Q-Values in the Watch Window

J. Stevenson *Digital Signal Processing Solutions*

ABSTRACT

Digital signal processor (DSP) architectures are designed to handle finite number precision. Compared to floating point architecture, fixed-point DSPs offer a limited degree of precision but are usually more cost effective, and offer far superior performance.

To work with fixed-point formats, programmers need to take into consideration that each of the numerical operations should avoid overflow, and that the fractional portion of the number should not be truncated.

Q-values are used whenever a fixed-point variable is used to store a floating-point value. For example, the Q15 is a popular format in which the most significant bit is the sign bit, followed by 15 bits of fraction. The Q15 number has a decimal range between –1 and 0.9999 (0x8000 to 0x7FFF). This Q-value specifies how many binary digits are allocated for the fractional portion of the number.

The main purpose of this application report is to demonstrate how to display variables with a specified Q-value in the watch window. This functionality already exists in the memory window. For example, if the binary form of a variable is 00010011b, and you are using a Q value of 2, then the number is positive since the most significant binary digit is 0, and the two least significant binary digits are 11b. Thus, the fractional portion of ($2^{-1} + 2^{-2} = .75$) and 00100b is the whole portion ($2^2 = 4$), so the variable has a value of 4.75.

This report supplies a selection of general extension language (GEL) functions that can be used in the watch window to display Q-values. By following the instructions provided in this report, you will be able to get the functions working in a matter of minutes.

Contents

List of Figures

1 Introduction

This application report describes how to display variables in the watch window using Q-values. Appendix A provides a GEL script file that contains all of the code necessary to display Q-values in the watch window. If you are familiar with GEL, then you may wish to skip to the appendix and simply create a GEL file with the supplied code.

For more details on the supplied functions and how to use them, continue reading.

1.1 Q-Values

All fixed-point arithmetic is defined by a Q format. One such format is the Q15 (1.15). In this format, the most significant bit is the signed bit followed by a hexadecimal point (also known as the imaginary bit), and then 15 bits of fraction (or mantissa) normalized to 1. A number in Q15 format has a decimal range between –1 and 0.9999, which corresponds to 0x8000h and 0x7FFFh). In terms of a summation, the Q format is shown below.

Fractional number *h* can be expressed as:

$$h = \sum_{n=0}^{N} b_n 2^{-n} \tag{1}$$

or

$$h = -b_0 2^0 + b_1 2^{-1} + b_2 2^{-2} + \ldots + b_n 2^{-N} \tag{2}$$

Where *h* is the fractional number, *b* is binary value 1 or 0.

1.2 Setting Up the GEL File

To use the supplied functions, you must first create and load a GEL file that contains these functions.

1.2.1 Create a New GEL File

From inside Code Composer Studio™, create a blank source file. Copy and paste the code from Appendix A into this file, and then save the file. When saving the file, make sure that you have *.gel selected as the file filter at the bottom of the Save File dialog box.

1.2.2 Load the GEL File

The next step is to load these GEL functions into Code Composer Studio, so that you can use them. From the File menu, select Load GEL… item. Specify the file that you created and press OK. Code Composer Studio will now parse the file and load these functions into memory so that you can use them. If there is an error during this process, it is likely due to a syntax error in the file. If this happens, correct the error and try again.

1.3 Using the GEL Functions in the Watch Window

Once the file is loaded, you can use these functions in any feature of Code Composer Studio that uses the GEL-expression evaluator. This includes the watch window.

1.3.1 QN(x) Functions

The functions Q1(x) through Q31(x) are designed to display Q-values 1 through 31, respectively. If you wish to display a Q-value of 15 for the variable *i*, simply enter *Q15(i)* into the watch window.

