CHALMERS
Department of Computer Science and Engineering

2006-11-07

Embedded systems
A/D- and D/A-converters

In this paper we will discuss analog-to-digital converters (a2d-converter,A/D-converters ADC) and digital-to-analog converters (d2a-converter, D/A-converter, DAC). Let us start by defining what we mean by an analog signal and a digital signal.

What is the difference between an analog and a digital signal?

The devices we will be discussing normally handle voltages although these voltages may represent, be converted from, some other unit for example fluid level or temperature. Let us use voltages from now on.

An analog signal can have *any* value meaning that the precision of the values are infinite. The presition is only limited by the number of digits we use in our representation and of our ability to measure the value detailed enough. Of course a value, a voltage, can not take on an infinitely high value so the values are normally restricted to a definition span, for example zero (0) to +10 Volts or a symmetrical span, for example from -10 Volts to +10 Volts. As we shall se later on the first example is called an *unipolar* signal while the other signal is *bipolar*.

A digital (discrete) signal on the other hand can only be represented by a limited, finite number of values. These values are normally evenly distributed so the step from one value to the next is always of equal height no matter what the values are.

If we now compare these two representations and have a value of say 1.5678 Volts this value can be exactly represented by a analog value although the value could have even more precision that got lost when we used only four decimals in our description but that is not a limitation in the value as such but in our representation of it. If we would like to represent the same value with a digital representation where the step between to values are 0.1 Volts this value would be described as 1.5 Volts or 1.6 Volts. There are two possibilities here depending on how the transformation from analog to digital value is performed. We will get 1.5 Volts if the analog value is *truncated* to the digital level, the extra decimals are chopped of, and we will get 1.6 Volts if the value is *rounded* to the nearest level. In the A/D-converters we will study later on we normally deal with truncation.

CHALMERS
Campus Lindholmen
Department of Computer Science and Enginnering
Sven Knutsson
Visiting address: Hörselgången 11
P.O.Box 8873
SE-402 72 Göteborg

Why convert between analog and digital values?

Now why do we have to convert between analog and digital values? Most of the things we measure and describe in our world are in some sense represented by analog values even if we often choose to limit the resolution and thereby restrict the number of values and thereby make it digital. For example we might measure a distance in meters although we could choose to go all the way down to micrometers or even further.
In the world of computers, which we are dealing with in this course, the situation is an other. Here we are restricted to digital values not by choice but by the properties of the system. Even if the signal span could be large and the number of steps big the resolution is always limited. This means that we have to convert a value from analog to digital if we want to move it from the analog world to our world of computers. In the same way we will often convert our values back to analog when we move the values out of the computer again and into the real world. We do this to get a smooth transition between signal levels.
Before we turn our attention to the converters as such we will have a look at how values could be represented in a computer. We will have a look at number representation.

Number representation

In a computer numbers can be represented in a number of ways but we will always have the limitation that the registers and memory positions we have in the computer consist of a limited number of digits. We have a given *word length*. The numbers we use in the computer can be described using different number bases. We shall look at the most common ones.

Number bases

Let us have a look at a normal integer value, for example the value 2346. This way do describe the number is a short form of the longer complete version

$$2346 = 2 \cdot 10^3 + 3 \cdot 10^2 + 4 \cdot 10^1 + 6 \cdot 10^0$$

That is the position in the number marks what power of ten the digit should be multiplied by. The same holds true for a number with decimals, for example

$$3.27 = 3 \cdot 10^0 + 2 \cdot 10^{-1} + 7 \cdot 10^{-2}$$

This means that each digit can take on ten different values (zero to nine). We call the digit with the highest power of 10 (the left most digit) the most significant digit (MSD). In the same way we call the digit with the lowest power of 10 (the right most digit) the least significant digit (LSD).
In a computer each position in our registers and memory cells could be thought of as an electronic switch that can have the position ON or OFF or alternatively described as 1 and 0. We call each position a *bit*. This means that we can not place our decimal numbers in these cells as they are. They have to be converted to numbers that can be described by 1:s and 0:s. We have to use a *binary number* base, the base is often called the *radix*. The bi-

nary radix is formed in a similar way to our decimal number base and we describe our numbers using powers of 2 instead of powers of ten and each binary digit can only take on the values one (1) and zero (0). Here we call the bit with the highest power of 2 (the left most bit) the most significant bit (MSB) and the bit with the lowest power of 2 (the right most bit) the least significant bit (LSB) Let´s have a look at an integer binary number and convert it to a decimal one.

$$(101101)_{binary} = (101101)_2 = \left(1\cdot 2^5 + 0\cdot 2^4 + 1\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0\right)_{10} = (45)_{10}$$

Thankfully we can use decimal numbers when we program our computers using a high level language like C. Actually C doesn´t support using binary numbers at all but it does support using hexadecimal numbers and, as we shall soon see, the relationship between binary and hexadecimal numbers is close.
Hexadecimal numbers use the number base 16 and because of that each hexadecimal digit can take on 16 different values. There are only ten decimal digits so we have to come up with six extra digits to complete our hexadecimal representation. These digits would in decimal be 10 to 15 but we prefer to use only one symbol per digit and we replace these numbers with the letters A to F.
We said that hexadecimal digits are closely related to binary bits and the reason for this is that $16 = 2^4$. This means that each groups of four binary digits can be replaced with one hexadecimal digit. For example

$$(01101101)_2 = (0110 \quad 1101)_2 = (6D)_{16} =$$

$$= \left(0\cdot 2^7 + 1\cdot 2^6 + 1\cdot 2^5 + 0\cdot 2^4 + 1\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0\right)_{10} = \left(6\cdot 16^1 + 13\cdot 16^0\right)_{10} = (109)_{10}$$

The separation into groups of four bits in the middle representation is only for convenience.
We will look at converting numbers with decimals to binary numbers later on.

Conversion from decimal to hexadecimal value

To convert a decimal value to hexadecimal representation we repeatedly do integer division of our decimal value with the new base 16 and keep the reminder from each division as the new digit. As we shall see in a couple of other examples this work with every base.
We will illustrate the process with an example and convert the decimal value 2346 to hexadecimal base

$$\frac{2346}{16} = 146\frac{10}{16} \quad \Rightarrow \quad (10)_{10} = (A)_{16}$$

$$\frac{146}{16} = 9\frac{2}{16} \quad \Rightarrow \quad 2_{10} = 2_{16}$$

$$\frac{9}{16} = 0\frac{9}{16} \quad \Rightarrow \quad 9_{10} = 9_{16}$$

We get the result

$$(2346)_{10} = (92A)_{16}$$

Conversion from decimal to octal value

The princip is the same as when we converted from decimal to hexadecimal value. We repeatedly do integer division of our decimal value with the new base 8 and keep the reminder from each division as the new digit. We will illustrate with an example using the same value as above and convert the value 2346 to octal base

$$\frac{2346}{8} = 293\frac{2}{8} \quad \Rightarrow \quad 2$$

$$\frac{293}{8} = 36\frac{5}{8} \quad \Rightarrow \quad 5$$

$$\frac{36}{8} = 4\frac{4}{8} \quad \Rightarrow \quad 4$$

$$\frac{4}{8} = 0\frac{4}{8} \quad \Rightarrow \quad 4$$

We get the result

$$(2346)_{10} = (4452)_{8}$$

Conversion from decimal to binary value

Once again we use the same method and repeatedly do integer division of our decimal value with the new base 2 and keep the reminder from each division as the new digit. We continue to convert the decimal value 2346 in our example

$$\frac{2346}{2} = 1173\frac{0}{2} \quad \Rightarrow \quad 0$$

$$\frac{1173}{2} = 586\frac{1}{2} \quad \Rightarrow \quad 1$$

$$\frac{586}{2} = 293\frac{0}{2} \quad \Rightarrow \quad 0$$

$$\frac{293}{2} = 146\frac{1}{2} \quad \Rightarrow \quad 1$$

$\frac{146}{2} = 73\frac{0}{2} \Rightarrow 0$ $\frac{73}{2} = 36\frac{1}{2} \Rightarrow 1$

$\frac{36}{2} = 18\frac{0}{2} \Rightarrow 0$ $\frac{18}{2} = 9\frac{0}{2} \Rightarrow 0$

$\frac{9}{2} = 4\frac{1}{2} \Rightarrow 1$ $\frac{4}{2} = 2\frac{0}{2} \Rightarrow 0$

$\frac{2}{2} = 1\frac{0}{2} \Rightarrow 0$ $\frac{1}{2} = 0\frac{1}{2} \Rightarrow 1$

We get the result

$(2346)_{10} = (1001\ \ 0010\ \ 1010)_2$

Word length

We said that our computers internally use binary representation to store numbers. In most cases the number of positions in these registers, the register length, the *word length*, is 8, 16, 32 or 64 bits. We use to refer to 8 bits as one *byte*, 16 bits as one *(short) word*, 32 bits as one *long word* or *double word* and 64 bits as one *long long word*.
So far we have presented a number of ways to describe numbers and the basic structure of the registers in a computer but we need more information to find out how the numbers are placed in the registers. For example if we want to place an integers value in a register of byte size in what position do we place the digit with the lowest power of two if we don´t fill all bits of the register? The answer is that it could depend on the application but in most cases we use right alignment which means that this digit is placed in the right most position in the byte and the empty digits in the left most part of the word will be filled with zeros (0). A trickier question is when we have numbers with decimals, or binals as they are called when we use binary representation, between which bits do we place the binary point? And what about negative numbers?
We need to look a little closer at how can we represent a number.

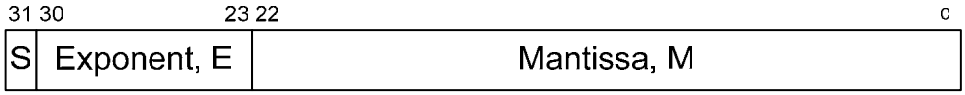
Representation of numbers

We can use a number of different criteria to divide our numbers into groups and these criteria does at some points overlap so the description is not all that clear.
To start with we can look at our representations as divided into two main groups, numbers represented in floating point form and numbers represented in fixed point form.

Floating point

A *floating point* number is described in the form $mantrissa \cdot 2^{exponent}$ (or $mantrissa \cdot 10^{exponent}$) and this means that the number of bits that we can use to represent

the numbers in the computer are divided into two parts, the mantissa and the exponent. With this representation we can describe a large span of values, observe that both the mantissa and the exponent in most cases can be both positive and negative and that both numbers as such have fixed point representation. The mantissa is normally a number in the interval 0.5 to 1.0 while the exponent is a positive or negative integer. Observe that the resolution of the value described by the mantissa is dependent of the exponent.
As an example of this representation we will give the standard for 32 bit floating point values as given by the US American organization IEEE (The Institute of Electrical and Electronic Engineers) in their standard *IEEE754*.
Here the 32 bits are divided into one sign bit (S), 8 bit exponent (E) and 23 bit mantissa (M).



Figur 1 IEE754

The actual value given by the representation is

$$N = (-1)^S \cdot 2^{E-127} \cdot (1.M)$$

We can se that we have a positive value if the sign bit S is zero and a negative value if it is one.
E is a positive integer restricted by $0 < E < 255$, observe the inequalities and the subtraction of 127 to get the true exponent. This means that our actual exponent is in the interval -126 to 127.
The value M represents the binals in our mantissa while we have a hidden integer one in front of this number. A number describing only binals is called a fractional number. We will get back to this in a while.
This means that the smallest value we can describe with IEEE754 is $2^{-126} \approx 1{,}8 \cdot 10^{-38}$ while the biggest value is $2^{127} \cdot (2 - 2^{-23}) \approx 3.4 \cdot 10^{38}$. Observe that the span of values is very big and that the smallest value is very small so the resolution is quit good.
Floating point representation is not really suited for our A/D- and D/A-converters so we leave them for now.

