CHAPTER 5

Basics of Embedded Audio Processing

Introduction

Audio functionality plays a critical role in embedded media processing. While audio takes less processing power in general than video processing, it should be considered equally important.

In this chapter, we will begin with a discussion of sound and audio signals, and then explore how data is presented to the processor from a variety of audio converters. We will also describe the formats in which audio data is stored and processed.

Additionally, we'll discuss some software building blocks for embedded audio systems. Efficient data movement is essential, so we will examine data buffering as it applies to audio algorithms. Finally, we'll cover some fundamental algorithms and finish with a discussion of audio and speech compression.

What Is Sound?

Sound is a longitudinal displacement wave that propagates through air or some other medium. Sound waves are defined using two attributes: amplitude and frequency.

The amplitude of a sound wave is a gauge of pressure change, measured in decibels (dB). The lowest sound amplitude that the human ear can perceive is called the "threshold of hearing," denoted by 0 dBSPL. On this SPL (sound pressure level) scale, the reference pressure is defined as 20 micropascals (20 μPa). The general equation for dBSPL, given a pressure change *x*, is

$$\text{dBSPL} = 20 \times \log (x\ \mu\text{Pa} / 20\ \mu\text{Pa})$$

Table 5.1 shows decibel levels for typical sounds. These are all relative to the threshold of hearing (0 dBSPL).

Table 5.1 **Decibel (dBSPL) values for various typical sounds**

Source (distance)	dBSPL
Threshold of hearing	0
Normal conversation (3-5 feet away)	60-70
Busy traffic	70-80
Loud factory	90
Power saw	110
Discomfort	120
Threshold of pain	130
Jet engine (100 feet away)	150

The main point to take away from Table 5.1 is that the range of tolerable audible sounds is about 120 dB (when used to describe ratios without reference to a specific value, the correct notation is dB without the SPL suffix). Therefore, all engineered audio systems can use 120 dB as the upper bound of dynamic range. In case you're wondering why all this is relevant to embedded systems, don't worry—we'll soon relate dynamic range to data formats for embedded media processing.

Frequency, the other key feature of sound, is denoted in Hertz (Hz), or cycles per second. We can hear sounds in the frequency range between 20 and 20,000 Hz, but this ability degrades as we age.

Our ears can hear certain frequencies better than others. In fact, we are most sensitive to frequencies in the area of 2–4 kHz. There are other quirky features about the ear that engineers are quick to exploit. Two useful phenomena, employed in the lossy compression algorithms that we'll describe later, are *temporal masking* and *frequency masking*. In temporal masking (Figure 5.1a), loud tones can drown out softer tones that occur at almost the same time. Frequency masking (Figure 5.1b) occurs when a loud sound at a certain frequency renders softer sounds at nearby frequencies inaudible. The human ear is such a complex organ that only books dedicated to the subject can do it justice. For a more in-depth survey of ear physiology, consult Reference 23 in the Appendix.

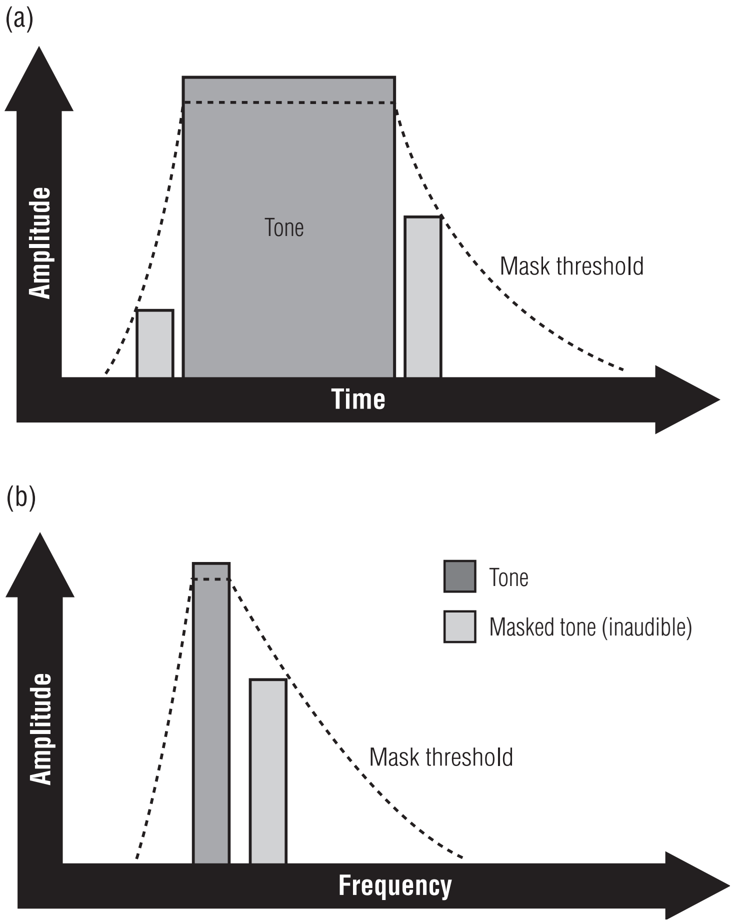


Figure 5.1 **(a) Loud sounds at a specific time can mask out softer sounds in the temporal vicinity (b) Loud sounds at a specific frequency can mask softer sounds at nearby frequencies**

Audio Signals

In order to create an analog signal that represents a sound wave, we must use a transducer to convert the mechanical pressure energy into electrical energy. A more common name for this audio source transducer is a microphone.

All transducers can be described with a sensitivity (or transduction) curve. In the case of a microphone, this curve dictates how well it translates pressure into an electrical signal. Ideal transducers have a linear sensitivity curve—that is, a voltage level is directly proportional to a sound wave's pressure.

Since a microphone converts a sound wave into voltage levels, we now need to use a new decibel scale to describe amplitude. This scale, called dBV, is based on a reference point of 1V. The equation describing the relationship between a voltage level x and dBV is

$$\text{dBV} = 20 \times \log (x \text{ volts} / 1.0 \text{ volts})$$

An alternative analog decibel scale is based on a reference of 0.775V and uses dBu units.

In order to create an audible mechanical sound wave from an analog electrical signal, we again need to use a transducer. In this case, the transducer is a speaker or headset.

Speech Processing

Speech processing is an important and complex class of audio processing, and we won't delve too deeply here. However, we'll discuss speech-processing techniques where they are analogous to the more general audio processing methods. The most common use for speech signal processing is in voice telecommunications for such algorithms as echo cancellation and compression.

Most of the energy in typical speech signals is stored within less than 4 kHz of bandwidth, thus making speech a subset of audio signals. However, many speech-processing techniques are based on modeling the human vocal tract, so these cannot be used for general audio processing.

Audio Sources and Sinks

Converting Between Analog and Digital Audio Signals

Assuming we've already taken care of converting sound energy into electrical energy, the next step is to digitize the analog signals. This is accomplished with an analog-to-digital converter (A/D converter or ADC). As you might expect, in order to create an analog signal from a digital one, a digital-to-analog converter (D/A converter or DAC) is used. Since many audio systems are really meant for a full-duplex media flow, the ADC and DAC are available in one package called an "audio codec." The term codec is used here to mean a discrete hardware chip. As we'll discuss later in the section on audio compression, this should not be confused with a software audio codec, which is a software algorithm.

All A/D and D/A conversions should obey the Shannon-Nyquist sampling theorem. In short, this theorem dictates that an analog signal must be sampled at a rate (Nyquist sampling rate) equal to or exceeding twice its highest-frequency component (Nyquist frequency) in order for it to be reconstructed in the eventual D/A conversion. Sampling below the Nyquist sampling rate will introduce aliases, which are low frequency "ghost" images of those frequencies that fall above the Nyquist frequency. If we take a sound signal that is band-limited to 0–20 kHz, and sample it at 2 × 20 kHz = 40 kHz, then the Nyquist Theorem assures us that the original signal can be reconstructed perfectly without any signal loss. However, sampling this 0–20 kHz band-limited signal at anything less than 40 kHz will introduce distortions due to aliasing. Figure 5.2 shows how sampling at less than the Nyquist sampling rate results in an incorrect representation of a signal. When sampled at 40 kHz, a 20 kHz signal is represented correctly (Figure 5.2a). However, the same 20 kHz sine wave that is sampled at a 30 kHz sampling rate actually looks like a lower frequency alias of the original sine wave (Figure 5.2b).

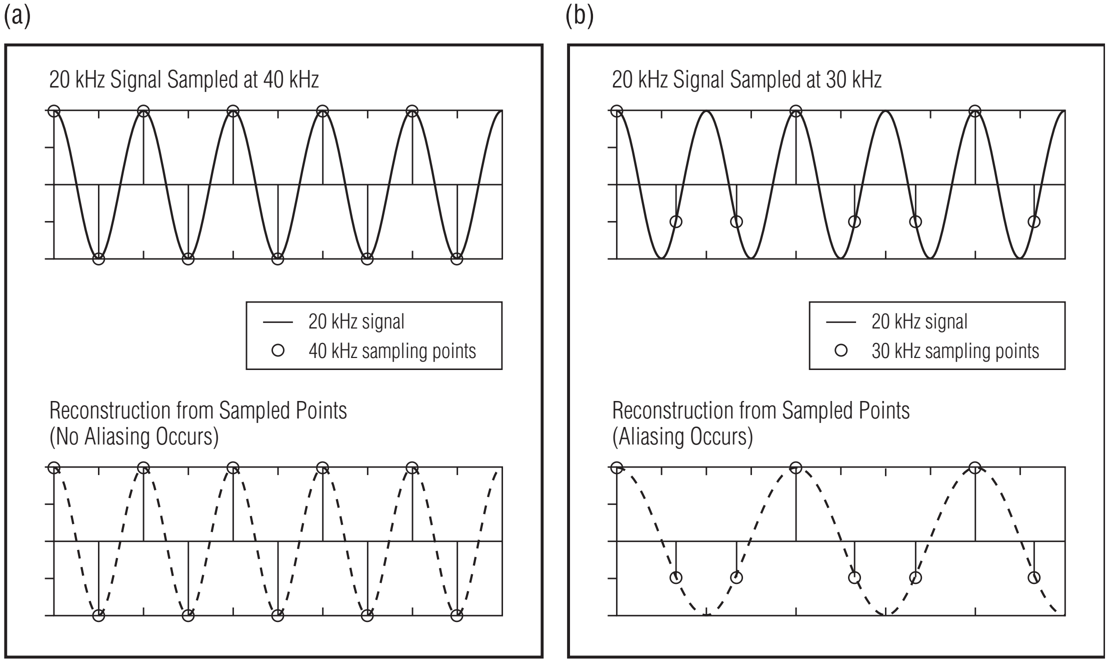


Figure 5.2 **(a) Sampling a 20 kHz signal at 40 kHz captures the original signal correctly (b) Sampling the same 20 kHz signal at 30 kHz captures an aliased (low frequency ghost) signal**

No practical system will sample at exactly twice the Nyquist frequency, however. For example, restricting a signal into a specific band requires an analog low-pass filter, but these filters are never ideal. Therefore, the lowest sampling rate used to reproduce music is 44.1 kHz, not 40 kHz, and many high-quality systems sample at 48 kHz in order to capture the 20 Hz–20 kHz range of hearing even more faithfully. As we mentioned earlier, speech signals are only a subset of the frequencies we can hear; the energy content below 4 kHz is enough to store an intelligible reproduction of the speech signal. For this reason, telephony applications usually use only 8 kHz sampling (= 2 × 4 kHz). Table 5.2 summarizes some sampling rates used by common systems.

