

Chapter 14

D-A and A-D Conversion

In Chapter 12, we looked at how digital data can be carried over an analog telephone connection. We now want to discuss the opposite — how analog signals can be carried over a digital connection. To do that, we first need to understand *digital-to-analog converters* (also called *D-to-A converters* or just DACs) and *analog-to-digital converters* (also called *A-to-D converters* or just ADCs.)

Digital-to-Analog Conversion

Fig. 14-1 shows a simple two-bit digital-to-analog converter. Switches A and B are our digital input, while the output can be measured with a voltmeter at the right. Using simple DC circuit theory with series and parallel resistors, we get the following table for the four possible combinations of switch settings:

Switch A	Switch B	Output voltage
down	down	0 v
down	up	1 v
up	down	2 v
up	up	3 v

Converting the down and up settings of the switches to the binary bits 0 and 1, respectively, gives us

Switch A	Switch B	Output voltage
0	0	0 v
0	1	1 v
1	0	2 v
1	1	3 v

Thus a two-bit binary number entered into the two switches is converted to a corresponding analog value — the voltage measured by a voltmeter.

In the 2-bit circuit, there are 2^2 voltage values ranging, in this case, from a low of 0 volts up to a maximum of 3 volts.

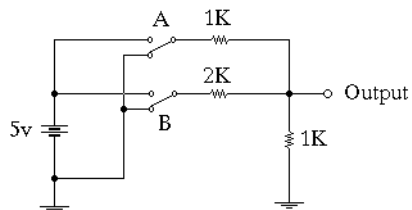


Fig. 14-1. 2-bit D-to-A Converter

This circuit can readily be extended to more than two bits. For example, Fig. 14-2 shows a 4-bit converter with four switches labelled A, B, C, and D. Switch A represents the most significant bit, while D is the least significant. Note how the scaling is done by using 1K, 2K, 4K, and 8K resistors: the most-significant bit (from switch A) gets the smallest resistor because it is supposed to contribute the most to the output.

Again using the switches to input binary numbers, the output voltages are as follows:

Switch A	Switch B	Switch C	Switch D	Output voltage
0	0	0	0	0 v
0	0	0	1	0.217 v
0	0	1	0	0.435 v
0	0	1	1	0.652 v
0	1	0	0	0.869 v
0	1	0	1	1.09 v
0	1	1	0	1.30 v
0	1	1	1	1.52 v
1	0	0	0	1.74 v
1	0	0	1	1.96 v
1	0	1	0	2.17 v
1	0	1	1	2.39 v
1	1	0	0	2.61 v
1	1	0	1	2.83 v
1	1	1	0	3.04 v
1	1	1	1	3.26 v

With 4 bits, we now have 2^4 or 16 possible output voltages, evenly spread out from 0 volts through 3.26 volts. For example, the last line value of 3.26 volts (which is the voltage for a binary input of 1111 or decimal 15) is 15 times the voltage for 0001, which is 0.217 volts.

The circuit of Fig. 14-2 could be expanded to more bits by adding more switches and resistors, but the resistors become a problem. Suppose we were to expand the circuit to 12 bits. We would have 2^{12} or 4096 evenly spaced output voltages in the range from 0 volts up to a

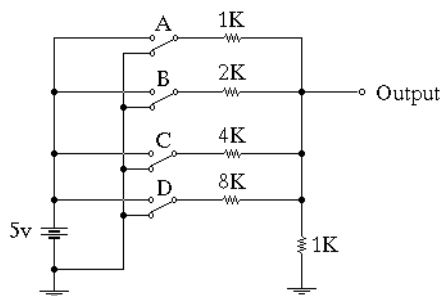


Fig. 14-2. 4-bit D-to-A Converter

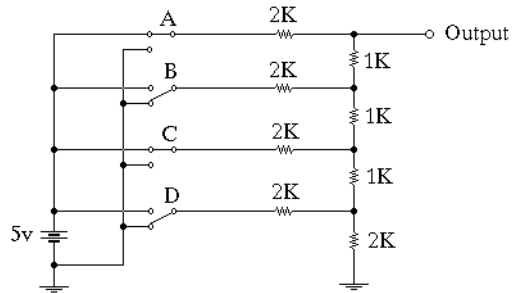


Fig14-3. A better and more popular DAC circuit

bit above 3 volts. Splitting the 3 volts into 4096 tiny steps would make the difference between adjacent voltage steps about $\frac{3}{4096}$ or about 0.0007 volts. Even a tiny error in one of the resistors could easily change the output by more than this amount. For that reason, a more popular circuit for digital-to-analog converters is shown in Fig. 14-3. It can easily be expanded to more bits as well.

One final comment — in an actual circuit, the switches in the previous circuits would be replaced by transistors, and the binary input, rather than being entered manually by setting switches, would come in as a parallel digital signal.

Analog-to-Digital Conversion

Digital-to-analog converters (DACs) are simple and cheap. Analog-to-digital converters (ADCs), on the other hand, are substantially more complex and more expensive.

There are a number of ways to convert an analog signal to a digital number. One common approach is shown in Fig. 14-4, which uses a DAC to make an ADC. A digital counter outputs a binary number, which is converted to analog by a DAC. The output from the DAC is then compared with the analog input voltage in

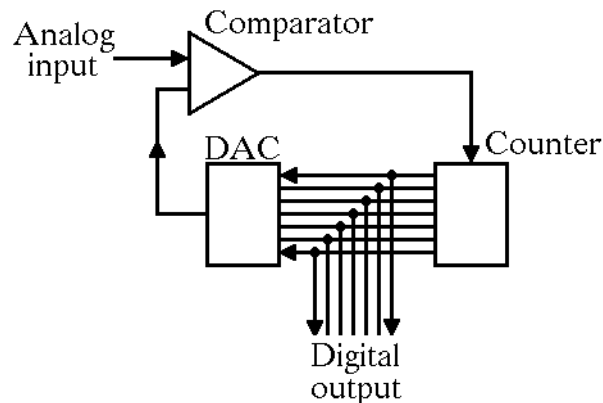


Fig. 14-4. Successive approximation A-to-D converter

a comparator. If the DAC output (and thus the counter's output) is too small, the comparator tells the counter to count up to a higher number. If, on the other hand, the DAC output (and thus the counter's output) is too big, then the comparator tells the counter to count down to a lower number. Either way, the DAC output eventually reaches the same voltage as the analog input, at which point the digital number in the counter (which also happens to be the digital output from the whole circuit) represents the analog input.