Fixed point

In *fixed point* representation we only have a value (mantissa) but no exponent (or the exponent is zero if you like). This means that our numbers have a value span limited by the smallest and the largest number that can be represented with the given number of bits, we have no exponent that can slide the value on the value axis.

Integer numbers

As the name implies the binal point is in a fixed position somewhere between two bits in our number. One frequent placement is to the right of the right most bit, which means that we have no binals and the value is an *integer*.

Fractional numbers

We get an other popular representation if we place the binal point to the left of the left most digit in our number which means that all bits are binals and we have a so called *fractional* number described as 0.N where N are the number of bits in the word. It is worth noticing that with this representation we can never reach the value one (1), our valus are always smaller than one (1) even if we are very close to one (1) if all bits are ones. We can realize that this representation can only give positive values.

With a slight modification we can also represent negative values. We introduce a sign bit and let our number with N bits is divided into one sign bit (the left most bit) and $N-1$ binals.

The fractional representation, both with and without sign bit, is very well suited for the converters we will discuss later on.

Of course there are also a number of other fixed point formats where the binal point is somewhere in the middle of the word but these representations are quit rare.

Now what about negative numbers?

Negative numbers

We said earlier that with fractional numbers we could describe *negative numbers* as well as positive numbers if we introduced a sign bit in our representation. We didn´t mention it but this holds true for integer values as well, that is our binary word with N bits could consist of one sign bit and N-1 bits describing an integer or a fractional value.

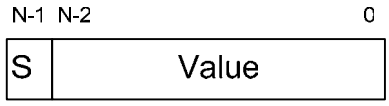


Figure 2 Number with sign bit

In most cases we don´t use a simple sign bit but we transform our negative values to what is called *two-complement* representation (*2-complement*). One might ask why? One of the reasons is that we can simplify our electronic circuits some what. In most of our microprocessor applications we need to add and subtract numbers and this is done with the help of electronic circuits that perform the arithmetic operations. To do both addition and subtraction we need two electronic circuits or one circuit that can perform both operations. If we represent our negative value in two complement form we can do away with the subtracter. Instead of subtracting the two numbers we start by converting the second value in the subtraction to a negative one by changing it to its two complement, which is a very simple operation, and then add the two values. That is

$$value_1 - value_2 = value_1 + \left(-value_2\right)$$

This is not as easily done if we only use a simple sign bit.

Now how do we take the two complement of a word, that is how do we make it negative. First we invert all the bits, that is if the bit is a zero (0) we make it a one (1) and if the bit is a one (1) we change it to a zero (0). Then we add one (1) to the LSB and we are done. An example let us take the two complement of the decimal number 26 and let´s treat the binary number as a byte, that is the word length is eight bits

$$\left(26\right)_{10} = \left(00011010\right)_2 \rightarrow \textit{invert all bits} \rightarrow \left(11100101\right)_2 \rightarrow \textit{add one to LSB} \rightarrow$$

$$\rightarrow \quad 11100101$$
$$\underline{+ \qquad 1}$$
$$11100110$$

Observe that if the value is negative then the MSB is always one.
Let us make sure that we can use this for subtraction. Let us subtract 26 from 38

$$(38)_{10} - (26)_{10} = (38)_{10} + (-26)_{10} = (00100110)_2 - (00011010)_2 =$$

$$(00100110)_2 + (11100110)_2 = \begin{array}{r} 00100110 \\ \underline{+\ 11100110} \\ 00001100 \end{array} = (12)_{10}$$

We get the expected result if we ignore the memory bit we get out of the addition.
Now let us move on to our main subject ADC:s and DAC:s.

General A/D- and D/A-qualities

Resolution

If we look at an analog-to-digital converter it is supposed to convert a analog value that can take on any value within it´s definition span into its corresponding digital value. Since the digital value have a limited number of bits, n, then the number of digital values we can represent is also limited

$$N = 2^n$$

For example if we have the word length byte (8 bits) then the number of possible digital values is $2^8 = 256$ and in most cases these values are evenly distributed within the definition range of the A/D-converter. This means that if the A/D-converter can handle voltages in the range zero (0) to $+U_{max}$ then the step in Volts between two consecutive digital values is

$$\Delta = \frac{U_{max}}{N} = \frac{U_{max}}{2^n}$$

We call Δ the *resolution* of our converter. In some cases we talk about the resolution as a fraction of the maximal value no matter what this is, that is we express the resolution as

$$\frac{1}{N} = \frac{1}{2^n}$$

Which is a number without unit.

Unipolar and bipolar converters

In the discussion above the converter only handled positive voltages. We call such a converter a *unipolar* converter. These converters are suitable when the property we want to convert from analog to digital always is positive. For example we might want to convert the measurement of the level in a tank and that level can hardly be negative.

In other cases we would like to handle both positive and negative voltages, in most cases with a equal span, that is we would like to handle voltages from $-U_{max}$ to $+U_{max}$ We call such a converter a *bipolar* converter. This is for example normaly the situation when we convert a sound signal to digital.

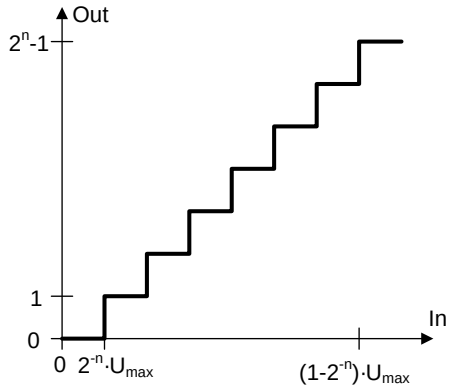


Figure 3 Unipolar converter

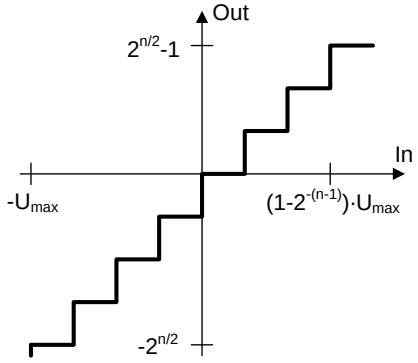


Figure 4 Bipolar converter

Conversion error, SQNR

Since our digital value has a limited number of values while the analog voltage can take on any value the conversion from analog to digital value will in most cases give an error. The analog value is not exactly at any of the discrete digital levels. We can realize that the largest error will be half the resolution, $\Delta/2$.

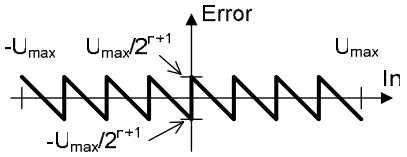


Figure 5 Rounded error signal

This error is often given as the *SQNR*, the *Signal to Quantization Noise Ratio* where we compare the maximal level of the converter with the error and describe this using logarithmic values in decibel. For the unipolar converter we have

$$SQNR_{unipolar} = 20 \cdot {}^{10}log \frac{maximal\ level}{maximal\ error\ level} = 20 \cdot {}^{10}log \frac{U_{max}}{\frac{\Delta}{2}} = 20 \cdot {}^{10}log \frac{U_{max}}{\frac{U_{max}}{2 \cdot N}} =$$

$$= 20 \cdot {}^{10}log \frac{U_{max}}{\frac{U_{max}}{2 \cdot 2^n}} = 20 \cdot {}^{10}log\left(2^{n+1}\right) = 20 \cdot (n+1) \cdot {}^{10}log(2) = 6.02 \cdot (n+1) \approx 6 \cdot (n+1)$$

For the bipolar converter we don´t compare the error with the maximum span of the converter but with the maximal amplitude, that is half the maximal span. Many converters can use an offset voltage to change the converter from unipolar to bipolar. In both cases the maximal span is the same meaning that we either have the span zero (0) to $+U_{max}$ for the unipolar converter or $-\frac{U_{max}}{2}$ to $+\frac{U_{max}}{2}$ for the bipolar converter

For the bipolar converter we get the SQNR

$$SQNR_{bipolar} = 20 \cdot {}^{10}log \frac{maximal\ amplitude}{maximal\ error\ level} = 20 \cdot {}^{10}log \frac{\frac{U_{max}}{2}}{\frac{\Delta}{2}} = 20 \cdot {}^{10}log \frac{\frac{U_{max}}{2}}{\frac{U_{max}}{2 \cdot N}} =$$

$$= 20 \cdot {}^{10}log \frac{\frac{U_{max}}{2}}{\frac{U_{max}}{2 \cdot 2^n}} = 20 \cdot {}^{10}log\left(2^n\right) = 20 \cdot n \cdot {}^{10}log(2) = 6.02 \cdot n \approx 6 \cdot n$$

We can see that the unipolar converter has a 6 dB higher SQNR than the bipolar ADC but this is fictive. The two calculations are not done in the same way.

Number of bits	8	12	16	20	24
SQNR [dB]	48	72	96	120	144

Table 1 SQNR for some commonly used ADC:s

If we compare the SQNR for some commonly used bipolar ADC:s we get *Table 1*.

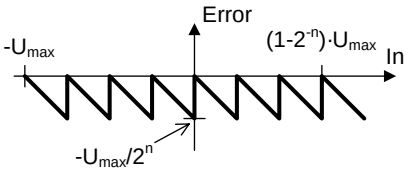


Figure 6 Truncated error signal

The assumptions used above are in most cases not really true. We assumed that the maximal error was half the resolution but for this to be true the analog value will have to be rounded to the nearest digital level in the conversion but in most cases the value is rounded to the nearest digital value lower than the analog value, that is it is truncated. This means that the maximal error is actually the same as the resolution of the converter and as a result our SQNR calculations will give 6 dB lower values.

Logarithmic converters, companding

The converters we have seen so far have a linear conversion span, that is the resolution is the same for all levels. This means that the conversion gives better relative resolution for large values than it does for small values. This can sometimes be a problem and we would like to have a converter with a more constant relative resolution. With this we mean a converter where the resolution compared to the present input value is constant and not the resolution compared to the total voltage span of the converter. Such a converter must have a logarithmic transfer function and if we want to make calculations on the converted digital values we have to start by expanding the values back to linear ones. We call the converter with logarithmic transfer function a *companding* ADC. Companding is a abbreviation of compression- expansion.

Now let us turn our attention to how to conctruct a converter. Since many ADC:s contain a DAC we will start with DAC:s.

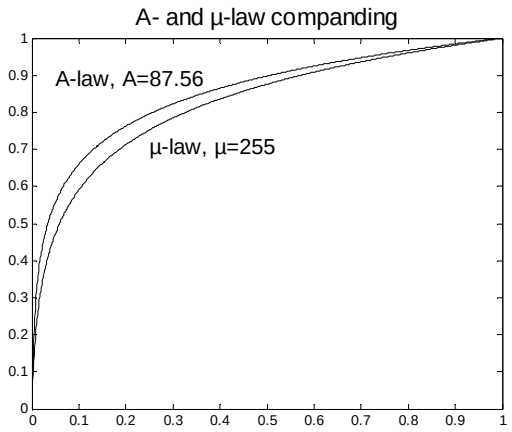


Figure 7 A- and µ-law companding

Operational amplifiers

To understand how D/A- and A/D-converters work we need to use *amplifiers* as building blocks. We will use integrated amplifiers, so called *operational amplifiers* (*OP*) or at least see our amplifiers as blocks and not concern ourselves with their internal construction.

The OP, *Figure 7*, is an integrated circuit that amplifies the differential signal supplied between the positive and the negative input terminal and delivers this signal to an output. The input impedance of the OP is very big (megaohms) which means that only a negligible current will flow in and out of the input terminals. The output impedance is at the same time very low (ohms or lower) and this means that most of the output voltage will be supplied to the following circuit. There will be no voltage division at the output.