Table 5.2 **Commonly used sampling rates**

System	Sampling Frequency
Telephone	8000 Hz
Compact Disc	44100 Hz
Professional Audio	48000 Hz
DVD Audio	96000 Hz (for 6-channel audio)

The most common digital representation for audio is a pulse-code-modulated (PCM) signal. In this representation, an analog amplitude is encoded with a digital level for each sampling period. The resulting digital wave is a vector of snapshots taken to approximate the input analog wave. All A/D converters have finite resolution, so they introduce quantization noise that is inherent in digital audio systems. Figure 5.3 shows a PCM representation of an analog sine wave (Figure 5.3a) converted using an ideal A/D converter, in which the quantization manifests itself as the "staircase effect" (Figure 5.3b). You can see that lower resolution leads to a worse representation of the original wave (Figure 5.3c).

For a numerical example, let's assume that a 24-bit A/D converter is used to sample an analog signal whose range is –2.828V to 2.828V (5.656 Vpp). The 24 bits allow for 2^{24} (16,777,216) quantization levels. Therefore, the effective voltage resolution is 5.656V / 16,777,216 = 337.1 nV. Shortly, we'll see how codec resolution affects the dynamic range of audio systems.

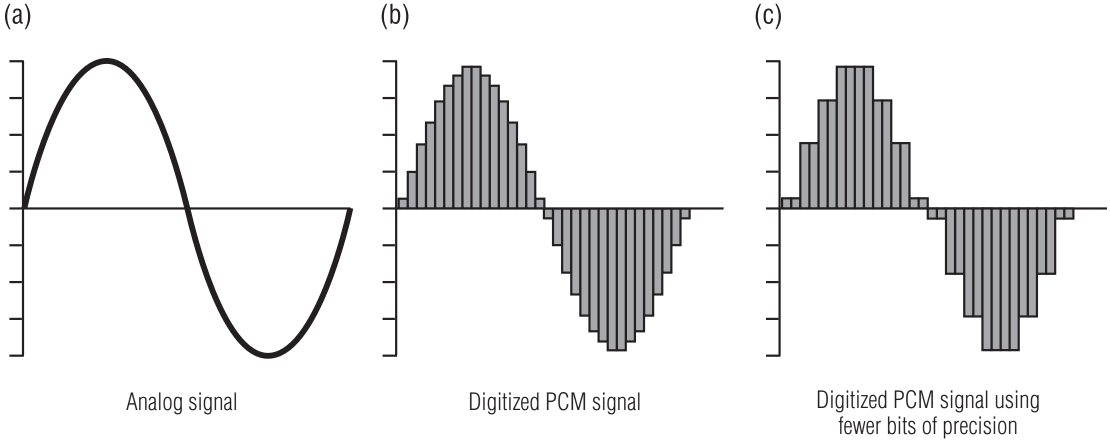


Figure 5.3 **(a) An analog signal (b) Digitized PCM signal (c) Digitized PCM signal using fewer bits of precision**

Background on Audio Converters

Audio ADCs

There are many ways to perform A/D conversion. One traditional approach is a successive approximation scheme, which uses a comparator to test the analog input signal against a number of interim D/A conversions to arrive at the final answer.

Most audio ADCs today, however, are sigma-delta converters. Instead of employing successive approximations to create wide resolutions, sigma-delta converters use 1-bit ADCs. In order to compensate for the reduced number of quantization steps, they are oversampled at a frequency much higher than the Nyquist frequency. Conversion from this super-sampled 1-bit stream into a slower, higher-resolution stream is performed using digital filtering blocks inside these converters, in order to accommodate the more traditional PCM stream processing. For example, a 16-bit 44.1 kHz sigma-delta ADC might oversample at 64x, yielding a 1-bit stream at a rate of 2.8224 MHz. A digital decimation filter (described in more detail later) converts this super-sampled stream to a 16-bit one at 44.1 kHz.

Because they oversample analog signals, sigma-delta ADCs relax the performance requirements of the analog low-pass filters that band-limit input signals. They also have the advantage of reducing peak noise by spreading it over a wider spectrum than traditional converters.

Audio DACs

Just as in the A/D case, sigma-delta designs rule the D/A conversion space. They can take a 16-bit 44.1 kHz signal and convert into a 1-bit 2.8224 MHz stream using an interpolating filter (described later). The 1-bit DAC then converts the super-sampled stream to an analog signal.

A typical embedded digital audio system may employ a sigma-delta audio ADC and a sigma-delta DAC, and therefore the conversion between a PCM signal and an oversampled stream is done twice. For this reason, Sony and Philips have introduced an alternative to PCM, called Direct-Stream Digital (DSD), in their Super Audio CD (SACD) format. This format stores data using the 1-bit high-frequency (2.8224 MHz) sigma-delta stream, bypassing the PCM conversion. The disadvantage is that DSD streams are less intuitive to process than PCM, and they require a separate set of digital audio algorithms, so we will focus only on PCM in this chapter.

Connecting to Audio Converters

An ADC Example

OK, enough background information. Let's do some engineering now. One good choice for a low-cost audio ADC is the Analog Devices AD1871, which features 24-bit conversion at 96 kHz. The functional block diagram of the AD1871 is shown in Figure 5.4a. This converter has left (VINLx) and right (VINRx) input channels, which is really just another way of saying that it can handle stereo data. The digitized audio data is streamed out serially through the data port, usually to a corresponding serial port on a signal processor (like the SPORT interface on Blackfin processors). There is also an SPI (serial peripheral interface) port provided for the host processor to configure the AD1871 via software commands. These commands include ways to set the sampling rate, word width, and channel gain and muting, among other parameters.

As the block diagram in Figure 5.4b implies, interfacing the AD1871 ADC to a Blackfin processor is a glueless connection. The analog representation of the circuit is simplified, since only the digital signals are important in this discussion. The oversampling rate of the AD1871 is achieved with an external crystal. The Blackfin processor shown has two serial ports (SPORTs) and an SPI port used for connecting to the AD1871. The SPORT, configured in I^{2}S mode, is the data link to the AD1871, whereas the SPI port acts as the control link.

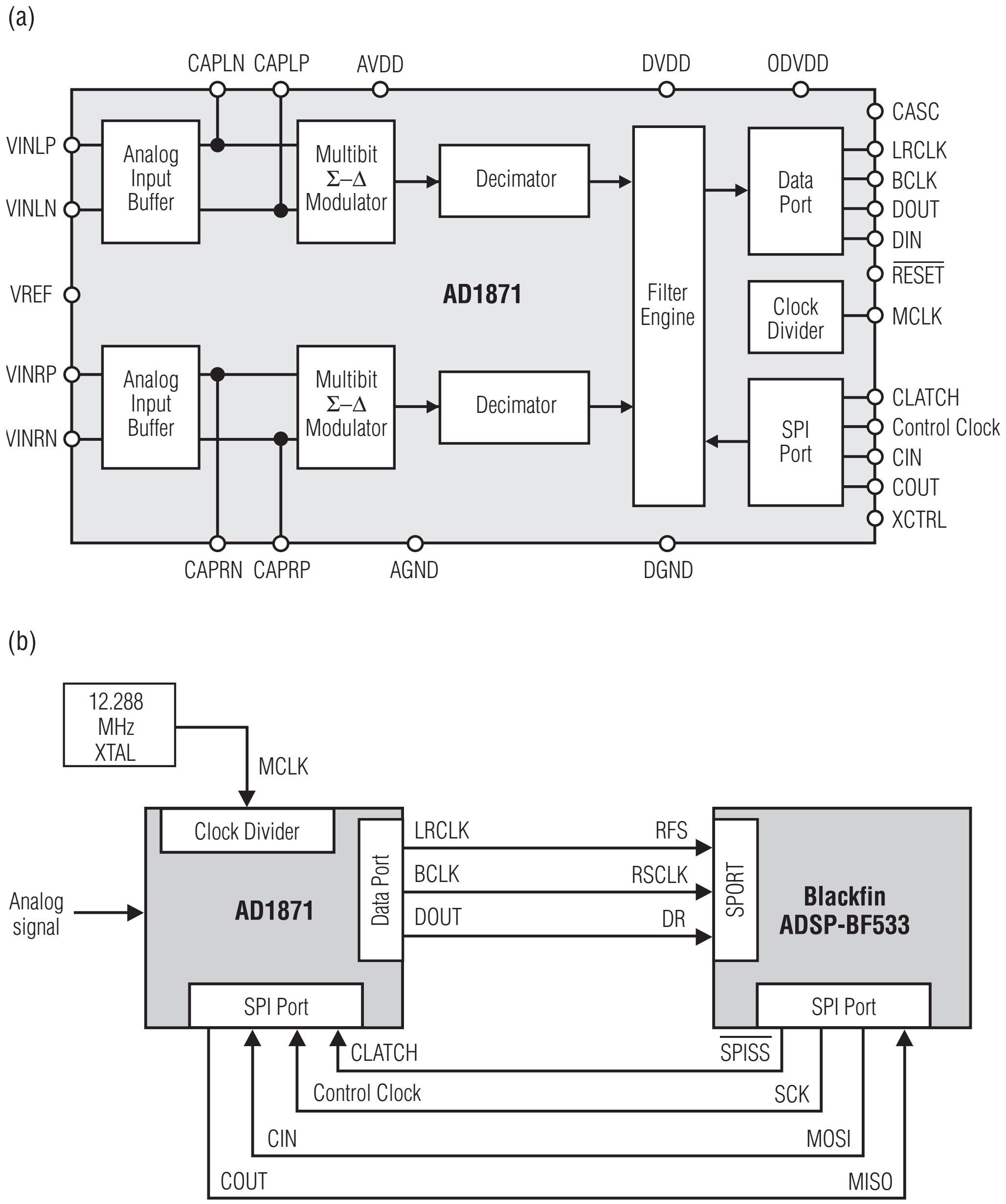


Figure 5.4 **(a) Functional block diagram of the AD1871 audio ADC**
(b) Glueless connection of an ADSP-BF533 media processor to the AD1871

I^{2}S (Inter-IC-Sound)

The I^{2}S protocol is a standard developed by Philips for the digital transmission of audio signals. This standard allows for audio equipment manufacturers to create components that are compatible with each other.

In a nutshell, I^{2}S is simply a three-wire serial interface used to transfer stereo data. As shown in Figure 5.5a, it specifies a bit clock (middle), a data line (bottom), and a left/right synchronization line (top) that selects whether a left or right channel frame is currently being transmitted.

In essence, I^{2}S is a time-division-multiplexed (TDM) serial stream with two active channels. TDM is a method of transferring more than one channel (for example, left and right audio) over one physical link.

