Microprocessors

Fall 2020

11. Analog Interfacing

Tentative Weekly Schedule

- Week x1 - Introduction to Course
- Week x2 - Architecture
- Week x3 - Assembly Language Introduction
- Week x4 - Assembly Language Usage, Memory and Faults
- Week x5 - Embedded C and Toolchain
- Week x6 - Exceptions and Interrupts
- Week x7 - GPIO, External Interrupts and Timers
- Week x8 - Timers
- Week x9 - Serial Communications I
- Week xA - Serial Communications II
- **Week xB - Analog Interfacing**
- Week xC - DMA
- Week xD - RTOS
- Week xE - Wireless Communications

Introduction

Why do we need analog interfacing?

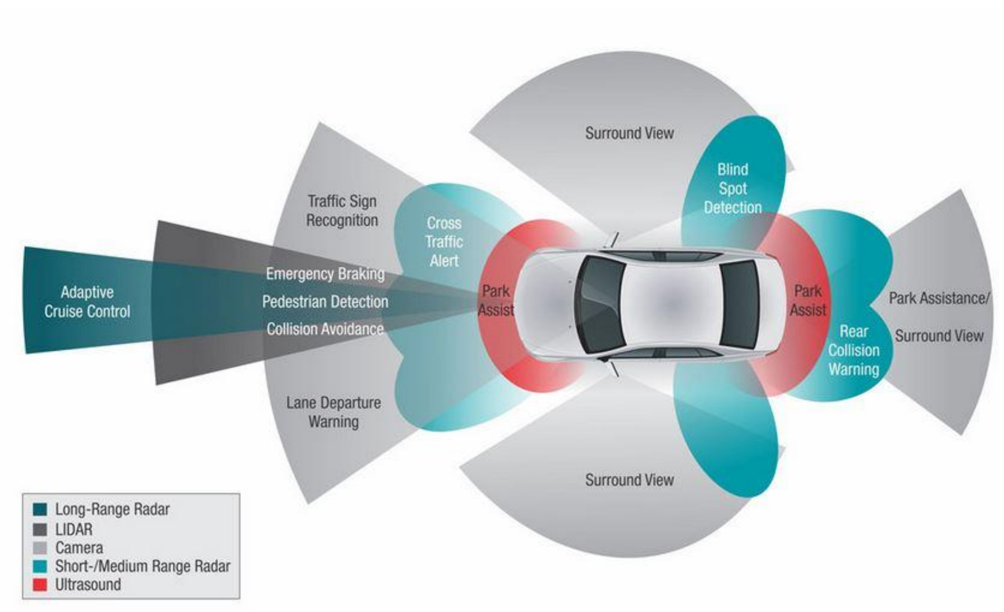
- Embedded systems often need to measure values of physical parameters

- These parameters are usually continuous (analog) and not in a digital form which computers (which operate on discrete data values) can process

We live in an analog world!

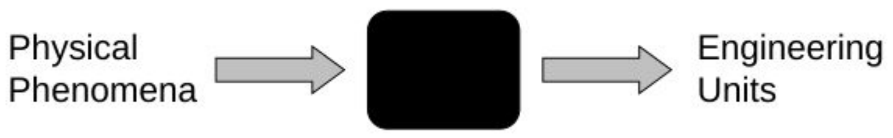
- **Temperature** - thermometer, thermostat, car engine, processor thermal management, reactors
- **Light intensity** - digital camera, IR remote control, optical communication
- **Pressure** - blood pressure monitor, altitude control, deep sea activity
- **Acceleration** - air bag controller, vehicle stability, video game remote, pendulum
- **Audio** - microphone, sonar, recording, noise cancelling
- **Position** - Rotary position, wind gauge, volume control, knobs
- **Monitoring** - battery level, motor control
- displacement, flow, radiation, ...

Example: Smart car sensors



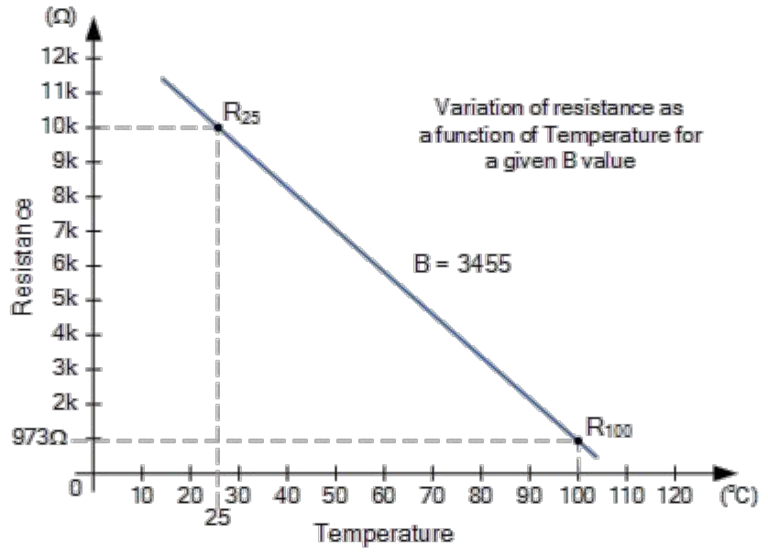
Going from analog to digital

- We need to somehow convert these physical phenomena into engineering units for processing

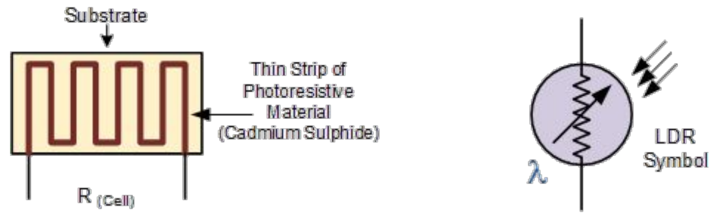


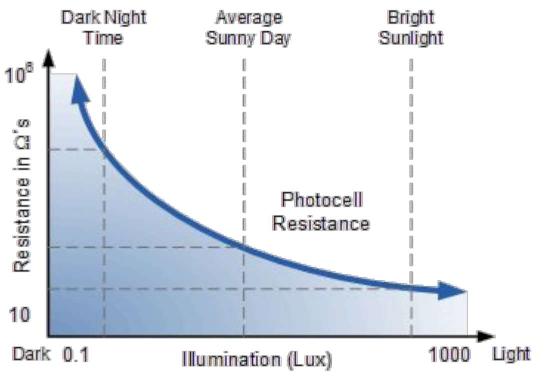
- The first part of the puzzle is to use some sort of **transducer** that will **translate one type of energy to another**.
- Usually to voltage or current.

