Contents

Chapter 15

DSP and Software Radio Design

In recent years, digital signal processing (DSP) technology has progressed to the point where it is an integral part of our radio equipment. DSP is rapidly replacing hardware circuits with software, offering amateurs flexibility and features only dreamed of in the past. This chapter, by Alan Bloom, N1AL, explores DSP and its use in radio design. DSP projects and additional background and support materials may be found on the *Handbook* CD.

Chapter 15 — CD-ROM Content

- A collection of DSP projects with supporting files
- A discussion of DSP calculations with samples and files that accompany the discussion

15.1 Introduction

Digital signal processing (DSP) has been around a long time. The essential theory was developed by mathematicians such as Newton, Gauss and Fourier in the 17th, 18th and 19th centuries. It was not until the latter half of the 20th century, however, that digital computers became available that could do the calculations fast enough to process signals in real time. Today DSP is important in many fields, such as seismology, acoustics, radar, medical imaging, nuclear engineering, audio and video processing, as well as speech and data communications.

In all those systems, the idea is to process a digitized signal so as to extract information from it or to control its characteristics in some way. For example, an EKG monitor in a hospital extracts the essential characteristics of the signal from the patient's heart for display on a screen. A digital communications receiver uses DSP to filter and demodulate the received RF signal before sending it to the speaker or headphones. In some systems, the signal to be processed may have more than one dimension. An example is image data, which requires two-dimensional processing. Similarly, the controller for an electrically-steerable antenna array uses multi-dimensional DSP techniques to determine the amplitude and phase of the RF signal in each of the antenna elements. A CT scanner analyzes X-ray data in three dimensions to determine the internal structures of a human body.

SOFTWARE-DEFINED RADIO

The concept of a *software-defined radio* (SDR) became popular in the 1990s. By then, DSP technology had developed to the point that it was possible to implement almost all the signal-processing functions of a transceiver using inexpensive programmable digital hardware. The frequency, bandwidth, modulation, filtering and other characteristics can be changed under software control, rather than being fixed by the hardware design as in a conventional radio. Adding a new modulation type or a new improved filter design is a simple matter of downloading new software. In addition, with the same hardware design, a single radio can have several different modulation modes.

SDR is appealing to regulatory bodies such as the FCC because it makes possible a communications system called *cognitive radio* in which multiple radio services can share the same frequency spectrum.[1] Each node in a wireless network is programmed to dynamically change its transmission or reception characteristics to avoid interference with other users. In this way, services that in the past enjoyed fixed frequency allocations but that only use their channels a small percentage of the time can share their spectrum with other wireless users with minimal interference.

DSP ADVANTAGES

Digital signal processing has the reputation of being more complicated than the analog circuitry that it replaces. In reality, once the analog signal has been converted into the digital domain, complicated functions can be implemented in software much more simply than would be possible with analog components. For example, the traditional "phasing" method of generating an SSB signal without an expensive crystal filter requires various mixers, oscillators, filters and a wide-band audio-frequency phase-shift network built with a network of high-precision resistors and capacitors. To implement the same function in a DSP system requires adding one additional subroutine to the software program — no additional hardware is needed.

There are many features that are straightforward with DSP techniques but would be difficult or impractical to implement with analog circuitry. A few examples drawn just from the communications field are imageless mixing, noise reduction, OFDM modulation and adaptive channel equalization. Digital signals can have much more dynamic range than analog signals, limited only by the number of bits used to represent the signal. For example, it is easy to add an extra 20 or 30 dB of headroom to the intermediate signal processing stages to ensure that there is no measurable degradation of the signal, something that might be difficult or impossible with analog circuitry. Replacing analog circuitry with software algorithms eliminates the problems of nonlinearity and drift of component values with time and temperature. The programmable nature of most DSP systems means you can make the equivalent of circuit modifications without having to unsolder any components.

DSP LIMITATIONS

Despite its many advantages, we don't mean to imply that DSP is best in all situations. High-power and very high-frequency signals are still the domain of analog circuitry. Where simplicity and low power consumption are primary goals, a DSP solution may not be the best choice. For example, a simple CW receiver that draws a few milliamps from the power supply can be built with two or three analog ICs and a handful of discrete components. In many high-performance systems, the performance of the analog-to-digital converter (ADC or A/D converter) and digital-to-analog converter (DAC or D/A converter) are the limiting factors. That is why, even with the latest generation of affordable ADC technology, it is still possible to obtain better blocking dynamic range in an HF receiver using a hybrid analog-digital system rather than going all-digital by routing the RF input directly to an ADC.

The plan of this chapter is first to discuss the overall hardware and software structure of a DSP system, including general information on factors to be considered when designing at the system level. Then we will cover the basic theory of digital signals, with emphasis on topics relevant to radio communications. Following that is a section on digital filters and another section that describes several other miscellaneous DSP applications. The concept of analytic signals (negative frequencies and all that) is important for understanding software-defined radios, so we cover that before getting into SDRs themselves. The final two sections, on SDR hardware and software, use many of the concepts explained in previous sections to show how a radio can be built with most of the signal processing done digitally. For the fullest understanding of this chapter the reader should have a basic familiarity of the topics covered in the **Electrical Fundamentals**, **Analog Basics** and **Digital Basics** chapters as well as some high-school trigonometry.

15.2 Typical DSP System Block Diagram

A typical DSP system is conceptually very simple. It consists of only three sections, as illustrated in **Fig 15.1**. An ADC at the input converts an analog signal into a series of digital numbers that represent snapshots of the signal at a series of equally spaced sample times. The digital signal processor itself does some kind of calculations on that digital signal to generate a new stream of numbers at its output. A DAC then converts those numbers back into analog form.

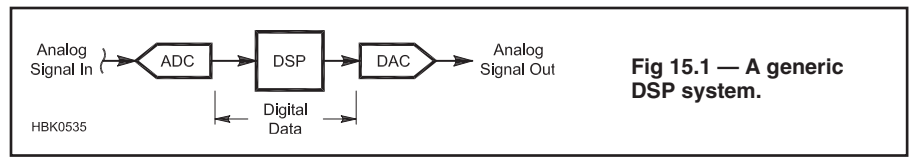

Fig 15.1 — A generic DSP system.

Some DSP systems may not have all three components. For example, a DSP-based audio-frequency generator does not need an ADC. Similarly, there is no need for a DAC in a measurement system that monitors some sensor output, processes the signal and stores the result in a computer file or displays it on a digital readout.

The term "DSP" is normally understood to imply processing that occurs in real time, at least in some sense. For example, an RF or microwave signal analyzer might include a DSP co-processor that processes chunks of sampled data in batch mode for display a fraction of a second later. However, a computer program that analyzes historical sunspot data or stock prices normally would not be called "digital signal processing" even though the types of calculations might be very similar.

15.2.1 Data Converters

In this chapter we will discuss only briefly several aspects of ADC and DAC specifications and performance that directly affect design decisions at the system level. The **Analog Basics** chapter has additional details that must be considered when doing an actual circuit design.

The first requirement when selecting a DAC or ADC is that it be able to handle the required *sample rate*. For communications-quality voice, a sample rate on the order of 8000 samples per second (8 ksps) should be adequate. For high-quality music, sample rates are typically an order of magnitude higher and for processing wideband RF signals, you'll need data converters with sample rates in the megasamples per second (Msps) range. In many systems the input and output sample rates are different. For example, a software-defined receiver might sample the input RF signal at 100 Msps while the output audio DAC is running at only 8 ksps.

The *resolution* of a data converter is expressed as the number of bits in the data words. For example, an 8-bit ADC can only represent the sampled analog signal as one of $2^8 = 256$ possible numbers. The smallest signal that it can resolve is therefore $\frac{1}{256}$ of full scale. Even with an ideal, error-free ADC, the *quantization error* is up to $\pm\frac{1}{2}$ of one least-significant bit (LSB) of the digital word, or $\pm\frac{1}{512}$ of full scale with 8-bit resolution. Similarly, a DAC can only generate the analog signal to within $\pm\frac{1}{2}$ LSB of the desired value. Later in the chapter we will discuss how to determine the required sample rate and resolution for a given application.

Another important data converter specification is the *spurious-free dynamic range* (SFDR). This is the ratio, normally expressed in dB, between a (usually) full-scale sine wave and the worst-case spurious signal. While higher-resolution ADCs and DACs tend to have better SFDR, that is not guaranteed. Devices that are intended for signal-processing applications normally specify the SFDR on the data sheet.

While sample rate, resolution and SFDR are the principal selection criteria for data converters in a DSP system, other parameters such as signal-to-noise ratio, harmonic and intermodulation distortion, full-power bandwidth, and aperture delay jitter can also affect performance. Of course, basic specifications such as power requirements, interface type (serial or parallel) and cost also determine a

device's suitability for a particular application. As with any electronic component, it is very important to read and fully understand the data sheet.

15.2.2 DSP

The term *digital signal processor* (DSP) is commonly understood to mean a special-purpose microprocessor with an architecture that has been optimized for signal processing. And indeed, in many systems the box labeled "DSP" in Fig 15.1 is such a device. A microprocessor has the advantage of flexibility because it can easily be re-programmed. Even with a single program, it can perform many completely-different tasks at different points in the code. On-chip hardware resources such as multipliers and other computational units are used efficiently because they are shared among various processes.

That is also the Achilles' heel of programmable DSPs. Any hardware resource that is shared among various processes can be used by only one process at a time. That can create bottlenecks that limit the maximum computation speed. Some DSP chips include multiple computational units or multiple *cores* (basically multiple copies of the entire processor) that can be used in parallel to speed up processing.

DIGITAL SIGNAL PROCESSING WITHOUT A "DSP"

Another way to speed up processing is to move all or part of the computations from the programmable DSP to an *application-specific integrated circuit* (ASIC), which has an architecture that has been optimized to perform some specific DSP function. For example, *direct digital synthesis* (DDS) frequency synthesizer chips are available that run at rates that would be impossible with a microprocessor-type DSP.

You could also design your own application-specific circuitry using a PC board full of discrete logic devices. Nowadays, however, it is more common to do that with a *programmable-logic device* (PLD). This is an IC that includes many general-purpose logic elements, but the connections between the elements are undefined when the device is manufactured. The user defines those connections by programming the device to perform whatever function is required. PLDs come in a wide variety of types, described by an alphabet soup of acronyms.

Programmable-array logic (PAL), *programmable logic array* (PLA), and *generic array logic* (GAL) devices are relatively simple arrays of AND gates, OR gates, inverters and latches. They are often used as "glue logic" to replace the miscellaneous discrete logic ICs that would otherwise be used to interface various larger digital devices on a circuit board. They are sometimes grouped under the general category of *small PLD* (SPLD). A *complex PLD* (CPLD) is similar but bigger, often consisting of an array of PALs with programmable interconnections between them.

A *field-programmable gate array* (FPGA) is bigger yet, with up to millions of gates per device. An FPGA includes an array of *complex logic blocks* (CLB), each of which includes some programmable logic, often implemented with a RAM *look-up table* (LUT), and output registers. *Input/output blocks* (IOB) also contain registers and can be configured as input, output, or bi-directional interfaces to the IC pins. The interconnections between blocks are much more flexible and complicated than in CPLDs. Some FPGAs also include higher-level circuit blocks such as general-purpose RAM, dozens or hundreds of hardware multipliers, and even entire on-chip microprocessors.

Some of the more inexpensive PLDs are *one-time programmable* (OTP), meaning you have to throw the old device away if you want to change the programming. Other devices are re-programmable or even *in-circuit programmable* (ICP) which allows changing the internal circuit configuration after the device has been soldered onto the PC board, typically under the control of an on-board microprocessor. That offers the best of both worlds, with speed nearly as fast as an ASIC but retaining many of the benefits of the reprogrammability of a microprocessor-type DSP. Most large FPGAs store their programming in *volatile memory*, which is RAM that must be re-loaded every time power is applied, typically by a ROM located on the same circuit board. Some FPGAs have programmable ROM on-chip.

Programming a PLD is quite different from programming a microprocessor. A microprocessor performs its operations sequentially — only one operation can be performed at a time. Writing a PLD program is more like designing a circuit. Different parts of the circuit can be doing different things at the same time. Special *hardware-description languages* (HDL) have been devised for programming the more complicated parts such as ASICs and FPGAs. The two most common industry-standard HDLs are Verilog and VHDL. (The arguments about which is "best" approach the religious fervor of the *Windows* vs *Linux* wars!) There is also a version of the C++ programming language called SystemC that includes a series of libraries that extend the language to include HDL functions. It is popular with some designers because it allows simulation and hardware description using the same software tool.

Despite the speed advantage of FPGAs, most amateurs use microprocessor-type devices for their DSP designs, supplemented with off-the-shelf ASICs where necessary. The primary reason is that the design process for an FPGA is quite complicated, involving obtaining and learning to use several sophisticated software tools. The steps involved in programming an FPGA are:

1. Simulate the design at a high abstraction level to prove the algorithms.
2. Generate the HDL code, either manually or using some tool.
3. Simulate and test the HDL program.
4. Synthesize the gate-level netlist.
5. Verify the netlist.
6. Perform a timing analysis.
7. Modify the design if necessary to meet timing constraints.
8. "Place and route" the chip design.
9. Program and test the part.

Table 15.1
PLD Manufacturers

Company	*Devices*	*URL*	*Notes*
Achronix	FPGA	**www.achronix.com**	High-speed FPGAs
Actel	FPGA	**www.actel.com**	Mixed-signal flash-based FPGAs
Altera	CPLD, FPGA, ASIC	**www.altera.com**	One of the two big FPGA vendors
Atmel	SPLD, CPLD, FPGA, ASIC	**www.atmel.com**	Fine-grain-reprogrammable FPGAs with AVR microprocessors on chip
Cypress Semiconductor	SPLD, CPLD	**www.cypress.com**	
Lattice Semiconductor	SPLD, CPLD, FPGA	**www.latticesemi.com**	Leading supplier of flash-based nonvolatile FPGAs
SiliconBlue	FPGA	**www.siliconbluetech.com**	Low-power FPGAs
Texas Instruments	SPLD, ASIC	**www.ti.com**	
Xilinx	CPLD, FPGA	**www.xilinx.com**	One of the two big FPGA vendors

Many of the software tools needed to perform those steps are quite expensive, although some manufacturers do offer free proprietary software for their own devices. Some principal manufacturers are listed in **Table 15.1**.

MICROPROCESSOR-TYPE DSP CHIPS

In contrast with designing an FPGA, programming a DSP chip is relatively easy. C compilers are available for most devices, so you don't have to learn assembly language. Typically you include a connector on your circuit board into which is plugged an *in-circuit programmer* (ICP), which is connected to a PC via a serial or USB cable. The software is written and compiled on the PC and then downloaded to the DSP. The same hardware often also includes an *in-circuit debugger* (ICD) so that the program can be debugged on the actual circuitry used in the design. The combination of the editor, compiler, programmer, debugger, simulator and related software is called an *integrated development environment* (IDE).

Until recently you had to use an *in-circuit emulator* (ICE), which is a device that plugs into the circuit board in place of the microprocessor. The ICE provides sophisticated debugging tools that function while the emulator runs the user's software on the target device at full speed. Nowadays, however, it is more common to use the ICD function that is built into many DSP chips and which provides most of the functions of a full-fledged ICE. It is much cheaper and does not require using a socket for the microprocessor chip.

The architecture of a digital signal processor shares some similarities to that of a general-purpose microprocessor but also differs in important respects. For example, DSPs generally don't spend much of their lives handling large computer files, so they tend to have a smaller memory address space than processors intended to be used in computers. On the other hand, the memory they do have is often built into the DSP chip itself to improve speed and to reduce pin count by eliminating the external address and data bus.

Most microprocessors use the traditional *Von Neumann architecture* in which the program and data are stored in the same memory space. However, most DSPs use a *Harvard architecture*, which means that data and program are stored in separate memories. That speeds up the processor because it can be reading the next program instruction at the same time as it is reading or writing data in response to the previous instruction. Some DSPs have two data memories so they can read and/or write two data words at the same time. Most devices actually use a modified Harvard architecture by providing some (typically slower and less convenient) method for the processor to read and write data to program memory.

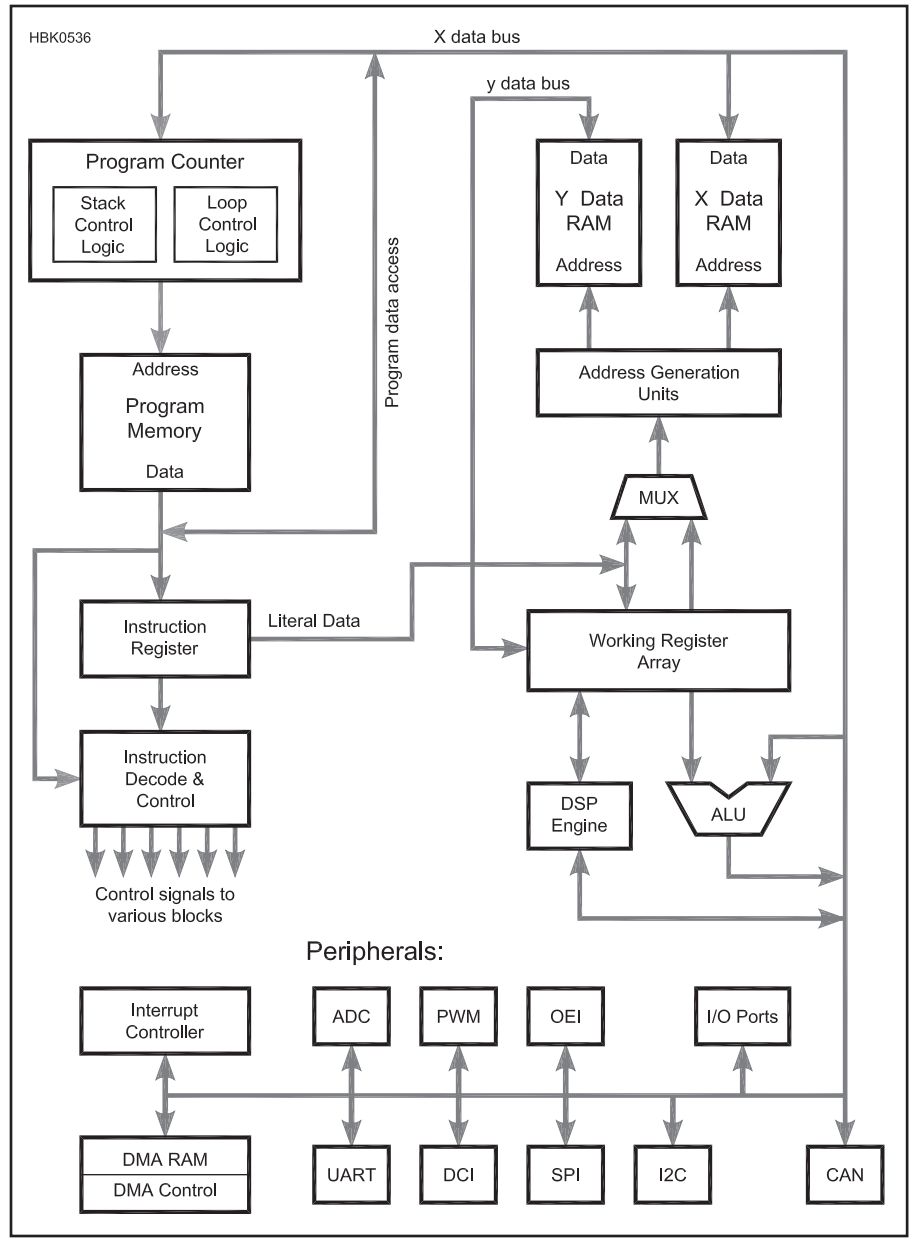

Fig 15.2 — Simplified block diagram of a dsPIC processor.

Probably the key difference between general-purpose and digital-signal processors is in the computational core, often called the *arithmetic logic unit* (ALU). The ALU in a traditional microprocessor only performs integer addition, subtraction and bitwise logic operations such as AND, OR, one-bit shifting and so on. More-complicated calculations, such as multiplication, division and operations with floating-point numbers, are done in software routines that exercise the simple resources of the ALU multiple times to generate the more-complicated results.

In contrast, a DSP has special hardware to perform many of these operations much faster. For example, the *multiplier-accumulator* (MAC) multiplies two numbers and adds (accumulates) the product with the previous results in a single step. Many common DSP algorithms involve the sum of a large number of products, so nearly all DSPs include this function. **Fig 15.2** is a simplified block diagram of the dsPIC series from Microchip. Its architecture is basically that of a general-purpose microcontroller to which has been added a DSP engine, which includes a MAC, a barrel shifter and other DSP features. It does use a modified Harvard architecture with two data memories that can be simultaneously accessed.

A *floating-point* number is the binary equivalent of scientific notation. Recall that the decimal integer 123000 is expressed as 1.23×10^5 in scientific notation. It is common

practice to place the decimal point after the first non-zero digit and indicate how many digits the decimal point must be moved by the *exponent* of ten, 5 in this case. The 1.23 part is called the *mantissa*. In a computer, base-2 binary numbers are used in place of the base-10 decimal numbers used in scientific notation. The *binary point* (equivalent to the decimal point in a decimal number) is assumed to be to the left of the first non-zero bit. For example the binary number 00110100 when converted to a 16-bit floating point number would have an 11-bit mantissa of 11010000000 (with the binary point assumed to be to the left of the first "1") and a 5-bit exponent of 00110 (decimal +6).

A floating point number can represent a signal with much more dynamic range than an integer number with the same number of bits. For example, a 16-bit signed integer can vary from –32768 to +32767. The difference between the smallest (1) and largest signal that can be represented is 20 log(65535) = 96 dB. If the 16 bits are divided into an 11-bit mantissa and 5-bit exponent, the available range is 20 log(2048) = 66 dB from the mantissa and 20 log(2^{32}) = 193 dB from the exponent for a total of 259 dB. The disadvantage is that the mantissa has less resolution, potentially increasing noise and distortion. Normally floating-point numbers are at least 32 bits wide to mitigate that effect.

Some DSPs can process floating-point numbers directly in hardware. Fixed-point DSPs can also handle floating-point numbers, but it must be done in software. The additional dynamic range afforded by floating-point processing is normally not needed for radio communications signals since the dynamic range of radio signals is typically less than can be handled by the 16-bit data words used by most integer DSPs. Using integer arithmetic saves the additional cost of a floating-point processor or the additional computational overhead of floating-point calculations on a fixed-point device. However, it requires careful attention to detail on the part of the programmer to make sure the signal can never exceed the maximum integer value or get so weak that the signal-to-noise ratio is degraded. If cost or computation time is not an issue, it is much easier to program in floating point since dynamic range issues can be ignored for most computations.

The term *pipeline* refers to the ability of a microprocessor to perform portions of several instructions at the same time. The sequence of operations required to perform an instruction is broken down into steps. Since each step is performed by a different chunk of hardware, different chunks can be working on different instructions at the same time. Most DSPs have at least a simple form of pipelining in which the next instruction is being fetched while the previous instruction is being executed. Some DSPs can do a multiply-accumulate while the next two multiplicands are being read from memory and the previous accumulated result is being stored so that the entire operation can occur in a single clock cycle. MACs per second is a common figure of merit for measuring DSP speed. For conventional microprocessors, a more common figure of merit is millions of instructions per second (MIPS) or floating-point operations per second (FLOPS).

Many DSPs have a sophisticated address generation unit that can automatically increment one or more data memory pointers so that repetitive calculations can step through memory without the processor having to calculate the addresses. *Zero-overhead looping* is the ability to automatically jump the address pointer back to the beginning of the array when it reaches the end. That saves several microprocessor instructions per loop that normally would be required to check the current address and jump when it reaches a predetermined value.

While most DSPs do not include a full hardware divider, some do include special instructions and hardware to speed up division calculations. A *barrel shifter* is another common DSP feature. It allows shifting a data word a specified number of bits in a single clock cycle. *Direct memory access* (DMA) refers to special hardware that can automatically transfer data between memory and various peripheral devices or ports without processor overhead.