In the AD1871 setup of Figure 5.4b, the ADC can use a divided-down version of the 12.288 MHz sampling rate it receives from the external crystal to drive the SPORT clock (`RSCLK`) and frame synchronization (RFS) lines. This configuration insures that the sampling and data transmission are in sync.

SPI (Serial Peripheral Interface)

The SPI interface, shown in Figure 5.5b, was designed by Motorola for connecting host processors to a variety of digital components. The entire interface between an SPI master and an SPI slave consists of a clock line (SCK), two data lines (MOSI and MISO), and a slave select ($\overline{\text{SPISS}}$) line. One of the data lines is driven by the master (MOSI), and the other is driven by the slave (MISO). In the example of Figure 5.4b, the Blackfin processor's SPI port interfaces gluelessly to the SPI block of the AD1871.

Audio codecs with a separate SPI control port allow a host processor to change the ADC settings on the fly. Besides muting and gain control, one of the really useful settings on ADCs like the AD1871 is the ability to place it in power-down mode. For battery-powered applications, this is often an essential function.

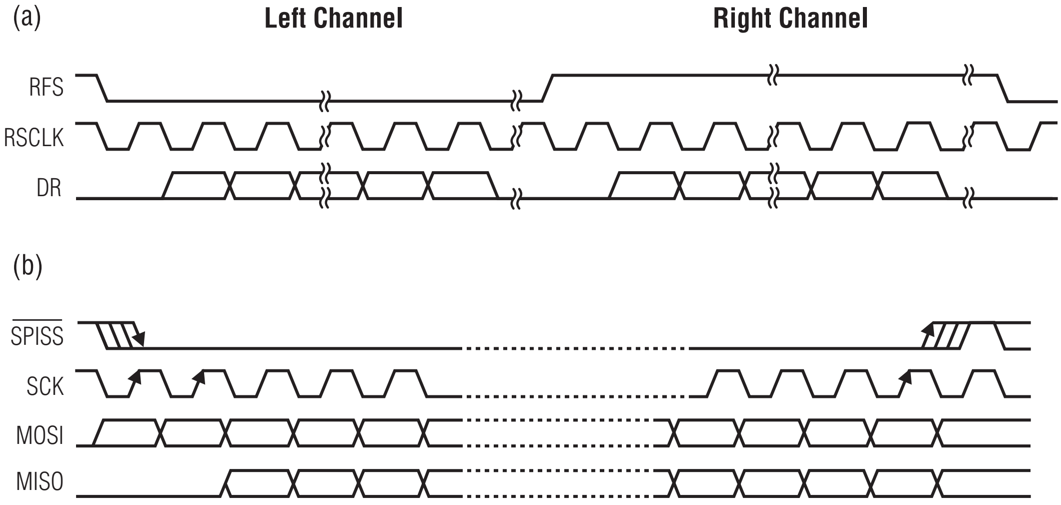


Figure 5.5 **(a) The data signals transmitted by the AD1871 using the I^{2}S protocol (b) The SPI interface used to control the AD1871**

DACs and Codecs

Connecting an audio DAC to a host processor is an identical process to the ADC connection we just discussed. In a system that uses both an ADC and a DAC, the same serial port can hook up to both, if it supports bidirectional transfers.

But if you're tackling full-duplex audio, then you're better off using a single-chip audio codec that handles both the analog-to-digital and digital-to-analog conversions. A good example of such a codec is the Analog Devices AD1836, which features three stereo DACs and two stereo ADCs, and is able to communicate through a number of serial protocols, including I^{2}S.

AC '97 (Audio Codec '97)

I^{2}S is only one audio specification. Another popular one is AC '97, which Intel Corporation created to standardize all PC audio and to separate the analog circuitry from the less-noise-susceptible digital chip. In its simplest form, an AC '97 codec uses a TDM scheme where control and data are interleaved in the same signal. Various timeslots in the serial transfer are reserved for a specific data channel or control word. Most processors with serial ports that support TDM mode can de-multiplex an AC '97 signal at the expense of some software overhead. One example of an AC '97 codec is the AD1847 from Analog Devices.

Speech Codecs

Since speech processing has slightly relaxed requirements compared to hi-fidelity music systems, you may find it worthwhile to look into codecs designed specifically for speech. Among many good choices is the dual-channel 16-bit Analog Devices AD73322, which has a configurable sampling frequency from 8 kHz all the way to 64 kHz.

PWM Output

So far, we've only talked about digital PCM representation and the audio DACs used to get those digital signals to the analog domain. But there is a way to use a different kind of modulation, called pulse-width modulation (PWM), to drive an output circuit directly without any need for a DAC, when a low-cost solution is required.

In PCM, amplitude is encoded for each sample period, whereas it is the duty cycle that describes amplitude in a PWM signal. PWM signals can be generated with general-purpose I/O pins, or they can be driven directly by specialized PWM timers, available on many processors.

To make PWM audio achieve decent quality, the PWM carrier frequency should be at least 12 times the bandwidth of the signal, and the resolution of the timer (i.e., granularity of the duty cycle) should be 16 bits. Because of the carrier frequency requirement, traditional PWM audio circuits were used for low-bandwidth audio, like subwoofers. However, with today's high-speed processors, it's possible to carry a larger audible spectrum.

The PWM stream must be low-pass-filtered to remove the high-frequency carrier. This is usually done in the amplifier circuit that drives a speaker. A class of amplifiers, called Class D, has been used successfully in such a configuration. When amplification is not required, then a low-pass filter is sufficient as the output stage. In some low-cost applications, where sound quality is not as important, the PWM streams can connect directly to a speaker. In such a system, the mechanical inertia of the speaker's cone acts as a low-pass filter to remove the carrier frequency.

Interconnections

Before we end this hardware-centric section, let's review some of the common connectors and interfaces you'll encounter when designing systems with embedded audio capabilities.

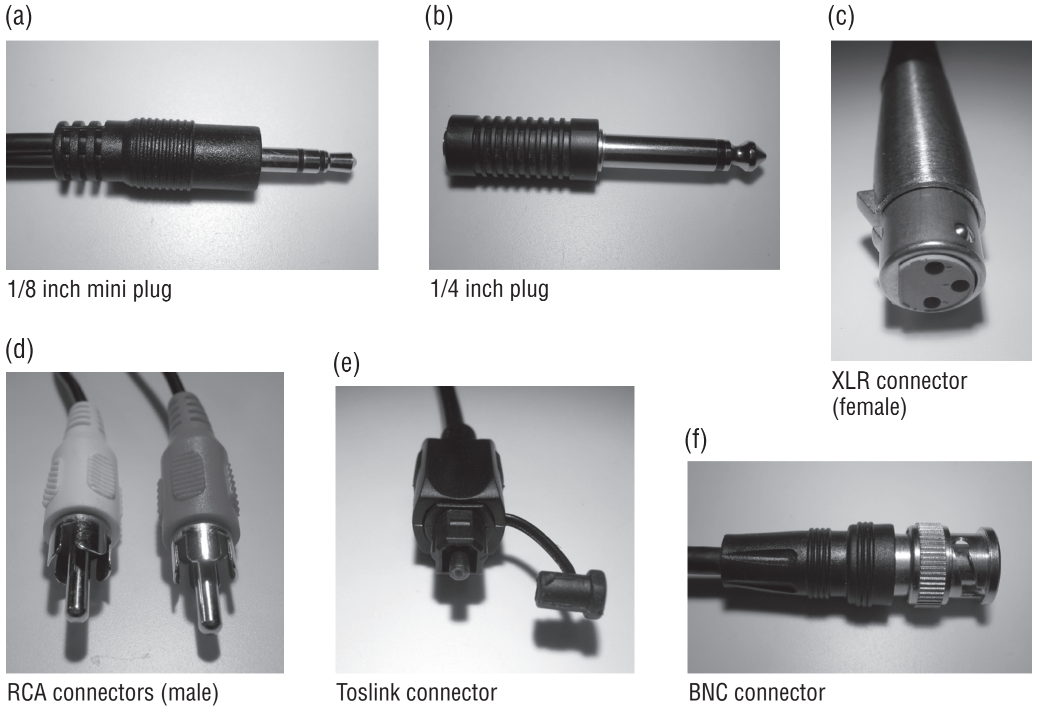
Connectors

Microphones, speakers, and other analog equipment connect to an embedded system through a variety of standard connectors (see Figure 5.6). Because of their small size, 1/8" connectors are quite common for portable systems. Many home stereo components support 1/4" connectors. Higher performance equipment usually uses RCA connectors, or even a coaxial cable connector, to preserve signal integrity.

Digital Connections

Some of the systems you'll design actually won't require any ADCs or DACs, because the input signals may already be digital and the output device may accept digital data. A few standards exist for transfer of digital data from one device to another.

The *S*ony *D*igital *I*nter*F*ace (SDIF-2) protocol is used in some professional products. It requires an unbalanced BNC coaxial connection for each channel. The Audio Engineering Society (AES) introduced the AES3 standard for serial transmission of



1/8 inch mini plug

1/4 inch plug

XLR connector (female)

RCA connectors (male)

Toslink connector

BNC connector

Figure 5.6 **Various audio connectors: (a) 1/8 inch mini plug (b) 1/4 inch plug (c) XLR connector (d) Male RCA connectors (e) Toslink connector (f) BNC connector**

data; this one uses an XLR connector. A more ubiquitous standard, the S/PDIF (Sony/Philips Digital InterFace), is prevalent in consumer and professional audio devices. Two possible S/PDIF connectors are single-ended coaxial cable and the Toslink connector for fiber optic connections.

Dynamic Range and Precision

We promised earlier that we would get into a lot more detail on dynamic range of audio systems. You might have seen dB specs thrown around for various products available on the market today. Table 5.3 lists a few fairly established products along with their assigned signal quality, measured in dB.

Table 5.3 **Dynamic range comparison of various audio systems**

Audio Device	Typical Dynamic Range
AM Radio	48 dB
Analog TV	60 dB
FM Radio	70 dB
16-bit Audio Codecs	90-95 dB
CD Player	92-96 dB
Digital Audio Tape (DAT)	110 dB
20-bit Audio Codecs	110 dB
24-bit Audio Codecs	110-120 dB

So what exactly do those numbers represent? Let's start by getting some definitions down. Use Figure 5.7 as a reference diagram for the following discussion.

As you might remember from the beginning of this chapter, the dynamic range for the human ear (the ratio of the loudest to the quietest signal level) is about 120 dB. In systems where noise is present, dynamic range is described as the ratio of the maximum signal level to the noise floor. In other words,

$$\mathit{Dynamic\ Range}\ (\mathit{dB}) = \mathit{Peak\ Level}\ (\mathit{dB}) - \mathit{Noise\ Floor}\ (\mathit{dB})$$

The noise floor in a purely analog system comes from the electrical properties of the system itself. On top of that, audio signals also acquire noise from ADCs and DACs, including quantization errors due to the sampling of analog data.

