Fixed Point Format and its Operations

Indian Institute of Technology, Hyderabad

Introduction

- VLSI implementation of digital signal processing algorithms usually requires fixed-point arithematic for the sake of chip area,operation speed, and power consumption.
- A value of a fixed-point data type is essentially an integer that is scaled by an implicit specific factor determined by the type. For example,

  1.23 => 1230 in a fixed-point format with a scaling factor of 1/1000

  1,230,000 =>1230 with a scaling factor of 1000.
- We use the following notation (W,Q,0/1 for unsigned/signed) to represent Fixed point numbers.
  - W represents total word length(in bits).

    W=integer word length(I)+fractional word length(Q)+1(signed bit)
  - Q represents total fraction bits.

  For Example:- To represent 9.692 in 32 bit integer Fixed point Format
    - W=32bits, I=4bits, Sign bit =1bit =>therefore Q=27bits

Conversion

- **Floating Point to Fixed Point Conversion**
  - 'a' be floating-point value,'b' be the converted fixed -point value and Q is the scaling factor which represents the amount of fractional length.

$$b = a * 2^{Q}$$

Examples:

a = -3.613 → binary of -3.613 is '-ve of 011.100111001111'

Q = 7 (Will be discussed in 5. Selection of Q - Type)

b = a * 2^{Q} → -3.613 * 2^{7} = -462.464

→ binary of -462.464 is '-ve of 0111001110.01111'

From a to b, there is a shift of decimal point by 7 positions.

While storing the value, we only use the integer value.

So, the resultant fixed point value is -462 i.e., 1111111000110010

C Code:

```
int16_t   fxpValue;
float     floatValue;
uint8_t   qValue;

fxpValue = (int16_t)(floatValue * (1<<qValue));
```

1<<qValue => 2^{Q}

Conversion

- **Fixed Point to Float Point Conversion**

  Let the value 'b', a fixed-point value, be converted to a floating-point value, 'a' and Q value represents the amount of fractional length.

$$a = b / 2^{Q}$$

Examples:

b = -462 → binary of -462 is '1111111000110010'

Q = 7

$a = b/2^{Q}$ → $-462/2^{7} = -3.609375$

→ binary of -3.609375 is '(-ve) of 011.1001110'

From a to b, there is a shift of decimal point by 7 positions.

While storing the value, we only use the integer value.

So, the resultant floating point value is -3.609375 => (-ve) of 011.1001110

C Code:

```
int16_t   fxpValue;
float     floatValue;
uint8_t   qValue;

floatValue = (float)fxpValue / (1<<qValue);
```

1<<qValue => 2^{Q}

Operations on Fixed Point Data

- **Addition of Fixed Point Data**
  - Addition of 2 Fixed-Point Data can be done only when the Q format of both the values is the same
  - Make sure that the Q formats are aligned and the resultant will also be in the same Q format.
  - If values are not in same Q format then we must bring to same Q format preferably in higher Q format

Example:

val1st = 128; (Q-Format = 12)

val2nd = 64; (Q-Format = 11)

val1st is $128 = x * (1 << 12)$, and
val2nd is $64 = y * (1 << 11)$

Thus, Addition should mean: x*(1<<12) + y*(1<<11)

but if we add val1st and val2nd directly, it becomes : 128+64 which is a mistake.

We must bring them to the same Q-format.

1. Lower Q-format
2. Higher Q-format

new val1st = 128 and

new val2nd = 64*2 (*2 is because, we are increasing Q-format from 11 to 12)

Thus, we add them,

128 + 128 = 256

where, 256 is in Q-format = 12.

