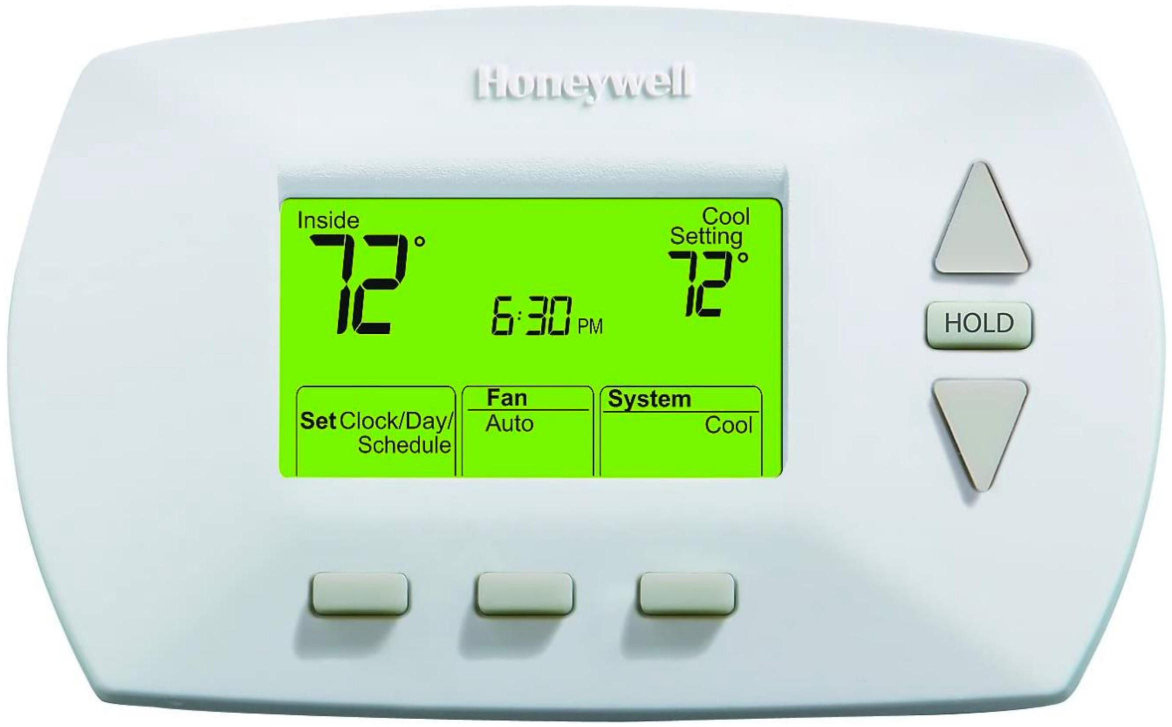


# PROGRAMMERING AV INBYGGDA SYSTEM

## Analog to digital and back

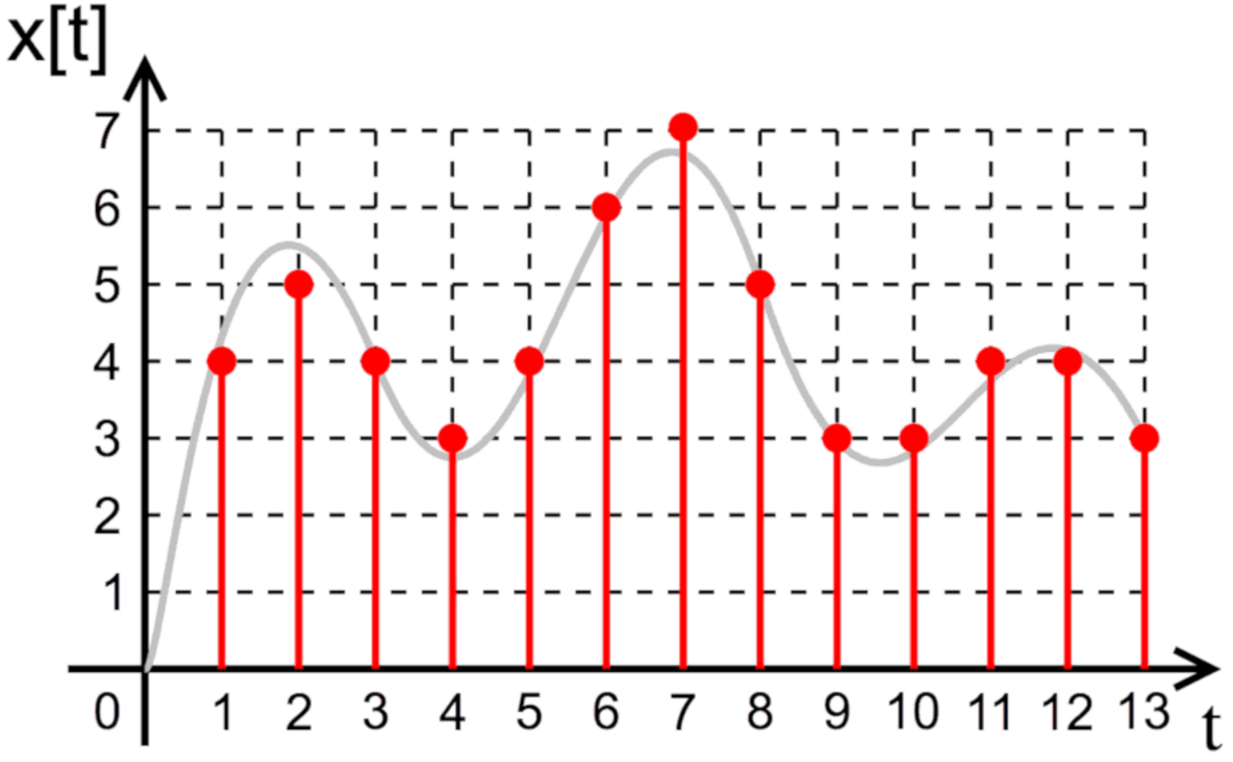
Dario Salvi, 2025

## Materials

- ESP32 Technical Reference Manual: SAR ADC, LED PWM and DAC
- Making embedded system: chapter 4 “Outputs, Inputs, and Timers” and chapter 9 “Maths”
- Online resources:
  - https://www.youtube.com/watch?v=UAJMLTzrM9Q (<<< excellent!)
  - https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem
  - https://docs.espressif.com/projects/esp-idf/en/stable/api-reference/peripherals/adc.html
  - https://learn.sparkfun.com/tutorials/pulse-width-modulation/all
  - https://www.analogictips.com/pulse-width-modulation-pwm/
  - https://docs.espressif.com/projects/esp-idf/en/stable/api-reference/peripherals/ledc.html
  - https://www.tutorialspoint.com/linear_integrated_circuits_applications/linear_integrated_circuits_applications_digital_to_analog_converters.htm
  - https://docs.espressif.com/projects/esp-idf/en/stable/api-reference/peripherals/dac.html