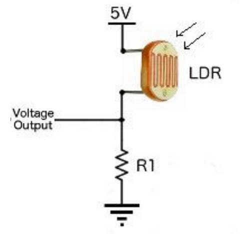
Example - Temperature Sensing



Example - Light Sensing



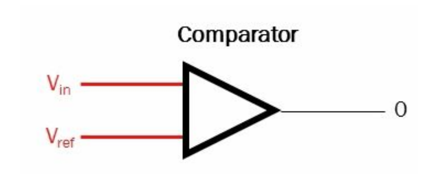




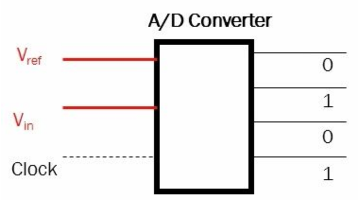
Going from analog to digital

For the second part of the puzzle we can use two approaches:

- A **comparator** tells us if **Vin** > **Vref**
  - Compares an analog input voltage with an analog reference voltage and determines which is larger, returning a 1-bit number



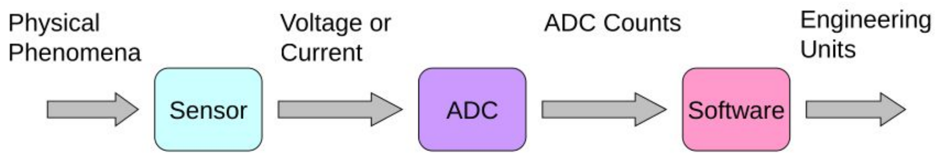
- An **Analog to Digital Converter (ADC, A/D)** tells us how large Vin is as a fraction of Vref.



Going from analog to digital

The last part is to use this **digital information** to produce the needed **engineering unit**.

```
#define ADCMASK (255)
#define VREF    (3)
adc_val = read_ADC();
adc_voltage = VREF * adc_val / ADCMASK;
temp = (adc_voltage - VTEMPOFFSET) / VTEMPGAIN;
```



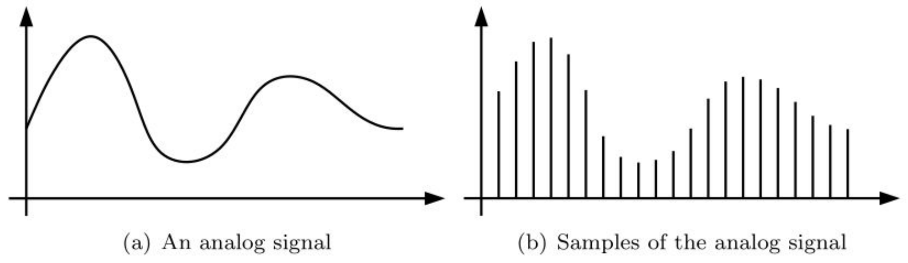
Analog to Digital Converter

- An Analog to Digital Converter **reads an analog input signal** (usually a voltage) and **produces a corresponding multi-bit number** at the output.
- Requires two reference voltages for minimum and maximum values. (Usually denoted as **AV$_{REF+}$** an **AV$_{REF-}$**)
  - Meaning, we can translate (or map) voltages in between these reference voltage values.
- Will round the voltage to the nearest possible digital value depending on the **resolution.** (can round down if not adjusted)

Representing an analog signal digitally

How do we represent an analog signal?

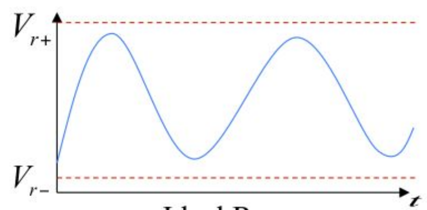
- As a time series of discrete values (**Sampling**)
- Map voltages to digital values with a linear correlation (**Quantization**)
- The range of digital values is the **resolution**.


(a) An analog signal (b) Samples of the analog signal

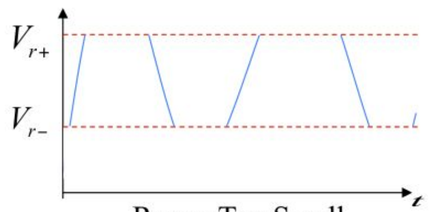
Choosing the horizontal range

We need to make sure we can represent the **whole voltage range effectively**.

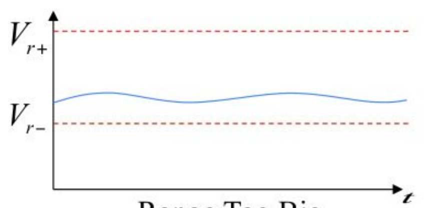
- What do the sample values represent?
- What range to use?



Ideal Range



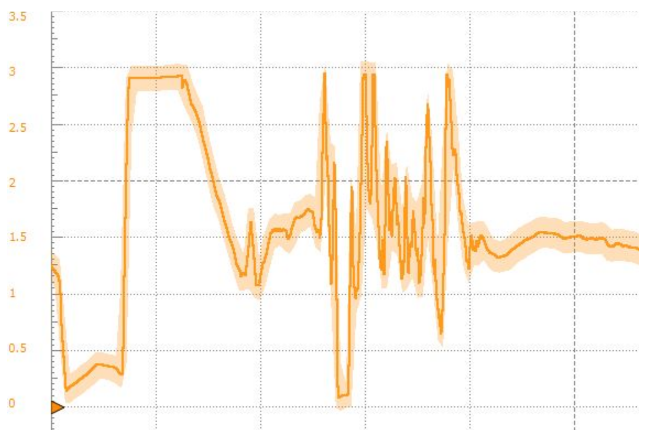
Range Too Small



Range Too Big

Example microphone signal

- It needs to be biased in the middle voltage level to get the full scale.

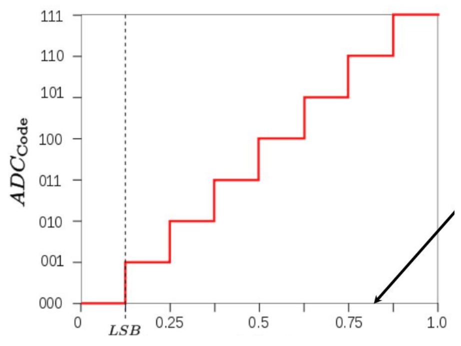


How to represent analog value digitally

- ADC Resolution is the **number of discrete values** that represent a range of analog values, denoted as **ADC bits** (8-bit, 12-bit, 16-bit, ...)

