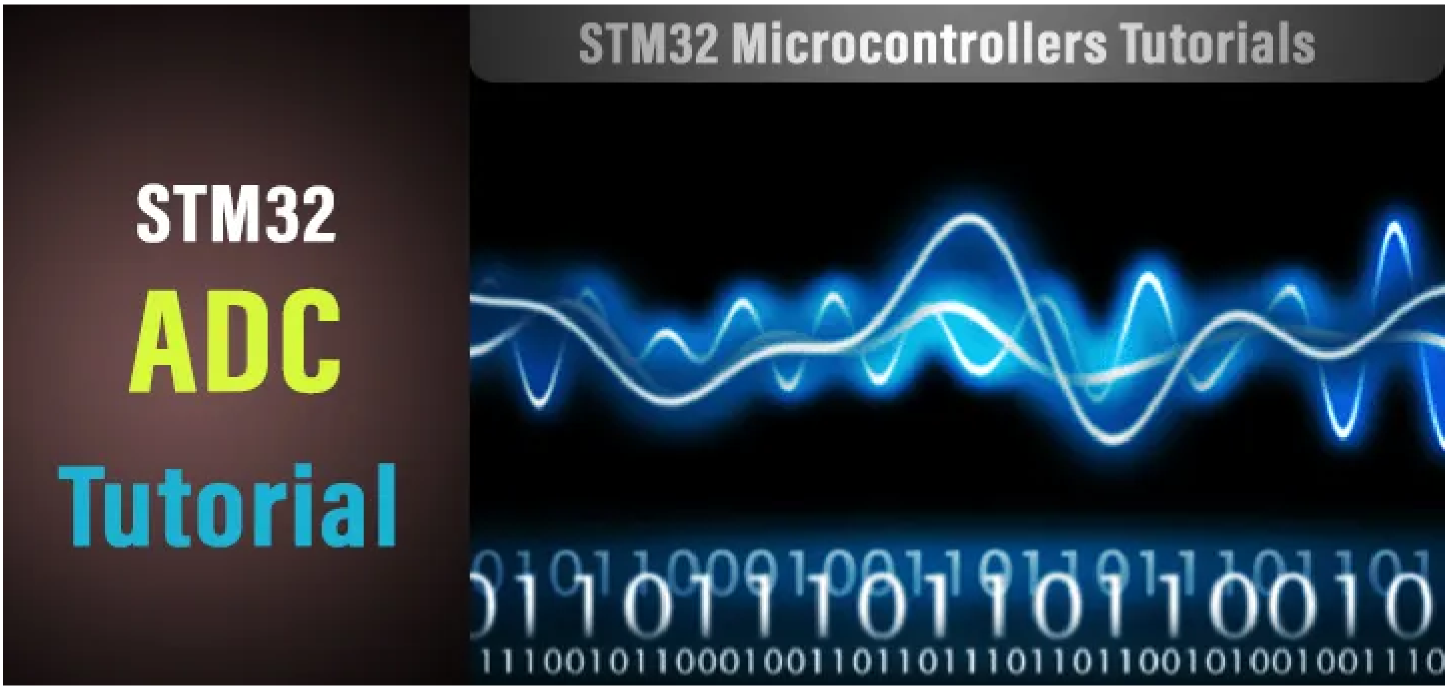
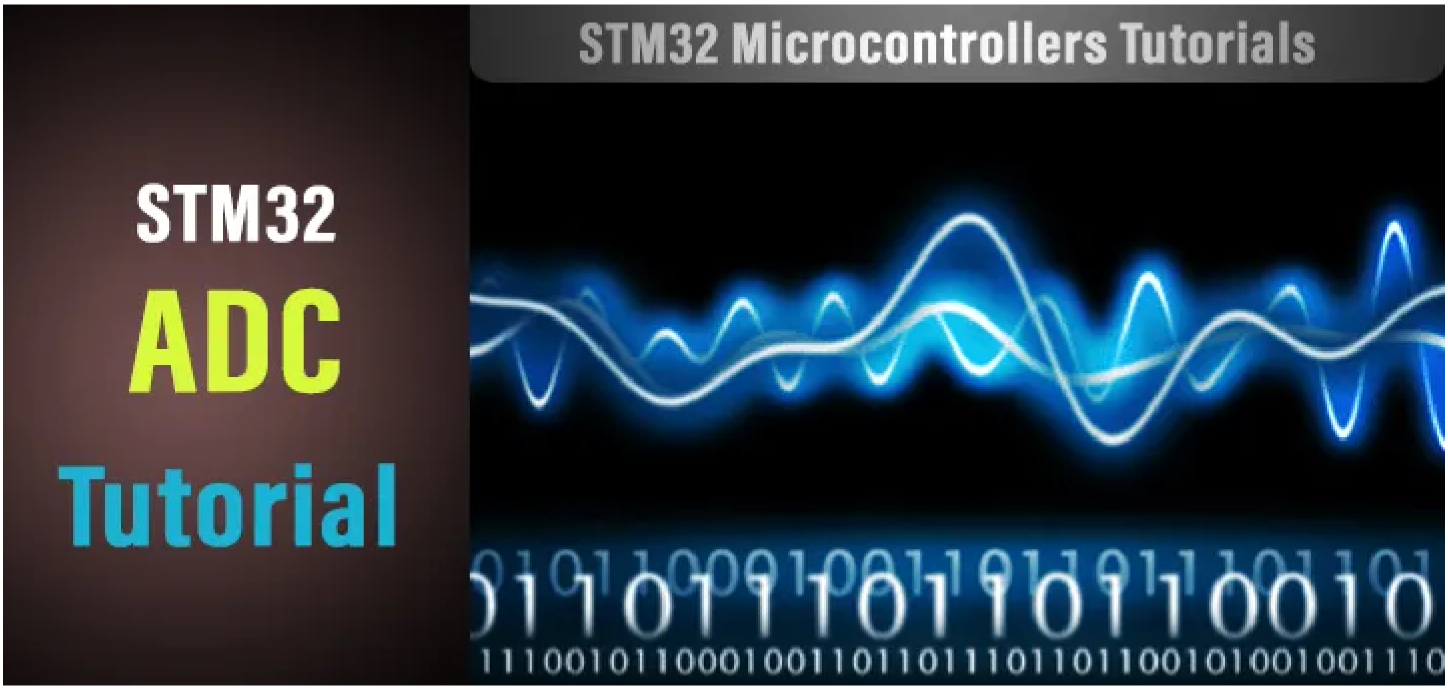