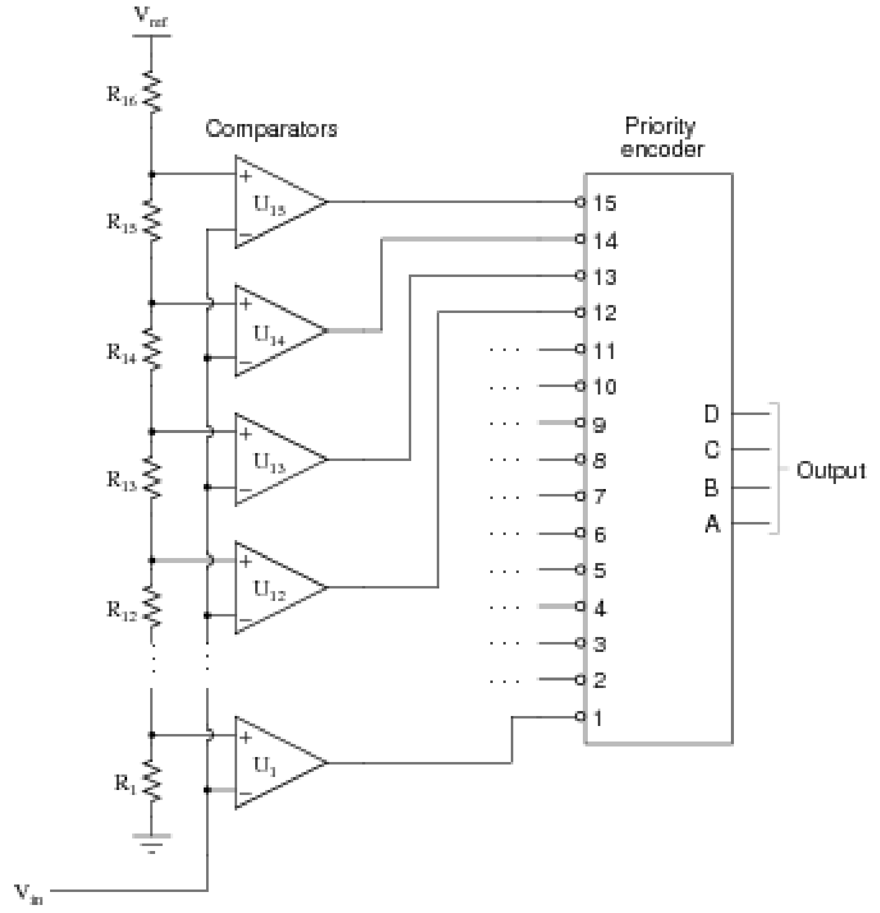
# Analogue to digital conversion

- In electronics, an **analogue-to-digital converter** (ADC, A/D, or A-to-D) is a system that converts an analogue signal, such as a sound picked up by a microphone, into a digital signal.
    - Note analog**ue** in British English, analog in American English.
- It involves 2 actions:
    - **Sampling**: taking one value every T time
    - **Quantising**: converting the value into a number of bits, which is limited by its nature



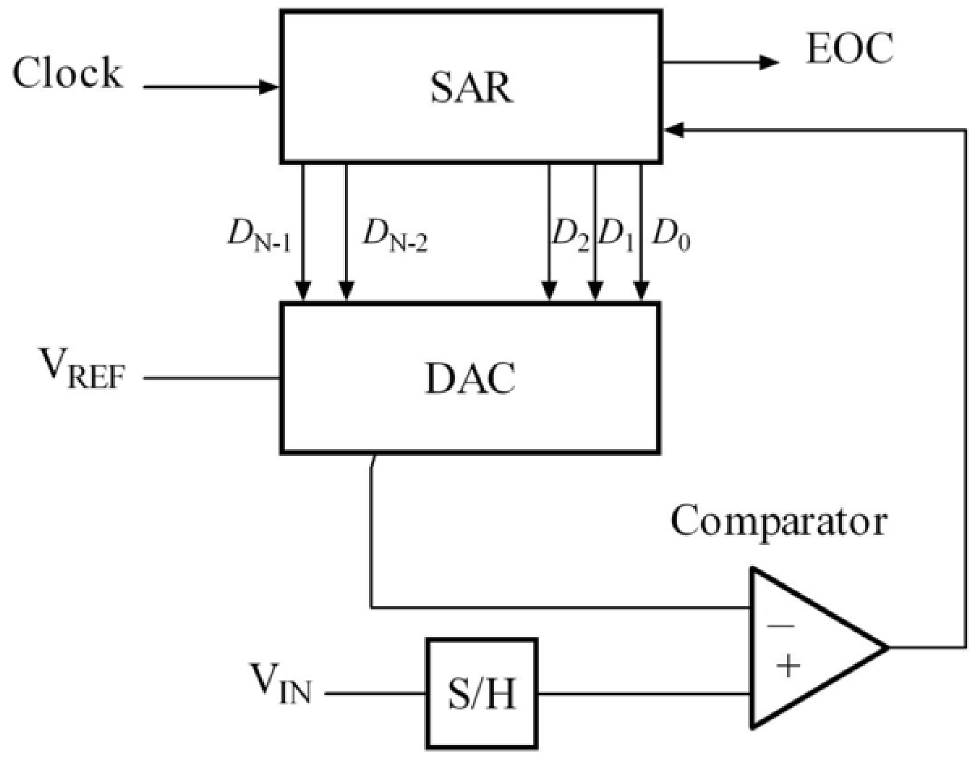
# Direct conversion

- A direct-conversion ADC has a **bank of comparators sampling the input signal in parallel**, each firing for their decoded voltage range.
- The comparator bank feeds a logic circuit that generates a code for each voltage range.
- Direct conversion is **very fast**, capable of gigahertz sampling rates, but usually has **low resolution**, since the number of comparators needed, $2^N - 1$, doubles with each additional bit, requiring a large, expensive circuit.

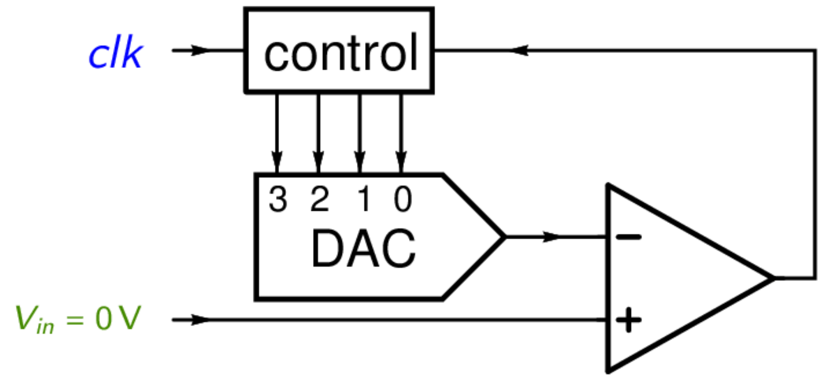


## Successive approximation

- A successive-approximation ADC uses a comparator to successively narrow a range that contains the input voltage.
- At each successive step, **the converter compares the input voltage to the output of an internal digital to analogue converter** which might represent the midpoint of a selected voltage range.
- At each step in this process, the approximation is stored in a successive approximation register (SAR).



