AN5537

Application note

How to use ADC Oversampling techniques to improve signal-to-noise ratio on STM32 MCUs

Introduction

All STMicroelectronics microcontrollers embed an ADC (analog-to-digital converter) with a given resolution (number of bits) and sampling rate.

For most applications, this resolution is sufficient, but in some cases where a higher accuracy is required, oversampling, and decimating the input signal can be implemented to avoid the use of an external ADC solution and the associated increase in application power consumption.

This application note presents the oversampling principle, then describes hardware and software oversampling implementation using a specific unit that is available on certain STM32 MCUs. It then compares the two possibilities in terms of power consumption.

For the software implementation, two ADC resolution improvement methods are described. These are based on oversampling the input signal with the maximum sampling rate of the ADC used, and decimating the input signal to enhance its resolution. The embedded software (STSW-STM32014 or X-CUBE-ADC_OVSP) delivered with this application note gives implementation examples for these two methods, and applies them to both medium- and high-density STM32F1 series products, as well as all STM32F3 series and STM32Lx series products.

For the hardware implementation, an overview of the on-chip hardware analog-to-digital converter (ADC) oversampling engine is provided. It is integrated in the STM32 products listed in Table 1.

The main user benefit of hardware oversampling is increased SNR (signal-to-noise ratio) with less CPU interaction, resulting in overall lower power consumption compared with the software-based implementation.

Formulas are provided to determine the oversampling ratio or the hardware oversampling unit configuration to use according to the desired resolution improvement. These theoretical formulas are compared to practical use cases.

Table 1. Applicable products

Type	Series
Microcontrollers	STM32U5 series, STM32U0 series, STM32H7 series, STM32H5 series, STM32F7 series, STM32F4 series, STM32F2 series, STM32F0 series, STM32L1 series, STM32F3 series, STM32F1 series, STM32L4+ series, STM32L4 series, STM32L0 series, STM32L5 series, STM32G4 series, STM32G0 series, STM32WB series

1 General information

This document applies to STM32 Arm®-based microcontrollers.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2 Oversampling as a way to improve the quality of signal acquisition

2.1 Quantization of noise and signal-to-noise ratio

Analog-to-digital converters (ADCs) transform analog signals into an array of digital codes. It is carried out by performing amplitude quantization of the analog input signal. The quantization resolution depends on the binary output word length, normally in the range of 6 to 18 bits. The error between the input signal and the quantized signal is called the quantization error.

The maximum error for an ideal converter when digitizing a signal is ±½ LSB (least significant bit), as shown in the transfer function (left side of Figure 1).

The LSB is also often called a quantum (q). Assuming that the user has an N-bit analog-to-digital converter (ADC) and a voltage reference, VAREF, the quantum, q, is the minimum distance between two adjacent ADC codes. Moreover, it is defined as follows:

$$q = \frac{V_{AREF}}{2^N} \quad (1)$$

where:

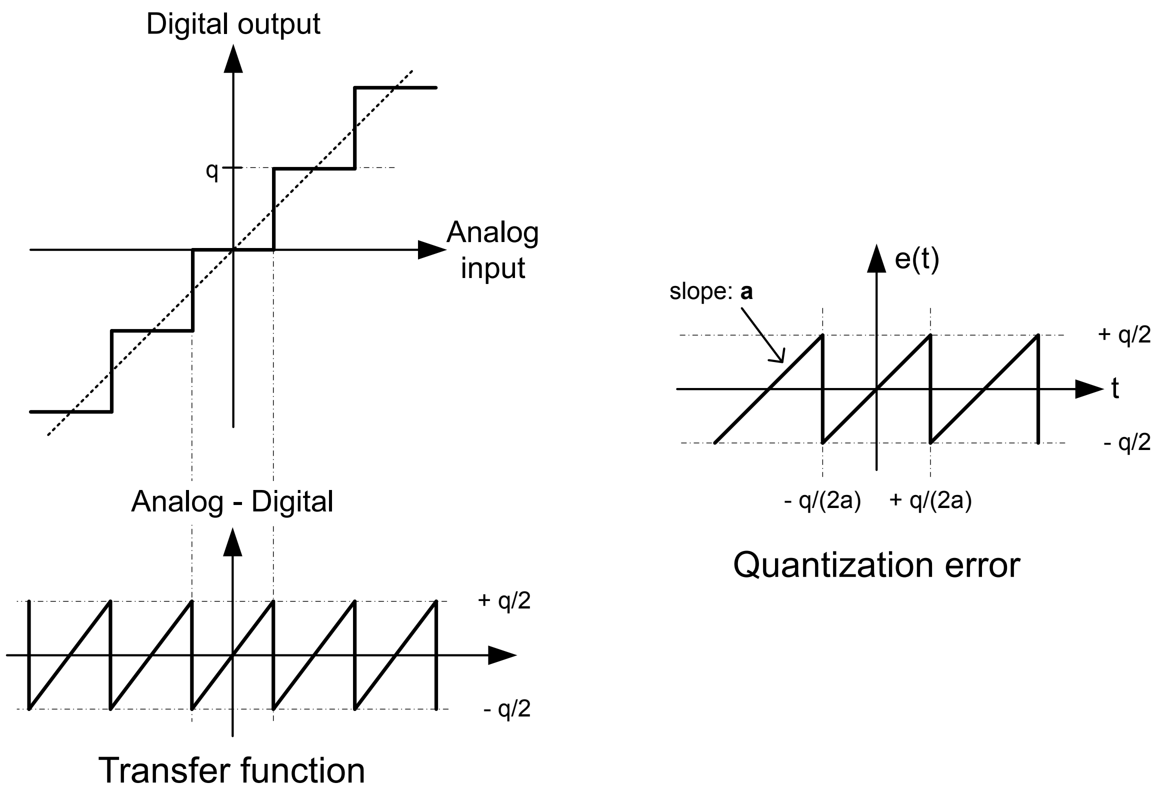
$$q/2a < t < +q/2a$$

with -q / 2a < t < +q / 2a

The quantization error e(t) as a function of time is shown in the right side of Figure 1.

This is approximated by a sawtooth signal, as it is considered that an uncorrelated sawtooth waveform is a good representation of the quantization error for any AC signal, and that it behaves like wideband noise.

Figure 1. **Ideal N-bit ADC quantization**



The quantization error e(t) is defined as:

$$e(t) = a \times t \quad with \quad -q/2a < t < +q/2a \quad (2)$$

Hence, the RMS value of e(t) is:

$$\sqrt{\bar{e}(t)^2} = \sqrt{\frac{a}{q} \cdot \int_{q/(2a)}^{(-q)/(2a)} \left(e(t)^2\right) dt} = \frac{q}{\sqrt{12}} \tag{3}$$

The SNR (signal-to-noise ratio) is the ratio of the ADC noise to the input signal power. For an ideal ADC, it is assumed that the SNR is equal to the ratio of the quantization noise to the input signal. No other noise source is considered.

For a full-scale input sine wave this is expressed as follows:

$$s(t) \;=\; q \times 2^{(N-1)} \times \sin(2\pi f t) \tag{4}$$

Using equations (3) and (4), the SNR of an ideal N-bit converter (ADC resolution) is calculated as follows:

$$SNR = 6.02 \times N + 1.76\, dB \tag{5}$$

It is important to note that the RMS quantization noise is measured over the full Nyquist bandwidth (from DC up to Fs/2).

It can be seen that when the SNR increases, the ADC effective number of bits (N in the equation 5) increases.

Note also that for a real ADC, different error sources must be considered: offset, gain - INL (integral nonlinear) and DNL (differential nonlinear). A brief description of these errors can be found in the STM32 MCU datasheets. These errors degrade the ideal ADC resolution and determine the real effective number of bits of the ADC (ENOB). Improving the SNR enhances the effective number of bits of the ADC. The following section demonstrates that sampling the input signal rates higher than the Nyquist frequency improves the SNR. The Nyquist frequency is discussed in the next paragraph.

2.2 Nyquist theorem and antialiasing low-pass filter relaxation

The Nyquist theorem states that to reconstruct the analog input signal, the signal must be sampled at a rate Fs (sampling frequency) that is greater than twice the maximum frequency component of the input signal.