That is the basic idea; there are a number of variations on the circuit. In some, the counter always starts at a count of zero and counts up toward the final number. In others, the counter can count both up and down, and can approach the final count from either side. Both of these have the disadvantage of being slow, since it takes the counter some time to get to the correct value. For example, suppose we have a 12-bit counter (which can count from 0 to 4095 in decimal), and the input voltage is fairly close to the maximum limit. If the counter starts at 0, it would take about 4000 counts before it gets to the final value.

A somewhat different approach starts the counter in the middle, using a method that mathematicians call *bisection*, that would go like this (assuming the input voltage is still close to the maximum):

(a) Since the digital output could be anything from 0 to 4095, try the midpoint of 2048.

(b) You discover the DAC output is too small, so the digital number is more than 2048. Since you now know it is somewhere between 2048 and 4095, again try the midpoint at 3072.

(c) You discover the DAC output is still too small, so the digital number is more than 3072. Since you now know it is somewhere between 3072 and 4095, again try the midpoint at 3584.

(d) You discover the DAC output is still too small, so the digital number is more than 3584. Since you now know it is somewhere between 3584 and 4095, again try the midpoint at 3840.

(e) Keep going until you reach the correct number.

The process sounds complicated, but in binary it's simple when you notice the following:

Decimal	Binary
2048	100000000000
3072	110000000000
3584	111000000000
3840	111100000000

Each step in this sequence gets one more bit correct, so it only takes 12 steps to get all 12 bits right.

An even faster method is a flash converter, which can convert an analog number into binary "in a flash"

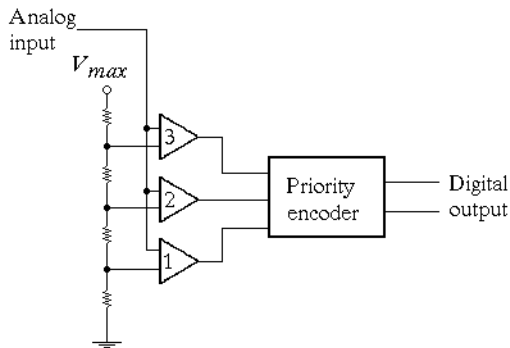


Fig. 14-5. 2-bit Flash A-to-D converter

— that is, in just one step. Fig. 14-5 shows the basic idea for a simple 2-bit ADC.

Suppose the voltage V_{max} is 4 volts, and the four resistors are all equal. Comparator 1 then gets 1 volt, comparator 2 gets 2 volts, and comparator 3 gets 3 volts from the resistor voltage divider. Now let's apply an analog input voltage. There are four possible results:

(1) If the input voltage is under 1 volt, all three comparators report that the analog voltage is smaller than what they get from the resistors. When the priority encoder gets the “low-low-low” report, it outputs the binary number 00.

(2) If the input voltage is between 1 and 2 volts, comparator 1 reports that the analog input is higher than its other input, while comparators 2 and 3 still report that it's lower. When the priority encoder gets the “high-low-low” report, it outputs the binary number 01.

(3) If the input voltage is between 2 and 3 volts, comparators 1 and 2 both report that the analog input is higher than their other input, while comparator 3 still reports that it's lower. When the priority encoder gets the “high-high-low” report, it outputs the binary number 10.

(2) If the input voltage is more than 3 volts, all three comparators report that the analog input is higher than their other input. When the priority encoder gets the “high-high-high” report, it outputs the binary number 11.

The priority encoder circuit is simply a combination of AND and OR gates which looks at its three inputs, and outputs a two-bit number representing the highest comparator which sent it a “high” signal.

While the flash ADC is very fast, it gets very complicated as we try to increase the number of bits. For example, a 12-bit converter (which therefore has to recognize 2^{12} or 4096 different voltage levels) would need 4096 resistors, 4095 comparators, and a priority encoder circuit with 4095 inputs and 12 outputs. Hence fast and accurate A-to-D converters get quite expensive.

Voltage limits

The analog voltage of each of the A-to-D and D-to-A converters we have looked at has certain limits. For example, the 2-bit A-to-D converter of Fig. 14-5 can recognize the following voltages:

Analog voltage range	Digital output
0 to 1 volt	00
1 to 2 volts	01
2 to 3 volts	10
above 3 volts	11

But what does “above 3 volts” mean? Does it mean we can input 1000 volts and expect it to give us a 11 output? (ZAP!)

Engineers normally consider the input range of an n -bit (n is 2 in this case) converter to be 2^n equal-sized voltage intervals. That would make the bottom line read

3 to 4 volts	11
--------------	----

We would then say that this A-to-D converter has an input voltage range of 0 to 4 volts, which it divides into four equal-sized steps.

Real-world converters let you specify the minimum and maximum voltage. For example, an actual flash converter might be wired like Fig. 14-6 (compare it with Fig. 14-5). V_{max} and V_{min} would both become pins on the converter, and the user could connect any voltage (within limits) to them. If you connect +4 volts to V_{max} and 0 volts (ground) to V_{min} , then the converter covers the range from 0 to 4 volts, and acts like the circuit of Fig. 14-5.

