

# The Scientist and Engineer's Guide to Digital Signal Processing

By Steven W. Smith, Ph.D.

Book Search:

[Home](#)[The Book by Chapters](#)[About the Book](#)[Steven W. Smith](#)[Blog](#)[Contact](#)[Chapter 28 - Digital Signal Processors](#) / Fixed versus Floating Point

## Chapter 28: Digital Signal Processors

### Fixed versus Floating Point

Digital Signal Processing can be divided into two categories, fixed point and floating point. These refer to the format used to store and manipulate numbers within the devices. Fixed point DSPs usually represent each number with a minimum of 16 bits, although a different length can be used. For instance, Motorola manufactures a family of fixed point DSPs that use 24 bits. There are four common ways that these  $2^{16} = 65536$  possible

bit patterns can represent a number. In unsigned integer, the stored number can take on any integer value from 0 to 65,535. Similarly, signed integer uses two's complement to make the range include negative numbers, from -32,768 to 32,767. With unsigned fraction notation, the 65,536 levels are spread uniformly between 0 and 1. Lastly, the signed fraction format allows negative numbers, equally spaced between -1 and 1.

In comparison, floating point DSPs typically use a minimum of 32 bits to store each value. This results in many more bit patterns than for fixed point,  $2^{32} = 4,294,967,296$  to be exact. A key feature of floating point notation is that the represented numbers are *not* uniformly spaced. In the most common format (ANSI/IEEE Std. 754-1985), the largest and smallest numbers are  $\pm 3.4 \times 10^{38}$  and  $1.2 \times 10^{-38}$ , respectively. The represented values are unequally spaced between these two extremes, such that the gap between any two numbers is about ten-million times smaller than the value of the numbers. This is important because it places large gaps between large numbers, but small gaps between small numbers. Floating point notation is discussed in more detail in Chapter 4.

All floating point DSPs can also handle fixed point numbers, a necessity to implement counters, loops, and signals coming from the ADC and going to the DAC. However, this doesn't mean that fixed point math will be carried out as quickly as the floating point operations; it depends on the internal architecture. For instance, the SHARC DSPs are optimized for both floating point and fixed point operations, and executes them with



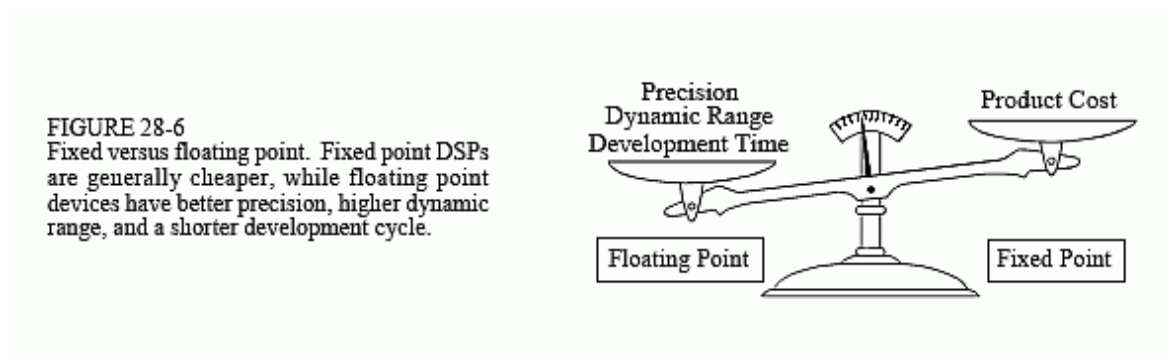
Download this chapter in PDF format

[Chapter28.pdf](#)



equal efficiency. For this reason, the SHARC devices are often referred to as "32-bit DSPs," rather than just "Floating Point."

Figure 28-6 illustrates the primary trade-offs between fixed and floating point DSPs. In Chapter 3 we stressed that fixed point arithmetic is much



faster than floating point in general purpose computers. However, with DSPs the speed is about the same, a result of the hardware being highly optimized for math operations. The internal architecture of a floating point DSP is more complicated than for a fixed point device. All the registers and data buses must be 32 bits wide instead of only 16; the multiplier and ALU must be able to quickly perform floating point arithmetic, the instruction set must be larger (so that they can handle both floating and fixed point numbers), and so on. Floating point (32 bit) has better precision and a higher dynamic range than fixed point (16 bit). In addition, floating point programs often have a shorter development cycle, since the programmer doesn't generally need to worry about issues such as overflow, underflow, and round-off error.