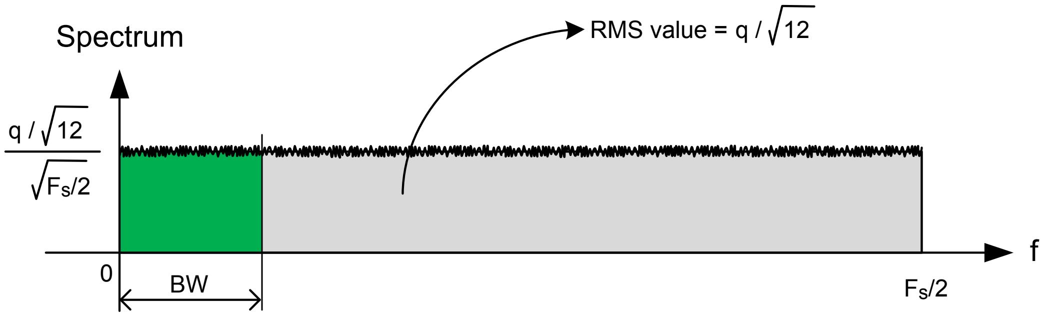
Noncompliance with the Nyquist theorem causes aliasing effects and the analog signal cannot be fully reconstructed from the input samples.

Therefore, for most applications, a low-pass filter is required at the ADC input to filter frequencies lower than half of the sampling frequency. It is difficult to handle the filter constraints with low sampling frequencies. The oversampling consists of sampling the analog input signal at higher rates than the Nyquist frequency limit, filtering the samples, and reducing the sample rate by decimation. Using this method relaxes the antialiasing low-pass filter constraints.

2.3 Processing gain achievable with oversampling

In most cases, we can consider that the quantization noise is uncorrelated with respect to the input signal. In this condition, the quantization noise is approximately Gaussian and spreads more or less uniformly over the Nyquist bandwidth (see Figure 2).

Figure 2. **Quantization noise spectrum**

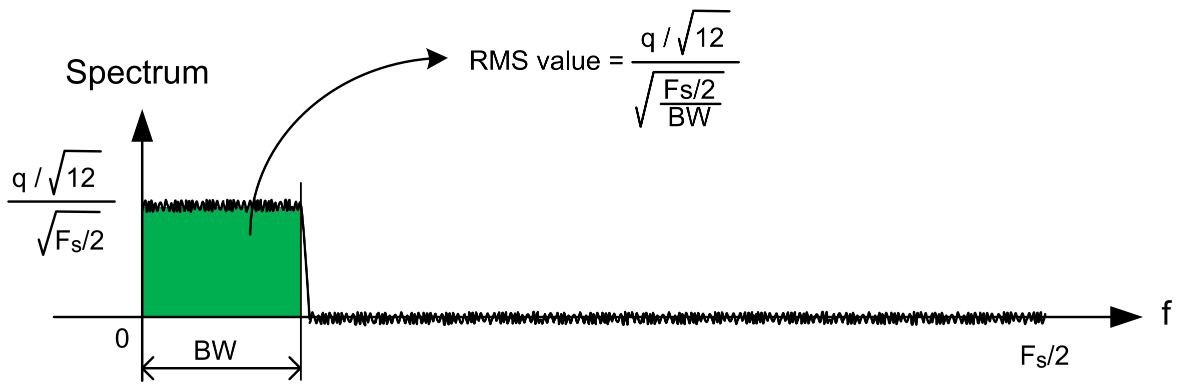


However, under certain conditions where the sampling clock and the signal are harmonically correlated, the quantization noise becomes correlated. In addition, its energy is concentrated in the harmonics of the signal. In conditions where the quantization noise does not appear as random noise, dithering must be applied (see Section 2.4 Dithering).

In many applications, the useful signal occupies a bandwidth (BW) smaller than Fs/2.

If digital filters are used to remove the noise outside the BW (this filter can be more precise than the antialiasing ones mentioned before), the total RMS noise is reduced (Figure 3); the RMS value of the quantization noise is divided by a ratio that depends on the useful bandwidth (BW) with respect to the sampling rate (Fs).

Figure 3. Quantization noise gain


RMS value = $\frac{q/\sqrt{12}}{\sqrt{\frac{Fs/2}{BW}}}$
Spectrum
$\frac{q/\sqrt{12}}{\sqrt{F_S/2}}$
0
BW
f
$F_S/2$

We can then reformulate the previous SNR expression taking into account this processing gain, by filtering the out-off band noise:

$$SNR = 6.02 \times N \times \; + 1.76\, dB + 10xLog_{10}OSR \tag{6}$$

This expression is valid over a bandwidth, BW, with an oversampling ratio given by:

$$OSR = FS/(2 \times BW) \tag{7}$$

2.4 Dithering

The technique presented above works well for a white quantization noise.

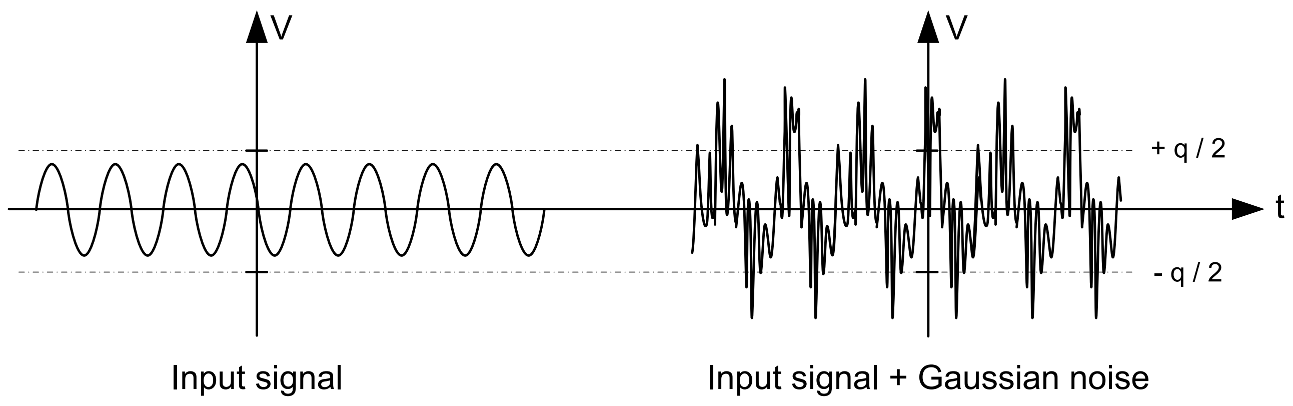
However, if the sampling clock and the signal are harmonically correlated (in this case the quantization noise becomes correlated as well), or when the input signal amplitude is smaller than q/2, the processing gain does not work properly.

This is because for the first case the quantization noise is no longer random, and for the second case there are (in theory) no code transitions when the signal is smaller than the quantization step.

One way to solve these issues is to use the dithering technique, where a small Gaussian noise is added to the input signal (see the left side of Figure 4), to obtain a signal (see the right part of the figure) that can ensure LSB toggling.

Dithering also ensures that the quantization noise is always random, independently from the input signal.

Figure 4. Dithering technique



The impact of SNR can be much reduced if the noise is shaped; for example if the dithering noise is filtered in the wanted bandwidth, and is only present outside of that bandwidth.

The embedded DAC can be used for generating the dithering signal. Also, in Section 3.2 **Oversampling using triangular dither**, we generate the dithering signal by means of a timer configured in PWM mode, and some additional electronic components.

If the application does not require the capture of signals smaller than the quantization step, and if the quantization error can be considered as wideband noise, the dithering technique can be omitted.

3 Software oversampling

This section presents two software-oversampling implementation methods. Each has advantages and disadvantages, which are compared.

The embedded software delivered with this application note is available in the STSW-STM32014 (or X-CUBE-ADC_OVSP) package.

3.1 Oversampling using white noise

3.1.1 Oversampled signal SNR with white input noise

Equation 6 in Section 2.3) gives the SNR obtained when oversampling the input signal with a sample rate OSR times faster than the Nyquist frequency, and low-pass filtering the signal band:

$$SNR = 6.02 \times N \times \; + 1.76\, dB + 10xLog_{10}OSR$$

This shows that each doubling of the sampling frequency reduces the in-band noise by 3 dB, and increases the measurement resolution by 0.5 bit. Therefore, a 6 dB SNR gain is required to add 1 resolution bit to the ADC. In general, if *p* additional bits are required by the application, the ADC sampling frequency should be at least:

$$F_{OVS} = 4^{p} \times F_{s} \tag{8}$$

Where F_s is the ADC sampling frequency used.