Sample Code for Addition/Subtraction

```c
#include <stdio.h>
#include<string.h>
#include<stdlib.h>

int main()
{
    float val1=0.03125;short int q1=12;
    float val2=0.03125;short int q2=11;
    short int format;


    int16_t fxpval1=(int16_t)(val1*(1<<q1)) ;    // Converted to Fixed point Q(12) format
    int16_t fxpval2=(int16_t)(val2*(1<<q2))  ;   // Converted to Fixed point Q(11) format

    if(q2>q1){                                   //Making both values to same Q format preferably to heigher Q format(12)
        fxpval1=fxpval1*(1<<(q2-q1));
        format=q2;
    }
    else{
        fxpval2=fxpval2*(1<<(q1-q2));
        format=q1;
    }

    int16_t result=fxpval1+fxpval2;               // Fixed point addition

    float resultflt= (float)result/(1<<format);   // Convert Fixed point back to floating value
    printf("Value1=%f",val1);
    printf("value2=%f",val2);
    printf("\nResult =%f\n",resultflt);             // Result

}
```

```
Value1=0.031250value2=0.031250
Result =0.062500
```

Operations on Fixed Point Data

- **Multiplication of Fixed Point Data**
  - Multiplication of 2 Fixed-Point Data of different Q formats can be done without any issue.
  - The resultant value will be in the Q format (Q = q1 + q2, where q1, q2 are the Q formats of 1st and 2nd Fixed-Point values). So, we must take care that the resultant is brought back to the intended Q format.

Example:

val1st = 128; (Q-Format = 12)

val2nd = 64; (Q-Format = 11)

val1st is 128 = x * (1<<12), and

val2nd is 64 = y * (1<<11).

Multiplication = [x * (1<<12)] * [y * (1<<11)] = x * y * (1<<23)

Now, the resultant should be converted back to the intended Q-format.

In case, the required result of the Q-format is 11.

The output has a Q-format of 23, as evident from above value.

Thus to bring it to the Q-format=11, we need to multiply the resultant by pow(2, 11-23).

Resultant = x * y *(1<<23) / (1<<12). => 8192/(1<<12) = 2

Hence the resultant is going to be x * y * (1<<11) or 2 in Q-format=11.

But care must be taken that these values don't overflow.

Sample Code for Multiplication

```c
#include <stdio.h>
#include<string.h>
#include<stdlib.h>
int16_t fxdMulScalar(int16_t num1,short int q1,int16_t num2,short int q2,short int resq);
int main()
{
    float val1=0.03125;short int q1=12;
    float val2=0.03125;short int q2=11;
    float resultflt;
    int16_t fxpval1=(int16_t)(val1*(1<<q1)) ;    // Converted to Fixed point Q(12) format
    int16_t fxpval2=(int16_t)(val2*(1<<q2))  ;   // Converted to Fixed point Q(11) format
    int16_t result;

    short int decFormat=11;                      // Final Required format is 11

    result=fxdMulScalar(fxpval1,q1,fxpval2,q2,decFormat);  // Multiplication in Fixed point

    resultflt=(float)(result)/(1<<decFormat);             // Converting back to Flt values
    printf("Value1=%f",val1);
    printf("Value2=%f\n",val2);
    printf("Result=%.7f\n",resultflt);


}


int16_t fxdMulScalar(int16_t num1,short int q1,int16_t num2,short int q2,short int resq){        //Multiplication Function in Fixed point
    int temp=(int) num1*num2;                                  // Multiplication of two 16 bit integer is stored in 32 bit integer
    int16_t res;
    res=temp/(1<<(q1+q2-resq));                                // Q(23)/Q(12+11-11)  =>  Q(23)/Q(12)  => Q(11)
    return res;

}
```

```
Value1=0.031250Value2=0.031250
Result=0.000977
```

Operations on Fixed Point Data

- **Division of Fixed Point Data**
  - Division of 2 Fixed-Point Data of different Q formats can be done without any issue.
  - The resultant value will be in the Q format (Q = q1 - q2, where q1, q2 are the Q formats of 1st and 2nd Fixed-Point values). So, we must take care that the resultant is brought back to the intended Q format.

Example:

val1st = 128; (Q-Format = 12)

val2nd = 64; (Q-Format = 11)

val1st is 128 = x * (1<<12), and

val2nd is 64 = y * (1<<11).

Division = [x * (1<<12)] / [y * (1<<11)] = (x / y) * (1<<1) => 128/64 = 2.

Now, the resultant should be converted back to the intended Q-format.