On the other hand, fixed point DSPs have traditionally been cheaper than floating point devices. Nothing changes more rapidly than the price of electronics; anything you find in a book will be out-of-date before it is printed. Nevertheless, cost is a key factor in understanding how DSPs are evolving, and we need to give you a general idea. When this book was completed in 1999, fixed point DSPs sold for between \$5 and \$100, while floating point devices were in the range of \$10 to \$300. This difference in cost can be viewed as a measure of the relative complexity between the devices. If you want to find out what the prices are *today*, you need to look *today*.

Now let's turn our attention to *performance*; what can a 32-bit floating point system do that a 16-bit fixed point can't? The answer to this question is signal-to-noise ratio. Suppose we store a number in a 32 bit floating point format. As previously mentioned, the gap between this number and its adjacent neighbor is about one ten-millionth of the value of the number. To store the number, it must be round up or down by a maximum of one-half the gap size. In other words, each time we store a number in floating point notation, we add *noise* to the signal.

The same thing happens when a number is stored as a 16-bit fixed point value, except that the added noise is much worse. This is because the gaps between adjacent numbers are much larger. For instance, suppose we store the number 10,000 as a signed integer (running from -32,768 to 32,767). The gap between numbers is one ten-thousandth of the value of the number we are storing. If we want to store the number 1000, the gap between numbers is only one one-thousandth of the value.

Noise in signals is usually represented by its *standard deviation*. This was discussed in

detail in Chapter 2. For here, the important fact is that the standard deviation of this quantization noise is about one-third of the gap size. This means that the signal-to-noise ratio for storing a floating point number is about 30 million to one, while for a fixed point number it is only about ten-thousand to one. In other words, floating point has roughly 30,000 times less quantization noise than fixed point.

This brings up an important way that DSPs are different from traditional microprocessors. Suppose we implement an FIR filter in fixed point. To do this, we loop through each coefficient, multiply it by the appropriate sample from the input signal, and add the product to an accumulator. Here's the problem. In traditional microprocessors, this accumulator is just another 16 bit fixed point variable. To avoid overflow, we need to scale the values being added, and will correspondingly add quantization noise on each step. In the worst case, this quantization noise will simply add, greatly lowering the signal-to-noise ratio of the system. For instance, in a 500 coefficient FIR filter, the noise on each output sample may be 500 times the noise on each input sample. The signal-to-noise ratio of *ten-thousand to one* has dropped to a ghastly *twenty to one*. Although this is an extreme case, it illustrates the main point: when many operations are carried out on each sample, it's bad, really bad. See Chapter 3 for more details.

DSPs handle this problem by using an extended precision accumulator. This is a special register that has 2-3 times as many bits as the other memory locations. For example, in a 16 bit DSP it may have 32 to 40 bits, while in the SHARC DSPs it contains 80 bits for fixed point use. This extended range virtually eliminates round-off noise while the accumulation is in progress. The only round-off error suffered is when the accumulator is scaled and stored in the 16 bit memory. This strategy works very well, although it does limit how some algorithms must be carried out. In comparison, floating point has such low quantization noise that these techniques are usually not necessary.

In addition to having lower quantization noise, floating point systems are also easier to develop algorithms for. Most DSP techniques are based on repeated multiplications and additions. In fixed point, the possibility of an overflow or underflow needs to be considered after each operation. The programmer needs to continually understand the amplitude of the numbers, how the quantization errors are accumulating, and what scaling needs to take place. In comparison, these issues do not arise in floating point; the numbers take care of themselves (except in rare cases).

To give you a better understanding of this issue, Fig. 28-7 shows a table from the SHARC user manual. This describes the ways that multiplication can be carried out for both fixed and floating point formats. First, look at how floating point numbers can be multiplied; there is only one way! That

Fixed Point	Floating Point
$\left. \begin{array}{l} R_n \\ MRF \\ MRB \end{array} \right  = R_x * R_y \quad \left( \begin{array}{c c c} S & S & F \\ \hline U & U & I \\ \hline FR \end{array} \right)$	$F_n = F_x * F_y$
$\left. \begin{array}{l} R_n \\ R_n \\ MRF \\ MRB \end{array} \right  = MRF \quad \left. \begin{array}{l} \\ \\ \\ \end{array} \right  + R_x * R_y \quad \left( \begin{array}{c c c} S & S & F \\ \hline U & U & I \\ \hline FR \end{array} \right)$	
$\left. \begin{array}{l} R_n \\ R_n \\ MRF \\ MRB \end{array} \right  = MRF \quad \left. \begin{array}{l} \\ \\ \\ \end{array} \right  - R_x * R_y \quad \left( \begin{array}{c c c} S & S & F \\ \hline U & U & I \\ \hline FR \end{array} \right)$	
$\left. \begin{array}{l} R_n \\ R_n \\ MRF \\ MRB \end{array} \right  = SAT \quad \left. \begin{array}{l} MRF \\ MRB \end{array} \right  \quad \left( \begin{array}{c} (SI) \\ (UI) \\ (SF) \\ (UF) \end{array} \right)$	
$\left. \begin{array}{l} R_n \\ R_n \\ MRF \\ MRB \end{array} \right  = RND \quad \left. \begin{array}{l} MRF \\ MRB \end{array} \right  \quad \left( \begin{array}{c} (SF) \\ (UF) \end{array} \right)$	
$\left. \begin{array}{l} MRF \\ MRB \end{array} \right  = 0$	
$\left. \begin{array}{l} MRxF \\ MRxB \end{array} \right  = R_n$	
$R_n = \left. \begin{array}{l} MRxF \\ MRxB \end{array} \right $	