3.1.2 Decimation

Averaging means adding m samples and dividing the result by m. Averaging several data from an ADC measurement is equivalent to a low-pass filter, which attenuates the signal fluctuation and noise. Averaging is therefore often used to smooth and remove spikes from the input signal.

Note: *Normal averaging does not increase the resolution of the conversion because the sum of m N-bit samples divided by m is an N-bit representation of the sample.*

Decimation is an averaging method. When combined with oversampling, decimation improves the ADC resolution.

In fact, adding 4^p (4 power of p) ADC N-bit samples, gives a representation of the signal on N+2p bits. To have p additional effective bits, the sum is shifted to the right by p bits.

This FIR filter with equal filter coefficients enables the user to filter the oversampling frequency by giving an output sample computed from the OSR input samples.

The oversampling method limits the maximum input frequency bandwidth. In the case of the STM32F1 series, STM32F3 series and STM32Lx series (with maximum sampling rate around 1 Msps), signals having components up to 500 kHz can be processed by the ADC. If for example, two additional resolution bits are required, the maximum input frequency is 500 kHz/16 = 31.25 kHz when the oversampling uses white noise.

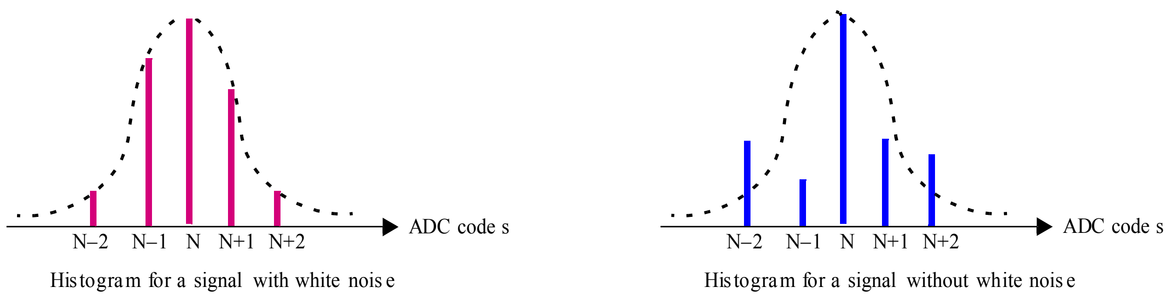
3.1.3 When is this method efficient?

For the oversampling and decimating method to work properly, the following requirements must be satisfied:

- There should be some noise in the input signal. This noise must approximate the white noise with a uniform power spectral density over the frequency band of interest.
- The noise amplitude must be sufficient to toggle the input signal randomly from sample to sample by an amount of at least 1 LSB. Otherwise, the input samples would have the same representation, and the sum and average operations would not give any extra resolution. For most applications, the internal ADC thermal noise and the input signal noise are sufficient to use this method. If the thermal noise does not have a high-enough amplitude to toggle the input signal randomly, then a dithering operation must be applied (see part 2.4). Regarding this point, two questions can be raised. The first is *How to evaluate the ADC noise and test its Gaussian criteria?* and *How to generate white noise if needed?*.

A practical way of detecting the Gaussian criteria of the input signal noise is to see the distribution of a clean DC signal over the ADC codes. The histogram method can be used to verify if the input noise follows a Gaussian distribution. The example in Figure 5 shows two possible situations.

Figure 5. **Histogram analysis**



In the case where an external noise dither must be added to the input signal, the thermal noise generated by a diode or a resistor can be injected into the input signal.

The input noise must not correlate with the useful input signal, and the input signal should have an equal probability of being between two adjacent ADC codes. This means that this method does not work for systems using a feedback process.

3.1.4 Implementation method on STM32F1, STM32F3, and STM32Lx series devices

This method describes the different steps undertaken to implement and test the oversampling method on the STM32F1 series, STM32F3 series and STM32Lx series devices.

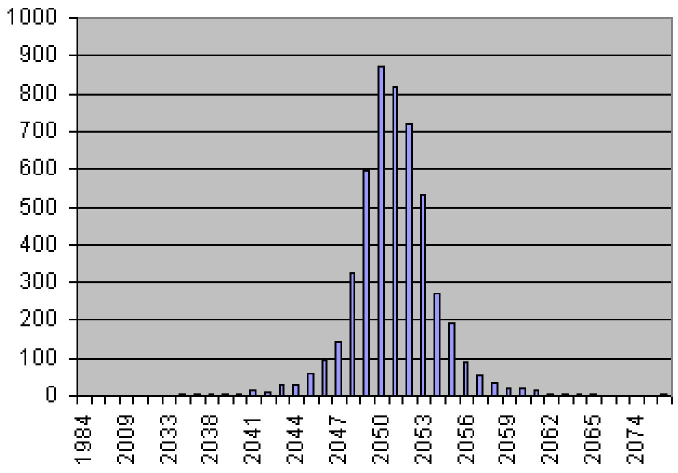
According to the previous section, to make this solution work properly, there must be some white noise to make the input signal toggle randomly by 1/2 LSB. For this, the application environment noise must be considered.

The first step consists in computing the ADC thermal noise to conclude if external white noise must be injected into the input signal. In a typical application board, the computed noise does not include only the ADC internal noise but also the possible noise generated by the different board components and the layout. Therefore, this evaluation depends on the application board but the methodology remains the same.

The histogram method is used for different DC input voltages. This input voltage is sampled a large number of times (example 5000). The related distribution can be easily interpreted using a spreadsheet.

For example, for a 1.65 V dc input voltage applied on the STM3210B-EVAL evaluation board, the histogram shown in Figure 6 is detected.

Figure 6. **Histogram analysis for DC = 1.65 V**



The ADC thermal noise can be computed from this histogram (although this can be shown, it is not the objective of this application note and the details are not offered here).

To carry on this ADC noise test, the user must do the following:

- Uncomment the line `#define Themal_Noise_Measure` in the *oversampling.h* file.
- Configure the `Total_Samples_Number` which is the number of ADC conversion operations. It must be smaller than 65535. The DMA channel is configured to store the number of ADC samples in a RAM buffer. At the end of the transfer, an interrupt is generated and the number of occurrences of each ADC code is computed.
- To compute the occurrence of the ADC codes, a variable giving the relevant ADC codes is defined.

When the code is run, `Relevant_ADC_Samples` ADC samples and their corresponding number of occurrences are displayed on the HyperTerminal. The HyperTerminal configuration is 8-bit data, no parity, 115 200 baud rate. If the effective number of ADC samples found is smaller than the defined `Relevant_ADC_Samples` variable, then 0 is displayed for both ADC code and ADC code occurrences. The user can capture them and build a histogram.

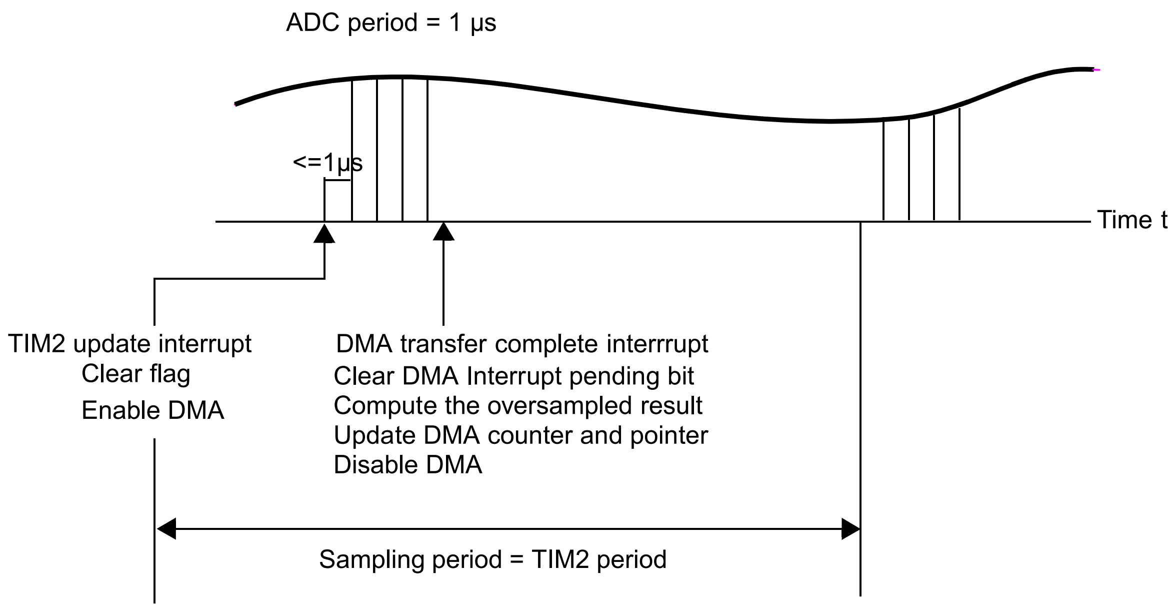
3.1.4.1 *Embedded-software flowchart for oversampling using white noise*

