Analog-to-digital converters and test using 10K pot meter

What is an ADC?

Definition

An analog-to-digital converter (also known as an ADC or an A/D converter) is an electronic circuit that measures a real-world signal (such as temperature, pressure, acceleration, and speed) and converts it to a digital representation of the signal.

How does an ADC work?

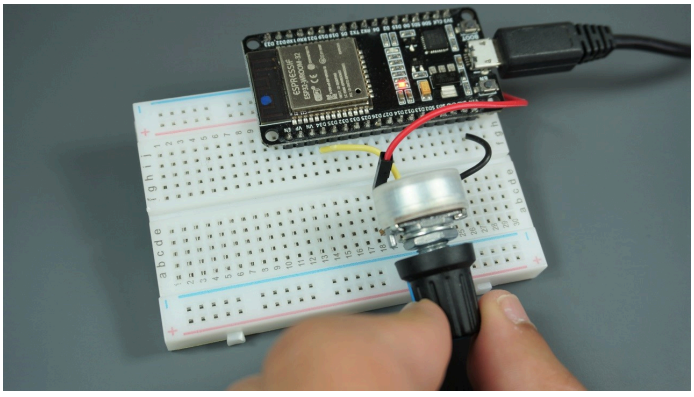
An ADC compares samples of the analog input voltage (produced using a Sample and Hold circuit) to a known reference voltage and then produces a digital representation of this analog input. The output of an ADC is a digital binary code.

By its nature, an ADC introduces a quantization error, which is simply the information that is lost. This error occurs because there are an infinite number of voltages for a continuous analog signal, but only a finite number of ADC digital codes. Therefore, the more digital codes that the ADC can resolve, the more resolution it has, and the less information lost to quantization error.

In A/D conversion, the Nyquist principle (derived from the Nyquist-Shannon sampling theorem) states that the sampling must be at least twice the maximum bandwidth of the analog signal being converted, in order to allow the signal to be accurately reproduced. The maximum bandwidth of the signal (half the sampling rate) is commonly called the Nyquist frequency (or Shannon sampling frequency). In real life, the sampling rate must be higher than that (because filters used to re-produce the original signal are not perfect). As an example, the bandwidth of a standard audio CD is a bit less than the theoretical maximum of 22.05kHz (based on the sample rate of 44.1kHz).

Bit Length	Levels	Step Size (5V Range)
8-bits	256	19.53 mV
10-bits	1024	4.88 mV
12-bits	4096	1.22 mV
16-bits	65536	76.29 µV
18-bits	262144	19.07 µV
20-bits	1048576	4.76 µV
24-bits	16777216	0.298 µV

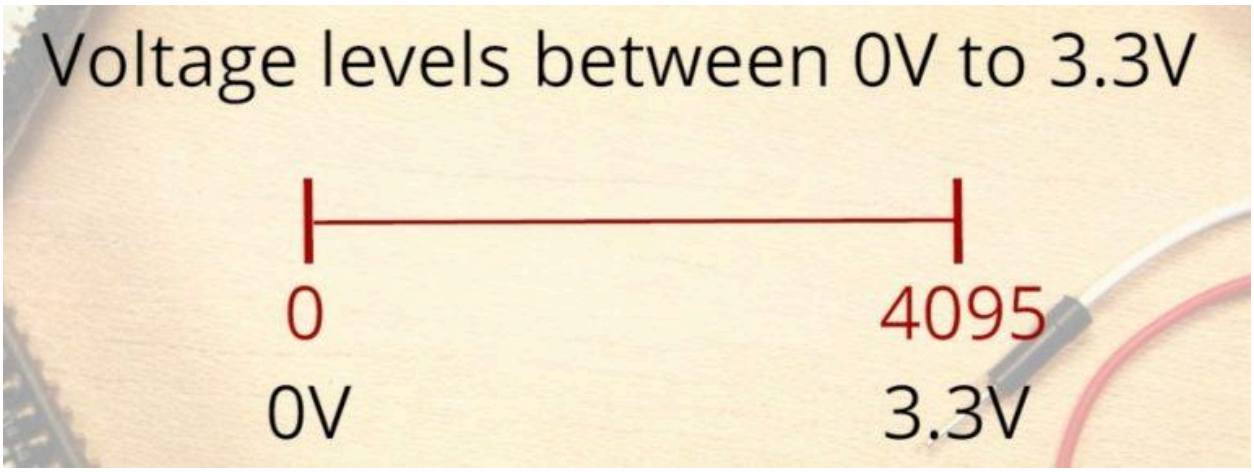
ADC interface with ESP32



Analog Inputs (ADC)

