Manuel Richey and Hossein Saiedian

Compressed Two's Complement Data Formats Provide Greater Dynamic Range and Improved Noise Performance

In this article, we present and analyze a new family of fixed-point data formats for signal processing applications. These formats represent compressed two's complement data formats where compression on the sign bit is undertaken on a sample by sample basis. The extra room provided by sign-bit compression is utilized to retain more bits of precision for each individual sample. Compressed two's complement data formats are shown to provide greater dynamic range and improved noise performance over traditional fixed-point data formats such as sign-magnitude, offset binary, and traditional two's complement. Traditional two's complement is shown to be a member of the compressed two's complement family where the compression factor (CF) is equal to one. The dynamic range of a compressed two's complement data format is shown to approach the dynamic range of a noncompressed data format raised to the power of the CF. Improved performance for digital signal processing (DSP) applications such as digital filters and transforms is presented for specific instances of this family.

INTRODUCTION

The binary bits that make up an information stream are typically not all of equal importance. This inequality has fostered numerous data compression techniques, many of which focus on removing the least important bits while maintaining the essence of the information stream. Data compression is usually applied to information files such as text and pictures to reduce the amount of space required for storage. It is applied to information streams such as video and audio to decrease the required bandwidth needed to send information from one point to another. In this article, we present another use for data compression that improves the dynamic range and noise performance of discrete time signals.

Two's complement is the data format typically used to represent and operate on digital samples in a fixed-point DSP system because it provides the same numeric precision as other fixed-point data formats but is easier to implement in digital hardware [2]. In a two's complement fixed-point data format, the decimal point is set at a fixed position relative to the individual bits. As numbers become smaller, the sign bit fills in the unused digits at the most significant positions in the binary format. The primary assumption for this format is that the smallest digits are the least important, and they are typically rounded to reduce algorithmic noise and fit in the binary word width of the chosen two's complement format.

FOR SIGNAL PROCESSING APPLICATIONS, THE LEAST IMPORTANT BITS IN A TWO'S COMPLEMENT NUMBER ARE THE LEADING ONES AND ZEROS, NOT THE LEAST SIGNIFICANT BITS.

For signal processing applications, the least important bits in a two's complement number are the leading ones and zeros, not the least significant bits (LSBs). A number only needs one sign bit. All of the other leading digits are used to identify the distance between the significant digits and the decimal point. These leading ones and zeros can be efficiently compressed and the resulting space can be used for additional numeric precision. Unlike file-based compression techniques, this compression is performed on a sample-by-sample basis.

Figure 1 provides an example. Suppose a set of fixed-point data values of 12 b in width [Figure 1(a)] must fit into an 8-b data format. Typically, the entire data set is scaled such that the largest number fills the entire eight most significant bits (MSBs) [automatic gain control (AGC)]. Then the entire data set is rounded from 12 b to 8 b as shown in Figure 1(b) for a typical value. The negative effect of this approach is that up to 1/2 LSB of noise has been injected into each data sample by rounding. Furthermore, some of the

(a)	A 12-b Fixed-Point Number	000000101001
(b)	Rounded to an 8-b Two's Complement Number	00000011
(c)	Ideal Compressed Two's Complement Number	00101001

[FIG1] **Data rounding from 12 b to 8 b of a typical fixed-point number (Red = sign bits, Boldface = retained data, Gray = potentially lost LSBs).**

Digital Object Identifier 10.1109/MSP.2011.942521
Date of publication: 1 November 2011

1053-5888/11/$26.00©2011IEEE

smallest signals may not be representable as they round to zero. This effectively decreases the dynamic range of the data set. For a compressed two's complement number, the number of sign bits is reduced and room is made to retain the bits that would have been rounded off as shown for an ideal case in Figure 1(c).

