EE|Times

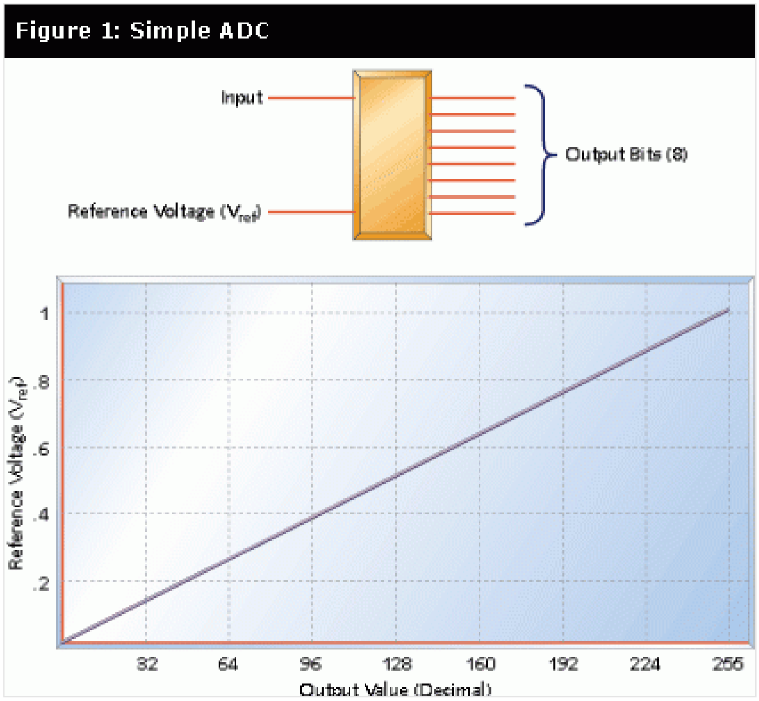
DESIGNLINES | **MCU DESIGNLINE< https://www.eetimes.com/designline/mcu-designline/>**

Analog-to-Digital Converters

By Stuart Ball 05.01.2001 3

The usual method of bringing analog inputs into a microprocessor is to use an analog-to-digital converter (ADC). Here are some tips for selecting such a part and calibrating it to fit your needs.

In analog-to-digital converter (ADC) accepts an analog input-a voltage or a current-and converts it to a digital value that can be read by a microprocessor.



< https://www.eetimes.com/wp-content/uploads/0105feat1fig1.gif>

Figure 1: Simple ADC

Figure 1 shows a simple voltage-input ADC. This hypothetical part has two inputs: a reference and the signal to be measured. It has one output, an 8-bit digital word that represents the input value.

The reference voltage is the maximum value that the ADC can convert. Our example 8-bit ADC can convert values from 0V to the reference voltage. This voltage range is divided into 256 values, or steps. The size of the step is given by:

$V_{ref}/256$

where V_{ref} is the reference voltage. The step size of the converter defines the converter's resolution. For a 5V reference, the step size is:

5V/256 = 0.0195V or 19.5mV

Our 8-bit converter represents the analog input as a digital word. The most significant bit of this word indicates whether the input voltage is greater than half the reference (2.5V, with a 5V reference). Each succeeding bit represents half the range of the previous bit.

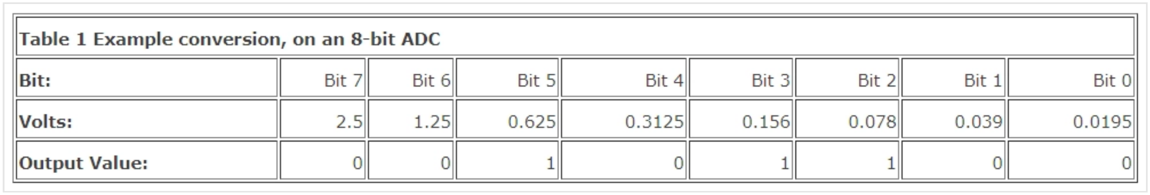


Table 1 Example conversion, on an 8-bit ADC

Bit:	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Volts:	2.5	1.25	0.625	0.3125	0.156	0.078	0.039	0.0195
Output Value:	0	0	1	0	1	1	0	0

< https://www.eetimes.com/wp-content/uploads/8-bit-ADC-Conversion.png>

Click on image for larger view

Table 1 illustrates this point. Adding the voltages corresponding to each set bit in 0010 1100, we get:

.625 + .156 + .078 = .859 volts

The resolution of an ADC is determined by the reference input and by the word width. The resolution defines the smallest voltage change that can be measured by the ADC. As mentioned earlier, the resolution is the same as the smallest step size, and can be calculated by dividing the reference voltage by the number of possible conversion values.