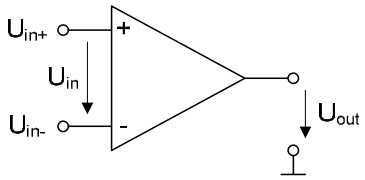


Figure 8 Operational amplifier

The amplification is very big (1 million times or more). These specifications make the OP very close to an ideal amplifier. The one specification that in many cases is not ideal is the demand for a unlimited bandwidth, meaning that it should give the same amplification no matter the frequency of the amplified signal. In reality there is a upper frequency limit that is far from infinite. In most cases the upper frequency limit f_g is inversingly proportional to the amplification A of the OP circuit

$$f_g \cdot A = constant$$

In our following studies we need to understand how a number of amplifier configurations work so we will describe these. The amplifier is in most cases an integrated circuit that we adopt to our needs using external components, in these cases these are resistors and in one case complemented by a capacitor. In the circuit diagrams we have hidden the fact that the circuits need to be supplied by a voltage from a power supply.

Comparator

The most simple configuration is the *comparator* where we compare the voltage supplied to the positive input terminal, U_{in+}, with the voltage supplied to the negative input terminal, U_{in-}, *Figure 9*.

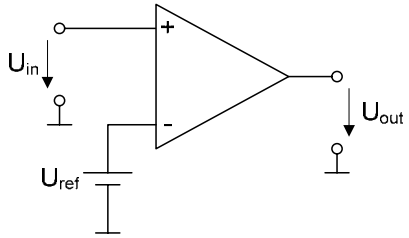


Figure 9 Comparator with reference

If the input signal to the amplifier, the difference $U_{in} = U_{in+} - U_{in-}$ is positive then the output signal will be positive but since the amplification of the OP is very big the amplifier will saturate and the output voltage will not be the amplification times the input signal but it will saturate the positive supply voltage to the circuit. If the differential input signal is negative then the amplifier will saturate at the negative supply voltage. This means that a input signal U_{in} larger than the reference voltage U_{ref} will give positive saturation while a smaller input voltage will give negative saturation.

This circuit can of couse also be used not to compare an input voltage to a given reference voltage but to compare two input voltages, *Figure 10*.

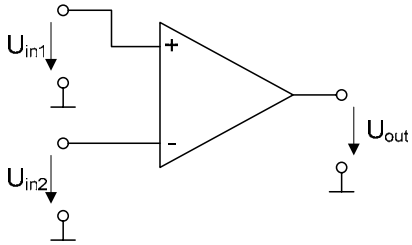


Figure 10 Comparator with two inputs

Inverting amplifier

The next circuit is what is called an *inverting amplifier*. Inverting means that a positive input signal will give a negative output signal and a negative input signal will give a positive output signal. We can see a typical schematic in *Figure 11*.

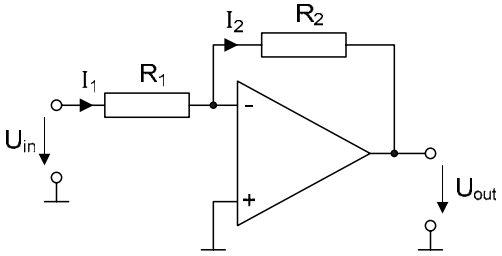


Figure 11 Inverting amplifier

For a simplified analys we can assume that the amplifier has infinite input resistans, that is no current will flow through the input terminals and this means that the voltage difference between the two input terminals is zero and since one of the terminals is grounded both terminals will have a potential of zero, 0 Volts. As a result the currents through the two resistors R_1 and R_2 are the same. We get

$$I_1 = I_2 \Rightarrow \frac{U_{in} - 0}{R_1} = \frac{0 - U_{out}}{R_2} \Rightarrow U_{out} = -\frac{R_2}{R_1} \cdot U_{in}$$

Summing amplifier

For our applications we will need to add more inputs to the inverting amplifier to get a *summing amplifier*. Let´s add a second input, *Figure 12*.

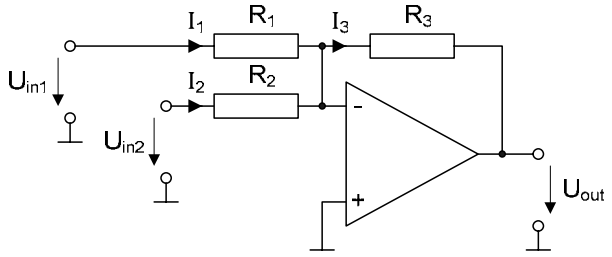


Figure 12 Summing amplifier

This time we get

$$I_1 + I_2 = I_3 \Rightarrow \frac{U_{in,1} - 0}{R_1} + \frac{U_{in,2} - 0}{R_2} = \frac{0 - U_{out}}{R_3} \Rightarrow U_{out} = -\frac{R_3}{R_1} \cdot U_{in,1} - \frac{R_3}{R_2} \cdot U_{in,2}$$

And we can draw the conclusion that we can sum the contributions from the inputs with their weighting factors $R_{out}/R_{in,x}$ and don´t forget the negative signs, the inversion.

Non-inverting amplifier

In the *non-inverting amplifier* a positive input signal will give a positive output signal, *Figure 13*.

Once again, since no current will flow through the input terminals then the two terminals will be at the same potential, that is $U_{in-} = U_{in+} = U_{in}$. Through voltage division we get

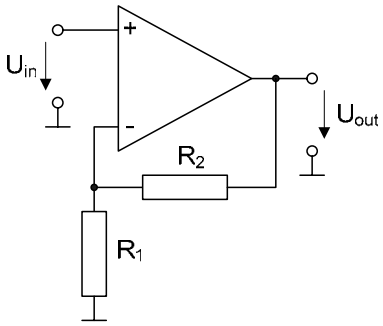


Figure 13 Non-inverting amplifier

$$U_{in-} = \frac{R_1}{R_1 + R_2} \cdot U_{out}$$

And if we rewrite the expression we will get

$$U_{out} = \frac{R_1 + R_2}{R_1} = \left(1 + \frac{R_2}{R_1}\right) \cdot U_{in}$$

We can see that the amplification is *always* larger than one (1).

Buffer

As a special case of the non-inverting amplifier we get a *buffer*, *Figure 14*. We get this from the non-inverting amplifier by letting the value of R_1 go towards infinity, that is be a break, and by letting the value of R_2 be zero (0), that is a short circuit. From the equation above we see that the buffer will have a amplification of one (1) and might seem meaningless but we do have a circuit with a very high input resistance and a very low output resistance which means that the circuit connected to the output of the buffer will not draw any current from the circuit connected to the input of the buffer and the high input recistance and the low output recistance also means that there will be no voltage division at neither the input nor the output, *Figure 15*.
The symbol for the buffer is often simplified according to *Figure 16*.

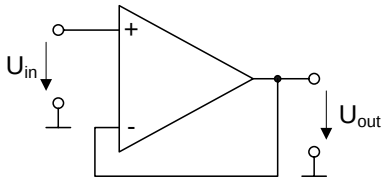


Figure 14 Buffer

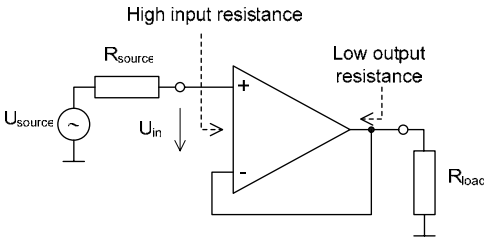


Figure 15 Buffer circuit

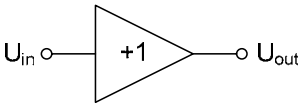


Figure 16 Simplified buffer symbol

Integrating amplifier

An *integrating amplifier* will integrate the input voltage over time, *Figure 17*.
If the input voltage U_{in} is constant then

$$U_{out} = -\frac{U_{in}}{R \cdot C} \cdot t$$

where t is the time, that is the output voltage is a negative going ramp, *Figure 18*.
Of couse the output voltage will after some time reach the negative supply voltage and stop there.
We can see that the slope of the ramp is proportional to the input voltage U_{in} which is something we will use later on.

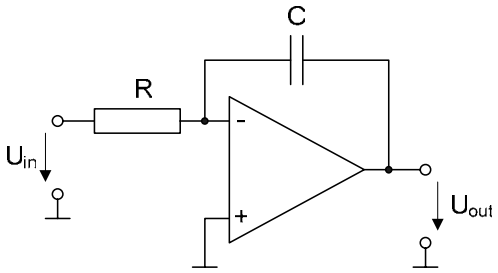


Figure 17 Integrating OP-circuit

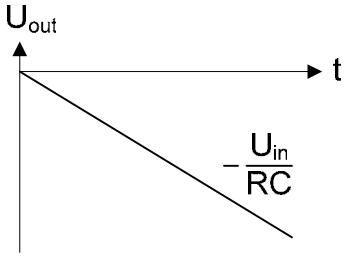


Figure 18 Output ramp

D/A-converters

Since some of the commonly used A/D-converters use a D/A-converter as a building block we will start by looking at D/A-converters.

Resistance ladder converter

The most direct way to create a digital-to-analog converter would be to use a set of resistors in a voltage divider configuration to create the voltages we need and a set of digitally controlled switches to select the voltage we want. Since the D/A-converter should have a fixed resolution, Δ, that is each voltage step should have the same height, then all the resistors in the ladder should be of equal value. The reference voltage, U_{ref}, will determine the voltage span of the converter. The digitally controlled switches form a net that will connect one out of N possible inputs to the output, this is called a *multiplexer* (*MUX*). Which of the inputs that is connected to the output is governed by the digital control word of the MUX. The number of bits in this word, n, depends on the number of inputs since $N = 2^n$. For the voltage divider to function properly the circuit connected to it should not draw any current from the net so we need to insert a buffer. In *Figure 19* we see a converter with a two bit control word (D_1 D0), that is we can have four inputs and need four different, equally spaced voltage levels.
Since the number of resistors and buffers will increase exponentially with the number of bits and the multiplexer will be more and more complex this type of converter is rarely used.