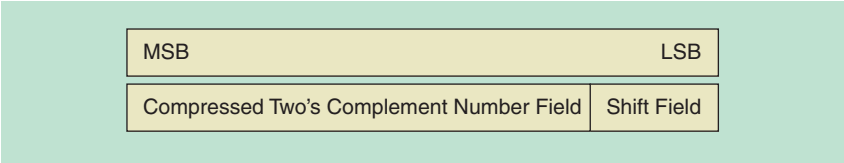
THE COMPRESSED TWO'S COMPLEMENT FORMAT

A compressed two's complement number has a predefined CF, and two data fields as shown in Figure 2 (CF is assumed).

Each member of this family of formats is identified with a different CF (one, two, three, etc.). The CF is assumed but not included in the individual bits of the data format. The CF determines how many bits each leading sign bit is to be expanded to for mathematical calculations. The shift field identifies how many digits to left shift the expanded number by. The compressed two's complement number is just a traditional two's complement number where each leading sign bit represents more than one leading sign bit when the number is expanded. The various widths of each of these fields constitute the different families in this class of data formats. The CF of a particular format generally determines the width of the shift field. The remaining bits constitute the compressed two's complement number field.

A CF of one yields a shift field of 0 b in width, and a standard two's complement number field of N b and results in a traditional fixed-point two's complement data format. A CF of two results in a shift field width of 1 b and a compressed two's complement number field of N-1 b. The minimum size of the shift field in bits is equal to the base two logarithm of the CF rounded up to the nearest integer. A CF of four results in a number field of N-2 b and a shift field of 2 b, and so forth.

An example may help. Let's illustrate a format with a CF of two and word length of five. The shift field for such a format is equal to $\log_2(\mathrm{CF})$, or 1 b in width. There are 4 b remaining for the compressed two's complement number field. Each leading sign bit is doubled, and if the shift bit is set to one, then the number is left shifted after being expanded. The resulting values are shown in Table 1.



[FIG2] Format of a compressed two's complement number.

With a CF of two, the 4 b two's complement number field is expanded into 8 b. This expansion obviously provides more dynamic range for the data format. What is not obvious at first glance is that a compressed data format also provides better numeric performance than a 5 b standard two's complement format would provide. Evidence of performance improvement is presented later in this article.

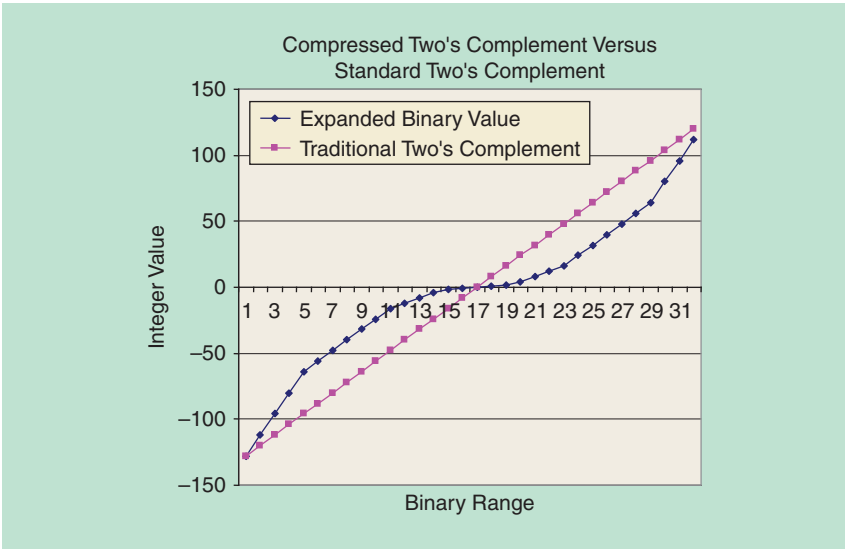
STEPS FOR COMPRESSION AND DECOMPRESSION

The steps followed for decompression with a CF of two are as follows:

[TABLE 1] A 5-b COMPRESSED TWO'S COMPLEMENT NUMBER WITH A COMPRESSION FACTOR OF TWO.