STM32 ADC Tutorial – Complete Guide With Examples

by Khaled Magdy

◀	Previous Tutorial	Tutorial 23	Next Tutorial	▶
	STM32 ADC Tutorial – Complete Guide With Examples			
	STM32 Course Home Page 🏠			



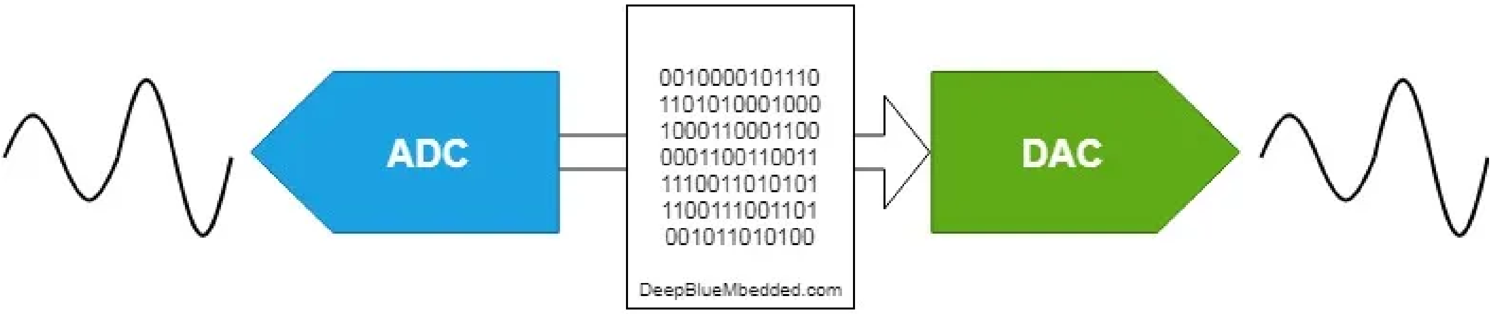
In this tutorial, we'll discuss the STM32 ADC (Analog-To-Digital Converter) module. Starting with an introduction for the ADC as a digital circuit and then shifting the attention to the STM32 ADC hardware and its features. We'll get into the functional description for the ADC in STM32 microcontrollers, how it works, and how to configure it and make the best use of it. And let's get right into it!

Tutorial Contents

Analog-To-Digital Converters (ADC) Preface

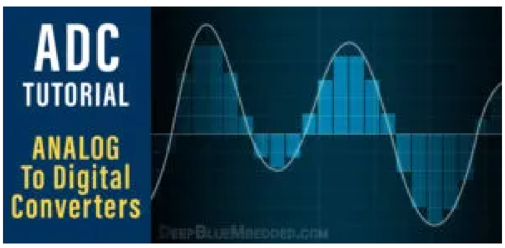
An ADC (Analog-To-Digital) converter is an electronic circuit that takes in an analog voltage as input and converts it into digital data, a value that represents the voltage level in binary code. The ADC samples the analog input whenever you trigger it to start conversion. And it performs a process called quantization so as to decide on the voltage level and its binary code that gets pushed in the output register.

The ADC does the counter operation that of a DAC, while an ADC (A/D) converts analog voltage to digital data the DAC (D/A) converts digital numbers to the analog voltage on the output pin.



The ADC is one of the most expensive electronic components especially when it does have a high sampling rate and high resolution. Therefore, it's a valuable resource in microcontrollers and different manufacturers provide us (the firmware engineers) with various features so as to make the best use of it. And the flexibility also to make a lot of decisions like sacrificing resolution in exchange for a higher resolution or having the ADC to trigger on an internal timer signal to periodically sample the analog channels, and much more as we'll see in this tutorial.

For those who like to have a solid introduction in ADC, how it works at the low level, different types of ADCs, ADC errors, equations, and all other details. The ADC Tutorial down below is a complete introductory guide for this topic and highly recommended.



STM32 ADC Brief

The STM32F103C8 (Blue Pill) & STM32F432KC have a 12-bit ADC which is a successive approximation analog-to-digital converter. It has up to 18 multiplexed channels allowing it to measure signals from sixteen external and two internal sources. A/D conversion of the various channels can be performed in single, continuous, scan, or discontinuous mode. The result of the ADC is stored in a left-aligned or right-aligned 16-bit data register.

The analog watchdog feature allows the application to detect if the input voltage goes outside the user-defined high or low thresholds. The ADC input clock is generated from the PCLK2 clock divided by a Prescaler and it must not exceed 14 MHz.