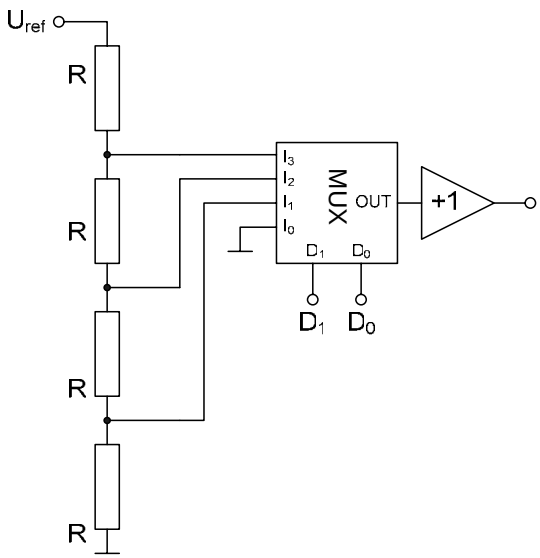


Figure 19 Two bit D/A-converter using a resistance ladder circuit

Current summing D/A-converter

Now let us have a look at the dominant type of D/A-converter.
We will use an inverting amplifier to build a DAC. Let us limit ourself to a unipolar converter and only use four bits in the digital word to simplify things.
If we consider the digital word we are about to convert into an analog voltage we can realize that a one (1) in LSB should give a output voltage of the same magnitude as the resolution, Δ, of the converter. A one (1) in the next bit should give twice the voltage, that is $2 \cdot \Delta$. The next (third) bit should double this, that is give a voltage of $4 \cdot \Delta$. And so on.
Let´s build the converter step by step and start with the LSB. We need to be able to supply a input voltage to the amplifier when LSB is one and take it away when LSB is zero. We do this by using an electronically controlled switch. Let´s use a input voltage U_{in} of the same magnitude as the maximal output voltage of the converter U_{max} with the difference

that the input voltage need to have a negative sign to give a positive output voltage since we use an inverting summation amplifier. We call this voltage $-U_{ref}$. For a one bit converter the LSB, the only bit, should generate an output voltage of

$$U_{out,LSB} = \frac{U_{max}}{2^1} = \frac{U_{max}}{2}$$

That is the amplification of our circuit should be $-\frac{1}{2}$ and if we refer back to *Figure 11* this means that $R_1 = 2 \cdot R_2$ and we get *Figure 20.*

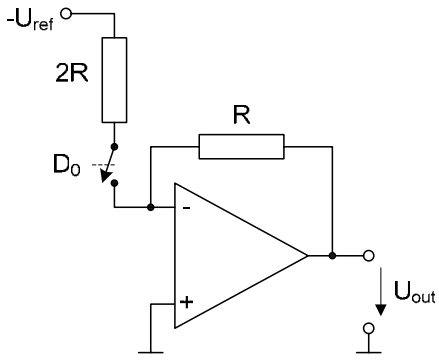


Figure 20 One bit DAC

In the same way we add another link for the next bit and get a two bit converter. Here the LSB should give a output voltage of

$$U_{out,LSB} = \frac{U_{max}}{2^2} = \frac{U_{max}}{4}$$

And the next bit twice that voltage. That is the amplification for the LSB should be $-\frac{1}{4}$ and $-\frac{1}{2}$ for the next bit. As a result we get *Figure 21*.

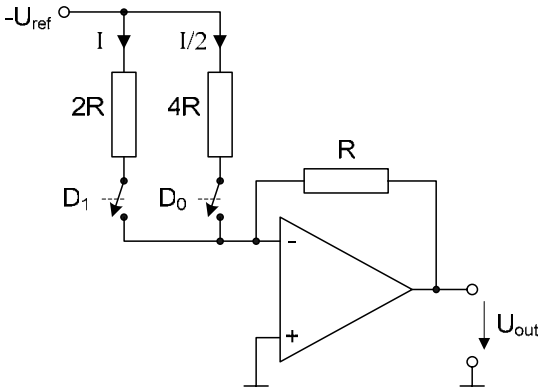


Figure 21 Two bit DAC

If we expand the circuit up to four bits we get the complete schematic in *Figure 22*.

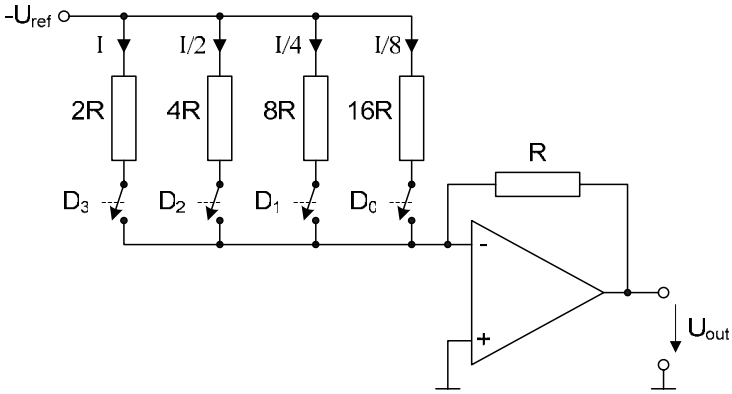


Figure 22 Four bit DAC

We can realize that we could change the output span of the converter by changing the value of the feedback resistor or by changing U_{ref} but the latter is not as common.

This is a basic schematic that won´t work that well if we increase the number of bits. If we have 16 bits the largest input resistor would have to have a resistance that is 32768 times the smallest resistor value. In most cases the converter is not a discrete schematic but it is made as an integrated circuit and it is quit hard to integrate resistors with so big differences in values and the way the values of the resistors change with temperature will not be the same for different values. We will therefore reconstruct our schematic with resistors of similar values.

R-2R ladder DAC

Let´s go back to the one bit converter and study *Figure 23*.
Let us calculate the current floating into the feedback resistor R_f. The current comes from the reference voltage source and are split into to resistors of equal value. Observe that the other end of the resistors are connected either to ground or to the negative input of the operational amplifier and this input is at the same potential as the positive input that is at ground level. We call the zero level at the negative input a *virtual ground*. The same current flows in both resistors and the total current is

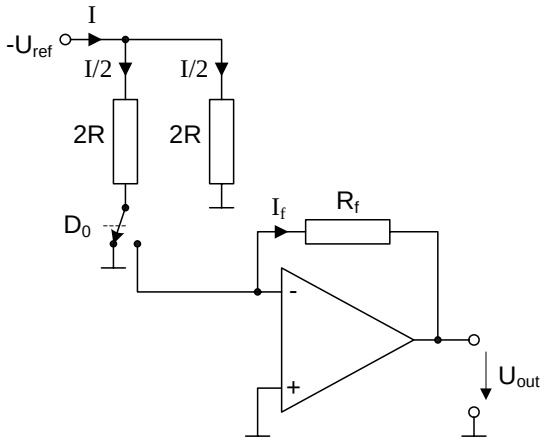


Figure 23 One bit R-2R ladder DAC

$$I = \frac{U_{ref}}{2 \cdot R} + \frac{U_{ref}}{2 \cdot R} = \frac{U_{ref}}{R}$$

but only one of the two currents is passed on to R_f to give

$$I_f = \frac{I}{2} = \frac{U_{ref}}{2 \cdot R}$$

This is as expected for a one bit converter. A one (1) in the input word (of one bit) should give a output voltage equal to the resolution

$$\Delta = I_f \cdot R_f = \frac{U_{max}}{2^1} = \frac{U_{max}}{2}$$

So far we can not see the meaning of the second resistor but let us add another bit to the digital word and add to the schematic to get *Figure 24*.
If we do some calculations to simplify the resistor net we shall see that the total current from U_{ref} still is

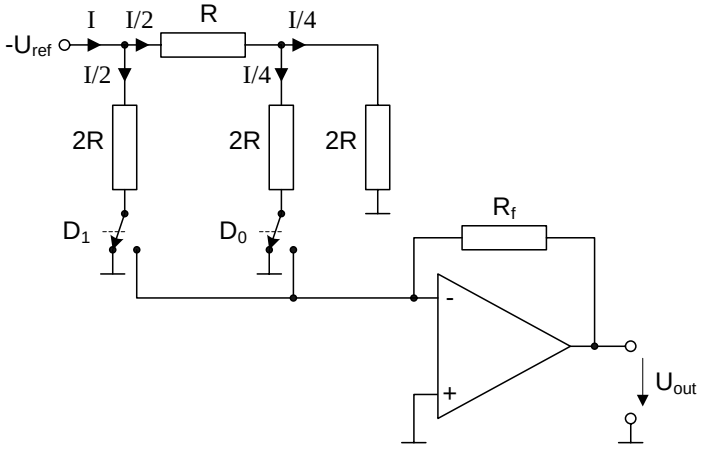


Figure 24 Two bit R-2R ladder DAC

$$I = \frac{U_{ref}}{R}$$

and a one (1) in MSB will give a output voltage of $\frac{U_{max}}{2}$ which is as expected.

We can see that the path leading from U_{ref} to MSB and the path leading to LSB carry the same current, $\frac{I}{2}$, which is half the total current. The path leading to LSB is then split into two paths carrying the same current, $\frac{I}{4}$. This means that the current through the MSB switch is twice as big as the current flowing through the LSB switch and the result is that the MSB path will give twice the output voltage of LSB as expected.
Now let us upscale the schematic to four bits and leave the analysis to private studies, *Figure 25*.

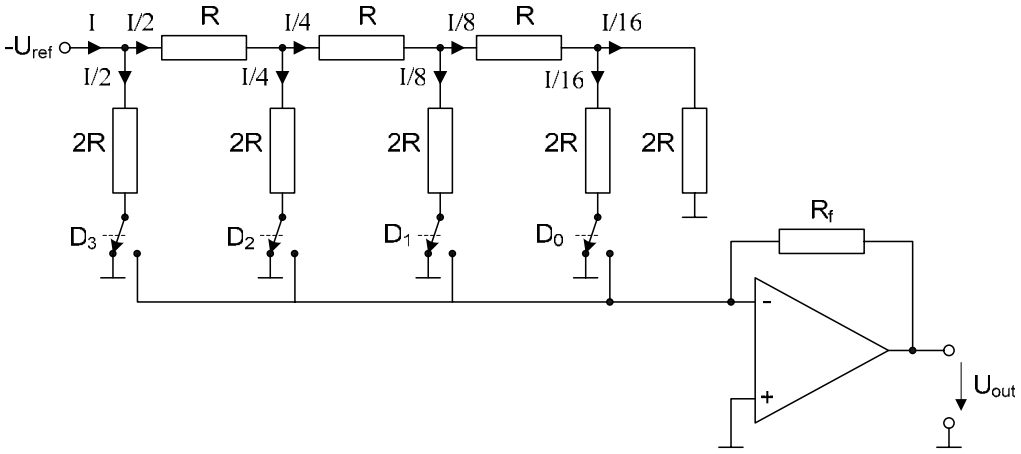


Figure 25 Four bit R-2R ladder DAC

As a conclusion we can see that the schematic only uses two different resistance values in the input net and this makes the circuit much easier to integrate while the changes with temperature will me more even. The resistance net is called a *R-2R ladder net*.

Smothing filter

If we look back we might remember that an analog voltage should be able to take any value within it´s definition span. This is not quit true for the output voltage from our DAC:s. It can only take on a number of discrete values since the digital word we are converting into an analog one only can give a discrete number of possible values $N = 2^n$ governed by the number of bits in the word. This means that if we update the output from the DAC on a regular basis we will get a series of values that jump between discrete voltage levels, *Figure 26*. To smooth this out we insert a low pass filter after the DAC.

A low pass filter (LP) is a filter that stops the higher frequency contents in the signal but lets the lower frequencies pass. The continuous update of the output voltage will generate a signal with the same frequency as the frequency of the update and overtones to this, that is signals of frequencies that are integer multipliers of the update frequency. The low pass filter is supposed to remove these signals of higher frequencies.

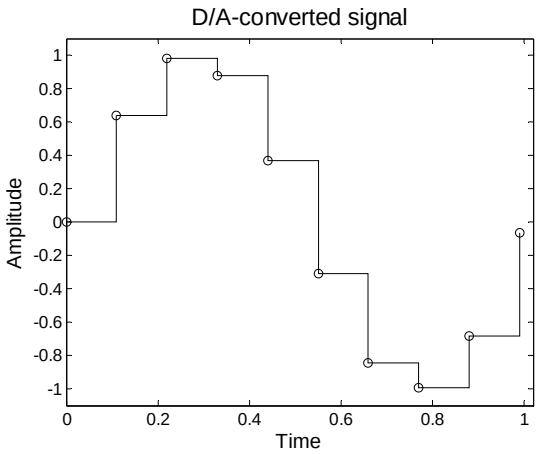


Figure 26 Output from DAC

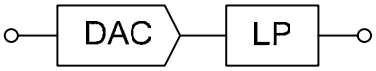


Figure 27 DAC and LP filter

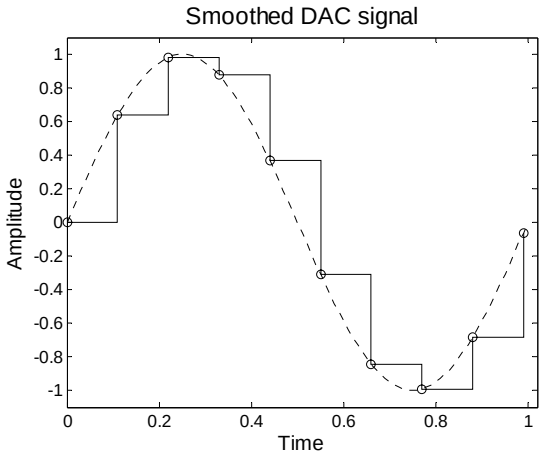


Figure 28 Unfiltered and filtered output from DAC

A/D-converters

Flash converter

The A/D-converter should convert an analog voltage to it´s corresponding digital value. The most direct approach here is to compare the analog input voltage with voltages corresponding to the discrete levels of the digital signal, *Figure 29*. We will use the same type of resistor ladder that we used in the resistor ladder D/A-converter and use comparators to do the comparison with the analog input signal.