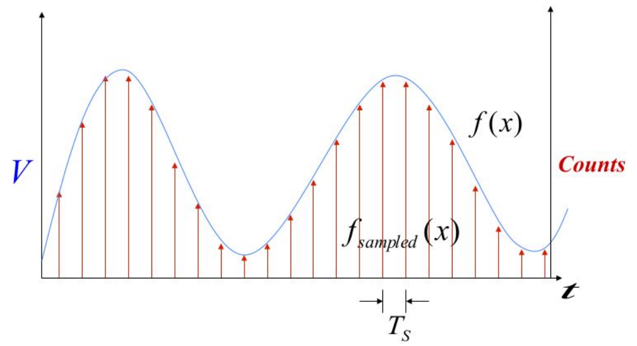
Example:

- For a 3-bit ADC, voltage range are represented in 2^3 = 8 different **levels**.
- **Step size** is voltage range/2^3 = 1/8 = 0.125 volts



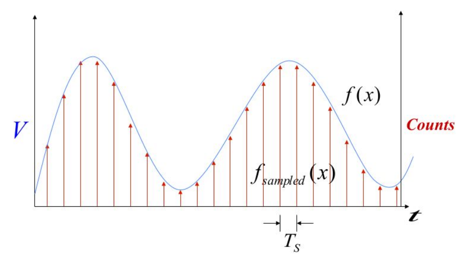
Sampling

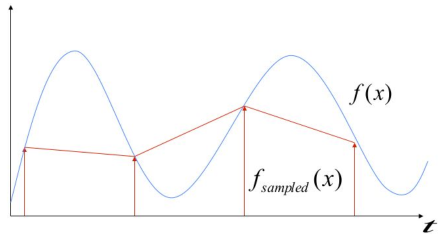
- In order to represent a continuous signal, we need to **sample** it **periodically**
- Each sample represents the **instantaneous amplitude** at the instant of sampling
- The interval between two samples is called the **sampling time**



Sampling Time (or rate)

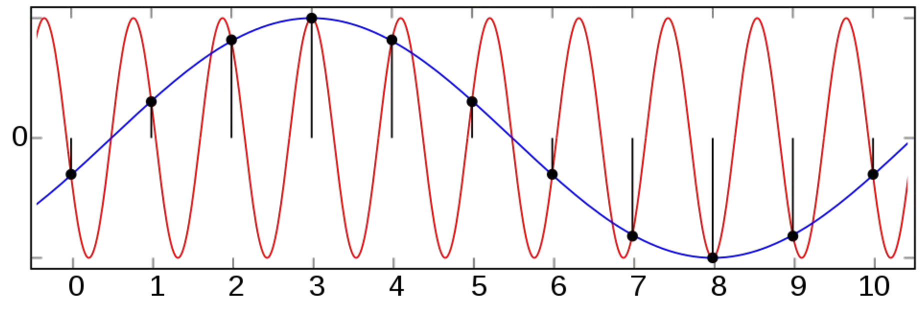
- If the sampling time is too large, we cannot reconstruct the signal
- If the sampling time is too small, we waste computation, energy and resources.





Sampling Problems

- If the sampling time is too large (or sampling rate is too low), we cannot reconstruct the signal
- Will mix up frequencies



Shannon-Nyquist sampling theorem

- If a continuous-time signal f(x) contains no frequencies higher than f_{max}, it can be completely determined by discrete samples taken at **double the desired max frequency**.

$$f_{\text{samples}} > 2 f_{\max}$$

- Frequency components above 1/2 $f_{samples}$ are **aliased**, distorting the measured signal.

Real world sampling

- Real world filters have more gentle roll-off
- Inexpensive filters are even worse.
- Thus, we usually need to go a little bit more higher

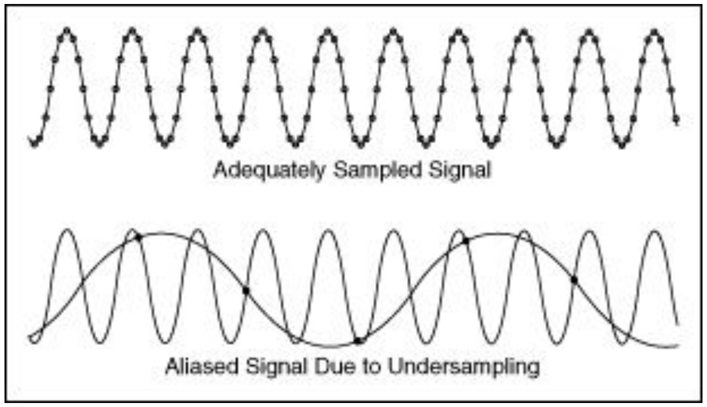
$$f_{\text{samples}} > 2 f_{\max}$$

Example:

- Humans can process audio signals between 20 Hz - 20 KHz
- Audio tracks sample at 44.1KHz

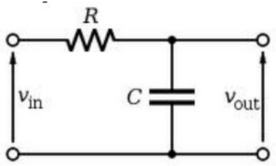
Aliasing

Aliasing causes a false lower frequency component to appear in the sampled data of a signal. The following figure shows an adequately sampled signal and an inadequately sampled signal.



Preventing aliasing

- Insert a Low-Pass Filter between the signal source and ADC (**anti-aliasing filter**)



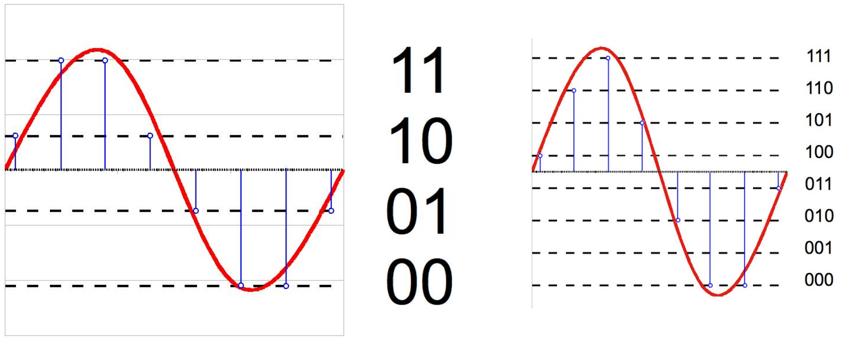
- Low-pass cutoff frequency is set to

$$f_{cutoff} = f_{Nyq} = \frac{1}{2 \times t_{samp}}$$

- This ensures no high frequencies are sampled by the ADC

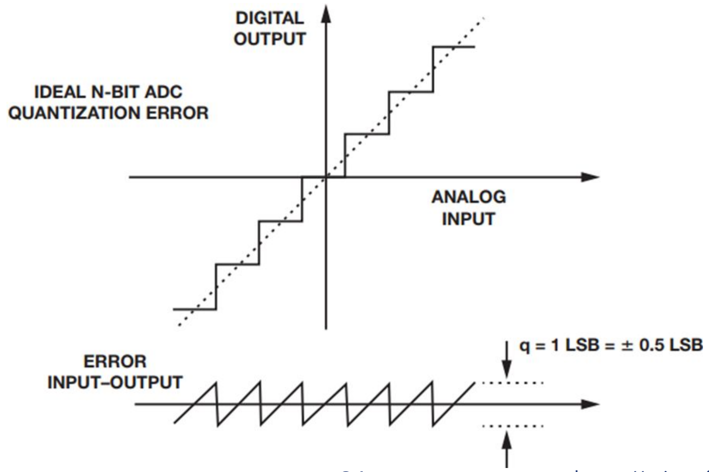
Quantization

- Quantization is the digital representation of the analog signal
- For the same signal with same sampling time but different resolution will result in different waveforms. - Higher resolution is better.