For the example we've been using so far, the resolution is 19.5mV. This means that any input voltage below 19.5mV will result in an output of 0. Input voltages between 19.5mV and 39mV will result in an output of 1. Between 39mV and 58.6mV, the output will be 2.

Resolution can be improved by reducing the reference input. Changing that from 5V to 2.5V gives a resolution of 2.5/256, or 9.7mV. However, the maximum voltage that can be measured is now 2.5V instead of 5V.

The only way to increase resolution without reducing the range is to use an ADC with more bits. A 10-bit ADC has 2^{10}, or 1,024 possible output codes. So the resolution is 5V/1,024, or 4.88mV; a 12-bit ADC has a 1.22mV resolution for this same reference.

Types of ADCs

ADCs come in various speeds, use different interfaces, and provide differing degrees of accuracy. The most common types of ADCs are flash, successive approximation, and sigma-delta.

Flash ADC

The flash ADC is the fastest type available. A flash ADC uses comparators, one per voltage step, and a string of resistors. A 4-bit ADC will have 16 comparators, an 8-bit ADC will have 256 comparators. All of the comparator outputs connect to a block of logic that determines the output based on which comparators are low and which are high.

The conversion speed of the flash ADC is the sum of the comparator delays and the logic delay (the logic delay is usually negligible). Flash ADCs are very fast, but consume enormous amounts of IC real estate. Also, because of the number of comparators required, they tend to be power hogs, drawing significant current. A 10-bit flash ADC may consume half an amp.

A variation on the flash converter is the half-flash, which uses an internal digital-to-analog converter (DAC) and subtraction to reduce the number of internal comparators. Half-flash converters are slower than true flash converters but faster than other types of ADCs. We'll lump them into the flash converter category.

Successive approximation converter

A successive approximation converter uses a comparator and counting logic to perform a conversion. The first step in the conversion is to see if the input is greater than half the reference voltage. If it is, the most significant bit (MSB) of the output is set. This value is then subtracted from the input, and the result is checked for one quarter of the reference voltage. This process continues until all the output bits have been set or reset. A successive approximation ADC takes as many clock cycles as there are output bits to perform a conversion.

Sigma-delta

A sigma-delta ADC uses a 1-bit DAC, filtering, and oversampling to achieve very accurate conversions. The conversion accuracy is controlled by the input reference and the input clock rate.

The primary advantage of a sigma-delta converter is high resolution. The flash and successive approximation ADCs use a resistor ladder or resistor string. The problem with these is that the accuracy of the resistors directly affects the accuracy of the conversion result. Although modern ADCs use very precise, laser-trimmed resistor networks, some inaccuracies still persist in the resistor ladders. The sigma-delta converter does not have a resistor ladder but instead takes a number of samples to converge on a result.

The primary disadvantage of the sigma-delta converter is speed. Because the converter works by oversampling the input, the conversion takes many clock cycles. For a given clock rate, the sigma-delta converter is slower than other converter types. Or, to put it another way, for a given conversion rate, the sigma-delta converter requires a faster clock.

Another disadvantage of the sigma-delta converter is the complexity of the digital filter that converts the duty cycle information to a digital output word. The sigma-delta converter has become more commonly available with the ability to add a digital filter or DSP to the IC die.

ADC comparison

< https://www.eetimes.com/wp-content/uploads/0105feat1fig2.gif>

Figure 2: ADC Comparison

Figure 2 shows the range of resolutions available for sigma-delta, successive approximation, and flash converters. The maximum conversion speed for each type is shown as well. As you can see, the speed of available sigma-delta ADCs reaches into the range of the successive approximation ADCs, but is not as fast as even the slowest flash

ADCs. What the tables do not show is the tradeoff between speed and accuracy. For instance, while you can get successive approximation ADCs that range from 8 to 16 bits, you won't find the 16-bit version to be the fastest in a given family of parts. The fastest flash ADC won't be the 12-bit part, it will be a 6- or 8-bit part.