DSP IN EMBEDDED SYSTEMS

An *embedded system* is a device that is not a computer but nevertheless has a micropro-

Table 15.2
Manufacturers of Embedded DSPs

Company	*Family*	*Data Bits*	*Speed MMACs*	*Nr.of Cores*	*ROM (bytes)*	*RAM (bytes)*	*Notes*
Analog Devices **www.analog.com**	ADSP-21xx	16	75-160	1	12k-144k	8k-112k	Easy assembly language
	SHARC	32/40 fp	300-900	1	2-4M	0.5-5M	Runs fixed or floating point
	Blackfin	16/32	400-2400	1-2	External	53k-328k	Many on-chip peripherals
Cirrus Logic	CS48xxxx	32	150	1		96k	Audio applications
www.cirrus.com	CS49xxxx	32	300	2	512k	296k-328k	Audio applications
Freescale	DSP568xx	16	32-120	1	2k-576k	2k-128k	Also a microcontroller
www.freescale.com	DSP563xx	24	80-275	1	External	576k	
	StarCore	16	1000-48,000	1-6	External	0-1436k	
Microchip **www.microchip.com**	dsPIC	16	30-70	1	6k-256k	256-32k	Also a microcontroller Free IDE software
Texas Instruments	C5000	16	50-600	1	8k-256k	0-1280k	
www.ti.com	C6000	16/64 fp	300-24,000	1-3	0-384k	32k-3072	Fixed or floating point ver.
Zilog **www.zilog.com**	Z89xxx	16	20	1	4k-8k	512	

cessor or DSP chip embedded somewhere in its circuitry. Examples are microwave ovens, automobiles, mobile telephones and software-defined radios. DSPs intended for embedded systems often include a wide array of on-chip peripherals such as various kinds of timers, multiple hardware interrupts, serial ports of various types, a real-time clock, pulse-width modulators, optical encoder interfaces, A/D and D/A converters and lots of general-purpose digital I/O pins. Some DSPs not only include lots of peripherals but in addition have architectures that are well-suited for general-purpose control applications as well as digital signal processing.

Table 15.2 lists some manufacturers of DSP chips targeted to embedded systems. It should be mentioned that microprocessors intended for personal computers made by Intel and AMD also include extensive DSP capability. However, they are large, complicated, power-hungry ICs that are not often used in embedded applications.

When selecting a DSP device for a new design, often the available development environment is more important than the characteristics of the device itself. Microchip's dsPIC family of DSPs was chosen for the examples in this chapter because their integrated development environment is extensive and easy to use and the IDE software is available for free download from their Web site.[2] The processor instruction set is a superset of the PIC24 family of general-purpose microcontrollers, with which many hams are already familiar. The company offers a line of low-cost evaluation boards and starter kits as well as an inexpensive in-circuit debugger, the ICD 3. The free IDE software includes a simulator that can run dsPIC software on a PC (at a much slower rate, of course), so that you can experiment with DSP algorithms before buying any hardware.

The Microchip DSP family is limited to 70 million instructions per second. In a system with, say, a 70 kHz sample rate, 1000 instructions per sample are available which should be plenty if the calculations are not too complex. However if the sample rate is 1 MHz, then you get only 70 instructions per sample, which likely would be insufficient.

If more horsepower is required, you'll need to select a processor from a different manufacturer. Look for one with a well-integrated suite of development software that is powerful and easy to use. Also check out the cost and availability of development hardware such as evaluation kits, programmers and debuggers. Once those requirements are met, then you can move on to selecting a specific device with the performance and features required for your application. It can be helpful at the beginning of a project to first write some of the key software routines and test them on a simulator to estimate execution times, in order to determine how powerful a processor is needed.

When estimating execution time, don't forget to include the effect of interrupts. Most DSP systems require real-time response and make extensive use of interrupts to ensure that certain events happen at the correct times. Although this is hidden from the programmer's view when programming in C, the interrupt service routines contain quite a bit of overhead each time they are called (to save the processor state when responding and to recall the state just before returning from the interrupt). Sometimes an interrupt may be called more often than you expect, which can eat up processor cycles and so increase the execution time of other unrelated routines.

In the past, may embedded systems were written in assembly language so save memory and increase processing speed. Many early microprocessors and DSPs did not have enough memory to support a high-level language. Today, most processors have sufficient memory and processing speed to support a C kernel and library without difficulty. For anything but the simplest of programs, it is not only faster and easier to develop software in C but it is easier to support and maintain as well, especially if people other than the original programmer might become involved. Far more people know the C programming language than any particular processor's assembly language. It is true that the version of C used on a DSP chip is usually modified from standard ANSI C to support specific hardware features, but it would still be far easier to learn for a programmer familiar with writing C code on a PC or on a different DSP.

A common technique is first to write the entire application in C. Then, if execution time is not acceptable, analyze the system to determine in which software routines the bottlenecks are occurring. Those routines can then be re-written in assembly language. Having an already-working version written in C (even if too slow) can be helpful in testing and troubleshooting the equivalent assembly language.

15.3 Digital Signals

Digital signals differ from analog signals in two ways. One is that they are digitized in time, a process called *sampling*. The other is that they are digitized in amplitude, a process called *quantization*. Sampling and quantization affect the digitized signal in different ways so the following sections will consider their effects separately.

15.3.1 Sampling — Digitization in Time

Sampling is the process of measuring a signal at discrete points of time and recording the measured values. An example from history is recording the number of sunspots. If an observer goes out at noon every day and writes down the number of observed sunspots, then that data can be used to plot sunspot number versus time. In this case, we say the *sample rate* is one sample per day. The data can then be analyzed in various ways to determine short and long-term trends. After recording only a few months of data it will quickly become apparent that sunspot number has a marked periodicity — the numbers tend to repeat every 27 days (which happens to be the rotation rate of the sun as seen from earth).

What if, instead of taking a reading once a day, the readings were taken only once per month? With a 30-day sample period, the 27-day periodicity would likely be impossible to see. Clearly, the sample rate must be at least some minimum value to accurately represent the measured signal. Based on earlier work by Harry Nyquist, Claude Shannon proved in 1948 that in order to sample a signal without loss of information, the sample rate must be greater than the *Nyquist rate*, which is two times the bandwidth of the signal. In other words, the bandwidth must be less than the *Nyquist frequency*, which is one-half the sample rate. This is known as the *Nyquist sampling criterion*.

That simple rule has some profound implications. If all the frequency components of a signal are contained within a bandwidth of B Hz, then sampling at a rate greater than 2B

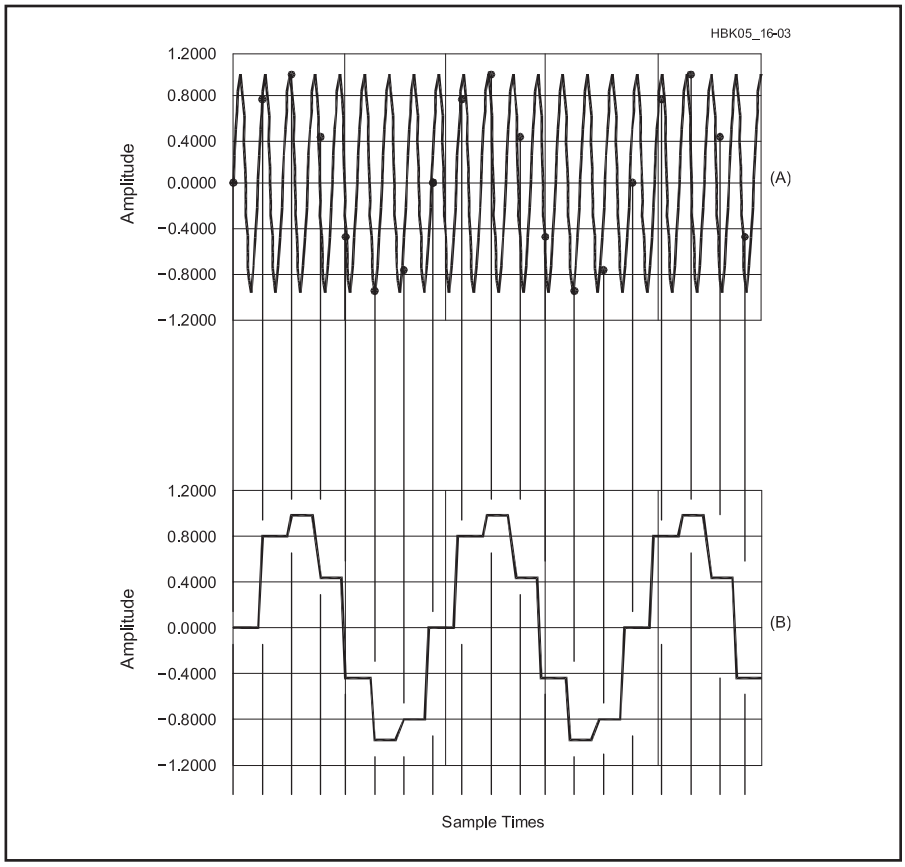

Fig 15.3 — Undersampled sine wave (A). Samples aliased to a lower frequency (B).

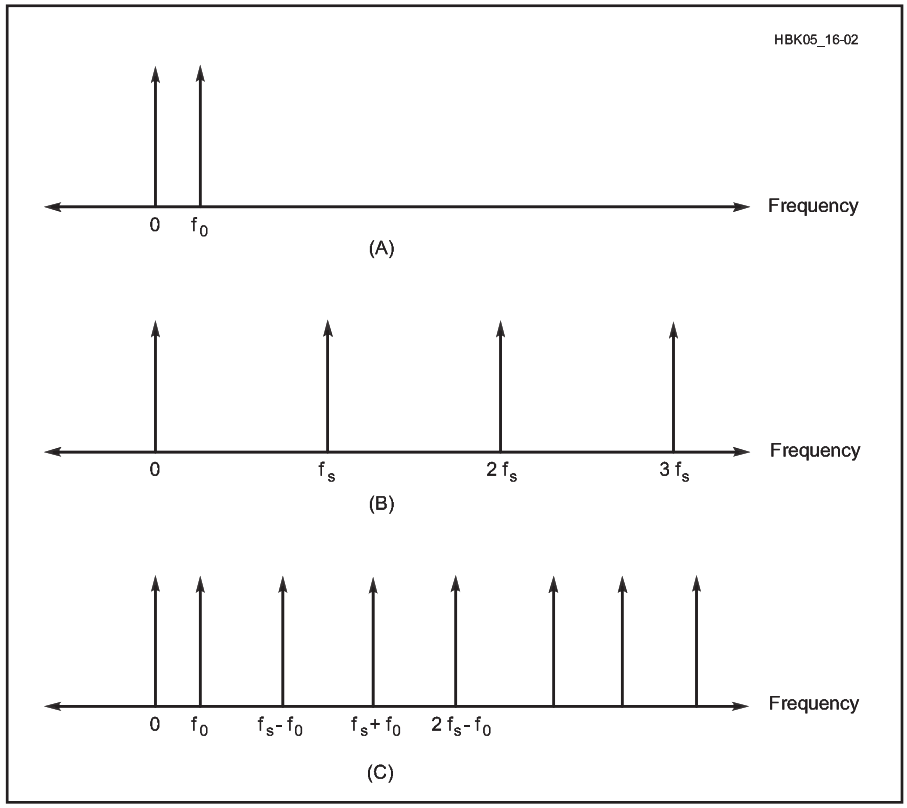

Fig 15.4 — Spectrum of an analog sine wave (A). The spectrum of the sampling function, including all harmonics (B). The spectrum of the sampled sine wave (C).

samples per second is sufficient to represent the signal with 100% accuracy and with no loss of information. It is theoretically possible to convert the samples back to an analog signal that is exactly identical to the original.

Of course, a real-world digital system measures those samples with only a finite number of bits of resolution, with consequences that we will investigate in the section on quantization that follows. In addition, sampling theory assumes that there is absolutely no signal energy outside the specified bandwidth; in other words the stopband attenuation is infinity dB. Any residual signal in the stop-band shows up as distortion or noise in the sampled signal.

To simplify the discussion, let's think about sampling a signal of a single frequency (a sine wave). **Fig 15.3** illustrates what happens if the sample rate is too low. As shown, the sample rate is approximately ⅞ the sine-wave frequency. You can see that the sampled signal has a period about 8 times greater than the period of the sine wave, or ⅛ the frequency. The samples are the same as if the analog signal had been a sine wave of ⅛ the actual frequency.

That is an example of a general principle. If the sample rate is too low, the sampled signal will be *aliased* to a frequency equal to the difference between the actual frequency of the analog signal and the sample rate. In the above example, the alias frequency f_o is

$$f_o = f_{sig} - f_s = \left(1 - \frac{7}{8}\right) f_{sig} = \left(\frac{1}{8}\right) f_{sig}$$

where f_{sig} is the frequency of the signal before sampling and f_s is the sample rate.

If the analog signal's frequency is even higher, then it aliases relative to whichever harmonic of the sample rate is closest. **Fig 15.4C** shows all the signal frequencies that alias to a frequency of f_o, calculated from the equation

$$f_o = \left| f_{sig} - N f_s \right|$$

where N is the harmonic number. One way to think of it is that a sampler is a harmonic mixer. The sampled signal (equivalent to the mixer output) contains the sum and difference frequencies of the input signal and all the harmonics of the sample frequency.

To avoid aliasing, most systems use an *anti-aliasing filter* before the sampler, as shown in **Fig 15.5**. For a baseband signal (one that extends to zero Hz), the anti-aliasing filter is a low-pass filter whose stopband extends from the Nyquist frequency to infinity. Of course, practical filters do not transition instantaneously from the passband to the stopband, so the bandwidth of the passband must be somewhat less than half the sample rate.

It is actually possible to accurately sample signals above the Nyquist frequency so long

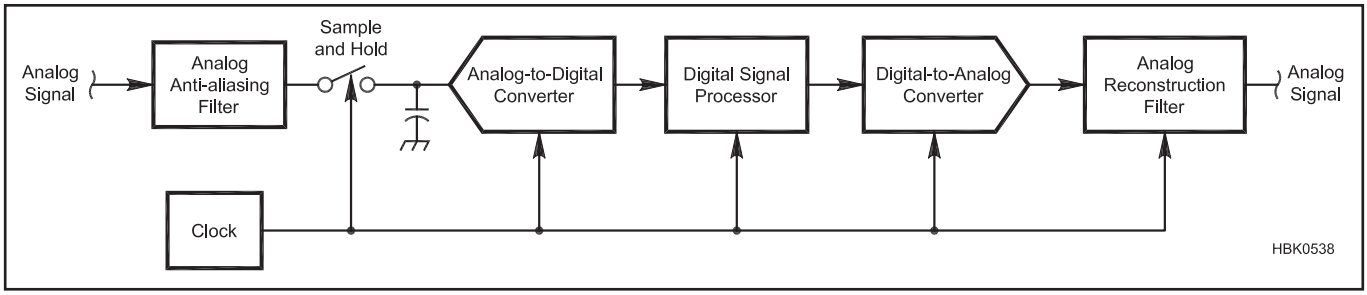

Fig 15.5 — A more complete block diagram of a DSP system.

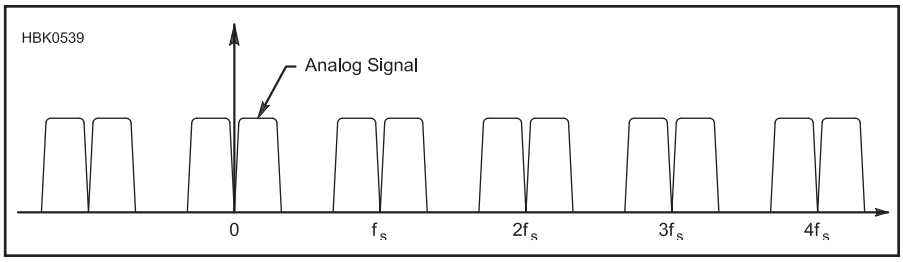

Fig 15.6 — An ideal sampled signal repeats the spectrum of the analog signal at all harmonics of the sample rate, f_s.

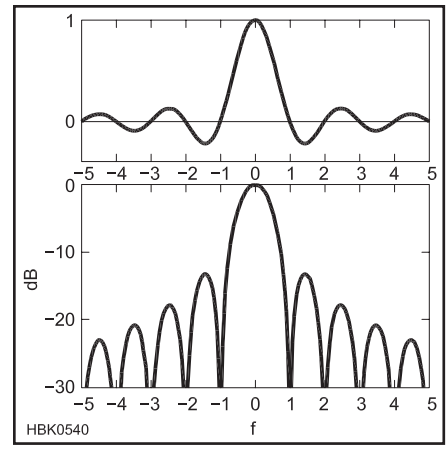

Fig 15.7 — The sinc function, where the horizontal axis is frequency normalized to the sample rate. At the bottom is the same function in decibels.

as their bandwidth does not violate the Nyquist criterion, a process called *undersampling* or *harmonic sampling*. Imagine an LSB signal at 455 kHz with a bandwidth of 3 kHz that is being sampled at a 48 kHz rate. The 455 kHz signal mixes with the ninth harmonic of the sample rate at 432 kHz, resulting in a sampled signal with its suppressed carrier at 455 – 432 = 23 kHz and extending 3 kHz below that to 20 kHz. So long as the incoming signal has no significant energy below 432 kHz or above 456 kHz [432 + (48/2)] kHz no unwanted aliasing occurs.

With harmonic sampling, the anti-alias filter must be a band-pass type. In the previous example, you'd probably need to use a crystal or mechanical filter in order to have a sufficiently sharp transition from the top edge of the passband slightly below 455 kHz to the stopband edge at 456 kHz.

Fig 15.3 shows each sample being held at a constant value for the duration of one sample period. However, sampling theory actually assumes that the sample is only valid at the instant the signal is sampled; it is zero or undefined at all other times. A series of such infinitely-narrow impulses has harmonics all the way to infinite frequency. Each harmonic has the same amplitude and is modulated by the signal being sampled. See **Fig 15.6**. When a digitized signal is converted back to analog form, unwanted harmonics must be filtered out by a *reconstruction filter* as shown in Fig 15.5. This is similar to the anti-aliasing filter used at the input in that its bandwidth should be no greater than one-half the sample rate. It is a low-pass filter for a baseband signal and a band-pass filter for an undersampled signal.

Most DACs actually do hold each sample value for the entire sample period. This is called *zero-order hold* and results in a frequency response in the shape of a sinc function

$$\mathrm{sinc}(f) = \frac{\sin(\pi f)}{\pi f}$$

where f is normalized to the sample rate, f = frequency / sample rate.

The graph of the sinc function in **Fig 15.7** shows both positive and negative frequencies for reasons explained in the Analytic Signals section. Note that the logarithmic frequency response has notches at the sample rate and all of its harmonics. If the signal bandwidth is much less than the Nyquist frequency, then most of the signal at the harmonics falls near the notch frequencies, easing the task of the reconstruction filter. If the signal bandwidth is small enough (sample rate is high enough), the harmonics are almost completely notched out and a reconstruction filter may not even be required.

The $\sin(\pi f)/\pi f$ frequency response also affects the passband. For example if the passband extends to sample rate / 4 (f = 0.25), then the response is

$$20 \log \frac{\sin(\pi \cdot 0.25)}{\pi \cdot 0.25} = -0.9 \text{ dB}$$

at the top edge of the passband. At the Nyquist frequency, (f = 0.5), the error is 3.9 dB. If the signal bandwidth is a large proportion of the Nyquist frequency, then some kind of digital or analog compensation filter may be required to correct for the high-frequency rolloff.

DECIMATION AND INTERPOLATION

The term *decimation* simply means reducing the sample rate. For example to decimate by two, simply eliminate every second sample. That works fine as long as the signal bandwidth satisfies the Nyquist criterion at the lower, output sample rate. If the analog anti-aliasing filter is not narrow enough, then a digital anti-aliasing filter in the DSP can be used to reduce the bandwidth to the necessary value. This must be done *before* decimation to satisfy the Nyquist criterion.

If you need to decimate by a large amount, then the digital anti-aliasing filter must have a very small bandwidth compared to the sample rate. As we will see later, a digital filter with a small bandwidth is computationally intensive. For this reason, large decimation factors are normally accomplished in multiple steps, as shown in **Fig 15.8A**. The first decimation is by a small factor, typically 2, so that the first anti-alias filter can be as simple as

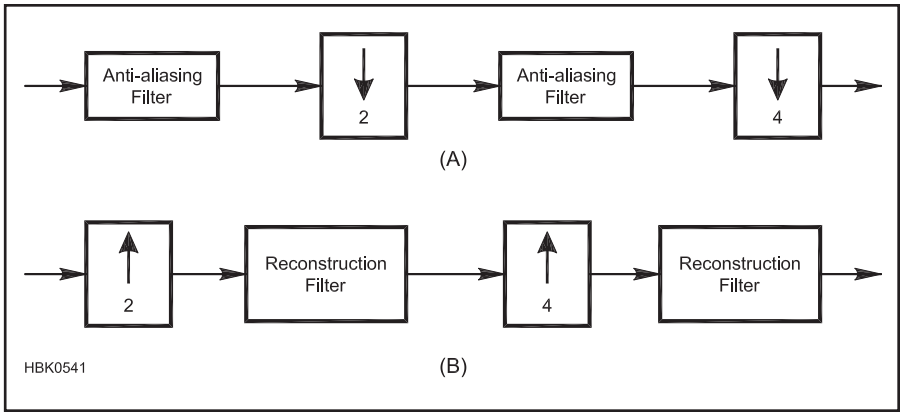

Fig 15.8 — Decimation (A) and interpolation (B). The arrow's direction indicates decimation (down) or interpolation (up) and the number is the factor.

possible. The second decimation stage then does not have to decimate by such a large factor, simplifying its task. In addition, since it is running at only half the input sample rate it has more time to do its calculations. Generally it is most efficient to decimate by the smallest factor in the first stage, a larger factor in the second, and the largest factors in the third and any subsequent stages. The larger the total decimation factor, the greater the number of stages is appropriate but more than three stages is uncommon.

Interpolation means increasing the sample rate. One way to do that is simply to insert additional zero-value samples, a process called *zero-stuffing*. For example, to interpolate by a factor of three, insert two zero-value samples after each input sample. That works, but may not give the results you expect. Recall that a sampled signal has additional copies of the baseband signal at all harmonics of the sample rate. All of those harmonics remain in the resampled signal, even though the sample rate is now higher. To eliminate them, the signal must be filtered after interpolation. After filtering, there is signal only at baseband and around the harmonics of the interpolated (higher-frequency) sample rate. It's as if the analog signal had been sampled at the higher rate to begin with.

Just as with decimation, interpolation by a large factor is best done in stages, as shown in Fig 15.8B. In this case, the stage running at the lowest sample rate (again the first stage) is the one with the lowest interpolation factor.

Zero-stuffing followed by filtering is not the only way to interpolate. Really what you are trying to do is to fill in between the lower-rate samples with additional samples that "connect the dots" in as smooth a manner as possible. It can be shown that that is mathematically equivalent to zero-stuffing and filtering. For example, if instead of inserting zero-value samples you instead simply repeat the last input sample, you have a situation similar to the zero-order hold of a DAC output. It is equivalent to zero-stuffing followed by a low-pass filter with a frequency response of $\sin(\pi f)/\pi f$. If you do a straight-line interpolation between input samples (a "first-order" interpolation), it turns out that it is equivalent to a low pass filter with a frequency response of $[\sin(\pi f)/\pi f]^2$, which has a sharper cutoff and better stop-band rejection than a zero-order interpolation. Higher-order interpolations have smoother responses in the time domain which translate to better filter responses in the frequency domain.

So far we have only covered decimation and interpolation by integer factors. It is also possible to change the sample rate by a non-integer factor, which is called *resampling* or *multi-rate* conversion. For example, if you want to increase the sample rate by a factor of ⅓, simply interpolate by 4 and then decimate by 3. That method can become impractical for some resample ratios. For example, to convert an audio file recorded from a computer sound card at 48 kHz to the 44.1 kHz required by a compact disc, the resample ratio is 44,100 / 48,000 = 147 / 160. After interpolation by 147, the 44.1 kHz input file is sampled at 6.4827 MHz, which would result in excessive processing overhead.

In addition, the interpolation/decimation method only works for resample ratios that are rational numbers (the ratio of two integers). To resample by an irrational number, a different method is required. The technique is as follows. For each output sample, first determine the two nearest input samples. Calculate the coefficients of the Nth-order equation that describes the trajectory between the two input samples. Knowing the trajectory between the input samples and the output sample's relative position between them, the value of the output sample can be calculated from the equation.

15.3.2 Quantization — Digitization in Amplitude

While sampling (digitization in time) theoretically causes no loss of signal information, quantization (digitization in amplitude) always does. For example, an 8-bit signed number can represent a signal as a value from –128 to +127. For each sample, the A/D converter assigns whichever number in that range is closest to the analog signal at that instant. If a particular sample has a value of 10, there is no way to tell if the original signal was 9.5, 10.5 or somewhere in between. That information has been lost forever.

When quantizing a complex signal such as speech, this error shows up as noise, called *quantization noise*. See **Fig 15.9**. The error is random — it is equally likely to be anywhere in the range of –1/2 to +1/2 of a single step of the ADC. We say that the maximum error is one-half of one *least-significant bit* (LSB). It can be shown mathematically that a series of uniformly-distributed random numbers between +0.5 LSB and –0.5 LSB has an RMS value of

$$\frac{\text{LSB}}{\sqrt{12}}$$

which is –10.79 dB less than one LSB. Each time you add one bit to the data word, the number of LSBs in the range doubles, which means each LSB gets two times smaller reducing the noise by 6.02 dB. A full-scale sine wave has an RMS power –3.01 dB from a full-scale dc voltage. Combining that information results in the following equation for signal-to-noise ratio in decibels for a data word of width b bits:

$$\text{SNR} = 1.76 + 6.02\text{b} \ \text{dB}$$

For example, with 8-bit data, SNR = 49.9 dB. An ideal 16-bit ADC would achieve a 98.1 dB signal-to-noise ratio. Of course, real-world devices are never perfect so actual performance would be somewhat less.