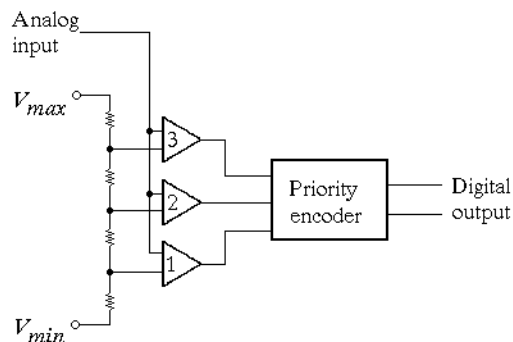


Fig. 14-6. ADC showing V_{max} and V_{min} connections

But you could just as well connect V_{max} to +2 volts and V_{min} to -2 volts. That would make the converter divide the range from -2 volts to +2 volts into four steps like this:

Analog voltage range	Digital output
-2 to -1 volts	00
-1 volts to 0 volts	01
0 to +1 volts	10
+1 to +2 volts	11

We can summarize this as follows:

- The analog voltage can range from V_{\min} up to V_{\max} .
- V_{\max} and V_{\min} are chosen to match the expected input signal voltages, and may be positive or negative.
- An n -bit converter divides this voltage range into 2^n equal-sized steps.
- The size of each step is

$$\frac{V_{\max} - V_{\min}}{2^n}$$

The step size determines how accurately the A-to-D converter can measure the input signal — the smaller the step size, the more accurate.



In most cases, V_{\max} and V_{\min} are constant, and so the step size is constant as well. But there are some exceptions.

If the analog input signal is fairly large, then you need large values of V_{\max} and V_{\min} to accommodate it. But if the signal is small, then V_{\max} and V_{\min} should also be small so as to be able to measure the input more accurately.

A problem comes up when the analog input voltage changes — if sometimes it is large, sometimes small. Finding compromise values of V_{\max} and V_{\min} is difficult. We will see this when we look at how A-to-D converters are used to digitize varying signals. There are two common solutions:

- Change V_{\max} and V_{\min} over time. Use big values when you expect the signal to be big for a while, change to smaller values when you expect the signal to be small.

OR

- Rather than use equal-sized steps all the time, make some big and some small.

More on these later.



Sample-and-Hold

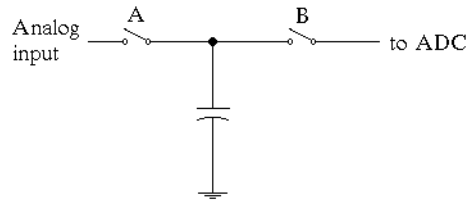


Fig. 14-7. Sample-and-hold

No A-to-D converter is instantaneous; even the flash converters take some time to work. What will the converter do if the analog input voltage is changing while it is trying to convert it to digital?

This is a major problem, because some converters can make huge errors if the input is changing during conversion. For example, if an 8-bit converter is trying to decide whether to output 10011111 or 10100000 (which are adjacent values) for a particular input, it might make a mistake and accidentally output part of one number and part of the other, such as the 100---- from the first and the ---00000 from the second, to make up 10000000. This would be a rather substantial error.

Fig. 14-7 shows the solution — a *sample-and-hold* circuit. The sample-and-hold is basically a capacitor, which is switched either to the analog input, or to the A-to-D converter by means of two switches (which are replaced by transistors in an actual circuit.)

When the circuit wants to do a conversion from analog to digital, it briefly closes switch A. This connects the analog input voltage to the capacitor and charges it up to the value of the input. It then opens switch A and closes switch B, which disconnects the capacitor from the input, and connects it to the A-to-D converter instead.

At this point, the analog input can change all it wants — the A-to-D converter doesn't see that. Instead, the A-to-D converter sees the constant voltage on the capacitor, which is equal to whatever the input was a fraction of a second earlier. In this way, the capacitor took a quick *sample* of the input voltage, and now *holds* it constant while the A-to-D converter measures and converts it.

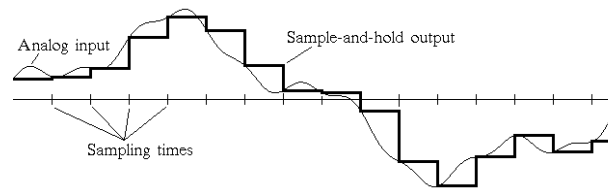


Fig. 14-8. Sample-and-hold output

In communications applications, the sample-and-hold is often used when the input is being sampled at regular intervals, as in Fig. 14-8. Here we see the analog input being sampled at evenly-spaced sampling times. At each point, the sample-and-hold grabs the voltage at that instant, and holds it constant until it's time for the next sample.

Sampling

Fig. 14-8 introduces the idea of sampling. Many communications systems — such as the modern telephone system — digitize an analog signal and then transmit its digital representation over the network. At the far end, this digital information is then converted back into analog form and sent to the user. The object is to send enough information that the analog waveform can be reconstructed without excessive error.

For instance, consider the sentence “H_w a_e y_u?” Even though there are some missing letters, with a bit of time you can probably figure out that the sentence should read “How are you?” This is an example of sampling. We can omit small parts of the text without losing any essential information. But we need to do two things to make sure you can correctly decode this sentence: (1) make sure that the samples come often enough that the missing parts between them are small and can be reliably filled in, and also (2) make sure that the samples themselves are correct. For instance, “H_ _ _e _o_?” does not contain enough samples, because it might be misunderstood to mean “His fee too?” Likewise, “H_s a_e y_u?” has a mistake and would also cause an error.

In terms of electrical sampling, these two requirements mean that samples must be taken often enough, and accurately enough, so that we can later fill in the missing pieces without introducing any new errors. Let's now look at these timing and accuracy requirements in more detail.