In case, the required result of the Q-format is 11.

The output has a Q-format of 1, as evident from above value.

Thus to bring it to the Q-format=11, we need to multiply the resultant by pow(2, 11-1).

Resultant = x * y *(1<<1)* (1<<10). => 2*(1<<10) = 2048

Hence the resultant is going to be x * y * (1<<11) or 2048 in Q-format=11.

Sample Code for Division

```c
#include <stdio.h>
#include<string.h>
#include<stdlib.h>
int16_t fxdDivScalar(int16_t num1,short int q1,int16_t num2,short int q2,short int resq);
int main()
{
    float val1=0.03125;short int q1=12;
    float val2=0.01125;short int q2=11;
    float resultflt;
    int16_t fxpval1=(int16_t)(val1*(1<<q1));    // Converted to Fixed point Q(12) format
    int16_t fxpval2=(int16_t)(val2*(1<<q2));   // Converted to Fixed point Q(11) format
    int16_t result;

    short int decFormat=11;                      // Final Required format is 11

    result=fxdDivScalar(fxpval1,q1,fxpval2,q2,decFormat);  // Division in Fixed point

    resultflt=(float)(result)/(1<<decFormat);             // Converting back to Flt values
    printf("Value1=%f",val1);
    printf("Value2=%f\n",val2);
    printf("Result=%f\n",resultflt);


}


int16_t fxdDivScalar(int16_t num1,short int q1,int16_t num2,short int q2,short int resq){
    int16_t res;
    res=(int16_t)(num1*(1<<(resq-(q1-q2)))/num2);                              // Q(12)*Q(11-1)/Q(11)  =>  Q(22)/Q(11)  => Q(11)
    return res;

}
```

```
Value1=0.031250Value2=0.011250
Result=2.782227
```

Selection of Q Format

- Q-Value must be chosen such that the fixed point representation doesn't overflow at any point while the next steps of processing are being done. In the event of overflow, reduce the Q-Value, and in case of loss of precision, increment the Q-Value.
- **Example:**

  val1 = 32766 (Q-format = 10)

  val2 = 32766 (Q-format = 10)

  If we need to find their sum and the resultant is expected in Q-format = 10 in a 16-bit data-type.

  Then resultant = val1 + val2 = 65532

  But the range of 16-bit data-type is [-32768, 32767].

  Hence, there is going to be overflow(and the result will be observed to be a -ve value).

  So, care must be taken here and Q-format should be reduced to compensate for this.

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Handling the Overflow

Overflowing can be avoided by changing the Q format according to the final output to be stored.

Example:- Multiplying 3.12678 and 12.547 results in 39.2319 which is stored in 16 bit integer as follows 6 integer bits(at least), 9 fractional bits and 1 sign bit .

```c
#include <stdio.h>
#include<string.h>
#include<stdlib.h>

int main()
{
    float val1 = 3.12678;
    float val2 = 12.547;

    int16_t fxpval1=val1*(1<<13) ; //Should be stored with Fixed point Format Q(2.13)
    int16_t fxpval2=val2*(1<<11) ; //Should be stored with Fixed point Format Q(4.11)
    int16_t result;

    //Suppose we want to store the result in Q(5.10) but this results in overflow as range exceeds.

    result=(int)(fxpval1*fxpval2)/(1<<14); //  Q(13)*Q(11)/Q(14) => Q(24)/Q(14)  => Q(10) stored in Q(5.10)

    float resultflt=(float)result/(1<<10); // Convert back to floting

    printf("Val1=%f\n",val1);
    printf("Val2=%f\n",val2);
    printf("Result=%f\n",resultflt);

}
```

```
Val1=3.126780
Val2=12.547000
Result=-24.770508
```

