

Data Converter Codes—Can You Decode Them?

by Walt Kester

INTRODUCTION

Analog-to-digital converters (ADCs) translate analog quantities, which are characteristic of most phenomena in the "real world," to digital language, used in information processing, computing, data transmission, and control systems. Digital-to-analog converters (DACs) are used in transforming transmitted or stored data, or the results of digital processing, back to "real-world" variables for control, information display, or further analog processing. The relationships between inputs and outputs of DACs and ADCs are shown in Figure 1.

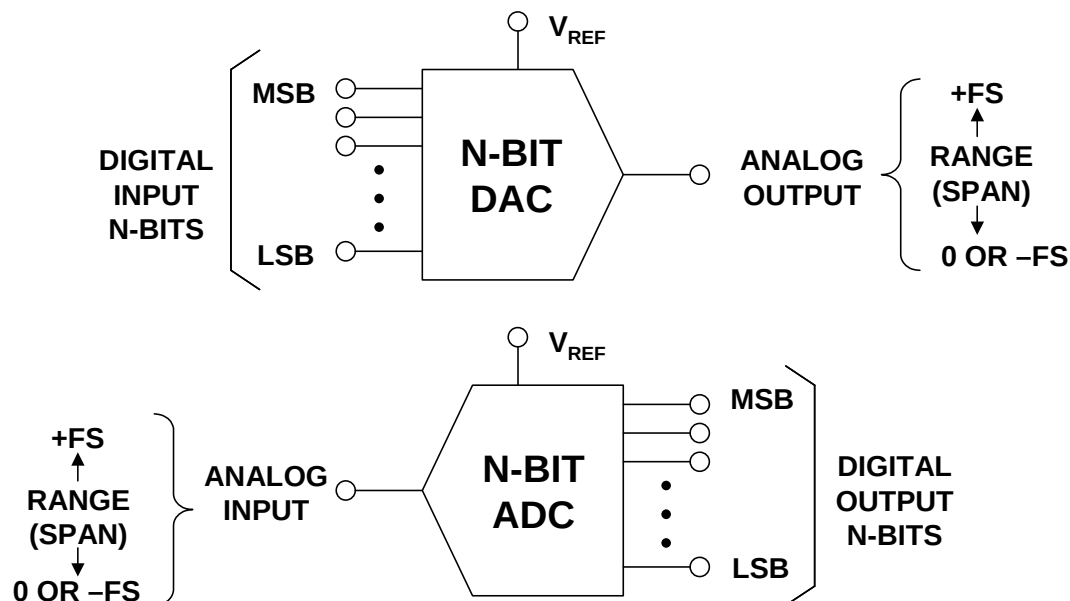


Figure 1: Digital-to-Analog Converter (DAC) and Analog-to-Digital Converter (ADC) Input and Output Definitions

Analog input variables, whatever their origin, are most frequently converted by transducers into voltages or currents. These electrical quantities may appear (1) as fast or slow "dc" continuous direct measurements of a phenomenon in the time domain, (2) as modulated ac waveforms (using a wide variety of modulation techniques), (3) or in some combination, with a spatial configuration of related variables to represent shaft angles. Examples of the first are outputs of thermocouples, potentiometers on dc references, and analog computing circuitry; of the second, "chopped" optical measurements, ac strain gage or bridge outputs, and digital signals buried in noise; and of the third, synchros and resolvers.

The analog variables to be dealt with in this article are those involving voltages or currents representing the actual analog phenomena. They may be either wideband or narrowband. They may be either scaled from the direct measurement, or subjected to some form of analog pre-processing, such as linearization, combination, demodulation, filtering, sample-hold, etc.

As part of the process, the voltages and currents are "normalized" to ranges compatible with assigned ADC input ranges. Analog output voltages or currents from DACs are direct and in normalized form, but they may be subsequently post-processed (e.g., scaled, filtered, amplified, etc.).

Information in digital form is normally represented by arbitrarily fixed voltage levels referred to "ground," either occurring at the outputs of logic gates, or applied to their inputs. The digital numbers used are all basically binary; that is, each "bit," or unit of information has one of two possible states. These states are "off," "false," or "0," and "on," "true," or "1." It is also possible to represent the two logic states by two different levels of current, however this is much less popular than using voltages. There is also no particular reason why the voltages need be referenced to ground—as in the case of emitter-coupled-logic (ECL), positive-emitter-coupled-logic (PECL) or low-voltage-differential-signaling logic (LVDS), for example.