COMPRESSED BINARY VALUE (TWO'S COMPLEMENT FIELD—SHIFT FIELD)	EXPANDED BINARY VALUE	NUMERIC VALUE
0000 0	00000000	0
0000 1	00000001	1*
0001 0	00000010	2
0001 1	00000100	4
0010 0	00001000	8
0010 1	00010000	16
0011 0	00001100	12
0011 1	00011000	24
0100 0	00100000	32
0100 1	01000000	64
0101 0	00101000	40
0101 1	01010000	80
0110 0	00110000	48
0110 1	01100000	96
0111 0	00111000	56
0111 1	01110000	112
1000 0	11000000	−64
1000 1	10000000	−128
1001 0	11001000	−56
1001 1	10010000	−112
1010 0	11010000	−48
1010 1	10100000	−96
1011 0	11011000	−40
1011 1	10110000	−80
1100 0	11110000	−16
1100 1	11100000	−32
1101 0	11110100	−12
1101 1	11101000	−24
1110 0	11111100	−4
1110 1	11111000	−8
1111 0	11111111	−1
1111 1	11111110	−2

*0s are shifted into the LSB for all cases but this one.



[FIG3] Compressed two's complement versus standard two's complement for a 5-b number with a compression factor of two.

1) Double the number of leading sign bits.
2) Left justify the number into a field twice as wide as the compressed two's complement number field.
3) Pad the empty bits to the right with zeros.
4) If the shift field equals one and the compressed number is not equal to zero, then shift the number left by one bit, shifting a zero into the LSB.
5) If the compressed number is equal to zero and the shift bit is set to one, then shift a one into the LSB instead of a zero.

Similar steps are performed for formats with CFs other than two. For example, the first step for decompression of a number with a CF of four is to quadruple the number of leading sign bits. The two bit shift field of a CF = 4 format allows left shifts in Step 4 from zero to three.

The steps followed to compress a two's complement number with a CF of two are as follows:

1) Reduce the number of leading sign bits by a factor of two, rounding up for odd numbers (e.g., five leading sign bits is rounded up to three).
2) If the number of original leading sign bits is odd, then set the shift bit to one.
3) Round the resulting number to the word width of the compressed two's complement number field. (e.g., if one is compressing from 8 b to 4 b, round the result from Step 1 to 4 bits). The technique used for rounding is very important, but that subject is adequately treated elsewhere [1].

WHEN COMPRESSED TWO'S COMPLEMENT NUMBER FORMATS ARE USED ON LARGER WORD WIDTHS, TRULY IMPRESSIVE THINGS BEGIN TO HAPPEN.

4) If during the rounding process, the leading significant digit (i.e., nonsign bit) overflows into a new compressed sign bit, then the leading compressed sign bits and shift bit must be adjusted.

The data from Table 1 is plotted in Figure 3. This figure illustrates that greater precision is provided for the smallest numbers in a compressed two's complement data format. This is similar to what is accomplished with 8-b μ-law or A-law companding, and with floating-point data formatting. The main advantage compressed two's complement has over floating point is that the arithmetic is easier to accomplish in hardware, and that more precision is provided for the largest numbers in the format. This last point is not important for many problems, but is very important for DSP problems.

It turns out that for problems that require even distribution of precision, floating-point formats outperform compressed two's complement formats. DSP applications typically do not fall into this category because data is often represented as a fraction, and AGC techniques are often used to fit a problem's dynamic range into a numeric format's dynamic range. This arrangement typically gives fixed-point data formats an advantage in numeric precision over floating-point formats of equivalent word width. However, in this situation, compressed two's complement formats have an advantage over both fixed- and floating-point data formats in terms of numeric precision.

Another example will serve to hammer down this process. For a 5-b number with a CF of four, the shift field would be 2-b wide ($\log_2(4)$) and the compressed number field would be 3-b wide. During uncompression, each leading sign bit would be quadrupled, and right shifts of zero to three would place the remaining bits in their appropriate position.