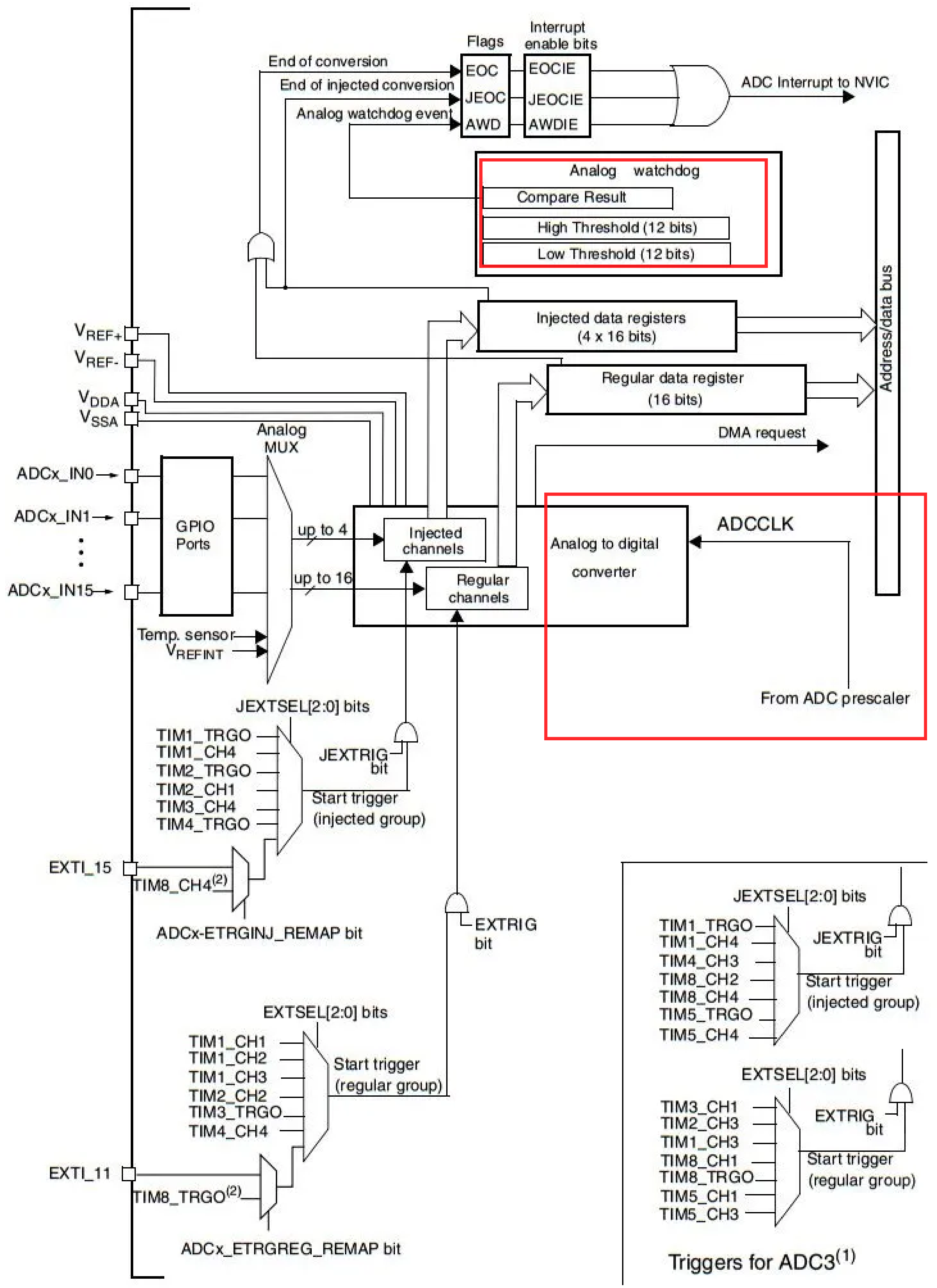
ADC Features

- 12-bit resolution
- Interrupt generation at End of Conversion, End of Injected conversion and Analog watchdog event
- Single and continuous conversion modes
- Scan mode for automatic conversion of channel 0 to channel 'n'
- Self-calibration
- Data alignment with in-built data coherency
- Channel by channel programmable sampling time
- External trigger option for both regular and injected conversion
- Discontinuous mode
- Dual-mode (on devices with 2 ADCs or more)
- ADC conversion time: 1 µs at 56 MHz (1.17 µs at 72 MHz)
- ADC supply requirement: 2.4 V to 3.6 V
- ADC input range: $V_{REF}– \leq V_{IN} \leq V_{REF}+$
- DMA request generation during regular channel conversion

STM32 ADC Functional Description

STM32 ADC Block Diagram

Single ADC block diagram



The ADC Clock

The ADCCLK clock provided by the Clock Controller is synchronous with the PCLK2 (APB2 clock). The RCC controller has a dedicated programmable Prescaler for the ADC clock, and it must not exceed 14 MHz.

Channel Selection

There are 16 multiplexed channels. It is possible to organize the conversions in two groups: regular and injected. A group consists of a sequence of conversions that can be done on any channel and in any order. For instance, it is possible to do the conversion in the following order: Ch3, Ch8, Ch2, Ch2, Ch0, Ch2, Ch2, Ch15.

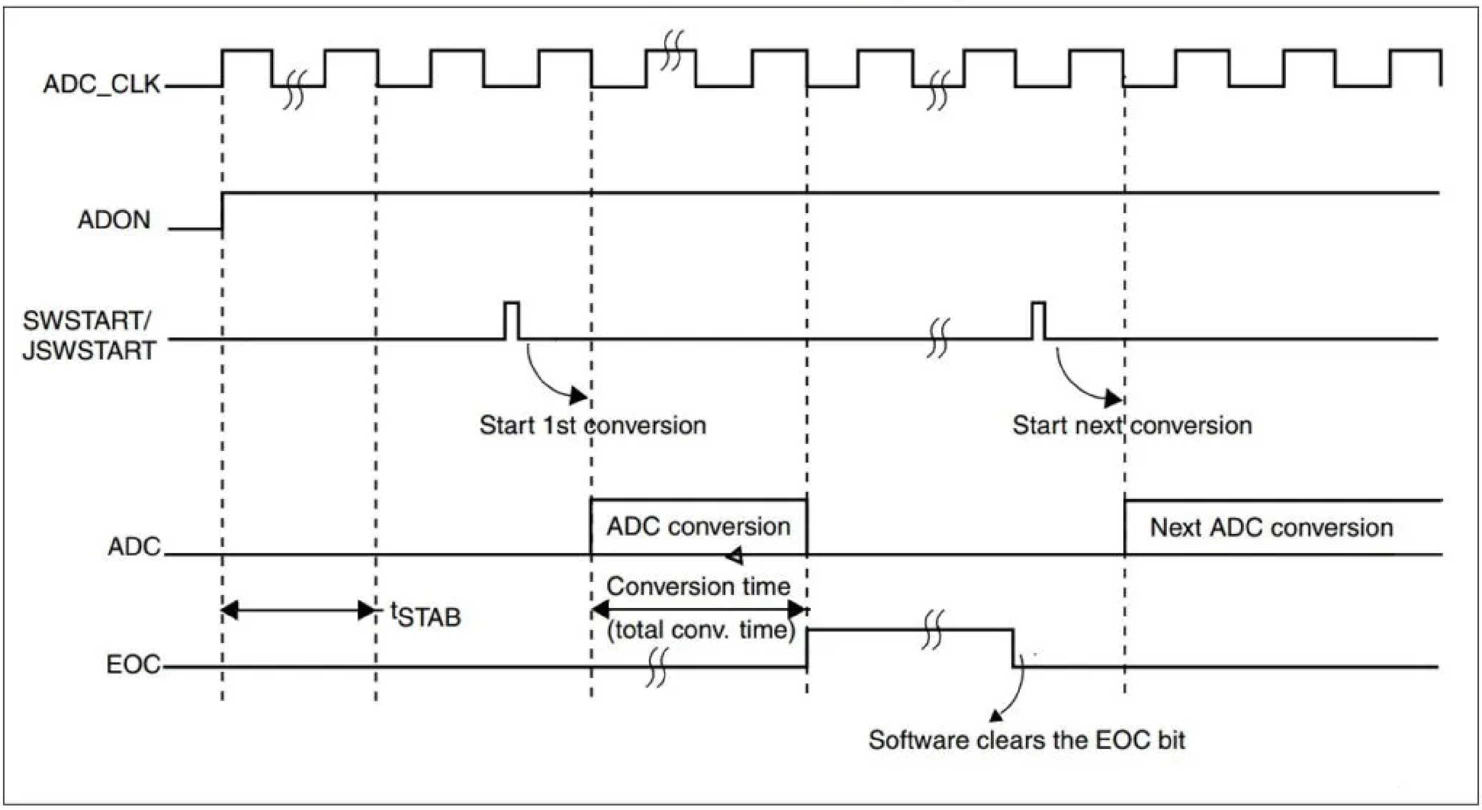
The **Regular Group** is composed of up to 16 conversions. The regular channels and their order in the conversion sequence must be selected in the ADC_SQRx registers. The total number of conversions in the regular group must be written in the L[3:0] bits in the ADC_SQR1 register.