Words are groups of levels representing digital numbers; the levels may appear simultaneously in *parallel*, on a bus or groups of gate inputs or outputs, *serially* (or in a time sequence) on a single line, or as a sequence of parallel bytes (i.e., "byte-serial") or nibbles (small bytes). For example, a 16-bit word may occupy the 16 bits of a 16-bit bus, or it may be divided into two sequential bytes for an 8-bit bus, or four 4-bit nibbles for a 4-bit bus.

Although there are several systems of logic, the most widely used choice of levels are those used in TTL (transistor-transistor logic) and, in which positive *true*, or 1, corresponds to a minimum output level of +2.4 V (inputs respond unequivocally to "1" for levels greater than 2.0 V); and *false*, or 0, corresponds to a maximum output level of +0.4 V (inputs respond unequivocally to "0" for anything less than +0.8 V). It should be noted that even though CMOS is more popular today than TTL, CMOS logic levels are generally made to be compatible with the older TTL logic standard.

A unique parallel or serial grouping of digital levels, or a *number*, or *code*, is assigned to each analog level which is quantized (i.e., represents a unique portion of the analog range). A typical digital code would be this array:

$$a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0 = 1 0 1 1 1 0 0 1$$

It is composed of eight bits. The "1" at the extreme left is called the "most significant bit" (MSB, or Bit 1), and the one at the right is called the "least significant bit" (LSB, or bit *N*: 8 in this case). The meaning of the code, as either a number, a character, or a representation of an analog variable, is unknown until the *code* and the *conversion relationship* have been defined. It is important not to confuse the designation of a particular bit (i.e., Bit 1, Bit 2, etc.) with the subscripts associated with the "a" array. The subscripts correspond to the power of 2 associated with the weight of a particular bit in the sequence.

The best-known code (other than base 10) is *natural or straight binary* (base 2). Binary codes are most familiar in representing integers; i.e., in a natural binary integer code having N bits, the LSB has a weight of 2^0 (i.e., 1), the next bit has a weight of 2^1 (i.e., 2), and so on up to the MSB, which has a weight of 2^{N-1} (i.e., $2^N/2$). The value of a binary number is obtained by adding up the weights of all non-zero bits. When the weighted bits are added up, they form a unique number having any value from 0 to $2^N - 1$. Each additional trailing zero bit, if present, essentially doubles the size of the number.

In converter technology, full-scale (abbreviated *FS*) is independent of the number of bits of resolution, N . A more useful coding is *fractional binary* which is always normalized to full-scale. Integer binary can be interpreted as fractional binary if all integer values are divided by 2^N . For example, the MSB has a weight of $\frac{1}{2}$ (i.e., $2^{(N-1)}/2^N = 2^{-1}$), the next bit has a weight of $\frac{1}{4}$ (i.e., 2^{-2}), and so forth down to the LSB, which has a weight of $1/2^N$ (i.e., 2^{-N}). When the weighted bits are added up, they form a number with any of 2^N values, from 0 to $(1 - 2^{-N})$ of full-scale. Additional bits simply provide more fine structure without affecting full-scale range. The relationship between base-10 numbers and binary numbers (base 2) are shown in Figure 2 along with examples of each.

WHOLE NUMBERS:

$$\text{Number}_{10} = a_{N-1}2^{N-1} + a_{N-2}2^{N-2} + \dots + a_12^1 + a_02^0$$

$\uparrow$
MSB

$\uparrow$
LSB

$$\begin{aligned} \text{Example: } 1011_2 &= (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\ &= 8 + 0 + 2 + 1 = 11_{10} \end{aligned}$$

FRACTIONAL NUMBERS:

$$\text{Number}_{10} = a_{N-1}2^{-1} + a_{N-2}2^{-2} + \dots + a_12^{-(N-1)} + a_02^{-N}$$

$\uparrow$
MSB

$\uparrow$
LSB

$$\begin{aligned} \text{Example: } 0.1011_2 &= (1 \times 0.5) + (0 \times 0.25) + (1 \times 0.125) + (1 \times 0.0625) \\ &= 0.5 + 0 + 0.125 + 0.0625 = 0.6875_{10} \end{aligned}$$

Figure 2: Representing a Base-10 Number with a Binary Number (Base 2)

UNIPOLAR CODES

In data conversion systems, the coding method must be related to the analog input range (or span) of an ADC or the analog output range (or span) of a DAC. The simplest case is when the input to the ADC or the output of the DAC is always a unipolar positive voltage (current outputs are very popular for DAC outputs, much less for ADC inputs). The most popular code for this type of signal is *straight binary* and is shown in Figure 3 for a 4-bit converter. Notice that there are 16 distinct possible levels, ranging from the all-zeros code 0000, to the all-ones code 1111. It is important to note that the analog value represented by the all-ones code is not full-scale

(abbreviated FS), but $FS - 1 \text{ LSB}$. This is a common convention in data conversion notation and applies to both ADCs and DACs. Figure 3 gives the base-10 equivalent number, the value of the base-2 binary code relative to full-scale (FS), and also the corresponding voltage level for each code (assuming a +10 V full-scale converter. The Gray code equivalent is also shown, and will be discussed shortly.

