Floating-point to Fixed-point conversion

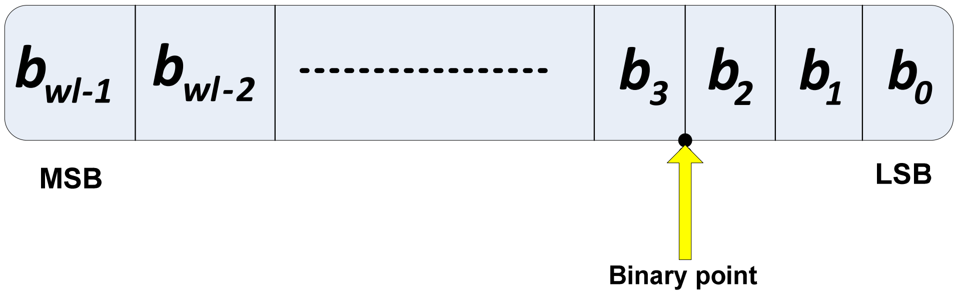
Fixed-Point Data Types

In a digital hardware, numbers are stored in binary words. A binary word is a fixed-length sequence of bits (1's and 0's). How hardware components or software functions interpret this sequence of 1's and 0's is defined by the **data type**. Binary numbers are represented as either **fixed-point** or **floating-point** data types. In order to implement an algorithm such as communication algorithms, the algorithm should be converted to the fixed-point domain and then it should be described with **Hardware Description Language (HDL)**. In HDL coding process, it is necessary to indicate the size of the variables and registers. The registers should be large enough to represent the value of parameters with the desired precision.

Fixed-point data type helps us to know what happens in the hardware. In the other words when an algorithm is represented in floating-point domain, all of the variables have 64 bits(in MATLAB programming). So all of the operations are done with large number of bits. We know that it is impossible to implement an algorithm with large number of flip flops. Because large number of flip flops need a larger area, and more power consumption. In order to solve this problem the algorithm should be converted to the fixed-point domain. In the fixed-point domain a pair ***(W,F)*** is considered for each of the parameters in the algorithm, where ***W*** is the word length of the parameters and ***F*** is the fractional length of the parameters. It is obvious that larger ***W*** and ***F*** results in a better performance and lower bit error rate (BER) but the design needs a large silicon area. On the other hand smaller ***W*** and ***F*** result in a larger BER but less area. So we should choose suitable values of **(*W,F*)** for each parameter in the algorithm. For this reason a simulation should be ran for the algorithm to get the *dynamic range* of the parameters. Simulation results indicate the dynamic range of the variables and the number of bits for ***W*** and ***F***, which are used to represent the variables with the desired precision.

According to the previous section, a fixed-point data type is characterized by the *word length* in bits, the position of the *binary point*, and whether it is *signed* or *unsigned*. The position of the binary point is the means by which fixed-point values are scaled and interpreted.

For example, a binary representation of a generalized fixed-point number (either signed or unsigned) is shown below:



Where:

b_i is the *ith* binary digit

wl is the word length in bits

b_{wl-1} is the location of the most significant, or highest, bit (MSB)

b_0 is the location of the least significant, or lowest, bit (LSB).

The binary point is shown three places to the left of the LSB. In this example, therefore, the number is said to have three fractional bits, or a fraction length of three.

Fixed-point data types can be either *signed* or *unsigned*. Signed binary fixed-point numbers are typically represented in one of these ways:

- *Sign/magnitude*
- *One's complement*
- *Two's complement*

Two's complement is the most common representation of signed fixed-point numbers and is the only representation used by *Fixed-Point Toolbox* in **MATLAB**.

Fixed-point numbers can be encoded according to the following scheme:

$$\boldsymbol{Real-value = 2^{-fractional-length} \times stored\ integer} \qquad (1)$$

where $stored\ integer$ is the raw binary number, in which the binary point assumed to be at the far right of the word.

Conversion of an algorithm from floating-point domain to fixed-point domain can be done through the MATLAB fixed-point toolbox.