The **Injected Group** is composed of up to 4 conversions. The injected channels and their order in the conversion sequence must be selected in the ADC_JSQR register. The total number of conversions in the injected group must be written in the L[1:0] bits in the ADC_JSQR register.

ADC Conversion Time & Timing Diagram

the ADC needs a stabilization time of t_{STAB} before it starts converting accurately. After the start of ADC conversion and after 14 clock cycles, the EOC flag is set and the 16-bit ADC Data register contains the result of the conversion.

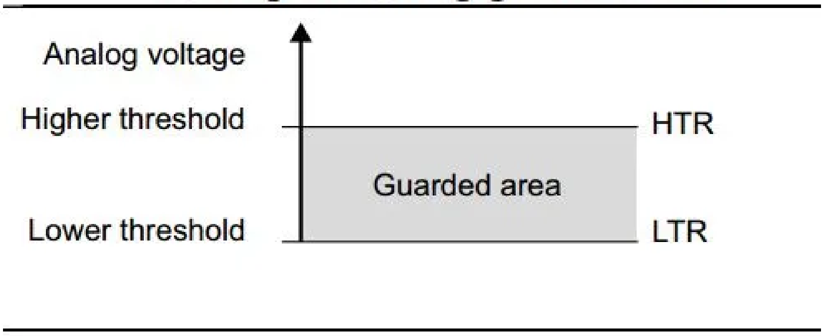
Timing diagram



The ADC Analog Watchdog (AWD)

The AWD analog watchdog status bit is set if the analog voltage converted by the ADC is below a low threshold or above a high threshold. These thresholds are programmed in the 12 least significant bits of the ADC_HTR and ADC_LTR 16-bit registers. An interrupt can be enabled by using the AWDIE bit in the ADC_CR1 register. The threshold value is independent of the alignment selected. And the analog watchdog can be enabled on one or more channels.

Analog watchdog guarded area



ADC Result Data Alignment

ALIGN bit in the ADC_CR2 register selects the alignment of data stored after conversion. Data can be left or right-aligned as shown in the diagram below.

The injected group channels converted data value is decreased by the user-defined offset written in the ADC_JOFRx registers so the result can be a negative value. The SEXT bit is the extended sign value. For regular group channels, no offset is subtracted so only twelve bits are significant.

Right alignment of data

Injected group

SEXT	SEXT	SEXT	SEXT	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

Regular group

0	0	0	0	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

Left alignment of data

Injected group

SEXT	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0

Regular group

D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0	0

Single Conversion Mode

In Single Conversion mode, the ADC does one conversion. This mode is started either by setting the ADON bit in the ADC_CR2 register (for a regular channel only) or by an external trigger (for a regular or injected channel), while the CONT bit is 0. Once the conversion of the selected channel is complete:

- If a regular channel was converted:
  – The converted data is stored in the 16-bit ADC_DR register
  – The EOC (End Of Conversion) flag is set
  – and an interrupt is generated if the EOCIE is set.
- If an injected channel was converted:
  – The converted data is stored in the 16-bit ADC_DRJ1 register
  – The JEOC (End Of Conversion Injected) flag is set
  – and an interrupt is generated if the JEOCIE bit is set.

The ADC is then stopped.

Continuous Conversion Mode

In continuous conversion mode, ADC starts another conversion as soon as it finishes one. This mode is started either by an external trigger or by setting the ADON bit in the ADC_CR2 register, while the CONT bit is 1. After each conversion:

- If a regular channel was converted:
  – The converted data is stored in the 16-bit ADC_DR register
  – The EOC (End Of Conversion) flag is set
  – An interrupt is generated if the EOCIE is set.
- If an injected channel was converted:
  – The converted data is stored in the 16-bit ADC_DRJ1 register
  – The JEOC (End Of Conversion Injected) flag is set
  – An interrupt is generated if the JEOCIE bit is set.

Scan Mode

This mode is used to scan a group of analog channels. A single conversion is performed for each channel of the group. After each end of conversion, the next channel of the group is converted automatically. If the CONT bit is set, conversion does not stop at the last selected group channel but continues again from the first selected group channel.

When using scan mode, DMA bit must be set and the direct memory access controller is used to transfer the converted data of regular group channels to SRAM after each update of the ADC_DR register. The injected channel converted data is always stored in the ADC_JDRx registers.

Discontinuous Mode

This mode is enabled by setting the DISCEN bit in the ADC_CR1 register. It can be used to convert a short sequence of n conversions (n <=8) which is a part of the sequence of conversions selected in the ADC_SQRx registers. The value of n is specified by writing to the DISCNUM[2:0] bits in the ADC_CR1 register.

When an external trigger occurs, it starts the next n conversions selected in the ADC_SQRx registers until all the conversions in the sequence are done. The total sequence length is defined by the L[3:0] bits in the ADC_SQR1 register.

ADC Conversion On External Triggers

Conversion can be triggered by an external event (e.g. timer capture, EXTI line). If the EXTTRIG control bit is set then external events are able to trigger a conversion. The EXTSEL[2:0] and JEXTSEL[2:0] control bits allow the application to select decide which out of 8 possible events can trigger conversion for the regular and injected groups.

When an external trigger is selected for ADC regular or injected conversion, only the rising edge of the signal can start the conversion. Down below is a table for the possible options for ADC1 & 2 external trigger inputs to start a conversion on regular group channels.