The STM32F1 series, STM32F3 series and STM32Lx series on-chip ADC conversion frequency is fixed to 1 MHz. The ADC DMA channel is configured to transfer the number of oversampled inputs from the ADC data register to a buffer in RAM. This transfer is configured to occur one time. At the end of the DMA transfer, an interrupt is triggered and the oversampled result is computed.

The general-purpose timer TIM2 is used to generate the input signal sampling frequency. For this, the TIM2 reference clock is configured at 1 µs. Its period determines the input signal sampling period. It is defined in the oversampling.h file as `#define Input_Signal_Sampling_Period`. When the TIM2 update interrupt is triggered, the DMA is reenabled and the converted ADC values can be treated.

Figure 7 summarizes the implemented functionality.

Figure 7. **Oversampling using a white noise flowchart**


ADC period = 1 µs
<=1µs
Time t
TIM2 update interrupt
Clear flag
Enable DMA
DMA transfer complete interrrupt
Clear DMA Interrupt pending bit
Compute the oversampled result
Update DMA counter and pointer
Disable DMA
Sampling period = TIM2 period

The oversampled datum is computed in the DMA transfer complete interrupt. For synchronization reasons, it is recommended to read it in the second TIM2 interrupt. Note that with this implementation, the TIM2 period must be greater than the time required by the ADC to convert OSR samples, and greater than the ADC interrupt execution time.

If the sampling frequency required by the application is exactly OSR µs, then the user is not required to use the timer TIM2 to generate the input sampling frequency. However, the DMA must be configured to be functional in continuous mode and the DMA transfer complete interrupt must be updated accordingly. The oversampled datum is usually computed in the DMA transfer complete interrupt.

3.1.4.2 *Oversampling using white noise - result evaluation*

To evaluate the oversampling method, the user must uncomment the `#define Oversampling_Test` line and configure the number of samples with an enhanced resolution.

When this line is uncommented, a buffer is created in the RAM to store the oversampled data. The buffer contents are then displayed on the HyperTerminal. The HyperTerminal configuration must be 8-bit data, no parity, and 115 200 baud rate. The user can capture them into a *.txt* file and then compare the expected results to the real ones.

To evaluate the new enhanced ADC, a ramp with a 50 Hz frequency and a 1 V amplitude is input to the ADC and sampled using the oversampling algorithm every 100 µs.

The embedded software example related to this method is located in the *WhiteNoiseMethod* folder.

The oversampling algorithm using white noise is run with the same ramp (50 Hz frequency and 1 V amplitude). Both Figure 8 and Figure 9 give the ADC oversampled data as a function of time in µs. Figure 8 is the result of adding one bit while Figure 9 is the result of adding two additional bits to the ADC on-chip resolution.

When the ramp is sampled without using any extra software resolution, with a 3.3 V reference supply, 1 V corresponds to the digital value 1250.

When one additional bit is added, 1 V is sampled as 2500 and when two additional bits are added, 1 V is sampled as 5000.

This means that the environment contains enough noise for this method to work.

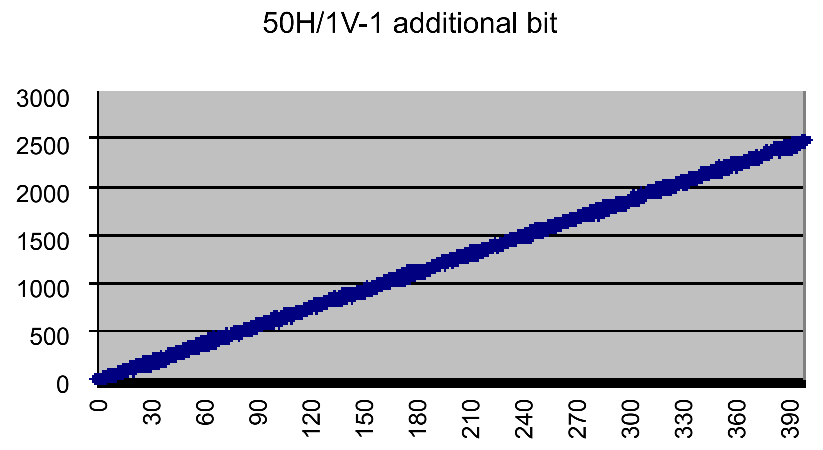


Figure 8. Ramp samples with 1 additional bit

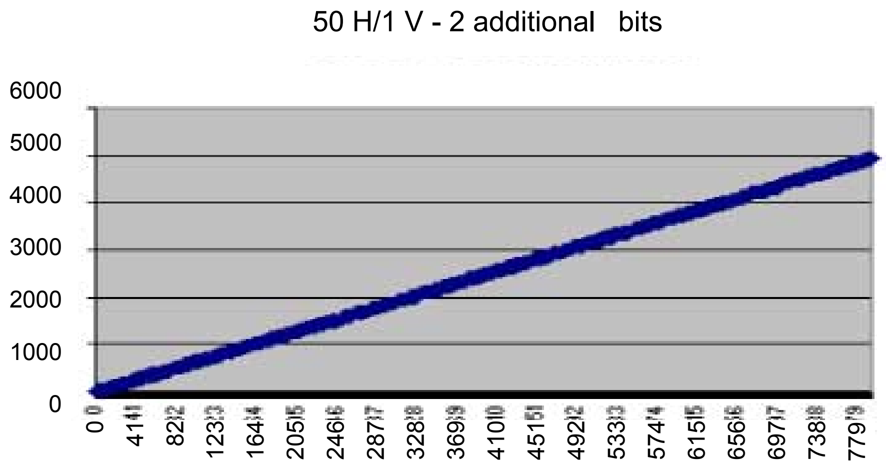


Figure 9. Ramp samples with 2 additional bits

3.2 Oversampling using triangular dither

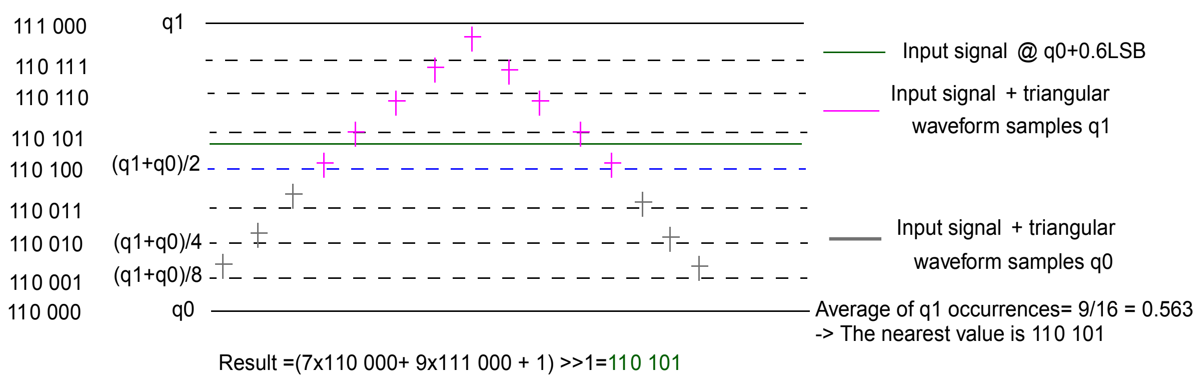
Assuming that the input signal is between two successive quantization steps q0 and q1 during the oversampling period, then the converter may convert it either to q0 or q1. Adding extra p bits of resolution means determining the relative position of the input signal between q0 and q1.

