Quick and Easy Binary to dB Conversion

George Weistroffer
gweistro@yahoo.com

Jeremy A. Cooper
cooperja@gmail.com

Jerry H. Tucker
jhtucker@vcu.edu

Virginia Commonwealth University

Abstract

An algorithm is developed for converting binary integers to decibels (dB). Compared to known alternative methods the algorithm is not only faster, but requires less memory. It also has the advantage of being easy to implement. The presented technique is useful for applications such as converting the output of an analog to digital converter to discrete dB values. The speed of the algorithm is due to the fact that no floating-point operations are required. In fact, the only real time arithmetic employed is one integer subtraction. The algorithm can easily be implemented in a microcontroller or FPGA.

1. Introduction

A frequent task performed by embedded systems is the real time collection of analog data. One such application is the collection of acoustic data using an array of MEMS microphones [7-9]. To monitor this process, it is desirable to provide an indication in decibels (dB) of the amplitudes of the analog inputs. In response to this need, a simple and fast algorithm has been developed for converting a binary word, representing an analog voltage, to a dB value.

The calculation of decibels is challenging because it involves obtaining a logarithm. Numerous methods have been proposed for calculating logarithms [1-6]. Any of these procedures could be used to first obtain the necessary logarithm and then use this logarithm to calculate the dB value. However, such an approach introduces unnecessary complexity and requires excessive computation time. An algorithm is presented that avoids these difficulties by directly obtaining the dB value. The method does not require the calculation of logarithms in real time, but consists of a two-table lookup process using very small tables. The assumption is made, as is the usual case, that a high degree of precision is not required. The effectiveness of the algorithm is demonstrated by the fact that the dB value of a 16-bit word can be obtained to within about ½ dB using two lookup tables each containing only 16-bytes. The processor overhead in performing this conversion is minimal because the dB conversion simply consists of shifting the input word to obtain the indexes into two tables, and then subtracting the two values read from the tables.

2. The Method

Let w be the value in an unsigned n-bit binary word, and let w_i be the i^{th} bit of that word such that

$$w = (w_{n-1} w_{n-2} \ldots w_i \ldots w_0)_2 \tag{1}$$

The binary word w is assumed to represent an analog to digital converter (ADC) voltage. Let G be the comparison in dB of w to the reference of $2^n - 1$, which is the maximum possible value of w. Then G is given by

$$G = 20\log_{10}\left(\frac{w}{2^n - 1}\right) \tag{2}$$

This can be rewritten as

$$G = 20\log_{10}(w) - 20\log_{10}(2^n - 1) \tag{3}$$

Considering the computation of equation (3), it is apparent that for any particular value of n, the term $20\log_{10}(2^n - 1)$ is a constant and does not have to be computed in real time. The values of $20\log_{10}(2^n - 1)$ for n from 8 to 20 are shown in Table 1.

Table 1. The G_c table showing the value of $20\log_{10}(2^n\text{-}1)$ for n from 8 to 20.

n	$20\log_{10}(2^n - 1)$
8	48.1
9	54.2
10	60.2
11	66.2
12	72.2
13	78.3
14	84.3
15	90.3
16	96.3
17	102.4
18	108.4
19	114.4
20	120.4

The difficulty in evaluating equation (3) is the computation of the $20\log_{10}(w)$ term. Since w changes in real time, $20\log_{10}(w)$ must be continually recalculated. Therefore, a fast method of calculating $20\log_{10}(w)$ is necessary. Developing such a method is made easier by recognizing that dB values typically do not need to be represented with high precision. Therefore, it is possible to use an approximation of w. To approximate w, only the most significant bits of w will be considered.

Assuming $w \neq 0$, w can be expressed as

$$w = (1w_{m-1}w_{m-2}...w_0)_2 \tag{4}$$

where bit m with $m < n$ is the left-most (most significant) "1" of w. The approximation of w includes only the first r-bits to the right of the most significant "1" of w. So that when $m > r$

$$w \geq (1w_{m-1}w_{m-2}...w_{m-r}0...0)_2 \tag{5}$$

and

$$w \leq (1w_{m-1}w_{m-2}...w_{m-r}1...1)_2 \tag{6}$$

Note that when $m \leq r$ the above approximation for w is equal to w since all significant bits of w are included.