External trigger for regular channels for ADC1 and ADC2

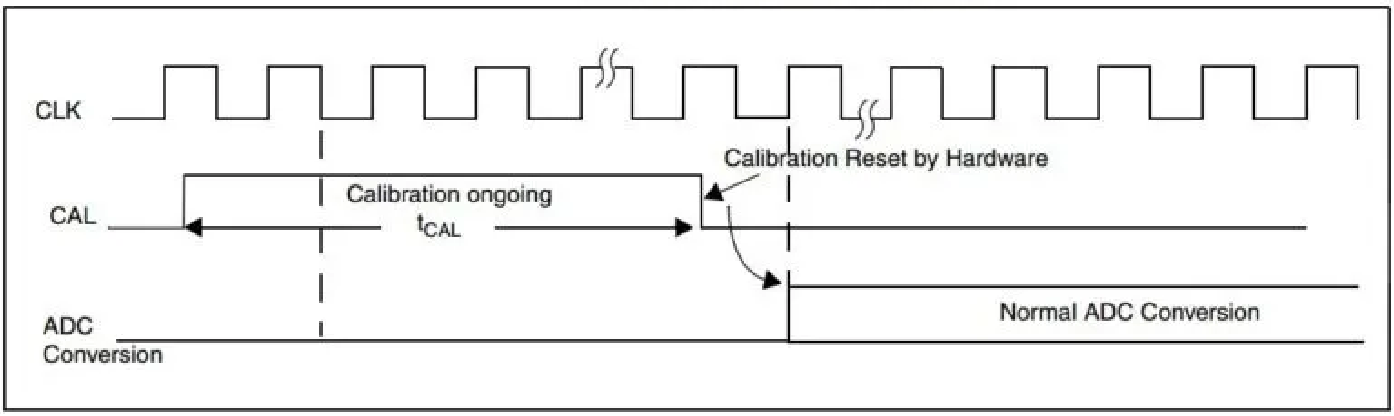
Source	Type	EXTSEL[2:0]
TIM1_CC1 event	Internal signal from on-chip timers	000
TIM1_CC2 event		001
TIM1_CC3 event		010
TIM2_CC2 event		011
TIM3_TRGO event		100
TIM4_CC4 event		101
EXTI line 11/TIM8_TRGO event(1)(2)	External pin/Internal signal from on-chip timers	110
SWSTART	Software control bit	111

STM32 ADC Calibration

The ADC has a built-in self-calibration mode. Calibration significantly reduces accuracy errors due to internal capacitor bank variations. During calibration, an error-correction code (digital word) is calculated for each capacitor, and during all subsequent conversions, the error contribution of each capacitor is removed using this code.

Calibration is started by setting the CAL bit in the ADC_CR2 register. Once calibration is over, the CAL bit is reset by hardware and normal conversion can be performed. It is recommended to calibrate the ADC once at power on. The calibration codes are stored in the ADC_DR as soon as the calibration phase ends. It is recommended to perform a calibration after each power-up.

Calibration timing diagram



The STM32 HAL does provide a function within the ADC APIs dedicated to starting the calibration process and as said before it's a recommended step after initializing the ADC hardware at the system power-up.

STM32 ADC Sampling Time

ADC samples the input voltage for a number of ADC_CLK cycles which can be modified using the SMP[2:0] bits in the ADC_SMPR1 and ADC_SMPR2 registers. Each channel can be sampled with different sample times.

The **Total ADC Conversion Time** is calculated as follows:

Tconv = Sampling time + 12.5 cycles

Example:

With an ADCCLK = 14 MHz and a sampling time of 1.5 cycles: Tconv = 1.5 + 12.5 = 14 cycles = 1 μs

The **ADC Sampling Rate (Frequency)** is calculated using this formula:

SamplingRate = 1 / Tconv

For The Previous example where Tconv = 1μs, The samplingRate = 1000000 = 1Ms/sec

STM32 ADC Resolution, Reference, Formulas

STM32 ADC Resolution

The STM32 ADC has a resolution of 12-Bit which results in a total conversion time of SamplingTime+12.5 clock cycles. However, higher sampling rates can be achieved by sacrificing the high-resolution. Therefore, the resolution can be dropped down to 10-Bit, 8-Bit, or 6-Bit, and hence the conversion time is much shorter and the sampling rate increases. This can be configured and implemented in software by the programmer and the STM32 HAL does provide APIs to set all the ADC parameters including its resolution.

ADC Reference Voltage

The ADC reference voltage pins are defined in the datasheet and assumed to be connected to a voltage level in a certain range. This is shown in the table below. You can choose to set the reference voltage to its maximum allowable level for a wider range of conversion but less voltage measurement resolution. Or alternatively, you can set the reference voltage to the minimum allowable value for better voltage reading resolution.

Name	Signal type	Remarks
V_{REF+}	Input, analog reference positive	The higher/positive reference voltage for the ADC, 2.4 V $\leq V_{REF+} \leq V_{DDA}$
V_{DDA} (1)	Input, analog supply	Analog power supply equal to V_{DD} and 2.4 V $\leq V_{DDA} \leq$ 3.6 V
V_{REF-}	Input, analog reference negative	The lower/negative reference voltage for the ADC, $V_{REF-} = V_{SSA}$
V_{SSA} (1)	Input, analog supply ground	Ground for analog power supply equal to V_{SS}

If you're using a development board, you may need to check out its schematic diagram as it may not be connecting the ADC Vref at all or connecting it to a 2.5v for example, so the ADC will saturate and give you 4096 before the input analog voltage reaches 3.3v and you're wondering why! it may be because the reference voltage is set to a value less than 3.3v, so it's something to consider.

STM32 ADC Formulas

ADC Conversion Time

Tconv = Sampling time + 12.5 cycles

ADC Sampling Rate

SamplingRate = 1 / Tconv

ADC Result Voltage (Analog Input Value)

Vin = ADC_Res x (Reference Voltage / 4096)

Where Reference Voltage = (V_{REF}+) – (V_{REF}-)

ADC & DMA

Since converted regular channels value are stored in a unique data register, it is necessary to use DMA for the conversion of more than one regular channel. This avoids the loss of data already stored in the ADC_DR register.

Only the end of conversion of a regular channel generates a DMA request, which allows the transfer of its converted data from the ADC_DR register to the destination location selected by the user.

STM32 ADC Interrupts

An interrupt can be produced on the end of conversion for regular and injected groups and when the analog watchdog status bit is set. Separate interrupt enable bits are available for flexibility.

ADC interrupts

Interrupt event	Event flag	Enable Control bit
End of conversion regular group	EOC	EOCIE
End of conversion injected group	JEOC	JEOCIE
Analog watchdog status bit is set	AWD	AWDIE

Different Ways To Read STM32 ADC

1 – The Polling Method

It’s the easiest way in code in order to perform an analog to digital conversion using the ADC on an analog input channel. However, it’s not an efficient way in all cases as it’s considered to be a blocking way of using the ADC. As in this way, we start the A/D conversion and wait for the ADC until it completes the conversion so the CPU can resume processing the main code.

2 – The Interrupt Method

The interrupt method is an efficient way to do ADC conversion in a non-blocking manner, so the CPU can resume executing the main code routine until the ADC completes the conversion and fires an interrupt signal so the CPU can switch to the ISR context and save the conversion results for further processing.

However, when you’re dealing with multiple channels in a circular mode or so, you’ll have periodic interrupts from the ADC that are too much for the CPU to handle. This will introduce jitter injection and interrupt latency and all sorts of timing issues to the system. This can be avoided by using DMA.

