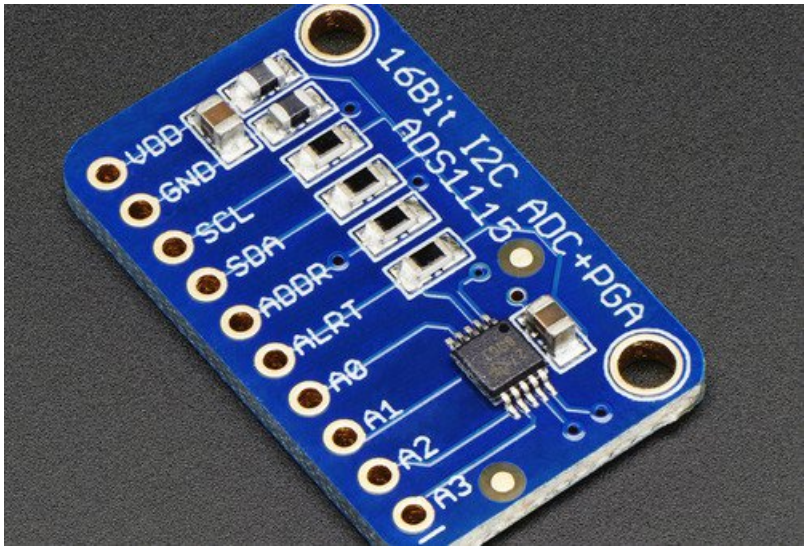
adafruit learning system

Choosing an ADC

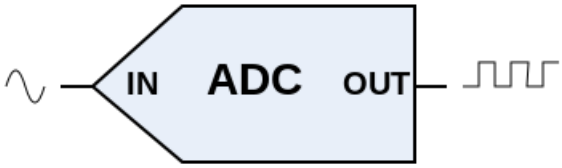
Created by mike stone



Last updated on 2019-05-24 05:16:48 PM UTC

Overview

This guide is intended to help engineers to choose an Analog to Digital Converter for their designs.



When you want to build a system that tracks an analog voltage, you need an Analog to Digital Converter (ADC) so your microcontroller or computer can read and record the results. If you want good measurements, you pick an ADC with the highest resolution you're willing to pay for, right?

Not so fast.

There's only a vague connection between an ADC's resolution and the quality of the output you get from it. A lot of 16-bit ADCs really operate as 7-bit ADCs and 9-bit random number generators.

What's Going On?

An ADC's static performance is controlled by a value called **noise free code resolution**, or **NFCR**, usually defined as six times the Root Mean Square (RMS) input-referred noise at the ADC's input pin.

An ADC's dynamic performance is controlled by a value I call **aperture slew rate**, or **ASR**. It depends on values you usually can't find in a datasheet.

Those statements contained a lot of undefined terms, and this tutorial will walk through the definitions. Once you understand them, you'll be able to make a more informed decision next time you look for an ADC.

Basic vocabulary

Let's start with some of the simplest terms first:

- The term **static** refers to signals that stay constant, or change slowly.
- The term **dynamic** refers to signals that change quickly.

The difference between 'slowly' and 'quickly' depends on what you're measuring, but usually boils down to, "do I care how much the signal changes while I'm measuring it?" If you can say "no", the input is static. If you have to say "yes", the input is dynamic.

For dynamic readings, the slew rate of the signal you want to measure (how quickly the signal changes) has a huge influence on the kind of ADC you'll need.

- A **code** is the output from an ADC.

In most cases a code is a binary number, but it doesn't have to be. The only real restriction is that different codes represent different voltages at the ADC's input.

Binary is inconvenient for precision analog work because different numbers of bits change from one consecutive value to the next: changing 0010 to 0011 only flips one bit, but changing 0111 to 1000 flips all four. If the bit-flipping has any

effect on the quality of the measurement (and in precision work you assume *everything* has some effect), the quality of the output can change with the output value itself.

