CHAPTER 4: DSP SOFTWARE

DSP applications are often programmed in the same languages as other science and engineering tasks such as C, BASIC, and assembly. C's power and versatility make it the preferred language for computer scientists and other professional programmers. BASIC's simplicity is ideal for scientists and engineers who only occasionally visit the programming world. Regardless of the language, most DSP software problems are buried far below in the realm of rotating ones and zeros. This includes topics such as how numbers are represented by bit patterns, rounding error in computer arithmetic, computing speed of different processor types. This section is about things you can do on a high level to avoid being chewed.

Computer Numbers

Digital computers are very capable of storing and recalling numbers; Unfortunately, this process is not error free. In some cases this error is pretty minor, in other cases it is catastrophic. Understanding how computers store and process numbers allows you to fix these problems before your program runs out of meaningless data. These problems arise because a fixed number of bits is allocated to store each number. In the simplest cases, 256-bit patterns are 0 to 255, 1 to 256, -127 to 128, etc. It can represent integers between. In a more unusual scheme, 256-bit patterns can represent 256 exponentially related numbers: 1, 10, 100, 1000,…, 10 254, 10 255. Anyone accessing the data must understand what value each bit pattern represents. While many encoding schemes are possible, only two general formats have become common, fixed point (also called integer numbers) and floating point (also called real numbers).

Fixed Point (Integers)

Fixed point representation is used to store integers, positive and negative integers. High-level programs such as C and BASIC usually reserve 16 bits to store each integer. In the simplest case, 2 16 = 65,536 possible bit patterns are assigned to numbers between 0 and 65,535. This is called unsigned integer format. The most important use of the offset pair is in ADC and DAC. Sign and magnitude are another simple way to represent negative integers. The leftmost bit is called the sign bit and is made a zero for positive numbers and a single for negative numbers. The other bits are a standard binary representation of the absolute value of the number. This results in a wasted bit pattern since there are two representations for zero, 0000 (positive zero) and 1000 (negative zero). The two's complement is the format loved by hardware engineers and the way integers are often represented on computers. Using 16 bits, the two's complement can represent numbers between -32,768 and 32,767. The leftmost bit is 0 if the number is positive or zero, and 1 if the number is negative. Consequently, the leftmost bit is called the sign bit, as in sign and magnitude representation. Converting between decimal and two's complement, simple for positive numbers, and for negative numbers, there is an algorithm: the decimal number's absolute value is taken and converted to binary. Then all bits are complemented and 1 is added.

Floating Point (Real Numbers)

The coding scheme for floating point numbers is more complex than for a fixed point. Floating point notation is similar to scientific notation, but everything is done on two bases rather than base ten. While several similar formats are in use, the most common is ANSI / IEEE Std. 754-1985. This standard defines the format of 32-bit numbers called single precision, as well as 64-bit numbers called double precision. The (-1) S term simply means that the sign bit S is 0 for a positive number and 1 for a negative number. The variable, E, is the number between 0 and 255 represented by eight exponent bits. Subtracting 127 from this number allows the exponential term to run so far. In other words, the exponent is stored in offset binary with an offset of 127. The manti M consists of 23 bits as a binary fraction. In other words, M = 1 + m 22 2 -1 + m 21 2 -2 + m 20 2 -3…. If bits 0 through 22 are all zero, then M takes the value of one. The IEEE standard for double precision adds more bits to a single precision format. Of the 64 bits used to store a double-precision number, bits 0 to 51 are mantiss, bits 52 to 62 are exponent, and 63 bits are sign bits.

Number Precision

Number representation errors are very similar to quantization errors during ADC. To keep the desired values in a continuous array; however, it can only represent a limited number of quantized levels. Some programming languages allow for a variable called a long integer that is stored as 32 bits, fixed point, and the sum of two. In fixed point variables, the spaces between adjacent numbers are always exactly one. In floating-point notation, the spaces between adjacent numbers vary within the range of numbers represented. If we choose a random floating-point number, the space next to that number is about ten million times smaller than the number itself, and the rounding error in each of the arithmetic operations causes the value X to gradually move away from its initial value. This deviation can take two forms depending on how the errors come together. If the rounding errors are randomly positive and negative, the value of the variable will randomly increase and decrease. If the errors predominantly have the same sign, the value of the variable will move away much faster and more uniformly. The additional error is roughly equal to the rounding error of a single transaction and multiplied by the total number of transactions. In comparison, random error only increases in proportion to the square root of the number of transactions. It is almost impossible to control or predict which of these two behaviors a given algorithm will experience.