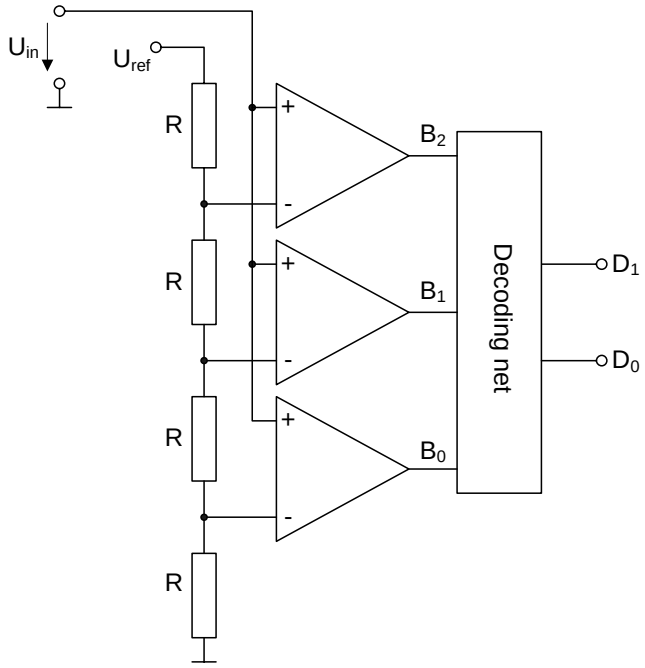


Figure 29 ADC with resistance ladder

The resulting output signal will not be the digital signal we are looking for. It will be a kind of thermometer scale where all comparators with a lower reference voltage than the analog input voltage will be positively saturated while the comparators with reference voltages higher than the input signal voltage will be negatively saturated. We can use a logical decoding net to convert this set of bits into the wanted digital word, *Table 2*.

Interval	*Bits from comparators*	*Binary word*
$0 - \frac{U_{ref}}{4}$	*000*	*00*
$\frac{U_{ref}}{4} - \frac{U_{ref}}{2}$	*001*	*01*
$\frac{U_{ref}}{2} - \frac{3 \cdot U_{ref}}{4}$	*011*	*10*
$\frac{3 \cdot U_{ref}}{4} - U_{ref}$	*111*	*11*

Table 2 Bits from comparators and binary words

This converter will use a number of parallel comparisons to do the conversion and the operation is very fast and this is the fastest type of ADC. We call it a *flash converter*. The circuit have the same draw backs as the resistance ladder DAC, the number of resistors and comparators will be big if we want many bits in the digital word. We get a complicated circuit and the number of elements to trim to get a precise result are large.

Counting ADC

We can get a simplier circuit if we use a D/A-converter as the center block of our A/D-converter. In the simpliest form this ADC use a binary counter, a counter that counts through all our binary values from zero to the highest value, to form the binary values that we use as the input words that the DAC converts into analog values. The counter is controlled by a clock that generates the counting pulse.

We compare the voltage generated by the DAC with the analog input voltage and stop the count when the output voltage from the DAC have reached the same level as the analog input voltage. This way we have found the digital value corresponding to our analog input voltage. We call the converter an *up-counting ADC*, *Figure 30* and *Figure 31*.

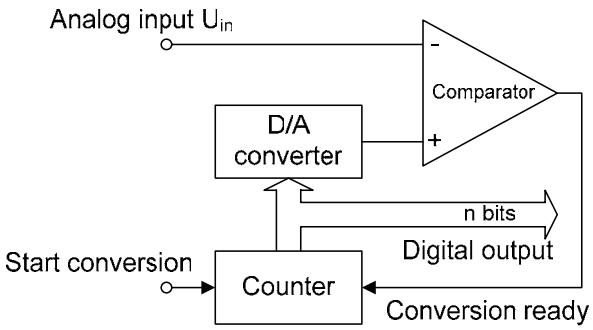


Figure 30 Up-counting ADC

The draw back of this circuit is that the time to find the digital output value is highly dependent of the level of the input voltage. If the voltage is low then the counter doesn´t have to generate many pulses to reach the correct digital value but if the input voltage is high then we will need many pulses. This have two negative effects. First of all the conversion time will be long (many clock pulses) for a high input voltage. Secondly the conversion time is very dependent on the voltage level and we have no way of knowing how long time each conversion will take, something that it often is an advantage to know.

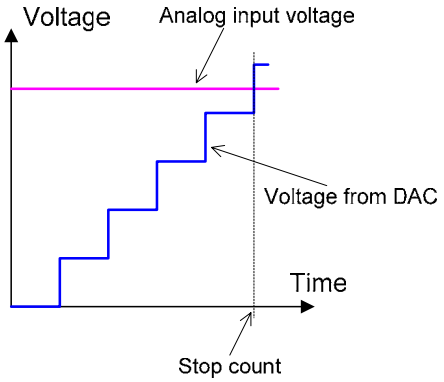


Figure 31 Up-counting ADC, time diagram

ADC using successive approximation

We need to improve on the up-counting ADC in two ways. We need to shorten the longest conversion time and we need to make the conversion time independent of the level of the analog input voltage. To do this we will replace the counter that controls the DAC with another digital net that tests bit by bit of the digital word, *Figure 32*.

We start by testing MSB in the digital world. That is we apply a digital

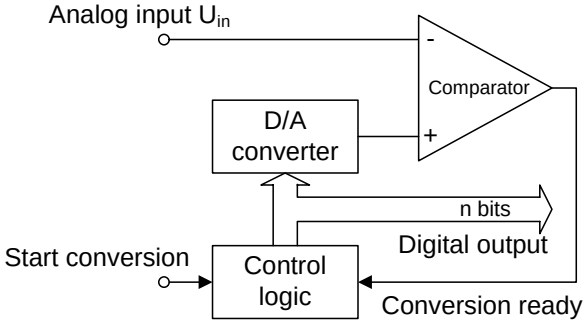


Figure 32 ADC using successive approximation

word to the DAC with a one (1) in MSB while the rest of the bits are zero (0) and compare the resulting voltage to the analog input voltage. This means that we test if the analog input voltage lies in the upper or lower half of the DAC:s voltage span. If we are in the upper half we keep the one (1) in MSB when we move on to the next bit while we reset MSB to zero (0) if we are in the lower half, *Figure 33*.
After this we continue with the next bit and test this bit with a one (1) while setting MSB to the level decided in the first step, *Figure 34*.
This way we continue bit by bit till we have passed LSB and are finished. This means that our conversion will always consist of *n* comparisons, where *n* are the number of bits in the digital word, and the conversion time is not dependent on the level of the analog input voltage. We call the conversion process *successive approximation*.
Figures 33 - 37 illustrate the flow for a three bit word where the resulting binary word is `1011`.

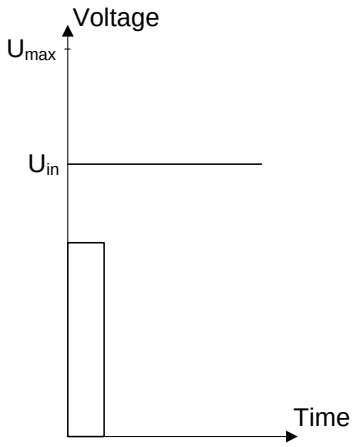


Figure 33 MSB of conversion

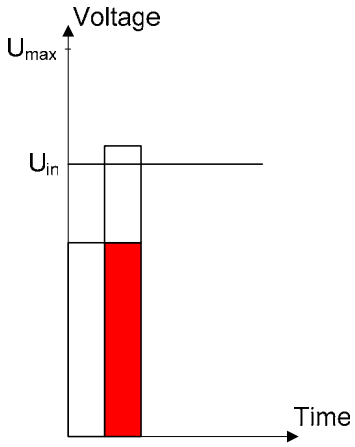


Figure 34 Bit 2 in conversion

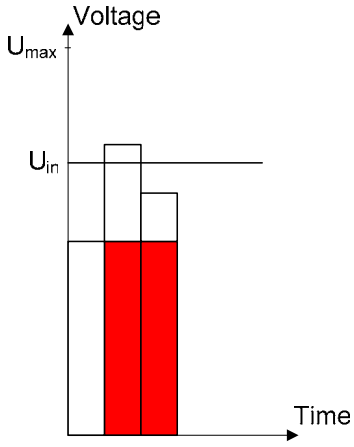


Figure 35 Bit 3 in conversion

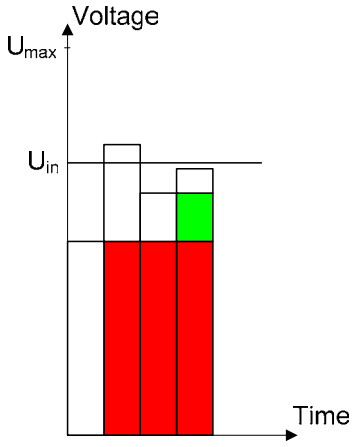


Figure 36 LSB in conversion

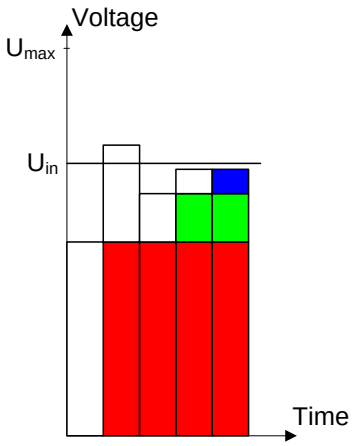


Figure 37 Result of conversion

Integrating ADC:s

An integrating ADC is in a way similar to a counting ADC but here we don´t use the counter value to control a ADC but we use it to count time. The time we count is the time taken for a capacitor to charge to the same potential as the analog input value. The higher the value, the longer the time. We use the integrating OP-circuit that we presented earlier in *Figure 17*.

Single-slope integrating ADC

If we use a negative input voltage $-U_{ref}$ to the integrator we can let the integrator ramp up to the same voltage as the analog input voltage while we let a counter count the number of clock pulses generated during the measurement time.

$$U_{out} = -\frac{\left(-U_{ref}\right)}{R \cdot C} \cdot t = -\frac{U_{ref}}{R \cdot C} \cdot t$$

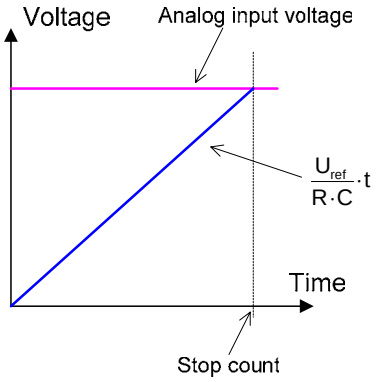


Figure 38 Up-counting ADC, time diagram

When we reach the same level as the analog input voltage we stop the count, *Figure 38*. We can see that the higher the input voltage the longer it will take to reach the end level so the count is directly proportional to the value of the input voltage.
The switch across the capacitor is there to make sure the capacitor is uncharged when the integration starts, that is it will open when the measurement period starts. We call the circuit a *single-slope integrating ADC*, *Figure 39*.
If we want some accuracy in the measurement we need the count to be pretty long so the converter is slow. We assume that the input voltage is constant under the integration time which means that we can only deal with slowly varying signals. But one positive thing here is that if we have noise on the input signal, that is small random fluctuations

in the analog voltage over time, then these variations could cancel out during the integration period. This converter is sensitive to variations in the resistor and capacitor value and to fluctuations in the frequency of the clock used to control the counter so the circuit is seldom used. But it has developed into another ADC that is very common specially in multimeters.