Many ADCs use Gray Code (https://adafru.it/ESj), at least internally, because only one bit changes from one Gray code value to the next. Other ADCs use more advanced techniques known as 'whitening transforms (https://adafru.it/ESk)' to make the changes from one value to the next look like random noise.

And now that I've used that word:

- The term **noise** itself is complicated enough to get its own page, but:
- **Input referred noise** describes a way of simplifying noise calculations.

No real electrical device is perfect, and every real circuit has noise in several places. Trying to describe that kind of system is a hard, and in most cases we don't even try.

Instead, we pretend the circuit is noise free and imagine connecting a noise generator to the input. As long as we get the same amount of output noise from the same input signal, the simplified model is good enough to be useful. That process -- replacing a real component with a perfect one and adding a noise source at the input -- is called 'referring the noise to the input'.

A term I haven't used yet, but will use a lot in the future, is:

- **One least significant bit** or **1LSB**.

1LSB has two common definitions that look alike, but are actually quite different:

1. The smallest change of input that will make the ADC shift from one output code to another.
2. The smallest change of input you can measure repeatably.

The difference between those definitions hinges on the word 'repeatably' and its connection to noise, so let's meet the villains of our piece...

Error, noise, and distortion

The terms 'error' and 'noise' are often used to mean the same thing: the difference between a measured value and the real value of the input.

I'm not going to use them that way. For this tutorial, 'error' has the definition given above, and 'noise' is a specfic kind of error. Also for the purposes of this tutorial, error has two properties called **repeatability** and **correlation**.

Repeatable errors are caused by flaws in a measuring system, and are the best kind of errors to have. You can identify them, measure them, and fix them. **Unrepeatable errors** are caused by randomness in the universe. We can identify what's causing them, and there are ways to minimize them, but we can never eliminate them completely.

The process of measuring repeatable error is called **calibration**. You do it by comparing your measurement to a measurement of the same signal from another instrument you trust more. If you adjust your measurement system to eliminate calibrated errors, it's called **correction**. If you just subtract the calibrated error from your measurements, it's callled **compensation**.

Correlated errors have some kind of relationship to the thing you're measuring. You can define the error as a function of the input. **Uncorrelated errors** stay the same regardless of the input.

If you draw a graph of a circuit's ideal input-output function, then modify it to account for errors in the real circuit, uncorrelated errors don't change the shape of the curve. They just shift the real curve some distance away from the ideal one, so we generally call them **offsets**. Correlated errors do change the shape of the curve, and are generally called **distortion**.

Arranged by order of difficulty, the major categories of error are:

- **Repeatable uncorrelated errors, or 'fixed offsets'.**

These are the easiest to calibrate and correct.

- **Repeatable and unrepeatable correlated errors, or 'distortion'.**

Distortion is a big problem, and most forms of distortion are impossible to fully correct, but there's a huge body of engineering knowledge about compensating distortion. We know how to build circuits that minimize the effects of distortion, and we know how to move distortion to places we can ignore.

- **Unrepeatable uncorrelated errors, or 'noise'.**

This is the part we can never truly eliminate. We can select components that generate as little noise as possible, and there are ways to build circuits that minimize the number of noise sources. There are even ways to move noise to frequencies that can be ignored. There will always be some amount of noise that's impossible to remove though.

It's impossible to predict the noise error in any single measurement, but we can analyze noise statistically.

RMS noise

One interesting property of noise is that there's no upper limit to how big a single pulse can be. It is possible to find the average size of the pulses (called the **mean noise error**) though.

Ideally, the mean noise error will be zero: on average, the positive and negative pulses will cancel each other. But sometimes the pulses going one way will be larger or more frequent than the ones going the other way, and the mean noise error will be positive or negative.

If that happens, you can treat the mean noise as an uncorrelated error (an offset) and subtract it from the readings to compensate them. It doesn't eliminate the error in any single measurement, it just means that noise will make half of your measurements too big and the other half too small.

The natural follow-up question is, "how much too large or too small?" To answer that, we find the average difference between any single error and the mean.