BASE 10 NUMBER	SCALE	+10V FS	BINARY	GRAY
+15	+FS – 1LSB = +15/16 FS	9.375	1 1 1 1	1 0 0 0
+14	+7/8 FS	8.750	1 1 1 0	1 0 0 1
+13	+13/16 FS	8.125	1 1 0 1	1 0 1 1
+12	+3/4 FS	7.500	1 1 0 0	1 0 1 0
+11	+11/16 FS	6.875	1 0 1 1	1 1 1 0
+10	+5/8 FS	6.250	1 0 1 0	1 1 1 1
+9	+9/16 FS	5.625	1 0 0 1	1 1 0 1
+8	+1/2 FS	5.000	1 0 0 0	1 1 0 0
+7	+7/16 FS	4.375	0 1 1 1	0 1 0 0
+6	+3/8 FS	3.750	0 1 1 0	0 1 0 1
+5	+5/16 FS	3.125	0 1 0 1	0 1 1 1
+4	+1/4 FS	2.500	0 1 0 0	0 1 1 0
+3	+3/16 FS	1.875	0 0 1 1	0 0 1 0
+2	+1/8 FS	1.250	0 0 1 0	0 0 1 1
+1	1LSB = +1/16 FS	0.625	0 0 0 1	0 0 0 1
0	0	0.000	0 0 0 0	0 0 0 0

Figure 3: Unipolar Binary Codes, 4-bit Converter

Figure 4 shows the transfer function for an ideal 3-bit DAC with straight binary input coding. Notice that the analog output is zero for the all-zeros input code. As the digital input code increases, the analog output increases 1 LSB (1/8 scale in this example) per code. The most positive output voltage is 7/8 FS, corresponding to a value equal to $FS - 1 \text{ LSB}$. The mid-scale output of 1/2 FS is generated when the digital input code is 100.

The transfer function of an ideal 3-bit ADC is shown in Figure 5. There is a range of analog input voltage over which the ADC will produce a given output code; this range is the *quantization uncertainty* and is equal to 1 LSB. Note that the width of the transition regions between adjacent codes is zero for an ideal ADC. In practice, however, there is always transition noise associated with these levels, and therefore the width is non-zero. It is customary to define the analog input corresponding to a given code by the *code center* which lies halfway between two adjacent transition regions (illustrated by the black dots in the diagram). This requires that the first transition region occur at $\frac{1}{2} \text{ LSB}$. The full-scale analog input voltage is defined by 7/8 FS, ($FS - 1 \text{ LSB}$).