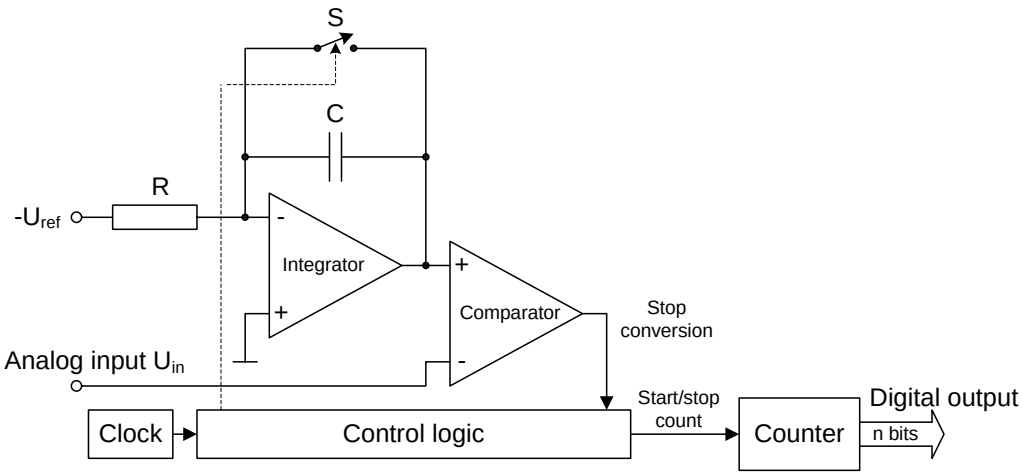


Figure 39 Single-slope integrating ADC

Dual-slope integrating ADC

The basic circuit of the dual-slope integrating ADC is very similar to the single-slope converter but with some added features, *Figure 40*.

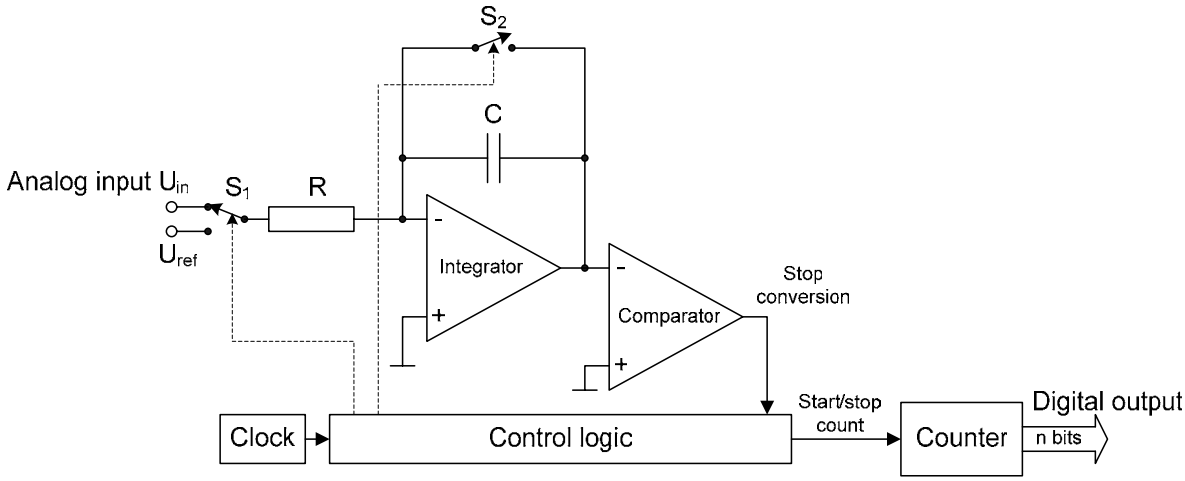


Figure 40 Dual-slope integrating ADC

In this circuit we start by letting the integrator integrate the input voltage U_{in} over a fixed time t_{ref}. This means that the output voltage after this period U_x is directly proportional to the input voltage U_{in}

$$U_{Phase\,1} = -\frac{U_{in}}{R \cdot C} \cdot t_{ref}$$

Then we replace the input voltage with a reference voltage U_{ref}. This voltage should have an opposite sign to the input voltage. This means that the output voltage will now start to ramp down

$$U_{Phase\,2} = U_{Phase\,1} - \frac{U_{ref}}{R \cdot C} \cdot t$$

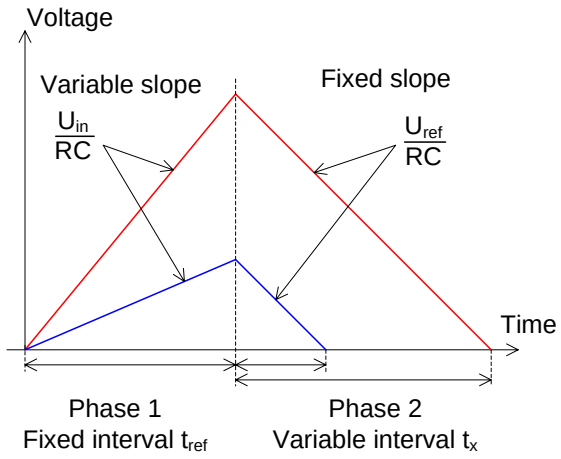


Figure 41 Ramps of dual-slope integrating ADC

During this second time we let a counter count the time and we stop the count when the output voltage reaches zero (0) at the time t_x.

The higher the input voltage U_{in}, the higher the output voltage after the first integration U_x and the longer the second integration time t_x. We get the relationship

$$\frac{U_x}{U_{ref}} = \frac{t_x}{t_{ref}}$$

We can see that if U_{ref} equals the maximal voltage of the converter, that is the voltage that gives the maximal value of our digital value, then this relationship will give time relative this maximal time.

This converter is more immune to fluctuations in the component values and the clock frequency than the single slope converter since we use the same components and clock in both integrations. Since we have two integrations here this converter is even slower than the single-slope converter. We call it a *dual-slope ADC*. This converter can cancel noise in the input signal the same way that the single-slope converter can.

Sampling

To introduce the concept of sigma-delta modulators we need to start by introducing the concepts of introduce continous A/D-conversion and sampling.

In most cases when we are about to convert a analog signal to it´s digital equivalent we need to take into consideration that the signal is changing with time and to get a true representation of the signal we need to do several conversions of the signal. In most cases we do this on a regular basis that is we have a fixed time interval, T, between each conversion. We say that we *sample* the signal with a *sampling frequency* of $f_s = \frac{1}{T}$.

The so called *sampling theorem* or the Nyquist theorem tell us that to get a true representation of the signal we need to sample the signal at least twice during each period of the signal. This means that our sampling frequency needs to be more than twice as high as the highest frequency content in the signal.

The sampling theorem

For faithful reproduction of the sampled signal the sampling frequency has to be more than twice the highest signal frequency

Aliasing

If we don´t obey this rule the high frequency content of our signal will disturb the low frequency part of the signal. The high frequency signals will give so called *aliasing* of the signal and show up as signals with frequencies lower than half the sampling frequency, *Figure 42*.

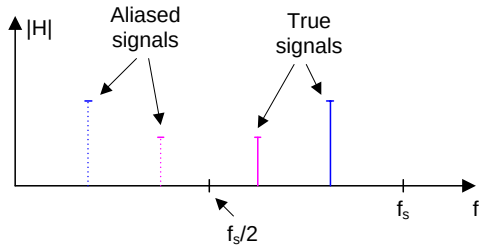


Figure 42 Aliasing

In a practical situation we can never guarantee that the signal we sample does not contain this higher frequency components, there might be disturbances or what ever, so we need to remove these frequency components before the sampling. To do this we let the signal pass a low pass filter (LP) that only lets signals with frequencies lower than half the sampling frequency pass. We introduce an *anti-aliazing filter*, *Figure 43*. In reality there are no filters that block out the high frequencies totally but we can at least attenuate them so much that they disappear in the resolution of the ADC. A well know example of this process is that CD-records are sampled with the sampling frequency 44,1 kHz to pass the signal frequencies we can hear, that is up to 20 kHz. This means that we need a sharp filter that can pass all signals with frequencies up to 20 kHz and attenuate all signals with frequencies higher than half the sampling frequency, that is higher than 22,05 kHz. The filter has to be placed before the sampling and the AD-conversion, that is while the signal still is analog and it is quit hard to synthesize so sharp an analog filter. This is one of the problems we are about to address with the sigma-delta converter.

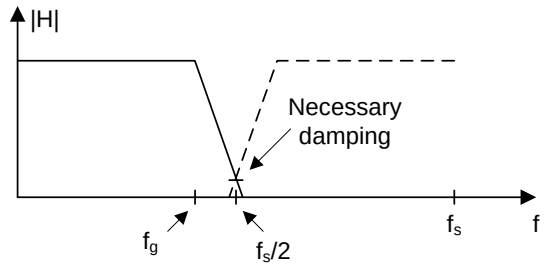


Figure 43 Anti-aliasing filter

Over-sampling

Over-sampling means that we use a sampling frequency that is much higher than what is needed to obey the sampling rule. The high sampling frequency is in most cases an integer multiple of the needed sampling frequency, $N \cdot f_s$. By doing this we can ease the demands on our anti-aliazing filter. Now the analog filter will only have to attenuate signals with frequencies above $N \cdot f_s - \frac{f_s}{2}$, *Figure 44*.

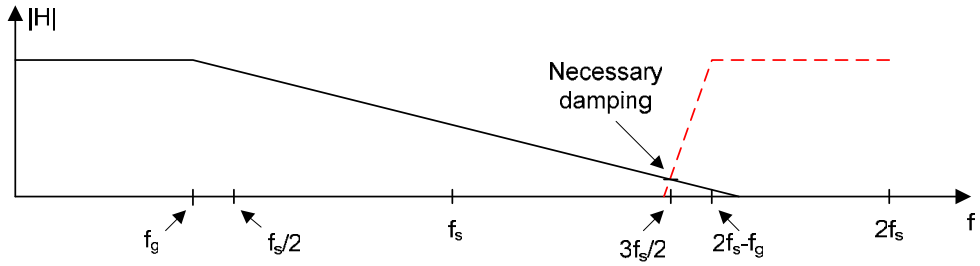


Figure 44 Anti-aliasing filter at 2 times oversampling

After the sampling we use a process called down-sampling where we decimate the sampling frequency to f_s. Before we do this we still need to introduce a filter that removes all signals with frequencies above $\frac{f_s}{2}$ but this time we use a digital filter, since we are still in the digital domain, and it is much easier to create a sharp digital filter than the corresponding analog filter, *Figure 45*.