## Successive Approximation – example of a 4-bit ADC

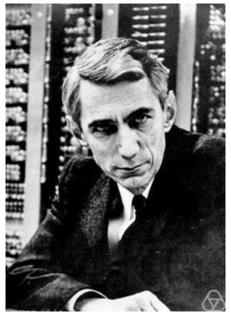


voltage
time
$V_{in} = 0\,\text{V}$

Resolution:

| | |
|---|---|
| $5\,\text{V} \times {}^{1}/_{2}$ | 2.5000 V |
| $5\,\text{V} \times {}^{1}/_{4}$ | 1.2500 V |
| $5\,\text{V} \times {}^{1}/_{8}$ | 0.6250 V |
| $5\,\text{V} \times {}^{1}/_{16}$ | 0.3125 V |
| ... | |
| $5\,\text{V} \times {}^{1}/_{1024}$ | 0.0049 V |

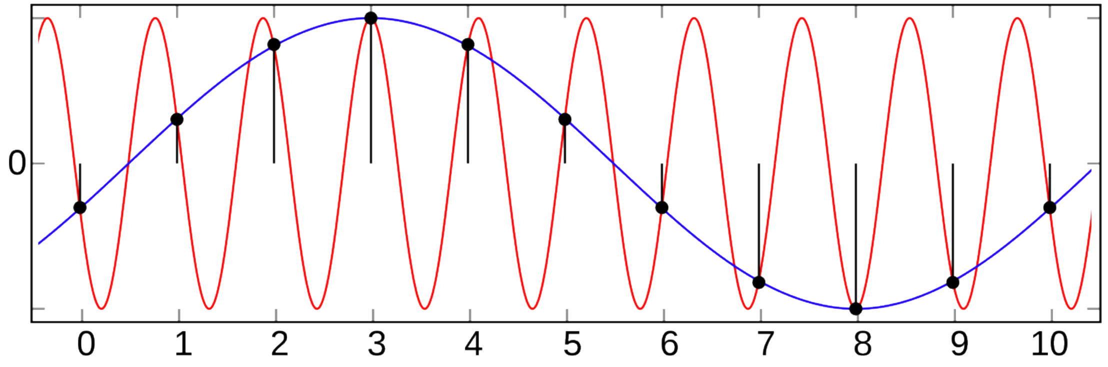
# Nyquist/Shannon theorem



- The Nyquist–Shannon sampling theorem serves as a fundamental bridge between continuous-time signals and discrete-time signals.
- It establishes that, in order to re-construct a sampled signal to is original form, **the sampling frequency $f_s$ must be at least 2 times the highest frequency B** (the *bandlimit*) of the original signal.
  - $f_s > 2B$ (where 2B is called the *Nyquist frequency*)
- When the bandlimit is too high (or there is no bandlimit), the reconstruction exhibits imperfections known as *aliasing*.

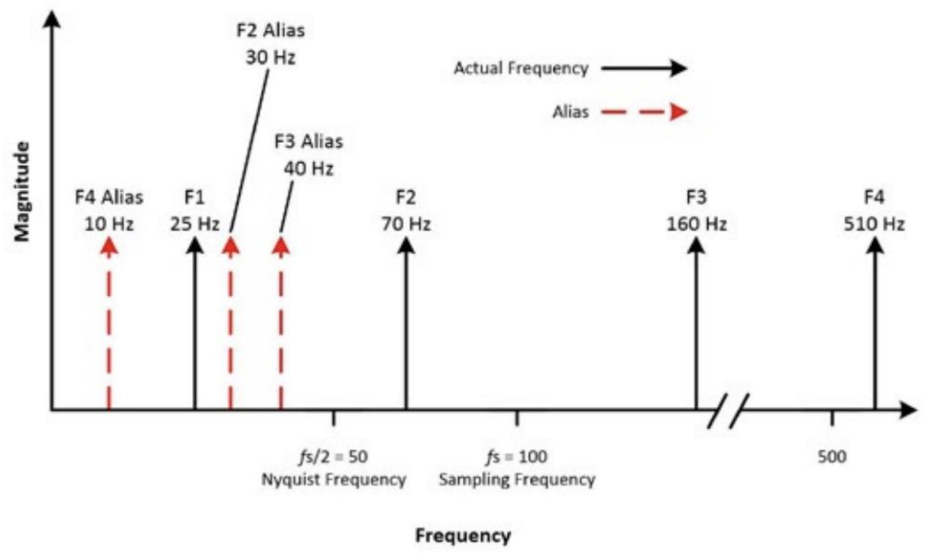
## Aliasing

- Aliasing is an effect that causes different signals to become indistinguishable (or aliases of one another) when sampled.
- It also often refers to the distortion or artefact that results when a signal reconstructed from samples is different from the original continuous signal.
- If the sampled signal has $B > 2f_s$ one can use a low-pass filter to reduce its bandwidth before sampling.



## Aliasing

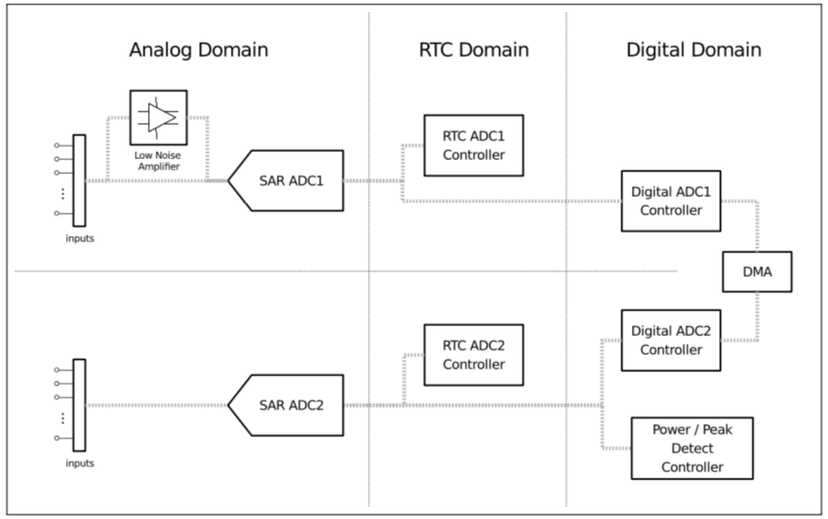
- If a sinusoid sin(2πf *t*) is sampled with a sampling frequency $f_s$, the sampled frequency presents aliases of the original one according to sin(2π( *f* +*N* $f_s$) *t*), *N* = 0, ±1, ±2, ±3,...