These charts are a snapshot of the current state of the technology. As CMOS processes have improved, successive approximation conversion times have moved from tens of microseconds to microseconds. Not all technology improvements affect all types of converters; CMOS process improvements speed up all families of converters, but the ability to put increasingly sophisticated DSP functionality on the ADC chip doesn't improve successive approximation converters. DSP functionality does improve sigma-delta types because it enables better, faster, and more complex filters to be added to the part.

Sample and hold

ADC operation is straightforward when a DC signal is being converted. But if the input signal varies by more than one least significant bit (LSB) during the conversion time, the ADC will produce an incorrect (or at least inaccurate) result. One way to reduce these errors is to place a low-pass filter ahead of the ADC. The filter parameters are selected to ensure that the ADC input does not change by more than one LSB within a conversion cycle.

Another way to handle changing inputs is to add a sample-and-hold (S/H) circuit ahead of the ADC.

< https://www.eetimes.com/wp-content/uploads/0105feat1fig3.gif>

Figure 3: Sample and hold

Figure 3 shows how a sample-and-hold circuit works. The S/H circuit has an analog (solid state) switch with a control input. When the switch is closed, the input signal is connected to the hold capacitor and the output of the buffer follows the input. When the switch is open, the input is disconnected from the capacitor.

The figure shows the waveform for S/H operation. A slowly rising signal is connected to the S/H input. While the control signal is low (sample), the output follows the input. When the control signal goes high (hold), disconnecting the hold capacitor from the input, the output stays at the value the input had when the S/H switched to hold mode. When the switch closes again, the capacitor charges quickly and the output again follows the input. Typically, the S/H will be switched to hold mode just before the ADC conversion starts, and switched back to sample mode after the conversion is complete.

In a perfect world, the hold capacitor would have no leakage and the buffer amplifier would have infinite input impedance, so the output would remain stable forever. In the real world, though, the hold capacitor will leak and the buffer amplifier input impedance is finite, so the output level will slowly drift down toward ground as the capacitor discharges.

The ability of an S/H circuit to maintain the output in hold mode is dependent on the quality of the hold capacitor, the characteristics of the buffer amplifier (primarily input impedance), and the quality of the sample/hold switch (real electronic switches have some leakage when open). The amount of drift exhibited by the output when in hold mode is called the droop rate, and is specified in millivolts per second, millivolts per microsecond, or microvolts per microsecond.

A real S/H circuit also has finite input impedance, because the electronic switch isn't perfect. This means that in sample mode, the hold capacitor is charged through some resistance. This limits the speed with which the S/H can acquire an input. The time that the S/H must remain in sample mode in order to acquire a full-scale input is called the acquisition time, and is specified in nanoseconds or microseconds.

Since some impedance is in series with the hold capacitor when sampling, the effect is the same as a low-pass RC filter. This limits the maximum frequency that the S/H can acquire. This is called the full power bandwidth, and is specified in kilohertz or megahertz.

As mentioned, the electronic switch is imperfect and some of the input signal appears at the output, even in hold mode. This is called feedthrough, and is typically specified in decibels.

The output offset is the voltage difference between the input and the output. S/H circuit datasheets typically show a hold mode offset and sample mode offset in millivolts.

Software

An ADC system that uses a S/H may have to accommodate the hardware quirks. In some systems, the software directly controls the S/H control input with a port or register output bit. Typically, the S/H is placed into sample mode, and the software must ensure that the acquisition time requirement is met. In some systems, this can be accomplished simply by leaving the S/H in sample mode until a conversion is needed.

After the S/H is placed into hold mode, another bit (or a write to an address or some other operation) starts the ADC. After the conversion is complete, the software reads the result. However, a problem may occur if any one interrupt (or a worst-case stackup of interrupts) causes the output of the S/H circuit to droop by more than one LSB. If this could happen, the software may need to disable interrupts before switching the S/H to hold mode and re-enable them after starting the conversion. This ensures that the ADC will complete the conversion before the S/H droop occurs.