Sampling Rate

The rate at which samples are taken is called the *sampling rate*. So how often should they be taken?

The answer comes from the *Nyquist Sampling Theorem*: In order for the samples to adequately represent the analog waveform, the sampling rate must be at least twice the highest frequency contained in the signal. (The fine print says that the sampling rate must be ever so slightly *more* than twice as high.) Looking at it from the other direction, the highest frequency component present in the analog signal being sampled must have a frequency less than half of the sampling rate.

There's still another way to look at this: Consider the fastest possible cycle in the analog waveform; if you have slightly more than two samples of that cycle, you can reconstruct the entire cycle from the samples.

A CD or Compact Disc makes a good example. The audio signal on a CD has a frequency range from 20 to 20,000 Hz. Since the highest frequency is 20,000 Hz, any sampling rate above 40,000 times per second — twice the highest audio frequency — will provide enough information to allow the CD player to completely reconstruct the audio signal from the samples. (Compact disks sample 44,100 times per second to provide a slight safety factor.)

To understand the Nyquist Sampling Theorem, note that its purpose is to make sure that the original waveform can be reconstructed from the samples. So let's see what is involved in reconstructing a wave from samples by looking at a “connect the dots” puzzle like the simple one in Fig. 14-9, which shows five numbered dots to be connected. Most people given this puzzle would use straight lines to connect the dots, as in the top trace. But when you think about it, there are zillions of different ways to connect two adjacent dots — you can use straight lines, curves, zigzags, curlicues, or anything else that strikes your fancy. In other words, if the dots represent samples of a wave, there isn't necessarily just one unique wave that can be reconstructed from these samples.

But suppose we lay down two simple rules:

1. *No sharp corners are allowed.* This automatically rules out using straight lines to connect dots, because this would always leave a corner where two such lines meet at a dot (unless three dots are lined up in a line, which is unlikely to happen very often.)

So this rule means that adjacent dots can only be connected with curves. But it also puts a constraint on the curves. When two curves meet at a dot, no sharp corner is allowed between them either — they must blend smoothly into each other without making a corner. This still leaves a lot of possible curves that would fit, so we make one more rule:

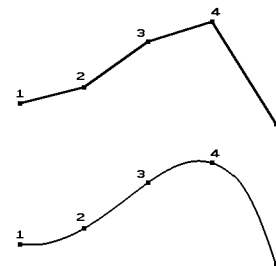


Fig. 14-9. Connecting the (sample) dots

2. *No sharp bends.* Any curve connecting two dots must be the “least curvy” curve that fits rule 1. In other words, only the smoothest curves are allowed, curves which bend as little as possible. Any bends in a curve must have the largest possible radius.

With these two rules in effect, we suddenly discover that there is only one possible way to connect all the dots, similar to the bottom trace in Fig. 14-9. This unique connection is the analog signal that would be reconstructed from the samples.

How do we enforce these two rules? We note that waves that contain a lot of tight curves or even corners contain very high frequency components (think of the square wave, which has right-angle corners and harmonics that extend way up to infinity.) The Nyquist theorem, by saying that the highest frequency in the analog signal must be less than half of the sampling rate, puts a limit on high frequencies.

In a very concise way, the Nyquist theorem essentially says this: An analog signal that has sharp bends also has high frequency components. This means that the sampling rate must be very fast — twice as fast as the highest frequency component in the wave. When you sample this fast, then the samples are so close together that the portion of the waveform that connects any two adjacent dots (samples) is so small that it doesn’t have a chance to do much bending.

If the sampling rate is not fast enough, then reconstructing the original wave from the samples will lead to a type of error called *aliasing*. Fig. 14-10 shows an example; curve A is the original wave being sampled, but the samples are not taken fast enough (i.e., there aren’t more than two samples in each cycle of the waveform). Curve B shows that there is another wave that also fits the same samples, and fits them better because it has more gentle curves and bends. Thus the reconstructed wave would be B instead of A, and this would lead to very serious errors.

Well-designed sampling systems thus always contain *anti-aliasing filters*, which are supposed to remove any analog signal whose frequency is more than half of the sampling frequency.

But filters are never perfect. In a CD recorder, for example, a filter which would remove *everything* above 20,000 Hz would also remove *some* of the signal below

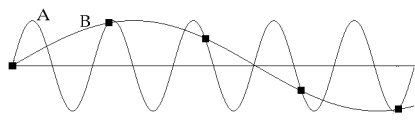


Fig. 14-10. Aliasing error

20,000 Hz, thereby reducing the frequency response of the CD. This explains why CDs are recorded with a sampling frequency of 44,100 Hz, instead of just 40,000 Hz. This gives an extra margin of safety, allowing the anti-aliasing filters to remove almost all signals above 44,100/2, or 22,050 Hz, yet retain almost all of the desired audio signals below 20,000 Hz.

Sampling accuracy

In addition to sampling an analog signal often enough, we must also sample it accurately. Here comes the rub — although the output of the sample-and-hold circuit is a (presumably) exact value at the instant of sampling, this output is then digitized by an A-to-D converter which only recognizes a limited number of voltages. This immediately introduces an error.

As we have seen, the A-to-D converter is set up to accept inputs over a range from some V_{\min} to some V_{\max} , which presumably matches the voltages of the incoming analog signal. If the converter uses n bits, it then divides that range into 2^n steps of size

$$\frac{V_{\max} - V_{\min}}{2^n}$$

If the A-to-D converter assigns the incoming voltage to the nearest step, it might make an error as big as half of the step size; if it takes a shortcut and always assigns it to the next smaller step, or the next bigger step, then the error might be as big as one whole step size. Either way, it makes an error.

The number of bits we use then has to be a compromise. We want to use a lot of bits per sample to reduce the error, but we want to use few bits per sample to cut costs and reduce the total number of bits we have to store or send.