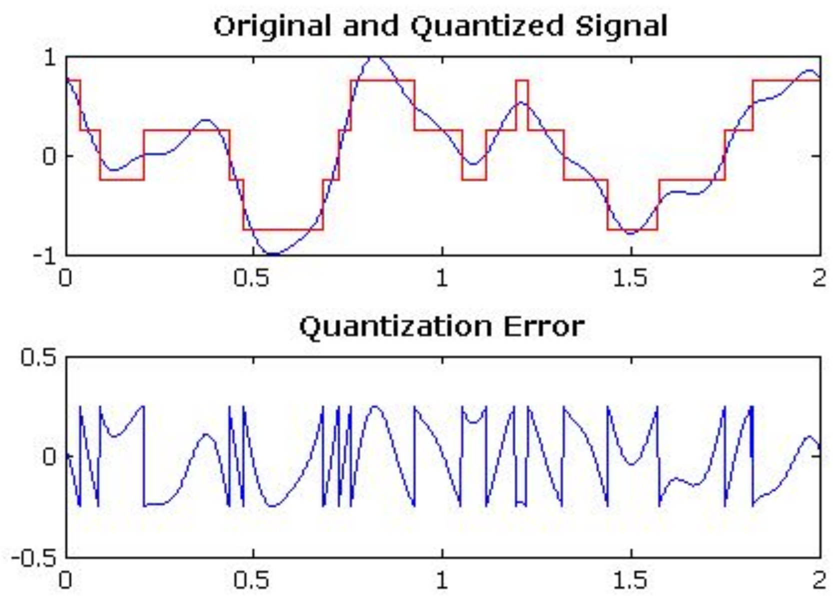


Quantization Error

- Quantization Error describes **how far off** discrete value is from the actual voltage
- +- 1/2 LSB for a total of 1 LSB



Example: Quantization and Error



Forward Transfer Function Equations

What code n will the ADC use to represent voltage V_{in}?

$$n = \left\lfloor \frac{V_{in} - V_{-ref}}{V_{+ref} - V_{-ref}} 2^N + 1/2 \right\rfloor$$

General Equation

n = converted code

V_{in} = sampled input voltage

V_{+ref} = upper voltage reference

V_{-ref} = lower voltage reference

N = number of bits of resolution in ADC

Forward Transfer Function Equations

We can simplify the equation if V-ref is 0 Volts.

$$n = \left\lfloor \frac{V_{in}}{V_{+ref}} 2^N + 1/2 \right\rfloor$$

Example:

$$n = \left\lfloor \frac{3.3V}{5V} 2^{10} + 1/2 \right\rfloor = 388$$

$$\lfloor X \rfloor = \text{floor}(X)$$

- **floor(X)** nearest integer I such that I <= X
- **floor(X + 0.5)** rounds x to the nearest integer

What if the Reference Voltage is not known?

- Example - running off of an unregulated battery
  - Measure a known voltage and an unknown voltage.

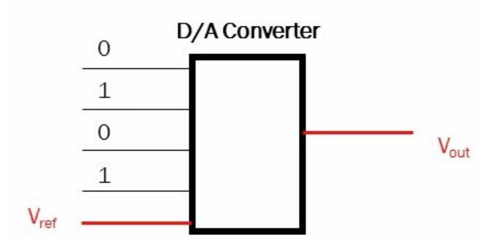
$$V_{unknown} = V_{known} \frac{n_{unknown}}{n_{known}}$$

- Many MCUs include an **internal fixed voltage source** which ADC can measure for this purpose
  - Can also solve for Vref

$$V_{ref} = V_{known} \frac{2^N}{n}$$

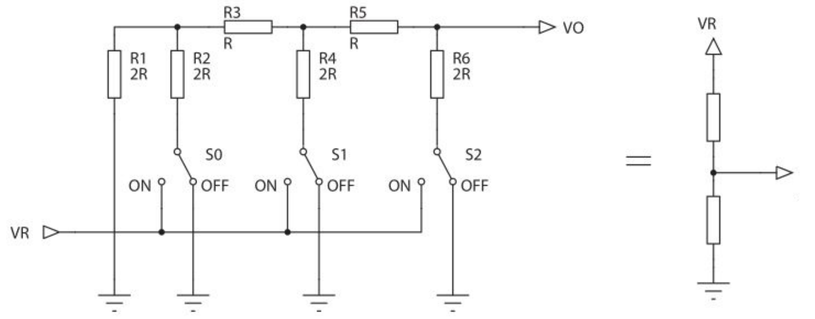
Digital to Analog Conversion

- While we want to read and sample analog values using sensors, sometimes we want to also **generate analog values** to **drive actuators.**
- **Digital to Analog Converter (DAC)** generates the analog voltage which is the fraction of Vref
- General equation is $\mathbf{V_{ref}\ n/(2^N)}$

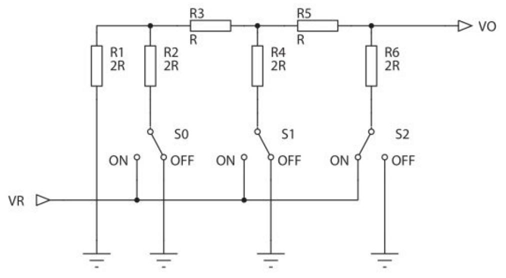


Example 3-bit DAC

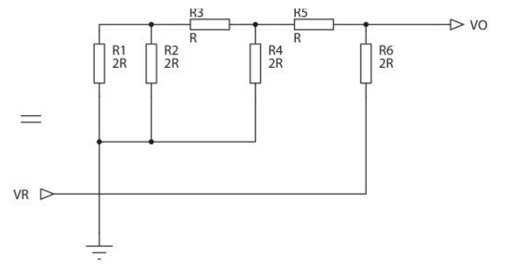
- Simple **R-2R** configuration
- By turning on each switch, you change the resistor ratios that affects the output voltage

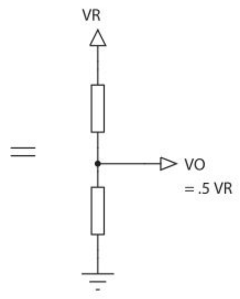


Example 3-bit DAC



S2	S1	S0	Vo
OFF	OFF	OFF	0
OFF	OFF	ON	$0.125 \times VR\ (1/8 \times VR)$
OFF	ON	OFF	$0.25 \times VR\ (2/8 \times VR)$
OFF	ON	ON	$0.375 \times VR\ (3/8 \times VR)$
ON	OFF	OFF	$0.5 \times VR\ (4/8 \times VR)$
ON	OFF	ON	$0.625 \times VR\ (5/8 \times VR)$
ON	ON	OFF	$0.75 \times VR\ (6/8 \times VR)$
ON	ON	ON	$0.875 \times VR\ (7/8 \times VR)$