One critical point that is sometimes overlooked is that quantization noise is spread over the entire bandwidth from zero Hz to the sample rate. If you are digitizing a 3 kHz audio channel with a 48 ksps sampler, only a fraction of the noise power is within the channel. For that reason, the effective signal-to-noise ratio depends not only on the number of bits but also the sample rate, f_s, and the signal bandwidth, B:

$$\text{SNR}_{\text{eff}} = \text{SNR} + 10\log\left(\frac{f_s}{2B}\right) \ \text{dB}$$

The reason for the factor of two in the denominator is that the bandwidth of a positive-frequency scalar signal should be compared to the Nyquist bandwidth, $f_s/2$. When filtering

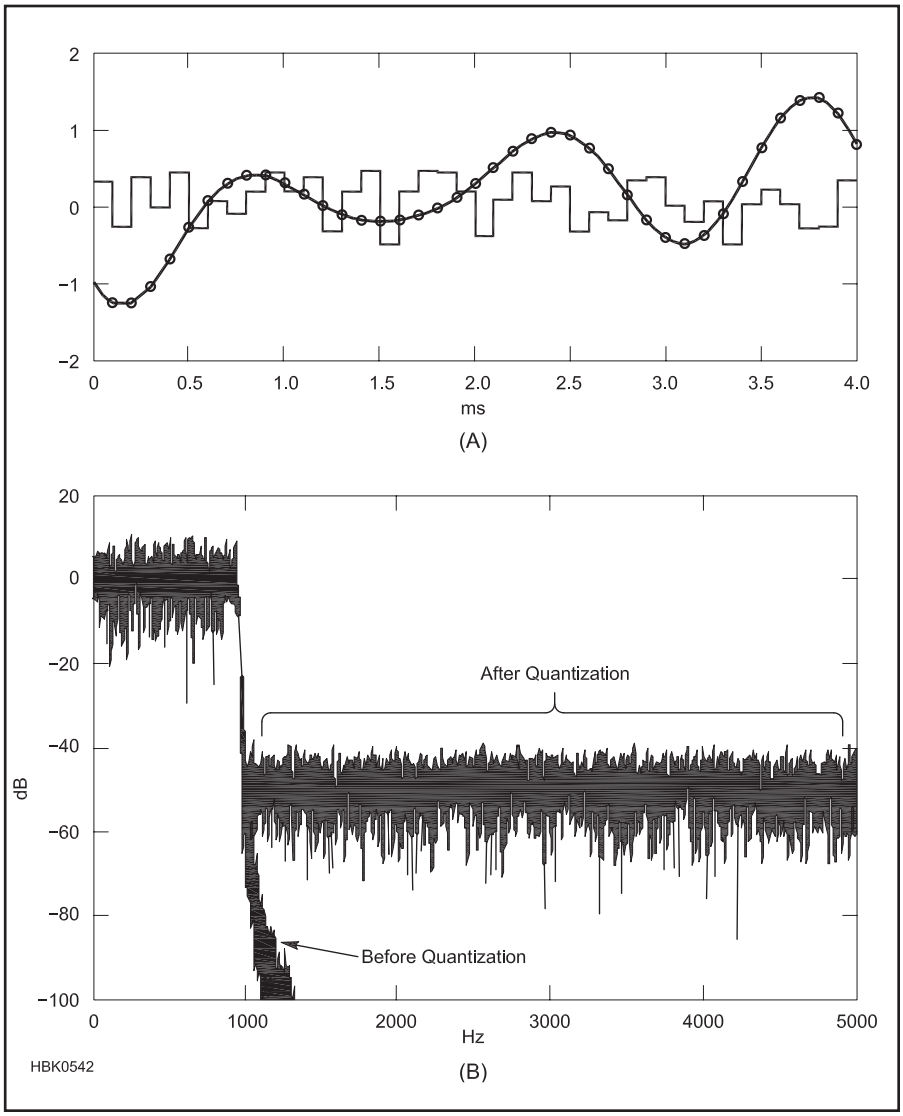

Fig 15.9 — Quantization error of a random noise signal that has been band-limited to 1 kHz to simulate an audio signal. (A) The sampler resolution is 8 bits and the sample rate is 10 kHz. Sample values are indicated by circles. Also shown is the quantization error, in units of LSB. Below is the frequency spectrum of the signal before and after quantization. (B)

a complex signal (one that contains I and Q parts), the 2B in the denominator should be replaced by B.

When choosing an A/D converter don't forget that the effective SNR depends on the sample rate. As an example, let's compare an Analog Devices AD9235 12-bit, 65 Msps ADC to an AD7653, which is a 16-bit 100 ksps ADC from the same manufacturer. Assume a 10 kHz signal bandwidth.

An ideal 12-bit ADC has a SNR of 1.76 + 6.02 × 12 = 74.0 dB. The AD9235's performance is not far from the ideal; its SNR is specified at 70.5 dB at its 65 Msps maximum sample rate. In a 10 kHz bandwidth, the effective SNR is 70.5 + 10 log (65,000/20) = 105.6 dB.

An ideal 16-bit ADC has an SNR of 98.1 dB. The AD7653 is specified at 86 dB. The effective SNR is 86 + 10log(100/20) = 93 dB.

So the 12-bit ADC with 70.5 dB SNR is actually 12.6 dB better than the 16-bit device with 86 dB SNR! Even an ideal 16-bit, 100 ksps ADC would only have an effective SNR of 98.1 + 10log(100/20) = 105.1 dB, still worse than the actual performance of the AD9235 when measured with the same bandwidth. Note that to actually realize 105.6 dB of dynamic range the signal from the ADC would need to be filtered to a 10 kHz bandwidth while increasing the bits of data resolution.

Oversampling is the name given to the technique of using a higher-than-necessary sample rate in order to achieve an improved S/N ratio. Don't forget that when the high-sample-rate signal is decimated the data words must have enough bits to support the higher dynamic range at the lower sample rate. As a rule of thumb, the quantization noise should be at least 10 dB less than the signal noise in order not to significantly degrade the SNR. In the AD9235 example, assuming a 100 kHz output sample rate, about 18 bits would be required: 1.76 + 6.02 × 18 + 10log(100/20) = 117.1 dB, which is 11.5 dB better than the 105.6 dB dynamic range of the ADC in a 10 kHz bandwidth.

Most ADCs and DACs used in high-fidelity audio systems use an extreme form of oversampling, where the internal converter may oversample by a rate of 128 or 256 times, but with very low resolution (in some cases just a 1-bit ADC!). In addition, such converters use a technique called noise shaping to push most of the quantization noise to frequencies near the sample rate, and reduce it in the audio spectrum. The noise is then removed in the decimation filter.

Although quantization error manifests itself as noise when digitizing a complex non-periodic signal, it can show up as discrete spurious frequencies when digitizing a periodic signal. **Fig 15.10** illustrates a 1 kHz sine wave sampled with 8-bit resolution at a 9.5 kHz rate. On average there are 9.5 samples per cycle of the sine wave so that the sampling error repeats every second cycle. That 500-Hz periodicity in the error signal causes a spurious signal at 500 Hz and harmonics. As the signal frequency is changed, the spurs move around in a complicated manner that depends on the ratio of sample rate to signal frequency. In real-world ADCs, nonlinearities in the transfer function can also create spurious signals that vary unpredictably as a function of the signal ampli-tude, especially at low signal levels.

In many applications, broadband noise is preferable to spurious signals on discrete frequencies. The solution is to add *dithering*. Essentially this involves adding a small amount of noise, typically on the order of an LSB or two, in order to randomize the quantization error. Some DACs have dithering capability built in to improve the SFDR, even though it does degrade the SNR slightly. Dithering is also useful in cases where the input signal to an ADC is smaller than one LSB. Even though the signal would be well above the noise level after narrow-band filtering, it cannot be detected if the ADC input is always below one LSB. In many systems there is sufficient noise at the input, both from input amplifiers as well as from the ADC itself, to cause natural dithering.

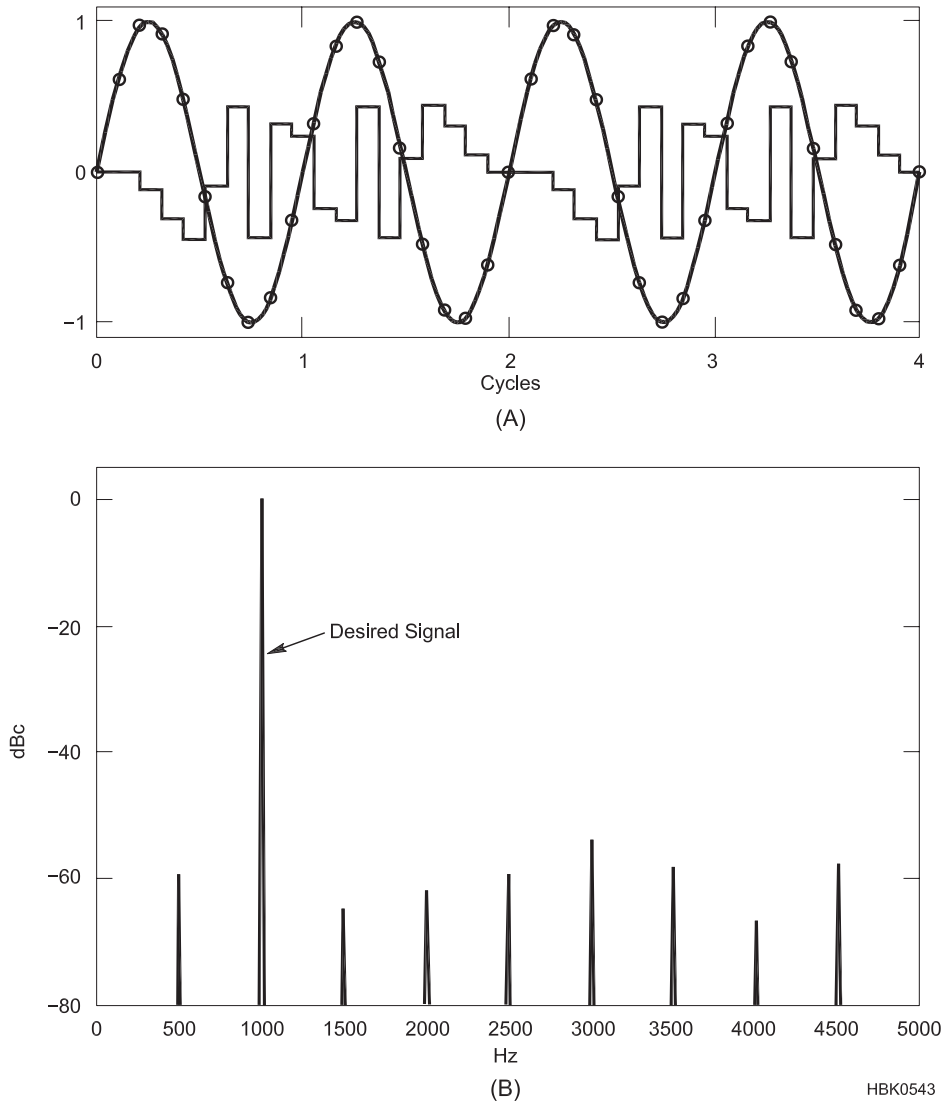

Fig 15.10 — Quantization error of a 1 kHz sine wave sampled at 9.5 kHz with 8-bit resolution (A). Sample values are indicated by circles. Also shown is the quantization error, in units of LSB. Below is the frequency spectrum, showing the spurious frequencies caused by the quantization. (B)

15.4 Digital Filters

As radio amateurs, most of us are well-acquainted with the concept of frequency. We know, for example, that a pure sine wave consists of a single frequency, which is inversely proportional to the wavelength. If the sine wave is distorted, additional harmonic frequencies appear at integer multiples of the fundamental. For example, a square wave consists of sine waves at the fundamental frequency and all the odd harmonics. In general, any periodic waveform can be decomposed into a combination of sine waves at various phase angles with frequencies that are integer multiples of the repetition rate of the waveform.

Even a non-periodic waveform can be decomposed into sine waves, although in this case they are not harmonically-related. For example a single pulse of width τ seconds has a frequency spectrum proportional to $\text{sinc}(f\tau) = \sin(\pi f\tau)/(\pi f\tau)$. You can think of this as an infinite number of sine waves spaced infinitely closely together with amplitudes that trace out that spectral shape. It is interesting to note that if τ is decreased, the value of f must increase by the same factor for any given value of $\sin(\pi f\tau)/(\pi f\tau)$. In other words, the narrower the pulse the wider the spectrum. Of course that applies to sine waves and other periodic waveforms as well — the smaller the wavelength the higher the frequency. In general, anything that makes the signal "skinnier" in the time domain makes it "fatter" in the frequency domain and vice versa.

As the pulse becomes narrower and narrower, the frequency spectrum spreads out more and more. In the limit, if the pulse is made infinitely narrow, the spectrum becomes flat from zero hertz to infinity. An infinitely-narrow pulse is called an *impulse* and is a very useful concept because of its flat frequency spectrum. If you feed an impulse into the input of a filter, the signal that comes out, the *impulse response*, has a frequency spectrum equal to the frequency response of the filter. One way to design a filter is to determine the impulse response that corresponds to the desired frequency spectrum and then design the filter to have that impulse response. That method is ideally suited for designing FIR filters.

15.4.1 FIR Filters

A *finite impulse response* (FIR) filter is a filter whose impulse response is finite, ending in some fixed time. Note that analog filters have an infinite impulse response — the out-

put theoretically rings forever. Even a simple R-C low-pass filter's output dies exponentially toward zero but theoretically never quite reaches it. In contrast, an FIR filter's impulse response becomes exactly zero at some time after receiving the impulse and stays zero forever (or at least until another impulse comes along).

Given that you have somehow figured out the desired impulse response, how would you design a digital filter to have that response? The obvious method would be to pre-calculate a table of impulse response values, sampled at the sample rate. These are called the filter *coefficients*. When an impulse of a certain amplitude is received, you multiply that amplitude by the first entry in the coefficient table and send the result to the output. At the next sample time, multiply the impulse by the second entry, and so on until you have used up all the entries in the table.

A circuit to do that is shown in **Fig 15.11**. The input signal is stored in a shift register. Each block labeled "Delay" represents a delay of one sample time. At each sample time, the signal is shifted one register to the right. Each register feeds a multiplier and the other input to the multiplier comes from one of the coefficient table entries. All the multiplier outputs are added together. Since the input is assumed to be a single impulse, at any given time all the shift registers contain zero except one, which is multiplied by the appropriate table entry and sent to the output.

We've just designed an FIR filter! By using a shift register with a separate multiplier for each tap, the filter works for continuous signals as well as for impulses. Since this is a linear system, the continuing signal is affected by the filter the same as an individual impulse.

It should be obvious from the diagram how to implement an FIR filter in software. You set up two buffers in memory, one for the filter coefficients and one for the data. The length of each buffer is the number of filter taps. (A *tap* is the combination of one filter coefficient, one shift register and one multiplier/accumulator.) Each time a new data value is received, it is stored in the next available position in the data buffer and the accumulator is set to zero. Next, a software loop is executed a number of times equal to the number of taps. During each loop, pointers to the two buffers are incremented, the next coefficient is multiplied by the next data value and the result is added to the current accumulator value. After the last loop, the accumulator contents are the output value. Normally the buffers are implemented as *circular buffers* — when the address pointer gets to the end it is reset back to the beginning.

Now you can see why a hardware multiplier-accumulator (MAC) is such an important feature of a DSP chip. Each tap of the FIR filter involves one multiplication and one addition. With a 1000-tap FIR filter, 1000 multiplications and 1000 additions must be performed during each sample time. Being able to do each MAC in a single clock cycle saves a lot of processing time.

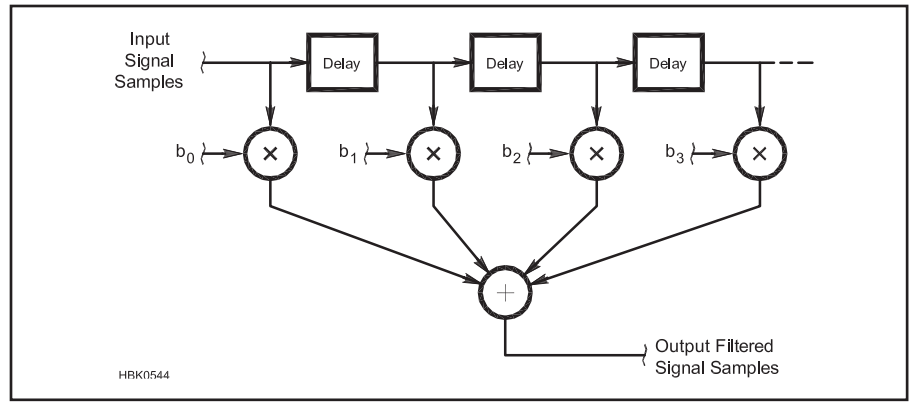

Fig 15.11 — A 4-tap FIR filter. The b_n values are the filter coefficients.

An FIR filter is a hardware or software implementation of the mathematical operation called *convolution*. We say that the filter *convolves* the input signal with the impulse response of the filter. It turns out that convolution in the time domain is mathematically equivalent to multiplication in the frequency domain. That means that the frequency spectrum of the output equals the frequency spectrum of the input times the frequency spectrum of the filter. Expressed in decibels, the output spectrum equals the input spectrum plus the filter frequency response, all in dB. If at some frequency the input signal is +3 dB and the filter is –10 dB compared to some reference, then the output signal will be 3 – 10 = –7 dB at that frequency.

An FIR filter whose bandwidth is very small compared to the sample rate requires a long impulse response with lot of taps. This is another consequence of the "skinny" versus "fat" relationship between the frequency and time domains. If the filter is narrow in the frequency domain, then its impulse response is wide. Actually, if you want the frequency response to go all the way to zero (minus infinity dB) throughout the stop band, then the impulse response theoretically becomes infinitely wide. Since we're designing a *finite* impulse response filter we have to truncate the impulse response at some point to get it to fit in the coefficient table. When you do that, however, you no longer have infinite attenuation in the stopband. The more heavily you truncate (the narrower the impulse response) the worse the stopband attenuation and the more ripple you get in the passband. Assuming optimum design techniques for selecting coefficients, you can estimate the minimum length L of the impulse response from the following equation:

$$L = 1 - \frac{10\log(\delta_1\delta_2) - 15}{14\left(\frac{f_T}{f_s}\right)} \text{ taps}$$

where

δ_1 and δ_2 = the passband and stopband ripple expressed as a fraction

f_T = the transition bandwidth (frequency difference between passband and stopband edges)

f_s = the sample rate.

For example, for a low-pass filter with a passband that extends up to 3 kHz, a stopband that starts at 4 kHz ($f_T = 4 - 3 = 1$ kHz), f_s = 10 kHz sample rate, ±0.1 dB passband ripple ($\delta_1 = 10^{0.1/20} - 1 = 0.0116$), and 60 dB stopband rejection ($\delta_2 = 10^{-60/20} = 0.001$), we get

$$L = 1 - \frac{10\log(0.0116 \times 0.001) - 15}{14\left(\frac{1}{10}\right)}$$

$$= 1 - \frac{-49.4 - 15}{1.4} = 47 \text{ taps}$$

Overflow is a potential problem when doing the calculations for an FIR filter. Multiplying two N-bit numbers results in a product with 2N bits, so space must be provided in the accumulator to accommodate that. Although the final result normally will be scaled and truncated back to N bits, it is best to carry through all the intermediate results with full resolution in order not to lose any dynamic range. In addition, the sum of all the taps can be a number with more than 2N bits. For example, if the filter width is 256 taps, then if all coefficients and data are at full scale, the final result could theoretically be 256 times larger, requiring an extra 8 bits in the accumulator. We say "theoretically" because normally most of the filter coefficients are much less than full scale and it is highly unlikely that all 256 data values would ever simultaneously be full-scale values of the correct polarity to cause overflow. The dsPIC processors use 16-bit multipliers with 32-bit results and a 40-bit accumulator, which should handle any reasonable circumstances.

After all taps have been calculated, the final

result must be retrieved from the accumulator. Since the accumulator has much more resolution than the processor's data words, normally the result is truncated and scaled to fit. It is up to the circuit designer or programmer to scale by the correct value to avoid overflow. The worst case is when each data value in the shift register is full-scale — positive when it is multiplying a positive coefficient and negative for negative coefficients. That way, all taps add to the maximum value. To calculate the worst-case accumulator amplitude, simply add the absolute values of all the coefficients. However, that normally gives an unrealistically pessimistic value because statistically it is extremely unlikely that such a high peak will ever be reached. For a low-pass filter, a better estimate is to calculate the gain for a dc signal and add a few percent safety margin. The dc gain is just the sum of all the coefficients (not the absolute values). For a band-pass filter, add the sum of all the coefficients multiplied by a sine wave at the center frequency.

CALCULATING FIR FILTER COEFFICIENTS

So far we have ignored the question of how to determine the filter coefficients. For an ideal "brick-wall" low-pass filter, the answer turns out to be pretty simple. A "brick-wall" low-pass filter is one that has a constant response from zero hertz up to the cutoff frequency and zero response above. Its impulse response is proportional to the sinc function:

$$C(n) = C_o \ \mathrm{sinc}(2Bn) = \frac{\sin(2\pi Bn)}{2\pi Bn}$$

where C(n) are the filter coefficients, n is the sample number with n = 0 at the center of the impulse response, C_o is a constant, and B is the single-sided bandwidth normalized to the sample rate, B = bandwidth / sample rate.

It is interesting that this has the same form as the frequency response of a pulse, as was shown in Fig 15.7. That is because a brick-wall response in the frequency domain has the same shape as a pulse in the time domain. A pulse in one domain transforms to a sinc function in the other. This is an example of the general principle that the transformation between time and frequency domains is symmetrical. We will discuss this more later, in the section on Fourier transforms.

Normally, the filter coefficients are set up with the peak of the sinc function, sinc(0), at the center of the coefficient table so that there is an equal amount of "tail" on both sides. That points up the principle problem with this method of determining filter coefficients. Theoretically, the sinc function extends from minus infinity to plus infinity. Abruptly terminating the tails causes the frequency response to differ from an ideal brick-wall filter. There is ripple in the passband and non-zero response in the stopband, as shown in the graph in the upper right of **Fig 15.12**. This is mainly caused by the abruptness of the truncation. In effect, all the coefficients outside the limits of the coefficient table have been set to zero. The passband and stopband response can be improved by tapering the edges of the impulse response instead of abruptly transitioning to zero.

The process of tapering the edges of the impulse response is called *windowing*. The impulse response is multiplied by a window, a series of coefficients that smoothly taper to zero at the edges. For example, a rectangular window is equivalent to no window at all. Many different window shapes have been developed over the years — at one time it seemed that every doctoral candidate in the field of signal processing did their dissertation on some new window. Each window has its advantages and disadvantages. A window that transitions slowly and smoothly to zero has excellent passband and stopband response but a wide transition band. A window that has a wider center portion and then transitions more abruptly to zero at the edges has a narrower transition band but poorer passband and stopband response. The equations for the windows in Fig 15.12 are included in a sidebar.

The routine shown in **Table 15.3** is written for a dsPIC processor so it can be used to calculate filter coefficients "on the fly" as the operator adjusts a bandwidth control. The same code should also work using a generic C compiler on a PC so the coefficients could be downloaded into an FIR filter implemented in hardware.

The windowed-sinc method works pretty well for a simple low-pass filter, but what if some more-complicated spectral shape is desired? The same general design approach still applies. You determine the desired spectral shape, transform it to the time domain using an inverse Fourier transform, and apply a window. There is lots of (often free) software available that can calculate the fast Fourier transform (FFT) and inverse fast Fourier transform (IFFT). So the technique is to generate the desired spectral shape, transform to the time domain with the IFFT, and multiply the resulting impulse response by the desired window. Then you can transform back to the frequency domain with the FFT to see how the window affected the result. If the result is not satisfactory, you can either choose a different window or modify the original spectral shape and go through the process again.

