

Analogue to Digital Converters Explained

Bill Naylor, Electronworks Ltd

This article explains the basics of analogue to digital converters, how they digitise an input signal, what output formats they have and how to write the software that drives them.

If you have any suggestions for improving this application note, please drop us a line at: enquiries@electronworks.co.uk. We welcome your feedback!

Introduction

This article describes how an analogue to digital converter (ADC) works, where you use them, some of the tricks associated with designing with them and some fun you can have with them. As always, there is a series of kits to go with this application note, so you can put the theory into practice.

What is an ADC? This is a component that converts a real world analogue signal (a voltage that can have an infinite number of values) into a number that can be understood by a microprocessor. Supposing you want to design a digital thermometer... You may have a sensor that outputs, say 1V per °C. You may choose to have a microprocessor that will then, say, drive the LCD display, provide you with a number of temperature ranges, provide alarm functions, go to sleep if after a certain time and a whole host of other features. But initially, you need to get the analogue voltage into a form that the microprocessor understands. This is where the ADC comes in.

The ADC

The part under the microscope is the TI TLC549. Why did we pick this one? Well it is a simple low speed ADC. It works the same as about 90% of the standard ADCs on the market and does exactly what we need to demonstrate how these components work. Really the only thing that distinguishes one ADC from another is resolution (how precise it is) and its speed (how many samples per second it can do).

Terminology

ADCs have a language of their own. Once mastered, you can appear like an ADC expert and confound and confuse even the experts and even if you are not too sure what you are talking about, you can at least sound like you do!

Resolution

This is the first thing to get your head around. Resolution is a measure of how many steps the ADC can digitise the signal to. This is similar to a ruler being able to measure in 1mm steps. Anything finer than 1mm cannot be measured. Resolution is specified in 'bits' and the TLC549 is an 8 bit ADC. This means it can digitise to 2^8 (or 256) steps. So a 5V input can be measured to an accuracy of 5/256 steps (or 19.53mV). A 10 bit ADC can measure to an accuracy of 2^{10} (1024) steps and a 12 bit ADC can measure to 4096 steps. With a 12 bit ADC, you can measure a 5V signal to an accuracy of 5/4096 steps or 1.22mV. Obviously if you have a very small signal, you need an ADC with a high resolution. 8 bit ADCs are about the lowest resolution you can get and ADCs can go up to over 24 bits (digitising to over 16 million steps).

Speed

The ADC takes a snapshot of the input voltage and converts it. How quickly it does this before being able to take the next snapshot is a measure of its speed.

Interface

This can either be serial or parallel. A parallel interface outputs each binary bit on a pin. With an 8 bit ADC, you have 8 output pins representing binary values from 2^7 (the *most significant bit, or MSB*) to 2^0 (the *least significant bit, or LSB*). These can either be connected to a microprocessor, LEDs, output port or another component/piece of test equipment. If all your

ELECTRONWORKS

outputs are logic 1, the ADC is outputting a code of 255 ($1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5$ etc...). If all your outputs are 0V, the ADC is outputting a code of 0 (or the part is broken!)

Later ADCs (in the last 15 years or so) have implemented a serial interface. Serial interfaces mean less pins and smaller, cheaper devices. If you are designing a small data logger, you do not want a massive data converter taking up loads of room. Serial devices are the way ahead and the TLC549 is a serial output ADC.

A serial interface has 3 pins (mostly). One pin is the chip select (CS). This turns on the device and tells the part to get ready to digitise the input. If you have a portable design that needs to run on very low power, you only want your ADC turned on when you need it. Some devices use the CS to start the data conversion; some only use it to turn on the device.

You then have the Data pin. This outputs the data (duh!), normally MSB first. Then there is the clock line. You take the clock line high (to 5V) and the first data bit appears on the data line. You take the clock line low then high again and the next bit appears. Repeat this 8 times and you have clocked out all your data and used 3 pins instead of 8. Big deal you may say... However, imagine if you are using a 12 bit ADC, you would use 3 pins instead of 12 and with a 24 bit ADC you use 3 pins instead of 24. The savings in package size start to get significant. Have a look at the software section at the end to see an example piece of code for clocking data out of the TLC549.

Internal/External Reference

To get its result, the ADC needs to compare the input voltage with a known stable voltage. This is the reference. Some ADCs have this built in (although they do not tend to be too accurate). Some ADCs allow you to connect an external reference, so you can pick what voltage you compare the input signal to, how stable the voltage is etc. More of this later...