Fixed-Point Toolbox provides fixed-point data types in MATLAB and enables algorithm development by providing fixed-point arithmetic. Fixed-Point Toolbox enables you to create the following types of objects:

- ***fi*** — Defines a fixed-point numeric object in the MATLAB workspace. Each *fi* object is composed of value data, a *fimath* object, and a *numerictype* object.
- ***fimath*** — Governs how overloaded arithmetic operators work with *fi* objects
- ***fipref*** — Defines the display, logging, and data type override preferences of *fi* objects
- ***numerictype*** — Defines the data type and scaling attributes of *fi* objects
- ***quantizer*** — Quantizes data sets

Normally complicated algorithms have many variables so the number of fixed-point objects grows significantly. Moreover, in some cases a long time simulation is needed to obtain the BER curves of the algorithm. In the above cases fixed-point simulation with MATLAB fixed-point toolbox needs a large amount of memory, time, and CPU usage and in most of the cases it will crash.

In order to solve the above problem a simple method for floating-point to fixed-point conversion is proposed in this tutorial. Simulation results with this method and simulation results with the MATLAB fixed-point toolbox are the same, but the simulation with the proposed method is significantly faster than the other. For example one iteration of *K-Best algorithm* simulation with MATLAB fixed-point toolbox, takes 237 seconds but simulation with the proposed method, needs only 36 seconds. So in a long-time simulation for example 5000 iteration MATLAB fixed-point toolbox doesn’t work well.

Floating-point to Fixed-point conversion:

In this part a simple method for floating-point to fixed-point conversion will describe. Then we consider the various arithmetic operations and mention a lot of examples for them and finally compare their results with the results of MATLAB fixed-point toolbox.

In order to convert a floating-point value to the corresponding fixed-point vlaue use the following steps.

Consider a floating-point variable, a :

Step 1: Calculate $b = a \times 2^F$, where *F* is the fractional length of the variable. Note that b is represented in decimal.

Step 2: Round the value of b to the nearest integer value. For example :

$round(3.56) = 4$

$round(-1.9) = -2$

$round(-1.5) = -2$

Step 3: Convert b from decimal to binary representation and name the new variable c.

Step 4: Now, we assume that c , needs n bits to represent the value of b in binary. On the other hand we obtain the values of *W* and F, from the simulation. So the value of *W* should be equal or larger than n. If Small value is chosen for W, we should truncate c. If W is larger than n, (W- n) zero-bits add to the leftmost of c.

Now consider the simulation is ran carefully and suitable values of (W,F) are obtained. It means that W is equal or larger than n. So (W- n) zero are added to leftmost of c. Then we select F bits of c from position 0 to F-1 as the fractional part of the fixed-point variable. Therefore the conversion from floating-point to fixed-point is finished by finding the position of binary point in c. In order to verify the result, we can do the same conversion with MATLAB fixed-point toolbox. The results of both methods are the same, but the proposed method is faster. Because in MATLAB method we should call a large number of fixed-point functions and fixed-point objects, which are time consuming and they need a large memory.

In the following section various examples are mentioned for different arithmetic operation such as addition, subtraction, multiplication, and norm. In each case the operation is done through the both methods and shown that the results are the same.

Note:

- *In the following examples **"Method 1"** shows the MATLAB fixed-point toolbox and **"Method 2"** shows the above method.*
- *The dot in the binary representation is used to separate the fractional part and the integer part of the variable. But it isn't a part of the variable.*

Example 1)

This example shows that the value of (W,F) should choose carefully from the simulation (according to the dynamic range of variables).

Method 1: $fi\ (3.613,1,7,4) = 3.625 \xrightarrow{\text{convert to binary with bin()}} 011.1010$

$fi\ (3.613,1,10,7) = 3.6094 \xrightarrow{\text{convert to binary with bin()}} 011.1001110$

$fi\ (3.613,1,15,12) = 3.613 \xrightarrow{\text{convert to binary with bin()}} 011.100111001111$

Example 2)

This example shows the conversion of a floating-point value to fixed-point value and then find the corresponding binary value and finally shows the conversion of a binary value to corresponding real-value by (1).

