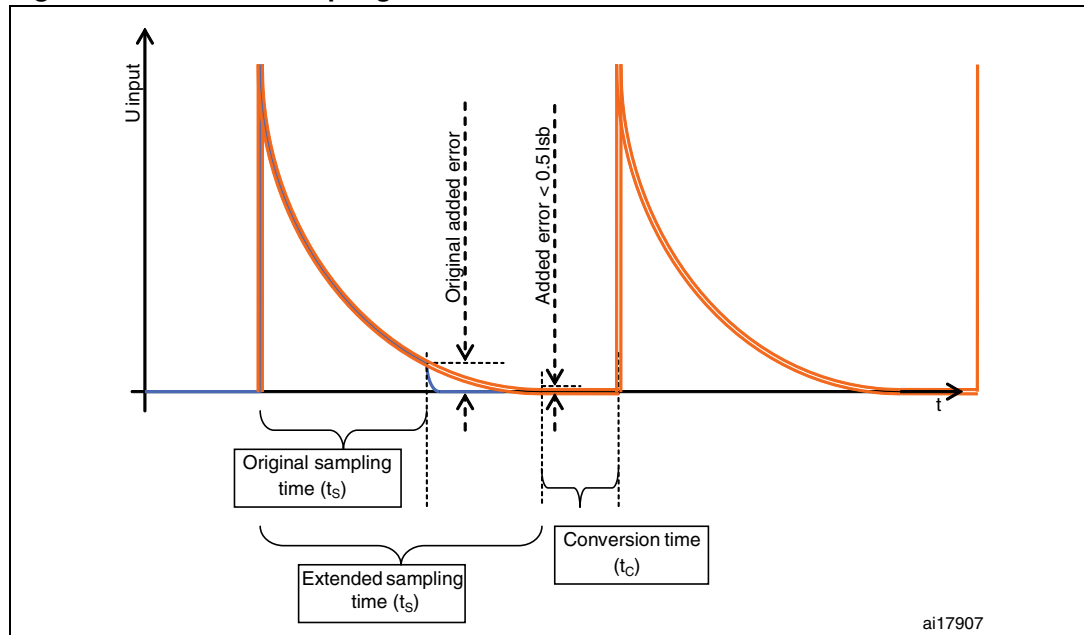
If the maximum register value of the sampling time setting is reached and the problem is still present, you need a more complex solution which is applicable also for measurements of source with extra high internal impedance (see the section *Workaround for extra high impedance sources* below).

Note that for this application you must take into account not only the internal sampling capacitance, but also any external parasitic capacitance (in parallel to C_{ext}), such as: pin capacitance, PCB path capacitance, etc.

Do not add any external capacitor (C_{ext}) to the input pin when applying this workaround. Its capacity will increase the timing constant ($R_{in} * C_{sh} \| C_{ext}$) and the problem will remain.

Workaround for extra high impedance sources

This workaround consists of 2 parts: a hardware change and a software change.

Hardware change:

The hardware change consists of adding a large external capacitor (C_{ext}) to the input pin. The capacity size connected to the input pin must reach the value that causes the discharging of the internal sampling capacity C_{sh} to the external capacitor C_{ext} without increasing the voltage on C_{ext} to more than 0.5 lsb. For example:

If the internal capacitor (C_{sh} = 16 pF) is charged to full scale (U_{max}, which corresponds to 4096 lsb), then the external capacitor C_{ext} must be charged at maximum 0.5 lsb voltage level (U_{lsb}) after discharging C_{sh} to it. The capacity of C_{ext} will then be:

$$C_{ext} \geq C_{sh} \cdot \frac{U_{max}}{U_{lsb}} = 16pF \cdot \frac{4096}{0.5} \approx 131nF$$

The closest larger standard value chosen here is: C_{ext} = 150 nF.

If the internal sampling capacitor C_{sh} is not charged to full voltage range (4096 level) before sampling, you can calculate the C_{ext} value replacing "4096" in the formula above. Calculating with 4096 level gives precise measurement results also in the case of ADC input channels switching (C_{sh} was charged from different ADC input in the previous measurement).

A side effect of this hardware workaround is the cyclical charging of C_{ext} which must be taken into account. Each ADC conversion transfers charge from C_{sh} to C_{ext}. One transfer charges the C_{ext} below 0.5 lsb, as described above, but more transfers can charge C_{ext} to larger values if it is not discharged between two conversions. *Figure 17* shows an example of this scenario where the ADC measurement is performed faster.