- The frequencies that get into the original signal are: $f_a$ = |(multiple of $f_s$ closest to f) - f |
- Example: a signal sampled at 100Hz has 4 sinuses at: 25Hz, 70Hz, 160Hz and 510Hz
- Aliases are:
  - |1 * 100 – 70| = 30 Hz
  - |2 * 100 – 160] = 40 Hz
  - |5 * 100 – 510| = 10 Hz

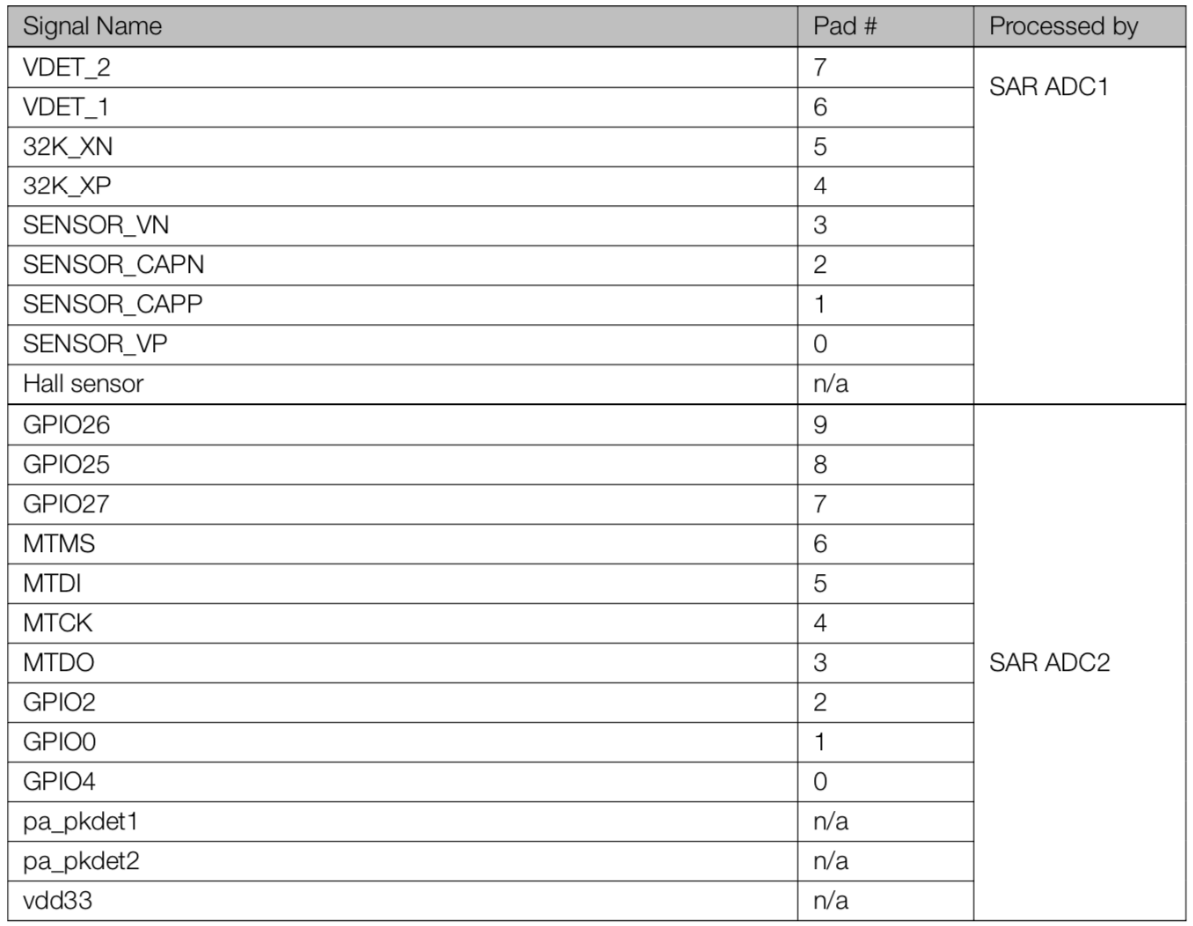
# ADC in ESP32

- ESP32 integrates **two 12-bit SAR ADCs**. They are managed by five controllers, and are able to measure signals from 18 pins. It is also possible to measure some internal signals (e.g. internal voltage).
  - Two controllers support high-performance multiple-channel scanning.
  - Two are used for low-power operation during deep sleep.
  - One is dedicated to power and peak detection.



# Routing to controllers

- Not all input signals come from external pins.



| Signal Name | Pad # | Processed by |
|---|---|---|
| VDET_2 | 7 | SAR ADC1 |
| VDET_1 | 6 | |
| 32K_XN | 5 | |
| 32K_XP | 4 | |
| SENSOR_VN | 3 | |
| SENSOR_CAPN | 2 | |
| SENSOR_CAPP | 1 | |
| SENSOR_VP | 0 | |
| Hall sensor | n/a | |
| GPIO26 | 9 | SAR ADC2 |
| GPIO25 | 8 | |
| GPIO27 | 7 | |
| MTMS | 6 | |
| MTDI | 5 | |
| MTCK | 4 | |
| MTDO | 3 | |
| GPIO2 | 2 | |
| GPIO0 | 1 | |
| GPIO4 | 0 | |
| pa_pkdet1 | n/a | |
| pa_pkdet2 | n/a | |
| vdd33 | n/a | |

# ADC in the ESP-IDF

- The ADC driver API supports
  - ADC1: 8 channels, attached to GPIOs 32 – 39
  - ADC2: 10 channels, attached to GPIOs 0, 2, 4, 12 - 15 and 25 – 27
- **ADC2 is used by the Wi-Fi driver**. Therefore the application can only use ADC2 when the Wi-Fi driver has not started.
- Two modes:
  - One shot: for **one conversion**
  - Continuous: for periodical sampling
- Initialization
  - Call *adc_oneshot_new_unit()* to allocate the resources
  - Call *adc_oneshot_config_channel()* to configure the ADC, including resolutions (n of bits) and attenuation (voltage range)
- Read
  - Call *adc_oneshot_read()*