Method 1:

$$fi\,(3.613,1,15,12) = 3.613 \xrightarrow{\text{convert to binary with bin()}} 011.100111001111 \qquad (W,F) = (15,12)$$

$$(011100111001111)_b = (14799)_d \xrightarrow{\text{convert to decimal by (1)}} 14799 \times 2^{-12} = 3.613$$

Example 3)

This example shows conversion of a floating-point value to corresponding fixed-point value in two methods. Both positive and negative values are covered in this example.

$a = 3.013 \;,\; (W,F) = (8,3)$

Method 1:

$$fi\,(3.013,1,8,3) = 3.00 \xrightarrow{\text{convert to binary with bin()}} 00011.000$$

Method 2:

Step 1: $b = a \times 2^{F} = 3.013 \times 2^{+3} = 24.1040$

Step 2: $round(24.1040) = 24$

Step 3: $c = dec2bin(b) = 11000$

Step 4: $c = 00011.000$

In both methods: $real\ value = integer\ value \times 2^{-F}$

Example 4)

$a = 9.51432\ ,\ (W, F) = (12{,}7)$

Method 1:

$fi\ (9.51432{,}1{,}12{,}7) = 9.5156 \quad \xrightarrow{\text{convert to binary with bin()}} \quad 01001.1000010$

Method 2:

Step 1: $b = a \times 2^F = 9.51432 \times 2^{+7} = 1217.8329$

Step 2: $round(1217.8329) = 1218$

Step 3: $c = dec2bin(b) = 010011000010$

Step 4: $c = 01001.1000010$

Example 5)

$a = -9.0514\ ,\ (W, F) = (14{,}9)$

Method 1:

$fi\ (-9.0514\ {,}1{,}14{,}9) = -9.0508 \quad \xrightarrow{\text{convert to binary with bin()}} \quad 10110.111100110$

Method 2:

Step 1: $b = a \times 2^F = -9.0514\ \times 2^{+9} = -4634.3$

Step 2: $round(-4634.3) = -4634$

Step 3: $c = dec2bin(b) = 10110111100110$

Step 4: $c = 10110.111100110$

Example 6) Multiplication 1

This example shows the conversion of a floating-point multiplication to fixed-point multiplication. In order to perform this conversion:

1st : Each of operands are converted to fixed-point only by step 1 and step 2.

2nd : Perform the multiplication with new values.

3rd : Apply the step 3 and step 4 on the multiplication result.

$a = 3.613\ ,\ \ (W,F) = (8,4)$, $b = 2\ ,\ \ (W,F) = (5,2)$

Note:

(W,F) for the result of multiplication is (13,6).

Method 1:

$d = fi(3.613,1,8,4) = 3.625$, $e = fi(2,1,5,2) = 2$

$mult = d \times e = 7.25 \xrightarrow{\text{convert to binary with bin()}} c = 0000111.010000$

Note:

Note that if the multiplication is performed before fixed-point conversion, the result will be different with the above result. It is better to perform fixed-point conversion for each operand, then perform the operation.

Method 2:

Step 1: $d = a \times 2^F = 3.613 \times 2^{+4} = 57.808$

Step 2: $round(57.808) = 58$

Step 1: $e = b \times 2^F = 2 \times 2^{+2} = 8$

Step 2: $round(8) = 8$

$c = a \times b$

$mult = round(d) \times round(e) = 58 \times 8 = 464$

Step 3: $c = dec2bin(mult) = 0111010000$

Step 4: $c = 0000111.010000$

Example 7) Multiplication 2.

$a = 2.13\,, (W, F) = (8{,}5\,)$, $b = 3.2456\,, (W, F) = (12{,}9)$

Note:

(W,F) for the result of multiplication is (20,14).