3 – The DMA Method

Lastly, the DMA method is the most efficient way of converting multiple ADC channels at very high rates and still transfers the results to the memory without CPU intervention which is so cool and time-saving technique.

Note That: Code Examples For The 3 Methods Are Provided in The Next Tutorial. After completion, click the next tutorial to see the examples.

STM32 ADC Conversion Errors

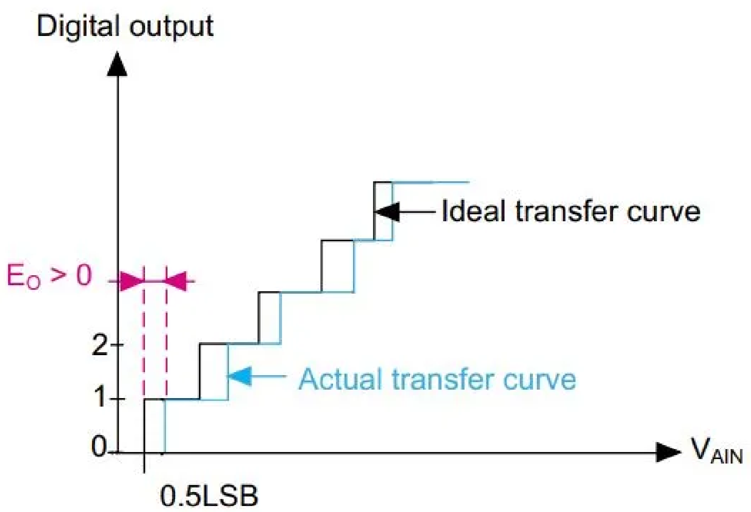
This section lists the main error types that have an effect on A/D conversion accuracy. These types of errors occur in all A/D converters and conversion quality depends on eliminating the source of each error of them.

1 ADC Errors Due To The ADC Itself

1.1 – ADC Offset Error

The offset error is the deviation between the first actual transition and the first ideal transition. The first transition occurs when the digital ADC output changes from 0 to 1. Ideally, when the analog input ranges between 0.5 LSB and 1.5 LSB, the digital output should be 1. Still, ideally, the first transition occurs at 0.5 LSB. The offset error is denoted by E_O. The offset error can easily be calibrated by the application firmware.

Positive offset error representation

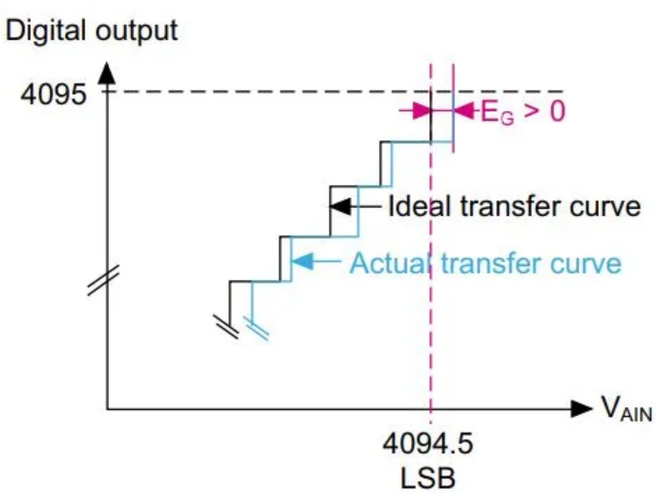


1.2- ADC Gain Error

The gain error is the deviation between the last actual transition and the last ideal transition. It is denoted by E_G. The last actual transition is the transition from 0xFFE to 0xFFF. Ideally, there should be a transition from 0xFFE to 0xFFF when the analog input is equal to V_{REF}+ – 0.5 LSB. So for V_{REF}+= 3.3 V, the last ideal transition should occur at 3.299597 V. If the ADC provides the 0xFFF reading for $V_{AIN} < V_{REF}$+ – 0.5 LSB, then a negative gain error is obtained. The gain error is obtained by the formula below:

E_G = Last actual transition – ideal transition

Positive gain error representation

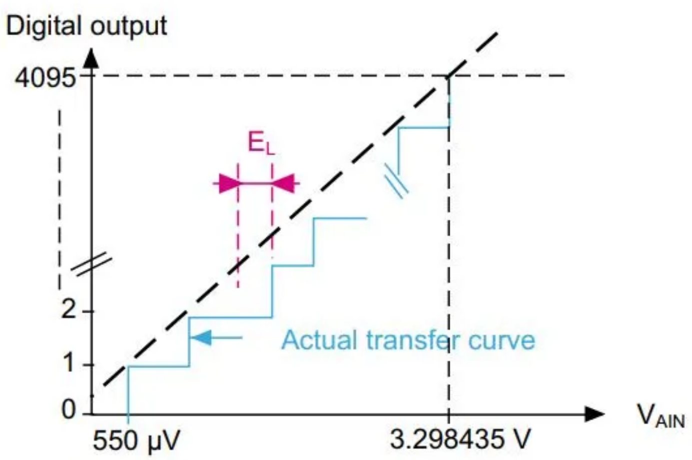


1.3 – Integral Linearity Error

The integral linearity error is the maximum deviation between any actual transition and the endpoint correlation line. The ILE is denoted by E_L. The endpoint correlation line can be defined as the line on the A/D transfer curve that connects the first actual transition with the last actual transition.

E_L is the deviation from this line for each transition. The endpoint correlation line thus corresponds to the actual transfer curve and has no relation to the ideal transfer curve. The ILE is also known as the integral non-linearity error (INL). The ILE is the integral of the DLE over the whole range.