Execution Speed: Programming Language

DSP programming loosely consists of three complexities: Assembly, Compiled, and Application Specific. Since it is difficult to work in a binary system, these ones and zeros are assigned names according to the function they perform. This level of programming is called assembly. Executable code can be run directly on the microprocessor. The next level of complexity can manipulate abstract variables without any reference to specific hardware. These are called compiled or high level languages. C, BASIC, FORTRAN, PASCAL, APL, COBOL, LISP. Like a dozen or so in common use. The compiler is used to convert high-level source code directly to machine code. This requires the compiler to assign hardware memory locations to each of the referenced abstract variables. The compiler breaks down complex mathematical expressions into more basic arithmetic. Microprocessors know only addition, subtraction, multiplication and division. Anything more complex should be done as a sequence of these basic operations. High-level languages separate the programmer from the hardware. This makes programming much easier and allows the source code to be moved between different microprocessors. Most importantly, the programmer using a compiled language does not need to know anything about the inner workings of the computer. Let's say you buy a newly developed DSP microprocessor to add to your existing project. These devices usually have many built-in features for DSP: analog inputs, analog outputs, digital I / O, anti-aliasing and reconstruction filters. Writing a routine in assembly is equivalent to a nearly two-year leap in hardware technology, as the speed of personal computers increases by about 40% each year. There are 3 reasons to program the DSP application in a language other than C: First, DSP has grown so fast that some organizations and individuals are stuck in other languages such as FORTRAN and PASCAL. Second, some applications require the highest efficiency achievable only with assembly programming. Third, C is not an easy language to learn, especially for part-time programmers.

Execution Speed: Hardware

The original IBM PC was introduced in 1981, based on an 8088 microprocessor with a 4.77 MHz clock speed and 8 bit bus. This was followed by a new generation of personal computers introduced every 3-4 years: 8088 → 80286 → 80386 → 80486 → 80586 (Pentium). Each of these new systems increased the speed of computing about five times compared to the previous technology. In 1996, the clock speed increased to 200 MHz and the bus to 32 bits. With other improvements, this has resulted in an increase in computing power of about a thousand in just 15 years. Since computers are made up of many subsystems, the time required to execute a given task will depend on two main factors: 1) the speed of the individual subsystems and 2) the time it takes to transfer data between these blocks. The Central Processing Unit (CPU) is the heart of the system and consists of up to a dozen registers, each capable of holding 32 bits. The CPU also includes digital electronics necessary for moving bits around and for basic operations such as fixed point arithmetic.

Complex math is handled by transferring data into a special hardware circuitry, also called the arithmetic logic unit or ALU, which is the math processor. The math coprocessor can be on the same chip as the CPU or it can be a separate electronic device. Most personal computer software can be used with or without a math coprocessor. This is accomplished by having the compiler generate machine code to handle both states, and all of this is stored in the last executable program. The speed with which data can be transferred between subsystems depends on the number of parallel data lines provided and the maximum speed of digital signals that can be transmitted along each line. Compared to data transfer between chips, digital data can often be transferred within a single chip at a much higher speed. A pipeline architecture divides the hardware required for a given task into several successive stages. The internal architecture used to achieve the increased speed includes: 1) a large number of very fast caches located on the chip, 2) separate buses for programs and data, allowing these two to be accessed simultaneously (called the Harvard Architecture), 3) located directly on the microprocessor quick hardware for math calculations and 4) a pipeline design.

Execution Speed: Programming Tips

While computer hardware and programming languages are important for maximizing execution speed, they are not something that changes daily. In contrast, how it is programmed can be changed at any time, and it greatly affects how long the program will run. 3 important suggestions can be given. First, integers are used instead of floating point variables whenever possible. Traditional microprocessors such as those used in personal computers process integers 10 to 20 times faster than floating point numbers. In systems without a math processor, the difference can be 200 to 1. An exception to this is integer division, which is usually performed by converting values to floating point. This makes the process very slow compared to other integer calculations. Second, avoid using complex operations. Third, find out what is fast and what is slow in the computer system. It comes with experience and tests, and there will always be surprises. Pay special attention to graphic commands and I / O. There are usually several ways to meet these requirements and speeds can vary widely. Functions are computed as a series of addition, subtraction, and multiplication.