

VIDEO#27Electronic Basics #27: ADC (Analog to Digital Converter)

Analog to Digital Converters (ADC)

Introduction to ADC

In a previous video, we explored how to use the digital output pins of an Arduino to generate an analog signal. Many of you may already know that the analog pins on an Arduino can be used in reverse to convert an analog voltage into a digital value. This process is done by an **Analog to Digital Converter** (ADC). For example, if you input a voltage of 3.48V with a reference voltage of 4.62V, the ADC converts it into a 10-bit digital value of 666.

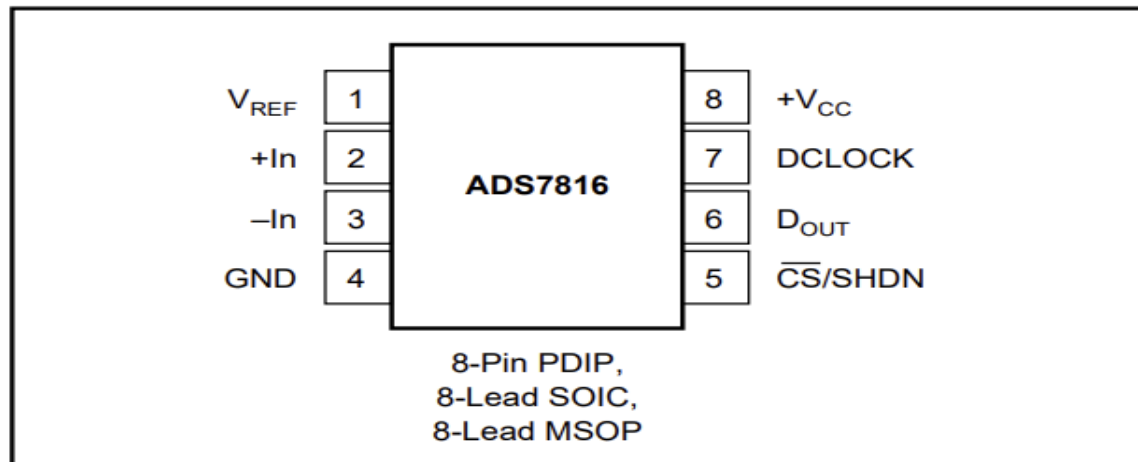
Key Specifications of an ADC

Sampling Rate

One of the most important specifications for an ADC is its **sampling rate**, which determines how frequently the ADC samples an input signal. Let's look at an example with a 10 kHz sine wave. The cycle duration of this wave is 100 microseconds, but the Arduino can only sample an analog signal every 112 microseconds by default. This results in a sampling rate of about 9 kHz.

According to the **Nyquist-Shannon theorem**, the sampling rate should be at least twice the frequency of the signal to avoid losing important data. However, even with this increased rate, the reconstructed signal may still appear questionable. A good rule of thumb is to use a sampling rate that is 10 times higher than the frequency of the signal to ensure a clean reconstruction. In the case of the Arduino, you can achieve a higher sampling rate by adjusting the ADC's pre scaler value down to 16. For dedicated ADCs like the ADS7816, sampling rates of up to 200 kHz are possible.

PIN CONFIGURATION



Resolution of ADC

The second important specification for an ADC is its **resolution**, which defines the precision with which the ADC can represent the input voltage. In simple terms, resolution determines the size of the voltage steps between digital values. For example, with a 4-bit resolution, the ADC can distinguish between 16 possible voltage levels. With a reference voltage of 5V, each step corresponds to 312.5 mV.

The Arduino uses a 10-bit ADC, which means there are 1,024 possible voltage levels between 0 and 5V. Each step corresponds to about 4.88 mV. For even higher precision, you can use ADCs with greater resolution, like the ADS7816, which has a 12-bit resolution, corresponding to 4,096 possible values, and a step size of 1.22 mV.

How ADC Works: Successive Approximation

The Arduino ADC uses a method called **Successive Approximation** to convert an analog input into a digital value. Here's how it works:

1. The input voltage is sampled and held steady by a capacitor.
2. The ADC compares the input voltage with a reference voltage using a **comparator**.
3. A **successive approximation register (SAR)** sets the most significant bit (MSB) and sends it to a **digital-to-analog converter (DAC)**.
4. The DAC generates a corresponding voltage, which is compared to the input voltage.
5. If the output voltage of the DAC is higher than the input, the bit is set to 0; otherwise, it is set to 1.

6. The process repeats for each bit until the digital value is determined.

PIN ASSIGNMENTS

PIN	NAME	DESCRIPTION
1	V _{REF}	Reference Input.
2	+In	Non Inverting Input.
3	-In	Inverting Input. Connect to ground or to remote ground sense point.
4	GND	Ground.
5	$\overline{\text{CS}}$ /SHDN	Chip Select when LOW, Shutdown Mode when HIGH.
6	D _{OUT}	The serial output data word is comprised of 12 bits of data. In operation the data is valid on the falling edge of DCLOCK. The second clock pulse after the falling edge of $\overline{\text{CS}}$ enables the serial output. After one null bit the data is valid for the next 12 edges.
7	DCLOCK	Data Clock synchronizes the serial data transfer and determines conversion speed.
8	+V _{CC}	Power Supply.



For example, with a 4-bit ADC and an input of 3V, the process results in a digital value of **2.8V** after completing all the steps.

Building an ADC

While many ADCs are built into microcontrollers like the Arduino, it is possible to build your own ADC. If you are interested in creating a higher-resolution ADC, you can use a dedicated ADC IC, such as the **ADS7816**, which is a 12-bit ADC. The process of integrating an external ADC into the Arduino system involves the following steps:

1. Study the datasheet of the ADC IC to understand its specifications.
2. Connect the ADC to the Arduino according to the pinout diagram.
3. Write an appropriate sketch to configure and control the ADC.
4. Upload the sketch to the Arduino, and monitor the output via the serial monitor.

The serial monitor will show the ADC values in both binary and decimal formats.

ADS7816 + AnalogReadSpeed Code file :  ADS7816.ino and  AnalogReadSpeed.ino

Flash ADC

Another type of ADC is the **Flash ADC**, which is suitable for applications that require very high-speed sampling but have lower resolution. A Flash ADC consists of multiple comparators and a resistance network. For example, a 2-bit Flash ADC uses four comparators to convert the input voltage into a 2-bit digital value. This design is very fast but is limited by its low resolution.

A higher-resolution Flash ADC would require many more comparators, making it impractical for use in higher bit resolutions (e.g., 8-bit Flash ADCs would require 255 comparators).

In conclusion, **ADC** plays a crucial role in converting analog signals into digital data for use in digital systems like Arduino. Understanding the **sampling rate**, **resolution**, and working principles of the ADC, like **successive approximation** or **flash ADC**, is essential when working with analog signals. While external ADCs like the **ADS7816** offer higher performance, building a simple **flash ADC** can be a great DIY project for learning about ADCs.