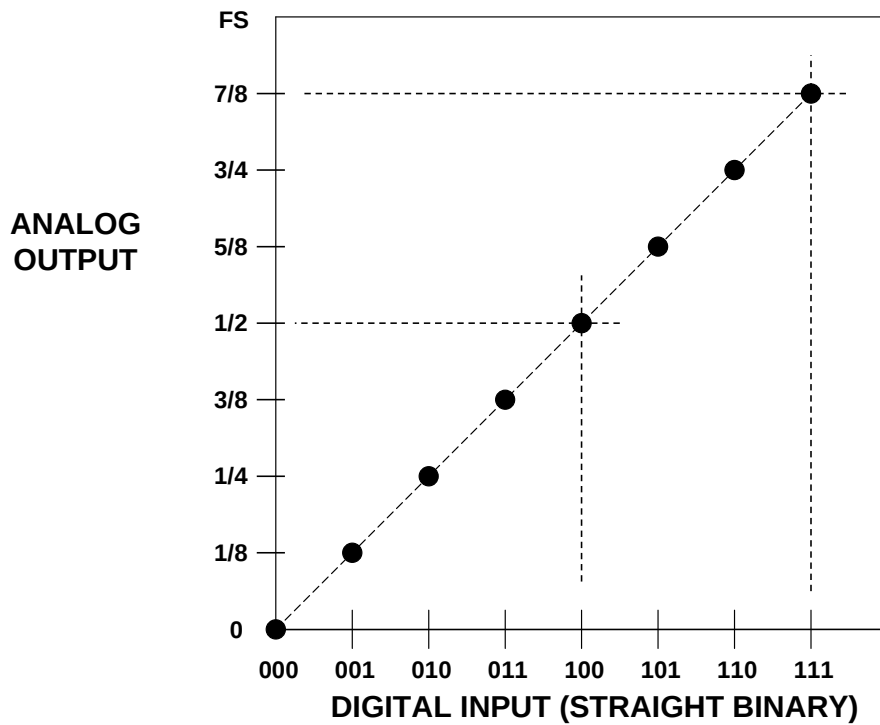


Figure 4: Transfer Function for Ideal Unipolar 3-bit DAC

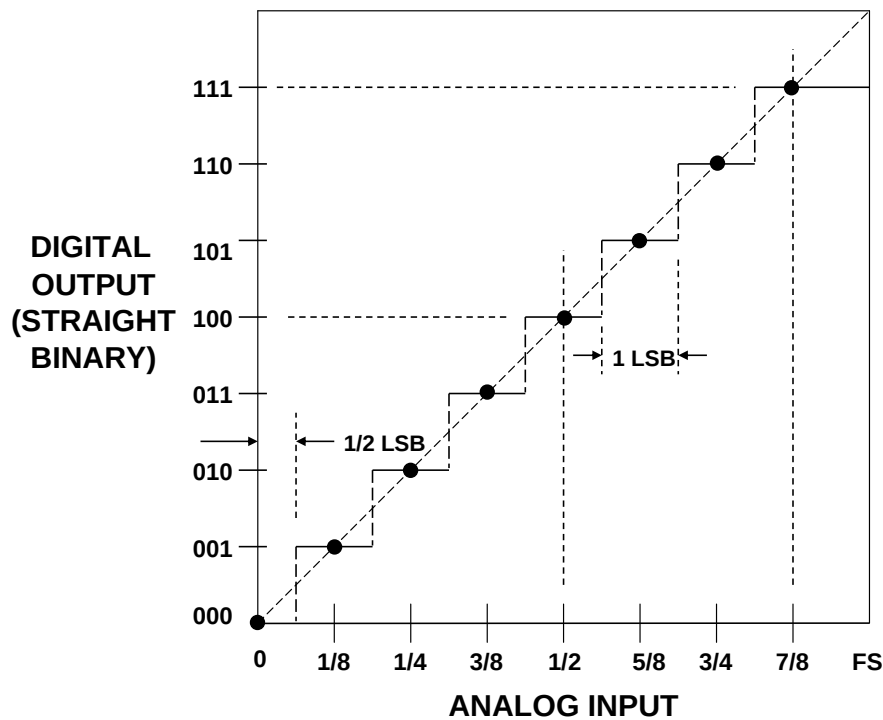


Figure 5: Transfer Function for Ideal Unipolar 3-bit ADC

GRAY CODE

Another code worth mentioning at this point is the *Gray code* (or *reflective-binary*) which was invented by Elisha Gray in 1878 (Reference 1) and later re-invented by Frank Gray in 1949 (see Reference 2). The Gray code equivalent of the 4-bit straight binary code is also shown in the last column of Figure 3. Although it is rarely used in computer arithmetic, it has some useful properties which make it attractive to A/D conversion. Notice that in Gray code, as the number value changes, the transitions from one code to the next involve only one bit at a time. Contrast this to the binary code where all the bits change when making the transition between 0111 and 1000. Some ADCs make use of it internally and then convert the Gray code to a binary code for external use.

As mentioned above, the Gray code has the property that adjacent levels differ by only one digit in the corresponding Gray-coded word. Therefore, if there is an error in a bit decision for a particular level, the corresponding error after conversion to binary code is only one least significant bit (LSB). In the case of mid-scale, note that only the MSB changes. It is interesting to note that this same phenomenon can occur in modern comparator-based flash converters due to comparator metastability. With small overdrive, there is a finite probability that the output of a comparator will generate the wrong decision in its latched output, producing the same effect if straight binary decoding techniques are used. In many cases, Gray code, or "pseudo-Gray" codes are used to decode the comparator bank. The Gray code output is then latched, converted to binary, and latched again at the final output.

Other examples where Gray code is often used in the conversion process to minimize errors are shaft encoders (angle-to-digital) and optical encoders.