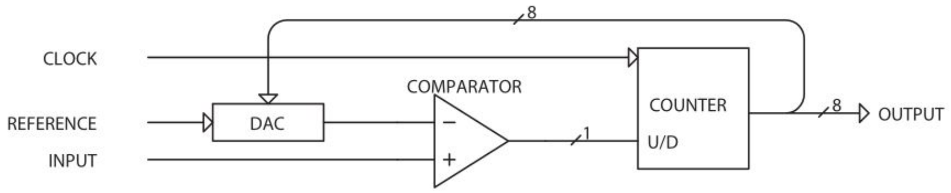


ADC Types

- There are several different ADC types each with their strengths and weaknesses.
  - Tracking ADC
  - Flash ADC
  - Successive Approximation ADC
  - Single/Dual-slope ADC
  - Sigma-delta (ΣΔ) ADC

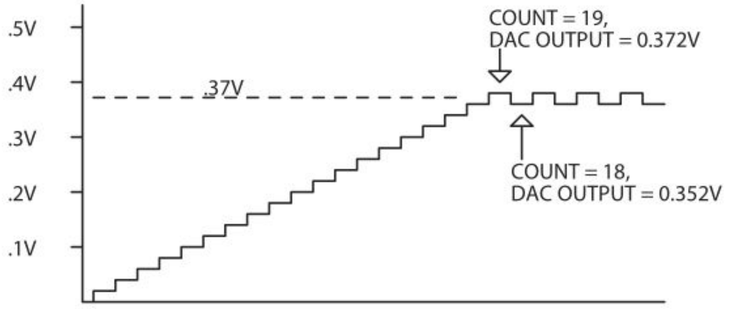
Tracking ADC

- The tracking ADC has a comparator, a counter and a digital-to-analog converter.
- The comparator compares the input voltage to the DAC output voltage.
  - If the input is higher than the DAC voltage, the counter counts up.
  - If the input is lower than the DAC voltage, the counter counts down.



Tracking ADC

- Conversion of 0.37V input using 0-5V ADC
- Counter starts at 0, making DAC output 0V
- Counter counts up, stepping DAC voltage up, until comparator output changes
- Counter then oscillates around input voltage value

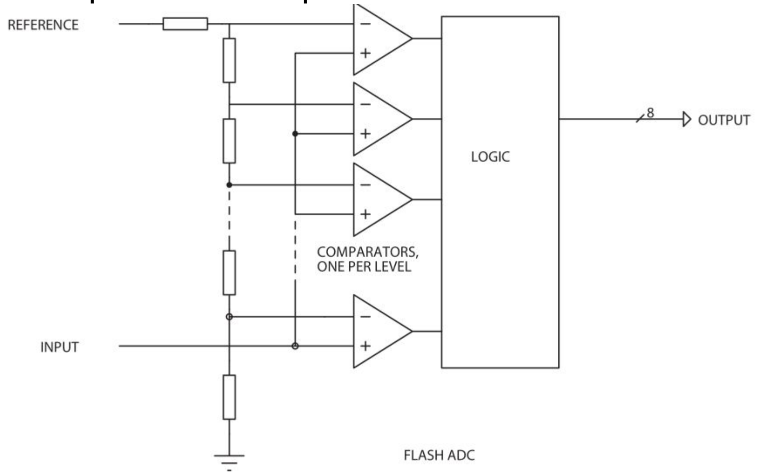


Flash ADC

- The flash ADC is the fastest type available.
- It has one comparator per voltage step.
  - A 4-bit ADC will have 16 comparators, and an 8-bit ADC will have 256 comparators.
- One input of all the comparators is connected to the input to be measured.
- The other input of each comparator is connected to one point in a string of resistors.
- As you move up the resistor string, each comparator trips at a higher voltage thus all comparator outputs determines the output based on which is on and which is off.

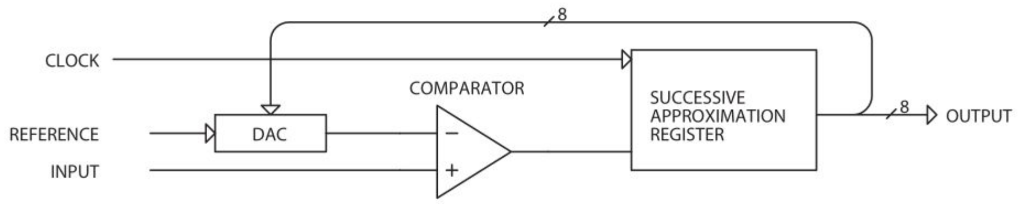
Flash ADC

- Flash ADCs are very fast (speed depends on the comparator and logic delays), but take large area and requires more power.



Successive Approximation Converter

- SAC is similar to the tracking ADC.
- The difference is that successive approximation register performs a **binary search** instead of just counting up or down by one.
- An 8-bit successive approximation ADC can do a conversion in 8 clocks regardless of the input voltage.