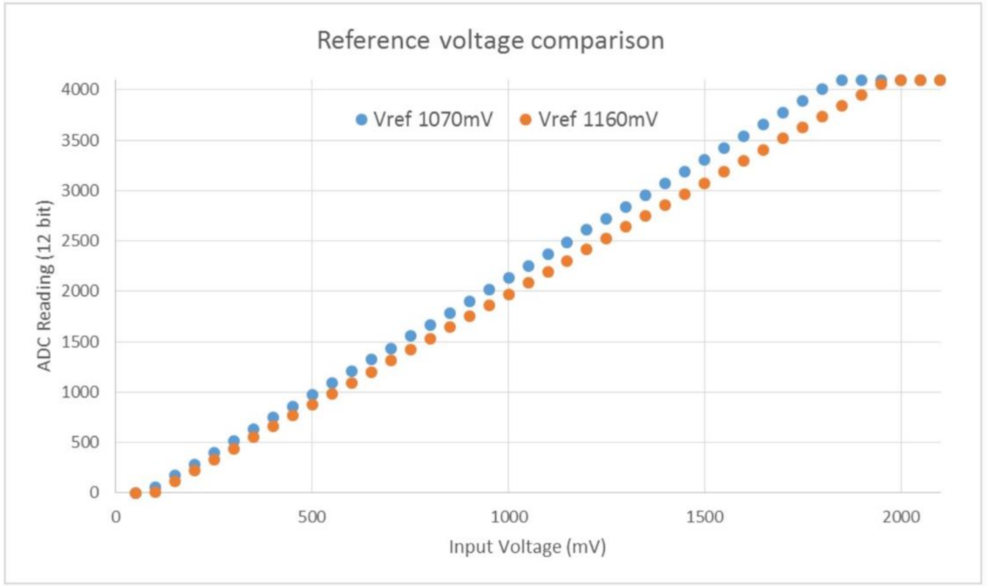
## Attenuation

- The default ADC full-scale voltage is 1.1V.
- To read higher voltages (up to the pin maximum voltage, usually 3.3V) requires setting >0dB signal attenuation for that ADC channel.
- When VDD_A is 3.3V:
  - 0dB attenuation (ADC_ATTEN_DB_0) gives full-scale voltage 1.1V
  - 2.5dB attenuation (ADC_ATTEN_DB_2_5) gives full-scale voltage 1.5V
  - 6dB attenuation (ADC_ATTEN_DB_6) gives full-scale voltage 2.2V
  - 11dB attenuation (ADC_ATTEN_DB_11) gives full-scale voltage 3.9V (or Vdd)
- The full-scale voltage is the voltage corresponding to a maximum reading (depending on configured bit width, this value is: 4095 for 12-bits, 2047 for 11-bits, 1023 for 10-bits, 511 for 9 bits.)

# Accuracy

- **Warning**: the ESP32 ADC is not very accurate, especially at the lower and higher voltages.
- Due to ADC characteristics, most accurate results are obtained within the following approximate voltage ranges:
  - 0dB attenuation (ADC_ATTEN_DB_0) between 100 and 950mV
  - 2.5dB attenuation (ADC_ATTEN_DB_2_5) between 100 and 1250mV
  - 6dB attenuation (ADC_ATTEN_DB_6) between 150 to 1750mV
  - 11dB attenuation (ADC_ATTEN_DB_11) between 150 to 2450mV

- A calibration API is available to compensate for inaccuracies.



*Graph illustrating effect of differing reference voltages on the ADC voltage curve.*

## Live coding

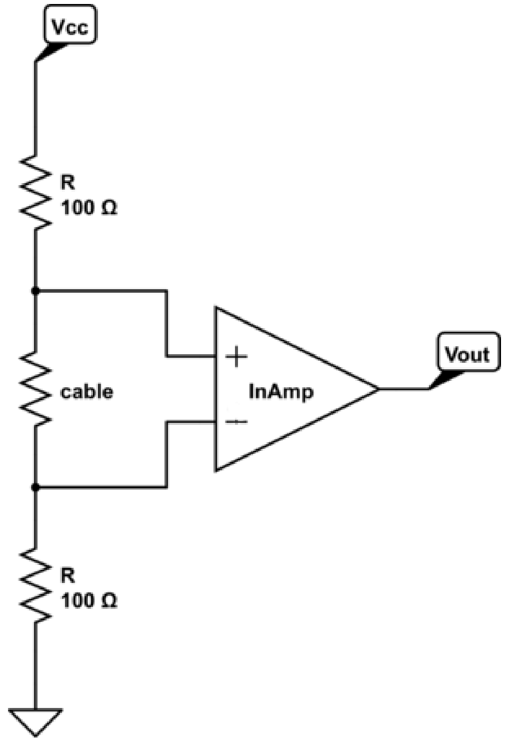
- Example of how to retrieve an ADC value.

## Converting measurements

- When using sensors data you want to use official units of measurements rather than “raw” data (= values in the ADC register).
- Example: measuring a resistance with an ADC.
  - The ADC gives a number (e.g. 0 to 4095).
  - But resistance is expressed in ohms !
  - The formula to convert the measured value to the resistance depends on the circuit, here it’s:
    R = 200 * Vout / (Vcc – Vout)

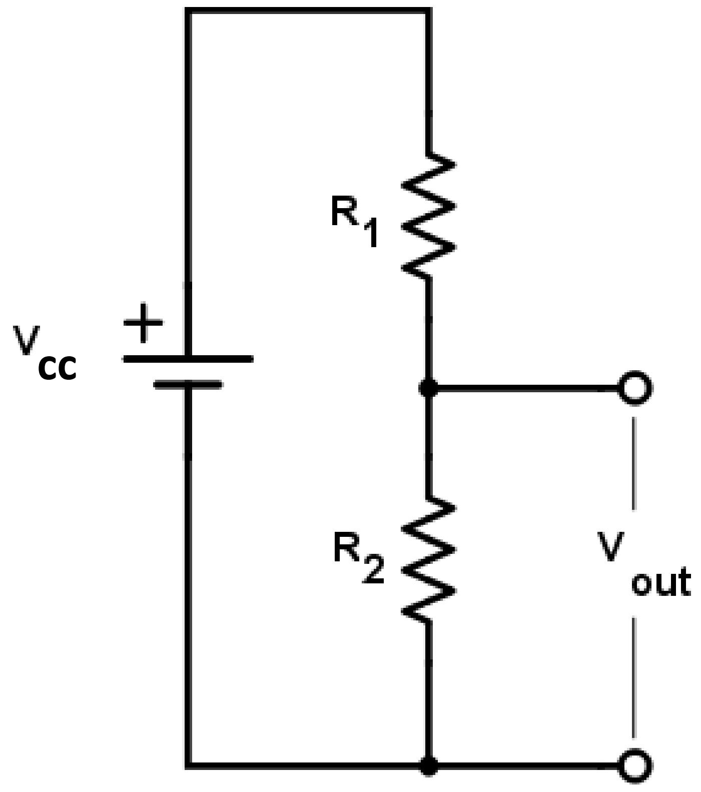
Raw values are difficult to understand and maintain

- They are highly hardware dependent
- Control algorithms become difficult to interpret



## How to convert from raw ADC to source unit

- ADC measures a voltage. Depending on the circuit you can infer a resistance, a voltage, or a current.
- Example: voltage divider.
  - $R_2$ is a sensor and its value changes depending on a physical quantity (e.g. a temperature).
  - $V_{cc}$ is the reference voltage (battery).
  - $V_{out}$ is what is measured by the ADC.
  - $V_{out} = V_{cc}\ R2 / (R1 + R2)$