WRONG OUTPUT (Q(5.10) format)

```c
#include <stdio.h>
#include<string.h>
#include<stdlib.h>

int main()
{
    float val1 = 3.12678;
    float val2 = 12.547;

    int16_t fxpval1=val1*(1<<13) ; //Should be stored with Fixed point Format Q(2.13)
    int16_t fxpval2=val2*(1<<11) ; //Should be stored with Fixed point Format Q(4.11)
    int16_t result;

    //Should be stored with Fixed Point Format Q(6.9) as result of multiplication = 39.2317

    result=(int)(fxpval1*fxpval2)/(1<<15); //  Q(13)*Q(11)/Q(15) => Q(24)/Q(15)  => Q(9) stored in Q(6.9)

    float resultflt=(float)result/(1<<9);

    printf("Val1=%f\n",val1);
    printf("Val2=%f\n",val2);
    printf("Result=%f\n",resultflt);

}
```

```
Val1=3.126780
Val2=12.547000
Result=39.228516
```

CORRECT OUTPUT (Q(6.9) format)

Example1 Convolution Code in Fixed Point

```
void convolutionflt(float X[MAXLEN],float H[MAXLEN], float Y[MAXLEN],int lenX,int lenH,int lenY){

    for(int n=0;n<lenY;n++)
    {
        Y[n]=0;                                 //initialize values to zeros
        for(int k=0;k<=n;k++)
        {

          if((n-k)>=lenH||k>=lenX)              //condition to make convolution in bounds
            continue;
          Y[n]=Y[n]+(X[k]*H[n-k]);              //convolution Formulae
        }
    }
}
```

```
int fxdMulScalar(int num1,int q1,int num2,int q2,int resq){
    long long temp=(long long) num1*num2;
    int res;
    res=temp/(1<<(q1+q2-resq));
    return res;

}
```

```
int bitshift(int num,int shift)
{
    return num*(1<<shift);
}
```

```
void convolutionfxd(int X[MAXLEN],int H[MAXLEN], int Y[MAXLEN],int lenX,int lenH,int lenY,int Q){

   for(int n=0;n<lenY;n++)
   {
       Y[n]=0;                                 //initialize values to zeros
       for(int k=0;k<=n;k++)
       {

         if((n-k)>=lenH||k>=lenX)              //condition to make convolution in bounds
           continue;
         //Y[n]=Y[n]+(X[k]*H[n-k]);

           Y[n]=fxdAddScalar(bitshift(Y[n],0),fxdMulScalar(X[k],Q,H[n-k],Q,Q));  //Fixed point Convolution Formulae Implementation.

       }
   }
}
```

```
int fxdAddScalar(int num1,int num2)
{

    return num1+num2;

}
```

Result

```
For Q=29

X :
0.3826, 0.8851, 0.7762, 0.9141, 0.7922, 0.3347, 0.3856, 0.4915, 0.6484, 0.4206,

H :
0.3616, 0.0270, 0.6893, 0.0589, 0.7622, 0.9251, 0.5395, 0.4256, 0.1718, 0.7353, 0.2108, 0.3676, 0.5664, 0.4286, 0.7812, 0.5295, 0.8611, 0.1229, 0.0669, 0.1349, 0.9281, 0.8012, 0.0220, 0.0579, 0.0689,

Y :
0.1384, 0.3304, 0.5683, 0.9842, 1.1900, 1.8469, 2.3653, 2.5207, 2.8439, 2.7759, 2.7719, 2.6322, 2.8381, 3.1330, 2.8813, 2.9771, 3.1011, 3.2577, 2.8941, 2.4450, 2.5319, 2.7837, 2.9344, 2.7836, 2.4747, 1.5
981, 0.9113, 0.9970, 1.1347, 0.9660, 0.4063, 0.0807, 0.0691, 0.0290,

X (Fixed point) :
205416144, 475192448, 416731968, 490746144, 425313312, 179672080, 207025152, 263876608, 348081152, 225796864,

 H (Fixed point) :
194153120, 14481034, 370070848, 31643740, 409223296, 496645824, 289620672, 228478528, 92249544, 394742240, 113166592, 197371120, 304101696, 230087536, 419413632, 284257312, 462320416, 65969152, 35934416,
 72405168, 498254816, 430140320, 11799361, 31107406, 37007084,

 Y output in Fixed point :
74286358, 177388515, 305119186, 528375825, 638888605, 991522446, 1269864015, 1353271279, 1526795074, 1490315719, 1488178962, 1413154801, 1523683134, 1682006850, 1546878929, 1598300068, 1664897161, 174898
5980, 1553771964, 1312626088, 1359304347, 1494461199, 1575415039, 1494433331, 1328570393, 857992787, 489226604, 535242074, 609191953, 518617858, 218118324, 43320401, 37076739, 15564418,

Y output converted to Flt point :
0.1384, 0.3304, 0.5683, 0.9842, 1.1900, 1.8469, 2.3653, 2.5207, 2.8439, 2.7759, 2.7719, 2.6322, 2.8381, 3.1330, 2.8813, 2.9771, 3.1011, 3.2577, 2.8941, 2.4450, 2.5319, 2.7837, 2.9344, 2.7836, 2.4747, 1.5
981, 0.9113, 0.9970, 1.1347, 0.9660, 0.4063, 0.0807, 0.0691, 0.0290,
Error(Q=29)=0.0000000037
```