Table 15.3
Routine for dsPIC Processor to Calculate Filter Coefficients

```
// Calculate FIR filter coefficients
// using the windowed-sinc method
void set_coef (
 double sample_rate;
 double bandwidth;)
{
extern int c[FIR_LEN]; // Coefficient array
int i; // Coefficient index
double ph; // Phase in radians
double coef; // Filter coefficient
int coef_int; // Digitized coefficient
double bw_ratio; // Normalized bandwidth

bw_ratio = 2 * bandwidth / sample_rate;
for (i = 0; i < (FIR_LEN/2); i++) {
 // Brick-wall filter:
 ph = PI * (i + 0.5) * bw_ratio;
 coef = sin(ph) / ph;
 // Hann window:
 ph = PI * (i + 0.5) / (FIR_LEN/2);
 coef *= (1 + cos(ph)) / 2;
 // Convert from floating point to int:
 coef *= 1 << (COEF_WIDTH - 1);
 coef_int = (int)coef;
 // Symmetrical impulse response:
 c[i + FIR_LEN/2] = coef_int;
 c[FIR_LEN/2 - 1 - i] = coef_int;
 }
}
```

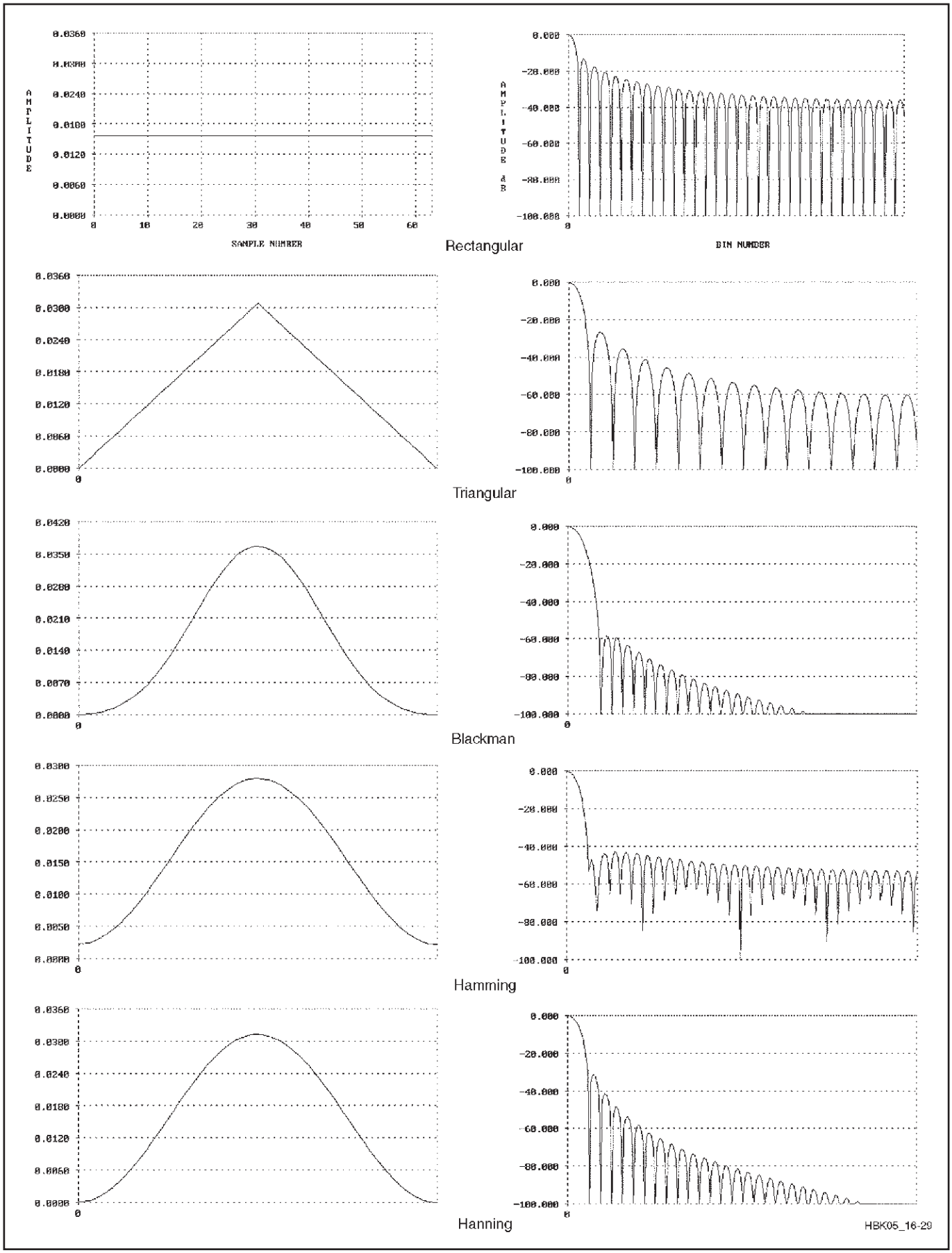

Fig 15.12 — Various window functions and their Fourier transforms.

Equations for Window Functions

For each window function, the center of the response is considered to be at time t = 0 and the width of the impulse is L. Each window is 1.0 when t = 0 and 0.0 when the | t | > L/2.

Rectangular:

$$w(t) = 1.0$$

Triangular (Bartlett):

$$w(t) = 2\left(\frac{L/2 - |t|}{L}\right)$$

Blackman:

$$w(t) = 0.42 + 0.5\cos\left(\frac{2\pi t}{L}\right)$$

$$+0.08\cos\left(\frac{4\pi t}{L}\right)$$

Hamming:

$$w(t) = 0.54 + 0.46\cos\left(\frac{2\pi t}{L}\right)$$

Hanning (Hann):

$$w(t) = 0.5 + 0.5\cos\left(\frac{2\pi t}{L}\right)$$

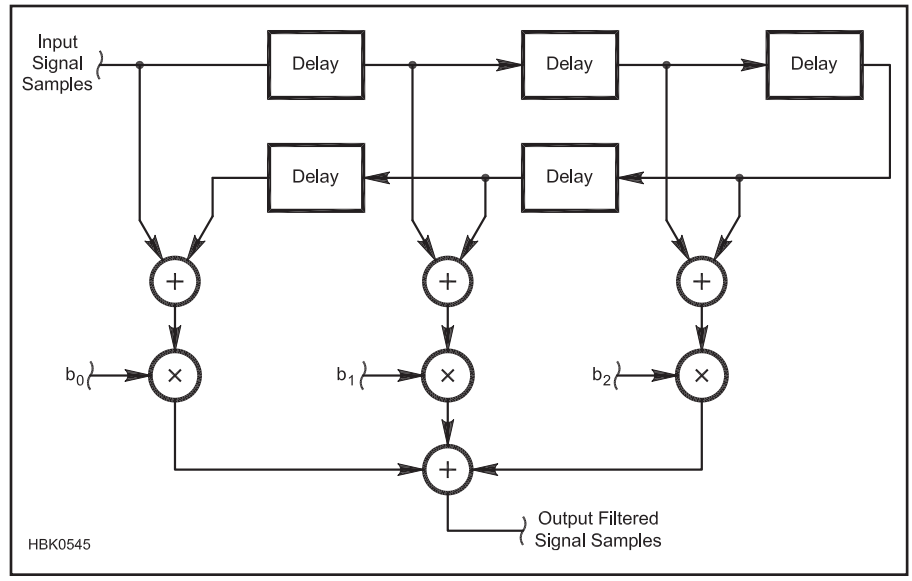

Fig 15.13 — A 6-tap FIR filter. Because the coefficients are symmetrical, the symmetrical taps may be combined before multiplication.

Windowing methods are useful because they are simple to program and the resulting software routines execute quickly. For example, you can include a bandwidth knob on your DSP filter and calculate filter coefficients "on the fly" as the user turns the knob. However, while the filter performance that results is pretty good, it is not "optimum" in the sense that it does not have minimum passband ripple and maximum stopband attenuation for a given number of filter coefficients. For that, you need what is known as an *equal-ripple*, or *Chebyshev* filter. The calculations to determine Chebyshev filter coefficients are more complicated and time-consuming. For that reason, the coefficients are normally calculated in advance on a PC and stored in DSP program memory for retrieval as needed.

Engineers have not had much success in devising a mathematical algorithm to calculate the Chebyshev coefficients directly, but in 1972 Thomas Parks and James McClellan figured out a method to do it iteratively. The *Parks-McClellan algorithm* is supported in most modern filter-design software, including a number of programs available for free download on the Web. Typically you enter the sample rate, the passband and stopband frequency ranges, the passband ripple and the stopband attenuation. The software then determines the required number of filter coefficients, calculates them and displays a plot of the resulting filter frequency response.

Filter design software typically presents the filter coefficients as floating-point numbers to the full accuracy of the computer. You will need to scale the values and truncate the resolution to the word size of your filter implementation. Truncation of filter coefficients affects the frequency response of the filter but does not add noise in the same manner as truncating the signal data.

As you look at impulse responses for various FIR filters calculated by various methods you soon realize that most of them are symmetrical. If the center of the impulse response is considered to be at time zero, then the value at time t equals the value at time –t for all t. If you know in advance that the filter coefficients are symmetrical, you can take advantage of that in the filter design. By re-arranging the adders and multipliers, the number of multipliers can be reduced by a factor of two, as shown in **Fig 15.13**. This trick is less useful in a software implementation of an FIR filter because the number of additions is the same and many DSPs take the same amount of time to do an addition as a multiply-accumulate.

In addition to the computational benefit, a symmetrical impulse response also has the advantage that it is *linear phase*. The time delay through such a filter is one-half the length of the filter for all frequencies. For example, for a 1000-tap filter running at 10 kHz the delay is 500/10,000 = 0.05 second. Since the time delay is constant for all frequencies, the phase delay is directly proportional to the frequency. For example, if the phase delay at 20 Hz is one cycle (0.05 second) it is ten cycles at 200 Hz (still 0.05 second). Linear phase delay is important with digital modulation signals to avoid distortion and inter-symbol interference. It is also desirable with analog modulation where it can result in more natural-sounding audio. All analog filters are non-linear-phase; the phase distortion tends to be worse the more abrupt the transition between passband and stopband. That is why an SSB signal sounds unnatural after being filtered by a crystal filter with a small shape factor even though the passband ripple may be small and distortion minimal.

A band-pass filter can be constructed from a low-pass filter simply by multiplying the impulse response by a sine wave at the desired center frequency. This can be done before or after windowing. The linear-phase property is retained but with reference to the center frequency of the filter, that is, the phase shift is proportional to the difference in frequency from the center frequency. The frequency response is a double-sided version of the low-pass response with the zero-hertz point of the low-pass filter shifted to the frequency of the sine wave.

15.4.2 IIR Filters

An *infinite impulse response* (IIR) filter is a filter whose impulse response is infinite. After an impulse is applied to the input, theoretically the output never goes to zero and stays there. In practice, of course, the signal eventually does decay until it is below the noise level (analog filter) or less than one LSB (digital filter).

Unlike a symmetrical FIR filter, an IIR filter is not generally linear-phase. The delay through the filter is not the same for all frequencies. Also, IIR filters tend to be harder to design than FIR filters. On the other hand, many fewer adders and multipliers are typically required to achieve the same passband and stop band ripple in a given filter, so IIR filters are often used where computations must be minimized.

All analog filters have an infinite impulse response. For a digital filter to be IIR it must have feedback. That means a delayed copy of some internal computation is applied to an earlier stage in the computation. A simple but useful example of an IIR filter is the exponential decay circuit in **Fig 15.14**. In the absence of a signal at the input, the output on the next clock cycle is always (1–δ) times the current output. The time constant (the time for the output to die to 1/e = 36.8% of the initial value) is very nearly

$$\tau = f_s\left(\frac{1}{\delta} - \frac{1}{2}\right)$$

where f_s is the sample rate. The circuit is the digital equivalent of a capacitor with a resistor in parallel and might be useful for example in a digital automatic gain control circuit.

One issue with IIR filters is resolution. Because of the feedback, the number of bits of resolution required for intermediate computations can be much greater than at the input or output. In the previous example, δ is very small for very long time constants. When the value in the register falls below a certain level the multiplication by (1–δ) will no longer be accurate unless the bit width is increased. In practice, the increased resolution required with IIR filters often cancels out part of the savings in the number of circuit elements.

Another issue with IIR filters is stability. Because of the feedback it is possible for the filter to oscillate if care is not taken in the design. Stability can also be affected by non-linearity at low signal levels. A circuit that is stable with large signals may oscillate with small signals due to the round-off error in certain calculations, which causes faint tones to appear when strong signals are not present. This is known as an unstable *limit cycle*. These issues are part of the reason that IIR filters have a reputation for being hard to design.

Design techniques for IIR filters mostly involve first designing an analog filter using any of the standard techniques and then transforming the design from the analog to the digital domain. The *impulse-invariant* method attempts to duplicate the filter response directly by making the digital impulse response equal the impulse response of the equivalent analog filter. It works fairly well for low-pass filters with bandwidths much less than the sample rate. Its problem is that it tries to duplicate the frequency response all the way to infinity hertz, but that violates the Nyquist criterion resulting in a folding back of the high-frequency response down into low frequencies. It is similar to the aliasing that occurs in a DSP system when the input signal to be sampled is not band-limited below the Nyquist frequency.

The *bilinear transform* method gets around that problem by distorting the frequency axis such that infinity hertz in the analog domain becomes sample rate / 2 in the digital domain. Low frequencies are fairly accurate, but high frequencies are squeezed together more and more the closer you get to the Nyquist frequency. It avoids the aliasing problem at the expense of a change in the spectrum shape,

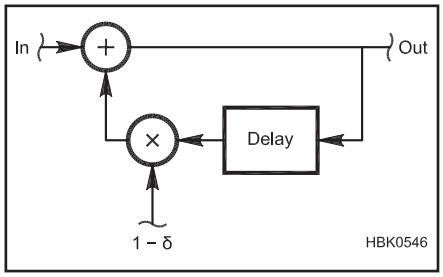

Fig 15.14 — An exponential decay circuit.

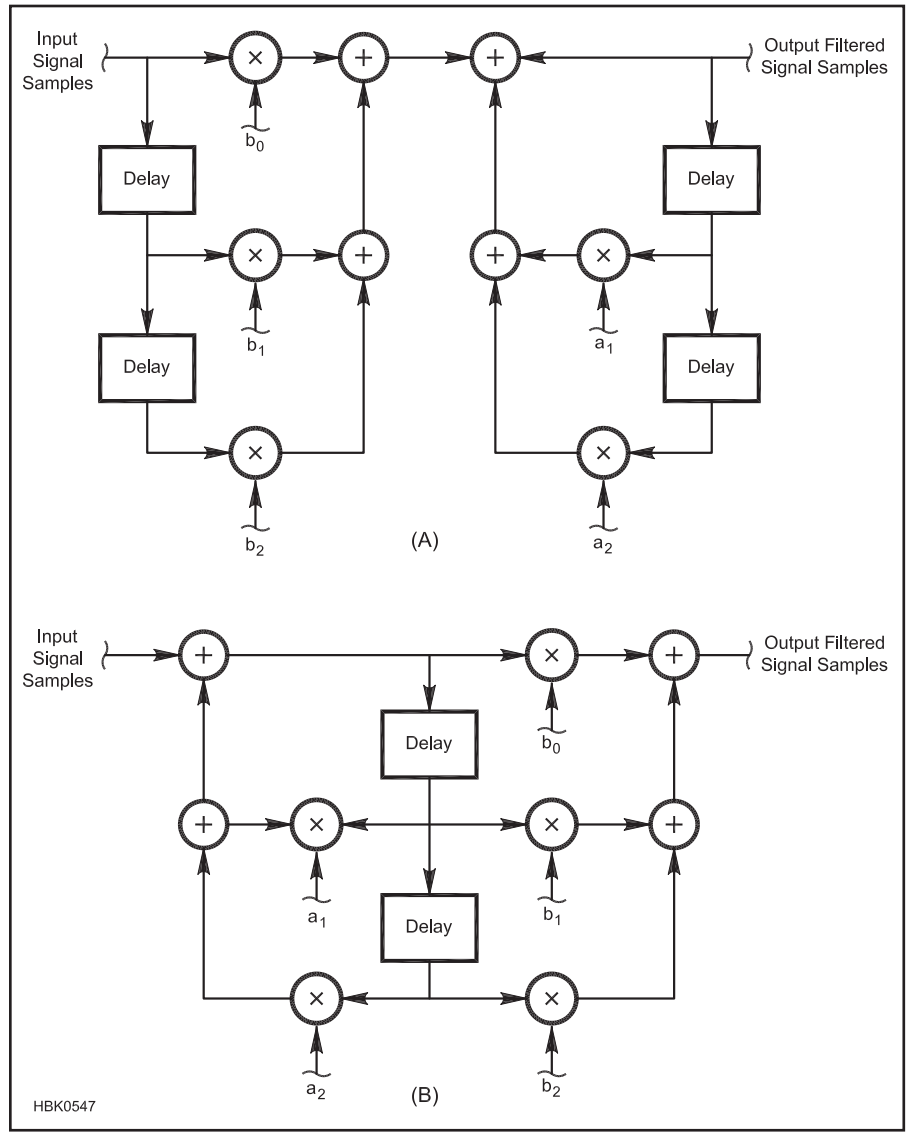

Fig 15.15 — An IIR filter with three feed-forward taps and two feed-back taps. Direct form I (A) and the equivalent direct form II (B).

especially at the high-frequency end. For example, when designing a low-pass filter it may be necessary to change the cutoff frequency to compensate. Again, the method works best for filters with passband frequencies much less than the sample rate.

In general, the output of an IIR filter is a combination of the current and previous input values (feed-forward) and previous output values (feed-back). **Fig 15.15A** shows the so-called *direct form I* of an IIR filter. The b_i coefficients represent feed-forward and the a_i coefficients feed-back. For example the previous value of the y output is multiplied by a_1, the second previous value is multiplied by a_2, and so on. Because the filter is linear, it doesn't matter whether the feed forward or feed back stage is performed first. By reversing the order, the number of shift registers is reduced as in Fig 15.15B. There are other equivalent topologies as well. The mathematics for generating the a_i and b_i coefficients for both the impulse-invariant and bilinear transform methods is fairly involved, but fortunately some filter design programs can handle IIR as well as FIR filters.

15.4.3 Adaptive Filters

An *adaptive filter* is one that automatically adjusts its filter coefficients under the control of some algorithm. This is often done in situations where the filter characteristics are not known in advance. For example, an adaptive channel equalizer corrects for the non-flatness in the amplitude and phase spectrum of a communications channel due to multipath propagation. Typically, the transmitting station periodically sends a known sequence of data, known as a *training sequence*, which is used by the receiver to determine the channel characteristics and adjust its filter coefficients accordingly.

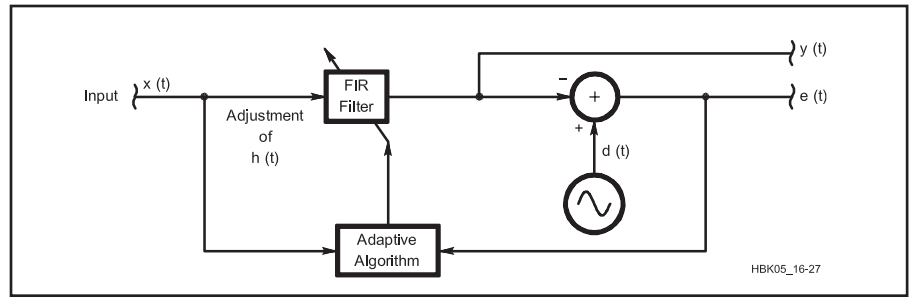

Fig 15.16 — An adaptive filter.

Another example is an automatic notch filter. An algorithm determines the frequency of an interfering tone and automatically adjusts the notch frequency to remove the tone. Noise cancellation is another application. It can be thought of as the opposite of a notch filter. In this case, all the sine-wave tones in the input signal are considered to be desired and the filter coefficients are configured to enhance them. That method works not only for CW signals but for voice as well since the human voice consists largely of discrete frequencies.

A generic block diagram of an adaptive filter is shown in **Fig 15.16**. The variable filter is typically an FIR type with coefficients calculated by the update algorithm. By some means, an estimate of the desired, unimpaired signal, d, is generated and compared to the filter output y. The difference between y and d is the error, e, which is used by the update algorithm to modify the filter coefficients to improve the accuracy of y. The algorithm is capable of acting as a noise-reduction filter and a notch filter simultaneously. Assuming d is in the form of a pure tone (sine wave), the tone is simultaneously optimized in the y output and minimized in the e output.

A common algorithm for minimizing the error signal is called *least mean squares* (LMS). The LMS algorithm includes a performance parameter, μ, which can be adjusted between 0 and 1 to control the tradeoff between adjustment speed and accuracy. A value near 1 results in fast convergence but the convergence is not very accurate. For better accuracy at the cost of slower adjustment, lower the value of μ. Some implementations adjust μ on the fly, using a large value at first to get faster lock-in when the error is large then a smaller value after convergence to reduce the error. That works as long as the signal characteristics are not changing too rapidly.

15.5 Miscellaneous DSP Algorithms

15.5.1 Sine Wave Generation

There are a number of techniques available for building a digital sine-wave generator, either in hardware or in software. One obvious idea is to make a digital oscillator, analogous to a conventional analog oscillator. Simply design a band-pass filter at the desired frequency and include positive feedback around it with a loop gain of unity. The problem is that, because of round-off error in the digital calculations, it is difficult to get the loop gain to be *exactly* 1.0. If it is slightly less, then the oscillation will eventually die out. If it is slightly greater, then the oscillation will gradually increase in amplitude until it exceeds the maximum signal that the digital circuitry can handle. There are two techniques to handle this problem. One idea is to include an automatic gain-control circuit to detect the amplitude, low-pass filter it and feed the result to a multiplier in the feedback path to control the gain. Another idea is to intentionally set the loop gain slightly greater than 1.0 and include a clipping stage in the feedback path that limits the peak amplitude in a controlled manner.

With both of those techniques there is a tradeoff between distortion and start-up time. It can take many oscillation cycles for the amplitude to stabilize. If the loop gain is increased or the AGC time constant is reduced to improve the start-up time, worse distortion results.

Probably the most common technique for generating sine waves is the *numerically-controlled oscillator* (NCO), or *direct digital synthesizer* (DDS) as shown in **Fig 15.17**. At each clock cycle, the phase accumulator increments the phase by an amount equal to 360° times f / f_s, where f is the sine-wave frequency and f_s is the sample rate. The current phase value is used as a pointer to the proper address in a sine-wave lookup table. As the phase increases, the pointer moves through the look-up table, tracing out the sine-wave amplitude. Different frequencies can be obtained by changing the step size in the phase accumulator.

The lookup table size is a factor of two and the accumulator output is scaled such that the maximum count, corresponding to 360°, accesses the final entry in the table. When the phase passes 360° it automatically jumps back to zero as required. For good frequency resolution, the word size of the phase accu-

Fourier Transform

The *Fourier transform* is the software equivalent of a hardware spectrum analyzer. It takes in a signal in the time domain and outputs a signal in the frequency domain that shows the spectral content of the input signal. The Fourier transform works on both periodic and non-periodic signals, but since the periodic case is easier to explain we will start with that.

A periodic signal is one that repeats every τ seconds, where τ is the period. That means that the signal can consist only of frequencies whose sinusoidal waveforms have an integer number of cycles in τ seconds. In other words, the signal is made up of sinusoids that are at the frequency $1/\tau$ and its harmonics. Fourier's idea was that you can determine if a frequency is present by multiplying the waveform by a sinusoid of that frequency and integrating the result. The result of the integration yields the amplitude of that harmonic. If the integration yields zero, then that frequency is not present.

To see how that works, look at **Fig 15-A1**. For the purpose of discussion, assume the signal to be tested consists of a single tone at the second harmonic as shown at (A). The first test frequency is the fundamental, shown at (B). When you multiply the two together you get the waveform at (C). Integrating that signal gives its average value, which is zero. However if you multiply the test signal by a sine wave at the second harmonic (D), the resulting waveform (E) has a large dc offset so the integration yields a large non-zero value. It turns out that all harmonics other than the second yield a zero result. That is, the second harmonic is *orthogonal* to all the others.

If the test waveform included more than one frequency, each of those frequencies would yield a non-zero result when tested with the equivalent-frequency sine wave. The presence of additional frequencies does not disturb the tests for other frequencies since they are all orthogonal with each other.

You may have noticed that this method only works if the test sine wave is in phase with the one in the signal. If they are 90° out of phase, the integration yields zero. The Fourier transform therefore multiplies the signal by both a sine wave and a cosine wave at each frequency. The results of the two tests then yield both the amplitude and phase of that frequency component of the signal using the equations

$$A = \sqrt{a^2 + b^2}$$

and

$$\varphi = \arctan\left(\frac{b}{a}\right)$$

where A is the amplitude, φ is the phase, a is the cosine amplitude and b is the sine amplitude.

If one period of the signal contains, say, 256 samples, then testing a single frequency requires multiplying the signal by the sine wave and by the cosine wave 256 times and adding the results 256 times as well, for a total of 512 multiplications and additions. There are 128 frequencies that must be tested, since the 128th harmonic is at the Nyquist frequency. The total number of calculations is therefore $512 \times 128 = 256^2$ multiplications and additions. That is a general result. For any sample size, n, calculating the digital Fourier transform requires n^2 multiply-accumulates.