Sample and Hold

The ADC needs a constant voltage to work on. If you were half way through a mathematical calculation and someone changed the numbers for you, you would be pretty annoyed. The ADC is no different. It makes sure it has a constant voltage by having a tiny capacitor inside which stores the input voltage for the ADC 'engine'. Between the capacitor and the input pin is a switch inside the chip that isolates the ADC engine from the outside world. When you tell the ADC to start converting, the switch opens isolating the input voltage from the ADC engine and the ADC works on the voltage on the capacitor. Simple!

How does an ADC work?

As with measuring anything, you have to measure with respect to something. Measure someone's height and you measure with respect to the ground they are standing on. Measure the length of a road and you need a start point. The same is true with an ADC. You need an accurate, stable voltage, called a reference voltage. The ADC looks at the reference, then looks at the input and compares the two. Before we explain this in more detail, we need a quick lesson in binary. If you understand binary you can skip the next section.

A binary number is made up of 1's and 0's each carrying a different *weight*. As with decimal, we go from 0 to 9 then add an extra digit to make 10. We then go from 10 to 99 then add a third digit to make 100. With decimal, each digit we add is 10^0 , or 10^1 , or 10^2

With binary we only have 1 and 0. Makes things tricky, but hey, that is life!

So we have 0 then 1 then we have to add an extra digit to get 10 (equals 2), then 11 (equals 3), then we add a third digit to get 100 (equals 4), then 101, 110 and 111 (equals 7). With binary, each digit we add is 2^0 , 2^1 , 2^2 , 2^3 etc.

So to convert a number from decimal to 8 bit binary, start with 2^7 (= 128) and see if this number is larger or smaller than the number we need to convert. If it is larger, put a 0 in that column and move to 2^6 . If 128 it is smaller, put a 1 in that column and subtract 2^7 (128) from

ELECTRONWORKS

your answer. Move on to 2^6 (= 64). Repeat with 2^6 but this time subtracting 2^6 (64) if our number is larger than 64.

Let's throw some examples at you to see if this clears the water...

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0 =	0	0	0	0	0	0	0	0
1 =	0	0	0	0	0	0	0	1
2 =	0	0	0	0	0	0	1	0
3 =	0	0	0	0	0	0	1	1

Now lets try 154 (a random number I picked).

Take 2^7 (=128). This is smaller than 154, so we put a 1 in the 2^7 column and subtract 128 from our result. My calculator tells me this leaves 26.

Move on to 2^6 (=64). This is higher than 26, so we put a 0 in the 2^6 column. Move on to 2^5 (= 32). This is higher than 26 so we put a 0 in the 2^5 column. Move on to 2^4 (=16). Hey! This is smaller than 26, so put a 1 in the 2^4 column, subtract 16 (=10) and move on. Repeat this until you end up with the following code:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	1	1	0	1	0

So 154 decimal is equal to 10011010.

To make life easy, the Calculator function in Windows does this for you.

If you need a number higher than 255, you have to increase your digits (9 bits takes you up to 511, 10 bits takes you up to 1023), just like the ADC *Resolution* description above.

With an ADC, we are comparing an input voltage with a reference voltage and converting the result into a number. With an 8 bit ADC, we are converting a voltage to a number between 0000 0000 and 1111 1111 (in binary). So 0V input gives an output of 0000 0000 and an input voltage equal to the reference voltage gives an output code of 1111 1111

Now here is an important bit to note: In decimal (base 10) if you want to divide something by 10 (the number of the 'base') you shift all the numbers right by one digit. So 100 shifted right gives 10. If you want to multiply something by 10 you shift all the numbers left by 10.

In binary (base 2), it is same, but shifting the numbers right or left by one digit divides or multiplies by 2 (the number of the 'base'). As an example, 100 (=4) shifted right by one digit gives 10 (=2). 100 (=4) shifted left by one digit gives 1000 (=8). So in binary, it is very easy to multiply and divide numbers by 2 (or multiples of 2).

Therefore with the ADC, if you input a voltage equal to the reference voltage, the output code will be 1111 1111 (=255). If you input a voltage equal to half the reference voltage, the output code will be 0111 1111 (=127) (the full scale code shifted right one bit). The smarter reader will notice that half of 255 is 127.5 and in fact if you input a voltage equal to half the reference voltage, the ADC will flicker between a code of 0111 1111 and 1000 0000 (=128).

OK, so back to the original question: How does an ADC work..?