Selection of Q Format

```
Error(Q=5)=0.2318611592
Error(Q=6)=0.1066082492
Error(Q=7)=0.0624600649
Error(Q=8)=0.0291990191
Error(Q=9)=0.0140108326
Error(Q=10)=0.0069477740
Error(Q=11)=0.0039606523
Error(Q=12)=0.0018823675
Error(Q=13)=0.0010710200
Error(Q=14)=0.0004083993
Error(Q=15)=0.0002321110
Error(Q=16)=0.0001150089
Error(Q=17)=0.0000526611
Error(Q=18)=0.0000259197
Error(Q=19)=0.0000123324
Error(Q=20)=0.0000079704
Error(Q=21)=0.0000035672
Error(Q=22)=0.0000017126
Error(Q=23)=0.0000007134
Error(Q=24)=0.0000003026
Error(Q=25)=0.0000001373
Error(Q=26)=0.0000000585
Error(Q=27)=0.0000000231
Error(Q=28)=-0.0000000333
Error(Q=29)=-0.0000000064
Error(Q=30)=2.2352941036
Error(Q=31)=2.7058823109
```

Observations:

1) High Q value indicates more bits given to fraction part hence less quantization error.
2) Q value is selected such that fixed point operations does not overflow(Observe that int32 datatype overflows for Q=30 & Q=31 hence more error).

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Example2:- Complex Matrix Inversion of 2x2

- Sufficiently large Q(15) fractional length of 15 bit is taken to minimize error.
- Same Q format is taken every where for simplicity.
- Sufficiently large data type (int 32) is taken to store Fixed point data after operations without overflowing.

$$A^{-1} = \frac{1}{|A|}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

- Matrix Inverse of Larger Dimensions can implement using special Algorithms such as Blockwise Inverse, Cholesky decomposition Inverse, Gauss Jordan Method.