Method 1:

$d = fi(2.13{,}1{,}8{,}5) = 2.125$, $e = fi(3.2456{,}1{,}12{,}9) = 3.2461$

$mult = d \times e = 6.8979 \xrightarrow{\text{convert to binary with bin()}} c = 000110.11100101111000$

Method 2:

Step 1: $d = a \times 2^{F} = 2.13 \times 2^{+5} = 68.16$

Step 2: $round(68.16) = 68$

Step 1: $e = b \times 2^{F} = 3.2456 \times 2^{+9} = 1661.7472$

Step 2: $round(1662) = 1662$

$c = a \times b$

$mult = round(d) \times round(e) = 68 \times 1662 = 113016$

Step 3: $c = dec2bin(mult) = 011011100101111000$

Step 4: $c = 000110.11100101111000$

Example 8) Addition.1

This example shows the conversion of a floating-point addition to fixed-point addition. In order to perform this conversion:

1st : Align the binary point of operands by adding zero in the right side of the operand, which has smaller fractional length.

2nd : Each of operands are converted to fixed-point only by step 1 and step 2.

3rd : Perform the addition with new values.

4th : Apply the step 3 and step 4 on the addition result.

$a = 3.613\ ,(W,F) = (7,3\ )$, $b = 2.3\ ,(W,F) = (7,2)$

Note:

It is necessary to consider one bit for carry. So the word length of the addition result is the larger word-length of operands plus one. The fractional-length of the addition is the larger fractional-length of operands. So the step 1 is done with final fractional-length (fractional-length of addition). Therefore in this example *(W,F)* of addition is equal to (8,3).

Method 1:

$d = fi(3.613,1,7,3) = 3.625$, $e = fi(2.3,1,7,2) = 2.25$

$add = d + e = 5.8750 \xrightarrow{\text{convert to binary with bin()}} c = 00101.111$

Method 2:

Step 1: $d = a \times 2^F = 3.613 \times 2^{+3} = 28.904$

Step 2: $round(28.904) = 29$

Step 1: $e = b \times 2^F = 2.3 \times 2^{+3} = 18.4$

Step 2: $round(18.4) = 18$

$c = a + b$

$add = round(d) + round(e) = 29 + 18 = 47$

Step 3: $c = dec2bin(add) = 101111$

Step 4: $c = 00101.111$

Example 9) Addition.2

This example shows the different between the following two ways in fixed-point simulation:

a- Perform the operation in floating-point domain and then convert the result to the fixed-point domain.

b- Convert the operands to the fixed-point domain and then perform the operation in fixed-point domain.

In order to show this note the Example8, which is done with the second way is performed again in the first way.

In order to have an efficient fixed-point simulation, it is necessary to perform the second way.

1st way:

$add = (3.613 + 2.3) = 5.913$,

$add\_fi = fi(add, 1,7,2) = 6 \xrightarrow{\text{convert to binary with bin()}} c = 00110.00$

2nd way:

$d = fi(3.613,1,7,2) = 3.5$, $e = fi(2.3,1,7,2) = 2.25$

$add = d + e = 5.75 \xrightarrow{\text{convert to binary with bin()}} c = 00101.11$

Example 10) Addition.3

$a = -9.613\ , (W, F) = (10,5\ )$, $b = -3.421\ , (W, F) = (8,5)$

Method 1:

$d = fi(-9.613,1,10,5) = -9.625$, $e = fi(-3.421,1,8,5) = -3.4063$

$add = d + e = -13.0313$ $\xrightarrow{\text{convert to binary with bin()}}$ $c = 110010.11111$

(W,F)=(11,5)

Method 2:

Step 1: $d = a \times 2^F = -9.613 \times 2^{+5} = -307.616$

Step 2: $round(-307.616) = -308$

Step 1: $e = b \times 2^F = -3.421 \times 2^{+5} = -109.472$

Step 2: $round(-109.472) = -109$

$c = a + b$

$add = round(d) + round(e) = (-308) + (-109) = -417$

Step 3: $c = dec2bin(add) = 11001011111$

Step 4: $c = 110010.11111$

Example 11) Addition.4

$a = -9.613\ , (W, F) = (10{,}5\ )$, $b = +3.421\, , (W, F) = (8{,}5)$

Method 1:

$d = fi(-9.613{,}1{,}10{,}5) = -9.625$, $e = fi(+3.421{,}1{,}8{,}5) = 3.4063$

$add = d + e = -6.2188$ $\xrightarrow{\text{convert to binary with bin()}}$ $c = 111001.11001$

(W,F)=(11,5)

Method 2:

Step 1: $d = a \times 2^{F} = -9.613 \times 2^{+5} = -307.616$

Step 2: $round(-307.616) = -308$

Step 1: $e = b \times 2^{F} = 3.421 \times 2^{+5} = 109.472$

Step 2: $round(109.472) = 109$

$c = a + b$

$add = round(d) + round(e) = (-308) + (109) = -199$

Step 3: $c = dec2bin(add) = 11100111001$

Step 4: $c = 111001.11001$

Example 12) Addition.5

$a = +9.613 \ , (W, F) = (10{,}5\ )$, $b = -3.421\ , (W, F) = (8{,}5)$

Method 1:

$d = fi(+9.613{,}1{,}10{,}5) = +9.625$, $e = fi(-3.421{,}1{,}8{,}5) = -3.4063$

$add = d + e = +6.2188$ $\xrightarrow{\text{convert to binary with bin()}}$ $c = 000110.00111$

(W,F)=(11,5)

Method 2:

Step 1: $d = a \times 2^F = +9.613 \times 2^{+5} = +307.616$

Step 2: $round(+307.616) = 308$

Step 1: $e = b \times 2^F = -3.421 \times 2^{+5} = -109.472$

Step 2: $round(-109.472) = -109$

$c = a + b$

$add = round(d) + round(e) = (+308) + (-109) = +199$

Step 3: $c = dec2bin(add) = 00011000111$

Step 4: $c = 000110.00111$

Example 13) Norm calculation

This example shows the conversion of a floating-point norm calculation to a fixed-point norm calculation.

$a = 3.25 + 4.26i \ \ ,(W,F) = (8,4)$

Method 1:

$b = fi(3.25 + 4.26i\,, 1, 8, 4) = 3.2500 \ + \ 4.2500i$

$c = abs(b) = 5.3750 \quad \xrightarrow{\text{convert to binary with bin()}} \quad bin(c) = 0101.0110$

Method 2:

Step 1: $d = Re\{b\} \times 2^F = 3.25 \times 2^{+4} = 52$

$e = Im\{b\} \times 2^F = 4.26 \times 2^{+4} = 68.16$

Step 2: $round(52) = 52$

$round(68.16) = 68$

Step 3: $f = abs(52 + 68i) = 85.6037$

Step 4: $round(85.6037) = 86$

Step 5: $dec2bin(86) = 01010110$

Step 6: $g = 0101.0110$

Note:

In the hardware implementation the norm operation is done by CORDIC. So in an efficient fixed-point conversion it is better to replace the corresponding command (i.e. abs()) with ***CORDIC****. But in the above code the difference between them is negligible.*

Floating-point to fixed-point conversion of an algorithm

In this section conversion of an algorithm from the floating-point to the fixed-point is shown. So a simple code is converted from the floating-point domain to the fixed-point domain.