ADCs which use the Gray code internally almost always convert the Gray code output to binary for external use. The conversion from Gray-to-binary and binary-to-Gray is easily accomplished with the exclusive-or logic function as shown in Figure 6.

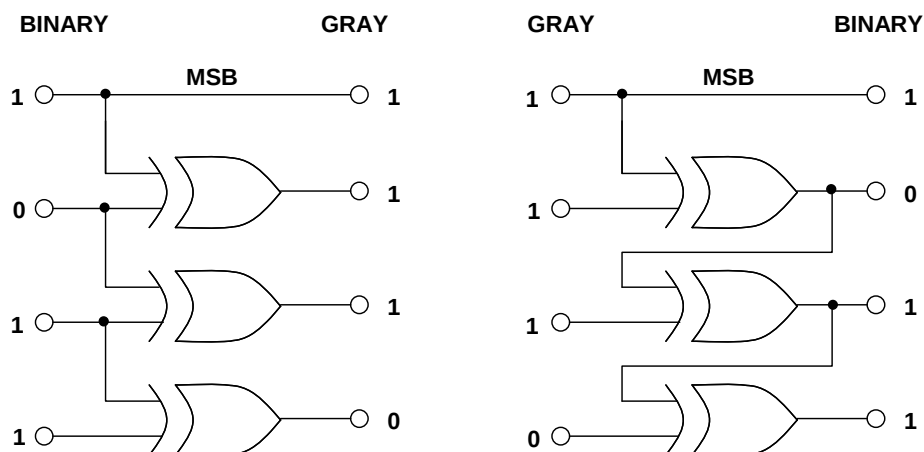


Figure 6: Binary-to-Gray and Gray-to-Binary Conversion Using the Exclusive-Or Logic Function

BIPOLAR CODES

In many systems, it is desirable to represent both positive and negative analog quantities with binary codes. Either *offset binary*, *twos complement*, *ones complement*, or *sign magnitude* codes will accomplish this, but offset binary and twos complement are by far the most popular. The relationships between these codes for a 4-bit systems is shown in Figure 7. Note that the values are scaled for a $\pm 5\text{-V}$ full-scale input/output voltage range.

BASE 10 NUMBER	SCALE	$\pm 5\text{V FS}$	OFFSET BINARY	TWOS COMP.	ONES COMP.	SIGN MAG.
+7	$+FS - 1\text{LSB} = +7/8 \text{ FS}$	+4.375	1 1 1 1	0 1 1 1	0 1 1 1	0 1 1 1
+6	$+3/4 \text{ FS}$	+3.750	1 1 1 0	0 1 1 0	0 1 1 0	0 1 1 0
+5	$+5/8 \text{ FS}$	+3.125	1 1 0 1	0 1 0 1	0 1 0 1	0 1 0 1
+4	$+1/2 \text{ FS}$	+2.500	1 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0
+3	$+3/8 \text{ FS}$	+1.875	1 0 1 1	0 0 1 1	0 0 1 1	0 0 1 1
+2	$+1/4 \text{ FS}$	+1.250	1 0 1 0	0 0 1 0	0 0 1 0	0 0 1 0
+1	$+1/8 \text{ FS}$	+0.625	1 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1
0	0	0.000	1 0 0 0	0 0 0 0	*0 0 0 0	*1 0 0 0
-1	$-1/8 \text{ FS}$	-0.625	0 1 1 1	1 1 1 1	1 1 1 0	1 0 0 1
-2	$-1/4 \text{ FS}$	-1.250	0 1 1 0	1 1 1 0	1 1 0 1	1 0 1 0
-3	$-3/8 \text{ FS}$	-1.875	0 1 0 1	1 1 0 1	1 1 0 0	1 0 1 1
-4	$-1/2 \text{ FS}$	-2.500	0 1 0 0	1 1 0 0	1 0 1 1	1 1 0 0
-5	$-5/8 \text{ FS}$	-3.125	0 0 1 1	1 0 1 1	1 0 1 0	1 1 0 1
-6	$-3/4 \text{ FS}$	-3.750	0 0 1 0	1 0 1 0	1 0 0 1	1 1 1 0
-7	$-FS + 1\text{LSB} = -7/8 \text{ FS}$	-4.375	0 0 0 1	1 0 0 1	1 0 0 0	1 1 1 1
-8	$-FS$	-5.000	0 0 0 0	1 0 0 0		

	ONES COMP.	SIGN MAG.
* 0+	0 0 0 0	0 0 0 0
* 0-	1 1 1 1	1 0 0 0

NOT NORMALLY USED
IN COMPUTATIONS (SEE TEXT)