FIGURE 28-7

Fixed versus floating point instructions. These are the multiplication instructions used in the SHARC DSPs. While only a single command is needed for floating point, many options are needed for fixed point. See the text for an explanation of these options.

is,  $F_n = F_x * F_y$ , where  $F_n$ ,  $F_x$ , and  $F_y$  are any of the 16 data registers. It could not be any simpler. In comparison, look at all the possible commands for fixed point multiplication. These are the many options needed to efficiently handle the problems of round-off, scaling, and format.

In Fig. 28-7,  $R_n$ ,  $R_x$ , and  $R_y$  refer to any of the 16 data registers, and  $MRF$  and  $MRB$  are 80 bit accumulators. The vertical lines indicate *options*. For instance, the top-left entry in this table means that all the following are valid commands:  $R_n = R_x * R_y$ ,  $MRF = R_x * R_y$ , and  $MRB = R_x * R_y$ . In other words, the value of any two registers can be multiplied and placed into another register, or into one of the extended precision accumulators. This table also shows that the numbers may be either signed or unsigned (S or U), and may be fractional or integer (F or I). The RND and SAT options are ways of controlling rounding and register overflow.

There are other details and options in the table, but they are not important for our present discussion. The important idea is that the fixed point programmer must understand *dozens* of ways to carry out the very basic task of multiplication. In contrast, the floating point programmer can spend his time concentrating on the algorithm.

Given these tradeoffs between fixed and floating point, how do you choose which to use? Here are some things to consider. First, look at how many bits are used in the ADC and DAC. In many applications, 12-14 bits per sample is the crossover for using

fixed versus floating point. For instance, television and other video signals typically use 8 bit ADC and DAC, and the precision of fixed point is acceptable. In comparison, professional audio applications can sample with as high as 20 or 24 bits, and almost certainly need floating point to capture the large dynamic range.

The next thing to look at is the complexity of the algorithm that will be run. If it is relatively simple, think fixed point; if it is more complicated, think floating point. For example, FIR filtering and other operations in the time domain only require a few dozen lines of code, making them suitable for fixed point. In contrast, frequency domain algorithms, such as spectral analysis and FFT convolution, are very detailed and can be much more difficult to program. While they can be written in fixed point, the development time will be greatly reduced if floating point is used.

Lastly, think about the money: how important is the cost of the product, and how important is the cost of the development? When fixed point is chosen, the cost of the product will be reduced, but the development cost will probably be higher due to the more difficult algorithms. In the reverse manner, floating point will generally result in a quicker and cheaper development cycle, but a more expensive final product.

Figure 28-8 shows some of the major trends in DSPs. Figure (a) illustrates the impact that Digital Signal Processors have had on the *embedded* market. These are applications that use a microprocessor to directly operate and control some larger system, such as a cellular telephone, microwave oven, or automotive instrument display panel. The name "microcontroller" is often used in referring to these devices, to distinguish them from the microprocessors used in personal computers. As shown in (a), about 38% of embedded designers have already started using DSPs, and another 49% are considering the switch. The high throughput and computational power of DSPs often makes them an ideal choice for embedded designs.