With the addition of an appropriate triangular signal, the quantizer generates a series of q1s and q0s. Averaging the q1 occurrences over a given interval determines the relative position of the input signal between the lower and the higher quantization steps.

The theory states that the best results are achieved when dithering the input signal using a triangular waveform with a period of OSR times the ADC sampling period and an amplitude of n + 0.5 LSB where n = 0,1,2,3.

The theory behind this method is quite complicated, so this Figure 10 is an example to illustrate how this method works. In this example, the ADC on-chip resolution is 3 and 3 extra bits are added by embedded software. The input signal is assumed to have an amplitude of q0+ 0.6LSB (q0 = 6 in this example). To add three additional bits, the input signal is sampled 2.23 times (16 times).

Figure 10. **How to perform oversampling by adding a triangular signal**


111 000
q1
110 111
110 110
110 101
110 100
(q1+q0)/2
110 011
110 010
(q1+q0)/4
110 001
(q1+q0)/8
110 000
q0
Input signal @ q0+0.6LSB
Input signal + triangular waveform samples q1
Input signal + triangular waveform samples q0
Average of q1 occurrences= 9/16 = 0.563
-> The nearest value is 110 101
Result =(7x110 000+ 9x111 000 + 1) >>1=110 101

If the input signal is not correlated with the triangular waveform, then it is demonstrated that the gain in the SNR is equal to:

$$SNR_{Gain} = 20\log\left(\frac{OSR}{2}\right) \tag{9}$$

Therefore, each doubling of the sampling frequency improves the SNR by 6 dB and adds 1 bit of ADC resolution.

In general, to add p-bit extra resolution, the oversampling frequency must be equal to:

$$F_{OVS} = 2 \cdot 2^{p} F_{s} \tag{10}$$

3.2.1 When does this method work?

In order to make this method work, the input signal must not vary by more than ± 0.5 LSB during the oversampling period and must not correlate with the triangular dither signal.

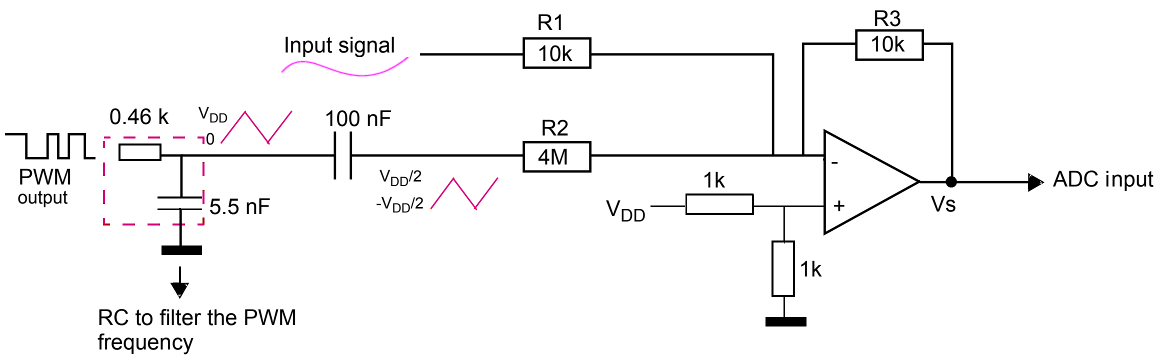
3.2.2 Implementation method on STM32F1, STM32F3, and STM32Lx Series devices

In order to implement the second solution, the following is needed:

- An operational amplifier to perform the sum of the input signal and the triangular waveform. For this, an op-amp inverter/summing stage is required. An STMicroelectronics LMV321 can be used.
- A triangular waveform with a period of OSR times the ADC conversion rate. The user can either use a signal generator or one of the on-chip timers and an RC network to generate this triangular signal. Indeed, the on-chip timer generates a PWM signal with a duty cycle varying from 0 to 100%. This PWM output can be filtered with an RC filter to generate a triangular signal varying from 0 to VDD. In order to generate an amplitude of 0.5 LSB, then the output is first passed through a capacitor (to cut the DC component) and then divided by the prescaler R2/R3 (see Figure 11. Hardware requirements of oversampling by adding a triangular signal). This prescaler is equal to the ADC number of words.

- The input signal must not be changed after the op-amp. For this reason, R1 should be equal to R3.
- The sum of the input signal and the triangular dither is inverted. For this purpose, a 3.3 V offset is required on the positive entry of the op-amp. After the oversampled data are computed, this offset is subtracted to give the input signal estimation with an extra resolution.

Figure 11. Hardware requirements of oversampling by adding a triangular signal



3.2.2.1 *Embedded-software flowchart for oversampling using triangular dither*

The STM32F1 series, STM32F3 series and STM32Lx series on-chip ADC conversion frequency is fixed at 1 MHz. The ADC DMA channel is configured to transfer the number of oversampled inputs from the ADC data register to a buffer in RAM. This transfer is configured to occur one time. At the end of the DMA transfer, an interrupt is triggered and the oversampled result is computed.

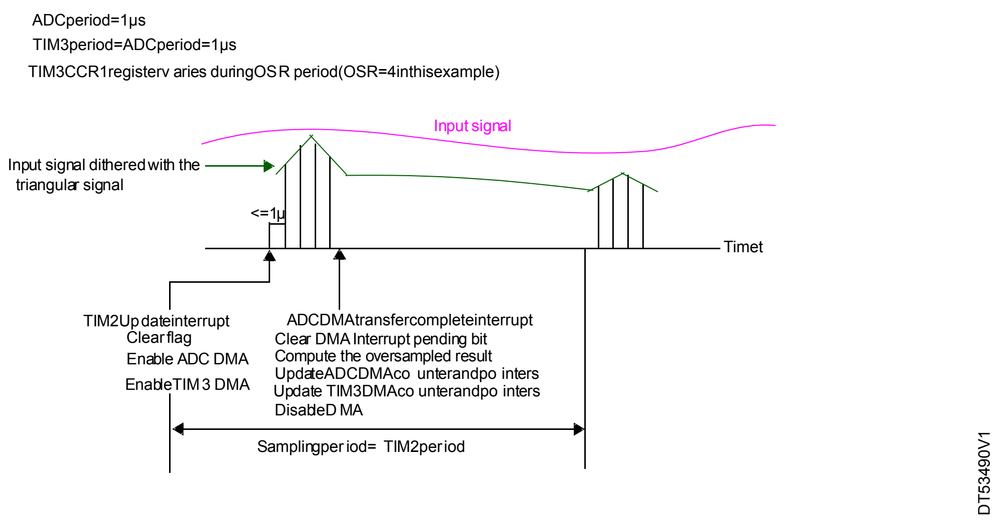
The general-purpose timer TIM2 is used to generate the input signal sampling frequency. For this, the TIM2 reference clock base is configured at 1 µs. Its period determines the input signal sampling period. It is defined in the oversampling.h file by `#define Input_Signal_Sampling_Period`.

The triangular dither is generated using the timer TIM3 configured in PWM mode by updating the Capture Compare Register CCR1. The timer TIM3 period must be equal to the ADC conversion rate and CCR1 must be updated OSR times where OSR is the oversampling factor. To do this, the possible CCR1 values are first computed and stored into a RAM buffer, then the DMA transfer is used to update the CCR1 register, removing the need for interrupts.

Note that the ADC conversion rate limits the oversampling factor. For example, in the case where the ADC is running at 1 MHz, the STM32F1 series is operating at 56 MHz. To have a period of 1 µs, the autoreload register of the timer TIM3 must be equal to 55. The maximum number of additional bits is then 4.

When a TIM2 update interrupt is triggered, the ADC and TIM3 DMA are reenabled and the converted ADC values can be treated to compute the new sample with the extra resolution bits. Figure 12 summarizes the implementation.