Figure 7: Bipolar Codes, 4-bit Converter

For *offset binary*, the zero signal value is assigned the code 1000. The sequence of codes is identical to that of straight binary. The only difference between a straight and offset binary system is the half-scale offset associated with analog signal. The most negative value ($-FS + 1 \text{ LSB}$) is assigned the code 0001, and the most positive value ($+FS - 1 \text{ LSB}$) is assigned the code 1111. Note that in order to maintain perfect symmetry about mid-scale, the all-zeros code (0000) representing negative full-scale ($-FS$) is not normally used in computation. It can be used to represent a negative off-range condition or simply assigned the value of the 0001 ($-FS + 1 \text{ LSB}$).

The relationship between the offset binary code and the analog output range of a bipolar 3-bit DAC is shown in Figure 8. The analog output of the DAC is zero for the zero-value input code 100. The most negative output voltage is generally defined by the 001 code ($-FS + 1 \text{ LSB}$), and the most positive by 111 ($+FS - 1 \text{ LSB}$). The output voltage for the 000 input code is available for use if desired, but makes the output non-symmetrical about zero and complicates the mathematics.

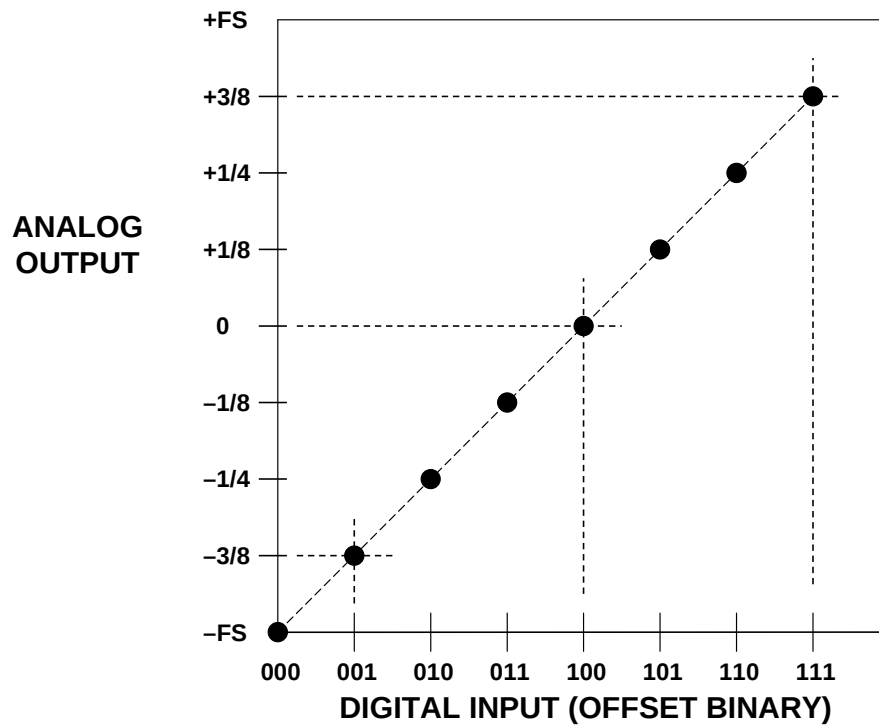


Figure 8: Transfer Function for Ideal Bipolar 3-bit DAC

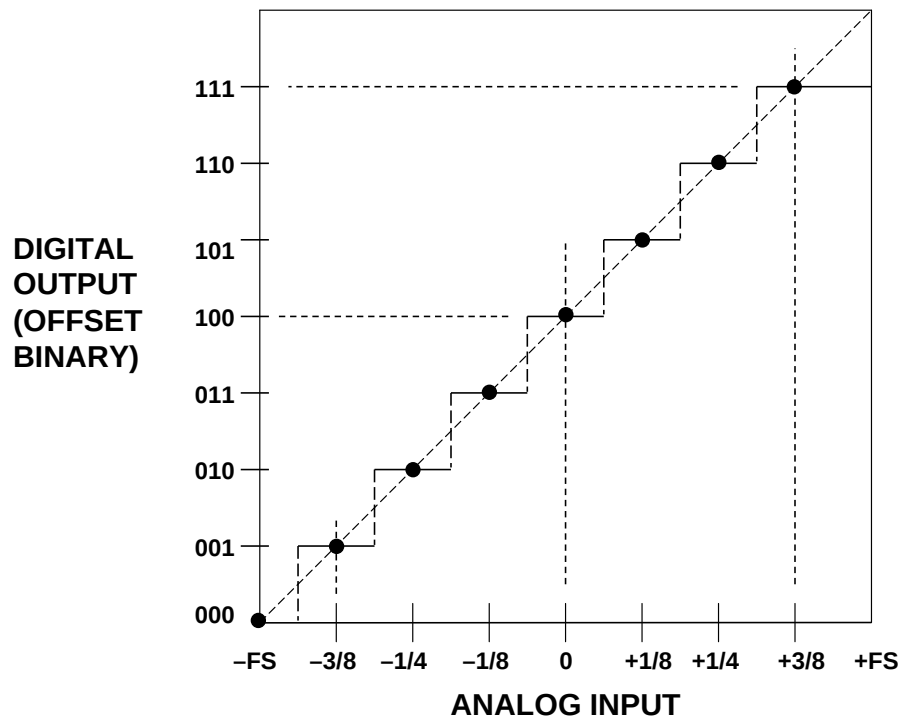


Figure 9: Transfer Function for Ideal Bipolar 3-bit ADC

The offset binary output code for a bipolar 3-bit ADC as a function of its analog input is shown in Figure 9. Note that zero analog input defines the center of the mid-scale code 100. As in the case of bipolar DACs, the most negative input voltage is generally defined by the 001 code ($-FS + 1 \text{ LSB}$), and the most positive by 111 ($+FS - 1 \text{ LSB}$). As discussed above, the 000 output code is available for use if desired, but makes the output non-symmetrical about zero and complicates the mathematics.

Twos complement is identical to offset binary with the most-significant-bit (MSB) complemented (inverted). This is obviously very easy to accomplish in a data converter, using a simple inverter or taking the complementary output of a "D" flip-flop. The popularity of twos complement coding lies in the ease with which mathematical operations can be performed in computers and DSPs. Twos complement, for conversion purposes, consists of a binary code for positive magnitudes (0 sign bit), and the twos complement of each positive number to represent its negative. The twos complement is formed arithmetically by complementing the number and adding 1 LSB. For example, $-3/8 \text{ FS}$ is obtained by taking the twos complement of $+3/8 \text{ FS}$. This is done by first complementing $+3/8 \text{ FS}$, 0011 obtaining 1100. Adding 1 LSB, we obtain 1101.