Let’s use the compact disc as an example. CD specifications often list the signal-to-noise ratio as about 96 dB. This means that any noise, such as what might be introduced by the errors in digitizing and reconstructing the signal from the samples, should be 96 dB weaker than the loudest music to be recorded. (Signal-to-noise measurements always use the loudest signal so as to give the best numbers.)

Remembering the formula for calculating dB from a voltage ratio, we note that

$$20 \log \frac{\text{signal}}{\text{noise}} = 96$$

Solving this for the signal-to-noise numerical ratio, we get about 63,100. Hence, if the music is to be 96 dB louder than the noise, then it must have about 63,100 times as much voltage. We will see shortly that the actual ratio used is 65,536, giving an actual signal-to-noise ratio of about 96.3 dB. In other words, any errors intro-

duced by the sampling must be smaller than about $\frac{1}{65536}$ of the maximum voltage. This implies that any voltage measurements in sampling the music must be accurate to at least one part out of 65,536 or so. That in turn says that our analog-to-digital converter must divide the range from the minimum voltage to the maximum voltage into at least 65,536 steps. Since 65,536 is 2^{16} , that says that the A-to-D conversion must use at least 16 bits.

There is another way to look at this. Suppose we used an n -bit analog-to-digital converter and got a signal-to-noise ratio of X . Using one more bit in the A-to-D converter would give twice as many steps; this would cut the step size in half; this would cut the error in half; this would double the signal-to-noise ratio to $2X$; this would increase the signal-to-noise voltage ratio by 6 dB. So every bit in the A-to-D process gives us 6 dB improvement. Hence 16 bits would give us about 16×6 dB. Which is exactly what we have on a compact disc — 16 bits give about 96 dB signal-to-noise ratio.

PCM: Pulse Code Modulation

The system we have been describing in the previous paragraphs is called PCM or *Pulse Code Modulation*. To review: we sample an analog signal (using the Nyquist sampling theorem to tell us how often to sample, and the desired signal-to-noise ratio to tell us how many bits to use per sample), convert that sample into a code — usually a binary number — and store or transmit those numbers, rather than sending the analog waveform itself. Although there can be some errors in the conversion process itself, once the conversion to a digital code is done, that code can then be stored or transmitted with absolutely no additional errors (as long as we are careful.)

The modern telephone network is a prime example. Before the 1970's, transmission over the long-distance telephone network was analog. Cross-country connections were noticeably noisier and more distorted than speaking with someone across town. Today, your voice is digitized as soon as it enters your local telephone company central office (and sometimes even sooner!) It is then carried as a digital signal almost all the way to its

destination. The distance it travels is unimportant, because the data arrives at its destination exactly as it left your town. You can no longer tell the difference between a local call and a long-distance call just by the quality of the connection (and, despite what some telephone carriers will tell you, it has nothing to do with whether the connection is through a fiber cable.)

Fig. 14-11 shows how PCM is used in a CD recorder and CD player. As the analog music signal comes in, a *sample-and-hold* circuit grabs samples of the input signal's voltage at the sampling rate. It then holds that voltage constant (even while the signal itself is changing) long enough for an A-to-D converter to measure the voltage and convert it to a binary number. This process is called *quantizing*. The resulting binary number is sent through a communications channel (or, in the case of a CD, recorded on the disc). Eventually, it goes back into a D-to-A converter, which converts it back into an analog signal. But this signal has jagged edges because it has been quantized. Fortunately, the jagged edges represent frequencies above one-half of the sampling frequency (and therefore above even the highest frequency of the desired signal), so they can safely be filtered out without removing any desired signal. The result is then the original audio.

The ADC cannot, however, capture the exact value. When it quantizes the voltage, it can only express it to the nearest allowable digital number, and so it generates a *quantization error*. In the case of audio, that quantization error sounds like noise which should not be there.

To clarify the errors, look at Fig. 14-12 and suppose that we have a converter which has a three-bit output; it can therefore detect eight (2^3) different voltage levels.

At the top of the figure, we have the same analog waveform we have seen in previous figures. The tick marks show us when the sample-and-hold circuit and the A-to-D converter sample the input signal. At each tick mark, the ADC looks at the analog signal, and assigns it (quantizes it) to the nearest of the eight voltages it can distinguish. (Most converters actually use the *next lower* value, rather than the *nearest* value.) These eight levels are labelled with their binary equivalents, from 000 for the lowest to 111 for the highest. The output

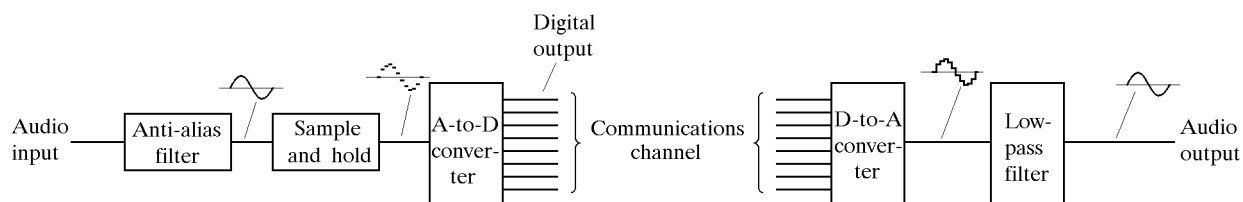


Fig. 14-11. PCM as used to transmit analog signals

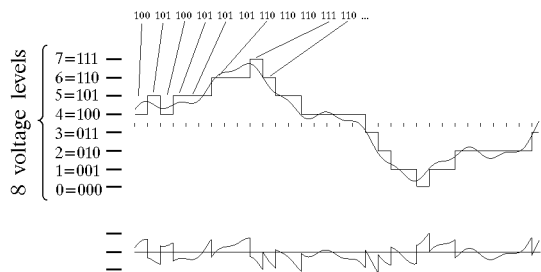


Fig. 14-12. Calculating the error in PCM coding

from the ADC would therefore be the series of numbers 100, 101, 100, 101, 101, ... as shown at the top of Fig. 14-12.