Figure 17. **Charging the external capacitor with too short time between conversions**
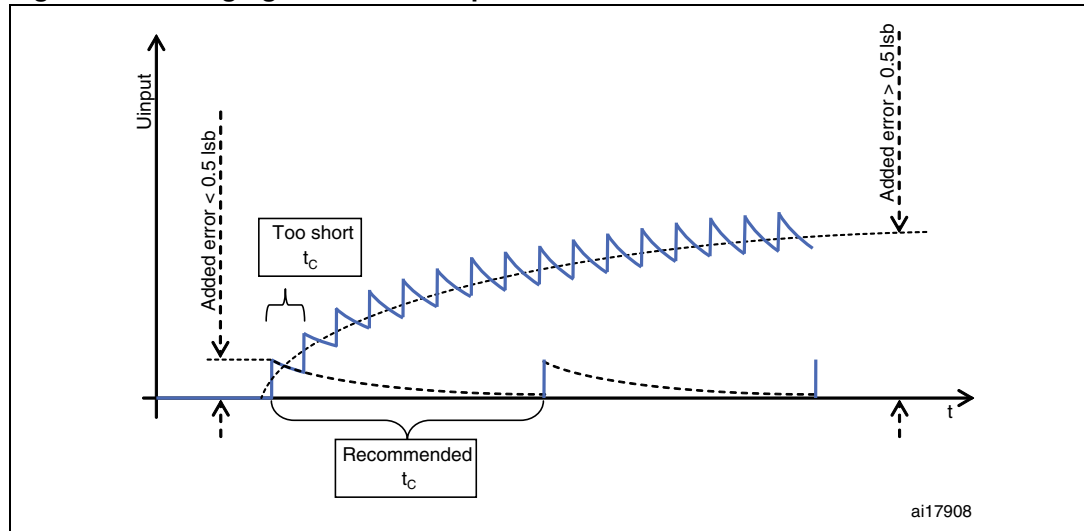
Software change:

The side effect mentioned above can be solved by software. The objective is to create a delay in order to let C_{ext} discharge through R_{in} (not measure so often) giving a "discharge time" between ADC conversions. The "discharge time" (t_C) is equal to the transferred charge from C_{sh} to C_{ext} (charging) and from C_{ext} to R_{in} (discharging). The assumption is that C_{ext} >> C_{sh}.

$$Q_{charging} = Q_{sh} = C_{sh} \cdot U_{max}$$

where U_{max} ..... 4096 lsb voltage level (worst case)

$$Q_{discharging} = \frac{U_{lsb}}{R_{in}} \cdot \int_{0}^{t_C} e^{-\frac{t}{R_{in}C_{ext}}} dt$$

where U_{lsb} ..... 0.5 lsb voltage level

$$Q_{charging} = Q_{discharging}$$

$$C_{sh} \cdot U_{max} = \frac{U_{lsb}}{R_{in}} \cdot \int_0^{t_C} e^{\frac{t}{R_{in}C_{ext}}} dt$$

After simplification, we obtain the final formula for the required waiting time between conversions:

$$t_C = -(R_{in} \cdot C_{ext}) \cdot \ln\left[1 - \frac{C_{sh}}{C_{ext}}\frac{U_{max}}{U_{lsb}}\right]$$

This final formula links the external capacitor C_{ext} and the required waiting time between two conversions if the precision U_{lsb} is needed.

From the same formula you can see that the argument in logarithm must be positive and therefore there is a condition for the minimal value of C_{ext}:

$$\left[1 - \frac{C_{sh}}{C_{ext}}\frac{U_{max}}{U_{lsb}}\right] > 0$$

$$1 > \frac{C_{sh}}{C_{ext}}\frac{U_{max}}{U_{lsb}}$$

$$C_{ext} > C_{sh} \cdot \frac{U_{max}}{U_{lsb}}$$

Choosing a larger C_{ext} decreases more the time between conversions (t_C).

An extra large C_{ext} ($C_{ext} >> C_{sh} \cdot \frac{U_{max}}{U_{lsb}}$) enables sampling more often.

However, increasing C_{ext} limits the frequency bandwidth of measurement signal (increasing the "external" timing constant $R_{in} \cdot C_{ext}$).