Twos complement makes subtraction easy. For example, to subtract $3/8 \text{ FS}$ from $4/8 \text{ FS}$, add $4/8$ to $-3/8$, or 0100 to 1101. The result is 0001, or $1/8$, disregarding the extra carry.

Ones complement can also be used to represent negative numbers, although it is much less popular than twos complement and rarely used today. The ones complement is obtained by simply complementing all of a positive number's digits. For instance, the ones complement of $3/8 \text{ FS}$ (0011) is 1100. A ones complemented code can be formed by complementing each positive value to obtain its corresponding negative value. This includes zero, which is then represented by either of two codes, 0000 (referred to as 0+) or 1111 (referred to as 0-). This ambiguity must be dealt with mathematically, and presents obvious problems relating to ADCs and DACs for which there is a single code which represents zero.

Sign-magnitude would appear to be the most straightforward way of expressing signed analog quantities digitally. Simply determine the code appropriate for the magnitude and add a polarity bit. Sign-magnitude BCD is popular in bipolar digital voltmeters, but has the problem of two allowable codes for zero. It is therefore unpopular for most applications involving ADCs or DACs.

Figure 10 summarizes the relationships between the various bipolar codes: offset binary, twos complement, ones complement, and sign-magnitude and shows how to convert between them.

To Convert From To	To			
	Sign Magnitude	2's Complement	Offset Binary	1's Complement
Sign Magnitude	No Change	If MSB = 1, complement other bits, add 00...01	Complement MSB If new MSB = 1, complement other bits, add 00...01	If MSB = 1, complement other bits
2's Complement	If MSB = 1, complement other bits, add 00...01	No Change	Complement MSB	If MSB = 1, add 00...01
Offset binary	Complement MSB If new MSB = 0 complement other bits, add 00...01	Complement MSB	No Change	Complement MSB If new MSB = 0, add 00...01
1's Complement	If MSB = 1, complement other bits	If MSB = 1, add 11...11	Complement MSB If new MSB = 1, add 11...11	No Change

Figure 10: Relationships Among Bipolar Codes

The last code to be considered in this section is *binary-coded-decimal (BCD)*, where each base-10 digit (0 to 9) in a decimal number is represented as the corresponding 4-bit straight binary word as shown in Figure 11. The minimum digit 0 is represented as 0000, and the digit 9 by 1001. This code is relatively inefficient, since only 10 of the 16 code states for each decade are used. It is, however, a very useful code for interfacing to decimal displays such as in digital voltmeters.