Integral linearity error representation



2 ADC Errors Due To The Environment

2.1 – ADC Reference Voltage Noise

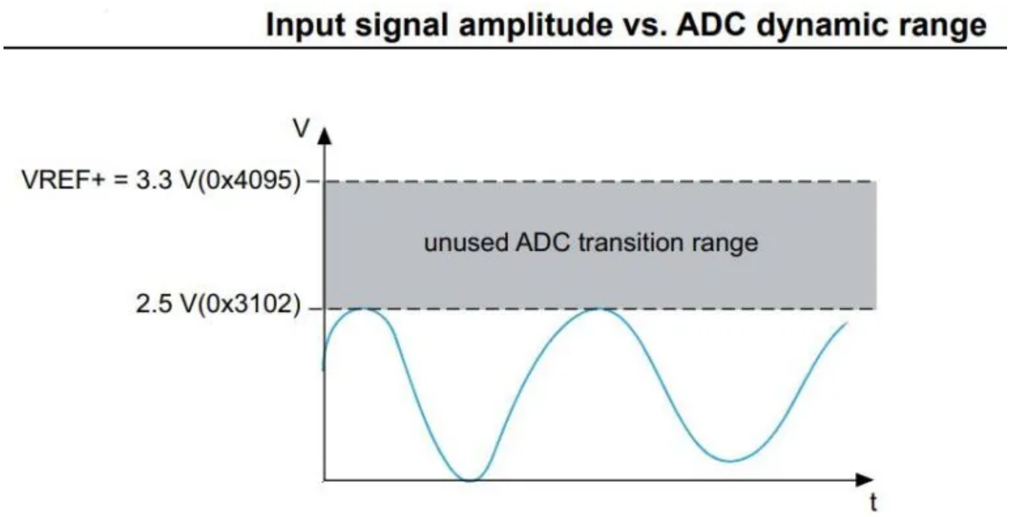
As the ADC output is the ratio between the analog signal voltage and the reference voltage, any noise on the analog reference causes a change in the converted digital value. V_{DDA} analog power supply is used on some packages as the reference voltage (V_{REF}+), so the quality of the V_{DDA} power supply has an influence on ADC error.

2.2 – Analog Input Signal Noise

Small but high-frequency signal variation can result in big conversion errors during sampling time. This noise is generated by electrical devices, such as motors, engine ignition, power lines. It affects the source signal (such as sensors) by adding an unwanted signal. As a consequence, the ADC conversion results are not accurate.

2.3 – ADC Dynamic Range Bad Matching

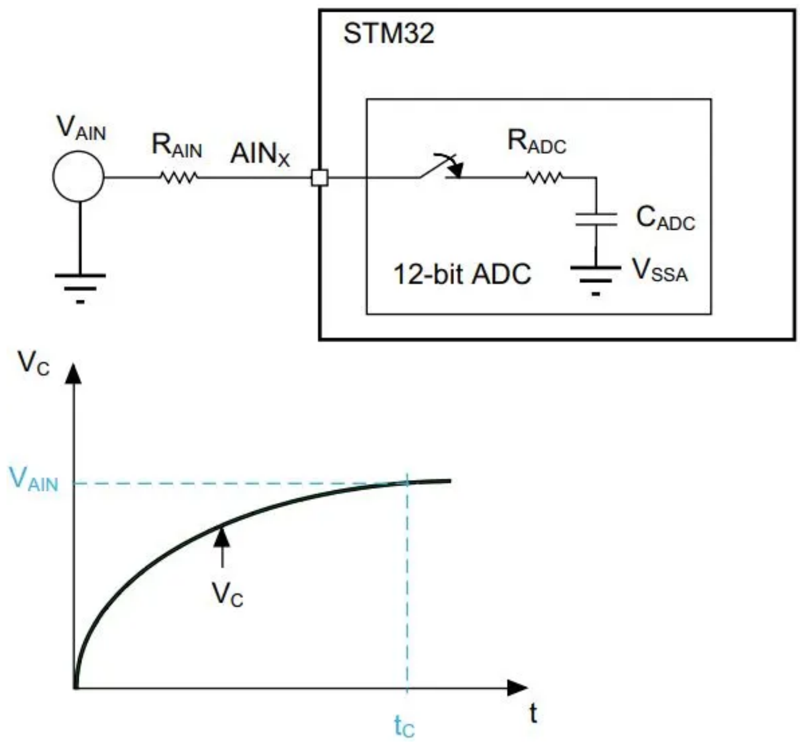
To obtain the maximum ADC conversion precision, it is very important that the ADC dynamic range matches the maximum amplitude of the signal to be converted. Let us assume that the signal to be converted varies between 0 V and 2.5 V and that VREF+ is equal to 3.3 V. The maximum signal value converted by the ADC is 3102 (2.5 V) as shown in the diagram down below. In this case, there are 993 unused transitions (4095 – 3102 = 993). This implies a loss in the converted signal accuracy.



2.4 – Analog Signal Source Impedance (Resistance)

The impedance of the analog signal source, or series resistance (R_{AIN}), between the source and pin, causes a voltage drop across it because of the current flowing into the pin. The charging of the internal sampling capacitor (C_{ADC}) is controlled by switches with a resistance R_{ADC}. With the addition of source resistance (with R_{ADC}), the time required to fully charge the hold capacitor increases.

If the sampling time is less than the time required to fully charge the C_{ADC} through R_{ADC} + R_{AIN} (ts < tc), the digital value converted by the ADC is less than the actual value.



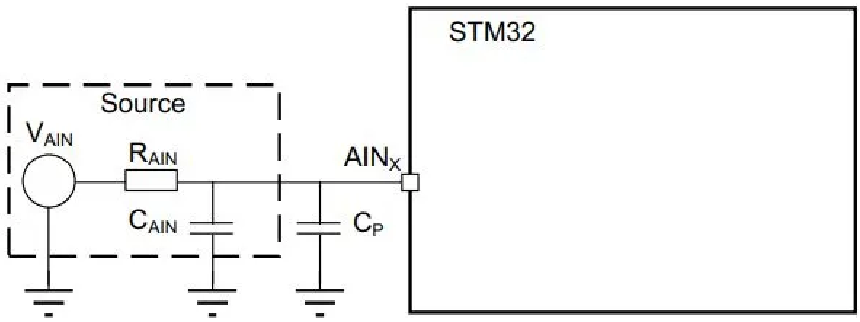
This error can be reduced or completely eliminated by setting the sampling time of the analog channel in such a way that guarantees an appropriate voltage level on the input pin is present before the ADC starts the conversion.

2.5 – Analog Signal Source Capacitance & Parasitics

When converting analog signals, it is necessary to account for the capacitance at the source and the parasitic capacitance seen on the analog input pin. The source resistance and capacitance form an RC network. In addition, the ADC conversion results may not be accurate unless the external capacitor (C_{AIN} + C_p) is fully charged to the level of the input voltage.

The greater value of (C_{AIN} + C_p), the more limited the source frequency. The external capacitance at the source and the parasitic capacitance are denoted by C_{AIN} and C_p, respectively.