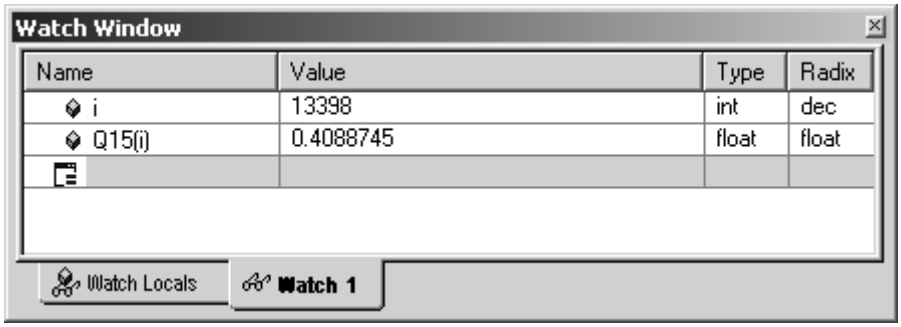

Figure 1. Watch Window

1.3.2 Q(x,y) Function

This function is designed to be more flexible so you can specify any Q-value as the first parameter. This function cannot be used in the watch window, as the Watch Window only accepts GEL functions that use 1 parameter.

1.3.3 pow(x,y) Function

This function was created to compute X to the power of Y. It is used by all of the Qn(x) and Q(x,y) functions. This is a useful function to have for use in your own scripts.

Appendix A Qvalue.gel

```
// Qvalue
// ======
//
// Q values are used when a fixed-point variable
// represents a floating-point value.  The Q represents the bit
// location where the decimal point is placed.
//
// This file provides a series of functions that are useful for
// entering Q-values into the watch window.  For example, if you
// have a variable n, and you want to use a Q–value of 2, you would
// enter Q2(n) into the watch window, where n is the variable.
//
// You can also create a generic Q() function that allows you to enter
// in the Q value as a parameter.  This one is useful in scripts,
// but currently cannot be added to the watch window, due to the fact
// that the watch window will only accept GEL functions with 1
// parameter.

Q(int value, int Q) {
   int variable = (int)value; // converts to a signed integer
   return (float)(variable/(pow(2,Q)));
}

Q1(int value) {
   return Q(value, 1);}
Q2(int value) {
   return Q(value, 2);}
Q3(int value) {
   return Q(value, 3);}
Q4(int value) {
   return Q(value, 4);}
Q5(int value) {
   return Q(value, 5);}
Q6(int value) {
   return Q(value, 6);}
Q7(int value) {
   return Q(value, 7);}
Q8(int value) {
   return Q(value, 8);}
Q9(int value) {
   return Q(value, 9);}
Q10(int value) {
   return Q(value, 10);}
Q11(int value) {
   return Q(value, 11);}
Q12(int value) {
   return Q(value, 12);}
Q13(int value) {
   return Q(value, 13);}
Q14(int value) {
   return Q(value, 14);}
```

```
Q15(int value) {
   return Q(value, 15);}
Q16(int value) {
   return Q(value, 16);}
Q17(int value) {
   return Q(value, 17);}
Q18(int value) {
   return Q(value, 18);}
Q19(int value) {
   return Q(value, 19);}
Q20(int value) {
   return Q(value, 20);}
Q21(int value) {
   return Q(value, 21);}
Q22(int value) {
   return Q(value, 22);}
Q23(int value) {
   return Q(value, 23);}
Q24(int value) {
   return Q(value, 24);}
Q25(int value) {
   return Q(value, 25);}
Q26(int value) {
   return Q(value, 26);}
Q27(int value) {
   return Q(value, 27);}
Q28(int value) {
   return Q(value, 28);}
Q29(int value) {
   return Q(value, 29);}
Q30(int value) {
   return Q(value, 30);}
Q31(int value) {
   return Q(value, 31);}


// pow()
// =====
//
// This function is used to calculate exponents.  It takes 2 parameters
// base and exp, and returns base to the power of exp.
// This function only works for whole positive numbers.

// C prototype: double pow(double base, double exp);


pow(double base, double exp) {             // used double to match Microsoft standard
       double answer = 1;// initialize
       int i;                              // counter for loop
       for (i = 1; i<=exp; i++)            // multiple answer by base exp times
              answer = answer * base;
       return answer;                      // return value stored in answer
}
```

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third–party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265