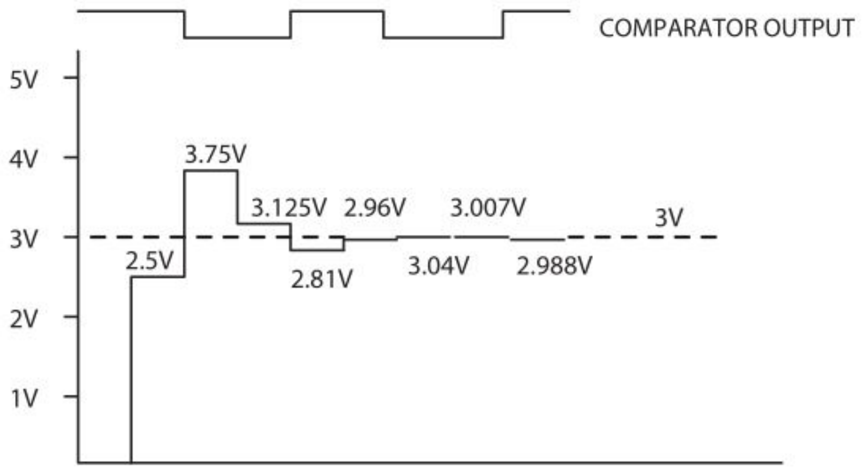


Successive Approximation Conversion

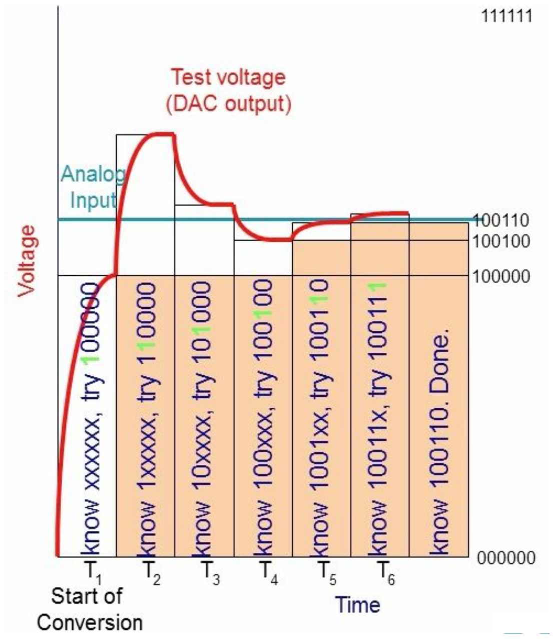
- To do a conversion first all the DAC bits are set to 0
- Start with DAC's MSB (half the voltage)
- Set next input bit for DAC to 1
- Wait for DAC and comparator to stabilize
- If the DAC output (test voltage) is smaller than the input then set the current bit to , else clear the current bit to 0

Successive Approximation Converter

- Conversion speed is consistent and usually faster.



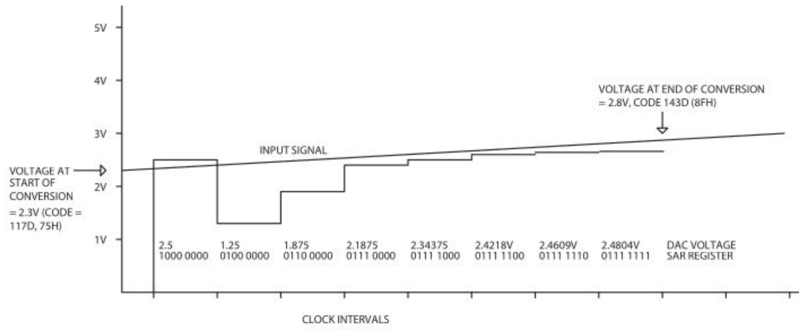
Successive Approximation Converter



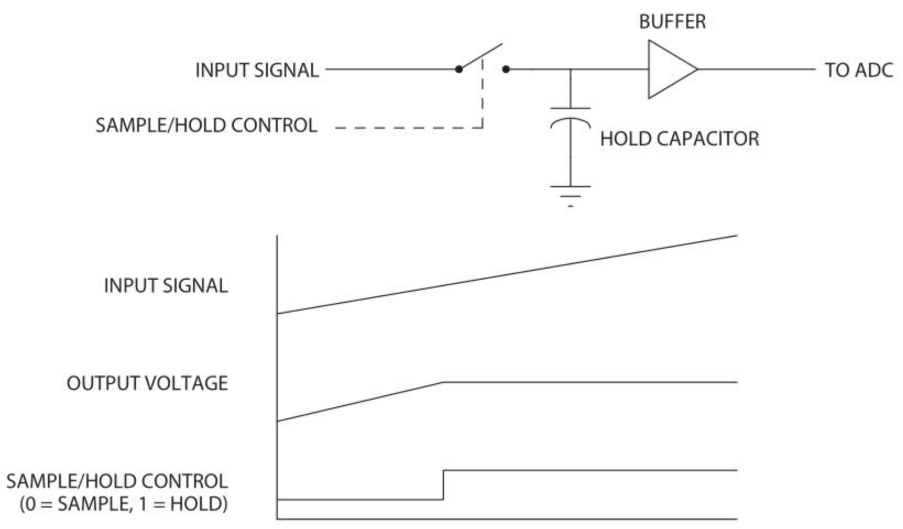
Sample and Hold

What happens if the signal is changing while we are sampling the ADC?

- Place a low-pass filter ahead of the ADC input to tolerate for conversion time
- Place a sample-and-hold (S/H) circuit

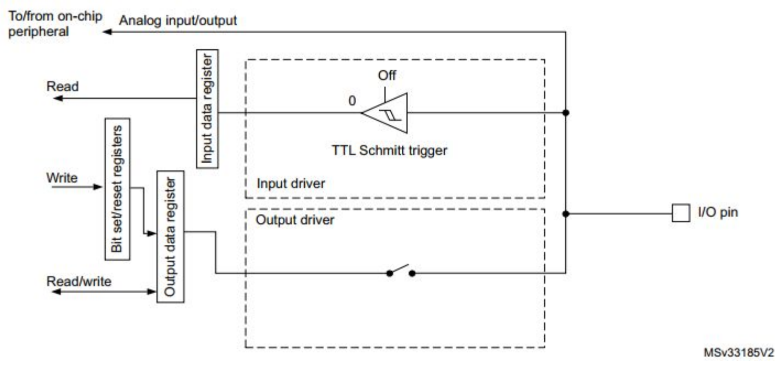


Sample and Hold Circuit



Analog Interfacing Peripherals

- GPIO pins may have different features which includes analog paths for ADCs and DACs.
- Not all the pins have analog functionality.
- Usually - port based.



STM32G0 - ADC General Features

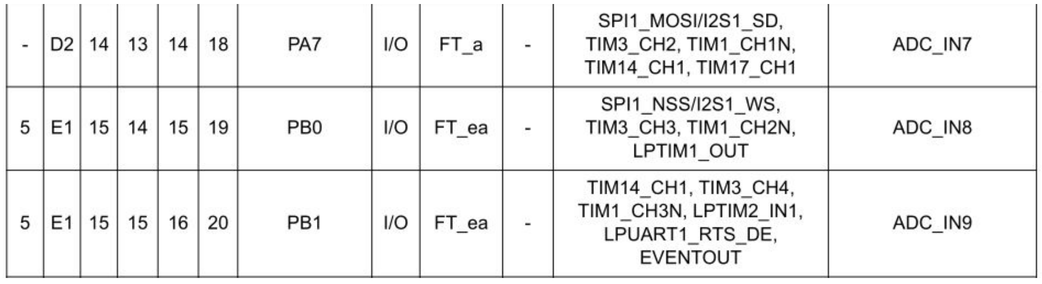
- STM32G0 has **12-bit successive approximation** ADC. Configurable to 12, 10, 8, 6-bits.
- ADC conversion time: 0.4 us for 12-bit resolution (**2.5 Msps**). Lower bits result higher conv. time.
- Up to 19 multiplexed channels allowing to measure signals from 16 external and 3 internal sources.
  - V_{SENSE} - internal temperature sensor
  - V_{REFINT} - internal reference voltage
  - V_{BAT} - monitoring external V_{BAT} power supply pin