The formulas below show how to choose the optimal C_{ext} value: signal bandwidth in correlation with sample time. Signal bandwidth is characterized by an "external" timing constant, so:

$$(R_{in} \cdot C_{ext}) = t_C$$

$$(R_{in} \cdot C_{ext}) = -(R_{in} \cdot C_{ext}) \cdot \ln\left[1 - \frac{C_{sh}}{C_{ext}}\frac{U_{max}}{U_{lsb}}\right]$$

$$-1 = \ln\left[1 - \frac{C_{sh}}{C_{ext}}\frac{U_{max}}{U_{lsb}}\right]$$

$$e^{-1} = 1 - \frac{C_{sh}}{C_{ext}}\frac{U_{max}}{U_{lsb}}$$

After simplification we obtain the final formula for optimal C_{ext}:

$$C_{ext} = \frac{C_{sh}\frac{U_{max}}{U_{lsb}}}{1 - e^{-1}} \approx 1,58 \cdot C_{sh}\frac{U_{max}}{U_{lsb}}$$

and the corresponding waiting time between conversions:

$$t_C \approx -(R_{in} \cdot C_{ext}) \cdot \ln\left[1 - \frac{1}{1,58}\right] \approx (R_{in} \cdot C_{ext})$$

Practically the firmware must not program the ADC in continuous mode but only in single mode and must ensure that there will be a time gap between conversions with duration equal to t_C. This waiting time is the software change which must be applied together with the hardware change (adding an external capacitor C_{ext}).

Without implementation of t_C waiting time in software (for instance, running a conversion just after the first one) the external capacitor C_{ext} will cyclically charge from the C_{sh} capacitor. After a lot of cycles the voltage on C_{ext} will reach a quite high error value (as previously shown in *Figure 17*).

A practical example of implementation for STM8L/STM8AL ADC is shown below:

C_{sh} = 16 pF   ..... ADC property

R_{in} = 150 kΩ ..... signal source property

U_{max} = 4096 lsb   ..... ADC property

U_{lsb} = 0.5 lsb   ..... required precision

$$C_{ext} = 1,58 \cdot C_{sh}\frac{U_{max}}{U_{lsb}} = 1,58 \cdot 16pF \cdot \frac{4096}{0.5} \approx 207nF \Rightarrow 220nF$$

$$t_C = -(R_{in} \cdot C_{sh}) \cdot \ln\left[1 - \frac{C_{sh}}{C_{ext}}\frac{U_{max}}{U_{lsb}}\right] = -(150k\Omega \cdot 220nF) \cdot \ln\left[1 - \frac{16pF}{220nF}\frac{4096}{0.5}\right] \approx 29891\mu s \Rightarrow 30ms$$

4.4.4   Source of described problem - ADC design

The following sections list some possible causes for the charging of the internal sampling capacitor C_{sh}. This is not an exhaustive list; only the main possible sources of the ADC design are listed.

Parasitic switch capacitance effect

The sampling switch inside ADC sampling circuit (see *Figure 14*) is not ideal. In reality the sample and hold switch (S1) is designed as 2 transistors (PMOS and NMOS):

Figure 18.   Implementation of sampling switch
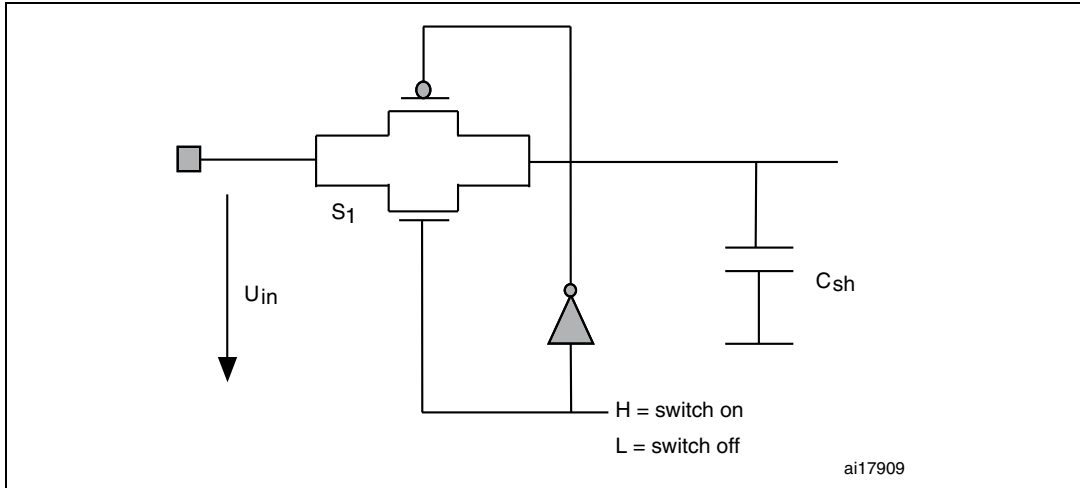
The switch is controlled by the gate voltages of transistors (inverted signal on PMOS transistor). This design is a standard bidirectional switch (for rail to rail range of input U_{in} voltages).