Software must also accommodate the charge time of the S/H. When the electronic switch closes and connects the input signal to the S/H capacitor, it takes a finite amount of time for the capacitor to charge because the switch and whatever source is driving the input both have nonzero impedances. If the sum of these impedances is large enough, the software may need to add a delay so the hold capacitor has time to charge to within one LSB of the final value before starting the conversion.

Internal microcontroller ADCs

Many microcontrollers contain on-chip ADCs. Typical devices include the Microchip PIC167C7xx family and the Atmel AT90S4434. Most microcontroller ADCs are successive approximation because this gives the best tradeoff between speed and the cost of real estate on the microcontroller die.

The PIC16C7xx microcontrollers contain an 8-bit successive approximation ADC with analog input multiplexers. The microcontrollers in this family have from four to eight channels. Internal registers control which channel is selected, start of conversion, and so on. Once an input is selected, a settling time must elapse to allow the S/H capacitor to charge before the A/D conversion can start. The software must ensure that this delay takes place.

Conversion accuracy

Some microcontrollers, such as the Microchip family, allow you to use one input pin as a reference voltage. This is normally tied to some kind of precision reference. The value read from the A/D converter after a conversion is:

$(V_{in}/V_{ref}) \times 256$

Some microcontrollers use the supply voltage as a reference. In a 5V system, this means that $V_{ref}$is always 5V. So measuring a 3.2V signal with an 8-bit ADC would produce the following result:

$(V_{in} \times 256)/V_{ref}$

$= (3.2v \times 256)/5V$

$= 163_{10}$

$= A3_{16}$

However, the result is dependent on the value of the 5V supply. If the supply voltage is high by 1%, it has a value of 5.05V. Now the value of the A/D conversion will be:

$(3.2V \times 256)/5.05V = 162_{10} = A2_{16}$

So a 1% change in the supply voltage causes the conversion result to change by one count. Typical power supplies can vary by 2% or 3%, so power supply variations can have a significant effect on the results. Power supply outputs can frequently vary with loading, temperature, AC input variations, and from one supply to the next.

This brings up an issue that affects all ADC designs: the accuracy of the reference. A typical ADC reference might be nominally 2.5V, but can vary between 2.47V and 2.53V (these values are from the data sheet for a real part). If this is a 10-bit ADC, converting a 2V input at the extremes of the reference ranges gives the following results:

At V_{ref}= 2.47V,

Result = $(2V \times 1{,}024)/2.47 = 829_{10}$

At V_{ref}= 2.53V,

Result = $(2V \times 1{,}024)/2.53 = 809_{10}$

The variation in the reference voltage from part to part can result in an output variation of 20 counts.

< https://www.eetimes.com/wp-content/uploads/0105feat1fig4.gif>

Figure 4: Reference voltage variation

Figure 4 shows the effect a reference variation has on the ADC result. Although the percentage of error stays the same throughout the range, the numerical error is of course greater for larger ADC values.

Software calibration

Sometimes you need an accurate reference, more accurate than the product cost will support. When manual adjustment is out of the question, the software can compensate for reference voltage variations. This is typically done by providing a known, precise input, which is used to calibrate the ADC. This reference can be very precise (and very expensive) because only a few are needed for the production line.

In the 2.47V example we've been looking at, a precise voltage of 2V might be input to the ADC. When the software reads the ADC, it knows the correct value should be 819; the calibration constant is given by 829/819, or 1.012. Similarly, the calibration constant for the 2.53V reference is 809/819, or 0.988.

This would seem to imply the need for floating-point math to correct the ADC value. If you are using a processor capable of floating-point, this is an acceptable approach. On simpler processors, though, you may not have the execution time or the code space available to implement floating-point calculations.

One way to handle the ADC correction is to use a lookup table. This has the drawbacks of requiring sufficient nonvolatile storage to maintain a lookup value for every possible ADC value-a 1,024-word table for a 10-bit ADC.

A voltage reference is fairly close to its nominal value-otherwise it would not be useful as a reference. Assuming that your reference is sufficiently stable over your operating temperature, the ADC error will be a constant percentage of the value you read from the ADC. Since the ADC has a finite resolution, there is no point in attempting to correct the ADC error with any precision greater than 1 LSB.