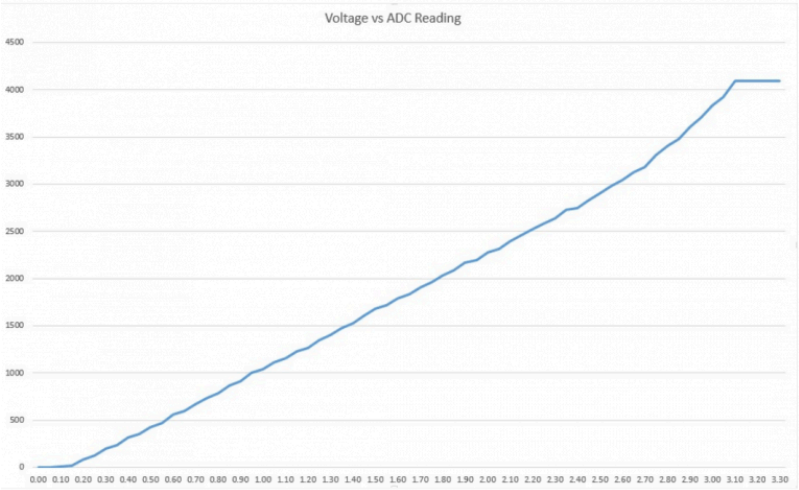
Reading an analog value with the ESP32 means you can measure varying voltage levels between 0 V and 3.3 V.

The voltage measured is then assigned to a value between 0 and 4095, in which 0 V corresponds to 0, and 3.3 V corresponds to 4095. Any voltage between 0 V and 3.3 V will be given the corresponding value in between.



ADC is Non-linear

Ideally, you would expect a linear behavior when using the ESP32 ADC pins. However, that doesn't happen. What you'll get is a behavior as shown in the following chart:



This behavior means that your ESP32 is not able to distinguish 3.3 V from 3.2 V. You'll get the same value for both voltages: 4095.

The same happens for very low voltage values: for 0 V and 0.1 V you'll get the same value: 0. You need to keep this in mind when using the ESP32 ADC pins.

There's a discussion on GitHub about this subject.

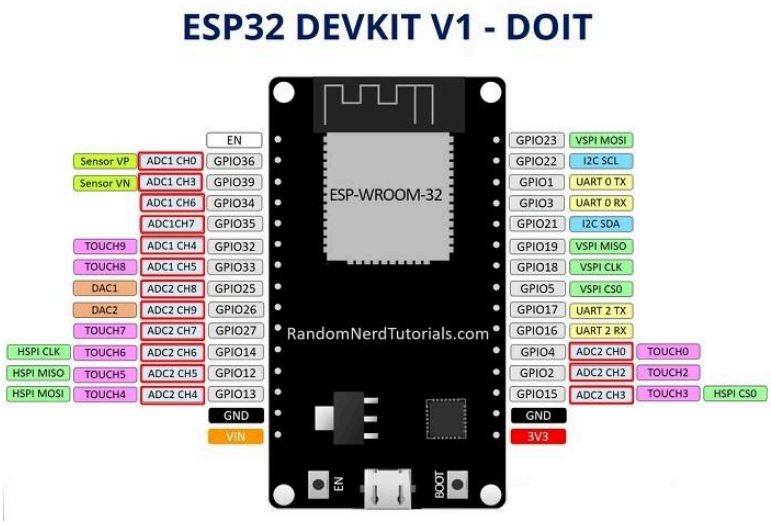
analogRead() Function

Reading an analog input with the ESP32 using the Arduino IDE is as simple as using the `analogRead()` function. It accepts as argument, the GPIO you want to read:

```
analogRead(GPIO);
```

The ESP32 supports measurements in 18 different channels. Only 15 are available in the DEVKIT V1 DOIT board (version with 30 GPIOs).

Grab your ESP32 board pinout and locate the ADC pins. These are highlighted with a red border in the figure below.



These analog input pins have 12-bit resolution. This means that when you read an analog input, its range may vary from 0 to 4095.