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

C Function to Implement Complex 2x2 Fixed point Matrix Inverse

```
1087  void invert2x2(int aReal[ROWS][COLUMN],int aImag[ROWS][COLUMN],int invReal[ROWS][COLUMN],int invImag[ROWS][COLUMN],int q)
1088  {
1089      // Calculating Determinant for real and Complex part separately.
1090      long int detReal=(fxdComplxMulScalar32(aReal[0][0],q,aReal[1][1],q,q)-fxdComplxMulScalar32(aImag[0][0],q,aImag[1][1],q,q))-
1091                              (fxdComplxMulScalar32(aReal[0][1],q,aReal[1][0],q,q)-fxdComplxMulScalar32(aImag[0][1],q,aImag[1][0],q,q));
1092
1093      long int detImag=(fxdComplxMulScalar32(aReal[0][0],q,aImag[1][1],q,q)+fxdComplxMulScalar32(aImag[0][0],q,aReal[1][1],q,q))-
1094                                      (fxdComplxMulScalar32(aReal[0][1],q,aImag[1][0],q,q)+fxdComplxMulScalar32(aImag[0][1],q,aReal[1][0],q,q));
1095
1096      long long int deno=(long long)detReal*detReal+(long long)detImag*detImag;          //Calculating determinant. Adding two FXP of same Q format => Q(q)+Q(q) => Q(q)
1097
1098      float invDetR=(float)detReal/deno;                                                 // Inverse Determinant Q(-q)
1099      float invDetI=(float)(-1*detImag)/deno;
1100
1101      int detR=fixFloat(invDetR,2*q);                                                    // Q(-q)*Q(2q)=>Q(q) Converting Inverse Determinant back to Q(q) format
1102      int detI=fixFloat(invDetI,2*q);
1103
1104
1105      invReal[0][0]=(fxdComplxMulScalar32(detR,q,aReal[1][1],q,q))-(fxdComplxMulScalar32(detI,q,aImag[1][1],q,q));        //inv[0][0]=(invdet)*a[1][1]  => Q(q)*Q(q)/Q(q) =>Q(q)
1106      invImag[0][0]=(fxdComplxMulScalar32(detR,q,aImag[1][1],q,q))+(fxdComplxMulScalar32(detI,q,aReal[1][1],q,q));
1107
1108
1109
1110      invReal[0][1]=(fxdComplxMulScalar32(detR,q,-1*aReal[0][1],q,q))-(fxdComplxMulScalar32(detI,q,-1*aImag[0][1],q,q));  //inv[0][1]=(invdet)*-a[0][1]  => Q(q)*Q(q)/Q(q) =>Q(q)
1111      invImag[0][1]=(fxdComplxMulScalar32(detR,q,-1*aImag[0][1],q,q))+(fxdComplxMulScalar32(detI,q,-1*aReal[0][1],q,q));
1112
1113
1114      invReal[1][0]=(fxdComplxMulScalar32(detR,q,-1*aReal[1][0],q,q))-(fxdComplxMulScalar32(detI,q,-1*aImag[1][0],q,q));  //inv[1][0]=(invdet)*-a[1][0]  => Q(q)*Q(q)/Q(q) =>Q(q)
1115      invImag[1][0]=(fxdComplxMulScalar32(detR,q,-1*aImag[1][0],q,q))+(fxdComplxMulScalar32(detI,q,-1*aReal[1][0],q,q));
1116
1117
1118      invReal[1][1]=(fxdComplxMulScalar32(detR,q,aReal[0][0],q,q))-(fxdComplxMulScalar32(detI,q,aImag[0][0],q,q));      //inv[1][1]=(invdet)*a[0][0]  => Q(q)*Q(q)/Q(q) =>Q(q)
1119      invImag[1][1]=(fxdComplxMulScalar32(detR,q,aImag[0][0],q,q))+(fxdComplxMulScalar32(detI,q,aReal[0][0],q,q));
1120
1121  }
1122
1123
1124
```

```
int fxdComplxMulScalar32(int num1,int q1,int num2,int q2,int resq){
    long long temp=(long long) num1*num2;
    int res;
    res=temp/(1<<(q1+q2-resq));
    return res;
}
```

```
1155  int fixFloat(float value, int q)
1156  {
1157      return value*(long long)(1<<q);
1158  }
```

Result

C code output

```
 2x2 Random Matrix:-
0.730270+0.191808i,0.354645+0.286713i,
0.595405+0.780220i,0.939061+0.501499i,

 Converted to fixed point (Q=15):-
23929+6285i,11621+9395i,
19510+25566i,30771+16433i,

 Fixed point 2x2 Inverse
54127+18398i,-21288-12106i,
-38343-36160i,40366+3805i,

 Converting Fixed Point Inverse to Float values:-
1.651825+0.561462i,-0.649658-0.369446i,
-1.170135-1.103516i,1.231873+0.116119i,
```

Matlab Verification

```
A =

   0.7303 + 0.1918i   0.3546 + 0.2867i
   0.5954 + 0.7802i   0.9391 + 0.5015i

>> inv(A)

ans =

   1.6519 + 0.5615i  -0.6497 - 0.3694i
  -1.1702 - 1.1035i   1.2319 + 0.1161i

>>
```

THANK YOU!