There's a problem though: if the mean noise error is zero, the difference between each pulse and the mean is simply the value of the pulse itself. If we add them together, we'll get zero again.

Instead, we get rid of the negative values by squaring the size of each pulse. Then we can add the squared values together, take the square root of the sum, then divide that by the number of samples. The result is a value called the **root mean square deviation** of the noise or **RMS value**.

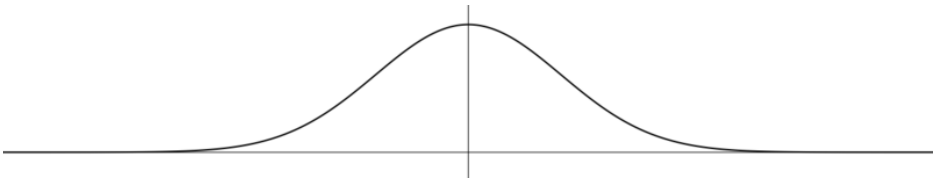
RMS values give us a way to describe how noise signals combine.

When multiple noise sources act on the same signal, they can either add together or cancel the same way the pulses from a single noise source cancel over time. We combine them the same way: squaring, adding, then finding the RMS value.

Noise distribution

One thing RMS values don't do is solve the 'no upper limit on the size of a single error' problem. Technically there is no solution, but we can deal with the problem using some more math and a few assumptions.

The most common kind of noise is called **Gaussian**, and has the same statistical properties as the **standard distribution:**

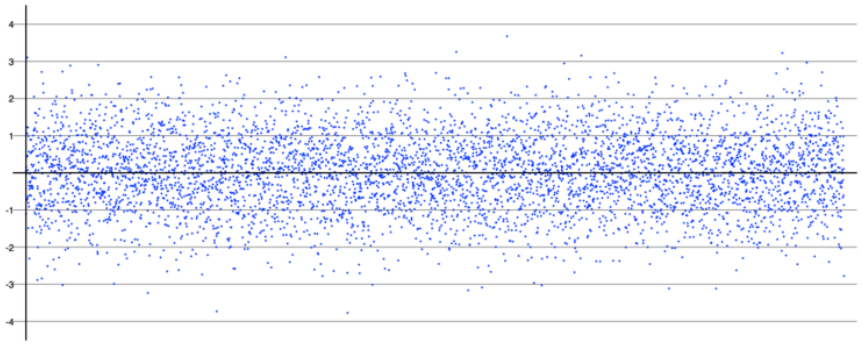


The RMSD of the standard distibution is called the **standard deviation**, which has qualities we can use.

For Gaussian noise, about 63% of the individual errors will be smaller than the standard deviation. The remaining 37% will be larger.

Of that 37%, about 63% of them will be smaller than two standard deviations, 63% of what's left will be smaller than three standard deviations, 63% of the remainder will be smaller than four standard deviations, and so on.

Here's a plot of 5000 Gaussian random numbers, with horizontal lines spaced one standard deviation apart:



As you can see, only a few values are more than 3 standard deviations from the mean (statistically, only 0.26% of them).

If we define an ADC's LSB in terms of standard deviations of the RMS noise, we can predict the distribution of output codes we'll get for a constant input. At 1 standard deviation per LSB, we can expect to see 8 different codes in a series of 5000 readings.

We usually set LSB at 6 standard deviations. That way if the input is halfway between two thresholds (+3 and -3 on the graph above), we can expect to get the same code 99.74% of the time.

If the input is equal to one of the thresholds (+6, 0, and -6), we can expect almost all of the output to be one of two codes (only one Gaussian value in 500 million is more than 6 standard deviations from the mean), and we can expect each of those two codes to appear 50% of the time.

NFCR revisited

Now we can put the definition of **Noise-Free Code Resolution** in known and useful terms:

> If we define NFCR as six times the RMS value of all input-referred noise in a sytem, the contribution from noise will only exceed 1LSB for 1 reading in 500 million.

That's why the resolution of the ADC itself has little to no effect on the quality of measurements:

- Only bits larger than the NFCR can be considered measurements of the signal.
- Bits smaller than the NFCR just measure the noise in that particular reading.