The FFT

The number of calculations grows rapidly with sample size. Calculating the Fourier transform on 1024 samples requires over a million multiply-accumulates. However, you may notice that there is some redundancy in the calculations. When testing the second harmonic, for example, each of the two cycles of the test sine wave is identical. It would be possible to pre-add signal data from the first and second halves of the sequence and then just multiply once by a single cycle of the test sine wave. Also, the first quarter cycle of a sine wave is just a mirror-image of the second quarter cycle and the first half is just the negative of the second half.

In 1965, J. W. Cooley and John W. Tukey published an algorithm that takes advantage of all the symmetries inherent in the Fourier transform to speed up the calculations. The *Cooley-Tukey algorithm*, usually just called the *fast Fourier transform* (FFT), makes the number of calculations proportional to $n\log_2(n)$ instead of n^2. For a 1024-point FFT, the calculation time is proportional to $1024\log_2(1024) = 10{,}240$ instead of $1024^2 = 1{,}048{,}576$, more than a 100-times improvement.

You'll notice that sample sizes are usually a power of two, such as $2^7 = 128$, $2^8 = 256$ and $2^9 = 512$. That is because the FFT algorithm is most efficient with sequences of such sizes. The algorithm uses a process called *radix-2 decimation in time*, that is, it first breaks the data into two chunks of equal size, then breaks each of those chunks into two still-smaller chunks of equal size, and so on. It is possible to squeeze even a little more efficiency out of the algorithm with a *radix-4* FFT which is based on decimation by four instead of by two. That is why you often see sample sizes that are powers of four, such as $4^3 = 64$, $4^4 = 256$ and $4^5 = 1024$. Other variations on the algorithm include decimation in frequency rather than time, mixed-radix FFTs that use different decimation factors at different stages in the calculation, and in-place calculation that puts the results into the same storage buffer as the input data, saving memory. The latter method causes the order of the output data to be scrambled by bit-reversing the address words. For example, address 01010000 becomes 00001010.

Non-periodic signals

So far we have assumed that the signal to be transformed is periodic, so that there is an integer number of cycles of each sine wave harmonic in the sequence. With a non-periodic signal, that is not necessarily so. The various frequencies in the signal are not exact harmonics of $1/\tau$ and are no longer

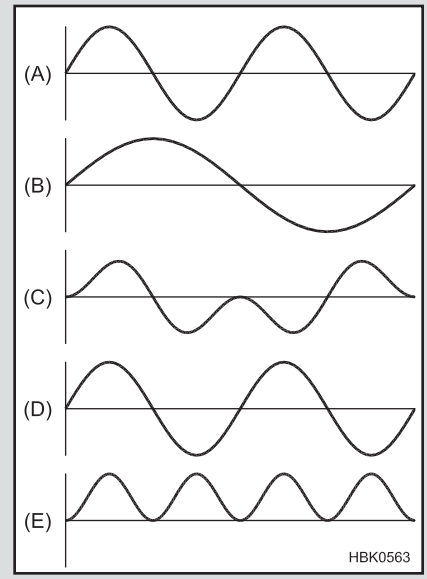

Fig 15-A1 — Signal to be tested for frequency content (A). Fundamental test frequency (B). Product of signal and fundamental (C). Second harmonic test frequency (D). Product of signal and second harmonic (E).

orthogonal to the test frequencies. The result is *spectral leakage*; a single frequency in the signal may give a non-zero result when tested at a number of different harmonic frequencies. In **Fig 15-A2**, (B) illustrates the FFT of a single sine wave at a non-harmonic frequency. You can see that the spurious response extends quite far from the actual frequency.

Those far-out spurious responses are primarily caused by the abrupt termination of the signal at the edges of the sequence. The spectrum can be cleaned up considerably by tapering the edges with a window, in a manner similar to windowing FIR filter coefficients as previously described. In fact, the same windows work for both. Fig 15-A2 part C illustrates the result of applying a Hamming window to the signal in (A) and the resulting improved spectrum is shown at (D). Just as with FIR filters, different windows excel in different areas. Windows with a gradual transition to zero at the edges do a better job of suppressing spurious responses but smear adjacent spectral lines, analogous to using a wider resolution bandwidth in an analog spectrum analyzer. Windows with a fatter center section and a more abrupt transition to zero at the edges have less smearing but worse spurious responses.

While it is interesting and instructional to write your own FFT from scratch, most programmers don't bother to try to re-invent the wheel. Many implementations have been published on the Web and in books and articles. Most of the software development systems offered by DSP vendors include an FFT library routine, which runs faster than anything you are likely to come up with on your own.

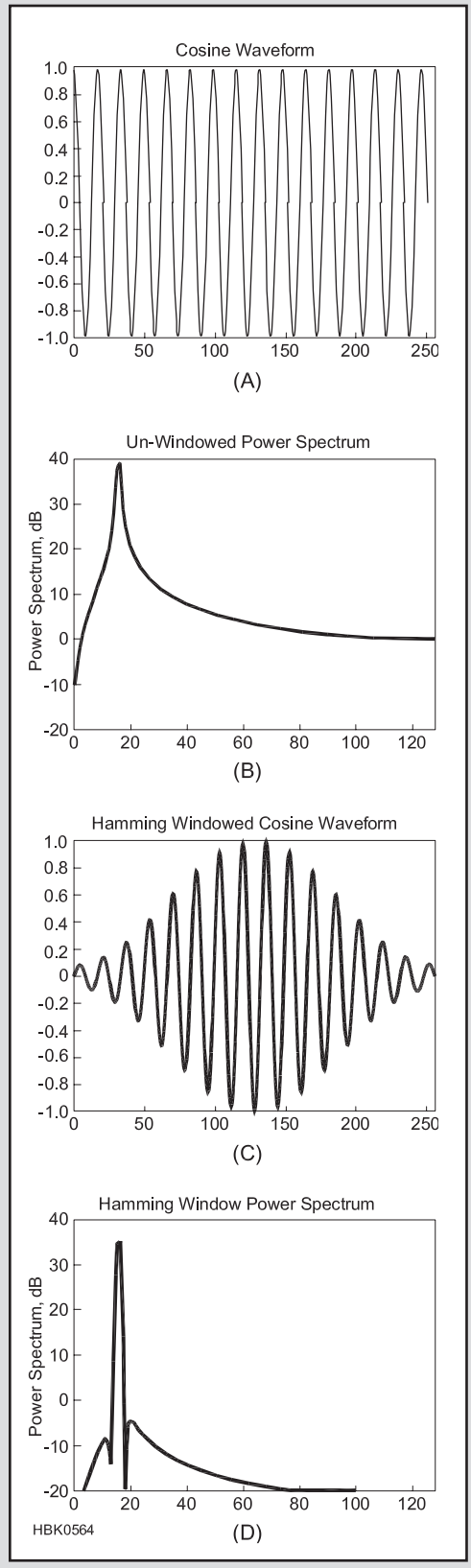

Fig 15-A2 — Illustrating the use of windowing to minimize spectral leakage, the figures show (A) a cosine waveform not at a harmonic frequency, (B) the resulting unwindowed power spectrum, (C) the same cosine waveform with a Hamming window, and (D) the much narrower power spectrum of the windowed waveform.

mulator, 32 in this example, is normally much greater than the address width of the look-up table, p. The less-significant accumulator bits are not used in the address. For example, with a 100 MHz clock rate and a 32-bit phase accumulator the frequency resolution is $100 \times 10^6 / 2^{32} = 0.023$ Hz.

The address width of the look-up table determines the number of table entries and thus the phase accuracy of the samples. The table size can be reduced by a factor of four by including only the first quarter-cycle of the sine wave in the table. The other three quadrants can be covered by modifying the look-up address appropriately and by negating the output when required under the control of some additional logic. For example, the method to transform quadrant three to quadrant one is illustrated in **Fig 15.18**. Another technique to improve waveform accuracy without increasing table size is to interpolate between the entries. Instead of using the look-up table output directly, the output value is calculated by interpolating between the two nearest table entries using either a straight-line interpolation as shown in **Fig 15.19** or a higher-order curve fit.

Taking that last idea to an extreme, it is possible to generate a good approximation to one quadrant of a sine wave using a fifth-order interpolation between the zero and 90° points. Assuming that the phase x has been scaled so that x = 1.0 corresponds to 90°, the sine formula is

$$\sin(x) = Ax + Bx^2 + Cx^3 + Dx^4 + Ex^5$$

where the coefficients are A = 3.140625, B = 0.02026367, C = -5.325196, D = 0.5446778 and E = 1.800293.

With those coefficients, which come from an old Analog Devices DSP manual, all the harmonics of the sine wave are more than 100 dB below the carrier.[3] With a 16-bit integer DSP the performance would be limited only by the roundoff error. The formula can be reformulated as

$$\sin(x) = (A + (B + (C + (D + Ex)x)x)x)x$$

which reduces the number of multiplications required from 15 to 5, as shown in **Fig 15.20**.

15.5.2 Tone Decoder

Tone decoders have a number of applications in Amateur Radio. A Morse code reader might use a tone decoder to determine the on and off states of the incoming CW signal. A sub-audible tone detector in a VHF FM receiver is another application. A DTMF decoder needs to detect two tones simultane-

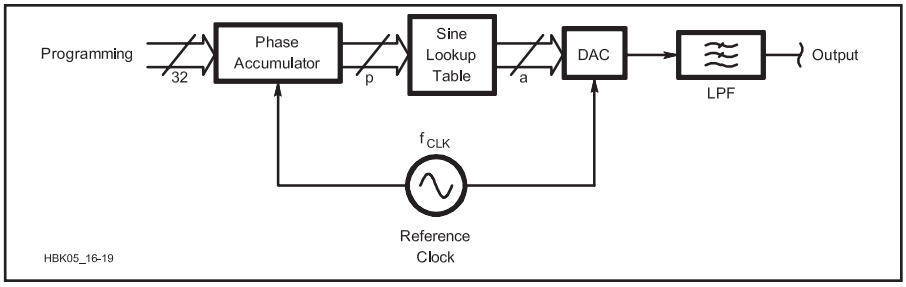

Fig 15.17 — DDS block diagram.

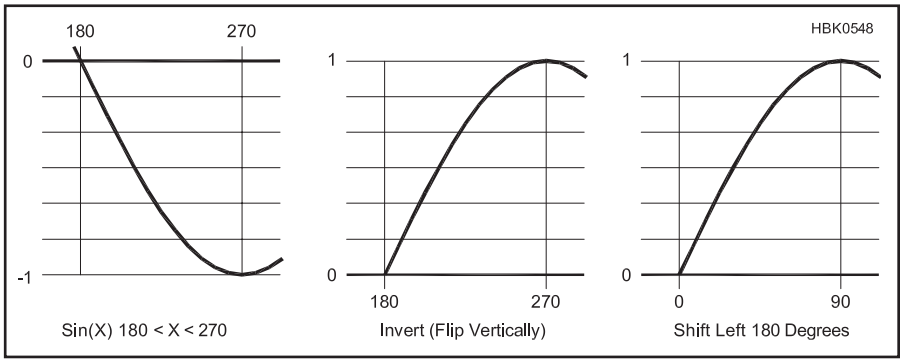

Fig 15.18 — The values of sin(x) between 180 and 270° are the same as those between 0 and 90°, after the curve has been flipped vertically and shifted 180°.

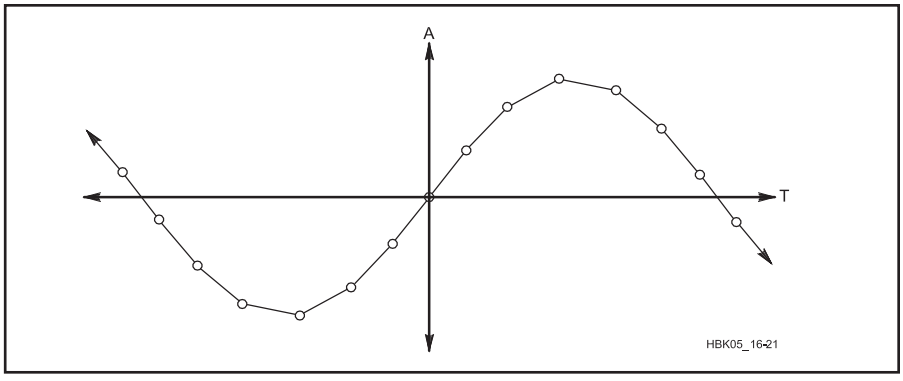

Fig 15.19 — An approximation of a sine wave using a straight-line interpolation between lookup table entries.

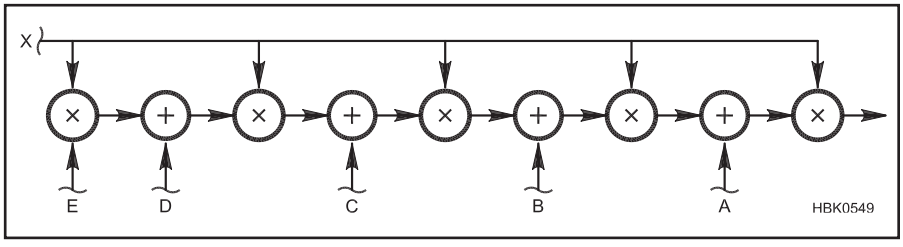

Fig 15.20 — Method to calculate a fifth-order interpolation between lookup table entries.

ously to determine which of the Touch-Tone keys has been pressed.

An FFT is one type of tone decoder. A 1024-point FFT simultaneously decodes 512 frequencies up to one-half the sample rate using a very efficient algorithm. You can control the detection bandwidth by choosing the number of points in the FFT; the spacing of the frequency samples is just the sample rate divided by the number of FFT points. However, in many applications you don't need that much resolution. For example, a DTMF signal consists of two tones, each of which can be one of four frequencies, for a total of 8 possible frequencies. Instead of performing a complete FFT, you can simply convolve sinusoidal waves of each of those 8 frequencies with the sequence of incoming samples to detect energy at those 8 frequencies. This will take less computation than a full FFT whenever the number of test frequencies is less than the base-2 logarithm of the number of points in the sequence. Since $\log_2(1024)$ is 10, decoding 8 frequencies separately would be more efficient than a 1024-point FFT. Spectral leakage is just as much of a problem with this method as with an FFT, so you still need to window the sequence of samples before performing the convolution.

If only a single frequency needs to be detected it might make more sense to mimic analog techniques and use a band-pass digital filter followed by an amplitude detector. That would have the advantage that the passband and stopband characteristics of the filter could be much more precisely controlled than with an FFT. In addition, the output is updated continuously instead of in "batch mode" after each batch of samples is collected and processed.

15.6 Analytic Signals and Modulation

In the area of modulation, the topic that seems to give people the most trouble is the concept of negative frequency. What in the world is meant by that? Consider a single-frequency signal oscillating at ω radians per second. (Recall that $\omega = 2\pi f$, where f is frequency in Hz.) Let's represent the signal by a cosine wave with a peak amplitude of 1.0, $x(t) = \cos(\omega t)$, where t is time. Changing the sign of the frequency is equivalent to running time backwards because $(-\omega)t = \omega(-t)$. By examining **Fig 15.21A** you can see that, because a cosine wave is symmetrical about the time t = 0 point, a negative frequency results in exactly the same signal. That is, as you may remember from high-school trigonometry, $\cos(-\omega t) = \cos(\omega t)$. If, for example, you add a positive-frequency cosine wave to its negative-frequency twin, you get the same signal with twice the amplitude.

That assumes that the phase of the signal is such that it reaches a peak at t = 0. What if instead we had a sine wave, which is zero at t = 0? From Fig 15.21B you can see that running time backwards results in a reversal of polarity, $\sin(-\omega t) = -\sin(\omega t)$. If you add positive and negative-frequency sine waves of the same frequency and amplitude, they cancel, resulting in zero net signal.

A sinusoidal wave of any arbitrary amplitude and phase may be represented by the weighted sum of a sine and cosine wave:

$$x(t) = I\cos(\omega t) + Q\sin(\omega t)$$

For computational purposes, it is convenient to consider the *in-phase* (*I*) and *quadrature* (*Q*) components separately. Since the I and Q components are 90° out of phase in the time domain, they are often plotted on a polar graph at a 90° angle from each other. See **Fig 15.22**. For example if Q = 0, then as time increases the signal oscillates along the I (horizontal) axis, tracing out the path back and forth between I = +1 and I = –1 in a sinusoidal fashion. Conversely, if I = 0, then the signal oscillates along the Q axis.

What if both I and Q are non-zero, for example I = Q = 1? Recall that the cosine and sine are 90° out of phase. When t = 0, $\cos(\omega t) = 1$ and $\sin(\omega t) = 0$. A quarter cycle later, $\cos(\omega t) = 0$ and $\sin(\omega t) = 1$. Comparing Fig 15.22 with Fig 15.21 it should not be hard to convince yourself that the signal is tracing out a circle in the counter-clockwise direction.

What about negative frequency? Again, it should not be hard to convince yourself that changing ω to –ω results in a signal that circles the origin in the clockwise direction. If you combine equal-amplitude signals of opposite frequency, the sine portions cancel out and you are left with a simple cosine wave of twice the amplitude:

$$\begin{aligned} x(t) &= \left[\cos(\omega t) + \sin(\omega t)\right] \\ &\quad + \left[\cos(-\omega t) + \sin(-\omega t)\right] \\ &= 2\cos(\omega t) \end{aligned}$$

You can see that graphically in **Fig 15.23**. Imagine the two vectors rotating in opposite directions. If you mentally add them by placing the tail of one vector on the head of the other, as shown by the dotted line, the result always lies on the I axis and oscillates between +2 and –2.

That is why we say that a single scalar sinusoidal signal, $\cos(\omega t)$, actually contains two frequencies, +ω and –ω. It also offers a logical explanation of why a mixer or modulator produces the sum and difference of the frequencies of the two inputs. For example, an AM modulator produces sidebands at the carrier frequency plus and minus the modulating frequency precisely because those positive and negative frequencies are actually already present in the modulating signal.

For many purposes, it is useful to separate the portion of the signal that specifies the amplitude and phase (I and Q) from the oscillating part ($\sin(\omega t)$ and $\cos(\omega t)$). For mathematical convenience, the I/Q part is represented by a complex number, $x = I + jQ$. The oscillating part is also a complex number $e^{-jwt} = \cos(\omega t) - j\sin(\omega t)$. (Don't worry if you don't know where that equation comes from — concentrate on the part to the right of the equals sign.[4]) In the equations, $j = \sqrt{-1}$. Of course, –1 does not have a real square root (any real number multiplied by itself is positive) so *j*, or any real number multiplied by *j*, is called an *imaginary number*. A number with both real and imaginary parts is called a *complex number*. The total *analytic signal* is a complex number equal to

$$x(t) = xe^{-j\omega t} = \left(I + jQ\right)\left(\cos(\omega t) - j\sin(\omega t)\right)$$

In the above equation, the $\cos(\omega t) - \sin(\omega t)$ portion generally represents an RF carrier, with ω being the carrier frequency (a positive or negative value). The $I + jQ$ part is the modulation. The *scalar* value of a modulated signal (what you would measure with an oscilloscope) is just the real part of the analytic signal. Using the fact that $j^2 = -1$,

$$\mathrm{Re}\left[x(t)\right] = \mathrm{Re}\left[\left(I + jQ\right)\left(\cos(\omega t) - j\sin(\omega t)\right)\right]$$

$$\mathrm{Re}\left[x(t)\right] = \mathrm{Re}\left[\begin{array}{l}\left(I\cos(\omega t) + Q\sin(\omega t)\right) \\ +j\left(Q\cos(\omega t) - I\sin(\omega t)\right)\end{array}\right]$$

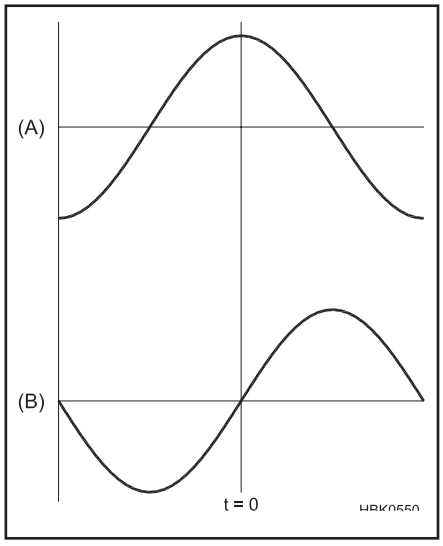

Fig 15.21 — Cosine wave (A) and sine wave (B).

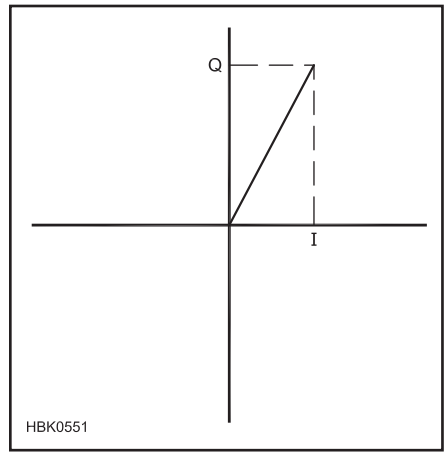

Fig 15.22 — In-phase (I) and quadrature (Q) portions of a signal.

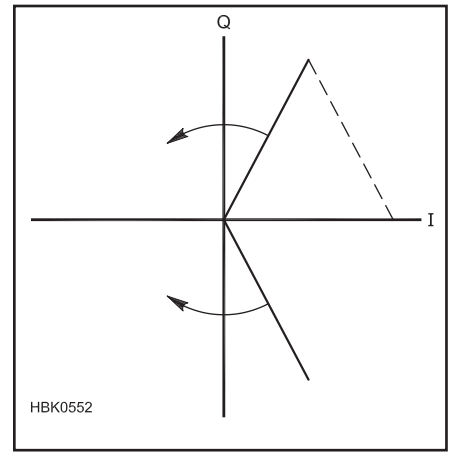

Fig 15.23 — A real frequency is the sum of a positive and negative analytic frequency.

$$\mathrm{Re}\left[x(t)\right] = I\cos(\omega t) + Q\sin(\omega t)$$

Note that if the modulation (I and Q) varies with time, the above equation assumes that the modulated signal does not overlap zero Hz. That is, I and Q have no frequency components greater than ω.

Normally the I/Q diagram shows only I and Q (the modulation) and not the oscillating part. We call such a representation a *phasor diagram*. The I/Q vector represents the *difference* in phase and amplitude of the RF signal compared to the unmodulated carrier. For example, if the I/Q vector is at 90°, that means the carrier has been phase-shifted by 90° from what it otherwise would have been. If the I/Q vector is rotating counter-clockwise 10 times per second, then the carrier frequency has been increased by 10 Hz.

It is worth noting that the modulation can be specified either by the in-phase and quadrature (I and Q) values as shown or alternatively by the amplitude and phase. The amplitude is the length of the I/Q vector in the phasor diagram,

$$A = \sqrt{I^2 + Q^2}$$

The phase is the angle of the vector with respect to the +I axis,

$$\varphi = \arctan\left[\frac{Q}{I}\right]$$

An alternative expression for the modulated analytic signal using amplitude and phase is

$$x(t) = Ae^{-j(\varphi+\omega t)}$$
$$= A\left[\cos(\varphi + \omega t) + j\sin(\varphi + \omega t)\right]$$

and for the scalar signal

$$\mathrm{Re}[x(t)] = A\cos(\varphi + \omega t)$$

One final comment. So far we have been looking at signals that consist of a single sinusoidal frequency. In any linear system, anything that is true for a single frequency is also true for a combination of many frequencies. Each frequency is affected by the system as though the others were not present. Since any complicated signal can be broken down into a (perhaps large) number of single-frequency sinusoids, all our previous conclusions apply to multi-frequency signals as well.

15.6.1 I/Q Modulation and Demodulation

An *I/Q modulator* is just a device that controls the amplitude and phase of an RF signal directly from the in-phase (I) and quadrature (Q) components. See **Fig 15.24A**. An *I/Q demodulator* is basically the same circuit in reverse. It puts out I and Q signals that represent the in-phase and quadrature components of the incoming RF signal. See Fig 15.24B. Assuming the demodulator's local oscillator is on the same frequency and is in phase with the carrier of the signal being received then the I/Q output of the receiver's demodulator is theoretically identical to the I/Q input at the transmitter end.

I/Q modulators and demodulators can be built with analog components. The LO could be a transistor oscillator and the 90° phase-shift network could be implemented with coils and capacitors. The circles with the multiplication symbol would be double-balanced mixers. Not shown in the diagram are trim adjustments to balance the amplitude between the I and Q channels and to adjust the phase shift as close as possible to 90°.