Let R be the value of $w_{m-1}w_{m-2}...w_{m-r}$ so that

$$R = \left(w_{m-1}w_{m-2}...w_{m-r}\right)_2 \tag{7}$$

Then equation (5) can be rewritten as

$$w \geq 2^m + R\times 2^{m-r} \tag{8}$$

and equation (6) can be rewritten as

$$w \leq 2^m + R\times 2^{m-r} + 2^{m-r} - 1 = 2^m + (R+1)\times 2^{m-r} - 1 \tag{9}$$

From equation (8)

$$\begin{aligned} \log_{10}(w) &\geq \log_{10}(2^m + R\times 2^{m-r}) \\ &= \log_{10}\left((1+R\times 2^{-r})2^m\right) \\ &= \log_{10}\left(2^m\right) + \log_{10}\left(1+R\times 2^{-r}\right) \\ &= \log_{10}(2)\log_2(2^m) + \log_{10}\left(1+R\times 2^{-r}\right) \\ &= \log_{10}(2)m + \log_{10}\left(1+R\times 2^{-r}\right) \end{aligned} \tag{10}$$

And from equation (9)

$$\begin{aligned} \log_{10}(w) &\leq \log_{10}(2^m + R\times 2^{m-r} + 2^{m-r} - 1) \\ &= \log_{10}\left((1+R\times 2^{-r} + 2^{-r} - 2^{-m})2^m\right) \\ &= \log_{10}\left(2^m\right) + \log_{10}\left(1+(R+1)\times 2^{-r} - 2^{-m}\right) \\ &= \log_{10}(2)\log_2(2^m) + \log_{10}\left(1+(R+1)\times 2^{-r} - 2^{-m}\right) \\ &= \log_{10}(2)m + \log_{10}\left(1-2^{-m} + (R+1)\times 2^{-r}\right) \end{aligned} \tag{11}$$

The minimum value of G, $G_{\min}$, is obtained from equation (3) and equation (10).

$$\begin{aligned} G_{\min} &= 20\log_{10}(2)m + 20\log_{10}\left(1+R\times 2^{-r}\right) \\ &\quad -20\log_{10}(2^n - 1) \end{aligned} \tag{12}$$

The maximum value of G, $G_{\max}$, is obtained from equation (3) and equation (11).

$$\begin{aligned} G_{\max} &= 20\log_{10}(2)m + 20\log_{10}\left(1-2^{-m} + (R+1)\times 2^{-r}\right) \\ &\quad -20\log_{10}(2^n - 1) \end{aligned} \tag{13}$$

After eliminating the common term in equation (12) and equation (13), the difference between $G_{\max}$ and $G_{\min}$ is seen to be

$$\begin{aligned} G_{\max} - G_{\min} &= 20\log_{10}\left(1-2^{-m} + (R+1)\times 2^{-r}\right) \\ &\quad -20\log_{10}\left(1+R\times 2^{-r}\right) \end{aligned} \tag{14}$$

But equation (14) can be written as

$$\begin{aligned} G_{\max} - G_{\min} &= 20\log_{10}\left(\frac{1-2^{-m}+(R+1)\times 2^{-r}}{1+R\times 2^{-r}}\right) \\ &= 20\log_{10}\left(\frac{1+R\times 2^{-r} - 2^{-m} + 2^{-r}}{1+R\times 2^{-r}}\right) \\ &= 20\log_{10}\left(1+\frac{2^{-r}-2^{-m}}{1+R\times 2^{-r}}\right) \\ &= 20\log_{10}\left(1+\frac{1-2^{-(m-r)}}{2^r+R}\right) \end{aligned} \tag{15}$$

Equation (15) is valid for $m > r$. When $m \leq r$, the approximation for w is the actual value of w, and $G_{\max} - G_{\min} = 0$.

It would be reasonable to approximate G as the average of $G_{\min}$ and $G_{\max}$. This is obtained by setting the approximation of G to $G_{\min}$ with an error reduction factor obtained by taking one half of $G_{\max} - G_{\min}$, as shown in equation (16).

$$G \approx G_{\min} + \frac{1}{2}\left(G_{\max} - G_{\min}\right) \qquad (16)$$

The problem with this approach is that $G_{\max} - G_{\min}$ is a function of both m and R, and this makes it impossible to establish a simple two-table lookup scheme. Although it appears possible to develop an error reduction factor that avoids this problem, only the simple case of using $G_{\min}$ as an approximation for G will be considered. Using this approach, the maximum error for any particular R and any particular $m > 0$ is the difference between $G_{\max}$ and $G_{\min}$ given by equation (15). For $m \le 0$, the error is 0.