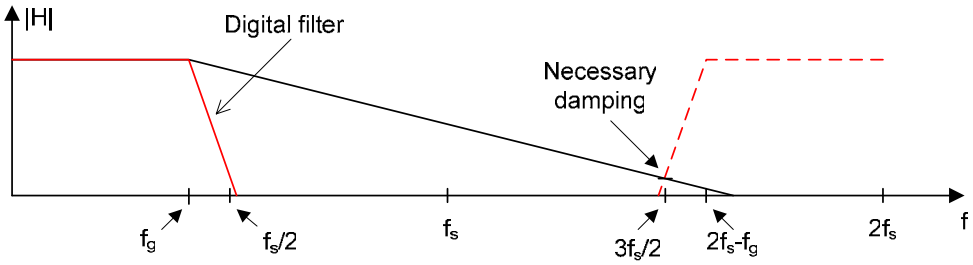


Figure 45 Anti-aliasing filter at 2 times oversampling complemented by digital filter

Sample and hold

Most A/D-converters, except the integrating ones, expect the input voltage to be constant during the conversion time. To make sure that this is true we can take a short sample of the signal and then use a capacitor to keep it constant under the conversion time, *Figure 46*.

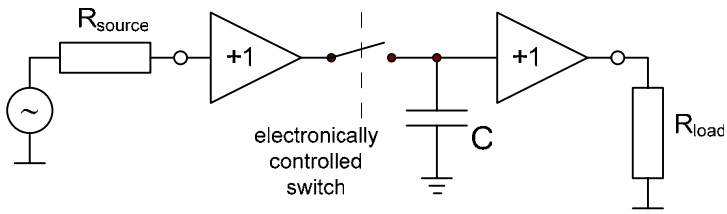


Figure 46 Sample and hold circuit

We call the circuit a *sample and hold* circuit (*S&H*), *Figure 47*.
In the figure we have added two buffers. The circuit connected before the sample and hold circuit will always have a finite output impedance and this will form a RC-circuit together with the capacitor in the S&H and this RC-net will have a time constant that prevents the capacitor from instantly reaching the input voltage. This time constant is $\tau = R \cdot C$ and we shorten the time by inserting the buffer which has a low output resistance.

On the other hand we find that the circuit connected after the S&H will discharge the capacitor through its input resistance and we make this discharge small by inserting the other buffer with its high input resistance.
In reality we can not make the time during which we sample the input signal infinitly short so we will track the input signal during a short period. We call this a *track and hold* circuit, *Figure 48*.

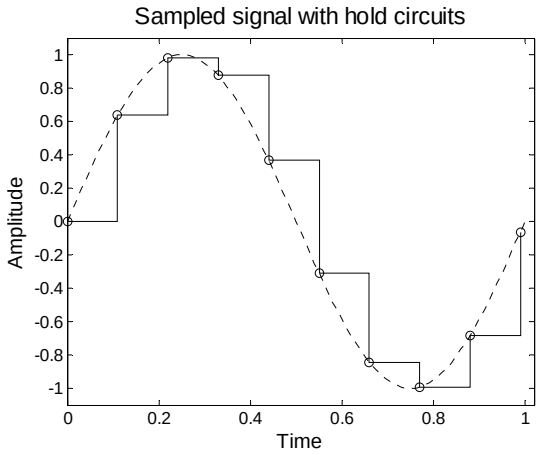


Figure 47 Sample and hold diagram

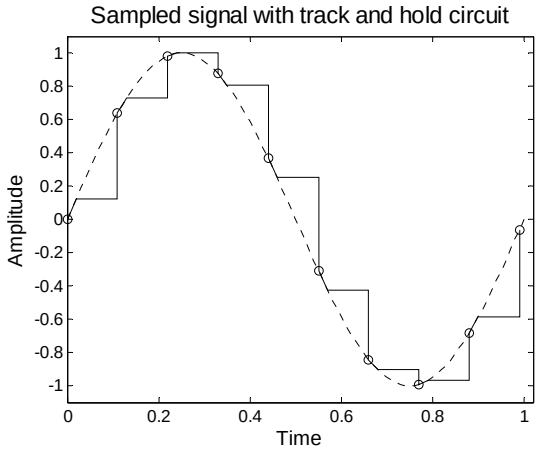


Figure 48 Track and hold diagram

The sigma-delta modulator

Now let us move on to the *sigma-delta converter (∑-Δ modulator*), *Figure 49*.
As we can see from *Figure 41* we compare the input signal with a signal that is feed back from the output of the circuit, we subtract the latter from the former. The signal from the subtraction is feed to a integrator that integrates this difference over time. The output from the integrator is connected to a one-bit ADC, which in practice is a

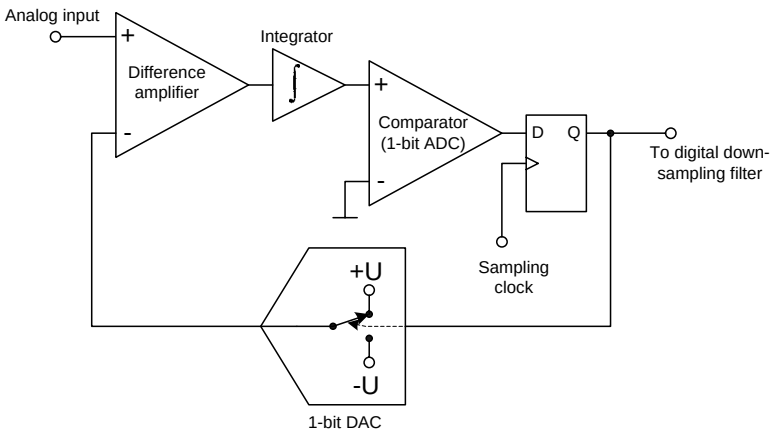


Figure 49 Sigma-delta modulator

comparator that generates a high output signal, a one (1), when the input signal to the comparator is higher than zero and a negative signal, a zero (0), when the signal is negative. The comparator will do this comparison with a over-sampled sampling frequency $N \cdot f_s$ governed by the latch controlled by the sampling clock. This means that this output signal is a series of ones and zeroes, a one-bit signal or a serial signal. This signal is then feed back through a one-bit DAC which in reality only is an interface that changes the level of the signal from one (1) and zero (0) to positive and negative supply voltage respectively.

The one-bit output signal is then down-sampled to the lower sampling frequency f_s and this process means that we take an average over the N samples during N sampling periods, N is the oversampling factor. We use this average to form a new word with more bits than one. The number of bits can be $^2log(N)$.

Now why do we do all this? We can remember from earlier that our conversion from analog to digital signal had a limited resolution governed by the number of bits used. If we study the frequency spectra of this signal we will find that the in addition to the signal we have a noise floor that is spread over the whole frequency band up to $\frac{f_s}{2}$, *Figure 50.*

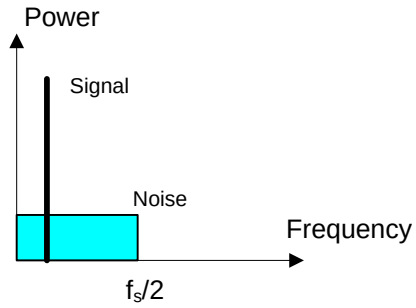


Figure 50 Signal with noise floor in the span up to $\frac{f_s}{2}$

When the signal is oversampled we spread the same noise over the larger frequency band $\frac{N \cdot f_s}{2}$ which means that the noise at each frequency gets lower, *Figure 51*.

When we down-sample the signal we use a low pass filter to remove the high frequency content above $\frac{f_s}{2}$ and that means that we filter of a large portion of the noise. For every time we increase the sampling frequency by a factor of four (4) the SQNR will increase by a factor of 6 dB which is the same as adding one bit to the word length. At the same time one can show that the sigma-delta modulation shapes the noise so that a large portion of noise will be pushed to higher frequencies. That is more of the noise will be moved to the frequency band that is filtered off in the down-sampling process and the gain in SQNR will be greater than stated above, *Figure 52*.

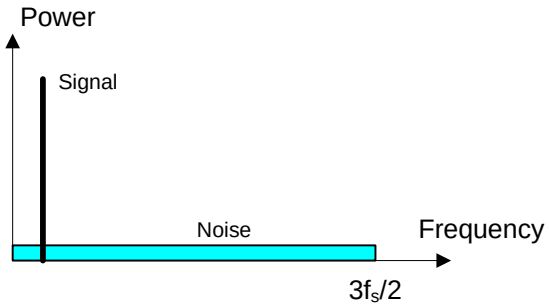


Figure 51 Signal with noise floor in the span up to $\frac{N \cdot f_s}{2}$

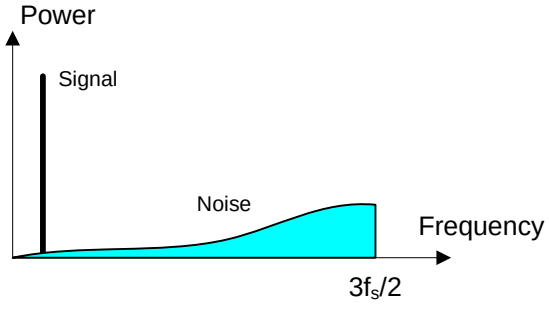


Figure 52 Noise shaping

ADC:s with multiplexer

In many cases we need to convert more than one analog signal to its digital counterpart. In this cases we can save money and circuitry my using only one ADC and connect the analog signals one by one to the ADC and do the conversions in sequence. To do this we connect an analog multiplexer (MUX) in front of the ADC to take care of the switching and in many cases we integrate the MUX into the ADC circuit, *Figure 53*.

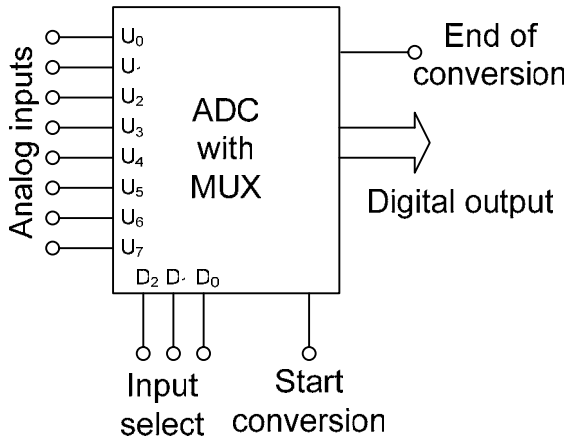


Figure 53 ADC with multiplexer

This savings have some draw backs. Later on we will talk about the conversion time, that is the time it takes for the ADC to perform one analog to digital conversion, and in that context we will see that the time between consecutive conversios on a single channel in the sequence will increase proportionally with the number of channels.

The we do the set of conversions in a sequence means that we can not sample and convert two signals at exactly the same time. This could be a disadvantage for example when we would like to compare the phase relation between two signals.

Angle decoders

We will now move to a little odd group of devices that don´t deal with voltages as input signals but still can be categorized as A/D-converters. These are *angle detectors* that give a digital reading decoding the rotation, the angle of an axis. We can use the same princip to detect linear movement instead of rotating movement.

We can divide this into to groups, incremental decoders and absolute decoders.

Incremental angle decoder

In the *incremental decoder* we do not detect the angle as such but the number of steps we have moved where the steps are the angular resolution of the detector, that is we don´t detect the absolute angle but the movement. To do this we attach a disc with slots to the axis. The slots are evenly distributed around the circumference of the disc and we use some kind of detector to detect when a slot passes and we count the pulses that are generated during the rotation, *Figure 54*. The detector is in many cases an optical device consisting of a LED and a photo transistor or a

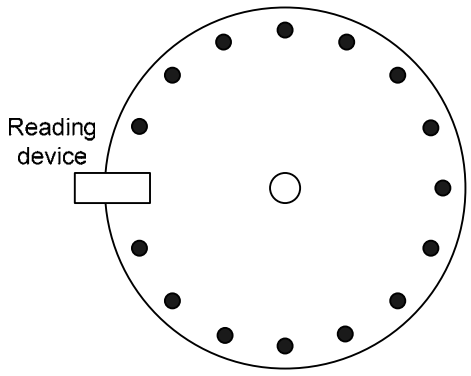


Figure 54 Incremental angle detector

magnetic device consisting of some kind of reed relay.
We can understand that to transform the count to a absolute angle we need to reset the device, that is place it at the angle 0º before we start the detection.
The counting of the number of resolution steps and not the absolute angle makes it possible to detect angles of more than 360º.
With just one circle of slots we can not detect the direction of the rotation. To do this we need two circles of slots that are slightly out of phase which means that we can detect the direction of the rotation by detecting which of the slots that passes the detector first, *Figure 55*.