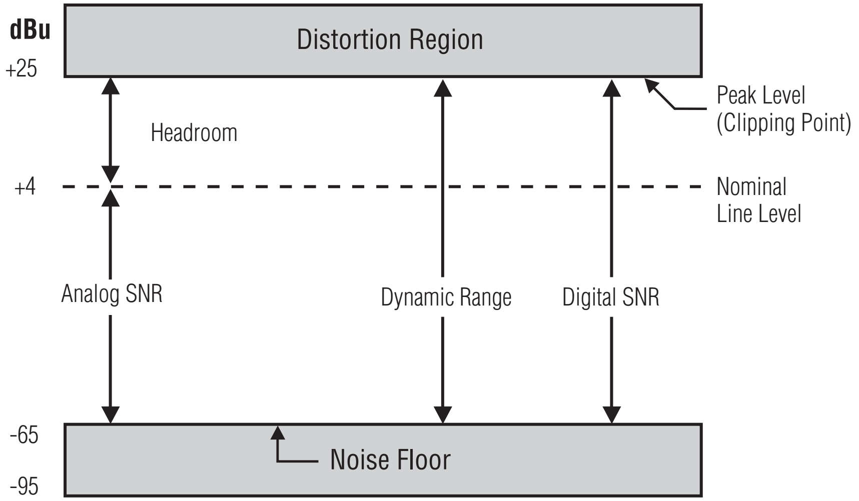


Figure 5.7 **Relationship between some important terms in audio systems**

Another important term is the signal-to-noise ratio (SNR). In analog systems, this means the ratio of the nominal signal to the noise floor, where "line level" is the nominal operating level. On professional equipment, the nominal level is usually 1.228 Vrms, which translates to +4 dBu. The headroom is the difference between nominal line level and the peak level where signal distortion starts to occur. The definition of SNR is a bit different in digital systems, where it is defined as the dynamic range.

Now, armed with an understanding of dynamic range, we can start to discuss how this is useful in practice. Without going into a long derivation, let's simply state what is known as the "6 dB rule". This rule holds the key to the relationship between dynamic range and computational word width. The complete formulation is described in Equation 5.1, but it is used in shorthand to mean that the addition of one bit of precision will lead to a dynamic range increase of 6 dB. Note that the 6 dB rule does not take into account the analog subsystem of an audio design, so the imperfections of the transducers on both the input and the output must be considered separately. Those who want to see the statistical math behind the rule should consult Reference 23 in the Appendix.

$$\textit{Dynamic Range } (dB) = 6.02\boldsymbol{n} + 1.76 \approx 6\boldsymbol{n}\ dB$$
$$\textit{where } \boldsymbol{n} = \textit{the number of precision bits}$$

Equation 5.1 **The 6 dB rule**

The 6 dB rule dictates that the more bits we use, the higher the quality of the system we can attain. In practice, however, there are only a few realistic choices. Most devices suitable for embedded media processing come in three word-width flavors: 16-bit, 24-bit and 32-bit. Table 5.4 summarizes the dynamic ranges for these three types of processors.

Table 5.4 **Dynamic range of various fixed-point architectures**

Computation word width	Dynamic Range (using 6 dB rule)
16-bit fixed-point precision	96 dB
24-bit fixed-point precision	144 dB
32-bit fixed-point precision	192 dB

Since we're talking about the 6 dB rule, it is worth noting something about nonlinear quantization methods typically used for speech signals. A telephone-quality linear PCM encoding requires 12 bits of precision. However, our ears are more sensitive to audio changes at small amplitudes than at high amplitudes. Therefore, the linear PCM sampling is overkill for telephone communications. The logarithmic quantization used by the A-law and μ–law companding standards achieves a 12-bit PCM level of quality using only 8 bits of precision. To make our lives easier, some processor vendors have implemented A-law and μ–law companding into the serial ports of their devices. This relieves the processor core from doing logarithmic calculations.

After reviewing Table 5.4, recall once again that the dynamic range for the human ear is around 120 dB. Because of this, 16-bit data representation doesn't quite cut it for high quality audio. This is why vendors introduced 24-bit processors that extended the dynamic range of 16-bit systems. The 24-bit systems are a bit nonstandard from a C compiler standpoint, so many audio designs these days use 32-bit processing.

Choosing the right processor is not the end of the story, because the total quality of an audio system is dictated by the level of the "lowest-achieving" component. Besides the processor, a complete system includes analog components like microphones and speakers, as well the converters to translate signals between the analog and digital domains. The analog domain is outside of the scope of this discussion, but the audio converters cross into the digital realm.

Let's say that you want to use the AD1871, the same ADC shown in Figure 5.4a, for sampling audio. The datasheet for this converter explains that it is a 24-bit converter, but its dynamic range is not 144 dB—it is 105 dB. The reason for this is that a converter is not a perfect system, and vendors publish only the useful dynamic range in their documentation.

If you were to hook up a 24-bit processor to the AD1871, then the SNR of your complete system would be 105 dB. The conversion error would amount to 144 dB – 105 dB = 39 dB. Figure 5.8 is a graphical representation of this situation. However, there is still another component of a digital audio system that we have not discussed yet: computation on the processor's core.

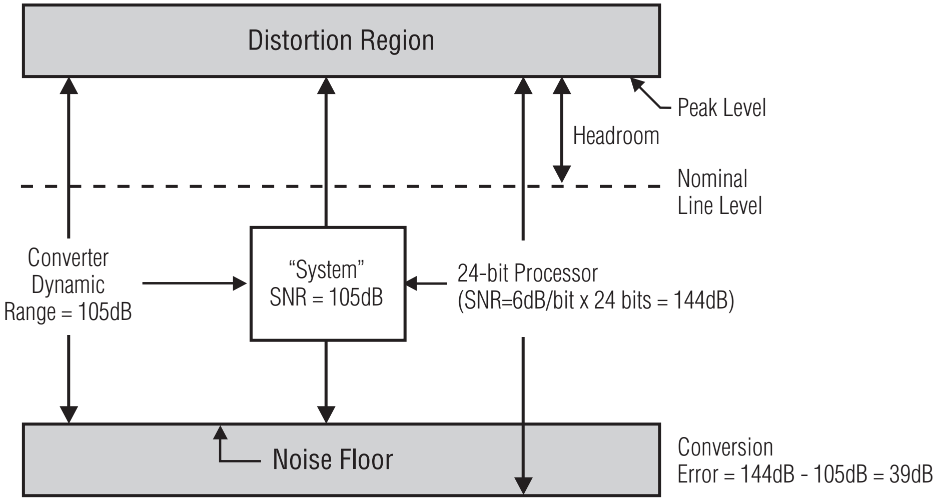

Distortion Region
Peak Level
Headroom
Nominal Line Level
Converter Dynamic Range = 105dB
"System" SNR = 105dB
24-bit Processor (SNR=6dB/bit x 24 bits = 144dB)
Noise Floor
Conversion Error = 144dB - 105dB = 39dB

Figure 5.8 **An audio system's SNR consists of the weakest component's SNR**

Passing data through a processor's computation units can potentially introduce rounding and truncation errors. For example, a 16-bit processor may be able to add a vector of 16-bit data and store this in an extended-length accumulator. However, when the value in the accumulator is eventually written to a 16-bit data register, then some of the bits are truncated.

Take a look at Figure 5.9 to see how computation errors can affect a real system. If we take an ideal 16-bit A/D converter (Figure 5.9a), then its signal-to-noise ratio would be 16 × 6 = 96 dB. If arithmetic and storage errors did not exist, then 16-bit computation would suffice to keep the SNR at 96 dB. 24-bit and 32-bit systems would dedicate 8 and 16 bits, respectively, to the dynamic range below the noise floor. In essence, those extra bits would be wasted.

However, most digital audio systems do introduce some round-off and truncation errors. If we can quantify this error to take, for example, 18 dB (or 3 bits), then it becomes clear that 16-bit computations will not suffice in keeping the system's SNR at 96 dB (Figure 5.9b). Another way to interpret this is to say that the effective noise floor is raised by 18 dB, and the total SNR is decreased to 96 dB – 18 dB = 78 dB. This leads to the conclusion that having extra bits below the converter's noise floor helps to deal with the nuisance of quantization.

Numeric Formats for Audio

There are many ways to represent data inside a processor. The two main processor architectures used for audio processing are fixed-point and floating-point. Fixed-point processors are designed for integer and fractional arithmetic, and they usually natively support 16-bit, 24-bit, or 32-bit data. Floating-point processors provide excellent performance with native support for 32-bit or 64-bit floating-point data types. However, they are typically more costly and consume more power than their fixed-point counterparts, and most real systems must strike a balance between quality and engineering cost.

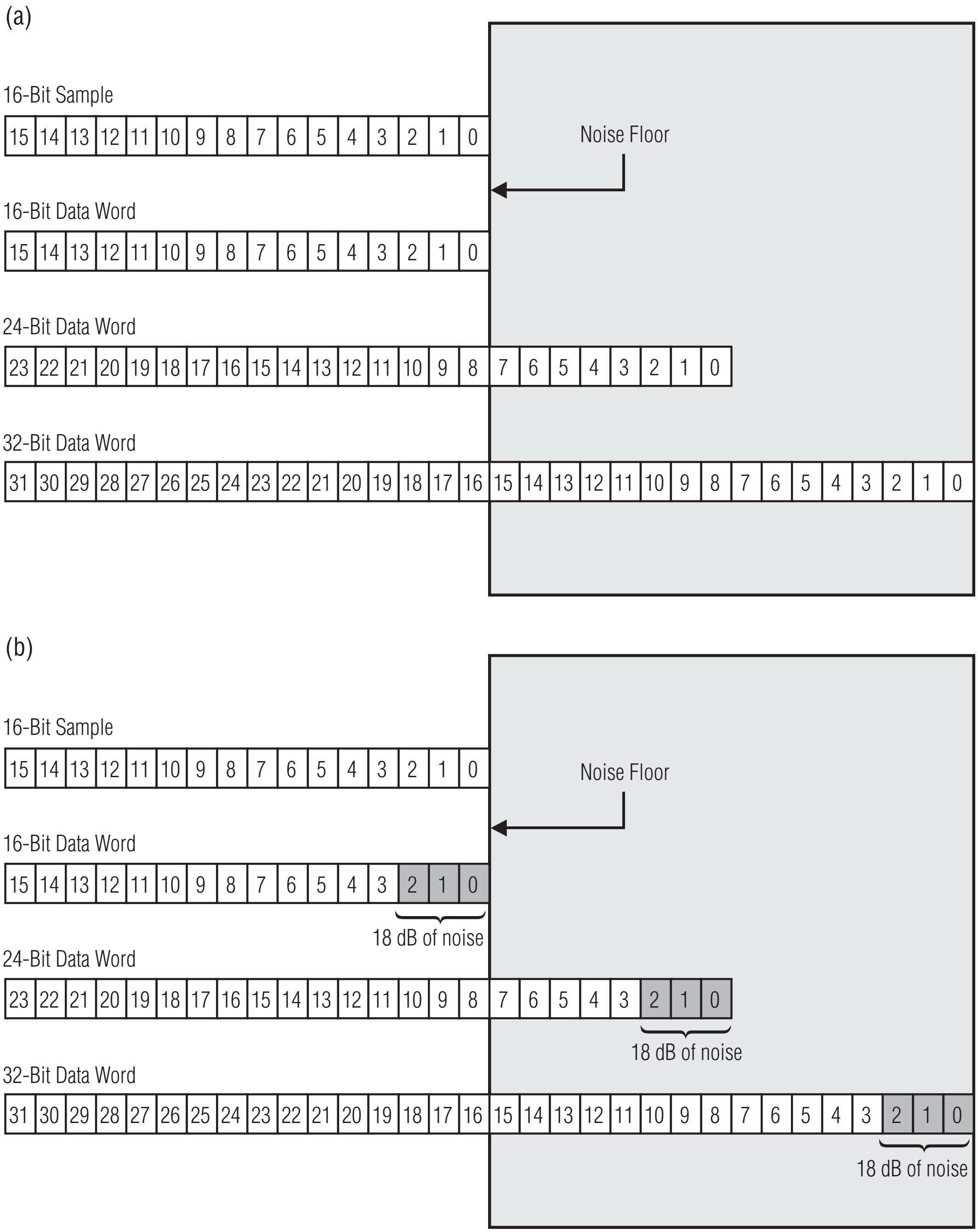


Figure 5.9 **(a) Allocation of extra bits with various word-width computations for an ideal 16-bit, 96 dB SNR system, when quantization error is neglected (b) Allocation of extra bits with various word-width computations for an ideal 16-bit, 96 dB SNR system, when quantization noise is present**

Fixed-Point Arithmetic

Processors that can perform fixed-point operations typically use a twos-complement binary notation for representing signals. A fixed-point format can represent both signed and unsigned integers and fractions. The signed fractional format is most common for digital signal processing on fixed-point processors. The difference between integer and fractional formats lies in the location of the binary point. For integers, the binary point is to the right of the least significant digit, whereas fractions usually have their binary point to the left of the sign bit. Figure 5.10a shows integer and fractional formats.