Figure 12. **Oversampling using triangular dither flowchart**


ADCperiod=1µs
TIM3period=ADCperiod=1µs
TIM3CCR1registerv aries duringOSR period(OSR=4inthisexample)
Input signal
Input signal dithered with the triangular signal
<=1µ
Timet
TIM2Up dateinterrupt
Clear flag
Enable ADC DMA
EnableTIM3 DMA
ADCDMAtransfercompleteinterrupt
Clear DMA Interrupt pending bit
Compute the oversampled result
UpdateADCDMAco unterandpo inters
Update TIM3DMAco unterandpo inters
DisableD MA
Samplingper iod= TIM2per iod
DT53490V1

For this method to work, the input signal must not vary by more than ±0.5LSB during the oversampling period. This means that for STM32F1 series, STM32F3 series or STM32Lx series devices operating from a 3.3 V VREF+, the maximum allowed variations of the input signal during the oversampling period is ~0.4 mV.

On the other side, a triangular waveform with an amplitude of 0.5 LSB means a 0.4 mV amplitude when operating the STM32F1 series, STM32F3 series or STM32Lx series from a 3.3 V VREF+. The application environment must therefore not be very noisy. Any disturbance of the triangular waveform has an impact on the computed oversampled data.

According to the implementation, the triangular waveform is generated by means of the STM32 timer and an RC filter that cuts the 1 MHz timer frequency. The timer PWM output signal is integrated to provide a triangular signal with a 3.3 V amplitude. The division is done with the ratio R3/R2.

The embedded software related to this method is located in the *TriangularDitherMethod* directory.

3.3 Comparison of software oversampling methods

The first method based on oversampling and averaging using white noise provides a half-bit additional resolution for each doubling of the oversampling rate. The maximum input frequency is drastically decreased with the additional number of additional bits.

For applications where this gain is sufficient, it is a good choice. It requires the presence of white noise in the input signal to make the signal toggle between two adjacent ADC codes. In general, the ADC thermal noise is sufficient and there is no need to add external hardware to act as an external white noise source. This makes the solution more cost effective.

The second method based on dithering the input signal using a triangular waveform and computing its relative position between two quantized steps provides one more bit for each doubling of the oversampling rate. This is twice the improvement given by the first method. To make this method work, the input signal must not correlate with the triangular signal and must not have a variation greater than 0.5 LSB during the oversampling period. However, external hardware is needed to add the input signal and the triangular waveform.

Table 2 summarizes the main differences between the two methods. It is not possible to say that one method is better than the other. Each method has its advantages and limitations. The user must select the one that better meets their application requirements (sampling frequency, number of effective bits, and so on).

Table 2. **Oversampling using white noise versus oversampling using triangular dither**

Implementation conditions	Oversampling using white noise	Oversampling using triangular dither
Oversampling factor to add p bits to the ADC on-chip resolution	4^p	2.2^p
Maximum input signal frequency	f_{ADC} max /(2.4^p)	f_{ADCmax}/(2.2.(2^p))
Dither signal	White noise with an amplitude of at least 1 LSB	Triangular signal with an amplitude of n+0.5LSB

Implementation conditions	Oversampling using white noise	Oversampling using triangular dither
External hardware	External white noise source needed if the input signal noise is not sufficient.	Triangular waveform generator: an on-chip timer can be used. In this case, an RC network is used to filter the PWM frequency. An op-amp is needed to add the triangular waveform and the input signal.

3.4 Hints for software oversampling

3.4.1 What is the maximum number of bits that can be added to the on-chip ADC resolution?

It can be easily shown that increasing the on-chip ADC resolution decreases the maximum frequency component of the input signal.

For example, when using the STM32F1 series, STM32F3 series or STM32Lx series ADC at 1 MHz and two additional bits are required by the application, then the maximum input frequency is divided by:

- 16 when using the white noise method (62.5 kHz)
- 4 when using the triangular dither method (125 kHz).

For the two methods, the estimation of the input signal is done during an oversampling period of OSR times the ADC conversion rate. In the case the ADC is running at 1 MHz, the input signal estimation is done over OSR µs. The signal must not vary by more than 1/2LSB for the white noise method and, by ±0.5LSB for the triangular waveform method.

- When using the white noise method, the maximum number of bits that can be added to the ADC resolution depends only on the input signal.
- When using the triangular dither method, the maximum number of bits that can be added to the ADC resolution does not depend only on the input signal. In fact, the steps defining the triangular signal depend on the ADC and APB frequencies. The timer period should be equal to the ADC rate:
  - $2x(2^{P}) \leq$ timer period
  - $P \leq \log2$ (timer period / 2)

In our example, running the ADC with a rate of 1 µs causes the STM32F1 series to operate at 56 MHz, which means that the timer period must be equal to 55. The maximum number of bits that can be added in this case is 4.

3.4.2 Taking advantage of the STM32 DAC implementation

Some STM32F1 series, STM32F3 series and STM32Lx series devices come with a DAC (digital-to-analog converter) that can be used in the oversampling method to avoid the use of external components.

The DAC can be used in the two oversampling methods as follows:

- In the first method, the DAC can be used to generate a white-noise waveform with programmable amplitude that can be injected into the input signal if noise is not sufficient. The waveform is generated thanks to the implemented pseudorandom algorithm. For more details, refer to the STM32F1 series, STM32F3 series and STM32Lx series reference manuals.
- In the second method, the DAC can be used to generate the triangular waveform. This removes the need for any additional external RC circuitry to filter the timer PWM frequency.

Note: *This is not implemented in the software described in this application note.*

3.4.3 Taking advantage of the STM32F1 series, STM32F3 series and STM32L4 series dual ADC mode implementation

In some STM32F1 series, STM32F3 series and STM32L4 series devices, the dual ADC mode is an interesting feature that allows two ADCs to convert at the same time. Using the dual ADC fast interleave mode, the same channel is converted alternately by ADC2 and ADC1. The time separating two successive samples is 7 ADC clock cycles. The input signal is therefore oversampled faster. In the example described in this application note, a sample is obtained every 1 µs. Using the dual ADC fast interleave mode, it is possible to have a sample every 7 ADC clock cycles that is every 0.5 µs when running the ADC at 14 MHz.

Note: *This hint is not implemented in the software given within the application note.*

3.4.4 Taking advantage of the hardware ADC oversampling implementation

On some STM32 devices, the ADC implements the oversampling feature in hardware. This feature is presented in Section 4 Hardware oversampling, and a comparison between hardware and software oversampling in Section 5 Hardware versus software oversampling comparison.

4 Hardware oversampling

This part presents the hardware oversampling unit available in the products listed in Table 1. Applicable products.

The main benefit that the user can get from the hardware oversampling is increased SNR (signal-to-noise ratio) with less CPU interaction, resulting in overall lower power consumption compared with the software-based implementation.

4.1 Hardware oversampling feature overview

Note: *This section concerns the STM32L4 series and information could slightly differ for other products. The dedicated documentation should be consulted.*

The hardware oversampling engine accumulates the results of ADC conversions. The accumulated output data can be right-shifted (and rounded) to provide selected bit-depth in relation to OSR. The output value is not updated every sampling period, but once N samples are accumulated, therefore, the output data rate is decimated by a factor of OSR.

The result is the average of accumulated samples as follows:

$$Result = \frac{1}{M} \times \sum_{0}^{N-1} Conversion(t_n) \qquad (11)$$

Where both N and M can be adjusted:

- N is the oversampling ratio. It is set with the OVFS[2:0] bits in the ADC_CFGR2 register. It can be a factor between 2x and 256x.
- M is the division coefficient (right bit shift). It is set with the OVSS[3:0] bits in the ADC_CFGR2 register. It can allow to right shift the sum up to 8 bits.

In the case of STM32L4 series, the oversampling engine begins summing N samples. The sum is then right shifted by M bits. The engine keeps the 16 least significant bits after the shift, and rounds the result to the nearest value according to the bits removed by the shifting.

The final result is saved in the ADC_DR data register and because of the 16-bit truncation, it cannot be represented on more than 16 bits.

How to operate the bit-depth obtained with oversampling

When N samples of X bits are accumulated, the result can be coded on up to X + (ln(N) / ln(2)) bits.

For example, if the oversampling ratio N is 256x and the samples accumulated are on 12 bits, the sum of N terms will be on 20 bits since ln(256) / ln(2) = 8 and 12 + 8 = 20.