No analog circuit is perfect, however. If the 90° phase-shift network is not exactly 90° or the amplitudes of the I and Q channels are not perfectly balanced, you don't get perfect opposite-sideband rejection. The modulator output includes a little bit of signal on the unwanted sideband and the I/Q signal from the demodulator includes a small signal rotating in the wrong direction. If there is a small dc offset in the amplifiers feeding the modulator's I/Q inputs, that shows up as carrier feedthrough. On receive, a dc offset makes the demodulator think there is a small signal at a constant amplitude and phase angle that is always there even when no actual signal is being received. Nor is analog circuitry distortion-free, especially the mixers. Intermodulation distortion shows up as out-of-channel "splatter" on transmit and unwanted out-of channel responses on receive.

All those problems can be avoided by going digital. If the analog I/Q inputs to the modulator are converted to streams of digital numbers with a pair of ADCs, then the mixers, oscillator, phase-shift network and summer can all be digital. In many systems, the I and Q signals are also generated digitally, so that the digital output signal has perfect unwanted sideband rejection, no carrier feedthrough and no distortion within the dynamic range afforded by the number of bits in the data words. A similar argument holds for a digital demodulator. If the incoming RF signal is first digitized with an ADC, then the demodulation can be done digitally without any of the artifacts caused by imperfections in the analog circuitry.

You can think of an I/Q modulator as a device that converts the analytic signal $I + jQ$ into a scalar signal at some RF frequency. The spectrum of the I/Q signal, both positive and negative frequencies, is translated upward in frequency so that it is centered on the carrier frequency. Thinking in terms of the phasor diagram, any components of the I/Q signal that are rotating counter-clockwise appear above the carrier frequency and clockwise components appear below.

15.6.2 SSB Using I/Q Modulators and Demodulators

As an example of how this works, let's walk through the process of generating an upper-sideband signal using an I/Q modulator. See **Fig 15.25**. We'll first describe the mathematics in the following paragraph and then give the equivalent explanation using the phasor diagram.

The modulating signal is a sine wave at a frequency of ω_m radians per second ($\omega_m / 2\pi$ cycles per second). Because ω_m is a positive frequency the signals applied to the I/Q inputs are $I(t) = \cos(\omega_m t)$ and $Q(t) = \sin(\omega_m t)$. Assume the modulating frequency ω_m is much less than the RF frequency ω. The analytic signal is

$$x(t) = \left[\cos(\omega_m t) + j\sin(\omega_m t)\right]$$
$$\times\left[\cos(\omega t) - j\sin(\omega t)\right]$$

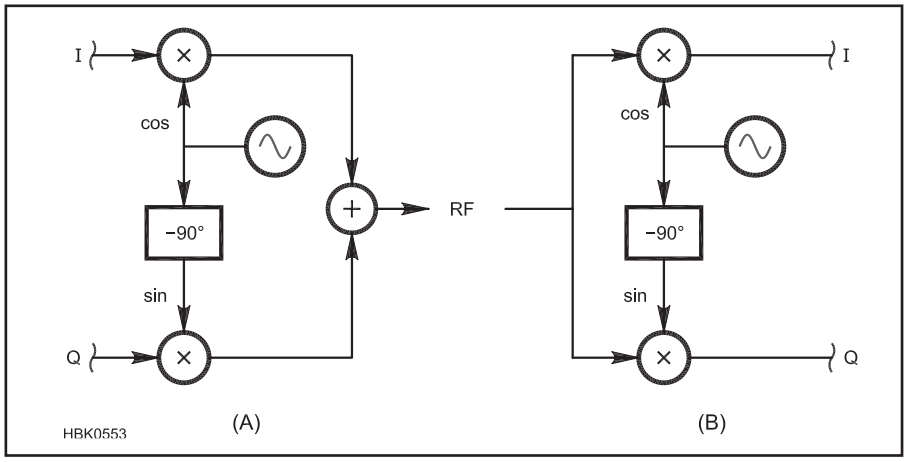

Fig 15.24 — I/Q modulator (A) and demodulator (B).

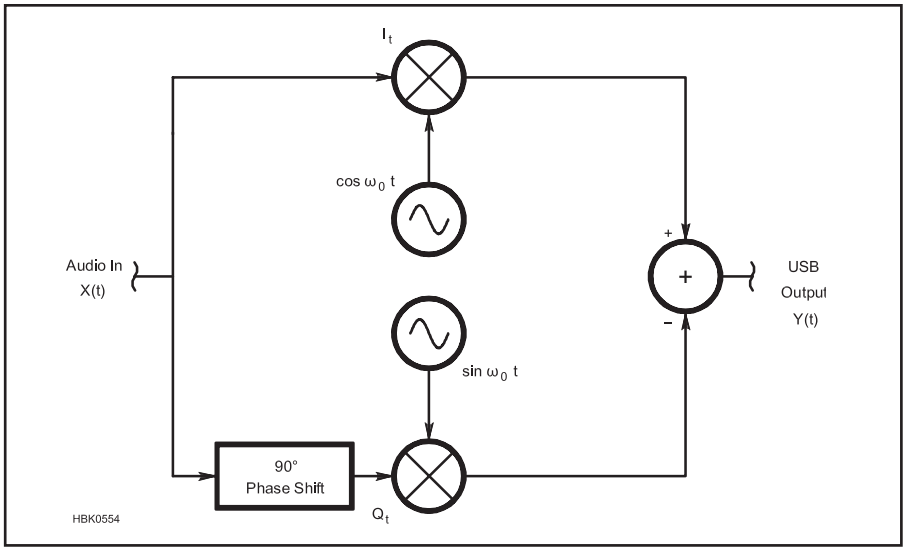

Fig 15.25 — Generating a USB signal with an I/Q modulator.

so that the real, scalar signal that appears at the modulator output is

$$\mathrm{Re}\left[x(t)\right] = \cos(\omega_m t)\cos(\omega t) + \sin(\omega_m t)\sin(\omega t)$$

At the moment when t = 0, then $\cos(\omega_m t) = 1$ and $\sin(\omega_m t) = 0$, so the real signal is just $\cos(\omega t)$, the RF signal with zero phase. One quarter of a modulation cycle later $\omega_m t = \pi/2$, so $\cos(\omega_m t) = 0$ and $\sin(\omega_m t) = 1$, and the real signal is now $\sin(\omega t)$, the RF signal with a phase of $+\pi/2$, or +90°. Every quarter cycle of the modulating signal, the RF phase, increases by 90°. That means that the RF phase increases by one full cycle for every cycle of the modulation, which is another way of saying the frequency has shifted by ω_m. We have an upper sideband at a frequency of $\omega + \omega_m$.

On the phasor diagram, the I/Q signal is rotating counterclockwise at a frequency of ω_m radians per second. As it rotates it is increasing the phase of the RF signal at the same rate, which causes the frequency to increase by ω_m radians per second. To cause the phasor to rotate in the opposite direction, you could change the polarity of either I or Q or you could swap the I and Q inputs. In that case you would have a lower sideband.

For that to work, the baseband signals applied to the I and Q inputs must be 90° out of phase. That's not hard to do for a single sine wave, but to generate a voice SSB signal, all frequencies in the audio range must be simultaneously phase-shifted by 90° without changing their amplitudes. To do that with analog components requires a broadband phase-shift network consisting of an array of precision resistors and capacitors and a number of operational amplifiers.

THE HILBERT TRANSFORMER

To do that with DSP requires a *Hilbert transformer*, an FIR filter with a constant 90° phase shift at all frequencies. Recall that a symmetrical FIR filter has a constant *delay* at all frequencies. That means that the phase shift is not constant — it increases linearly with frequency. It turns out that with an *anti-symmetrical* filter, in which the top half of the coefficients are the negative of the mirror image of the lower half, the phase shift is 90° at all frequencies, which is exactly what we need to generate an SSB signal.

The Hilbert transformer is connected in series with either the I or Q input, depending on whether USB or LSB is desired. Just as with any FIR filter, a Hilbert transformer has a delay equal to half its length, so an equal delay must be included in the other I/Q channel as shown in **Fig 15.26**. It is possible to combine the Hilbert transformer with the normal FIR filter that may be needed anyway to filter the baseband signal. The other I/Q channel then simply uses a similar filter with the same delay but without the 90° phase shift.

Because the RF output of the modulator is normally at a much higher frequency than the audio signal, it is customary to use a higher sample rate for the output signal than for the input. The FIR filters can still run at the lower rate to save processing time, and their output is then upsampled to a higher rate with an interpolator. It is convenient to use an output sample rate that is exactly four times the carrier frequency because each sample advances the RF phase by exactly 90°. The sequence of values for the sine wave is 0, 1, 0 and –1. To generate the 90° phase shift for the cosine wave, simply start the sequence at the second sample: 1, 0, –1, 0. The complete block diagram is shown in **Fig 15.27**.

A Hilbert transformer may also be used in an SSB demodulator at the receiver end of the communications system. It is basically the same block diagram drawn backwards, as illustrated in **Fig 15.28**.

Amateurs who have been in the hobby for many years may recognize this as the "phasing method" of SSB generation. It was popular when SSB first became common on the amateur bands back in the 1950s because suitable crystal filters were expensive or difficult to obtain.[5] The phasing method had the reputation of producing signals with excellent-quality audio, no doubt due to the lack of the phase distortion caused by crystal filters.

It is important to note that an ideal Hilbert transformer is impossible to construct because it theoretically has an infinitely-long impulse response. However, with a sufficiently-long impulse response, the accuracy is much better than an analog phase-shift network. Just as with an analog network, the

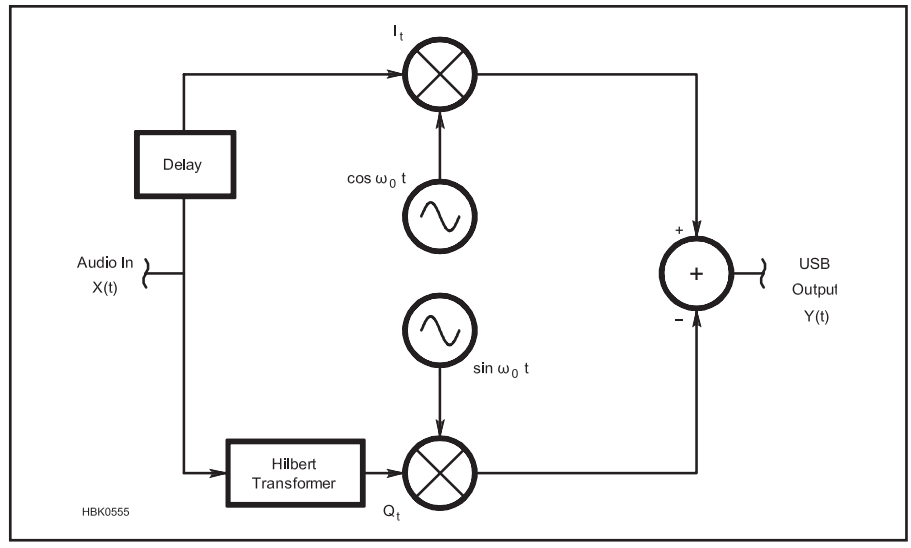

Fig 15.26 — Generating a non-sinusoidal USB signal with an I/Q modulator.

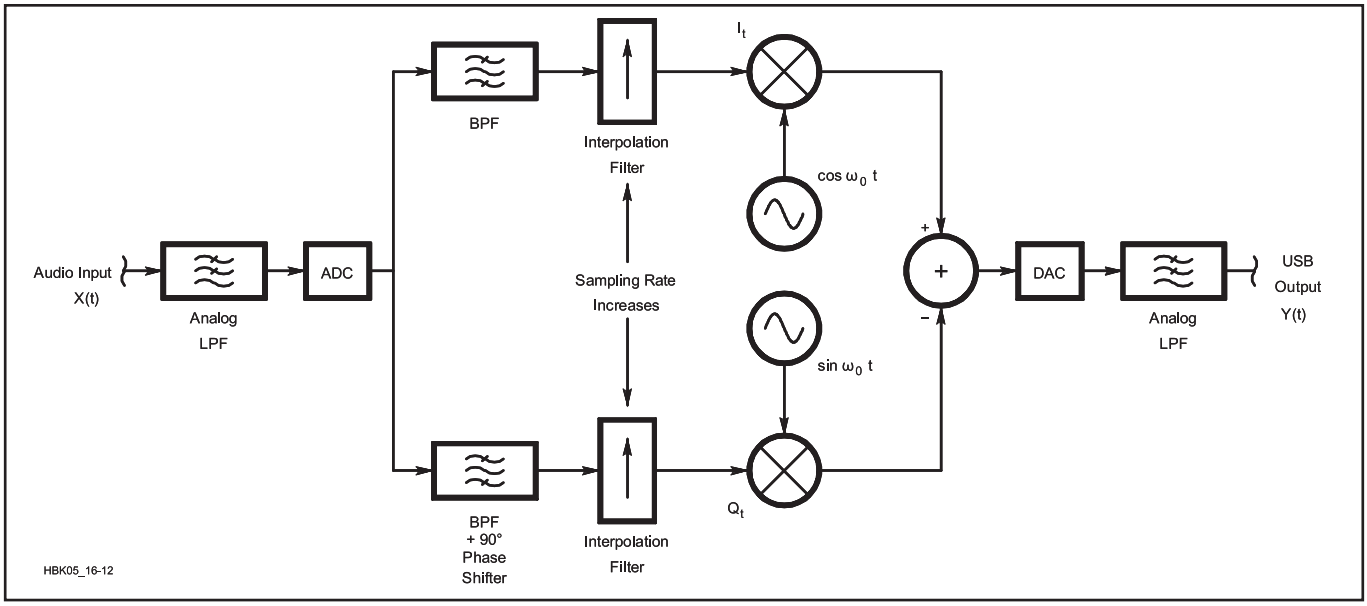

Fig 15.27 — Block diagram of a digital SSB modulator.

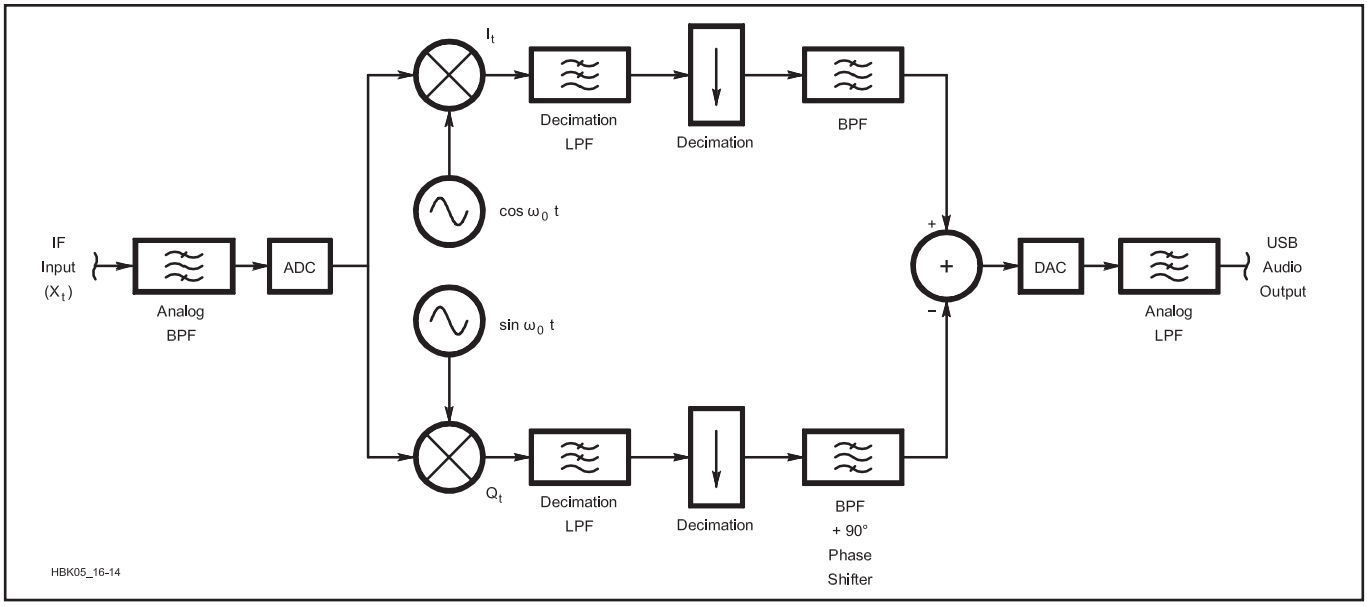

Fig 15.28 — Block diagram of a digital SSB demodulator.

frequency passband must be limited both at the low end as well as the high end. That is, the audio must be band-pass filtered before the 90° phase shift. Actually, the filtering and phase shifting can be combined into one operation using the following method.

First design a low-pass FIR filter with a bandwidth one-half the desired audio bandwidth. For example, if the desired passband is 300 to 2700 Hz, the low-pass filter bandwidth should be (2700 – 300)/2 = 1200 Hz. Then multiply the impulse response coefficients with a sine wave of a frequency equal to the center frequency of the desired passband, (2700 + 300)/2 = 1500 Hz in this case. That results in a band-pass filter with the desired 300 – 2700 Hz response. By using sine waves 90° out of phase for the I and Q channels, you end up with two band-pass filters with the same amplitude response and delay but a 90° phase difference at all frequencies. Multiply by a cosine for zero phase and by a sine for a 90° phase shift.

Old timers may notice that this bears a striking resemblance to the Weaver method, the so-called "third method" of SSB generation, that was used back in the late 1950s to eliminate the need for a wide-band audio phase-shift network.[6,7] It is almost as if there is no such thing as truly new technology, just old ideas coming back with new terminology!

Analog modulators and demodulators using the phasing and Weaver methods are covered in the **Transmitters and Transceivers** and **Mixers, Modulators and Demodulators** chapters.

USES FOR I/Q MODULATORS AND DEMODULATORS

While I/Q modulators and demodulators can be used for analog modes such as SSB, they really shine when used with digital modulation modes. The **Modulation** chapter shows how the modulation states of the various digital formats map to positions in the phasor diagram, what is called a *constellation*

diagram. The transmitter can generate the correct modulation states simply by placing the correct values on the I and Q inputs to the I/Q modulator. In the receiver, the filtered I and Q values are sampled at the symbol decision times to determine which modulation state they most closely match.

I/Q modulators and demodulators can also be used as so-called *imageless mixers*. A normal mixer with inputs at f_1 and f_2 produces outputs at f_1, f_2, $f_1 + f_2$, and $f_1 - f_2$. A balanced mixer eliminates the f_1 and f_2 terms but both the sum and difference terms remain, even though normally only one is desired. By feeding an RF instead of AF signal into the input of an SSB modulator, we can choose the sum or difference frequency in the same way as choosing the upper or lower sideband. If the input signal is a sine wave, the Hilbert transformer can be replaced by a simple 90° phase shifter. Similarly, a mixer with the same architecture as an SSB demodulator can be used to downconvert an RF signal to IF with zero image response. Analog imageless mixers are covered in the **Receivers** chapter. They are sometimes used in microwave receivers and transmitters where it is difficult to build filters narrow enough to reject the image response, but they typically only achieve image rejection in the 20-30 dB range. With a digital imageless mixer, the image rejection is "perfect" within the dynamic range of the bit resolution.

15.7 Software-Defined Radios (SDR)

There has been much, sometimes heated, discussion about the precise definition of a *software-defined radio* (SDR). Most feel that, at minimum, an SDR must implement in software at least some of the functions that are normally done in hardware. Others feel that a radio doesn't count as an SDR unless nearly all the signal-processing functions, from the input mixer to the audio output (for the receiver) and from the microphone ADC to the power amplifier input (for the transmitter), are done in software. Others add the requirement that the software must be reconfigurable by downloading new code, preferably open-source. For our purposes we will use a rather loose definition and consider any signal-processing function done in software to fall under the general category of SDR.

Some SDRs use a personal computer to do the computational heavy lifting and external hardware to convert the transmitted and received RF signals to lower-frequency signals that the computer's sound card can handle. Some SDRs avoid the use of the sound card by including their own audio codec and transferring the data to the PC via a USB port. Modern PCs provide a lot of computational power for the buck and are getting cheaper and more powerful all the time. They also come with a large color display, a keyboard for easy data entry, and a large memory and hard disk, which allows running logging programs and other software while simultaneously doing the signal processing required by the SDR.

Other SDRs look more like conventional analog radios with everything contained in one box, which makes for a neater, more compact installation. The signal processing is done with one or more embedded DSPs. For those who prefer a knob and button user interface, this is much preferred to having to use a mouse. Especially for contesting and competitive DXing, it is much faster to have a separate control for each critical function rather than having to select from pull-down menus. In addition SDRs of this type often have some performance advantages over PC-based SDRs, as we shall see.

Either method offers all the most-important advantages of applying DSP techniques to signal processing. The channel filter can have a much better *shape factor* (the ratio between the width of the passband and the frequency difference of the stopband edges). FIR filters are linear phase and have less ringing than analog filters of the same bandwidth and shape factor. Once the signal is in the digital domain all the fancy digital signal processing algorithms can be applied such as automatic notch filters, adaptive channel equalization, noise reduction, noise blanking, and feed-forward automatic gain control. Correcting bugs, improving performance or adding new features is as simple as downloading new software.

15.7.1 SDR Hardware

The transition between analog and digital signals can occur at any of several places in the signal chain between the antenna and the human interface. Back in 1992, Dave Hershberger W9GR designed an audio-frequency DSP filter based on the TMS320C10, one of the earliest practical DSP chips available.[8] This was an external unit that plugged into the headphone jack of a receiver and included FIR filters with various bandwidths, an automatic multi-frequency notch filter, and an adaptive noise filter. The advantage of doing the DSP at AF is that it can easily be added to an unmodified analog radio as in **Fig 15.29**. It is the technique used today to implement many digital modulation modes using the sound card of a PC connected to the audio input and output of a conventional transceiver.

A related technique is to downconvert a slice of the radio spectrum to baseband audio using a technique similar to the direct-conversion receivers popular with simple low-power CW transceivers. This idea was pioneered by Gerald Youngblood, AC5OG (now K5SDR), with the SDR-1000 transceiver, which he described in a series of *QEX* articles in 2002-2003.[9] The receiver block diagram is shown in Fig **15.30.** It uses a unique I/Q demodulator designed by Dan Tayloe, N7VE, to convert the RF frequency directly to baseband I and Q signals, which are fed to the stereo input of a PC's sound card, represented by the low-pass filters and A/D converters in the figure.[10] Software in the PC does all the signal processing and demodulation. The transmitter is the

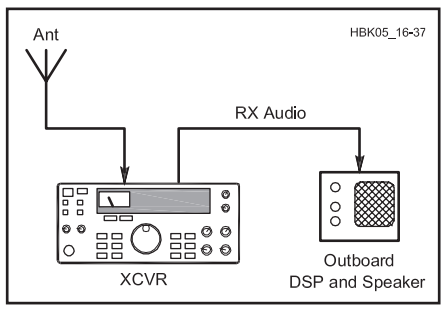

Fig 15.29 — An outboard DSP processor.

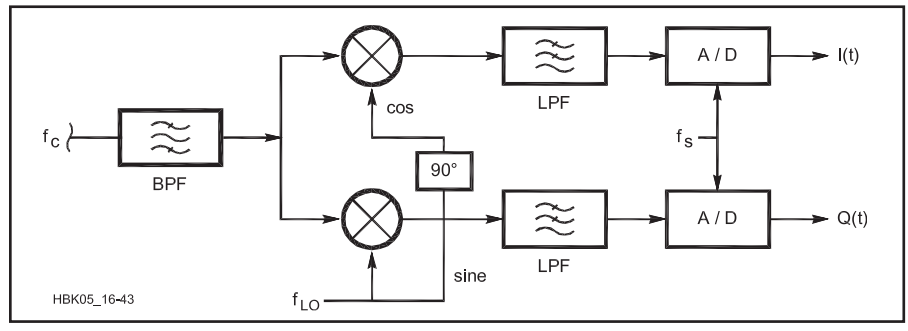

Fig 15.30 — Block diagram of K5SDR's direct-conversion software-defined receiver.

same block diagram in reverse, with an I/Q modulator converting an I/Q signal from the sound card up to the RF frequency where it is filtered and amplified to the final power level.

The sound card method manages to achieve reasonable performance with simple inexpensive hardware. Once the signal is digitized by the A/D converters in the sound card, the powerful DSP capability of the PC can do amazing things with it. The software for the SDR-1000 is open-source and available for free download on the Web.[11] In addition to implementing conventional transceiver functions such as several types of detector, variable-bandwidth filters, software AGC, an S-meter and speech compression, the software includes some extra goodies such as an automatic notch filter, noise reduction, and a panadapter spectrum display.