Knowing this, you can simplify the ADC correction process. Instead of a lookup table, you store a value that tells the software what (binary) percentage to add or subtract from the ADC reading to correct the error. You can add or subtract 1/8, 1/16, or 1/24, all the way down to 1 LSB of accuracy. You only need to store a single calibration constant, and your division process consists of a series of shift-and-add or shift-and-subtract operations.

The 2.47V example could be corrected by multiplying the ADC value by .988. The same thing can be achieved by subtracting 1/128 then 1/256 then 1/512 of the initial value. Using the original 2V input example, and doing this with integer math, we get the following:

829 — 829/128 — 829/156 — 829/512

= 829 — 6 — 3 — 1

= 819

This result corrects the ADC reading to 819, which is the ideal value if the reference were the nominal 2.5V. Similarly, values read with the 2.53V reference can be corrected by adding 1/128 plus 1/256.

Note that you do not need to apply the precise calibration voltage to the input you are using. You can use any spare ADC input, so long as that ADC uses the reference you want to calibrate.

You need to be sure that your reference is sufficiently stable over your expected operating temperature range or the results will only be good near the temperature during calibration. If the temperature stability of the reference is not good enough, you will have to get a better reference or break your operating temperature range into multiple segments and use one calibration value for each segment. Of course, this implies that you must have a thermistor or other means of measuring the temperature, too.

This approach does result in rounding errors caused by the truncation that occurs when you shift the result. I made a spreadsheet using the 2.47V example, and in all cases the corrected value was within two counts of the ideal value. Most corrected values were exactly right or off by just one. This degree of correction is significantly better than the original variation (10 counts) for a 2V input, and is all that many applications require. If your application can't stand even this error, then you may really need a better reference or you may have to resort to a manual adjustment.

This calibration technique can also be used to compensate for other system inaccuracies, such as resistor tolerance stackup. If whatever you are measuring consists of a voltage input, you can apply the precision voltage to that input and do one calibration to compensate for reference variations in the ADC and resistor tolerance effects in the input conditioning.

Calculating and using the calibration value

The calibration value can be calculated by reading a known reference and then finding which correction factors (binary submultiples) to use. For the example given, the difference between the ideal and the worst-case ADC value will never be more than 1.2%, so there is no point in starting with one-half or one-fourth of the original value. The only values tested and used are 1/128, 1/256, and 1/512. You want to start with something close to the value you expect to see.

Finding the correction factors is easy with a calculator, but if you have to calculate it on the fixed-point processor you will be using in your application, you need an integer-based approach.

< https://www.eetimes.com/wp-content/uploads/Figure-5-Calculating-and-using-the-calibration-values.gif>

Figure 5: Calculating and using the calibration values

Figure 5 shows in flowchart form the algorithm used in this example to calculate and use the calibration constant. In this method, a single byte (or word) is used to store the calibration constant. Bit 7 indicates whether the reference voltage is low (calibration values need to be subtracted) or high (calibration values added). Bits 0, 1, and 2 indicate whether the 1/128, 1/256, and 1/512 factors are used.

Of course, you could use a separate byte for each possible factor, with a fourth byte to indicate whether the reference is high or low.

Writing the calibration values

Whether you use a table or a calibration constant, how do you get the calibration values into the system? A key component of any calibration scheme is the availability of nonvolatile storage. Many microcontrollers have on-chip EEPROM. Calibration is typically performed when the circuit board is tested. In a high volume production environment, this will probably be done by some kind of bed-of-nails automatic test equipment.

You will typically want to put the processor into some kind of "calibration mode," possibly by grounding a pin. The production test equipment can be programmed to apply a very precise voltage to an analog input and ground the calibration pin. The microcontroller can then enter calibration mode, where it reads the reference value and calculates the compensation value or creates a lookup table.

In some cases, you don't have sufficient memory to add the calibration code to the microcontroller. In this case, you can have the microcontroller return the ADC value to an output pin (serially) or to a group of pins (parallel), where it is read by the production test equipment. An external computer can then calculate the calibration or table values and return them to the microcontroller via the same interface.