The examples we have presented illustrate the process and use of compressed two's complement data formats. But we have restricted ourselves to using small word widths to improve the ease of illustration. When compressed two's complement number formats are used on larger word widths, truly impressive things begin to happen. For example, using a compressed two's complement data format with a CF of two and a 16-b word width, the dynamic range is doubled to approximately 180 dB and the round-off noise is reduced when compared with traditional two's complement arithmetic. A CF of three provides almost a tripling of dynamic range in bits (272 dB) together with improved round-off noise performance. This is illustrated in Figure 4.

IMPLEMENTATION DETAILS

It may appear at first glance that we have thrown out one of the major advantages of the two's complement

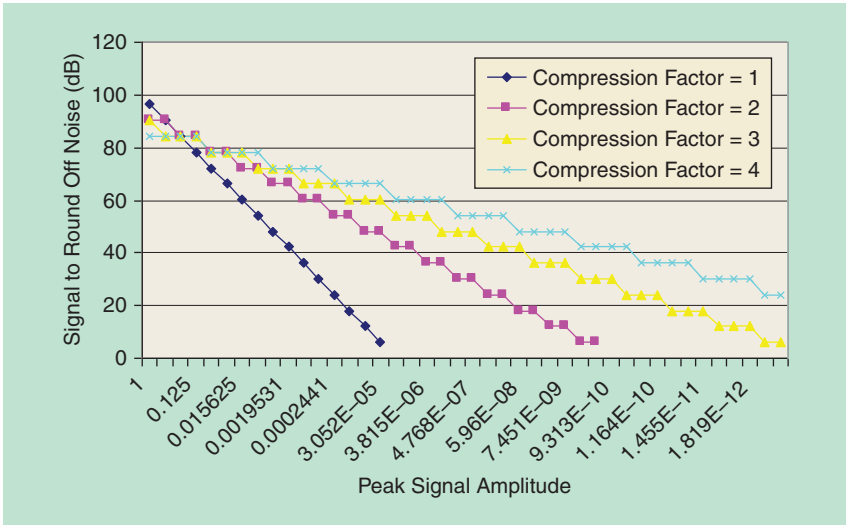
numbering systems by using a compressed format, that being the simplicity of the format. To some extent, this is true. However, we will endeavor to show that the sacrificed simplicity is not as great as first suspected. Furthermore, for many applications, the sacrificed simplicity in arithmetic implementation is more than made up for by increased simplicity in fixed-point signal processing algorithms.

Compressed two's complement algorithms can be implemented in either software or hardware. Software implementation is appropriate for low-speed applications and where data storage is emphasized over throughput. Examples of potential uses would be for stored audio on a compact disc or compressed speech. A software implementation of a compressed two's complement format with a CF of two is presented elsewhere in the form of a C++ class [3].

An appropriate hardware implementation would be within the arithmetic unit of a programmable digital signal processor integrated circuit. Such a circuit typically contains a multiply-accumulator circuit [5] for implementing digital filters. A decompression circuit should precede the traditional fixed-point arithmetic circuit where calculations are performed. When calculation is completed, data should flow through a compression circuit before being stored back into memory. The accumulator component of the arithmetic circuit would be larger than a traditional accumulator by the CF. For example, a 16-b data format with a CF of four would require a 64-b arithmetic logic unit (ALU). However, the multiplier for such an arithmetic unit would be smaller as only 14 b of precision need be multiplied. Such an arithmetic unit for a CF of 4 is shown in Figure 5.