The simple hardware does impose some performance limitations. Because of imperfections in the analog downconverter, unwanted-sideband rejection is not perfect. This is called "image rejection" in the SDR-1000 literature. On the panadapter display, strong signals show up weakly on the opposite side of the display, equally-spaced from the center. Dc offset in the analog circuitry causes a spurious signal to appear at the center of the bandwidth. To prevent an unwanted tone from appearing in the audio output, the software demodulator is tuned slightly off frequency, but that means interference at the image frequency can cause problems because of the imperfect image rejection. The dynamic range depends on the sound card performance as well as the RF hardware. Some newer SDRs include an integrated audio codec optimized for the application so that the PC's sound card is not needed.

Among the integrated, one-box, software-defined radios, the most common place to perform the analog-digital transition is at an intermediate frequency. In the receiver, placing the ADC after a crystal IF filter improves the *blocking dynamic range* (BDR) for interfering signals that fall outside the crystal filter bandwidth. BDR is the ratio, expressed in dB, between the noise level (normally assuming a 500 Hz bandwidth) and an interfering signal strong enough to cause 1 dB gain reduction of the desired signal. With careful design, a receiver with such an architecture can achieve up to about 140 dB of BDR. The third-order dynamic range is similar to what can be achieved with a conventional analog architecture since the circuitry up to the crystal filter is the same.

Another advantage of the IF-based approach compared to sampling right at the final RF frequency is that the ADC does not have to run at such a high sample rate. In fact, because the crystal filter acts as a high-performance, narrow-bandwidth anti-aliasing filter, undersampling is possible. With bandwidths

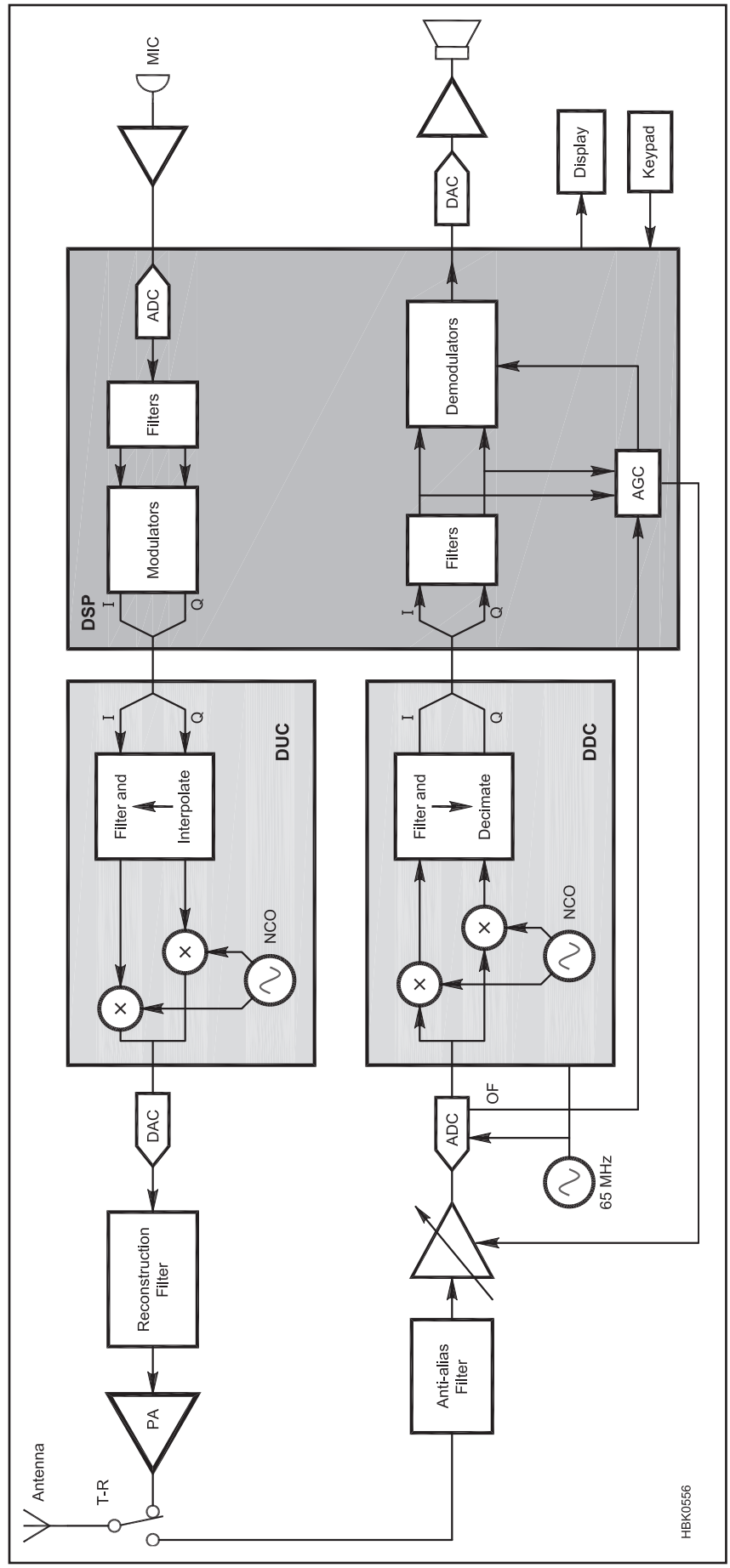

Fig 15.31 — An SDR transceiver that samples directly at the RF frequency.

of a few kHz or less, sample rates in the 10s of kHz can be used even though the center frequency of the IF signal is much higher, so long as the ADC's sample-and-hold circuit has sufficient bandwidth. Another common approach is to add a conventional analog mixer after the crystal filter to heterodyne the signal down to a second, much lower IF in the 10-20 kHz range, which is then sampled by an inexpensive, low-sample-rate ADC in the normal fashion. IF-based SDRs tend to have the highest overall performance at the expense of additional complexity.

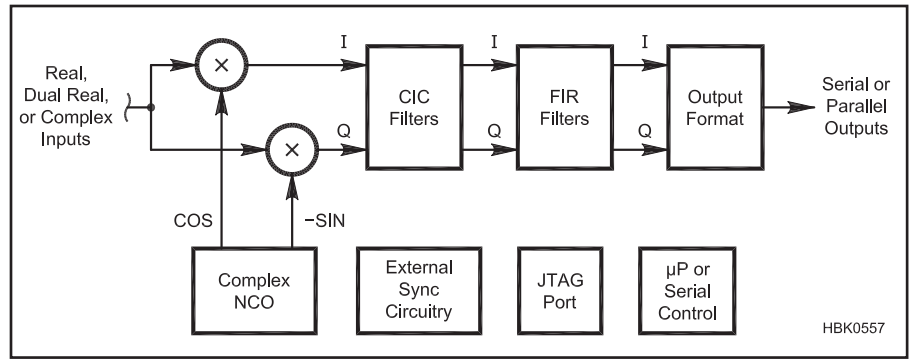

Fig 15.32 — The Analog Devices AD6620 is a digital downconverter (DDC) IC. The CIC filters and FIR filter are all decimating types.

SAMPLING AT RF

The ultimate SDR architecture is to transition between the analog and digital domains right at the frequency to be transmitted or received. In the receiver, the only remaining analog components in the signal chain are a wide-band anti-aliasing filter and an amplifier to improve the noise figure of the ADC. See **Fig 15.31**. The local oscillator, mixer, IF filters, AGC, demodulators and other circuitry are all replaced by digital hardware and software. It has only been fairly recently that low-cost high-speed ADCs have become available with specifications good enough to allow reasonable performance in a communications receiver. Today it is possible to achieve blocking dynamic range in the low 120s of dB. That is not as good as the best analog radios but is comparable to some medium-priced models currently available on the Amateur Radio market.

Third-order dynamic range is not a meaningful specification for this type of radio because it assumes that distortion products increase 3 dB for each 1 dB increase in signal level, which is not true for an ADC. The level of the distortion products in an ADC tends to be more-or-less independent of signal level until the signal peak exceeds full scale, at which point the distortion spikes up dramatically. Compared to a conventional analog mixer, ADCs tend to give very good results with a two-tone test but don't do as well when simultaneously handling a large number of signals, which results in a high peak-to-average ratio. It is important to read the data sheet carefully and note the test conditions for the distortion measurements.

There are definite advantages to sampling at RF. For one thing, it saves a lot of analog circuitry. Even if the ADC is fairly expensive the radio may be end up being cheaper because of the reduced component count. Performance is improved in some areas. For example, image rejection is no longer a worry, as long as the anti-aliasing filter is doing its job. The dynamic range theoretically does not depend on signal spacing — close-in dynamic range is often better than with a conventional architecture that uses a wide roofing filter. With no crystal filters in the signal chain, the entire system is completely linear-phase which can improve the quality of both analog and digital signals after demodulation.

The biggest challenge with RF sampling is what to do with the torrent of high-speed data coming out of the receiver ADC and how to generate transmit data fast enough to keep up with the DAC. To cover the 0-30 MHz HF range without aliasing requires a sample rate of at least 65 or 70 MHz. That is much faster than a typical microprocessor-type DSP can handle. The local oscillator, mixer and decimator or interpolator must be implemented in digital hardware so that the DSP can send and receive data at a more-reasonable sample rate. Analog Devices makes a series of *digital downconverters* (DDC) which perform those functions and output a lower-sample-rate digital I/Q signal to the DSP.[12] See **Fig 15.32**. It would also be possible to implement your own DDC in an FPGA. The same company also makes *digital upconverters* (DUC) that do the same conversion in reverse for the transmitter. Some of their DUCs even include the capability to encode several digital modulation formats such as GMSK, QPSK and π/4 DQPSK.

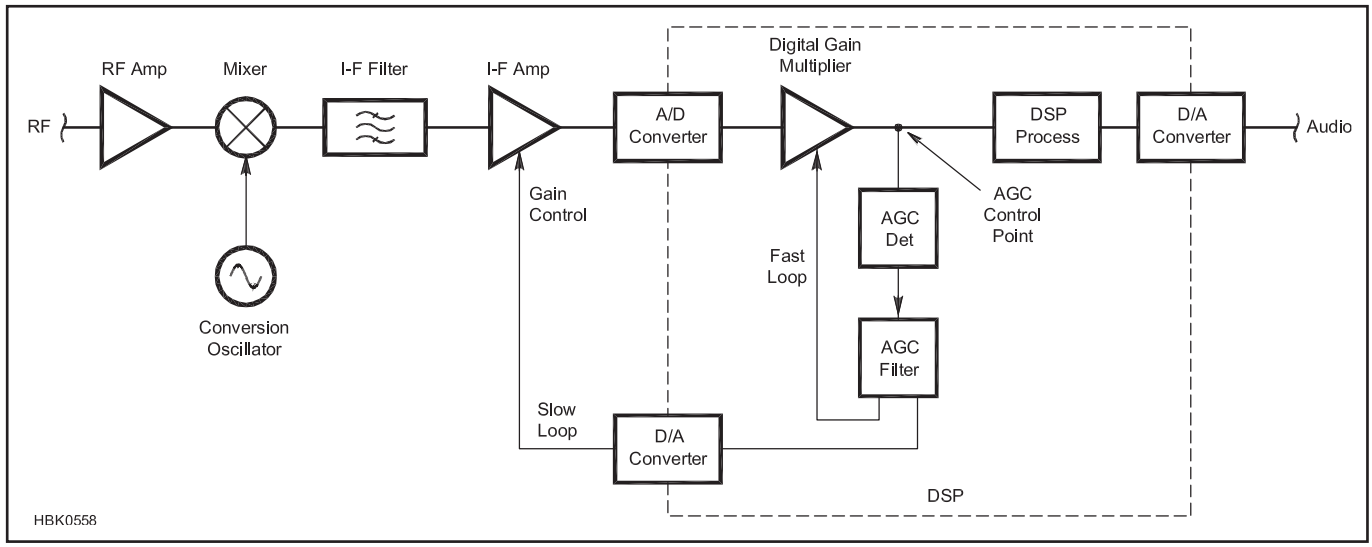

Fig 15.33 — DSP-based feedback type of AGC showing a combination of analog and digital gain-control points.

DIGITAL AGC

In the transmitter portion of a software-defined radio, dynamic range is generally not a problem because the transmitted signal always has approximately the same power level. Even if power control is implemented digitally, a 1-100 W adjustment range only adds 20 dB to the dynamic range. The receiver is another story. Assuming 6 dB per S-unit, the difference between S1 and 60 dB over S9 is 108 dB. Considering peak power rather than average, the actual range is much greater than that. While AGC implemented in software can be quite effective in regulating the signal level at the speaker output, it does nothing to prevent the ADC from being overloaded on signal peaks. For that, some kind of hardware AGC is needed in the signal path ahead of the A/D converter. This device could be a switched attenuator or variable-gain amplifier as illustrated in **Fig 15.33**. It normally runs at full gain and only attenuates the incoming signal when very strong peak signal levels are encountered. It could be totally self-contained with its own threshold detector or it can be controlled as shown by the DSP, which activates the hardware AGC whenever it detects ADC overflow.

One issue with most AGC systems is response time. The level detector is normally placed after the gain-control stage. Because of delays in the feedback loop, by the time the AGC circuit detects an over-range condition it is already too late to reduce the gain without overshoot. One of the advantages of digital AGC is that it is easy to use feed-forward rather than feed-back control. In **Fig 15.34**, the gain control stage is placed after the level detector. A small delay is included in the signal path so that the AGC circuit can reduce the gain just before the large signal arrives at the gain multiplier. With proper design, that totally eliminates overshoot and makes for a very smooth-operating AGC.

THE LOCAL OSCILLATOR

With direct-RF sampling, the digital local oscillator is normally implemented with a direct digital synthesizer, operating totally in digital hardware. DDS operation was explained previously in the sine-wave generation section. A separate DDS chip with a built-in DAC is sometimes used in IF-sampled SDRs as well as in some analog radios. One advantage that a DDS oscillator has over a phase-locked loop (PLL) synthesizer is very fast frequency changing. That can be important in transceivers that use the same local oscillator for both the receiver and transmitter. If the transmitter and receiver are tuned to different frequencies, each time the rig is keyed the LO frequency must settle at its new value before a signal is transmitted.

The phase noise of the DDS clock is just as important as the phase noise of the local oscillator in a conventional radio. Phase noise shows up as broadband noise that gradually diminishes the farther you get from the oscillator frequency. In a receiver, phase noise causes a phenomenon called *reciprocal mixing*, in which a strong off-channel signal mixes with off-channel phase noise to cause a noise-modulated spurious signal to appear in the receiver passband. In many receivers, dynamic range measurements are phase-noise-limited because the spurious response due to reciprocal mixing is louder than the distortion products. One way to reduce the phase noise from the DDS is to use a conventional PLL synthesizer to generate a signal with large frequency steps and combine it with a DDS synthesizer to obtain the fine-grained frequency resolution, as suggested in **Fig 15.35**. In this way, you get the phase noise of the DDS and PLL within the loop bandwidth of the PLL and the phase noise of the VCO outside that bandwidth.

One advantage a PLL has over a DDS oscillator is lower spurious signal levels. A DDS with a wideband spurious-free dynamic range (SFDR) specification of 60 dB would be better than most, but that could cause spurious responses in the receiver only 60 dB down. The hybrid PLL/DDS technique can suppress these spurs as well.

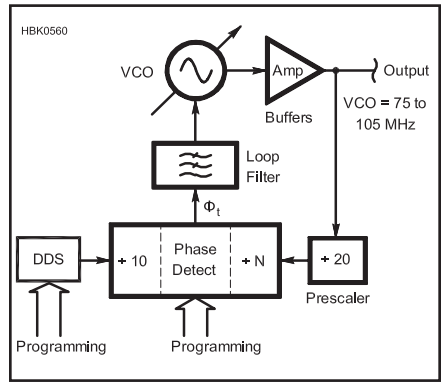

Fig 15.35 — A hybrid DDS/PLL synthesizer.

15.7.2 SDR Software

When designing DSP software, it is sometimes surprising how much of your intuition about analog circuits and systems transfers directly over to the field of digital signal processing. The main difference is that you need to forget much of what you have learned about the imperfections of analog circuitry. For example, a multiplier is the DSP equivalent of an ideal double-balanced mixer. The multiplier output contains frequencies only at the sum and difference of the two input signals. There is no intermodulation distortion to create spurious frequencies.

Multiplication by a constant is equivalent to an amplifier or attenuator, but with no dc offset and with a very precisely-set gain that

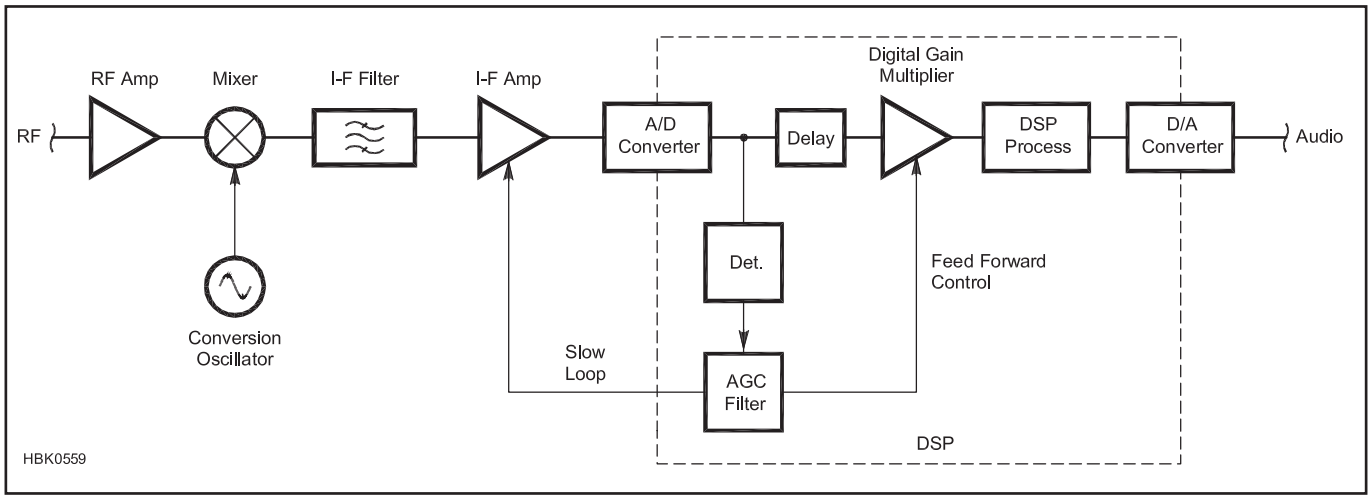

Fig 15.34 — DSP-based AGC with analog feedback and digital feed forward control.

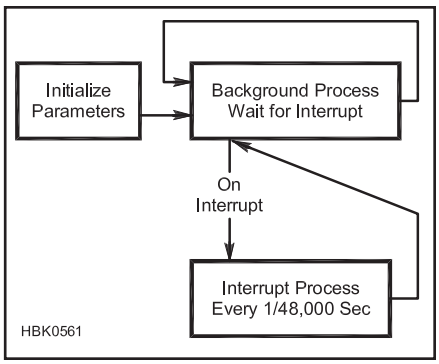

Fig 15.36 — Main program flow of a typical DSP program.

does not drift with time or temperature. A distortionless AM "diode" detector is just a software routine that forces the signal to zero whenever it is negative. To build an ideal full-wave diode detector, just take the absolute value of the signal.

When you design an analog circuit you have to take into account all the things that don't appear on the schematic. For example your crystal filter may show a beautiful frequency response in your filter design program, but in the actual circuit the passband is skewed because of the input impedance of the post-amplifier and the stopband response is degraded by signals leaking around the filter due to the PC board layout. With software, what you see in the simulation is what you get (assuming you did the calculations correctly!)

SOFTWARE ARCHITECTURE

Incoming data to the DSP from A/D converters or other devices is normally handled in an *interrupt service routine* (ISR) that is called automatically whenever new data is ready. The AM and AGC detection code in Table 15.5 could be included right in the ISR, but it is almost always better to limit the ISR function to the minimum necessary to service the hardware that calls the interrupt and do all the signal processing elsewhere. For example, **Table 15.4** shows an ISR that inputs 16-bit I and Q data coming in on the dsPIC serial data communications interface (DCI). The first line is a secret incantation that defines this function to be the interrupt service routine for the DCI interface.

As you can see, there is minimal functionality in the ISR itself. All the heavy computational lifting is done in processes that run in the background and are interrupted periodically when new data is available. The data_flag variable is a semaphore to signal the signal-processing routine that new data is ready. **Fig 15.36** illustrates the basic program architecture of some DSP projects for the Analog Devices EZ-Kit Lite DSP development board described in the ARRL publication *Experimental Methods in RF Design*. That book, by the way, is an excellent source for practical "how-to" information on designing DSP projects.

SOFTWARE MODULATORS AND DEMODULATORS

In this chapter we've already covered many of the algorithms needed for a software-defined radio. For example, we know how to make I/Q modulators and demodulators and use them to build an SSB modulator and detector. Let's say we want our software-defined transceiver to operate on AM voice as well. How do you make an AM modulator and demodulator?

The modulator is easy. Simply add a constant value, representing the carrier, to the audio signal and multiply the result by a sine wave at the carrier frequency, as shown in **Fig 15.37**.

Demodulation is almost as easy. We could just simulate a full-wave rectifier by taking the absolute value of the signal, as mentioned previously, and low-pass filter the result to remove the RF energy. If the signal to be demodulated is complex, with I and Q components, then instead of absolute value we take the magnitude

$$A = \sqrt{I^2 + Q^2}$$

The dc bias can be removed by adding a "series blocking capacitor" — a high-pass filter with a suitable cut-off frequency.

A little more elegant way to do it would be to include the AM detector as part of the AGC loop. In the C code snippet shown in **Table 15.5**, the variable "carrier" is the average AM carrier level. It is passed to another subroutine to control the gain.

Note that no "series capacitor" is needed since the audio signal is computed by subtracting the average historical value, carrier, from the magnitude of the current I/Q signal, am. A small fraction of its value is added to the historical value so that the AGC tracks the average AM carrier level. AGC speed is controlled by that fraction. Dividing by 2^{10} = 1024 gives a time constant of about 1024 clock cycles.

Another type of detector we haven't discussed yet is for frequency modulation. For a scalar signal, the *quadrature detector* shown in **Fig 15.38** is one elegant solution. This is the same circuit whose analog equivalent is used today in millions of FM receivers around the world. In the digital implementation, the delay block is a FIFO buffer constructed from a series of shift registers. Multiplying the signal by a delayed version of itself gives an output with a cosinusoidal response versus frequency. The response crosses zero whenever the carrier

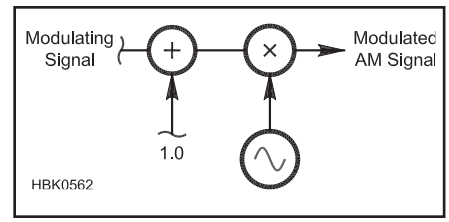

Fig 15.37 — A digital AM modulator.

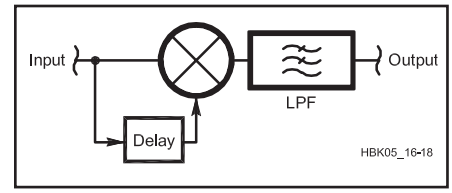

Fig 15.38 — A digital quadrature detector.

Table 15.4
Interrupt Service Routine

```
void __attribute__((auto_psv, interrupt)) _DCIInterrupt(void)
{
extern int i, q, data_flag;

// Clear interrupt flag:
IFS3bits.DCIIF = 0;
// Input data is 2's complement:
i = RXBUF0;
q = RXBUF1;
data_flag = 1;
return;
}
```

Table 15.5
AM Detector

```
static long int carrier;
long int am;
int i, q, signal;

/* Code that generates i and q omitted */
am = (long int)sqrt((long int)i*i + (long int)q*q);
signal = am - carrier;
// Divide signal by 2^10:
carrier += signal >> 10;
// Audio output to DAC via SPI bus:
SPI1BUF = signal;
```

frequency is f = N/(4τ), where N is an odd integer and the delay in seconds is τ=n/fs, where n is the number of samples of delay and f_s is the sample frquency. As the carrier deviates above and below the zero-crossing frequency the output varies above and below zero, just what we want for an FM detector.

For an I/Q signal, probably the most straightforward FM detector is a phase detector followed by a differentiator to remove the 6 dB per octave rolloff caused by the phase detector. The phase is just

$$\varphi = \arctan\left(\frac{Q}{I}\right)$$

You have to be a little careful since there is a 180° phase ambiguity in the arctangent function. For example,

$$\arctan\left(\frac{1}{1}\right) = \arctan\left(\frac{-1}{-1}\right)$$

Software will have to check which quadrant of the phasor diagram the I/Q signal is in and add 180° when necessary. If there is no arctan function in the library, one can be constructed using a look-up table. Frequency is the derivative of the phase. A differentiator is nothing more than a subtractor that takes the difference between successive samples.

$$f = \frac{\varphi_n - \varphi_{n-1}}{2\pi\, f_s}$$

where n is the sample number and f_s is the sample rate. It is important to make sure that the difference equation functions properly around 360°. If the phase variable is scaled so that 360° equals the difference between the highest and lowest representable numbers, then standard two's complement subtraction should roll over to the right value at the 360° / 0° transition. Another thing to watch out for is that the derivative of the phase may be a rather small signal, so it might be necessary to carry through all the calculations using long integers or floating point numbers.