While the fixed-point convention simplifies numeric operations and conserves memory, it presents a tradeoff between dynamic range and precision. In situations that require a large range of numbers while maintaining high resolution, a radix point that can shift based on magnitude and exponent is desirable.

Floating-Point Arithmetic

Using floating-point format, very large and very small numbers can be represented in the same system. Floating-point numbers are quite similar to scientific notation of rational numbers. They are described with a mantissa and an exponent. The mantissa dictates precision, and the exponent controls dynamic range.

There is a standard that governs floating-point computations of digital machines. It is called IEEE 754 (Figure 5.10b) and can be summarized as follows for 32-bit floating-point numbers. Bit 31 (MSB) is the *sign bit*, where a 0 represents a positive sign and a 1 represents a negative sign. Bits 30 through 23 represent an exponent field (*exp_field*) as a power of 2, biased with an offset of 127. Finally, bits 22 through 0 represent a fractional mantissa (*mantissa*). The hidden bit is basically an implied value of 1 to the left of the radix point.

The value of a 32-bit IEEE 754 floating-point number can be represented with the following equation:

$$(-1)^{sign\_bit} \times (1.mantissa) \times 2^{(exp\_field - 127)}$$

With an 8-bit exponent and a 23-bit mantissa, IEEE 754 reaches a balance between dynamic range and precision. In addition, IEEE floating-point libraries include support for additional features such as ±∞, 0 and NaN (not a number).

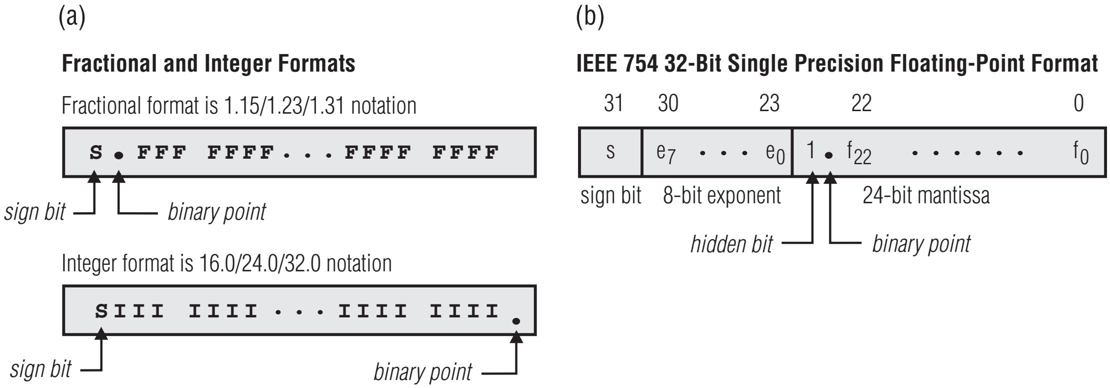


Figure 5.10 **(a) Fractional and integer formats (b) IEEE 754 32-bit single-precision floating-point format**

Table 5.5 shows the smallest and largest values attainable from the common floating-point and fixed-point types.

Table 5.5 **Comparison of dynamic range for various data formats**

Data type	Smallest positive value	Largest positive value
IEEE 754 Floating-Point (single-precision)	$2^{-126} \approx 1.2 \times 10^{-38}$	$2^{128} \approx 3.4 \times 10^{38}$
1.15 16-bit fixed-point	$2^{-15} \approx 3.1 \times 10^{-5}$	$1-2^{-15} \approx 9.9 \times 10^{-1}$
1.23 24-bit fixed-point	$2^{-23} \approx 1.2 \times 10^{-7}$	$1-2^{-23} \approx 9.9 \times 10^{-1}$

Emulation on 16-Bit Architectures

As we explained earlier, 16-bit processing does not provide enough SNR for high quality audio, but this does not mean that you shouldn't choose a 16-bit processor for an audio system. For example, a 32-bit floating-point machine makes it easier to code an algorithm that preserves 32-bit data natively, but a 16-bit processor can also maintain 32-bit integrity through emulation at a much lower cost. Figure 5.11 illustrates some of the possibilities when it comes to choosing a data type for an embedded algorithm.

In the remainder of this section, we'll describe how to achieve floating-point and 32-bit extended-precision fixed-point functionality on a 16-bit fixed-point machine.

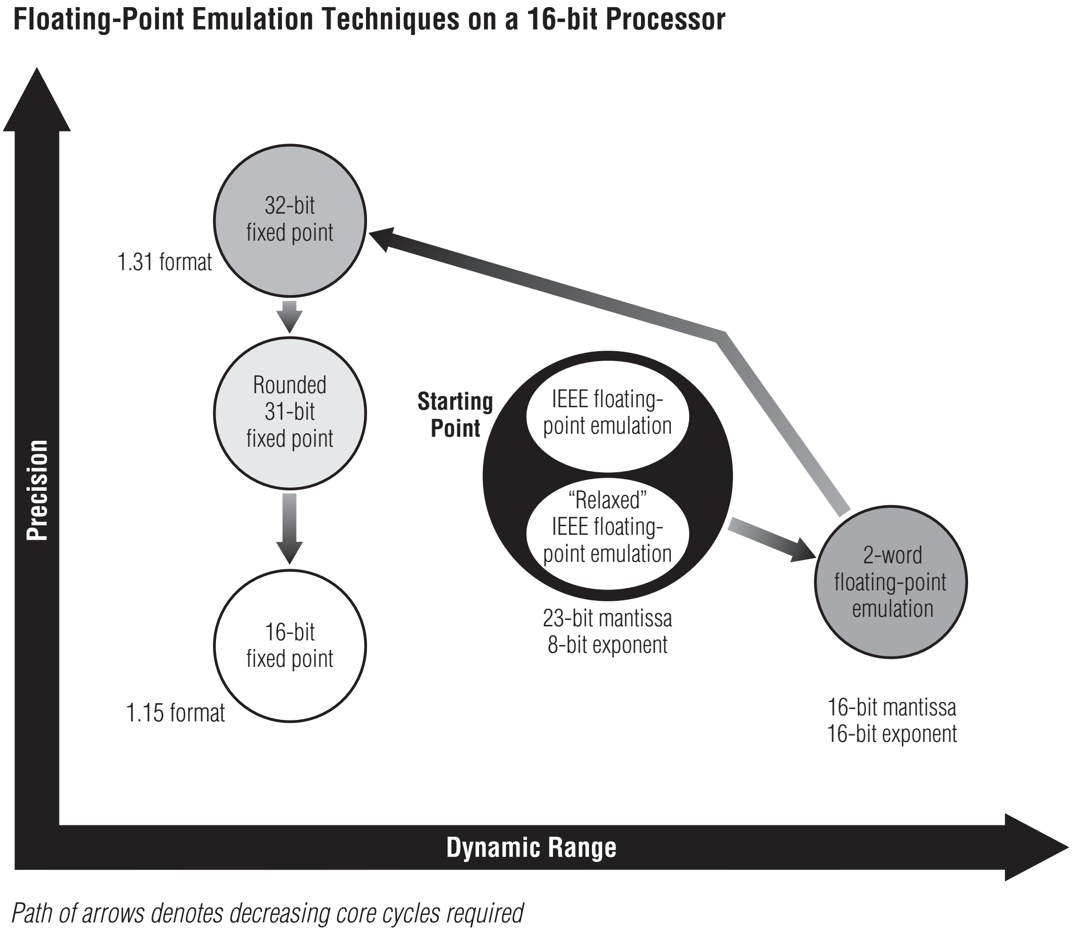

Floating-Point Emulation Techniques on a 16-bit Processor
32-bit fixed point
1.31 format
Rounded 31-bit fixed point
16-bit fixed point
1.15 format
Starting Point
IEEE floating-point emulation
"Relaxed" IEEE floating-point emulation
23-bit mantissa
8-bit exponent
2-word floating-point emulation
16-bit mantissa
16-bit exponent
Precision
Dynamic Range
Path of arrows denotes decreasing core cycles required

Figure 5.11 **Depending on the goals of an application, there are many data types that can satisfy system requirements**

Floating-Point Emulation on Fixed-Point Processors

On most 16-bit fixed-point processors, IEEE 754 floating-point functions are available as library calls from either C/C++ or assembly language. These libraries emulate the required floating-point processing using fixed-point multiply and ALU logic. This emulation requires additional cycles to complete. However, as fixed-point processor core-clock speeds venture into the 500 MHz–1 GHz range, the extra cycles required to emulate IEEE 754-compliant floating-point math become less significant.

It is sometimes advantageous to use a "relaxed" version of IEEE 754 in order to reduce computational complexity. This means that the floating-point arithmetic doesn't implement features such ∞ and NaN.

A further optimization is to use a processor's native data register widths for the mantissa and exponent. Take, for example, the Blackfin architecture, which has a register file set that consists of sixteen 16-bit registers that can be used instead as eight 32-bit registers. In this configuration, on every core-clock cycle, two 32-bit registers can source operands for computation on all four register halves. To make optimized use of the Blackfin register file, a two-word format can be used. In this way, one word (16 bits) is reserved for the exponent and the other word (16 bits) is reserved for the fraction.

Double-Precision Fixed-Point Emulation

There are many applications where 16-bit fixed-point data is not sufficient, but where emulating floating-point arithmetic may be too computationally intensive. For these applications, extended-precision fixed-point emulation may be enough to satisfy system requirements. Using a high-speed fixed-point processor will insure a significant reduction in the amount of required processing. Two popular extended-precision formats for audio are 32-bit and 31-bit fixed-point representations.