With that information, you can use NFCR as a design value. If you want to use an ADC with a given resolution, you have to build a circuit whose RMS noise is less than 1/6th of the ADC's LSB.

Here's how those numbers crunch for several common ADC resolutions working with a signal that swings between 0v and 5v:

BITS	1 LSB	MAXIMUM RMS NOISE
8	19.5 mV	3.25 mV
10	4.9 mV	814 uV
12	1.2 mV	203 uV
16	76.3 uV	12.7 uV
24	298 nV	49.7 nV
32	1.2 nV	192 pV

... and now you know why I twitch a little when people talk about wanting to use 24-bit or 32-bit ADCs.

Given all the problems we've covered so far, what do chip manufacturers mean when they advertise an umpty-many bit ADC?

> An ADC's resolution means the ADC itself won't contribute errors larger than the specified LSB, and can perform at that resolution under suitable conditions.
>
> It's up to the circuit designer to learn what those conditions are, to build a circuit that can live up to the ADC's capabilities, and to choose signals that don't exceed the ADC's limits.

Dealing with noise is only one part of working with an ADC. Even if your signal is perfectly noise-free, you can only be sure you're getting correct readings if you're measuring a DC voltage.

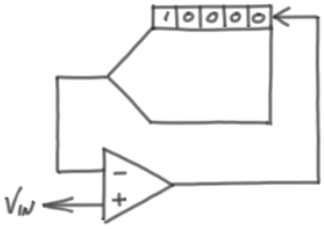
If you want to measure a signal that changes over time, you have to make sure your ADC can capture it correctly. To do that, you need to know something about the way ADCs work.

ADC architectures

The two kinds of ADC architecture most commonly used today are the **Successive Approximation Register** (SAR) and the **Sigma-Delta**.

Successive approximation

There are several ways to build a SAR, but the simplest uses a comparator and a DAC:



The process starts by setting the DAC's most significant bit to 1, which sets the DAC's output voltage to half its maximum output voltage. If the input is higher than the DAC's output, the comparator's output will be 1. If the input is lower than the DAC's output, the comparator's output will be 0.

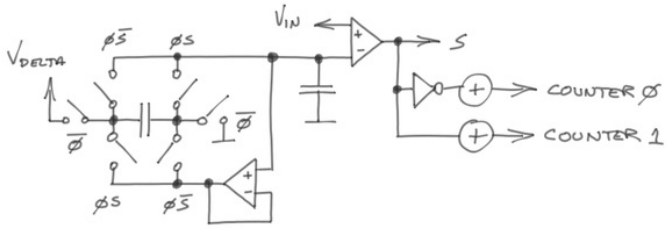
The SAR copies the comparator's output to the DAC's MSB, sets the DAC's next most significant bit to 1, and does the whole thing again. On the second pass the DAC's output only changes by a quarter of its maximum value, on the third pass it changes by an eighth, and so on.

If you're familiar with computer algorithms, a SAR does the hardware equivalent of a binary search.

When the DAC runs out of bits, the SAR has finished its approximation of the input voltage, and returns the DAC setting as output.

Sigma-delta

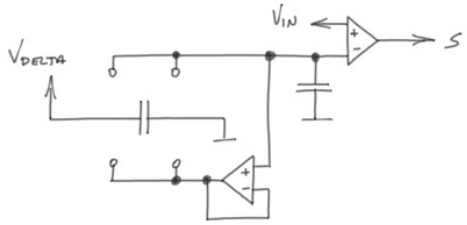
A Sigma-Delta ADC uses a comparator, two capacitors, two counters, and a clock:



The capacitor at the left, with all the switches around it, is called the flying capacitor. The capacitor on the right, at the input of the comparator, is called the summing capacitor. The summing cap is usually much larger than the flying cap.

The switches around the flying cap are controlled by the clock and the comparator's output. The schematic above looks complicated because it shows several possible connections all at once.