Note: ADC2 pins cannot be used when Wi-Fi is used. So, if you're using Wi-Fi and you're having trouble getting the value from an ADC2 GPIO, you may consider using an ADC1 GPIO instead, that should solve your problem.

Other Useful Functions

There are other more advanced functions to use with the ADC pins that can be useful in other projects.

- `analogReadResolution(resolution)`: set the sample bits and resolution. It can be a value between 9 (0 – 511) and 12 bits (0 – 4095). Default is 12-bit resolution.
- `analogSetWidth(width)`: set the sample bits and resolution. It can be a value between 9 (0 – 511) and 12 bits (0 – 4095). Default is 12-bit resolution.
- `analogSetCycles(cycles)`: set the number of cycles per sample. Default is 8. Range: 1 to 255.
- `analogSetSamples(samples)`: set the number of samples in the range. Default is 1 sample. It has an effect of increasing sensitivity.
- `analogSetClockDiv(attenuation)`: set the divider for the ADC clock. Default is 1. Range: 1 to 255.

- `analogSetAttenuation(attenuation)`: sets the input attenuation for all ADC pins. Default is `ADC_11db`. Accepted values:
- `ADC_0db`: sets no attenuation. ADC can measure up to approximately 800 mV (1V input = ADC reading of 1088).
- `ADC_2_5db`: The input voltage of ADC will be attenuated, extending the range of measurement to up to approx. 1100 mV. (1V input = ADC reading of 3722).
- `ADC_6db`: The input voltage of ADC will be attenuated, extending the range of measurement to up to approx. 1350 mV. (1V input = ADC reading of 3033).
- `ADC_11db`: The input voltage of ADC will be attenuated, extending the range of measurement to up to approx. 2600 mV. (1V input = ADC reading of 1575).
- `analogSetPinAttenuation(pin, attenuation)`: sets the input attenuation for the specified pin. The default is `ADC_11db`. Attenuation values are the same from the previous function.
- `adcAttachPin(pin)`: Attach a pin to ADC (also clears any other analog mode that could be on). Returns TRUE or FALSE results.
- `adcStart(pin)`, `adcBusy(pin)` and `resultadcEnd(pin)`: starts an ADC conversion on the attached pin's bus. Check if conversion on the pin's ADC bus is currently running (returns TRUE or FALSE). Get the result of the conversion: returns 16-bit integer.
- There is a very good video explaining these functions that you can watch here.

Read Analog Values from a Potentiometer with ESP32

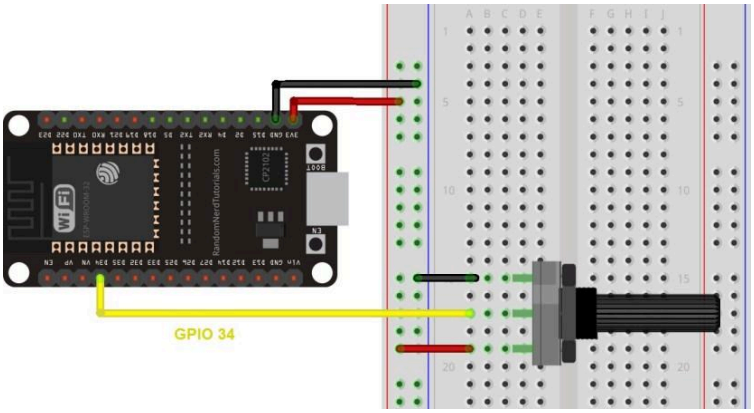
To see how everything ties together, we'll make a simple example to read an analog value from a potentiometer.

For this example, you need the following parts:

- ESP32 DOIT DEVKIT V1 Board (read Best ESP32 development boards)
- Potentiometer
- Breadboard
- Jumper wires

Schematic

Wire a potentiometer to your ESP32. The potentiometer middle pin should be connected to GPIO 34. You can use the following schematic diagram as a reference.



```
#include <Arduino.h>
#define potpin 34

int potval = 0;

void setup() {
  Serial.begin(115200);
  delay(1000);
}

void loop() {
  potval = analogRead(potpin);
  Serial.println(potval);
  delay(500);
}
```

Serial monitor :
0
0
0
70
464
1103
1999
2819
3860
4095
4095
4095
4095
3840
894
0
0

This code simply reads the values from the potentiometer and prints those values in the Serial Monitor.