32-Bit-Accurate Emulation

32-bit arithmetic is a natural software extension for 16-bit fixed-point processors. For processors whose 32-bit register files can be accessed as two 16-bit halves, the halves can be used together to represent a single 32-bit fixed-point number. The Blackfin processor's hardware implementation allows for single-cycle 32-bit addition and subtraction.

For instances where a 32-bit multiply will be iterated with accumulation (as is the case in some algorithms we'll talk about soon), we can achieve 32-bit accuracy with

16-bit multiplications in just three cycles. Each of the two 32-bit operands (R0 and R1) can be broken up into two 16-bit halves (R0.H | R0.L and R1.H | R1.L).

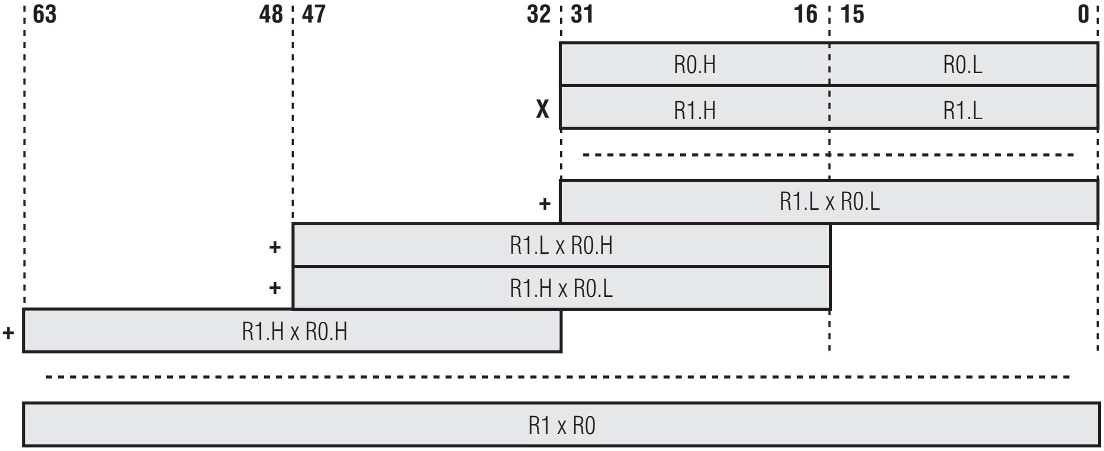


Figure 5.12 **32-bit multiplication with 16-bit operations**

From Figure 5.12, it is easy to see that the following operations are required to emulate the 32-bit multiplication R0 × R1 with a combination of instructions using 16-bit multipliers:

- Four 16-bit multiplications to yield four 32-bit results

    1. R1.L × R0.L
    2. R1.L × R0.H
    3. R1.H × R0.L
    4. R1.H × R0.H

- Three operations to preserve bit place in the final answer (the >> symbol denotes a right shift). Since we are performing fractional arithmetic, the result is 1.62 (1.31 × 1.31 = 2.62 with a redundant sign bit). Most of the time, the result can be truncated to 1.31 in order to fit in a 32-bit data register. Therefore, the result of the multiplication should be in reference to the sign bit, or the most significant bit. This way the least significant bits can be safely discarded in a truncation.

1. (R1.L × R0.L) >> 32
2. (R1.L × R0.H) >> 16
3. (R1.H × R0.L) >> 16

The final expression for a 32-bit multiplication is

$$((R1.L \times R0.L) >> 32 + (R1.L \times R0.H) >> 16) \\ + ((R1.H \times R0.L) >> 16 + R1.H \times R0.H)$$

On the Blackfin architecture, these instructions can be issued in parallel to yield an effective rate of a 32-bit multiplication in three cycles.

31-Bit-Accurate Emulation

We can reduce a fixed-point multiplication requiring at most 31-bit accuracy to just two cycles. This technique is especially appealing for audio systems, which usually require at least 24-bit representation, but where 32-bit accuracy may be a bit excessive. Using the 6 dB rule, 31-bit-accurate emulation still maintains a dynamic range of around 186 dB, which is plenty of headroom even with rounding and truncation errors.

From the multiplication diagram shown in Figure 5.12, it is apparent that the multiplication of the least significant half-word R1.L × R0.L does not contribute much to the final result. In fact, if the result is truncated to 1.31, then this multiplication can only have an effect on the least significant bit of the 1.31 result. For many applications, the loss of accuracy due to this bit is balanced by the speeding up of the 32-bit multiplication through eliminating one 16-bit multiplication, one shift, and one addition.

The expression for 31-bit accurate multiplication is

$$((R1.L \times R0.H) + (R1.H \times R0.L)) >> 16 + (R1.H \times R0.H)$$

On the Blackfin architecture, these instructions can be issued in parallel to yield an effective rate of two cycles for each 32-bit multiplication.

Audio Processing Methods

Getting Data to the Processor's Core

There are a number of ways to get audio data into the processor's core. For example, a foreground program can poll a serial port for new data, but this type of transfer is uncommon in embedded media processors, because it makes inefficient use of the core.

Instead, a processor connected to an audio codec usually uses a DMA engine to transfer the data from the codec link (like a serial port) to some memory space available to the processor. This transfer of data occurs in the background without the core's intervention. The only overhead is in setting up the DMA sequence and handling the interrupts once the data buffer has been received or transmitted.

Block Processing versus Sample Processing

Sample processing and block processing are two approaches for dealing with digital audio data. In the sample-based method, the processor crunches the data as soon as it's available. Here, the processing function incurs overhead during each sample period. Many filters (like FIR and IIR, described later) are implemented this way, because the effective latency is low.

Block processing, on the other hand, is based on filling a buffer of a specific length before passing the data to the processing function. Some filters are implemented using block processing because it is more efficient than sample processing. For one, the block method sharply reduces the overhead of calling a processing function for each sample. Also, many embedded processors contain multiple ALUs that can parallelize the computation of a block of data. What's more, some algorithms are, by nature, meant to be processed in blocks. For example, the Fourier transform (and its practical counterpart, the fast Fourier transform, or FFT) accepts blocks of temporal or spatial data and converts them into frequency domain representations.

Double-Buffering

In a block-based processing system that uses DMA to transfer data to and from the processor core, a "double buffer" must exist to handle the DMA transfers and the core. This is done so that the processor core and the core-independent DMA engine do not access the same data at the same time, causing a data coherency problem. To facilitate the processing of a buffer of length N, simply create a buffer of length 2 × N. For a bidirectional system, two buffers of length 2 × N must be created. As shown in Figure 5.13a, the core processes the `in1` buffer and stores the result in the `out1` buffer, while the DMA engine is filling `in0` and transmitting the data from `out0`. Figure 5.13b depicts that once the DMA engine is done with the left half of the double buffers, it starts transferring data into `in1` and out of `out1`, while the core processes data from `in0` and into `out0`. This configuration is sometimes called "ping-pong buffering," because the core alternates between processing the left and right halves of the double buffers.

Note that, in real-time systems, the serial port DMA (or another peripheral's DMA tied to the audio sampling rate) dictates the timing budget. For this reason, the block processing algorithm must be optimized in such a way that its execution time is less than or equal to the time it takes the DMA to transfer data to/from one half of a double-buffer.

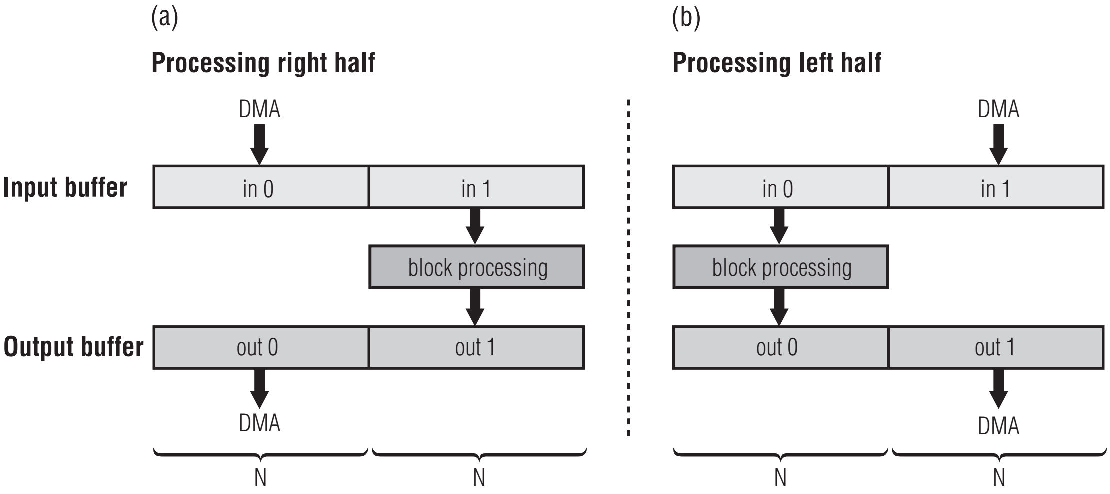


Figure 5.13 **Double-buffering scheme for stream processing**

2D DMA

When data is transferred across a digital link like I²S, it may contain several channels. These may all be multiplexed on one data line going into the same serial port. In such a case, 2D DMA can be used to de-interleave the data so that each channel is linearly arranged in memory. Take a look at Figure 5.14 for a graphical depiction of this arrangement, where samples from the left and right channels are de-multiplexed into two separate blocks. This automatic data arrangement is extremely valuable for those systems that employ block processing.

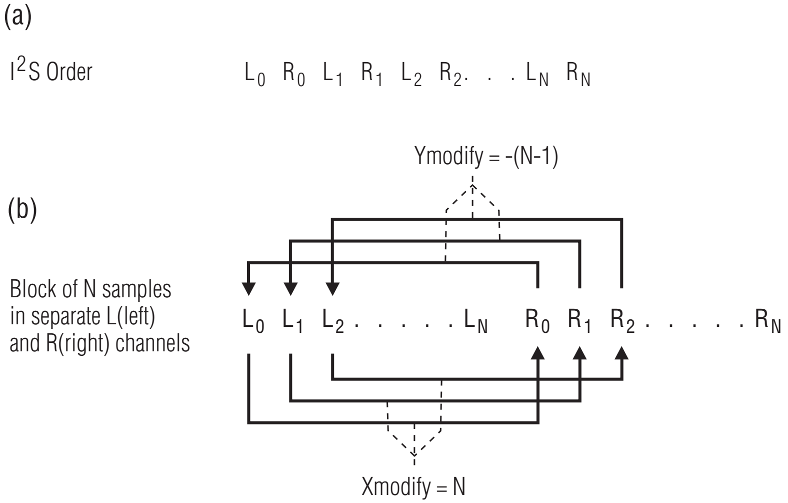

(a)
I^{2}S Order
L_0 R_0 L_1 R_1 L_2 R_2. . . L_N R_N
Ymodify = -(N-1)
(b)
Block of N samples in separate L(left) and R(right) channels
L_0 L_1 L_2 . . . . . L_N R_0 R_1 R_2 . . . . . R_N
Xmodify = N

Figure 5.14 **A 2D DMA engine used to de-interleave (a) I²S stereo data into (b) separate left and right buffers**

Basic Operations

There are three fundamental building blocks in audio processing. They are the summing operation, multiplication, and time delay. Many more complicated effects and algorithms can be implemented using these three elements. A summer has the obvious duty of adding two signals together. A multiplication can be used to boost or attenuate an audio signal. On most media processors, multiple summer and/or multiplier blocks can execute in a single cycle. A time delay is a bit more complicated. In many audio algorithms, the current output depends on a combination of previous inputs and/or outputs. The implementation of this delay effect is accomplished with a delay line, which is really nothing more than an array in memory that holds previous data. For example, an echo algorithm might hold 500 ms of input samples. The current output value can be computed by adding the current input value to a slightly attenuated prior sample. If the audio system is sample-based, then the programmer can simply keep track of an input pointer and an output pointer (spaced at 500 ms worth of samples apart), and increment them after each sampling period.