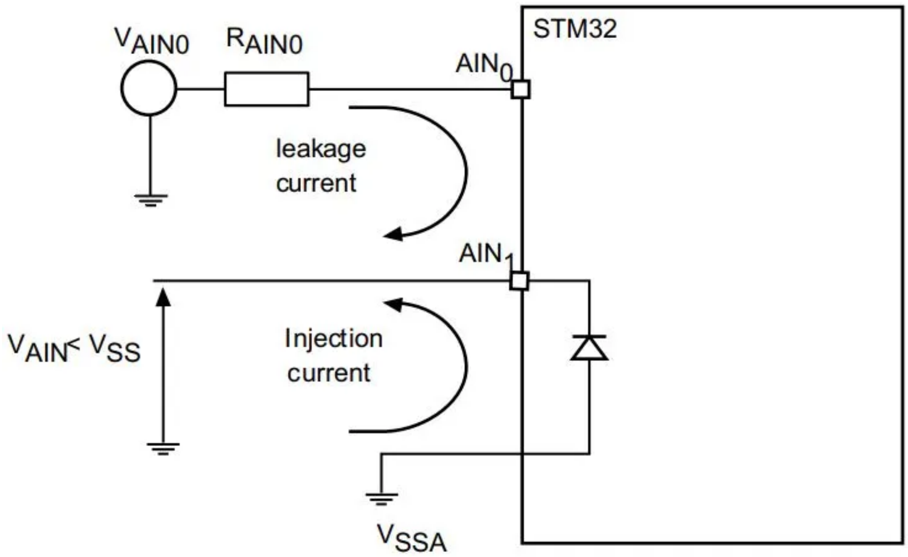
Analog input with R_{AIN}, C_{AIN} and C_p



2.6 – Injection Current Effect

A negative injection current on any analog pin (or a closely positioned digital input pin) may introduce leakage current into the ADC input. The worst case is the adjacent analog channel. A negative injection current is introduced when V_{AIN} < V_{SS}, causing current to flow out from the I/O pin. Which can potentially shift the voltage level on the pin and distort the measurement result.

Effect of injection current

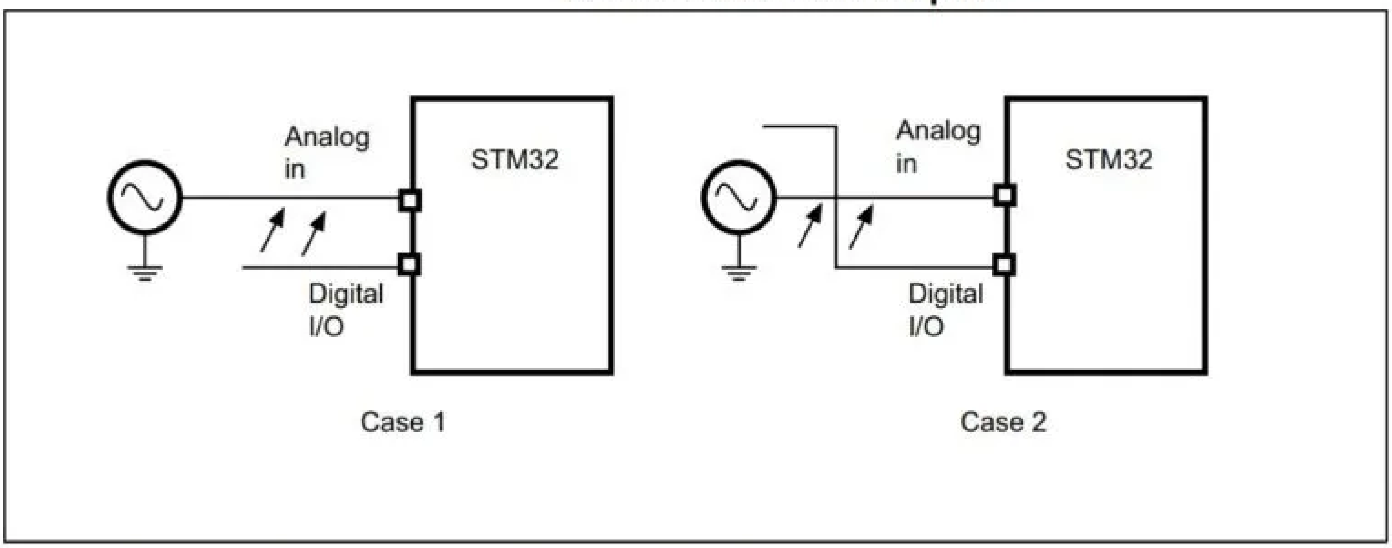


2.7 – IO Pins Cross-Talking

Switching the I/Os may induce some noise in the analog input of the ADC due to capacitive coupling between I/Os. Crosstalk may be introduced by PCB tracks that run close to each other or that cross each other.

Internally switching digital signals and IOs introduces high-frequency noise. Switching high sink I/Os may induce some voltage dips in the power supply caused by current surges. A digital track that crosses an analog input track on the PCB may affect the analog signal.

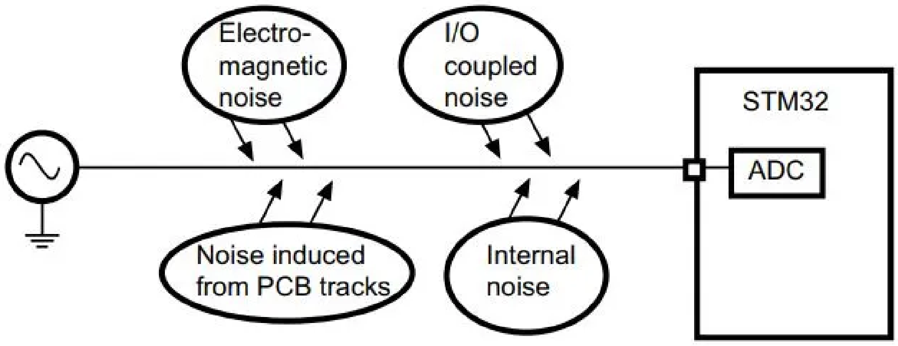
Crosstalk between I/O pins



1. Case 1: Digital and analog signal tracks that pass close to each other.
2. Case 2: Digital and analog signal tracks that cross each other on a different PCB side.

2.8 – EMI-Induced Noise

Electromagnetic emissions from neighboring circuits may introduce high-frequency noise in an analog signal because the PCB tracks may act as an antenna. It's called electromagnetic interference (EMI) noise.



ADC Example Applications

In the next few tutorials, we'll be practicing the ADC peripheral and do some practical LABs in order to learn how to configure and program the ADC to do certain tasks in different ways. The ADC example applications will include the following:

- Analog LED Dimmer With A Potentiometer Input
- Temperature Sensor Interfacing With LCD Display
- Servo Motor Control With Potentiometer
- LDR Light Sensor Detection
- Sound Recording & Playback using DAC, and power amp As Well

Did you find this helpful? If yes, please consider supporting this work and sharing these tutorials!

Stay tuned for the upcoming tutorials and don't forget to SHARE these tutorials. And consider SUPPORTING this work to keep publishing free content just like this!