For any particular value of m, equation (15) takes on the maximum value when $R = 0$. Therefore,

$$G_{max} - G_{\min} \le 20\log_{10}\left(1 + \frac{1-2^{-(m-r)}}{2^r}\right) \qquad (17)$$

Recall that for any particular application, n and r are fixed. It is only m and R that vary in real time. Separating equation (12) into the part that depends on m, the part that depends on R, and the constant part that depends on neither m nor R gives

$$G_{\min} = G_m(m) + G_R(R,r) - G_c(n) \qquad (18)$$

Using this as the approximation for G gives

$$G \approx G_m(m) + G_R(R,r) - G_c(n) \qquad (19)$$

Where

$$G_m(m) = 20\log_{10}(2)m \qquad (20)$$

$G_R(R,r)$ in real time is effected only by R and is given by

$$G_R(R,r) = 20\log_{10}\left(1 + R \times 2^{-r}\right) \qquad (21)$$

$G_c(n)$ does not vary in real time, and is given by

$$G_c(n) = 20\log_{10}(2^n - 1) \qquad (22)$$

Equation (17) predicts the maximum error and leads to the conclusion that

$$\max\ error \le 20\log_{10}\left(1 + \frac{1-2^{-(m-r)}}{2^r}\right) \text{ when } m > r \qquad (23)$$

and

$$\max\ error = 0 \text{ when } m \le r \qquad (24)$$

Since $m < n$, equation (23) takes on its maximum value when $m = n$-1. Therefore, the worst-case error is given by

$$worst\ case\ error = 20\log_{10}\left(1 + \frac{1-2^{-(n-1-r)}}{2^r}\right) \qquad (25)$$

This worst-case error based on equation (25) is shown in Table 2.

Table 2. The worst-case error in dB as a function of *n* and *r*.

n	$r = 1$	$r = 2$	$r = 3$	$r = 4$	$r = 5$
8	3.48	1.88	.96	.46	.2
9	3.50	1.91	.99	.49	.23
10	3.51	1.92	1.01	.51	.25
11	3.52	1.93	1.02	.52	.26
12	3.52	1.93	1.02	.52	.26
13	3.52	1.94	1.02	.52	.27
14	3.52	1.94	1.02	.53	.27
15	3.52	1.94	1.02	.53	.27
16	3.52	1.94	1.02	.53	.27
17	3.52	1.94	1.02	.53	.27
18	3.52	1.94	1.02	.53	.27
19	3.52	1.94	1.02	.53	.27
20	3.52	1.94	1.02	.53	.27

3. Implementation

It is apparent from equation (18) that G can be calculated using two lookup tables: one for G_m and another for G_R. The G_m-table contains n-entries and uses m as the index into the table. The G_R-table contains 2^r-entries and uses R as the index into the table. The basic procedure for finding G based on equation (19) is summarized by Algorithm 1.

Algorithm 1. The basic algorithm.

To convert w to dB, perform the following steps.

Initial Steps:
1. Set up the G_m and G_R tables.
2. Calculate G_c.

Real time steps:
1. From w find m and R.
2. Using m as the index, look up the value of G_m.
3: Using R as the index, look up the value of G_R.
4: Sum G_m and G_R, and subtract the constant G_c.

In practice, it is recommended that several refinements be made to the basic algorithm. First, it is not necessary to subtract G_c each time a dB value is computed. It would be slightly more efficient to combine the G_c value with each entry in one of the tables, such as G_m, and look up the combined value. Another refinement that can be made, to avoid having to deal with fractional values, is to scale the entries in the tables so that the integer values in the tables represent fractions of dB, such as ½ or ¼ dB. A final useful refinement follows from the fact that G will always be negative. To avoid having to deal with negative numbers, the negation of G can be represented as an unsigned integer.

The above observations lead to the conclusion that in practice it is better to compute $-kG$, where k is a scale factor and G is given by equation (19). $-kG$ is expressed in the form

$$-kG \approx k\left(G_c - G_m\right) - kG_R \tag{26}$$

The actual procedure for obtaining the dB value is based on the above equation and consists of setting up two tables: one for $k(G_c - G_m)$ and one for kG_R. The procedure is outlined in Algorithm 2.

Algorithm 2. The refined algorithm.

To convert w to dB, perform the following steps.

Initial step:
1. Set up tables for $k(G_c - G_m)$ and kG_R.

Real time steps:
1. From w, find m and R.
2. Using m as the index, look up the value of $k(G_c - G_m)$.
3. Using R as the index, look up the value of kG_R.
4. Subtract kG_R from $k(G_c - G_m)$ to produce the negative of the scaled value of G.