OTHER SOFTWARE FUNCTIONS

A *carrier-locked loop* is a circuit that automatically tunes a receiver so that it is centered on the carrier of the incoming signal. One way to achieve that is to make the receiver local oscillator controllable by a frequency detector. For example if the local oscillator (LO) in the receiver were an analog voltage-controlled oscillator (VCO), the output of the FM detector described above could be applied to a DAC that generates an error voltage to pull the VCO on frequency. If the LO were an NCO or other digitally-controlled synthesizer, then the error signal could be used to control the frequency digitally. An even more elegant way to do it is to leave the LO alone and tune the frequency of the I/Q signal directly. Conceptually, you determine the amplitude

$$A = \sqrt{I^2 + Q^2}$$

and phase

$$\varphi = \arctan\left(\frac{Q}{I}\right)$$

of the signal, add or subtract the phase error from ϕ to keep its average value from changing and then convert back to I = Acos(ϕ) and Q = Asin(ϕ) again for further processing. This is easy to do with a signal that does not change phase, such as AM phone. For an FM or PM signal, considerable averaging must be done of the error signal so that it represents the average phase of the carrier rather than the instantaneous phase of the modulation.

Speech processing is a function that lends itself well to digital signal processing. The human voice has a high peak-to-average power ratio, typically on the order 15 dB. That means, that without processing, a 100-W PEP SSB transmitter is only putting out about 3 W average! Most SSB transmitters do have an *automatic level control* (ALC) circuit that can reduce the peak-to-average ratio by 3-6 dB, but that still means your 100-W transmitter is only putting out 6-12 W on average.

The problem is that if the ALC setting is too aggressive, considerable distortion of the audio can result. A transmitter's ALC circuit operates much like the AGC in a receiver. It can do a fair job of keeping the peak power from overdriving the amplifier but it can do little to reduce the short-term power variations between speech syllables. With digital processing, it is fairly easy to use feed forward gain control rather than feed back, in a manner similar to the AGC system illustrated in Fig 15.34.

The gold standard of speech processing of SSB signals is RF clipping. By clipping at radio frequency instead of at audio, many of the distortion products fall outside the passband where they can be filtered out, by a crystal filter in an analog radio or by a digital band-pass filter in an SDR. "RF" clipping doesn't actually have to be done at a high RF frequency. An IF of a few kHz is sufficient, so long as the center frequency is greater than about twice the audio bandwidth. That can all be done in software and then the signal can be converted back to baseband audio if desired.

Developing DSP software is a wonderful homebrew activity for the Radio Amateur. As electronic devices have become smaller and smaller and more and more sophisticated it has become harder and harder to get a soldering iron on the tiny pins of surface-mount ICs. Software development allows hobbyists to experiment to their heart's content with no danger that an expensive piece of electronics will be destroyed by one moment of clumsiness. With nothing more than a PC and some free software, the enthusiast can while away hours exploring the fascinating world of digital signal processing and software radios.

15.8 Glossary

Adaptive filter — A filter whose coefficients can be changed automatically.

Analog-to-digital converter (ADC) — A device that samples an analog signal and outputs a digital number representing the amplitude of the signal.

Analytic signal — A representation of the phase and amplitude of a signal (often in the form of the in-phase and quadrature components), without explicitly including the oscillating part (the carrier).

Anti-aliasing filter — A band-limiting filter placed before a sampler to make sure the incoming signal obeys the Nyquist criterion.

Anti-symmetrical — A function that is anti-symmetrical about point x=0 has the property that $f(x) = -f(-x)$.

Application-specific integrated circuit (ASIC) — A non-programmable IC that is designed for a particular application.

Arithmetic logic unit (ALU) — The portion of a microprocessor that performs basic arithmetic and logical operations.

Automatic gain control (AGC) — The circuit in a receiver that keeps the signal level approximately constant.

Automatic level control (ALC) — The circuit in a transmitter that keeps the peak signal level approximately constant.

Barrel shifter — A circuit in a microprocessor that can bit-shift a number by multiple bits at one time.

Baseband — The low-frequency portion of a signal. This is typically the modulation.

Bilinear transform — A design technique for IIR filters in which the frequency axis is transformed to prevent the filter bandwidth from violating the Nyquist criterion.

Binary point — The symbol that separates the integer part from the fractional part of a binary number.

Blocking dynamic range (BDR) — The difference between the noise level (usually in a 500-Hz bandwidth) and the signal level that causes a 1 dB reduction in the level of a weaker signal.

Carrier-locked loop — A feedback control loop to automatically tune a receiver or demodulator to the frequency of a received carrier.

Chebyshev filter — A filter with equal ripple in the passband, stopband or both.

Circular buffers — A buffer in which the final entry is considered to be adjacent to the first.

Cognitive radio — A radio system in which a wireless node automatically changes its transmission or reception parameters to avoid interference with other nodes.

Complex number — A number that contains real and imaginary parts.

Complex PLD (CPLD) — A programmable logic device that is more complex than a small PLD, such as a PAL, but with a similar architecture.

Constellation diagram — A phasor diagram showing the locations of all the modulation states of a digital modulation type.

Convolution — A mathematical operation that modifies a sequence of numbers with another sequence of numbers so as to produce a third sequence with a different frequency spectrum or other desired characteristic. An FIR filter is a convolution engine.

Cooley-Tukey algorithm — Another name for the fast Fourier transform.

Decimation — Reduction of sample rate by an integer factor.

Decimation in time — The division of a sequence of numbers into successively smaller sub-sequences in order to facilitate calculations such as the Fourier transform.

Digital downconverter (DDC) — A device that translates a band of frequencies to baseband, typically at a lower sample rate.

Digital signal processing — The processing of sequences of digital numbers that represent signals.

Digital signal processor (DSP) — A device to do digital signal processing. The term normally is understood to refer to a microprocessor-type device with special capabilities for signal processing.

Digital-to-analog converter (DAC) — A device that converts digital numbers to an analog signal with an amplitude proportional to the digital numbers.

Digital upconverters (DUC) — A device that frequency-translates a baseband signal to a higher frequency, typically at a higher sample rate.

Direct digital synthesis (DDS) — The generation of a periodic waveform by directly calculating the values of the waveform samples.

Direct form — A circuit topology for an IIR filter that corresponds directly to the standard filter equation.

Direct memory access (DMA) — The ability of a microprocessor chip to transfer data between memory and some other device without the necessity to execute any processor instructions.

Dithering — Randomly varying the amplitude or phase of a signal in order to overcome quantization effects.

Embedded system — A system that includes a microprocessor for purposes other than general-purpose computing.

Equal-ripple filter — A filter in which the variation in passband or stopband response is constant across the band.

Exponent — The number of digits that the radix point must be moved to represent a number.

Fast Fourier transform (FFT) — An algorithm that can calculate the discrete Fourier transform with an execution time proportional to nlog(n), instead of n^2 as is required by the straight-forward application of the Fourier transform equation.

Field-programmable gate array (FPGA) — An IC that contains a large array of complex logic blocks whose function and connections can be re-programmed in the field.

Filter coefficient — One of a series of numbers that define the transfer function of a filter.

Finite impulse response (FIR) — An impulse response that is zero for all time that is greater than some finite amount from the time of the impulse.

Floating-point — Refers to a number whose value is represented by a mantissa and an exponent.

Fourier transform — A mathematical operation that derives the frequency spectrum of a time-domain signal.

Hardware-description languages (HDL) — A computer language to specify the circuitry of a digital device or system.

Harmonic sampling — The use of a sample rate that is less than twice the highest frequency of the signal to be sampled. The sample rate must be greater than two times the bandwidth of the signal.

Harvard architecture — A computer architecture in which the program and data are stored in separate memories.

Hilbert transformer — A filter that creates a constant 90° phase difference over a band of frequencies.

Imageless mixer — A mixer in which the output contains only the sum or difference of the two input frequencies, but not both.

Imaginary number — A real number multiplied by the square root of minus one.

Impulse — A pulse of finite energy with a width that approaches zero.

Impulse-invariant — A design technique for IIR filters in which the impulse response is the same as the impulse response of a certain analog filter.

Impulse response — The response versus time of a filter to an impulse.

In-circuit emulator (ICE) — A device that emulates the operation of a microprocessor while providing debugging tools to the operator. The ICE normally plugs into an IC socket that normally holds the microprocessor.

In-circuit debugger (ICD) — A device that uses debugging features built into the microprocessor so that it can be tested while in the circuit.

In-circuit programmable (ICP) — A programmable IC that can be programmed while it is connected to the application circuit.

In-circuit programmer (ICP) — A device to facilitate programming of programmable ICs while they are connected to the application circuit.

In-phase (I) — The portion of a radio signal that is in phase with a reference carrier.

Infinite impulse response (IIR) — An impulse response that theoretically never goes to zero and stays there.

Integrated development environment (IDE) — An integrated collection of software and hardware tools for developing a microprocessor project.

I/Q demodulator — A device to derive the in-phase and quadrature portions of an oscillating signal.

I/Q modulator — A device to generate an oscillating signal with specified in-phase and quadrature parts.

Interpolation — Increasing the sample rate by an integer factor.

Interrupt service routine (ISR) — A software subroutine that is called automatically when the main routine is interrupted by some event.

Least mean squares (LMS) — An algorithm for adaptive filters that minimizes the mean square error of a signal.

Least-significant bit (LSB) — When used as a measurement unit, the size of the smallest step of a digital number.

Limit cycle — A non-linear oscillation in an IIR filter.

Linear phase — Refers to a system in which the delay is constant at all frequencies, which means that the phase is linear with frequency.

Mantissa — The decimal or binary part of a logarithm or floating-point number.

Multiplier-accumulator (MAC) — A device that can multiply two numbers and add the result to a previous result all in one operation.

Multi-rate — Refers to a system with more than one sample rate.

Numerically-controlled oscillator (NCO) — An oscillator that synthesizes the output frequency from a fixed timebase. A DDS oscillator.

Nyquist criterion — The requirement that the sample rate must be at least twice the bandwidth of the signal.

Nyquist frequency — One half the sample rate.

Nyquist rate — Twice the signal bandwidth.

One-time programmable (OTP) — A programmable device that may not be re-programmed.

Orthogonal — Perpendicular. In analogy with the mathematics of perpendicular geometrical vectors, the term is used in communications to refer to two signals that produce zero when convolved.

Oversampling — Use of a sample rate higher than required by the Nyquist criterion in order to improve the signal-to-noise ratio.

Parks-McClellan algorithm — An optimized design technique for equal-ripple filters.

Phasor diagram — A polar plot of the magnitude of the in-phase and quadrature components of a signal.

Pipeline — An arrangement of computational units in a microprocessor or other digital device so that different units can be working on different instructions or signal samples at the same time.

Programmable-array logic (PAL) — A type of small PLD that consists of an array of AND gates, OR gates, inverters and latches.

Programmable-logic device (PLD) — A device with many logic elements whose connections are not defined at manufacturer but must be programmed.

Quadrature (Q) — The portion of a radio signal that is 90° out of phase with a reference carrier.

Quadrature detector — An FM detector that multiplies the signal by a delayed version of the same signal.

Quantization — The representation of a continuous analog signal by a number with a finite number of bits.

Quantization error — The difference in amplitude between an analog signal and its digital samples.

Quantization noise — Noise caused by random quantization error.

Radix – The base of a number system. Binary is radix 2 and decimal is radix 10.

Radix point — The symbol that separates the integer part from the fractional part of a number.

Reciprocal mixing — A spurious response in a receiver to an off-channel signal caused by local oscillator phase noise at the same frequency offset as the interference.

Reconstruction filter — A filter located after a digital-to-analog conversion or interpolation to filter out sampling spurs.

Resampling — Changing the sample rate by a non-integer ratio.

Resolution — The number of bits required to represent a digital number from its smallest to its largest value.

Sample rate — The rate at which samples are generated, processed or output from a system.

Sampling — The process of measuring and recording a signal at discrete points of time.

Software-defined radio (SDR) — A transmitter and/or receiver whose principal signal processing functions are defined by software.

Spectral leakage — In a Fourier transform, the indication of frequencies that are not actually present in the signal due to inadequate windowing.

Tap — One processing block, consisting of a coefficient memory, signal register, multiplier and adder, of an FIR filter.

Training sequence — A sequence of one or more known symbols transmitted for the purpose of training the adaptive filter in a receiver.

Undersampling — Harmonic sampling.

Volatile memory — A memory that requires the presence of power supply voltage for data retention.

Von Neumann architecture — A computer architecture that includes a processing unit and a single separate read/write memory to hold both program and data.

Windowing — Tapering the edges of a data sequence so that the samples do not transition abruptly to zero. This avoids passband and stopband ripple in an FIR filter and spectral leakage in a Fourier transform.

Zero-order hold — Holding of a data value for the entire sample period.

Zero-overhead looping — The ability of a microprocessor to automatically jump from the end of a memory block back to the beginning without additional instructions.

Zero-stuffing — Interpolation by inserting zero-valued samples in between the original samples.

15.9 References and Bibliography

REFERENCES

1. The FCC report and order authorizing software-defined radios in the commercial service is available at **www.fcc.gov/Bureaus/Engineering_Technology/Orders/2001/fcc01264.pdf.**
2. The Microchip IDE software can be downloaded free at **www.microchip.com**.
3. Mar, chapter 4. [See bibliography]
4. This is Euler's famous formula, $e^{jx} = \cos(x) + j\sin(x)$, named after Leonhard Euler, an 18th-century mathematician. The factor $e \approx 2.718$ is the base of the natural logarithm and $j = \sqrt{-1}$ is the imaginary unit.
5. Blanchard, R., W6UYG, "Sugar-Coated Single Sideband," *QST*, Oct 1952, p 38ff.
6. Weaver, D.K., Jr, "A Third Method of Generation and Detection of Single-Sideband Signals" *Proc. IRE*, Dec. 1956.
7. Wright, Jr., H., W1PNB, "The Third Method of S.S.B.," *QST*, Sep 1957, pp 11-15.
8. Hershberger, D., W9GR, "Low-Cost Digital Signal Processing for the Radio Amateur," *QST*, Sep 1992, pp 43-51.
9. Youngblood. (See the Articles section below.)
10. Tayloe, D., N7VE, "Notes on 'Ideal' Commutating Mixers (Nov/Dec 1999)" Letters to the Editor, *QEX*, Mar 2001, p 61.
11. *PowerSDR* software is available on the Flex Radio Web site, **www.flex-radio.com**.
12. Analog Devices has lots of good free tutorial literature available for download on their Web site, **www.analog.com**.

BIBLIOGRAPHY

(Key: **D** = disk included, **A** = disk available, **F** = filter design software)

DSP Software Tools

Alkin, O., *PC-DSP*, Prentice Hall, Englewood Cliffs, NJ, 1990 (**DF**).

Kamas, A. and Lee, E., *Digital Signal Processing Experiments*, Prentice Hall, Englewood Cliffs, NJ, 1989 (**DF**).

Momentum Data Systems, Inc., *QEDesign*, Costa Mesa, CA, 1990 (**DF**).

Stearns, S. D. and David, R. A., *Signal Processing Algorithms in FORTRAN and C*, Prentice Hall, Englewood Cliffs, NJ, 1993 (**DF**).

Textbooks

Frerking, M. E., *Digital Signal Processing in Communication Systems*, Van Nostrand Reinhold, New York, NY, 1994.

Hayward, Wes, W7ZOI et al, *Experimental Methods in RF Design*, chapter 10 "DSP Components" and chapter 11 "DSP Applications in Communications," ARRL, 2009 (**D**).

Ifeachor, E. and Jervis, B., *Digital Signal Processing: A Practical Approach*, Addison-Wesley, 1993 (**AF**).

Madisetti, V. K. and Williams, D. B., Editors, *The Digital Signal Processing Handbook*, CRC Press, Boca Raton, FL, 1998 (**D**).

Mar, Amy (Editor), *Digital Signal Processing Applications Using the ADSP-2100 Family, Volume I*, Prentice Hall 1990. While oriented to the ADSP-2100, there is much good general information on DSP algorithms. Both volume I and II are available for free download on the Analog Devices Web site, **www.analog.com**.

Oppenheim, A. V. and Schafer, R. W., *Digital Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1975.

Parks, T. W. and Burrus, C.S., *Digital Filter Design*, John Wiley and Sons, New York, NY, 1987.

Proakis, J. G. and Manolakis, D., *Digital Signal Processing*, Macmillan, New York, NY, 1988.

Proakis, J. G., Rader, C. M., et. al., *Advanced Digital Signal Processing*, Macmillan, New York, NY, 1992.

Rabiner, L. R. and Schafer, R. W., *Digital Processing of Speech Signals*, Prentice Hall, Englewood Cliffs, NJ, 1978.

Rabiner, L. R. and Gold, B., *Theory and Application of Digital Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1975.

Sabin, W. E. and Schoenike, E. O., Eds., *HF Radio Systems and Circuits*, rev. 2nd ed., Noble Publishing Corp, Norcross, GA, 1998.

Smith, D., *Digital Signal Processing Technology: Essentials of the Communications Revolution*, ARRL, 2001.

Widrow, B. and Stearns, S. D., *Adaptive Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1985.

Zverev, A. I., *Handbook of Filter Synthesis*, John Wiley and Sons, New York, NY, 1967.

Articles

Albert, J. and Torgrim, W., "Developing Software for DSP," *QEX*, Mar 1994, pp 3-6.

Ahlstrom, J., N2ADR, "An All-Digital SSB Exciter for HF," *QEX*, May/Jun 2008, pp 3-10.

Ahlstrom, J., N2ADR, "An All-Digital Transceiver for HF," *QEX*, Jan/Feb 2011, pp 3-8.

Anderson, P. T., "A Simple SSB Receiver Using a Digital Down-Converter," *QEX*, Mar, 1994, pp 17-23.

Anderson, P. T., "A Faster and Better ADC for the DDC-Based Receiver," *QEX*, Sep/Oct 1998, pp 30-32.

Applebaum, S. P., "Adaptive arrays," *IEEE Transactions Antennas and Propagation*, Vol. PGAP-24, pp 585-598, September, 1976.

Åsbrink, L., "Linrad: New Possibilities for the Communications Experimenter," *QEX*, Part 1, Nov/Dec 2002; Part 2, Jan/Feb 2003; Part 3 May/Jun 2003.

Ash, J. et al., "DSP Voice Frequency Compandor for Use in RF Communications," *QEX*, Jul 1994, pp 5-10.

Bloom, J., "Measuring SINAD Using DSP," *QEX*, Jun 1993, pp 9-18.

Bloom, J., "Negative Frequencies and Complex Signals," *QEX*, Sep 1994.

Bloom, J., "Correlation of Sampled Signals," *QEX*, Feb 1996.

Brannon, B., "Basics of Digital Receiver Design," *QEX*, Sep/Oct 1999, pp 36-44.

Cahn, H., "Direct Digital Synthesis — An Intuitive Introduction," *QST*, Aug 1994, pp 30-32.

Cercas, F. A. B., Tomlinson, M. and Albuquerque, A. A., "Designing With Digital Frequency Synthesizers," *Proceedings of RF Expo East*, 1990.

de Carle, B., "A Receiver Spectral Display Using DSP," *QST* Jan 1992, pp 23-29.

Dick, R., "Tune SSB Automatically," *QEX*, Jan/Feb 1999, pp 9-18.

Dobranski, L., "The Need for Applications Programming Interfaces (APIs) in Amateur Radio," *QEX*, Jan/Feb 1999, pp 19-21.

Emerson, D., "Digital Processing of Weak Signals Buried in Noise," *QEX*, Jan 1994, pp 17-25.

Forrer, J., "Programming a DSP Sound Card for Amateur Radio," *QEX*, Aug 1994.

Forrer, J., KC7WW, "A DSP-Based Audio Signal Processor," *QEX*, Sep 1996.

Forrer, J., KC7WW, "Using the Motorola DSP56002EVM for Amateur Radio DSP Projects," *QEX*, Aug 1995.

Gradijan, S., WB5KIA, "Build a Super Transceiver — Software for Software Controllable Radios," *QEX*, Sept/Oct 2004, pp 30-34.

Green, R., "The Bedford Receiver: A New Approach," *QEX*, Sep/Oct, 1999, pp 9-23.

Hale, B., "An Introduction to Digital Signal Processing," *QST*, Sep 1992, pp 43-51.
Hershberger, D., W9GR, "DSP — An Intuitive Approach," *QST*, Feb 1996.
Hightower, M., KF6SJ, "Simple SDR Receiver," *QEX*, Mar/Apr 2012, pp 3-8.
Hill, C., W5BAA, "SDR2GO: A DSP Odyssey," *QEX*, Mar/Apr 2011, pp 36-43.
Kossor, M., "A Digital Commutating Filter," *QEX*, May/Jun 1999, pp 3-8.
Larkin, B., W7PUA, "The DSP-10: An All-Mode 2-Meter Transceiver Using a DSP IF and PC-Controlled Front Panel, Part 1," *QST*, Sep 1999, pp 33-41.
Larkin, B., W7PUA, "The DSP-10: An All-Mode 2-Meter Transceiver Using a DSP IF and PC-Controlled Front Panel, Part 2," *QST*, Oct 1999, pp 34-40.
Larkin, B., W7PUA, "The DSP-10: An All-Mode 2-Meter Transceiver Using a DSP IF and PC-Controlled Front Panel, Part 3," *QST*, Nov 1999, pp 42-45.
Mack, Ray, W5IFS, "SDR: Simplified," *QEX*, columns beginning Nov 2009.
Morrison, F., "The Magic of Digital Filters," *QEX*, Feb 1993, pp 3-8.
Nickels, R., W9RAN, "Cheap and Easy SDR," *QST*, Jan 2013, pp 30-35.
Olsen, R., "Digital Signal Processing for the Experimenter," *QST*, Nov 1994, pp 22-27.
Reyer, S. and Herschberger, D., "Using the LMS Algorithm for QRM and QRN Reduction," *QEX*, Sep 1992, pp 3-8.
Rohde, D., "A Low-Distortion Receiver Front End for Direct-Conversion and DSP Receivers," *QEX*, Mar/Apr 1999, pp 30-33.
Runge, C., *Z. Math. Physik*, Vol 48, 1903; also Vol 53, 1905.
Scarlett, J., "A High-Performance Digital Transceiver Design," *QEX*, Part 1, Jul/Aug 2002; Part 2, Mar/Apr 2003.
Smith, D., "Introduction to Adaptive Beamforming," *QEX*, Nov/Dec 2000.
Smith, D., "Signals, Samples and Stuff: A DSP Tutorial, Parts 1-4," *QEX*, Mar/Apr-Sep/Oct, 1998.
Stephensen, J., KD6OZH, Software Defined Radios for Digital Communications," *QEX*, Nov/Dec 2004, pp 23-35. [Open-source platform]
Veatch, J., WA2EUJ, "The DSP-610 Transceiver," *QST*, Aug 2012, pp 30-32.
Ward, R., "Basic Digital Filters," *QEX*, Aug 1993, pp 7-8.
Wiseman, J., KE3QG, "A Complete DSP Design Example Using FIR Filters," *QEX*, Jul 1996.
Youngblood, G., "A Software-Defined Radio for the Masses," *QEX*; Part 1, Jul/Aug 2002; Part 2, Sep/Oct 2002; Part 3, Nov/Dec 2002; Part 4, Mar/Apr 2003.

SDR and DSP Receiver Testing

Hakanson, E., "Understanding SDRs and their RF Test Requirements," Anritsu Application Note. Available online from **www.us.anritsu.com/downloads/files/11410-00403.pdf**.
Kundert, K. "Accurate and Rapid Measurement of IP2 and IP3," Designer's Guide Consulting, 2002-2006. Available online from **www.designers-guide.org/Analysis/intercept-point.pdf**.
MacLeod, J.R.; Beach, M.A.; Warr, P.A.; Nesimoglu, T., "Software Defined Radio Receiver Test-Bed," IEEE Vehicular Tech Conf, Fall 2001, VTS 54th, Vol 3, pp 565-1569. Available online from **http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/7588/20685/00956461.pdf**.
R. Sierra, Rhode & Schwarz, "Challenges In Testing Software Defined Radios," SDR Forum Sandiego Workshop, 2007.
Sirmans, D. and Urell, B., "Digital Receiver Test Results," Next Generation Weather Radar Program. Available online from **www.roc.noaa.gov/eng/docs/digital%20receiver%20test%20results.pdf**.
"Testing and Troubleshooting Digital RF Communications Receiver Designs," Agilent Technologies Application Note 1314. Available online from **http://cp.literature.agilent.com/litweb/pdf/5968-3579E.pdf**.