STM32G0 - ADC General Features

- A/D conversion of the various channels can be performed in single, continuous, scan or discontinuous mode.
- Has an **analog watchdog** feature that allows the application to detect if the input voltage goes outside the user-defined higher or lower thresholds
- Start of conversion can be initiated using software, or **hardware triggers** (using GPIO inputs or **Timers**)
- ADC input range $V_{SSA} \leq V_{IN} \leq V_{REF+}$

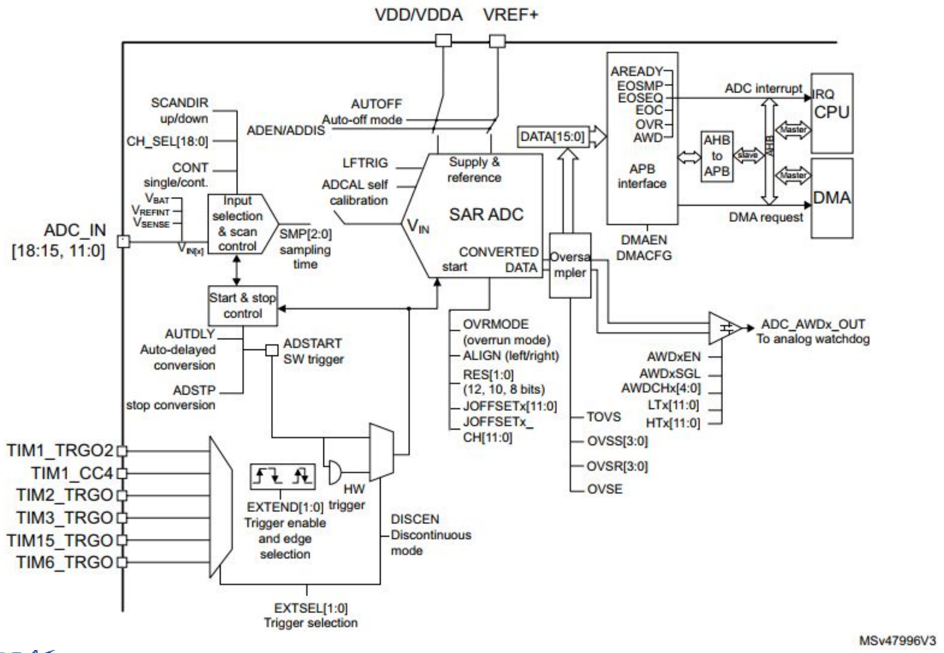
ADC pins in STM32G031K8

- Check the Pin assignment and description table from STM32G031 datasheet (Table 12)
- Pin number and ADC number can be different
- For example PB0 can be used as ADC_IN8



-	D2	14	13	14	18	PA7	I/O	FT_a	-	SPI1_MOSI/I2S1_SD, TIM3_CH2, TIM1_CH1N, TIM14_CH1, TIM17_CH1	ADC_IN7
5	E1	15	14	15	19	PB0	I/O	FT_ea	-	SPI1_NSS/I2S1_WS, TIM3_CH3, TIM1_CH2N, LPTIM1_OUT	ADC_IN8
5	E1	15	15	16	20	PB1	I/O	FT_ea	-	TIM14_CH1, TIM3_CH4, TIM1_CH3N, LPTIM2_IN1, LPUART1_RTS_DE, EVENTOUT	ADC_IN9

STM32G0 - ADC Block Diagram



STM32G0 - ADC External Triggers

- ADC can be externally using different sources.
- Configure the EXTSEL bits in the relevant register to select the source.
- For example, we can use TIM3 to trigger ADC sampling.

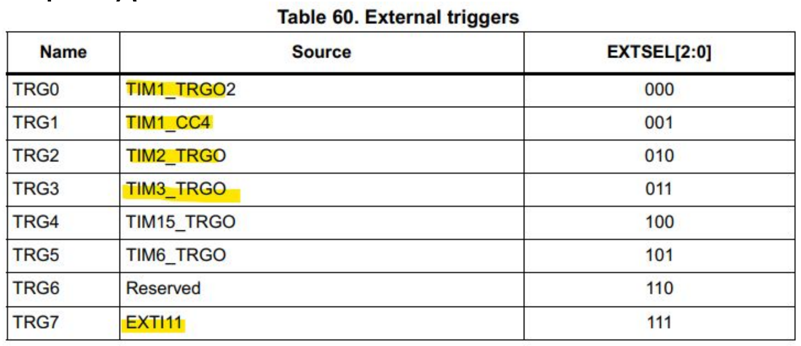


Table 60. External triggers

Name	Source	EXTSEL[2:0]
TRG0	TIM1_TRGO2	000
TRG1	TIM1_CC4	001
TRG2	TIM2_TRGO	010
TRG3	TIM3_TRGO	011
TRG4	TIM15_TRGO	100
TRG5	TIM6_TRGO	101
TRG6	Reserved	110
TRG7	EXTI11	111

STM32G0 - ADC Voltage Regulator

- ADC has a specific internal voltage regulator which must be enabled and stable before using the ADC.
- Can be enabled by setting ADVREGEN bit to 1 in CR register.
- The software need to wait for the ADC voltage regulator startup time (**20 us**)

STM32G0 - ADC Calibration

- ADC has a calibration feature that removes the offset error which may vary from chip to chip due to process variations.
- Calibration is initiated by software by setting bit ADCAL=1 in the relevant register.
- **Can only be initiated when ADC voltage regulator is enabled and ADC is disabled.**
- ADCAL stays 1 until calibration is complete, so wait for that amount. (or EOCAL=1 in status register)
- Calibration factor is written to the CALFACT reg.

STM32G0 - ADC Channel selection

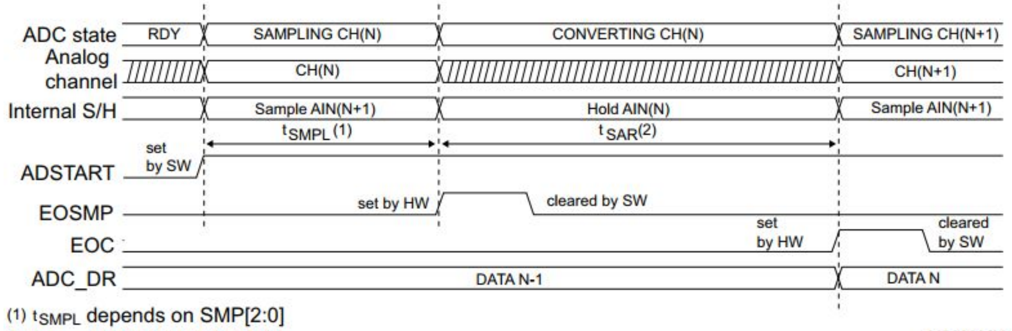
- It is possible to convert a single channel or a sequence of channels.
- The scan sequencer can be used in two different modes:
  - **Not fully configurable** - The order in which the channels are scanned can be defined by the channel number (forward or backward)
  - **Fully configurable** - The order in which the channels are scanned can be programmed using SQx bits up to 8 channels.