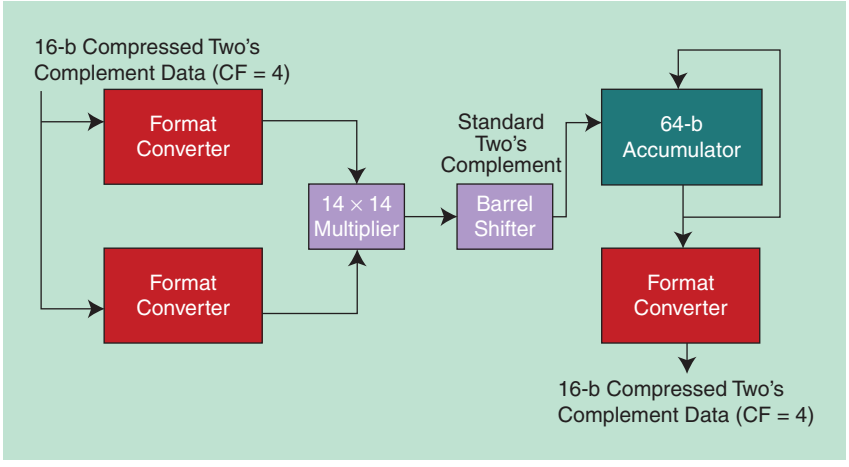
PERFORMANCE IMPROVEMENTS

Much of the information presented here to validate performance improvements has been previously published [4], but is provided here to correlate it with compressed two's complement formatting. The improvement in dynamic range by using compressed two's complement is easily calculated. In decibels, that is 3.06* (CF – 1) * [word width – $\log_2$ (CF)]. The result is a dramatic yet obvious improvement. What is less obvious is that while compressed two's complement improves dynamic range, it also improves round-off noise performance.

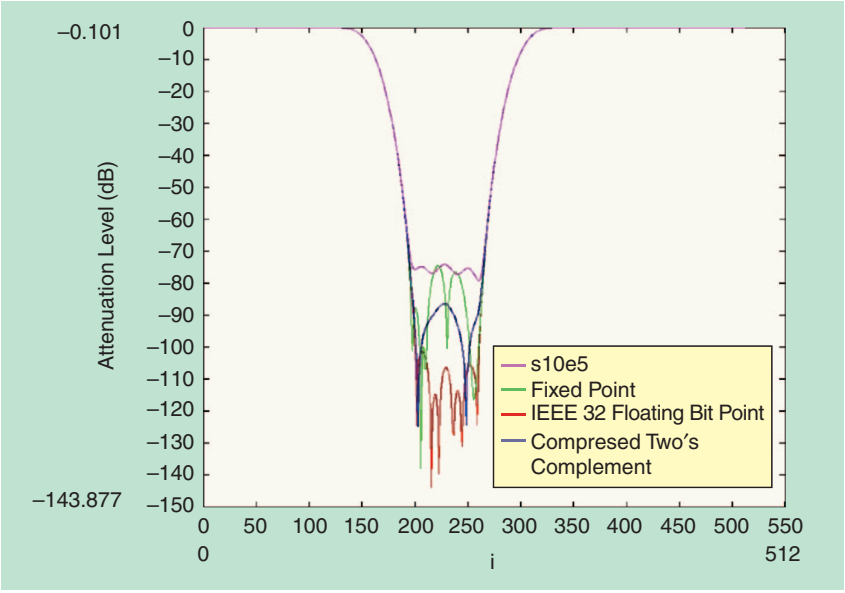


[FIG4] Peak signal versus peak round-off noise for several small compression factors in a fractional compressed two's complement format.

Figure 6 provides a powerful illustration of the performance gains that can be achieved through the use of compressed two's complement data formats. In this figure, the frequency spectrum of a digital notch filter is shown using four different formats. The filter was created using IEEE 754 32-b floating-point coefficients. The IEEE 754 floating-point format is shown in red. The coefficients were then converted to three other 16-b data formats and then converted back to IEEE floating point. The frequency spectrum was then plotted for the other three data formats. Traditional 16-b two's complement coefficients are plotted in green. The magenta line shows the result using a 16-b floating-point format (similar to IEEE Binary16) with 5 b of exponent and 10 b of mantissa. The purple line shows the same coefficients using a 16-b compressed two's complement data format with a CF of two. As can be seen, the rejected stop band is ten decibels lower with the compressed two's complement format than for either of the uncompressed formats. Other than conversion to and from a 16-b data format,



[FIG5] A multiply-accumulate circuit for a compression factor of four.



[FIG6] The frequency spectrum of a notch filter comparing the performance of various data formats.

no arithmetic was performed in the example illustrated in Figure 6.