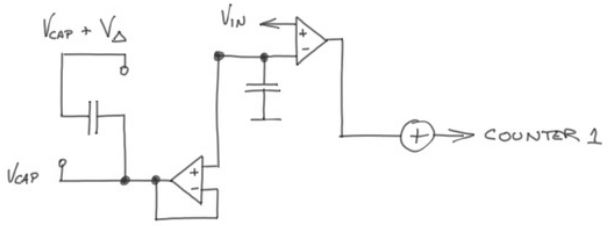
When the clock is low, the connections simplify to this:



The flying capacitor charges to a predetermined voltage called V_{delta}, while the comparator decides whether the summing cap's voltage is higher or lower than the input voltage. That decision is stored as the value S.

There are two possible sets of connections once the clock goes high.

If the clock is high and S is high, the connections simplify (almost) to this:



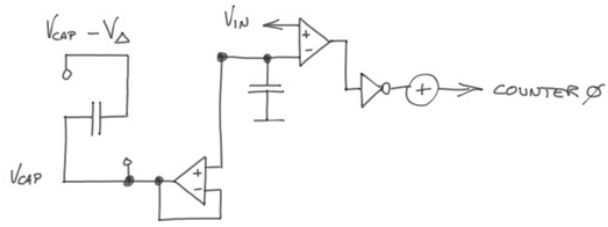
NOTE: I've removed the connection between the flying capacitor and the summing capacitor to make the flying cap's job easier to understand.

The voltage buffer (bottom center) copies the voltage at the top of the summing capacitor (V_{cap}). The negative end of the flying cap connects to that, which would put the positive end of the flying capacitor at $V_{cap} + V_{delta}$ if there was no connection to the summing cap.

In fact, the flying cap does connect to the summing cap, and dumps its charge into the summing cap. The capacitors reach equilibrium when the flying cap's voltage is 0V and the summing cap's voltage is slightly higher than it was before.

At the same time, the counter that tracks the number of times S is high increases by 1.

If the clock is high and S is low, the connections simplify (almost) to this:



This time the positive end of the flying cap is connected to the voltage buffer, and the negative end's voltage would be $V_{cap} - V_{delta}$ without a connection to the summing cap. In fact, the flying cap's voltage drops to 0V and the summing cap's voltage decreases slightly.

At the same time, the counter that tracks the number of times S is low increases by 1.

As long as the summing cap's voltage is lower than the input voltage, S will stay high and the 'S is high' counter will increase while the 'S is low' counter doesn't. The opposite is true as long as the summing cap's voltage is lower than the input: S stays low, and the 'S is low' counter advances while the 'S is high' counter doesn't.

Eventually the summing cap's voltage will get close enough to the input voltage that S changes with every tick of the clock: adding charge from the flying cap makes the summing cap voltage higher than the input, subtracting charge from the flying cap makes the summing cap voltage lower than the input, and both counters advance at the same rate.

Subtracting the 'S is low' counter from the 'S is high' counter, then multiplying the result by V_{delta} and the size of the flying cap tells us the amount of charge in the summing cap. Dividing that by the size of the summing cap tells us the summing cap's voltage (and the input voltage).

Mathematically, we use the symbol Sigma to represent the sum of a series. We use the symbol Delta to represent a small change. Putting them together as Sigma-Delta means 'the sum of a series of small changes', which describes what the ADC does.

Relative merits

A SAR can usually collect samples faster than a Sigma-Delta ADC with the same resolution. The SAR only needs N steps to approximate 2^N possible values, while the Sigma-Delta might need to count several thousand pulses of the flying cap.

A SAR also creates less noise in the circuit around it. Changing a DAC's output voltage doesn't require much current, and the changes get smaller and smaller with each step. A Sigma-Delta has to charge the flying cap over and over again, and needs a fast clock, both of which create sudden changes in the current load. It's been said that SARs are analog circuits that happen to produce digital output, while Sigma-Deltas are digital circuits whose output has a coincidental relationship with some analog voltage.