In the code, you start by defining the GPIO the potentiometer is connected to. In this example, GPIO 34.

```
const int potPin = 34;
```

In the `setup()`, initialize a serial communication at a baud rate of 115200.

```
Serial.begin(115200);
```

In the `loop()`, use the `analogRead()` function to read the analog input from the `potPin`.

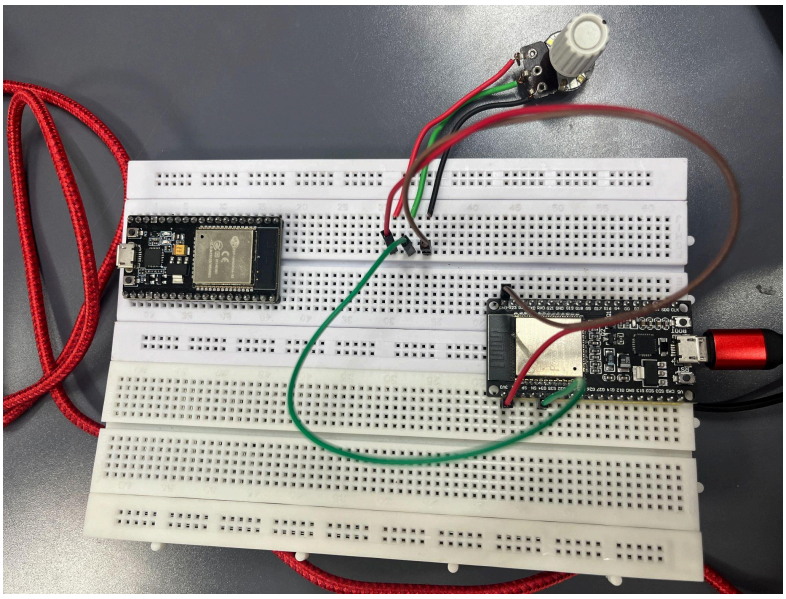
```
potValue = analogRead(potPin);
```

Finally, print the values read from the potentiometer in the serial monitor.

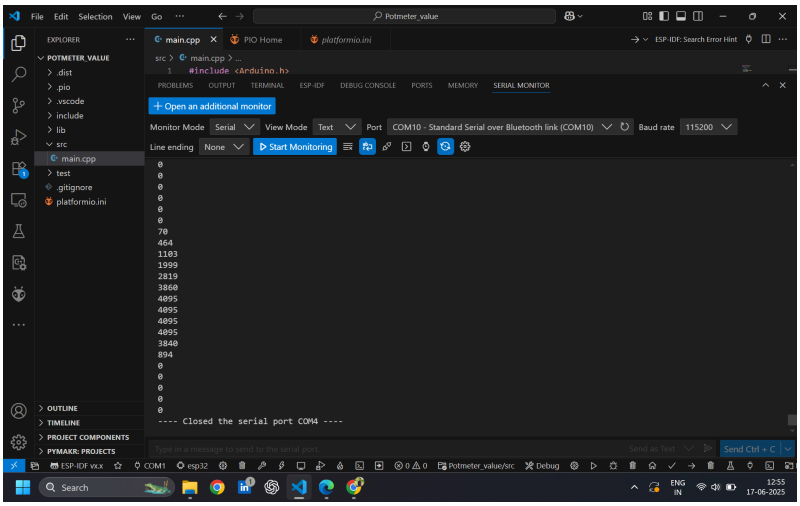
```
Serial.println(potValue);
```

Testing the Example

After uploading the code and pressing the ESP32 reset button, open the Serial Monitor at a baud rate of 115200. Rotate the potentiometer and see the values changing.



The maximum value you'll get is 4095 and the minimum value is 0.



Wrapping Up

In this article you've learned how to read analog inputs using the ESP32 with the Arduino IDE. In summary:

- The ESP32 DEVKIT V1 DOIT board (version with 30 pins) has 15 ADC pins you can use to read analog inputs.
- These pins have a resolution of 12 bits, which means you can get values from 0 to 4095.
- To read a value in the Arduino IDE, you simply use the `analogRead()` function.
- The ESP32 ADC pins don't have a linear behavior. You'll probably won't be able to distinguish between 0 and 0.1V, or between 3.2 and 3.3V. You need to keep that in mind when using the ADC pins.