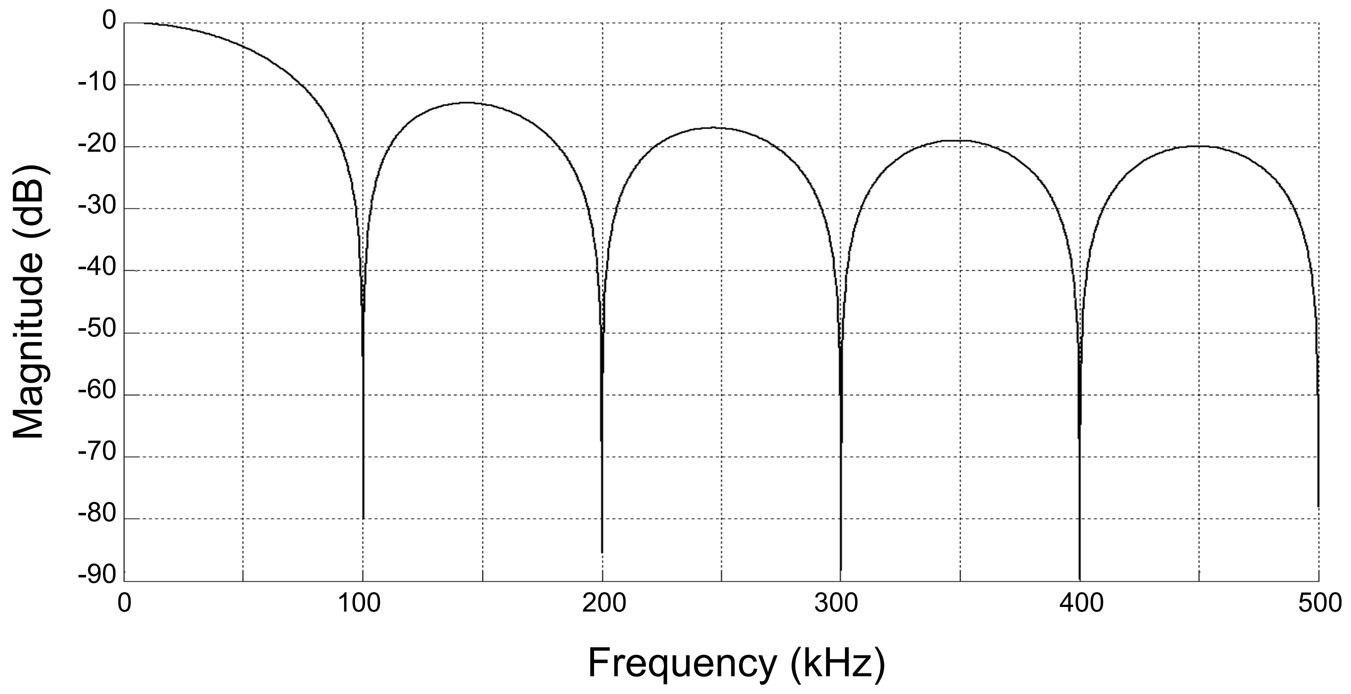
Next, the right shifting, which is up to 8 bits has to be taken into account.

Finally, the bit-depth is given by X + (ln(N) / ln(2)) – M but is limited to 16 bits because of the truncation.

Note: *The number of bits X for a sample depends on the product used and can be found in its datasheet.*

The *Accumulate and average* stage can be thought of as a kind of digital filter (often called accumulate-and-dump). The frequency response of such filter is equivalent to a first order Cascaded-integrator-comb (CIC1) Hogenauer filter. The frequency response in case of sampling frequency 1 MHz and OSR = 10 can be seen in Figure 13. Frequency response of accumulate-and-dump filter.

Figure 13. **Frequency response of accumulate-and-dump filter**



Although this is not a perfect low-pass filter, the very high attenuation of the sampling frequency is a useful property. It is effective in canceling the out-of-band noise resulting in an increased signal-to-noise ratio.

5 Hardware versus software oversampling comparison

The ADC oversampling method can be implemented by hardware or by developing a dedicated software routine.

The advantage of hardware implementation is that the total energy budget needed for processing the ADC acquired samples is reduced in comparison to the software implementation where all the data processing needs to be done by the core. However, the hardware oversampling unit is not available on every product.

Two test projects emulating the common data acquisition tasks have been developed and executed on the same system to evaluate the energy difference and to demonstrate how much energy can be saved by using the hardware oversampling.

5.1 Software implementation

The project demonstrating the software oversampling implementation method consists of the following steps, which are repeated every 100 ms:

1. Configuring the system/data acquisition.
2. Capturing of 64 samples by ADC and storing them in the memory by using DMA while the core is in Sleep low-power mode.
3. Processing the data acquired by the CPU to get an oversampled value.
4. Putting the system in Stop mode for the rest of the 100 ms interval.

5.2 Hardware implementation

The project showing the hardware implementation carries out the same task, except that the data processing is done by the ADC oversampling engine. Hence, the CPU can be inactive during the acquisition and oversampling:

1. Configuring the system/data acquisition.
2. Capturing of 64 samples and processing them by the ADC oversampling engine while the core is in Sleep low-power mode.
3. Putting the system in Stop mode for the rest of the 100 ms interval.

5.3 Results

The energy consumption for the data acquisition and processing task, and the average current consumption for the whole 100 ms period for both demonstration projects are detailed in Table 3.

Table 3. **Comparison of SW and HW implementation of ADC oversampling technique**

Implementation	Data acquisition and processing time	Acquisition task charge		Average current (during 100 ms)
Hardware	6.06 ms	896 pAh	3.23 μC	37 μA
Software	6.80 ms	1099 pAh	3.96 μC	44 μA

The hardware oversampling implementation can save about 20% of the energy consumed to complete the acquisition and data processing task with lower coding effort and CPU time.

6 ENOB (effective number of bits) measurement

The formulas to apply for a desired resolution improvement using each method are presented in the table Table 4.

Table 4. **Formulas for ENOB improvement**

Method	Formula *(X is the ADC resolution OSR is the oversampling ratio)*
Hardware oversampling	Resolution = X + (ln (OSR) / ln(2)) – M *M is the division coefficient of the hardware oversampling engine*
Software oversampling with white noise	Resolution = X + p with OSR = 4^p
Software oversampling with dithering	Resolution = X + p with OSR = $2*2^p$

In practice, this resolution never reaches its theoretical value. A good indication of the efficiency of an ADC is to determine its ENOB (efficient number of bits). This parameter can be considered as a 'real-life' resolution that takes into account potential noise, distortion, and circuit imperfections. It also gives a good indication of its dynamic performance.

It is good practice to measure this parameter to verify that the resolution of an ADC is not degraded too much by its implementation and configuration.

The ENOB can be determined by several methods. In the context of this document, a formula that links it to two other parameters is used: SINAD (signal-to-noise and distortion ratio) and THD+N (total harmonic distortion + noise).

The formulas are as follows:

$$ENOB = \left(SINAD - 1.76 + 20_{\log}\left(Full\_scale\_amp / Input\_amp\right)\right)/6.02 \qquad (12)$$

$$ENOB = \left([THD + N] - 1.76 + 20\log_{10}(Full\_scale\_amp - Input\_amp)\right)/6.02 \qquad (13)$$

Full_scale_amp is the maximum amplitude that can be measured by the ADC.

Input_amp is the amplitude of the signal applied to the ADC.

These formulas result directly from equation (5). The difference is that noise and distortions are taken into account by replacing the SNR by the SINAD or the THD+N, making it closer to a real-life situation. The amplitude of the input signal used for ENOB measurement is also taken into account thanks to the ratio Full-scale amp./ Input amp. Indeed, if the amplitude of the input signal does not fill the full amplitude reading ability of the ADC, this has to be considered when computing a ratio featuring the level of this signal.

Note: If the bandwidth of the measurement is DC to Fs/2 (the Nyquist bandwidth, Fs is the sampling frequency), THD + N is equal to SINAD. That is what we consider for the two formulas above.