Since delay lines are meant to be reused for subsequent sets of data, the input and output pointers will need to wrap around from the end of the delay line buffer back to the beginning. In C/C++, this is usually done by appending the modulus operator (%) to the pointer increment.

This wrap-around may incur no extra processing cycles if you use a processor that supports circular buffering (see Figure 5.15). In this case, the beginning address and length of a circular buffer must be provided only once. During processing, the software increments or decrements the current pointer within the buffer, but the hardware takes care of wrapping around to the beginning of the buffer if the current pointer falls outside of the buffer's boundaries. Without this automated address generation, the programmer would have to manually keep track of the buffer, thus wasting valuable processing cycles.

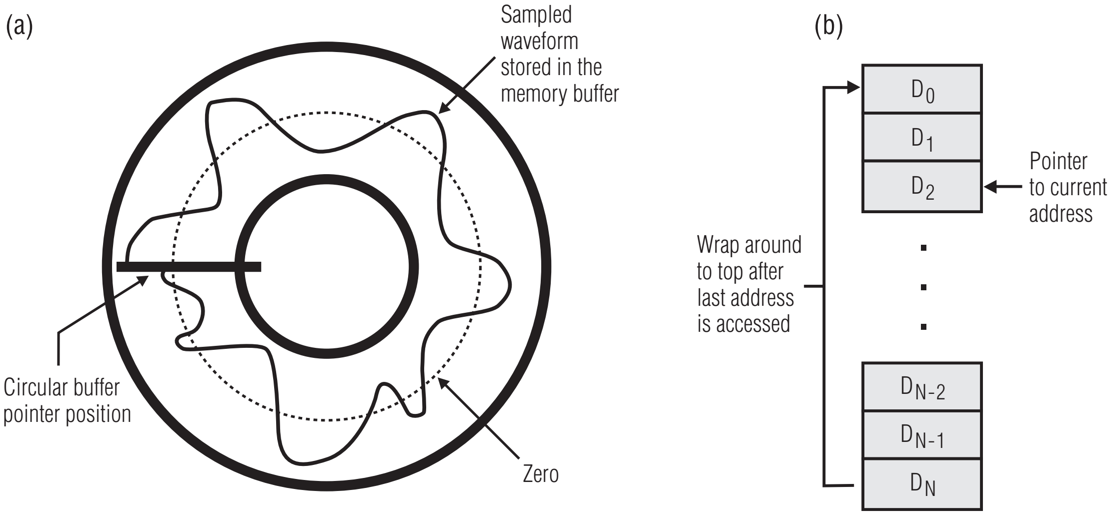

(a)
Sampled waveform stored in the memory buffer
Circular buffer pointer position
Zero
(b)
D_0
D_1
D_2
Pointer to current address
Wrap around to top after last address is accessed
D_{N-2}
D_{N-1}
D_N

Figure 5.15 **(a) Graphical representation of a delay line using a circular buffer (b) Layout of a circular buffer in memory**

An echo effect derives from an important audio building block called the comb filter, which is essentially a delay with a feedback element. When multiple comb filters are used simultaneously, they can create the effect of reverberation.

Signal Generation

In some audio systems, a signal (for example, a sine wave) might need to be synthesized. Taylor Series function approximations can emulate trigonometric functions. Moreover, uniform random number generators are handy for creating white noise.

However, synthesis might not fit into a given system's processing budget. On fixed-point systems with ample memory, you can use a table lookup instead of generating a signal. This has the side effect of taking up precious memory resources, so hybrid methods can be used as a compromise. For example, you can store a coarse lookup table to save memory. During runtime, the exact values can be extracted from the table using interpolation, an operation that can take significantly less time than computing a full approximation. This hybrid approach provides a good balance between computation time and memory resources.

Filtering and Algorithms

Digital filters are used in audio systems for attenuating or boosting the energy content of a sound wave at specific frequencies. The most common filter forms are high-pass, low-pass, band-pass and notch. Any of these filters can be implemented in two ways. These are the finite impulse response filter (FIR) and the infinite impulse response filter (IIR), and they constitute building blocks to more complicated filtering algorithms like parametric equalizers and graphic equalizers.

Finite Impulse Response (FIR) Filter

The FIR filter's output is determined by the sum of the current and past inputs, each of which is first multiplied by a filter coefficient. The FIR summation equation, shown in Figure 5.16a, is also known as *convolution*, one of the most important operations in signal processing. In this syntax, x is the input vector, y is the output vector, and h holds the filter coefficients. Figure 5.16a also shows a graphical representation of the FIR implementation.

Convolution is such a common operation in media processing that many processors are designed to execute a multiply-accumulate (MAC) instruction along with multiple data accesses (reads and writes) and pointer increments in one cycle.

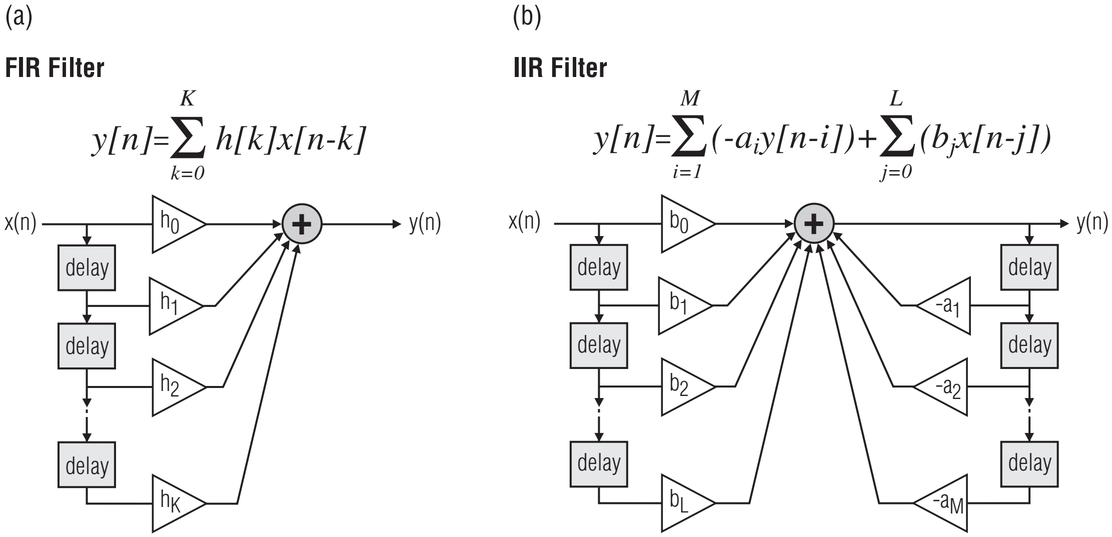


Figure 5.16 **(a) FIR filter equation and structure (b) IIR filter equation and structure**

Infinite Impulse Response (IIR) Filter

Unlike the FIR, whose output depends only on inputs, the IIR filter relies on both inputs and past outputs. The basic equation for an IIR filter is a difference equation, as shown in Figure 5.16b. Because of the current output's dependence on past outputs, IIR filters are often referred to as *recursive filters*. Figure 5.16b also gives a graphical perspective on the structure of the IIR filter.

Fast Fourier Transform

Quite often, we can do a better job describing an audio signal by characterizing its frequency composition. A Fourier transform takes a time-domain signal and translates it into the frequency domain; the inverse Fourier transform achieves the opposite, converting a frequency-domain representation back into the time domain. Mathematically, there are some nice property relationships between operations in the time domain and those in the frequency domain. Specifically, a time-domain convolution (or an FIR filter) is equivalent to a multiplication in the frequency domain. This tidbit would not be too practical if it weren't for a special optimized implementation of the Fourier transform called the fast Fourier transform (FFT). In fact, it is often more efficient to implement an FIR filter by transforming the input signal and coefficients into the frequency domain with an FFT, multiplying the transforms, and finally transforming the result back into the time domain with an inverse FFT.

There are other transforms that are used often in audio processing. Among them, one of the most common is the modified discrete cosine transform (MDCT), which is the basis for many audio compression algorithms.

Sample Rate Conversion

There are times when you will need to convert a signal sampled at one frequency to a different sampling rate. One situation where this is useful is when you want to decode an audio signal sampled at, say 8 kHz, but the DAC you're using does not support that sampling frequency. Another scenario is when a signal is oversampled, and converting to a lower sampling frequency can lead to a reduction in computation time. The process of converting the sampling rate of a signal from one rate to another is called sampling rate conversion (or SRC).

Increasing the sampling rate is called interpolation, and decreasing it is called decimation. Decimating a signal by a factor of M is achieved by keeping only every Mth sample and discarding the rest. Interpolating a signal by a factor of L is accomplished by padding the original signal with L–1 zeros between each sample.

Even though interpolation and decimation factors are integers, you can apply them in series to an input signal to achieve a rational conversion factor. When you upsample by 5 and then downsample by 3, then the resulting resampling factor is 5/3 = 1.67.

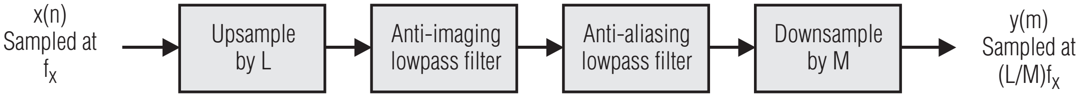


Figure 5.17 **Sample-rate conversion through upsampling and downsampling**

To be honest, we oversimplified the SRC process a bit too much. In order to prevent artifacts due to zero-padding a signal (which creates images in the frequency domain), an interpolated signal must be low-pass-filtered before being used as an output or as an input into a decimator. This anti-imaging low-pass filter can operate at the input sample rate, rather than at the faster output sample rate, by using a special FIR filter structure that recognizes that the inputs associated with the L–1 inserted samples have zero values.

Similarly, before they're decimated, all input signals must be low-pass-filtered to prevent aliasing. The anti-aliasing low-pass filter may be designed to operate at the decimated sample rate, rather than at the faster input sample rate, by using a FIR filter structure that realizes the output samples associated with the discarded samples need not be computed. Figure 5.17 shows a flow diagram of a sample rate converter. Note that it is possible to combine the anti-imaging and anti-aliasing filter into one component for computational savings.