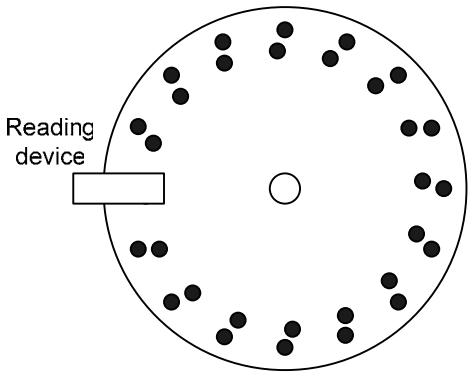


Figure 55 Incremental angle detector with direction detector

Absolute angle decoder

In an absolute angle decoder we detect the absolute angle which means that we need a way to tell different angles apart. We do this by placing a number of detection circles around the shaft. The direct way is to use as many circles as we have bits in the digital word that gives the angle. Let us look at the normal binary coding, for simplicities sake we use only three bits, *Figure 56*.

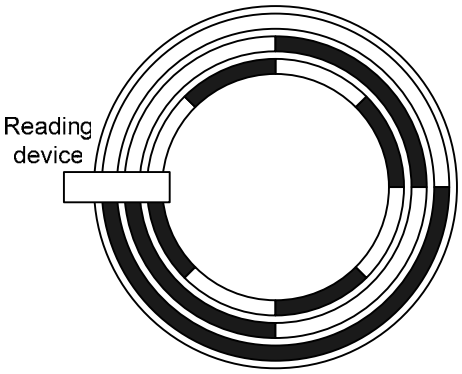


Figure 56 Absolute angle detector

Now when we rotate the shaft we will go from code to code. Let us look at the transition from 3 to 4 that is from 011 to 100. In this transition we can see that all three bits will change their state but the reading of the code will have some resolution and we can be pretty sure that the transition of the three bits will not be detected at exactly the same angle. This means that the transition will give a number of false codes for example the series in *Table 4*.

Decimal	*Bit 2*	*Bit 1*	*Bit 0*
0	*0*	*0*	*0*
1	*0*	*0*	*1*
2	*0*	*1*	*0*
3	*0*	*1*	*1*
4	*1*	*0*	*0*
5	*1*	*0*	*1*
6	*1*	*1*	*0*
7	*1*	*1*	*1*

Table 3 Binary coding, three bits

This is clearly not the wanted behaviour. To get a correct function we need to change the coding from the ordinary binary coding to a coding that only changes the state of *one* bit in each transition. On such code is the *Gray code*, *Table 5*.
If we then want to translate this to the ordinary decimal code we only need a look up table to match the two codes.

Bit 2	Bit 1	Bit 0	Decimal	Correct/false
0	1	1	3	Correct
0	1	0	2	False
1	1	0	6	False
1	0	0	4	Correct

Table 4 Fake codes in the transition from 011 to 100

Decimal	Bit 2	Bit 1	Bit 0
0	0	0	0
1	0	0	1
2	0	1	1
3	0	1	0
4	1	1	0
5	1	1	1
6	1	0	1
7	1	0	0

Table 5 Gray coding, three bits

Voltage to frequency converter

The last type of ADC we will mention briefly is the voltage to frequency converter. This circuit consist of a oscillator whose frequency is linearily proportional to a control voltage. The output is still analog since the frequency can take on any value but it is quit easy to use a digital counter to measure the period of the generated pulse signal and from this generate a digital reading that is proportional to the analog input signal to the voltage to frequency converte, *Figure 57*.

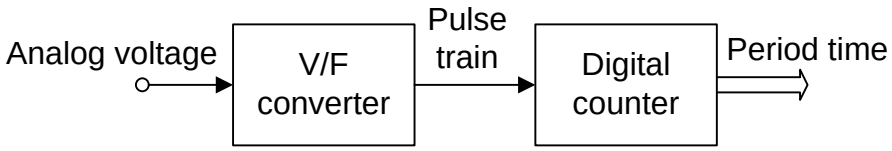


Figure 57 Voltage to frequency converter

A/D- and D/A-specifications

When we studie the data for a given converter or are about to decide which converter to use we need to studie the converter specifications in the datasheets. We will study the most important data. Some of the specifications are the same for the A/D-converter and the D/A-converter while some of them differ.

Voltage span

The voltage span is for the A/D-converter the analog input voltage that the converter can accept while it is the possible output voltages for the D/A-converter. The span is in most cases unipolar or bipolar meaning that it either spans from 0 Volts to som maximal voltage U_{max} or have a symmetrical span $\pm U_{max}$. In some cases we can use a offset voltage to change the span of the converter from unipolar to bipolar. Observe that the output voltage from a DAC will never reach U_{max}. It will always be one resolution step lower, that is $U_{max} - \Delta$.

Resolution

We have talked about the resolution earlier but if we look in the datasheet this will normally not be specified av a fraction of full scale deflection (FSD) but only be given as the number of bits used by the converter.

Accuracy

Earlier we didn´t see any difference between the resolution and the accuracy, that is we meant in both cases the smallest voltage difference we could detect or generate. If we look in the datasheets they usually mean something else with accuracy. Every conversion generates some error due to the design of the converter and this is given as a maximal error in number of resolution steps and it is normally given as $\pm x\ LSB$ and typical values might be $\pm 1/4\ LSB$, $\pm 1/2\ LSB$ or $\pm 1\ LSB$.

Conversion time

This only applies to A/D-converters and is the necessary time to convert the analog input value to the digital output value. In some cases the datasheet give more than one time with different levels of accuracy. Of course a shorter time will give a lower accuracy.

The conversion time, t_c, will limit the maximal sampling frequency in our system. We have to be able to carry out the conversion before a new sample arrives and this means that the maximal sampling frequency will be

$$f_{s,max} = \frac{1}{t_c}$$

If we have a system where the ADC contains a multiplexer to connect a sequence of N analog inputs to the converter then we have to carry out N conversions before we can make the next conversion on the same channel and this means that the maximal sampling frequency on each channel will be

$$f_{s,max,MUX} = \frac{1}{N \cdot t_c}$$

Settling time

This is the equivalent to conversion time when we talk about D/A-converters. It is the time it takes before the analog output signal converted from the digital word have settled to a stable value. The output voltage stability is given as being within some error range, for example $\pm 1/2\ LSB$.

Offset error

An *offset error* is a shifting of the signal upward or downward meaning that the A/D- or D/A-conversion will always generate an error of the same amount no matter the signal value, *Figure 58*.

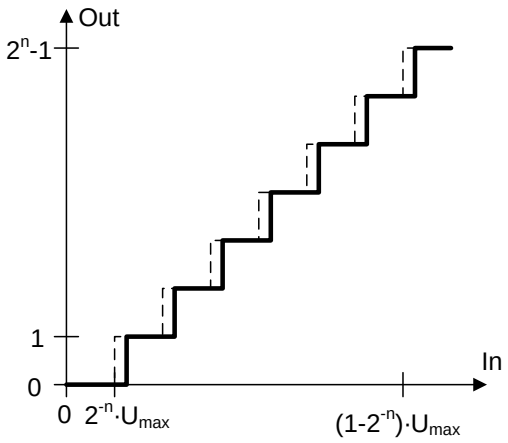


Figure 58 Offset error

Amplification error, scale factor error

The *amplification error* is an error in the slope of the transfer curve, *Figure 59*. In contrast to the offset error the amplification error will increase with the magnitude of the signal. Another name for this error is *scale factor error*.

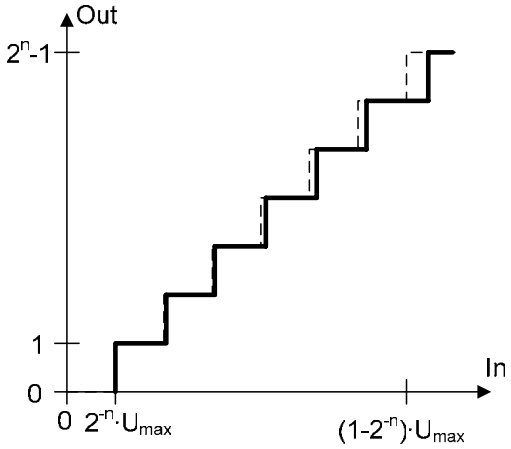


Figure 59 Amplification error

Linearity error

The *linearity error* is an error in the linearity of the transfer function meaning that the error at different levels in the transfer function differ but it will not necesserily increase with the magnitude of the signal, *Figure 60.*

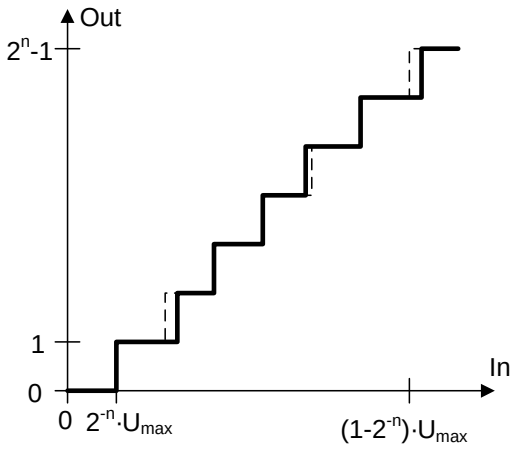


Figure 60 Linearity error

Parallel or serial interface

So far we have hardy mentioned the digital interface from the ADC and to the DAC. These interfaces could be either parallel or serial. In the parallel case all the bits in the digital word are present att the same time at parallel wires. In the serial case we clock the bits in and out of the interface bit by bit, that is we shift the bits in to or out of the registers one bit at a time.
It would seem to be a faster and simplier way to use the parallel interface but in spite of this modern equipment tend to use serial interfaces. The reason for this is that we are constantly moving towards faster and faster data transfers and it gets harder to be certain that all the parallel bits are precent at exactly the same time so that we don´t reed some bit to late or to early. We have reached the stage when the tracing and length of the copper paths on the printed circuit board affect the result. Are the paths for different bits of different lengths then the timing might get wrong, *Figure 61*. At the same time it can be quit hard to route all this parallel copper paths.
When we use a serial interface then the number of paths are smaller and we only need to clock one bit at a time which is simplier.
We will get back to this when we talk about communication interfaces.

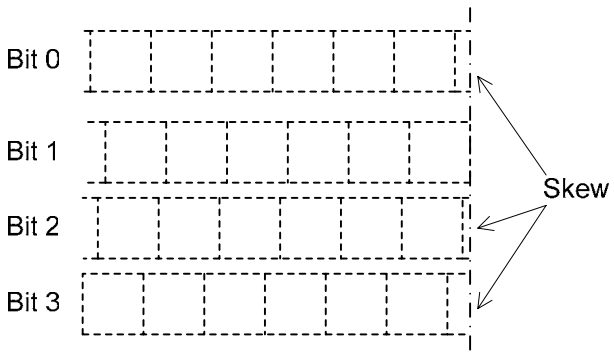


Figure 61 Skew in a parallell signal