# How to convert from raw ADC to source unit

- If A is the digital value, which can range from 0, when 0V are measured to $A_{max}$ when $V_{cc}$ is measured:
  - Assuming linearity: $V_{out} / A = V_{cc} / A_{max}$
    - **->** $A = \frac{Vout}{Vcc} A_{max} = \frac{R_2}{R_1+R_2} A_{max}$
- Therefore A and $R_2$ are related to each other by means of $R_1$ and $A_{max}$ only ($V_{cc}$ is not involved!):
  - **->** $R2 = \frac{A\,R_1}{A_{max}-A}$
- If $R_2$ is related to the actual physical quantity measured Q (for example a temperature) by $Q = f(R_2)$, the relationship between Q and A is:
  - $Q = f(\frac{A\,R_1}{A_{max}-A})$

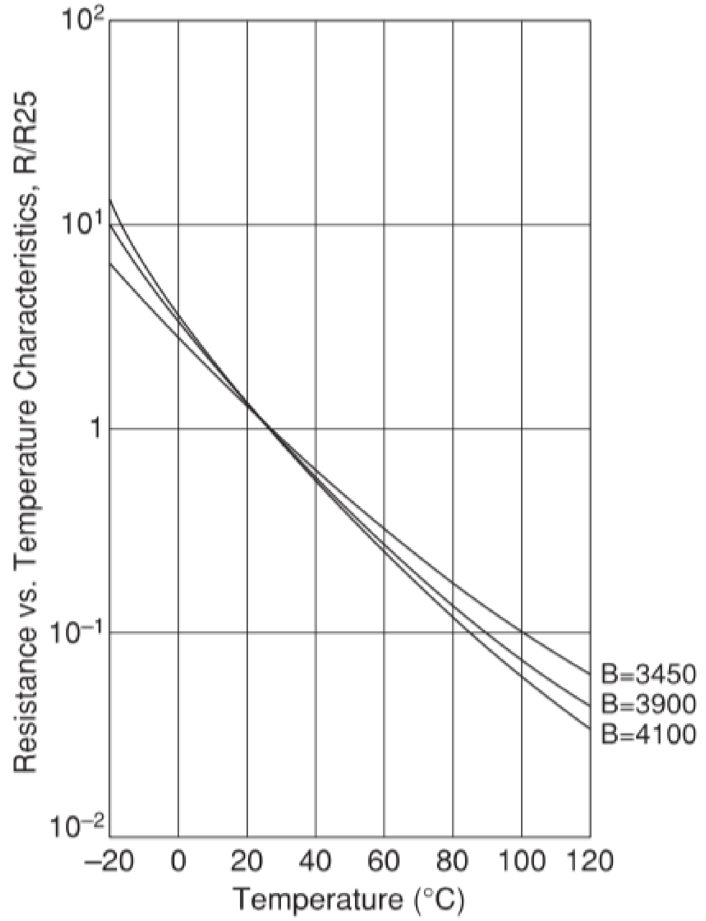
## Implementation

- Option 1: Use math to do the conversion.
  - Computationally expensive.
  - Math coprocessor not always available
- Option 2: Linearize the formula.
  - Less expensive.
  - Approximated.
- Option 3: use conversion tables.
  - Quick and easy.
  - Uses memory.

# Example: thermistor

- R= $R_0 \, e^{B(\frac{1}{T}-\frac{1}{T_0})}$
- Where:
  - R: Resistance in ambient temperature T (in Kelvin)
  - $R_0$: Resistance in ambient temperature $T_0$
  - B: Constant of Thermistor

Resistance vs. Temperature



https://www.murata.com/~/media/webrenewal/support/library/catalog/products/thermistor/ntc/r44e.ashx

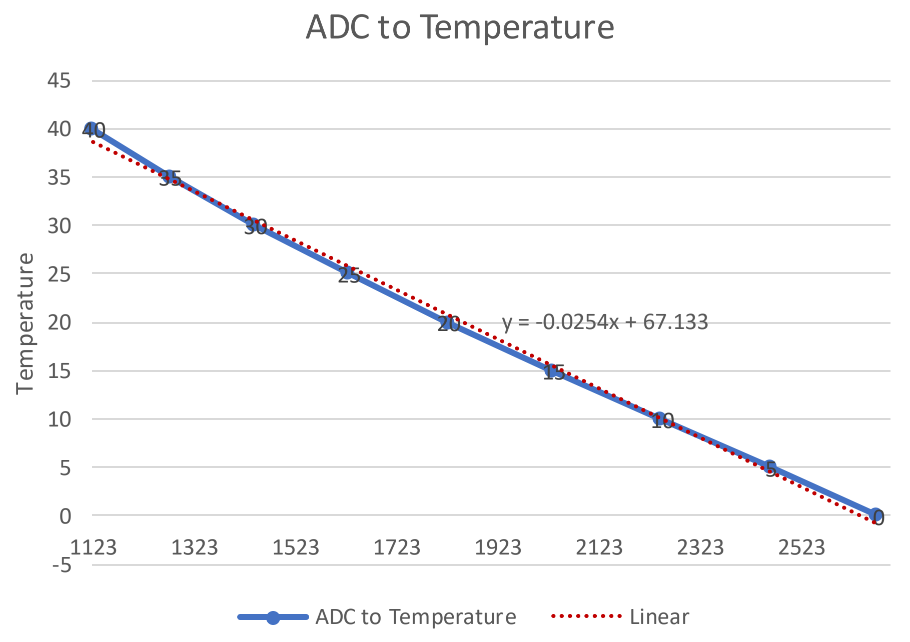
# Example: thermistor

- If V$_{cc}$ is 5V, R$_1$ is 500 Ohms and the ADC has 12bits resolution.
  - $R_0\ e^{B(\frac{1}{T}-\frac{1}{T_0})} = R_2 = \frac{A\ R_1}{A_{max}-A}$
  - $\frac{R_0(A_{max}-A)}{AR_1} = e^{-B(\frac{1}{T}-\frac{1}{T_0})}$
  - $\frac{1}{T_0} - \frac{1}{T} = \frac{log_e\left(\frac{R_0(A_{max}-A)}{AR_1}\right)}{B}$
  - $\frac{1}{T_0} - \frac{log_e\left(\frac{R_0(A_{max}-A)}{AR_1}\right)}{B} = \frac{B-T_0\ log_e\left(\frac{R_0(A_{max}-A)}{AR_1}\right)}{T_0 B} = \frac{1}{T}$
  - $T = \frac{T_0 B}{B-T_0\ log_e\left(\frac{R_0(A_{max}-A)}{AR_1}\right)}$