STM32G0 - ADC Timings

- The elapsed time between the start of a conversion and the end of conversion is the sum of the configured sampling time plus the successive approximation time depending on the data resolution

$t_{CONV} = t_{SMPL} + t_{SAR} = [1.5\ _{|min} + 12.5\ _{|12bit}] \times t_{ADC\_CLK}$

$t_{CONV} = t_{SMPL} + t_{SAR} = 42.9\text{ ns}\ _{|min} + 357.1\text{ ns}\ _{|12bit} = 0.400\ \mu s\ _{|min}$ (for $f_{ADC\_CLK} = 35$ MHz)



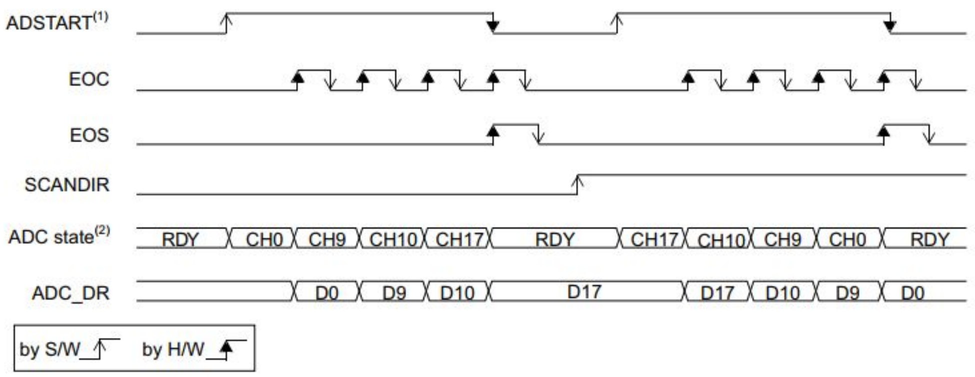
(1) t_{SMPL} depends on SMP[2:0]

(2) t_{SAR} depends on RES[2:0]

MS30336V1

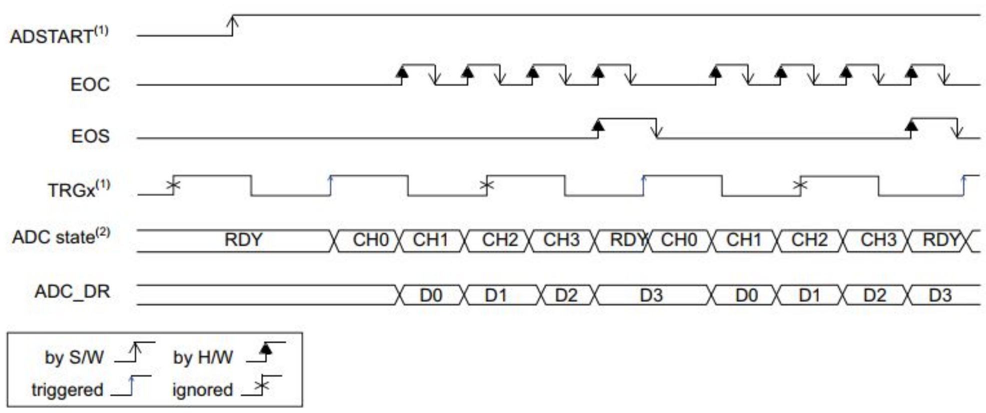
STM32G0 - ADC EOC EOS

- It is possible to generate interrupt for both end of conversion (single channel) and end of sequence (multi-channel selection) events
- Single conversions of a sequence, software trigger:



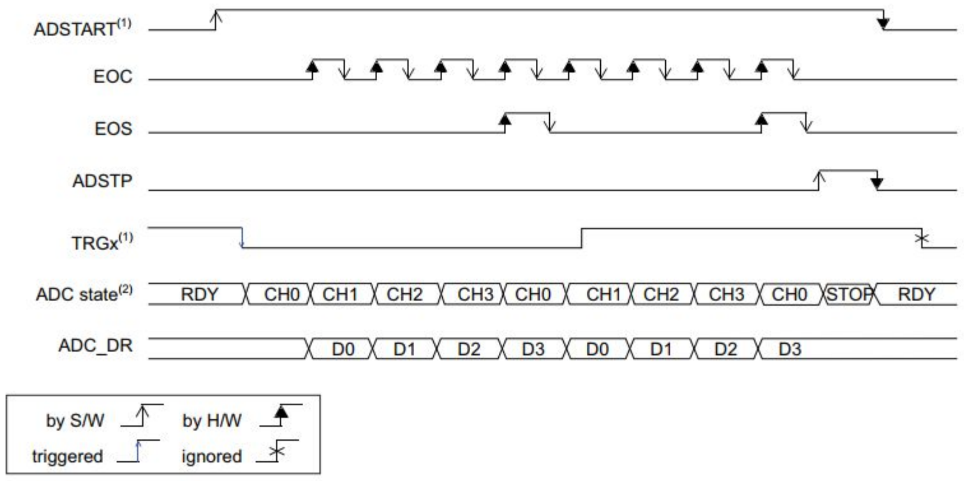
STM32G0 - ADC Hardware Trigger

- To set a sampling time, we can setup a timer and generate trigger events to start conversions
- In single conversions, each trigger start next **conversion**
- Single conversions of a sequence, hardware trigger:



STM32G0 - ADC Hardware Trigger

- In continuous conversions, each trigger start next **sequence**
- Continuous conversions of a sequence, hardware trigger:



STM32G0 - ADC Setup Continuous mode

1. Choose the relevant GPIO pin as Analog Input (Table 12 from stm32g031k8 datasheet)
2. Enable ADC Clock
3. Enable ADC Voltage Regulator, wait for at least 20us
4. Enable Calibration, wait until completion
5. Enable end of calibration or sequence interrupts
6. Configure number of bits to sample (6, 8, 10, 12)
7. Configure single/continuous mode
8. Select sampling time from SMPR register
9. Enable channels (for the ANx pins)
10. Enable ADC and wait until it is ready
11. Start conversion

STM32G0 - ADC Setup Trigger mode

1. Additionally, in ADC setup choose TRGO from EXTSEL and choose trigger edge from EXTEN.
   a. Note that ADC should be in single mode (not continuous)
2. Setup relevant Timer module normally. No need for interrupts.
3. Choose Update mode in the MMS register for the Timer
4. Don't forget the NVIC stuff for ADC interrupts
5. Each time timer overflows, it will generate a trigger out that will trigger the ADC which in turn will interrupt when the conversion ends. (EOC)