4. An Example Implementation

The refined algorithm can be illustrated by presenting a specific example. Let w be a 16-bit unsigned word so that $n = 16$. Assume it is desired to find G with a resolution of ½ dB. Since Table 2 shows that $r = 4$ will result in a resolution of about ½ dB, let $r = 4$. To use integers to represent a precision of ½ dB, let the scale factor $k = 2$. In this case, equation (26) can be written as

$$-2G \approx 2\left(G_c - G_m\right) - 2G_R \tag{27}$$

Where G_c is given by Table 1 where $n = 16$ as

$$G_c(16) = 20\log_{10}(2^{16} - 1) = 96.3\,\text{dB} \tag{28}$$

G_m is given by equation (20), and G_R is given by equation (21).

Applying the above equations results in Table 3 and Table 4.

Table 3. The G_R-table showing G_R and $2G_R$ in dB when $n = 16$ and $r = 4$.

R	G_R	$2G_R$
0	0	0
1	.53	1
2	1.02	2
3	1.50	3
4	1.94	4
5	2.36	5
6	2.77	6
7	3.15	6
8	3.52	7
9	3.88	8
10	4.22	8
11	4.54	9
12	4.86	10
13	5.17	10
14	5.46	11
15	5.74	11

Table 4. The $(G_c - G_m)$ table showing in dB's G_m, $(G_c - G_m)$, and $2(G_c - G_m)$ when $n = 16$ and $r = 4$.

m	G_m	$G_c - G_m$	$2(G_c - G_m)$
0	0	96.33	193
1	6.02	90.31	181
2	12.04	84.29	169
3	18.06	78.27	157
4	24.08	72.25	145
5	30.10	66.23	132
6	36.12	60.21	120
7	42.14	54.19	108
8	48.16	48.16	96
9	54.19	42.14	84
10	60.21	36.12	72
11	66.23	30.10	60
12	72.25	24.08	48
13	78.27	18.06	36
14	84.29	12.04	24
15	90.31	6.02	12

To illustrate how to use these tables, consider the case where $n = 16$, $r = 4$, and

$$w = (1010{,}1010{,}1100{,}0011)_2 \tag{29}$$

From equation (4) $m = 15$ and from equation (7) $R = 5$. The last column of Table 4 shows that $2(G_c - G_m) = 12$ when $m = 15$. In the last column of Table 3, when $R = 5$, it is seen that $2G_R = 5$. Subtracting these two values gives

$-2G = 12 - 5 = 7$. Therefore, $G \approx -3.5$ dB. The actual value based on equation (2) is $G = -3.517$ dB.

4.1. An assembly language implementation

An 8051 assembly language implementation of the presented algorithm for the case when n = 16 and r = 4 is shown below. The assembly language subroutine is written such that it can be called from an 8051 C program using the Silicon Labs [10] and Keil [11] development tools. The value of w is passed in as a 16-bit unsigned integer using *R6* for the most significant byte and *R7* for the least significant byte. The -2G value is returned as an unsigned character in *R7*.