# Example: thermistor

We can build a table for some given values of interest (T= 0 to 40):

| T | Resistance | Voltage | ADC |
|---|---|---|---|
| 0 | 942 | 3,26629681 | 2675 |
| 5 | 754 | 3,00637959 | 2462 |
| 10 | 608 | 2,74368231 | 2247 |
| 15 | 493 | 2,48237664 | 2033 |
| 20 | 402 | 2,22838137 | 1825 |
| 25 | 330 | 1,98795181 | 1628 |
| 30 | 272 | 1,76165803 | 1443 |
| 35 | 226 | 1,55647383 | 1275 |
| 40 | 189 | 1,37155298 | 1123 |



# Linearization

- From the table we can derive a linear approximation of the relationship between the ADC value and temp:
- Temp = -0.0254 *ADC + 67.133

- What if our MCU does not support floating points?
  - Solution: use milliCelsius and multiply everything by 1000

```
int temp(uint32_t rawData) {
   return (- 25 * rawData + 67133);
}
```

# Linearization error

- Warning: approximations bring in an error!

| T | Resistence | Voltage | ADC | Approximated | Error |
|---|---|---|---|---|---|
| 0 | 942 | 3,26629681 | 2675 | 0 | 0 |
| 5 | 754 | 3,00637959 | 2462 | 6 | -1 |
| 10 | 608 | 2,74368231 | 2247 | 11 | -1 |
| 15 | 493 | 2,48237664 | 2033 | 16 | -1 |
| 20 | 402 | 2,22838137 | 1825 | 22 | -2 |
| 25 | 330 | 1,98795181 | 1628 | 26 | -1 |
| 30 | 272 | 1,76165803 | 1443 | 31 | -1 |
| 35 | 226 | 1,55647383 | 1275 | 35 | 0 |
| 40 | 189 | 1,37155298 | 1123 | 39 | 1 |

## Conversion table

- Alternative: a ”lookup table”
- The ADC value is 1123 at 40˚ and 2675 at 0 ˚
  - We need to map 1552 possible values

- Advantage: Quick to execute
- Disadvantage: Takes 1.5 Kb of memory

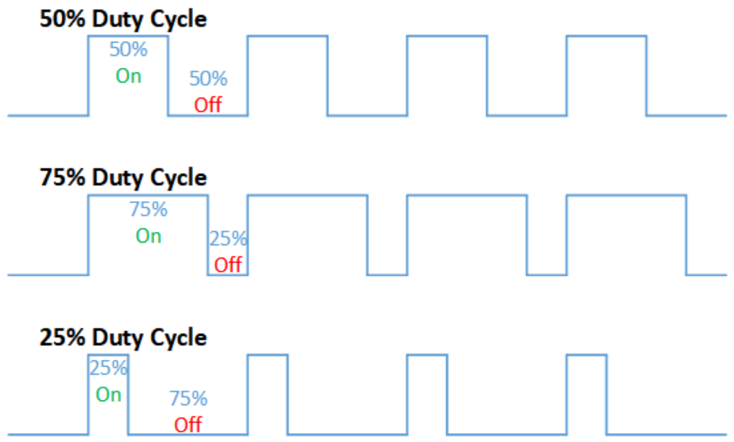
```
int8_t table[1552];

table[0] = 40;
… (here in between you need to interpolate)
table[152] = 35;
…
table[320] = 35;
…
…
table[1552] = 0;

int8_t temp(uint32_t rawData) {
  return table[rawData - 1123];
}
```

# Digital to analogue: PWM

- **Pulse-width modulation** (PWM), is a method of reducing the average power delivered to a load, by turning the switch between supply and load on and off at a fast rate. The longer the switch is on compared to the off periods, the higher the total power supplied to the load.
- **Duty cycle** describes the proportion of 'on' time to the period of time.
  - Duty cycle is expressed in percent, 100% being fully on and 0% being fully off.

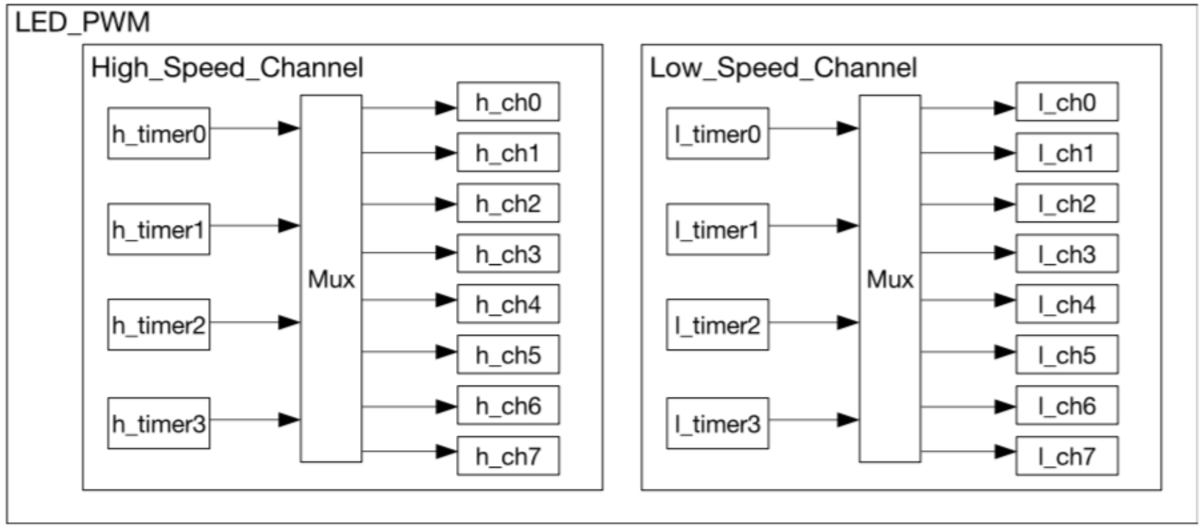


# PWM in ESP32

- The ESP32 has 2 types of dedicated hardware for producing PWM signals:
  - **LED PWM**: primarily designed to control the intensity of LEDs.
    - It has 16 channels which can generate independent waveforms that can be used to drive RGB LED devices.
    - High-speed and low-speed channels can be driven from one of four high-speed/low-speed timers.
    - The controller has the ability to automatically increase or decrease the duty cycle gradually, allowing for fades without any processor interference
  - **Motor Control PWM**: is intended for motor and power control.
    - It provides six PWM outputs that can be set up to operate in several topologies to control motor rotation speed and rotation direction.
    - The timing and control resources inside are allocated into two major types of submodules: PWM timers and PWM operators. Each PWM *timer* provides timing references that can either run freely or be synced to other timers or external sources. Each PWM *operator* has all necessary control resources to generate waveform pairs for one PWM channel.