If the production equipment also programs the microcontroller in-circuit, the calibration data can be embedded into the data programmed into the flash memory. If the reference being calibrated is inside the microcontroller, the test equipment may have to first load a calibration program into the microcontroller, perform the calibration, then load the actual application code.

Finally, some very small microcontrollers simply don't have enough pins to give any up for calibration. In this case, you can often make an output pin do double duty as a calibration pin. You pull the pin up with an external resistor. The production equipment grounds the pin prior to power-up to select calibration mode.

The way this works is the microcontroller powers up with all the pins in the input state. It reads the calibration pin before configuring the pin as an output. If the pin is high, normal operation is started. If the pin is low, it must be externally grounded, so the microcontroller enters calibration mode. Of course, the output has to be one that won't damage anything when the pin is externally grounded.

Finally, if you are calibrating the reference applying the precise voltage to a spare ADC input, you can use that input itself to put the system into calibration mode. Use a resistor to pull the spare input to the zero-scale ADC voltage (ground, in the examples we've been using). Then have the software enter calibration mode when a voltage over some predetermined threshold (say, two-thirds of the full-scale voltage) is detected on the pin.

When selecting a calibration voltage, you want to choose the largest value that will not saturate the ADC when the reference voltage is at its lowest possible value. This ensures that you don't lose accuracy in calculating the calibration constant (or table) because of bit rounding errors. This will typically put the calibration voltage above 90% of the full scale value, although you may want to choose the nearest standard reference voltage to make the design easy.

In some applications, you can get around the reference issue by looking for a change in the ADC input. You might be able to watch for an optical sensor to change by 10% instead of comparing it to a fixed value, or you might watch for a temperature to go down by 25%. Of course, the accuracy of the sensor enters into this as well, but that topic is beyond the scope of this article.

Although it is sometimes difficult to know which ADC to use for your application, the wide array of parts available assures that you will find one to suit your needs. Matching the software to the hardware ensures that you will get the accuracy and reliability your product calls for.

Stuart Ball is an electrical engineer with over 20 years of experience designing embedded systems. He is the author of three books about embedded systems. The material in this article is adapted from Stuart's latest book, Analog Interfacing to Embedded Microprocessors, which was published earlier this year by Butterworth-Heinemann. You can contact him at **SBall85964@aol.com** or **stuart@stuartball.com**.

To learn more about simplifying power designs, register for this ***free webinar < https://event.on24.com/eventRegistration/EventLobbyServlet?target=reg20.jsp&referrer=&eventid=1639121&sessionid=1&key=B964A3362103AB81A8571CCDEE3EC760&regTag=&sourcepage=register>*** *, "Simplify Power Designs with Micromodules Products" sponsored by Analog Devices.*

Share this:

Twitter < https://www.eetimes.com/analog-to-digital-converters/?share=twitter&nb=1>

Facebook < https://www.eetimes.com/analog-to-digital-converters/?share=facebook&nb=1>

LinkedIn < https://www.eetimes.com/analog-to-digital-converters/?share=linkedin&nb=1>

SPONSORED CONTENT

WIND
THE BUSINESS CASE FOR FULL SYSTEM SIMULATION IN EMBEDDED DEVELOPMENT

Why Hardware and System Simulation is a Game-Changer in Embedded Development

When embedded systems software development and testing are constrained by the availability of target hardware and systems, the entire business faces problems. Read this paper to learn how Wind River Simics software enables developers and testers to work in virtual labs using software-based simulations of any target system.

Sponsored By Wind River

RELATED ARTICLES

EDN < https://www.edn.com/analog-to-digital-converters/?

Analog-to-Digital Converters < https://www.edn.com/analog-to-digital-converters/?

By EDN 01.05.01

EDN < https://www.edn.com/adcs-for-dsp-part-1/?

ADCs for DSP, part 1 < https://www.edn.com/adcs-for-dsp-part-1/?

By EDN 04.10.07

WIRELESS INDUSTRY RESOURCES

Intro to Bluetooth® LE Development

Provides foundation-level information and hands-on labs that walk you through developing a Bluetooth LE device.

DOWNLOAD HERE

ARTNER: Bluetooth®