BASE 10 NUMBER	SCALE	+10V FS	DECADE 1	DECADE 2	DECADE 3	DECADE 4
+15	+FS – 1LSB = +15/16 FS	9.375	1 0 0 1	0 0 1 1	0 1 1 1	0 1 0 1
+14	+7/8 FS	8.750	1 0 0 0	0 1 1 1	0 1 0 1	0 0 0 0
+13	+13/16 FS	8.125	1 0 0 0	0 0 0 1	0 0 1 0	0 1 0 1
+12	+3/4 FS	7.500	0 1 1 1	0 1 0 1	0 0 0 0	0 0 0 0
+11	+11/16 FS	6.875	0 1 1 0	1 0 0 0	0 1 1 1	0 1 0 1
+10	+5/8 FS	6.250	0 1 1 0	0 0 1 0	0 1 0 1	0 0 0 0
+9	+9/16 FS	5.625	0 1 0 1	0 1 1 0	0 0 1 0	0 1 0 1
+8	+1/2 FS	5.000	0 1 0 1	0 0 0 0	0 0 0 0	0 0 0 0
+7	+7/16 FS	4.375	0 1 0 0	0 0 1 1	0 1 1 1	0 1 0 1
+6	+3/8 FS	3.750	0 0 1 1	0 1 1 1	0 1 0 1	0 0 0 0
+5	+5/16 FS	3.125	0 0 1 1	0 0 0 1	0 0 1 0	0 1 0 1
+4	+1/4 FS	2.500	0 0 1 0	0 1 0 1	0 0 0 0	0 0 0 0
+3	+3/16 FS	1.875	0 0 0 1	1 0 0 0	0 1 1 1	0 1 0 1
+2	+1/8 FS	1.250	0 0 0 1	0 0 1 0	0 1 0 1	0 0 0 0
+1	1LSB = +1/16 FS	0.625	0 0 0 0	0 1 1 0	0 0 1 0	0 1 0 1
0	0	0.000	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

Figure 11: Binary Coded Decimal (BCD) Code

COMPLEMENTARY CODES

Some forms of data converters (for example, early DACs using monolithic NPN quad current switches), require standard codes such as natural binary or BCD, but with all bits represented by their complements. Such codes are called *complementary codes*. All the codes discussed thus far have complementary codes which can be obtained by this method. A *complementary* code should not be confused with a *ones complement* or a *twos complement* code.

In a 4-bit complementary-binary converter, 0 is represented by 1111, half-scale by 0111, and FS – 1 LSB by 0000. In practice, the complementary code can usually be obtained by using the complementary output of a register rather than the true output, since both are available.

Sometimes the complementary code is useful in inverting the analog output of a DAC. Today many DACs provide differential outputs which allow the polarity inversion to be accomplished without modifying the input code. Similarly, many ADCs provide differential logic outputs which can be used to accomplish the polarity inversion.

ACKNOWLEDGEMENT

The author would like to thank Dan Sheingold of Analog Devices for the permission to use material directly from his classic 1986 *Analog-Digital Conversion Handbook* (Reference 3).

REFERENCES:

1. K. W. Cattermole, *Principles of Pulse Code Modulation*, American Elsevier Publishing Company, Inc., 1969, New York NY, ISBN 444-19747-8. (An excellent tutorial and historical discussion of data conversion theory and practice, oriented towards PCM, but covers practically all aspects. This one is a must for anyone serious about data conversion!)
2. Frank Gray, "Pulse Code Communication," *U.S. Patent 2,632,058*, filed November 13, 1947, issued March 17, 1953. (detailed patent on the Gray code and its application to electron beam coders).
3. Dan Sheingold, *Analog-Digital Conversion Handbook, 3rd Edition*, Analog Devices and Prentice-Hall, 1986, ISBN-0-13-032848-0. (the defining and classic book on data conversion).

Copyright 2009, Analog Devices, Inc. All rights reserved. Analog Devices assumes no responsibility for customer product design or the use or application of customers' products or for any infringements of patents or rights of others which may result from Analog Devices assistance. All trademarks and logos are property of their respective holders. Information furnished by Analog Devices applications and development tools engineers is believed to be accurate and reliable, however no responsibility is assumed by Analog Devices regarding technical accuracy and topicality of the content provided in Analog Devices Tutorials.