# LED PWM

- The LED_PWM controller contains eight high-speed and eight low-speed channels.
- **High speed** mode is implemented in hardware and offers automatic and glitch-free changing of the PWM duty cycle.
  - There are four high-speed timers for the high-speed channels, from which one h_timerx can be selected.
- In **Low speed mode**, the moment of change depends on the application software.
  - There are four low-speed timers for the low-speed channels, from which one l_timerx can be selected.



## PWM in ESP-IDF

- Getting LEDPWM to work on a specific channel in either high or low speed mode is done in three steps:
  - **Configure Timer** by using ledc_timer_config() and specifying the PWM signal's frequency and duty cycle resolution.
  - **Configure Channel** by using ledc_channel_config() and associating it with the timer and GPIO to output the PWM signal. This will start the PWM, which can be stopped with ledc_stop().
  - **Change PWM Signal** that drives the output in order to change LED's intensity.
    - **By software**, use ledc_set_duty(). After that, call ledc_update_duty() to activate the changes.
    - Using **the hardware**: enable fading with ledc_fade_func_install() and then configure it by calling one of the available fading functions: ledc_set_fade_with_time(), ledc_set_fade_with_step(), ledc_set_fade(). Finally start fading with ledc_fade_start(). As an optional step, it is also possible to set up an interrupt on the fade end using ledc_isr_register().

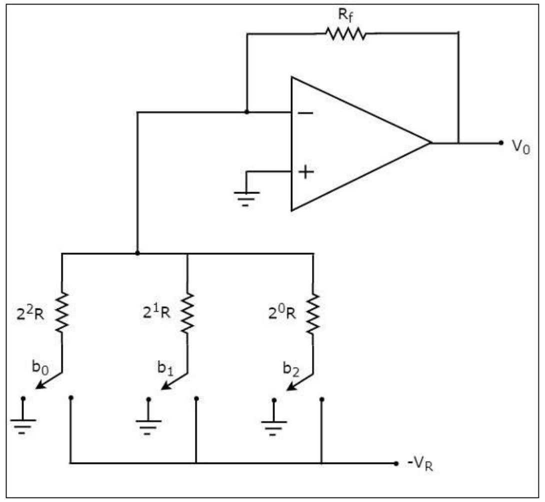
## Live coding

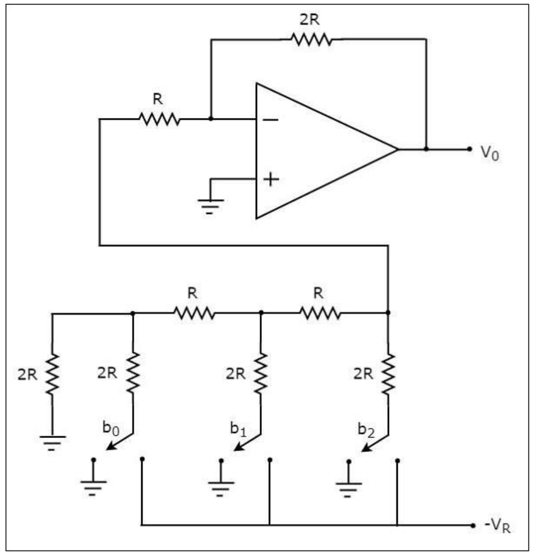
- Example of how to fade an LED using LED PWM.

THIS STATION IS NOW THE ULTIMATE POWER IN THE UNIVERSE
DOESN'T DO PULSE WIDTH MODULATION THOUGH DOES IT
SONICSTATE.COM
imgflip.com

## DAC

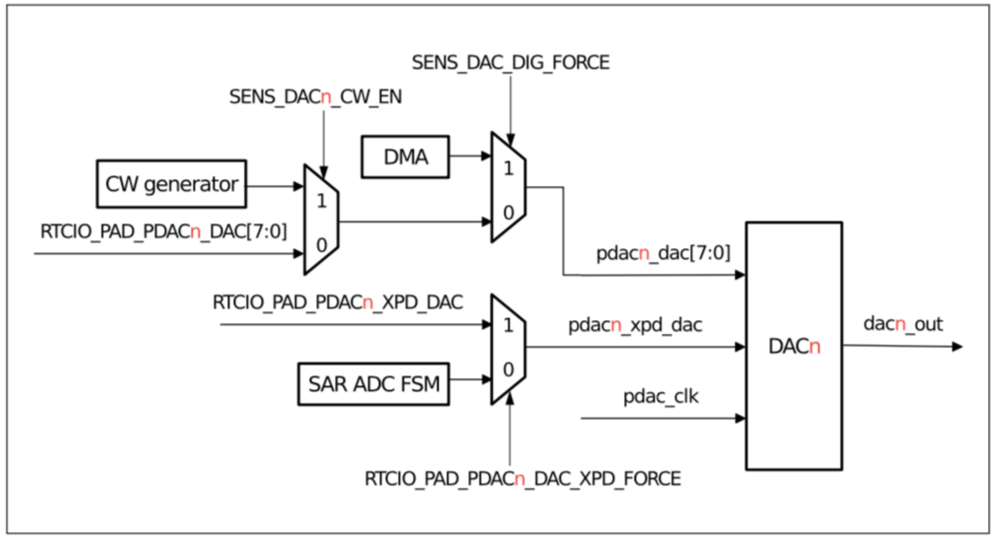
- The digital to analogue conversion takes a digital information (a number) and outputs a voltage that is proportional to it: $V_o = n * V_{ref}$
- It produces a “real” voltage, not an emulated one as in PWM.
- There are two types of DACs
  - **Weighted Resistor DAC**: uses binary weighted resistors in an inverting adder circuit.
  - **R-2R Ladder DAC**: uses an R-2R ladder network in the inverting adder circuit.





# DAC in ESP32

- Includes **two 8-bit DAC channels**. Channel 1 is GPIO25 and channel 2 is GPIO 26.
- Can be used as voltage reference (to measure battery).
- Includes a cosine waveform (CW) generator.



## DAC in ESP-IDF

- Use *dac_output_enable()* to enable DAC and *dac_output_voltage()* to set the output voltage (0 to 255 corresponds to 0V to 3.3V).
- Use *dac_cw_generator_enable()* and *dac_cw_generator_config()* to control the cosine wave generator

## Live coding

- Example of how to fade an LED using LED PWM.

## Example exam questions

- What is the difference between direct and successive approximation ADC conversion? What are the advantages and disadvantages of each technique?
- Suppose that you have connected your 10-bit ADC to a distance sensor that produces 2V when the distance from an object is 0m and 0V when the distance is 50m (with anything in between linearly scaled between those values). Write the formula that converts the ADC value into distance (in meters).
- When converting a raw ADC value to a physical quantity, what are the advantages of using a lookup table? And the disadvantages?
- What is aliasing and how can it be avoided?
- I am sampling a 50Hz sinusoid signal with a sampling frequency of 60Hz, what frequency should I observe in my sampled signal?
- Define the duty cycle in a PWM modulation.
- What are the advantages of using a DAC vs PWM?