Sigma-Deltas have the strong advantage that input-referred Gaussian noise makes them work better.

When the summing cap's voltage is close to the input voltage, and the value of S is supposed to change with every tick of the clock, noise will push the summing cap's voltage a little higher or lower. Occasionally the voltage will change enough to make S stay high or low for more than one tick as the flying cap gets rid of the noise voltage.

If the input voltage is only slightly below V_{cap}, there's a good chance that noise will make V_{cap} stay higher than the input for more than one tick of the clock. The opposite is true if the input voltage is only slightly higher than V_{cap}. The two counters won't advance at the same average rate until V_{cap}'s voltage is halfway between the next-higher and next-lower voltages around it.

That combination of (relative) simplicity, ability to use noise, and almost unlimited resolution makes Sigma-Delta ADCs popular.

When it comes to applications, SARs tend to be more popular for jobs that require a high sampling rate and low-to-moderate resolution. Sigma-Deltas tend to be more popular for jobs that can tolerate a lower sampling rate but need higher resolution.

ADC timing

What most ADCs have in common

SAR and Sigma-Delta ADCs measure the input voltage in very different ways, but they both have one thing in common:

It takes time to complete a reading.

If the input voltage changes between the instant a reading starts and the instant it ends, it hurts the accuracy of the reading.

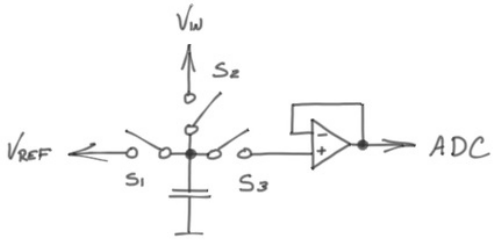
The SAR architecture is most sensitive to that kind of problem because it makes its largest assumptions first. A worst-case example would be a signal that starts just lower than VREF/2 when the SAR sets its most sigificant bit, then rises above VREF/2. The SAR's output will be 01111 regardless of how far above VREF/2 the input goes. The DAC's output can never cross VREF/2 after the MSB has been set to 0.

Sigma-Delta ADCs are better at tracking a moving signal, but are sensitive to changes near the end of the counting cycle. The worst-case example there is to have the summing capacitor still trying to reach the input voltage when the counting cycle ends.

ADC designers are painfully aware of those problems, and have found a solution that makes life better for both kinds of ADC:

The sample-and-hold buffer

A sample-and-hold circuit uses the same 'store voltage in a capacitor' trick as a Sigma-Delta ADC. The usual S&H circuit has a capacitor, a set of analog switches, and a voltage buffer:



The sample-and-hold process starts with S1 closed and the other two switches open. The capacitor charges to known voltage V_{ref}.

The next step is to open S1 and close S2, letting the capacitor charge to V_{in}.

A capacitor's voltage can't change instantly, and in most cases the rate of change is controlled by the size of the capacitor and the resistance of the signal source charging it. The product of those two values is called the **RC time constant**, and has the units of seconds. The capacitor's voltage gets 63% closer to Vin every RC time constant.

For an ADC to work accurately, the sample-and-hold circuit needs to let the capacitor to charge within 1LSB of Vin. Here's a table showing the minimum number of RC time constants necessary to bring the capacitor voltage within 1LSB of Vin for various ADC resolutions:

ADC Resolution	1LSB Fraction	RC Time Constants
8	0.0039	5.5
10	0.00097	6.9
12	0.00024	8.3
16	0.000015	11
24	0.000000059	16.6
32	0.00000000023	22

To get the capacitor voltage to within the recommended 1/6th of 1LSB, you'd need to add another 1.7 RC time constants to the values above.

Sampling capacitors are small (usually around 10pF), so with 1k of resistance from the signal source, one RC time constant is only 10 nanoseconds. Even so, the capacitor needs to stay connected to V_{in} long enough to charge properly.

The capacitor also needs to start from V_{ref} every time. If it starts at a different voltage (like the voltage of the previous sample), there will be some carry-over from one sample to the next. The rule for precision work is to try and do things the same way every time.