In other words, the gradual changes in the input voltage that occur between samples are ignored; the ADC only sees the signal as it exists at the instant it is sampled. As far as the ADC is concerned, the signal might as well be the squared-up wave that is shown superimposed over the analog signal. When the digital signal is eventually converted back to analog, we will get this squared-up signal, but the filter circuit in the output will round off the sharp corners.

The problem, of course, is that the squared-up wave (even after it is rounded off again at the end) is not quite the same as the original analog signal, and this introduces an error.

The bottom curve in Fig. 14-12 plots the difference between the original analog signal, and the squared-off quantized signal that the ADC thinks it sees — this is the noise or distortion. Let's try to compute its size in terms of the voltage steps that the ADC can detect.

The maximum peak-to-peak amplitude of the original analog signal is 8 divisions. The error signal at the bottom of Fig. 14-12 has a maximum peak-to-peak amplitude of 2 divisions. But notice that this signal has very sharp tips at its maximum and minimum points; after the filtering that occurs at the end, those peaks will be rounded off, and the peak-to-peak amplitude of the error will be less than one division peak-to-peak.

We now have a peak-to-peak desired signal level of 8 divisions, and a peak-to-peak noise level of roughly 1 division (or less). The signal-to-noise ratio is then about 8-to-1; which works out to

$$20 \log \frac{8}{1} = 20 \times 0.9 = 18 \text{ dB}$$

In this case, we had 3-bit numbers giving us 18 dB signal-to-noise ratio, so we again see that each bit gives about 6 dB of signal-to-noise ratio.

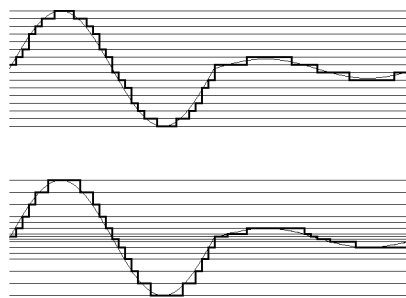


Fig. 14-13. Variable step size A-to-D encoding

A-law and μ -law compression

In the telephone network, PCM is used for sending your voice through the network. Since the audio frequency range of telephone systems is about 300 to about 3400 Hz, sampling is done 8000 times per second. Each sample consists of 8 bits, so the overall data rate for telephone PCM is 64,000 bits per second.

8-bit audio gives about 48 dB signal-to-noise ratio. Unfortunately, this is not really enough for good voice quality. It would be good enough if we all spoke clearly and at maximum volume, but people don't. They often cradle the handset mouthpiece under their chin, or talk softly to avoid disturbing others in the room. The result is that there may be much less than 48 dB difference between their average speech and the noise, perhaps as little as 20 or 30 dB.

The telephone system could get around this by using more bits to increase the signal-to-noise ratio, but this would increase the overall data rate significantly and raise the cost. Telephone companies therefore use compression schemes called A-law and μ -law compression; they are specially designed for digitized voice to reduce the apparent distortion and noise without increasing the number of bits. μ -law is the standard in the U.S., Japan, and several other countries, while A-law is used more in Europe. The two systems are similar, but differ in some details.

With even spacing between the quantization steps, which we have used so far in this chapter and show in the top curve in Fig. 14-13 (which shows only sixteen steps, though normal telephone audio uses 256 steps), loud sounds (those having a large peak-to-peak value, and shown at the left) are quantized with far more steps (and therefore much less distortion) than weak sounds (shown at the right). This is the opposite of what we normally get with analog methods, where softer sounds tend to be clearer and less distorted than loud sounds, and it sounds unnatural.

The solution is to change the spacing between quantization steps so that the steps in the middle are closer

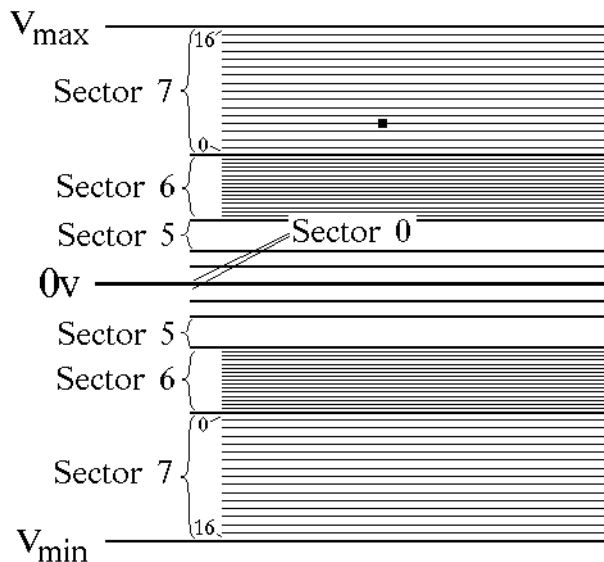


Fig. 14-14. u-Law encoding

together. A small waveform will now be quantized with finer steps, and its shape better preserved than before. The bottom curve in Fig. 14-13 shows the effect. There is a bit more error in the large wave at the left, but this is outweighed by the better accuracy for the smaller wave at the right.

Fig. 14-14 shows how encoding is done. μ -Law compression encodes each sample as an 8-bit number like this:

1-bit sign	3-bit Sector number	4-bit step within that sector
---------------	------------------------	----------------------------------

The first bit denotes the sign: 0 for minus, 1 for plus. For example, the large square dot in Fig. 14-14 is in + area, so its first bit would be a 1.