Well it does exactly what we have done above. It looks at the input voltage and compares it to half the reference. Thus it can establish if the input voltage is in the upper or lower half of the voltage range – see FIG 1. If the voltage is higher than half the reference voltage (2.5V), it sets the Most Significant Bit (2^7) high. So in our table above the 2^7 column will have a 1 in it. We then know that the input voltage is in the top half of the input range (between 2.5 and

5V). We now need to split this range into half again and repeat the process to see if our input voltage is in the range (2.5 to 3.75V) or (3.75V to 5V). We repeat this until we have 'homed in' on our input voltage.

Let us assume our input voltage is 3.4V and we are using a 5V reference. The arrow represents our input voltage. The ADC starts by setting the comparison voltage to half the reference voltage (=2.5V) and compared this with the input voltage to see which half our input voltage lies in. Our input voltage is higher than this, so a 1 is set in the Most Significant Bit column.

We then need to determine if the input voltage is between 2.5V and 3.75V or 3.75V and 5V.

We set the reference voltage equal to 3.75V (half way) and do the comparison again. Our input voltage is lower than 3.75V, so we put a 0 in the next column (2^6). We know our input voltage is somewhere between 2.5V and 3.75V. We set the reference to 3.125V (half way) and do the comparison again. Our input is higher than 3.125, so we put a 1 in the next column, then set the reference voltage to the half way point and repeat the comparison.....

Actually, this varying comparison voltage is generated using a Digital to Analogue Converter, or DAC. This component converts a digital code into an analogue voltage (the opposite of an ADC). See our article on **Digital to Analogue Converters Explained** for a more detailed lowdown.

As you can see, by continuously setting the reference point to the half way point, we are making the comparison voltage 'home in' on the input voltage. Note that the above process is exactly the same as earlier where we compare our number with 2^7 then subtract 2^7 and compare with 2^6 , then subtract 2^6 etc

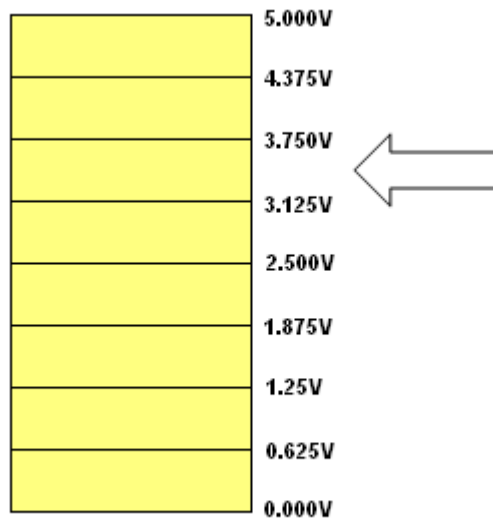


FIG 1

The ADC in FIG 1 has only 8 steps so is a 3 bit ADC ($2^3 = 8$). Our ADC is 8 bit so has 256 steps, so as you can imagine, the steps between each different voltage are a lot smaller (19.5mV) and FIG 1 would look a lot more crowded, so I hope you get the idea with my 8 step example.

In fact, it takes as many steps to reach the final output code as there are bits, so an 8 bit ADC takes 8 steps to reach the final result. This form of ADC produces good accuracy with a fast conversion time.



Software

Here is a snippet of code that will enable you to clock data out of the TLC549. Please note: on the first conversion, you will end up with garbage. You have to do 2 conversions to clock out valid data.

```
sample_adc
    movlw    0x08                ; load counter with 8
    movwf   count
    bcf     PORTA, cs           ; take cs low

loop1
    bcf     STATUS, C           ; precondition carry bit
    btfsc  PORTA, din          ; look at data line
    bsf     STATUS, C           ; set carry
    rlf    adc_data, f         ; rotate carry into LSB
    bsf     PORTA, sclk        ; set clock line high
    bcf     PORTA, sclk        ; set clock line low
    decfsz count, f
    goto   loop1
    bsf     PORTA, cs           ; take cs high
; we have now extracted a byte of data from the ADC
    retlw  0
```

If you buy any of the following kits, you can evaluate the above theory with a practical circuit:

Voltage to Binary Converter Kit

Demonstrates how an ADC converts an input voltage to either a serial or parallel output
<http://www.electronworks.co.uk/ck12.htm>

ADC Internals Demonstration Kit

Demonstrates the internal operation of an Analogue to Digital Converter
<http://www.electronworks.co.uk/ck13.htm>