Table 5. An 8051 assembly language implementation of the refined algorithm when n = 16 and r = 4.

```
NAME BITDB
?PR?_bitdb?BITDB     SEGMENT CODE
     PUBLIC   _bitdb
     RSEG  ?PR?_bitdb?BITDB
_bitdb:
;CHECK IF W = 0
     MOV  A,R6     ;MOST SIG BYTE OF W
     ORL  A,R7     ;CHECK IF W = 0
     JNZ  NOTZ     ;JUMP IF W > 0 ELSE
     INC  R7       ;SET W = 1
NOTZ:
;DO 16 BIT SHIFT TO FIND m AND R
     MOV  A,#16    ;SET M TO N.
SHIFT:
     DEC  A        ;DECREMENT M.
     XCH  A,R7     ;GET LEAST SIG BYTE OF
     ADD  A,ACC    ;W AND SHIFT IT
     XCH  A,R7     ;THEN SAVE IT.
     XCH  A,R6     ;GET MOST SIG BYTE OF
     ADDC A,ACC    ;W AND SHIFT IT
     XCH  A,R6     ;THEN SAVE IT.
     JNC  SHIFT    ;LOOP TILL MOST SIG 1
;DO TABLE LOOK UP
     XCH  A,R6     ;GET R AND PUT IT IN
     SWAP A        ;LEAST SIG NIBBLE OF A
     ANL  A,#0FH   ;SAVE ONLY THE R BITS
     MOV  DPTR,#GRTBL ;POINT TO GRTBL
     MOVC A,@A+DPTR ;DO LOOKUP TO GET GR
     XCH  A,R6     ;SAVE GR IN R6, GET M
     MOV  DPTR,#GMTBL ;POINT TO GMTBL
     MOVC A,@A+DPTR ;LOOKUP Gc - Gm
     CLR  C        ;CLEAR BORROW
     SUBB A,R6     ;FIND G
     MOV  R7,A     ;RETURN WITH dB VALUE
     RET           ;OF -2G IN R7

;TABLE FOR 2GR IN dB's
GRTBL:  DB      0       ;R = 0
        DB      1       ;R = 1
        DB      2       ;R = 2
        DB      3       ;R = 3
        DB      4       ;R = 4
        DB      5       ;R = 5
        DB      6       ;R = 6
        DB      6       ;R = 7
        DB      7       ;R = 8
        DB      8       ;R = 9
        DB      8       ;R = 10
        DB      9       ;R = 11
        DB      10      ;R = 12
        DB      10      ;R = 13
        DB      11      ;R = 14
        DB      11      ;R = 15
;TABLE FOR 2(Gc - Gm) IN dB's
GMTBL:  DB      193     ;M = 0
        DB      181     ;M = 1
        DB      169     ;M = 2
        DB      157     ;M = 3
        DB      145     ;M = 4
        DB      132     ;M = 5
        DB      120     ;M = 6
        DB      108     ;M = 7
        DB       96     ;M = 8
        DB       84     ;M = 9
        DB       72     ;M = 10
        DB       60     ;M = 11
        DB       48     ;M = 12
        DB       36     ;M = 13
        DB       24     ;M = 14
        DB       12     ;M = 15
        END
```

5. Conclusion

A technique, that is both fast and easy to implement, has been developed for converting a binary integer, w, to its dB value. The method takes advantage of the fact that in practice decibels do not need to be calculated to high precision. This makes it possible to use two small lookup tables to find the dB value.

The method has been illustrated for the specific case where w is a 16-bit unsigned integer, and where the four bits following the most significant "1" of w are used to approximate w. In this case, using two 16-byte lookup tables, it is possible to find the dB value of w to within about ½ dB. An 8051 assembly language program is provided that implements this case. The algorithm could also easily be coded in VHDL and synthesized for implementation in an FPGA.

6. References

[1] Jean-Pierre Deschamps, Gery Jean Antoine Bioul, and Gustavo D. Sutter, *Synthesis of Arithmetic Circuits FPGA, ASIC, and Embedded Systems*, Wiley-Interscience, Hoboken, New Jersey, 2006.

[2] Milos D. Ercegovac and Tomas Lang, *Digital Arithmetic*, Morgan Kaufmann, San Francisco, 2004.

[3] Kai Hwang, *Computer Arithmetic Principles, Architecture, and Design*, John Wiley and Sons, New York, 1979.

[4] J. Koren, *Computer Arithmetic Algorithms*, Prentice Hall, Englewood Cliffs, NJ, 1993.

[5] D. K. Kostopoulos, “An Algorithm for the Computation of Binary Algorithms,” IEEE Trans. On Computers, Vol 40, pp. 1267-1270, November 1991.

[6] Yi Wan and Chin-Long Wey, “Efficient Algorithms for Binary Logarithmic Conversion and Addition,” IEEE Proceedings Computers and Digital Techniques, Vol 146, pp. 168-172.

[7] W. M. Humphreys, Q. A. Shams, S. S. Graves, B. S. Sealey, S M. Bartram, T. Comeaux, “Applications of MEMS Microphone Array Technology to Airframe Noise Measurements”, Extended abstract presented at the 11th AIAA/CEAS Aeroacoustics Conference, Monterey, CA. May 23-25, 2005.

[8] Qamar A. Shams, “Design and Fabrication of 128-channel MEMS-based Acoustic Array” presented at Acoustic Society of America, San Diego, CA. November 16, 2004.

[9] Q. A. Shams, S. M. Bartram, W. H. Humphreys, and A. J. Zuckewar. “Phase Calibration of Microphones by Measurements in the Free-field,” Inter-Noise 2006, Honolulu, Hawaii, December 3-6, 2006.

[10] http://www.silabs.com/tgwWebApp/public/index.htm

[11] http://www.keil.com/