A large DSP simulation has also been performed to validate improved performance with the compressed two's complement data formats [3]. An amplitude modulation (AM) receiver simulation was used to compare the noise performance for various data formats. The AM format was selected because this problem is well understood and still used even if in decline. This simulation contained several typical DSP algorithms that include the following: quantization to simulate 16-b analog-to-digital conversion, finite impulse response and infinite impulse response filters; demodulation; AGC; Hanning window; fast Fourier transform; and signal-to-noise ratio measurement via Parseval's theorem. AGC techniques were used following several stages to improve the native performance of the fractional format.

The simulation included the IEEE 32-b floating-point format (as a comparison baseline), together with four 16-b fixed-point formats. These were the s10e5 16-b floating point, a 16-b logarithmic format, a 16-b two's complement fractional fixed point, and a compressed two's complement fractional format with a CF of two. The simulation was performed for both weak signal and strong signal cases and both with and without the use of a single large post-multiply accumulator. Noise was not added to the simulation, so the resulting noise is a consequence of round-off errors during calculation, quantization to simulate A/D conversion, and out of band filter rejection (just over 50 dB). The simulation results are shown in Table 2.

[TABLE 2] SUMMARY OF SIMULATION RESULTS (IN dB).

FORMAT	WEAK SIGNAL SNR WITHOUT ACCUM	WEAK SIGNAL SNR WITH ACCUM	STRONG SIGNAL SNR	DYNAMIC RANGE
IEEE 754 32-B FLOATING POINT	31.97	31.97	50.53	1530
16-B s10e5 FLOATING POINT	8.44	7.90	42.06	***252***
16-B LOGARITHMIC	8.23	8.32	38.61	**385**
16-B FIXED-POINT FRACTIONAL	***13.30***	***24.93***	***44.42***	96
16-B COMPRESSED TWO'S COMPLEMENT FRACTIONAL (CF = 2)	**21.91**	**27.16**	***50.13***	181

Note: Among the 16-b formats, boldface and underlines indicates the best performer and italicized boldface the second best performer.

As can be seen from Table 2, the compressed two's complement format significantly outperformed the other 16-b formats in terms of noise performance and approached the performance of 32-b floating point for this simulation. It also provides almost twice the dynamic range of traditional fixed point.

CONCLUDING REMARKS

In this article, we have introduced and analyzed a new family of compressed fixed-point data formats for signal processing applications. These sign-bit compressed two's complement data formats are shown to provide greater dynamic range and improved noise performance over traditional fixed-point and floating-point data formats. Of course, the most desirable implementation for compressed two's complement would be in the ALU of a high-speed programmable DSP. Our results indicate that such a DSP should outperform a traditional DSP of equivalent data width in terms of algorithm performance.

AUTHORS

Manuel Richey (manuel.richey@honeywell.com) is a principal engineer at Honeywell International in Kansas where he has worked for over 25 years. He is also a computer science instructor at Fort Scott Community College and holds ten U.S. patents.

Hossein Saiedian (saiedian@eecs.ku.edu) is a professor and an associate chair in the Department of Electrical Engineering and Computer Science and a member of the Information and Telecommunication Center Lab at the University of Kansas.

REFERENCES

[1] C. Maxfield and A. Brown, *How Computers Do Math*. Hoboken, NJ: Wiley, 2005.

[2] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. New York: Oxford Univ. Press, 1999.

[3] M. Richey, "The application of irregular data formats for improved performance in 16-bit digital signal processing systems," M.S. thesis, Dept. Elect. Eng. Comp. Sci., Univ. Kansas, 2006.

[4] M. F. Richey and H. Saiedian, "A new class of floating-point data formats with applications to 16-bit digital-signal processing systems," *IEEE Commun.*, vol. 47, no. 7, pp. 94–101, 2009.

[5] E. J. Tan and W. B. Heinzelman, "DSP architectures: Past, present and future," *ACM SIGARCH*, vol. 31, no. 3, pp. 6–19, 2003.

[SP]