As illustrated in (b), about twice as many engineers currently use fixed point as use floating point DSPs. However, this depends greatly on the application. Fixed point is more popular in competitive consumer products where the cost of the electronics must be kept very low. A good example of this is cellular telephones. When you are in competition to sell millions of your product, a cost difference of only a few dollars can be the difference between success and failure. In comparison, floating point is more common when greater performance is needed and cost is not important. For

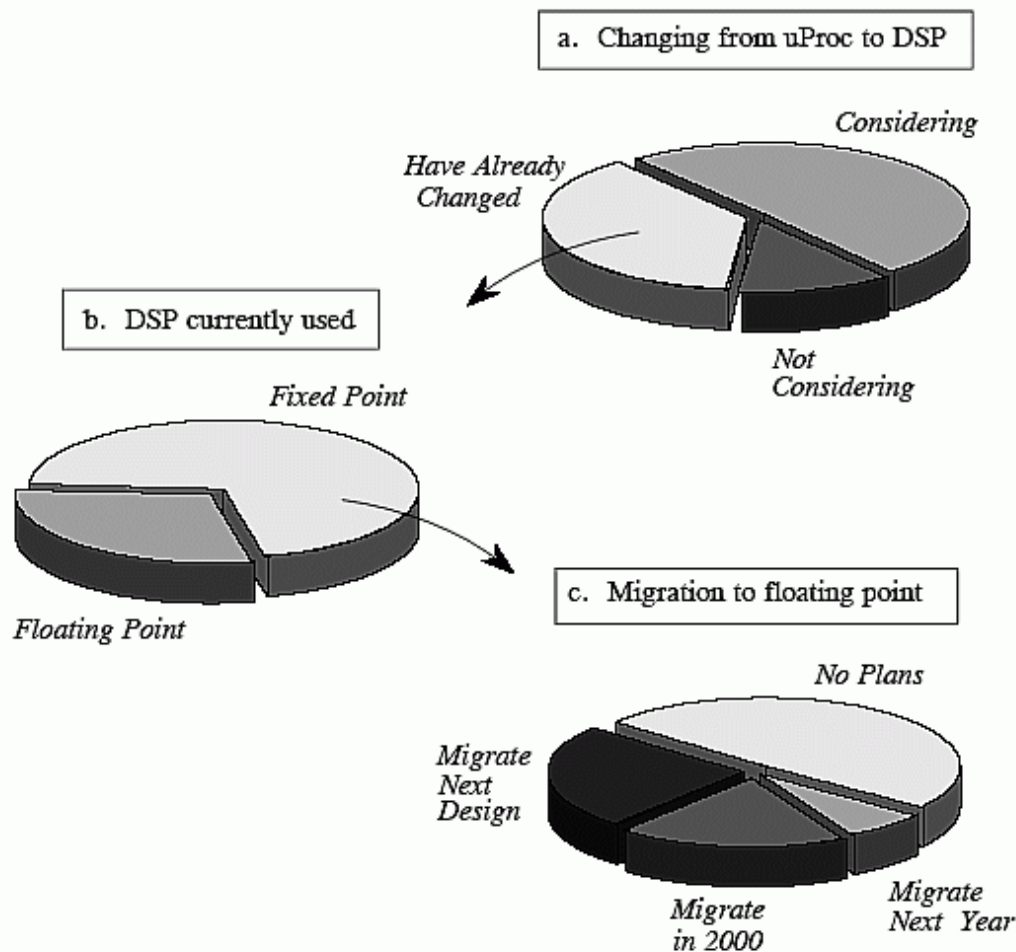


FIGURE 28-8

Major trends in DSPs. As illustrated in (a), about 38% of embedded designers have already switched from conventional microprocessors to DSPs, and another 49% are considering the change. In (b), about twice as many engineers use fixed point as use floating point DSPs. This is mainly driven by consumer products that must have low cost electronics, such as cellular telephones. However, as shown in (c), floating point is the fastest growing segment; over one-half of engineers currently using 16 bit devices plan to migrate to floating point DSPs

instance, suppose you are designing a medical imaging system, such a computed tomography scanner. Only a few hundred of the model will ever be sold, at a price of several hundred-thousand dollars each. For this application, the cost of the DSP is insignificant, but the performance is critical. In spite of the larger number of fixed point DSPs being used, the floating point market is the fastest growing segment. As shown in (c), over one-half of engineers using 16-bits devices plan to migrate to floating point at some time in the near future.

Before leaving this topic, we should reemphasize that floating point and fixed point usually use 32 bits and 16 bits, respectively, *but not always*. For instance, the SHARC family can represent numbers in 32-bit fixed point, a mode that is common in digital audio applications. This makes the  $2^{32}$  quantization levels spaced uniformly over a relatively small range, say, between -1 and 1. In comparison, floating point notation places the  $2^{32}$  quantization levels logarithmically over a huge range, typically  $\pm 3.4 \times 10^{38}$ . This gives 32-bit fixed point better *precision*, that is, the quantization error on any one sample will be lower. However, 32-bit floating point has a higher *dynamic range*, meaning there is a greater difference between the largest number and the

smallest number that can be represented.

Next Section: [C versus Assembly](#)

---

copyright 1997-2007 by California Technical Publishing