Once the capacitor voltage is sufficiently close to V_{in}, the sample-and-hold circuit can open S2 and close S3, connecting the capacitor to the input of a voltage buffer. The buffer protects the sampling capacitor from any disturbances caused by the ADC itself.

Ideally the capacitor voltage would stay constant until the sample-and-hold circuit opened S3 and closed S1, taking the capacitor voltage back to V_{ref} again. Unfortunately, all circuits have some leakage current, and it doesn't take much to create significant errors in a 10pF capacitor. The ADC's designers have to make sure the voltage remains stable long enough for the ADC to collect its reading.

Aperture slew rate

While a sample-and-hold circuit makes things easier for an ADC, it doesn't solve every problem.

Sampling time, LSBs, and ASR

The time the sampling capacitor is connected to V_{in} is known as its **aperture**. If the signal changes during the aperture, the best we can hope to say is:

> The captured voltage will be somewhere between the input's minimum and maximum values during the aperture.

That's a problem, but we already know how to solve it.

If we treat the unknown part of the captured voltage as noise, we can apply the same reasoning we used to define NFCR:

> As long as the input changes less than 1/6th of 1LSB during the aperture, we can assume the error due to that change will only exceed 1LSB for 1 reading in 500 million.

That gives us a limit on how much the signal can change, and if we know how long the aperture is, we know how fast the signal can be allowed to change, aka: the **Aperture Slew Rate**

So, all you have to do is open up the datasheet for an ADC, look for something like 'aperture', 'sampling interval', or some other equivalent term..

... and you probably won't find anything.

Some high-end ADCs will tell you their sampling interval, but most don't. We can estimate minimum aperture times though, based on the number of RC time constants it takes to charge the sampling capacitor to Vin:

ADC Resolution	Aperture for 1LSB	Aperture for 1/6 LSB
8	55ns	72ns
10	69ns	86ns
12	83ns	100ns
16	110ns	127ns
24	166ns	183ns
32	220ns	237ns

Those tell us how long the aperture lasts, and we know how much we can allow the voltage to change by calculating 1/6 LSB for an ADC's resolution and reference voltage. With those two values, we can figure out how quickly we can allow the signal to change.

A change in voltage over time is called a **slew rate**, and since this one is related to the aperture of a sample-and-hold circuit, I call it **aperture slew rate**:

ADC Resolution	1/6 LSB @ 5V	Aperture Slew Rate
8	3.25mV	45.1kV/s
10	814uV	9.5kV/s
12	203uV	2kV/s
16	12.7uV	100V/s
24	49.7nV	272mV/s
32	192pV	810uV/s

45.1 kilovolts per second sounds like a lot, but a 1Hz sine wave between 0V and 1V has a maximum slew rate of 3.14V/s. Increasing the amplitude so it swings between 0V and 5V takes the maximum slew rate to 15.7V/s.

The maximum frequencies you can read with an ADC while maintaining the aperture slew rate are:

ADC Resolution	Max frequency @ 1Vpp	Max frequency @ 5Vpp
8	14.4kHz	2.9kHz
10	3kHz	600Hz
12	637Hz	127Hz
16	31.8Hz	6.4Hz
24	86.6mHz	17.3mHz
32	257.3uHz	51.6uHz

Those frequencies are low, but remember they're calculated for a 10pF sampling capacitor charging from a source whose resistance is 1k. Lowering the resistance of the signal source will let the sampling capacitor charge faster, making it possible to measure higher frequencies.

The values are also calculated for less than 1LSB of error per sample. There's nothing to stop you from trying to measure signals faster than the values above, but you'll have to accept more error in your readings.

Putting the numbers to work

Now that you know about NFCR and ASR, you can use them to choose an ADC that's right for the job you want it to do.

Start by asking, "how big is the signal I want to measure?" and "how fast do I expect it to move?" Those will give you a minimum slew rate.

