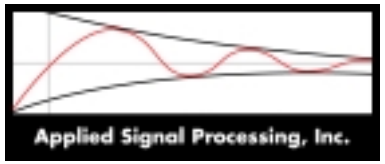
Q-Math Refresher

Understanding Q Math Basics

Shawn Steenhagen - Applied Signal Processing, Inc.



3 Marsh Court
Madison, WI 53718
Tele: 608-441-9921
Fax: 608-441-9924
Web: www.appliedsignalprocessing.com

Fixed Point Representation of Real Numbers

- A fixed point value, *xq*, represents a real number, *x*, by using an implied number of fractional bits.
- The number of implied fractional bits is referred to as the "Q-type" of the value xq.
- Notation: Q15/16 → Q15 value in 16 bits.

$$xq = Rnd(x \cdot 2^{Q})$$

$$x = \frac{xq}{2^{Q}}$$

Multiply Operation

- Multiplying two 16 bit values creates a 32 bit result.
- What is the Q-type of this 32 bit Result?
- Want to calculate z = x*y:

$$zq = xq \cdot yq$$
$$zq = x2^{QX} \cdot y2^{QY}$$
$$zq = (xy)2^{QX+QY}$$
$$zq = z2^{QX+QY}$$

- By Definition, zq is (QX+QY) in 32 bits.
- For two Q15 Numbers, a product is then Q30/32

Divide Operation

- The divide of two 16 bit numbers can be extended to provide a 32 bit result in the form:

- This is the ratio of the two 16 bit numbers as a Q16/32
- The resulting Q-type of a 32 bit divide result is then:

$$Acc \equiv \left(\frac{xq}{yq}\right)2^{16} = \left(\frac{x2^{QX}}{y2^{QY}}\right)2^{16} = \left(\frac{x}{y}\right)2^{(16+QX-QY)} = z2^{(16+QX-QY)}$$

which by definition, is z=x/y as a Q(16+ QX - QY) in 32 bits.

Mixed Q Math Operations

- Conversion from one Q-type to another is a right or left shift. qz_z = qx_x<<(QZ-QX)
- Matter of tracking the Q-type of a 32 bit result and applying the correct number of shifts to get the result in the desired Q format.
- qz_z=qx_x*qx_y<<(QZ-(QX+QY))
- qz_z=qx_x/qx_y<<(QZ-(16 + QX - QY))
- Typical 32 bit to 16 bit conversion: q15_c = HI_WORD((q15_a*q15_b)<<1)
- CAUTION: C Compiler dependent code may not provide proper overflow/underflow protection.

Coding Tips

- Typedefs and naming conventions.
- Macros for Auto-Initialization/Assignment
- Macros and Functions for mixed Q Math Operation.

Typedefs and Naming Conventions

- Use a “hungarian” naming convention to show Q-type of the variable somewhere within the name. i.e. q15_x, q8_y etc.
- Use typedefs to help readability in variable declarations: q15 q15_var, my_var;

```
typedef int q15;  /* 16 Bit Fractional Two's Complement Value */
typedef int q0;    /* all other q types */
typedef int q1;
typedef int q2;
typedef int q3;
// etc.

typedef long q31;
// etc.
```

Macros For Q-Type Assignment

- Keeps your thinking in the “floating point” domain.
- Allows assignments/initializations of the form:
  q8_var = Q8(1.67);
  q15_mu = Q15(0.001);

```
#define Q0(x) ( (x)<0 ?  ((int) (1.0*(x) - 0.5)) : (min(32767,(int) ((32767.0/32768.0)*(x) + 0.5))) )
#define Q1(x) ( (x)<0 ?  ((int) (2.0*(x) - 0.5)) : (min(32767,(int) ((32767.0/16384.0)*(x) + 0.5))) )
#define Q2(x) ( (x)<0 ?  ((int) (4.0*(x) - 0.5)) : (min(32767,(int) ((32767.0/8192.0)*(x) + 0.5))) )
.
.
#define Q13(x) ( (x)<0 ? ((int) (8192.0*(x) - 0.5))  : (min(32767,(int) ((32767.0/4.0)*(x) + 0.5))) )
#define Q14(x) ( (x)<0 ? ((int) (16384.0*(x) - 0.5)) : (min(32767,(int) ((32767.0/2.0)*(x) + 0.5))) )
#define Q15(x) ( ((x)<(0.0)) ? ((int) (32768.0*(x) - 0.5)) : (min(32767,(int) (32767.0*(x) + 0.5))) )
```

Macros and Functions for Mixed Q Math Operations.

```
// Macros
#define HI_WORD(x)   ((int)((x)>>16))  // Get to the high portion of a long.
#define QMRSHIFT(QRESULT,QA,QB) (QA + QB -(QRESULT))
#define Q15MULT(a,b)     ((int) (((long)a * (long)b)>>15))
#define Q15MULTR(a,b)     ((int) (((long)a * (long)b + (long) 0x4000)>>15))
#define QMULT(a,b,right_shval) ((int) (((long)a * (long)b)>>right_shval))
#define QMSHIFT(QVZ,QVX,QVY) (QVZ-QVX-QVY+1)
```

```
// Function prototypes.
qz   qz_QMultOVUN(qx x, qy y, int left_shiftval);
q31 l_q15q31mult(q31 x, q15 y);
qx qx_InverseQ15(q15 x, int qval);
q15  q15_InverseInt(int x);
qx qx_Divide(qx num, qx den, int desiredQval);
qx qx_NormDivide(long num, long den, int qval);
```

Examples:

q15_c = Q15MULT(q15_a,q15_b);

q12_var = qz_QMultOVUN(q15_x,q8_y,QMSHIFT(12,15,8));

Use of Tools

- Keep yourself in the “floating point” domain during development/debugging.
- Some emulators provide Q-type support.
- Q-Smart™ Watch Window Tool (Code Composer Studio).

QSmart Watch Window

Name	Value	Raw Value	Q Type	Type	Radix
myglobal1	1234.567627	0x004D2915	12	long	float
myglobal2	45.799805	0x0002DCCC	12	long	float
my_var1	99.660004	0x063A8F60	20	long	float
new_global	987.652344	0x0003DBA7	8	long	float
new_global2	320.281250	0x00014048	8	long	float
my_var2	2004.437500	0x0001F51C	6	long	float

Conclusion/Summary

- Using suggested techniques can help improve code readability in fixed point environments and help speed up development time.
- Having a well tested core of mixed Q type support functions and tools provides a more robust development environment that lets you focus on the application.