Audio Compression

Even though raw audio requires a lower bit rate than raw video, the amount of data is still substantial. The bit rate required to code a single CD-quality audio channel (44.1 kHz at 16 bits) using the standard PCM method is 705.6 kbps—one minute of stereo sound requires over 10 Mbytes of storage! Sending this much data over a network or a serial connection is inefficient, and sometimes impossible. The solution comes in the form of compression algorithms called audio codecs. These software codecs, not to be confused with hardware ADCs and DACs discussed already, compress raw data either for low-bandwidth transfer or for storage, and decompress for the reverse effect.

There are lossless codecs and lossy codecs available for audio. Lossless codecs are constructed in such a way that a compressed signal can be reconstructed to contain the exact data as the original input signal. Lossless codecs are computationally intensive, and they can reduce audio bit rate by up to about ½. Lossy codecs can compress audio much more (10× or more, depending on desired quality), and the audio decoded from a lossy stream sounds very close to the original, even though information is lost forever in the encoding process. Lossy codecs can throw out data and still preserve audio integrity, because they are based on a psycho-acoustical model that takes advantage of our ears' physiology. In essence, a lossy codec can cheat by dropping data that will not affect how we'll ultimately perceive the signal. This technique is often referred to as "perceptual encoding."

Earlier, we mentioned frequency and temporal masking. Another useful feature for perceptual encoding—called *joint stereo encoding*—deals with multiple channels. The basic premise is that data in two or more channels is correlated. By decoupling unique features of each channel from the shared features, we can drastically reduce the data needed to encode the content. If one channel takes 196 kbps, then an encoder that recognizes the redundancy will allocate much less data than 2 × 196 kbps for a stereo stream, while still retaining the same perceived sound. The general rule of thumb is that multichannel audio can be efficiently encoded with an amount of data proportional to the square root of the number of channels (see Reference 23 in the Appendix for more details).

In practice, audio encoders use two techniques: sub-band coding and transform coding. Sub-band coding splits the input audio signal into a number of sub-bands, using band-pass filters. A psycho-acoustical model is applied on the sub-bands to define the number of bits necessary to maintain a specified sound quality. Transform coding uses a transform like the FFT or an MDCT on a block of audio data. Then, a psycho-acoustical model is used to determine the proper quantization of the frequency components based on a masking threshold to ensure that the output sounds like the input signal.

Let's take a look at some of the currently available audio codecs. Some of the algorithms are proprietary and require a license before they can be used in a system. Table 5.6 lists common audio coding standards and the organizations responsible for them.

Table 5.6 **Various audio codecs**

Audio Coding Standard	(Licensing/Standardization Organization)
MP3	ISO/IEC
AAC	ISO/IEC
AC-3	Dolby Labs
Windows Media Audio	Microsoft
RealAudio	RealNetworks
Vorbis	Xiph.org
FLAC	Xiph.org

MP3

MP3 is probably the most popular lossy audio compression codec available today. The format, officially known as MPEG-1 Audio Layer 3, was released in 1992 as a complement to the MPEG-1 video standard from the Moving Pictures Experts Group. MPEG is a group of ISO/IEC, an information center jointly operated by the International Organization for Standardization and the International Electrotechnical Commission. MP3 was developed by the German Fraunhofer Institut Integrierte Schaltungen (Fraunhofer IIS), which holds a number of patents for MP3 encoding and decoding. Therefore, you must obtain a license before incorporating the MP3 algorithm into your embedded systems.

MP3 uses polyphase filters to separate the original signal into sub-bands. Then, the MDCT transform converts the signal into the frequency domain, where a psychoacoustical model quantizes the frequency coefficients. A CD-quality track can be MP3-encoded at a 128–196 kbps rate, thus achieving up to a 12:1 compression ratio.

AAC

Advanced Audio Coding (AAC) is a second-generation codec also developed by Fraunhofer IIS. It was designed to complement the MPEG-2 video format. Its main improvement over MP3 is the ability to achieve lower bit rates at equivalent sound quality.

AC-3

The AC-3 format was developed by Dolby Laboratories to efficiently handle multichannel audio such as 5.1, a capability that was not implemented in the MP3 standard. The nominal stereo bit rate is 192 kbps, whereas it's 384 kbps for 5.1 surround sound.

A 5.1 surround-sound system contains five full-range speakers, including front left and right, rear left and right, and front center channels, along with a low-frequency (10 Hz–120 Hz) subwoofer.

WMA

Windows Media Audio (WMA) is a proprietary codec developed by Microsoft to challenge the popularity of MP3. Microsoft developed WMA with paid music distribution in mind, so they incorporated Digital Rights Management (DRM) into the codec. Besides the more popular lossy codec, WMA also supports lossless encoding.

RealAudio

RealAudio, developed by RealNetworks, is another proprietary format. It was conceived to allow the streaming of audio data over low bandwidth links. Many Internet radio stations use this format to stream their content. Recent updates to the codec have improved its quality to match that of other modern codecs.

Vorbis

Vorbis was created at the outset to be free of any patents. It is released by the Xiph.org Foundation as a completely royalty-free codec. A full Vorbis implementation for both floating-point and fixed-point processors is available under a free license from Xiph.org. Because it is free, Vorbis is finding its way into increasing numbers of embedded devices.

According to many subjective tests, Vorbis outperforms MP3, and it is therefore in the class of the newer codecs like WMA and AAC. Vorbis also fully supports multi-channel compression, thereby eliminating redundant information carried by the channels. Refer to Chapter 9 for further discussion on the Vorbis codec and its implementation on an embedded processor.

FLAC

FLAC is another open standard from the Xiph.org Foundation. It stands for Free Lossless Audio Codec, and as the name implies, it does not throw out any information from the original audio signal. This, of course, comes at the expense of much smaller achievable compression ratios. The typical compression range for FLAC is 30–70%.

Speech Compression

Speech compression is used widely in real-time communications systems like cell phones, and in packetized voice connections like Internet phones.

Since speech is more band-limited than full-range audio, it is possible to employ audio codecs, taking the smaller bandwidth into account. Almost all speech codecs do, indeed, sample voice data at 8 kHz. However, we can do better than just take advantage of the smaller frequency range. Since only a subset of the audible signals within the speech bandwidth is ever vocally generated, we can drive bit rates even lower. The major goal in speech encoding is a highly compressed stream with good intelligibility and short delays to make full-duplex communication possible.

The most traditional speech coding approach is code-excited linear prediction (CELP). CELP is based on linear prediction coding (LPC) models of the vocal tract and a supplementary residue codebook.

The idea behind using LPC for speech coding is founded on the observation that the human vocal tract can be roughly modeled with linear filters. We can make two basic kinds of sounds: voiced and unvoiced. Voiced sounds are produced when our vocal cords vibrate, and unvoiced sounds are created when air is constricted by the mouth, tongue, lips and teeth. Voiced sounds can be modeled as linear filters driven by a fundamental frequency, whereas unvoiced ones are modeled with random noise sources. Through these models, we can describe human utterances with just a few parameters. This allows LPC to predict signal output based on previous inputs and outputs. To complete the model, LPC systems supplement the idealized filters with residue (i.e., error) tables. The codebook part of CELP is basically a table of typical residues.

In real-time duplex communications systems, one person speaks while the other one listens. Since the person speaking is not contributing anything to the signal, some codecs implement features like voice activity detection (VAD) to recognize silence, and comfort noise generation (CNG) to simulate the natural level of noise without actually encoding it at the transmitting end.

Table 5.7 **Various speech codecs**

Speech Coding Standard	Bit rate	Governing Body
GSM-FR	13 kbps	ETSI
GSM-EFR	12.2 kbps	ETSI
GSM-AMR	4.75, 5.15, 5.90, 6.70, 7.40, 7.95, 10.2, 12.2 kbps	3GPP
G.711	64 kbps	ITU-T
G.723.1	5.3, 6.3 kbps	ITU-T
G.729	6.4, 8, 11.8 kbps	ITU-T
Speex	2 – 44 kbps	Xiph.org

GSM

The GSM speech codecs find use in cell phone systems around the world. The governing body of these standards is the European Telecommunications Standards Institute (ETSI). There is actually an evolution of standards in this domain. The first one was GSM Full Rate (GSM-FR). This standard uses a CELP variant called Regular Pulse Excited Linear Predictive Coder (RPELPC). The input speech signal is divided into 20-ms frames. Each of those frames is encoded as 260 bits, thereby producing a total bit rate of 13 kbps. Free GSM-FR implementations are available for use under certain restrictions.

GSM Enhanced Full Rate (GSM-EFR) was developed to improve the quality of speech encoded with GSM-FR. It operates on 20-ms frames at a bit rate of 12.2 kbps, and it works in noise-free and noisy environments. GSM-EFR is based on the patented Algebraic Code Excited Linear Prediction (ACELP) technology, so you must purchase a license before using it in end products.

The 3rd Generation Partnership Project (3GPP), a group of standards bodies, introduced the GSM Adaptive Multi-Rate (GSM-AMR) codec to deliver even higher quality speech over lower-bit-rate data links by using an ACELP algorithm. It uses 20-ms data chunks, and it allows for multiple bit rates at eight discrete levels between 4.75 kbps and 12.2 kbps. GSM-AMR supports VAD and CNG for reduced bit rates.

The "G-Dot" Standards

The International Telecommunication Union (ITU) was created to coordinate the standards in the communications industry, and the ITU Telecommunication Standardization Sector (ITU-T) is responsible for the recommendations of many speech codecs, known as the G.x standards.

G.711

G.711, introduced in 1988, is a simple standard when compared with the other options presented here. The only compression used in G.711 is companding (using either the μ-law or A-law standards), which compresses each data sample to 8 bits, yielding an output bit rate of 64 kbps.

G.723.1

G.723.1 is an ACELP-based dual-bit-rate codec, released in 1996, that targets Voice-Over-IP (VoIP) applications like teleconferencing. The encoding frame for G.723.1 is 30 ms. Each frame can be encoded in 20 or 24 bytes, thus translating to 5.3 kbps and 6.3 kbps streams, respectively. The bit rates can be effectively reduced through VAD and CNG. The codec offers good immunity against network imperfections like lost frames and bit errors. This speech codec is part of video conferencing applications described by the H.324 family of standards.

G.729

Another speech codec released in 1996 is G.729, which partitions speech into 10-ms frames, making it a low-latency codec. It uses an algorithm called Conjugate Structure ACELP (CS-ACELP). G.729 compresses 16-bit signals sampled at 8 kHz via 10-ms frames into a standard bit rate of 8 kbps, but it also supports 6.4 kbps and 11.8 kbps rates. VAD and CNG are also supported.

Speex

Speex is another codec released by Xiph.org, with the goal of being a totally patent-free speech solution. Like many other speech codecs, Speex is based on CELP with residue coding. The codec can take 8 kHz, 16 kHz, and 32 kHz linear PCM signals and code them into bit rates ranging from 2 to 44 kbps. Speex is resilient to network errors, and it supports voice activity detection. Besides allowing variable bit rates, another unique feature of Speex is stereo encoding. Source code is available from Xiph.org in both a floating-point reference implementation and a fixed-point version.

What's Next?

Now that you've got a backgrounder on audio as it relates to embedded media processing, we'll strive to provide you with a similar level of familiarity with video. Taken together, audio and video provide the core of most multimedia systems, and once we have explored both, we'll start looking at multimedia frameworks in embedded systems.