Then ask, "how often am I willing to accept at least 1LSB of error?" There's no point in paying for an ADC if you expect more than 1LSB of error in most of the readings, but are you willing to live with 37%? Honestly, that's good enough for a lot of things ADCs are called upon to do.

Once you know how often you're willing to accept errors, decide how big you want your LSB to be. It's much easier to get 1-in-500 million at 8-bit resolution than at 16-bit resolution.

Once you've chosen an LSB, run the numbers on ASR for the minimum slew rate you need. If you need an aperture time in the low picosecond range, talk to the bank and see if a mortgage on your house will cover the cost of that ADC (they exist, but generally cost more than the building around them).

Also run the numbers on NFCR and find out how quiet the circuit will have to be. When you subtract the cost of the ADC from your mortgage, can you still afford equipment capable of measuring at that level?

(bonus points for remembering that you have to add RMS noise and ASR to get the total error, and the sum will be 1.414 times either error by itself, assuming they're equal)

You may find it necessary to relax your initial expectations about resolution and error rate to find parts you can actually buy and circuits you can actually build.

Running the numbers and tradeoffs will give you specific things to look for when you start comparing ADCs:

- How much does it cost to hit your minimum slew rate at various resolutions?
- How much are you willing to trade increased error rate for increased noise tolerance or slew?
- Which ADC gives you the best resolution and slew rate in a given price range?
- Is the difference in performance between two chips worth the difference in price?

The answers will change from one project to the next, but knowing how to ask the questions will give you a chance of finding an ADC that actually does what you expect it to.

Glossary

This tutorial introduced a lot of terms, so here's a recap in case you need to refresh your memory on one:

1LSB:

One Least-Significant Bit: The smallest change of input voltage that will make an ADC's ouput change from one code to another, or the smallest voltage an ADC can measure repeatably.

ADC:

Analog-to-Digital Converter: A circuit whose output consists of digital patterns called codes. Each code corresponds to a specific input voltage.

Aperture:

The time when a sample-and-hold circuit is connected to the input.

ASR:

Aperture Slew Rate: The fastest a signal can change and still be measured accurately by an ADC.

Calibration:

Comparing a measurement to a better one, or measuring a known signal, to find a circuit's error.

Code:

A pattern of ones and zeros. In this case, a code is the output produced by an ADC.

Compensation:

Subtracting a known error from a measurement without actually correcting the circuit to remove the error.

Correction:

Modifying a circuit to remove a repeatable error.

Correlation:

A relationship between the error and the signal being measured.

Distortion:

Error that changes depending on the size of the input signal.

Dynamic signals:

Signals that change at least 1LSB during an ADC's sampling interval.

Error:

The difference between the actual input voltage and the voltage represented by the ADC's output code.

Gaussian distribution:

A common kind of random signal with convenient statistical properties.

Input-referred error:

The signal you'd have to add to a perfect ADC's input to make it produce the same output as a real ADC.

NFCR:

Noise-Free Code Resolution: The amount of useful information you can get from an ADC in a given set of operating conditions.

Noise:

For the purposes of this tutorial, noise is random error that can't be corrected or compensated.

Offset:

A DC voltage added to another signal.

Repeatability:

The ability to reproduce a condition on demand.. the main difference between ideal circuits/signals and real ones.

RMS:

Root-mean-square: A way to measure AC signals that lets us compare them to DC measurements.

S&H:

Sample and Hold: The input circuit for most ADCs. It captures the input quickly, then holds that voltage stable while the ADC generates an output code.

Sample rate:

The number of different codes an ADC can produce in one second.

Sampling interval:

The time it takes an ADC to generate a code.

SAR:

Successive Approximation Register: an ADC architecture that compares the input to its last guess several times, reducing the error with each new guess.

Sigma-Delta:

An ADC architecture that uses positive and negative pulses of known charge to make a capacitor's voltage equal the input voltage.

Slew rate:

The amount a signal's voltage changes over time.

Standard deviation:

The band of values around a signal's average that contains 68% of the readings.

Static signals:

Signals that stay at a fixed voltage, or change less than 1LSB during an ADC's sampling interval.