Both transistors have parasitic capacitances between gate and source. If those capacitances are charged (close to the switch), then their charge can be transferred to the sampling capacitor.

Figure 19. Parasitic capacitances of sampling switch
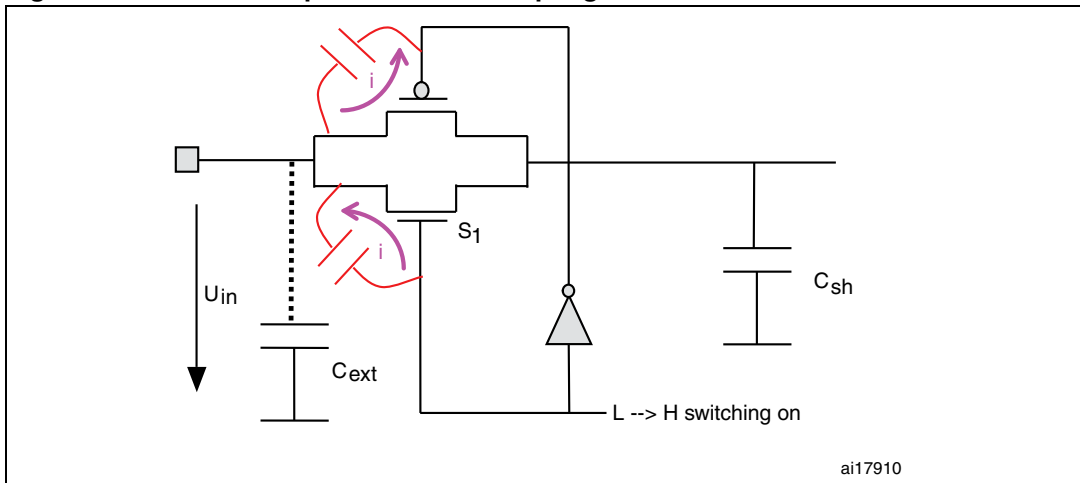
This charging and discharging currents (PMOS and NMOS asymmetric capacitances) can cause charge transfer to sampling capacitor Csh.

Internal charging of sampling capacitor

It is possible that after the conversion process (successive approximation process in SAR type of ADC) the sample and hold capacitor C_{sh} is charged to some voltage. The reason can be:

● some leakage current to C_{sh} (parasitic current inside ADC structure, see *Figure 20*)

● residual charge transfer from the switches when ADC structure is switched back to default state before next conversion

● other reasons (related to internal ADC parasitic structures)

Figure 20. Parasitic current example inside ADC structure
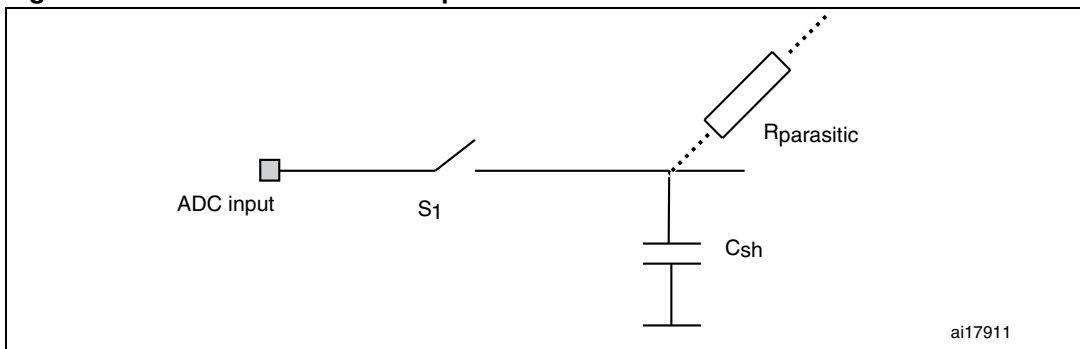
4.5 Design rules for minimizing errors

ADC precision can only be significantly improved if external influences on the measurement are minimized. Therefore any precision improvement method also includes rules for correct design.