The next three bits denote a sector number. The range from V_{\min} to V_{\max} is divided into 16 sectors, 8 in the negative region, and 8 in the positive region, numbered from sector 0 through sector 7. Let's just look at the ones in the + region.

- The entire top half of the positive range is sector number 7.
- Half of what's left under it is sector 6.
- Half of what's left under that is sector 5.
- Half of what's left under that is sector 4, and so on.

Thus sector 7 is big, sector 6 is half as big, sector 5 is half of that, and so on, until sector 0, squeezed down at the very bottom of the + region, is tiny. The same thing, except upside down, is done in the negative region.

The large square dot in Fig. 14-14 is in sector 7, so its binary sector number is 111.

Finally, each of the sectors is divided into 16 steps. Since sector 7 is fairly large, its steps are big; the steps in sector 6 are half as big, and so on, until the steps in sector 0 are tiny.

The square dot in Fig. 14-14 is on step 4, so its code is 11110100 — 1 for plus, sector 111, step 0100.

This method of compressing the data works quite well for telephone-quality voice signals, and makes a very noticeable difference in audio quality. In fact, μ -law compression makes 8 bits sound almost as good as 13 or so bits of uncompressed voice-quality audio (though it would still not be suitable for music or other high-quality applications).

One last, but very important note: Zero volts can be considered either *plus*, sector 0, step 0 (which would be coded 10000000) or *minus*, sector 0, step 0 (whose code would be 00000000). But there is no need for two different codes for the same voltage, so it is agreed that zero volts will always be 10000000, *never* 00000000. This will become very important when we consider the T1 system widely used by telephone companies, because it means that the code 00000000 will never occur in digitized voice signals.

DPCM: Differential PCM

Differential Pulse Code Modulation or DPCM is another way of reducing the number of bits required to carry the information without actually changing the accuracy.

The idea hinges on the fact that, for most analog waveforms, any two successive samples are usually fairly similar. In other words, the difference between any two successive samples is fairly small — and generally smaller than the overall size of the waveform. Hence the number of bits needed to specify the difference between two samples is smaller than the number of bits needed to give the entire value of each sample.

For example, in Fig. 14-12, we saw that the first few quantized steps in binary were 100, 101, 100, 101, 101, 101, 110, 110, 110, 111, 110, etc. In decimal, these are the numbers 4, 5, 4, 5, 5, 5, 6, 6, 6, 7, 6, and so on. To store these numbers, we would need three bits for each.

With DPCM, we can save bits by writing down only the *differences* between consecutive numbers. Once we know the first value (4), we can specify the others by noting that the second value (5) is one more; the next one (4) is one less (less than the previous value of 5), and so on. So we only need the sequence

+1 -1 +1 0 0 +1 0 0 +1 -1

which would require fewer bits than expressing the entire sequence 100, 101, 100, etc.

As we've already seen, the system only works if successive samples are fairly similar — since there

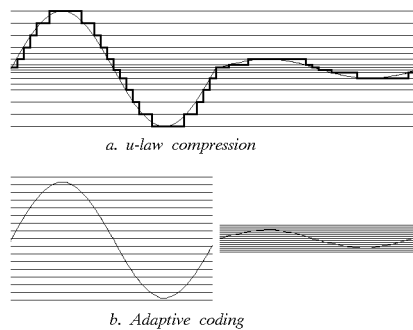


Fig. 14-15. Comparison of adaptive and u-law step sizes

aren't big jumps from one sample to another, only a few bits are needed to specify the difference. The system breaks down if there are many sudden jumps between samples, but this doesn't generally happen with speech or with most audio.

APCM and ADPCM

The A in APCM and ADPCM means *Adaptive*. What it means is that the step size changes to adapt to the signal. This sounds a bit like A-law or μ -law compression, but it's actually very different. Let's explain the difference with the help of Fig. 14-15.

A-law and μ -law compression use different step sizes at the same time, as in Fig. 14-15a. The step sizes are different, but they don't change with time. Different parts of the same sine wave cycle are digitized with different step sizes.

In adaptive coding, all the steps are the same size at any given instant, but the step size varies with the average size of the signal. When the signal is large, as in the left cycle in Fig. 14-15b, the step size is large too. When the signal gets small, as in the right cycle, the step size drops too. The change in step sizes may occur relatively quickly (for example, it might change on each syllable you speak) or it might only change infrequently.

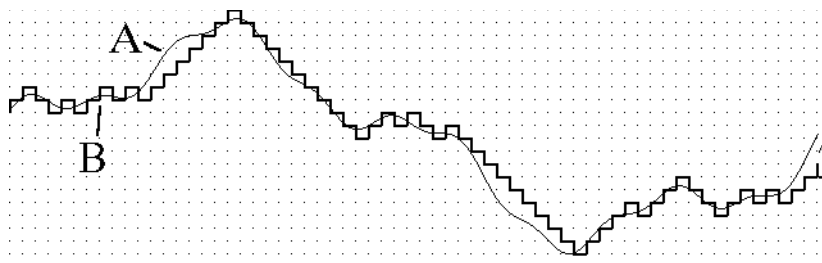


Fig. 14-16. Delta modulation

Delta Modulation

Delta modulation is an extreme case of differential PCM. In delta modulation, the sampling rate is much higher than required by the Nyquist theorem. We use the term *oversampling* to describe it — for example, a system where the sampling rate is 16 times higher than the minimum required would be called “sixteen times oversampling.”

At first glance, this sounds bad: taking more samples than necessary implies more bits than necessary. The difference, however, comes in the number of bits needed per sample. If you sample fast enough, the difference between successive samples becomes so small that there is at most one step difference between any two successive samples. Then you need only one bit to express that difference.