The following steps can be followed to measure the ENOB of an ADC:

- With a high precision signal generator, inject a sinusoid on one of the tested ADC channels with a frequency respecting the maximums given in Table 2. Oversampling using white noise versus oversampling using triangular dither for software oversampling, or fADCmax/(2*N) for hardware oversampling with N being the oversampling ratio of the oversampling engine. The sinusoid amplitude should be 90% of the ADC full-scale to avoid saturation.
- Configure the ADC to acquire some samples of the signal. The best is to get a rounded number of the signal period. 4096 is a good example but might need to be adjusted in function of the frequency of the input signal.
- Make the successive binary codes operated by the ADC available for measurement (parallel/serial transmission, file recording…).
- Analyze the ADC measured signal with a frequency analyzer capable to do SINAD or THD+N measurement.
- With the SINAD or THD+N measurement features, get the value for one of these two parameters. Measuring both parameters enable doing a comparison of the ENOB values obtained.
- Apply the formula to determine the ENOB of the ADC (see Eq. (12) or Eq. (13)).

To provide practical data and be able to analyze the effect of oversampling on the ENOB, the above steps have been followed with the following equipment and tools.

The input sinusoid has been generated with the analog output of the **Audio Precision AP2722 Audio Analyzer**. Several input frequencies have been tested to analyze the effect on the ENOB obtained. The signal output by the AP2722 is 0-centered, so a conversion stage is needed to set its amplitude between 0 and 3.0 V (VDD=VDDA=3.3 V on STMicroelectronics EVAL and Nucleo boards).

Note: This conversion stage behaves like an HP filter and influences the signal measured by the ADC (lower signal resolution for lower frequencies). This can give a good representation of a real-life use case.

The STM32L476G-EVAL board has been used to process the ADC measurement and transmitting/recording it. The ADC sampling is done on the pin PA4 linked to the STM32L4 ADC1 Channel 9 and to ground through a 4.7nF capacitor to filter high frequency noise. This pin is available on the connector CN7 of the EVAL board.

The application running on the STM32L476G-EVAL board saves 4096 ADC samples in RAM thanks to the DMA peripheral.

The oversampling unit is used to analyze its effect and configured as presented in table 6.

A timer is set up to trigger the transfer of each sample (even the ones used for oversampling). The frequency of this timer is adjusted according to the oversampling configuration wanted.

When the 4096 samples are saved, they are transferred through the STM32 UART interface to be recovered in a file on a PC thanks to a Python script.

The resulting file is formatted so that it can be processed with MATLAB®.

The sampling rate chosen for the test is 12.5 kHz (when oversampling is used, this is the final sampling rate) to be able to keep a constant ADC clock frequency (80 MHz) and sampling time (12.5 cycles).

MATLAB® enables computing the SINAD or THD+N of the ADC signal. It has a native sinad function that is used to analyze the ADC signal saved into the file created previously. Then, applying eq.X gives us the ENOB measured.

To emphasize oversampling effects, each oversampling ratio possible has been tested while fixing the right shift coefficient to target the best resolution offered.

Table 5 presents the resolution that can be achieved in function of the oversampling ratio and the right shifting. 16 bits have been targeted. For the ratio values 2, 4 and 8 a left shifting (respectively by 3, 2 and 1 bit) has been processed to achieve the 16-bite target.

Table 5. Theoretical ENOB values for hardware oversampling unit versus configuration

OSR\M	Hardware oversampling unit coefficient (M)(1)								
	0	1	2	3	4	5	6	7	8
2	13	12	11	10	9	8	7	6	5
4	14	13	12	11	10	9	8	7	6
8	15	14	13	12	11	10	9	8	7
16	16	15	14	13	12	11	10	9	8
32	**17**	16	15	14	13	12	11	10	9
64	**18**	**17**	16	15	14	13	12	11	10
128	**19**	**18**	**17**	16	15	14	13	12	11
256	**20**	**19**	**18**	**17**	16	15	14	13	12

1. *Bold entries have no significance since the hardware oversampling unit output is limited to 16-bit data width.*

Table 6. **Practical ENOB measurement with the hardware oversampling unit versus configuration**

OSR	Hardware oversampling unit coefficient (M)[1]								
	0	1	2	3	4	5	6	7	8
2	13	12	11	10	9	8	7	6	5
4	14	13	12	11	10	9	8	7	6
8	15	14	13	12	11	10	9	8	7
16	16	15	14	13	12	11	10	9	8
32	-	16	15	14	13	12	11	10	9
64	-	-	16	15	14	13	12	11	10
128	-	-	-	16	15	14	13	12	11
256	-	-	-	-	16	15	14	13	12

1. *Bold entries have no significance since the hardware oversampling unit output is limited to 16-bit data width.*

Table 7. **Practical ENOB measurement with the hardware oversampling unit versus configuration**

OSR	Software oversampling with white noise		Software oversampling with dithering	
	Theoretical resolution	Practical ENOB	Theoretical resolution	Practical ENOB
2	-	-	12	-
4	13	-	13	-
8	-	-	14	-
16	14	-	15	-
32	-	-	16	-
64	15	-	17	-
128	-	-	18	-
256	16	-	19	-

As mentioned before, the signal test input is a 3.0Vpp sinusoid and the sampling rate is 12.5 kHz. Thus, to respect the Nyquist criteria, the following frequencies have been tested: 500 Hz, 1 kHz, 1.5 kHz, 2 kHz, 2.5 kHz.

Table 8. **Effect of oversampling on ENOB**

OVS ration	OVS right shift	OVS left shift	ENOB				
			500 Hz	1 kHz	1.5 kHz	2 kHz	2.5 kHz
None	None	None	10.4126	10.3622	10.3543	10.2774	10.3084
2	None	3	10.6245	10.6302	10.9172	10.8618	10.9524
4	None	2	10.9567	11.2234	11.2249	11.3322	11.3531
8	None	1	11.1692	11.3454	11.4646	11.5817	11.7653
16	None	None	11.2158	11.4962	11.6551	11.8414	11.9138
32	1	None	11.2718	11.6126	11.8103	12.0408	12.2725
64	2	None	11.3109	11.6220	11.8968	12.1124	12.3626
128	3	None	11.3259	11.6568	11.9050	12.1579	12.6038
256	4	None	11.3582	11.7032	11.9301	12.2419	12.8259

Note: *For reference, after the conversion stage and at the ADC pin level, the following THD+N values were measured: -82.5 dB at 500 Hz (this is equivalent to 13.55 ENOB according to eq. Y), -93 dB at 1 kHz (15.29 ENOB), -96 dB at 1.5 kHz (15.79 ENOB), -97 dB at 2 kHz (15.96 ENOB) and -98 dB at 2.5 kHz (16.12 ENOB). At the audio precision analyzer output, which is before the conversion stage, -105 dB were measured (17.28 ENOB).*

In the STM32L476 datasheet, it is given that the typical ENOB of the ADC is 10.5 (the ADC is configured as single-ended).

Thus, Table 8 shows that it is possible to get over this typical value and even over the real ADC resolution. However, the theoretical 16-bit target stays far from the results and corresponds more to an idea of the performance of the oversampling configuration.

The result also highlights that a higher oversampling ratio gives a better ENOB despite limiting the sampling frequency and so the input signal frequency.

7 Conclusion

This application note has explained the basics of the oversampling technique used to improve the SNR performances (and thus the effective resolution) of ADCs integrated in most of the STM32 microcontrollers.

The cornerstones of the oversampling technique are:

- The RMS quantization noise of an ADC is $q / \sqrt{(12)}$, over the Nyquist bandwidth (q is the ADC quantum: LSB value)
- If the wanted bandwidth is smaller than the Nyquist bandwidth, the quantization noise is reduced in proportion by using a filter to remove the out of band noise
- Dithering can be used if the quantization noise does not behave like a wideband noise

The hardware implementation of the ADC oversampling technique reduces the time and energy needed by the CPU for the data processing tasks. It results in lowering the overall power consumption.

When the hardware oversampling unit is not available on the STM32 used, it is still possible to implement entirely the technique via software as it has been presented in this document.

The effect of oversampling on the effective ADC resolution (ENOB) has also been analyzed. With oversampling it is possible to get the effective resolution over the real one.

Revision history

Table 9. **Document revision history**

Date	Version	Changes
25-Sep-2023	1	Initial version.

Contents

List of tables

List of figures