Main design rules for minimizing design errors:

● Grounding of analog/digital power:
   – star topology
● V_{DDA}, V_{SSA} filtering:
   – RC, LC filtering
   – avoid noise from noisy digital power
● V_{REF} selection - offset, value, precision:
   – reference voltage source precision and stability matched to application precision requirements and ADC capability
   – reference voltage source value and precision in line with the expected measured range
● Source impedance vs. input impedance knowledge:
   – use input buffers for measured signal
   – impedance in relation with required conversion speed (proper sampling time)
● External preamplifier usage:
   – for low (and also high) level signals
   – amplifier speed and precision properties
   – amplifier dependency on frequency
● Select appropriate ADC mode, speed, trigger
● Software methods: averaging, FFT, … (see *Chapter 4.3: Software methods*)

5 Conclusion

The ADC in the STM8L/STM8AL microcontroller families is a standard ADC converter with various operation modes and settings which are described in this document.

This ADC can have common errors. To minimize these errors and improve precision, the main methods and application design rules have been described. You can apply these methods to your application.

The choice of method depends on the application requirements and is always a compromise between speed/precision, enough computation power and design topology. The published methods lead to a precision improvement and are optimized for the design of an ADC converter using the SAR (successive approximation register) principle.

Appendix A Source code examples

A.1 Project code example

A firmware package is linked with this document. It contains source codes in C-language which gives examples of some of the precision improvement methods described in this application note. This complete project is designed for the *ST Visual Develop* (*STVD*) development environment and *Cosmic/Raisonance* compiler and using the *STM8L1526 evaluation board*. With these examples you can see how the methods are implemented in practice and you can test them by comparing ADC precision with improvement methods and without them.

Package name: *STM8L_ADC_improvement.zip*
Project name: *STM8L_ADCPrecisionImprovement.stw*

A.2 Source code description

The example consists of a complete STVD project. This project contains source codes in files:

main.c	main program file (MCU initialization, main program loop)
ADCAveraging.c	testing - each file for given ADC precision improvement method (uses STM8L library)
ADCCalibration.c	
ADCWhiteNoise.c	
Filter50Hz.c	
FFT.c	
mono_lcd.c	basic driver for writing to LCD display (on the STM8L evaluation board)
mallocR.c	malloc implementation for Raisonance compiler
ADCdriverFFT.c	auxiliary functions for FFT implementation (hardware access and general functions)
ADCfncFFT.c	
stm8_interrupt_vector.c	contains interrupt table address definition for Cosmic compiler
"*.h" files	corresponding header files for used C - files and definitions
/Library	subdirectory which contains complete STM8L15x library (*stm8l15x_adc.c, stm8l15x_beep.c, stm8l15x_clk.c, stm8l15x_comp.c, stm8l15x_dac.c, stm8l15x_dma.c, stm8l15x_exti.c, stm8l15x_flash.c, stm8l15x_gpio.c, stm8l15x_i2c.c, stm8l15x_irtim.c, stm8l15x_itc.c, stm8l15x_iwdg.c, stm8l15x_lcd.c, stm8l15x_pwr.c, stm8l15x_rst.c, stm8l15x_rtc.c, stm8l15x_spi.c, stm8l15x_syscfg.c, stm8l15x_tim1.c, stm8l15x_tim2.c, stm8l15x_tim3.c, stm8l15x_tim4.c, stm8l15x_usart.c, stm8l15x_wfe.c, stm8l15x_wwdg.c*)

A.2.1 Program flow

The *main.c* file contains the main program loop - in function *main()*. At the beginning, the basic MCU initialization is performed: clock, interrupts, display... Then the main testing loop begins - it consists of series of steps.

Each test step contains a write to the LCD display to display information about the running test and then contains the call to test routine itself. The test routines are provided in the corresponding file. To continue from one test to the next, the user must press a "Key" button on evaluation board. After all tests are done, then the test loop runs again.