The previous paragraph is not entirely correct; it's a bit over-simplified. Actually, we *need slightly more* than one bit to express the difference between two successive samples. The reason is this: A given sample measurement could be either (1) larger than the previous sample, (2) smaller than the previous sample, or (3) the same as the previous sample. One bit can only express two possible conditions, not three, so you need more than one bit.

Delta modulation, however, ignores the third possibility — the case where one sample is the same as the previous one. When there are several samples of the same value, instead of coding them as “same, same, same, same”, delta modulation introduces a slight wiggle into the waveform by coding them as “bigger, smaller, bigger, smaller”. By using a 1 to mean “bigger” and a 0 to mean “smaller”, delta modulation gets by with just one bit per sample.

Fig. 14-16 shows an example, illustrating two kinds of possible errors. At A, we see that the analog signal is rising faster than the digital approximation can follow. This can happen if the system does not sufficiently oversample. This error could be reduced by increasing the sample rate (which, unfortunately, requires more bits) or increasing the step size (which, unfortunately, would reduce the signal-to-noise ratio.)

At B, we see that the analog signal is almost constant. Ideally, the digitized signal should also remain constant. But because the bit coding in delta modulation requires that the digitized waveform cannot stay constant — it must go either up or down — this introduces a slight “wiggle” which adds a slight amount of noise into

the signal. Fortunately, with oversampling the sampling frequency is much higher than the highest signal frequency (for example, with 16 times oversampling, the sampling frequency would be over 32 times higher than the highest signal frequency). The frequency of this “wiggle” is therefore way above the signal range, and so it can easily be filtered out with a low-pass filter.

On the surface, delta modulation seems not to offer any advantages. For example, consider an 8-bit PCM system. By using 16-times oversampling, you would reduce the number of bits per sample from 8 to 1, but you increase the number of samples by 16. That hardly seems an advantage.

Yet delta modulation has one big feature: it greatly simplifies the hardware, and this drops the price. Fig. 14-17 shows a very simplified diagram of a one-bit delta modulation A-to-D converter.

The analog input is compared with the voltage on a capacitor by a comparator. If the analog input is bigger, the comparator outputs a 1; if it is smaller, it outputs a 0.

Besides being the output data, this bit also controls a charge pump. At each clock pulse, the charge pump sends a jolt of charge into the capacitor: if the output bit is a 1, the charge is positive and it makes the capacitor voltage rise one step. If the output bit is a 0, the charge is negative and it makes the capacitor voltage go down one step.

The capacitor voltage therefore follows the analog input voltage, going up and down in steps just like the heavy lines in Fig. 14-16.

Delta-Sigma converters

Although delta modulation may require more bits than plain PCM, conversion between delta modulation and PCM is easy — all you need do is count the bits. The result is called Delta-Sigma or $\Delta \Sigma$ conversion, and the name comes from the fact that mathematicians use the Greek letter Delta (Δ) to signify the change in a number, and the letter Sigma (Σ) for summing or counting. $\Delta \Sigma$ analog-to-digital converters are currently a very popular circuit (and are sometimes called Sigma-Delta rather than Delta-Sigma converters; but it’s the same thing.)

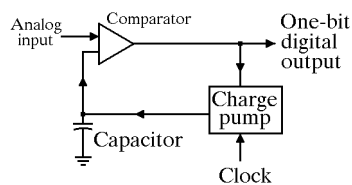


Fig. 14-17. One-bit delta modulation A-to-D converter

Let’s explain the concept by giving an example: Suppose an input signal contains frequencies up to almost 500 Hz; the Nyquist theorem therefore states that the minimum sample rate is 1000 samples per second. Further, suppose that you want a 48 dB signal-to-noise ratio, so you need 8-bit samples.

Using Delta-Sigma conversion, you decide to use 16-times oversampling. so you will sample 16,000 times per second. Each $1/16,000$ of a second, you will have a one-bit number from your delta modulation circuit. In $16/16,000$ of a second (which is $1/1000$ of a second) you will have 16 bits, and it is now time to produce one eight-bit PCM code. Suppose you got these 16 consecutive bits:

1111101101100010

You note that there are ten ONES and four ZEROS, so the signal went up 10 and down 4. It is therefore 6 steps up from where it started, so your first 8-bit PCM code is 00001110 (which is binary for 6.)

If the next set of 16 bits is

0000101010111111

you note that there are nine ONES and seven ZEROS, so the signal went up 9 and down 7, a net gain of 2 steps up. Since you were at step 6 before, you are now at step 8, and so the 8-bit PCM code is 00001000.

This pulse counting can be done with relatively simple hardware, and so the Delta-Sigma conversion is very economical. Integrated circuits which do 16-, 20-, and even 24-bit $\Delta \Sigma$ conversion at reasonable speed are now readily available. And a similar approach can also be used for conversion from digital back to analog.

What is interesting is that oversampling and $\Delta \Sigma$ conversion is not only more economical than other approaches, but also has some technical advantages and offers improved performance. Hence many manufacturers tout “1-bit converters” and “oversampling” as a way of *raising* the price of their equipment.

The technical advantages come in two forms. As we mentioned earlier, an A-to-D converter needs an anti-aliasing filter to precede it. Since a $\Delta \Sigma$ converter samples at a much higher frequency, it is much easier and cheaper to design an appropriate filter for it.

A second advantage comes from the fact that the $\Delta \Sigma$ converter has much less noise. Remember that quantizing error always introduces a bit of noise. Consider a CD with a 44.1 kHz sampling rate — its noise will be fairly evenly distributed over its entire frequency range from 0 to 22 kHz. Almost all of this noise is therefore in the audible range.

But with 16× oversampling, the sampling frequency is raised to 16×44.1 kHz, or about 700 kHz. Its quantization noise is spread out over the entire range from 0 to 350 kHz (half of the sampling rate). But most of this is way above the audible range and can easily be filtered out.