Tests are:

● *TestADCAveraging();*
  – ADC in single mode - channel 24
  – averaging method applied
● *TestADCWhiteNoise()*;
  – ADC in single mode - channel 24
  – averaging method applied
● *TestADCDigitalFilter50Hz()*;
  – ADC in single mode - channel 24
  – external timer trigger
  – simple digital comb 50 Hz filter applied
● *TestADCCalibration()*;
  – ADC in single mode
  – simple linear model applied (offset and gain calibrated)
● *TestADCFFT()*;
  – ADC in single mode
  – external timer trigger
  – 16-point FFT applied

A.2.2 User interface description

A user interface is available to test the described example. It consists of an LCD display and buttons on the evaluation board. These are the steps to follow:

1. Load the example firmware onto the evaluation board
2. Run the program (resetting the evaluation board); the invitation screen appears.
3. Connect the signal to the BNC connector
4. Press the joystick

The main program loop begins, with the following methods shown:

Averaging method:

● the first line on LCD shows the difference (maximum-minimum) from raw samples (buffer is 200 samples)
● the second line on LCD shows the difference (maximum-minimum) from averaged samples (floating average of 10 samples)
● to stop the method press the "Key" button

White noise modulation method:

● external signal should be modulated by white noise or small triangle signal (for instance, from function waveform generator)

● the second line on LCD shows sub LSB resolution from measured signal - averaged 200 samples

● to stop the method press the "Key" button

Digital filter method:

● a 50 Hz rejection filter (comb filter) is implemented

● the first line on LCD shows the difference (maximum-minimum) from raw samples (buffer is 500 samples)

● the second line on LCD shows the difference (maximum-minimum) from 50 Hz filtered samples

● 50 Hz suppression is seen when a DC signal with 50 Hz added noise is applied

● to stop the method press the "Key" button

ACD calibration method (linear model):

● a linear ADC curve approximation is implemented: $Y = k.X + Q$

● samples for calibration are sampled first: more samples means better approximation

● the first line on LCD is editable - a real precise value must be entered (measured by calibration device)

● the second line on LCD shows sample number, navigation cursor and value measured by ADC

● use the joystick for entering real value:
   – left/right = cursor left/right movement
   – up/down = increase/decrease of real value
   – center = enter values pair as valid (sample number increases)

● change the input signal to different level

● enter real value by joystick

● continue with another signal level (> 5 samples, for example)

● to stop entering sampling pairs, press the "Key" button

● an information screen appears:
   – the first line on LCD shows standard deviation before calibration
   – the second line on LCD shows standard deviation after linear calibration model calculation

● next screen appears after 3 seconds:
   – measured value with applied linear model (calibrated ADC)
   – you can compare the real calibration value with the value on LCD; it should be very close
   – to stop the method press the "Key" button and before releasing it press the joystick

FFT analysis:

● 16 point FFT calculation of 50 Hz signal is implemented

● FFT is applied on 6 ADC channels: AIN0 - AIN5

● FFT method shows 3-phase AC signal power measurement:

  – even channels (0, 2, 4) represent voltages

  – odd channels (1, 3, 5) represent currents

  – power is calculated as sum of the power of each harmonic frequencies (16 point FFT = 7 harmonic)

  $Pa = U_1*I_1*cosF_1 + ... + U_n*I_n*cosF_n$ (n=7)   ... for phase "a"

  $Pb = U_1*I_1*cosF_1 + ... + U_n*I_n*cosF_n$ (n=7)   ... for phase "b"

  $Pc = U_1*I_1*cosF_1 + ... + U_n*I_n*cosF_n$ (n=7)   ... for phase "c"

● connect 6 signals to AIN0 - AIN5 pins; signals should have frequency CCA 50 Hz

● the calculation takes CCA up to 3 seconds

● the second line on LCD shows calculation results: phase number, effective values of U and I, harmonic power, phase shift, distortion of U and I (all for each of the 3 phases)

● to stop the method press the "Key" button (wait for the end of the result list, "done" will appear on the screen)

Now the program starts again from the averaging method.

A.2.3   Hardware and software requirements

To run the project example you must have installed *STVD* (version 4.1.3 or newer) and *Cosmic/Raisonance* compiler for STM8L/STM8A microcontrollers. Both tools are free for download from STMicroelectronics and Cosmic/Raisonance websites.

The hardware requirement is to have the STM8L1526 evaluation board with the STM8L152C6 chip installed.

Revision history

Table 2. **Document revision history**

Date	Revision	Changes
02-Apr-2010	1	Initial release.
07-Jan-2011	2	Added the following new sections: – *Chapter 4.4: High impedance source measurement* – *Chapter A.2.2: User interface description*
15-Nov-2012	3	Document updated to include STM8AL products. Added *Table 1: Applicable products*.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST's terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST'S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY TWO AUTHORIZED ST REPRESENTATIVES, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER'S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2012 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com