The corresponding equation, which is described in the following MATLAB codes is:

$$Partial\ \boldsymbol{E}uclidean\ \boldsymbol{D}istance(\boldsymbol{PED}) = \sum_{j=1}^{j=N_T} |Z - RCS|^2$$

Method1:

```
function PED = FixedPED2(R, S, C, Z);

R_fi = fi(R,1,12,10);     %fi-object definitions
C_fi = fi(C,1,14,13);
S_fi = fi(S,1,4,0);
Z_fi = fi(Z,1,16,12);

RCS = R*S*C;             %The corresponding floating-point operation
RCS_fi = R_fi*C_fi*S_fi; %Perform the multiplication in fixed-point
domain
RCS_fi_ = fi(RCS_fi,1,16,12); %Limit the (W,F) of the result

PED_inter1 = Z-RCS;      %The corresponding floating-point operation
PED_inter1_fi = Z_fi-RCS_fi_;
PED_inter1_fi_ = fi(PED_inter1_fi,1,16,12);  %Limit the (W,F) of the
result
PED_inter2_fi = abs(PED_inter1_fi_);     %Perform the norm calculation

for j=1:length(R(:,1))      %Calculate the power  operation
    PED_inter3_fi(j,1)=PED_inter2_fi(j,1)*PED_inter2_fi(j,1);
end
FF=fimath;
PED_inter4_fi = fi(PED_inter3_fi,1,16,12);   %Limit the (W,F) of the
result
PED = fi(sum(PED_inter4_fi),1,16,12);  %Perform the Sum operation
```

NOTE:

In order to perform the summation operation in the above equation you can call the above function (i.e. FixedPED2) in a loop with a proper value for the loop counter, which is N_T in this equation. This process doesn't affect on your fixed-point conversion.

Method2:

```
function PED = FixedPED3(R, S, C, Z);

R_Frac=8;           %The Fractional Length and
R_WordLength=12;    %The Word Length of the parameters (W,F)
S_Frac=0;
S_WordLength=4;
C_Frac=14;
C_WordLength=15;
Z_Frac=12;
Z_WordLength=16;

RCS_Frac=R_Frac+S_Frac+C_Frac;
% PED_inter1_Frac=max(Z_Frac,RCS_Frac);

R_fi0=R*2^R_Frac;              %Step1 in the Method2
S_fi0=S*2^S_Frac;
C_fi0=C*2^C_Frac;
Z_fi0=Z*2^Z_Frac;

R_fi=round(R_fi0);             %Step2 in the Method2
S_fi=round(S_fi0);
C_fi=round(C_fi0);
Z_fi=round(Z_fi0);

RCS_fi = R_fi*S_fi*C_fi;       %Performing the multiplication
RCS_fi1=RCS_fi*2^(-RCS_Frac);  %Calculation of the real-value of the
                                RCS_fi1 by (1)
RCS = R*S*C;                   %The corresponding floating-point
                                operation


RCS_Frac = Z_Frac;             %Equalize the Fractional Length of the
                                two operands
RCS_fi2 = RCS_fi1 *2^(RCS_Frac);    %Step1 in the Method2
RCS_fi3 = round(RCS_fi2);           %Step2 in the Method2


if(RCS_Frac<Z_Frac)         %The two operands of the addition,should
                             have the same Fractional length.
    RCS_fi4=RCS_fi3*2^(Z_Frac-RCS_Frac);
    Z_fi1=Z_fi;                     %In general This condition is
                                     %used to equalize the fractional
else          %(RCS_Frac>=Z_Frac)    %length of the two operands.
    Z_fi1=Z_fi*2^(RCS_Frac-Z_Frac);  %But in this code, in the
                                      %previous lines this action is
    RCS_fi4=RCS_fi3;                 %done with "RCS_Frac = Z_Frac;"
end
```

```
PED_inter1_fi = Z_fi1-RCS_fi4;
PED_inter1_Frac = Z_Frac;         %Update the fractional length of
                                  %the result of subtraction
PED_inter1 = Z-RCS;               %The corresponding floating-point
                                  %operation

for j=1:length(R(:,1))
    PED_inter2_fi(j,1)=abs(PED_inter1_fi(j,1));  %Performing the norm
                                                 %calculation
    PED_inter2(j,1)=abs(PED_inter1(j,1));     %The corresponding
                                              floating-point operation
end

PED_inter2_Frac=PED_inter1_Frac;   %Update the fractional length of
                                   %the result of norm calculation

PED_inter3_fi=PED_inter2_fi.^2;    %Performing the power operation
PED_inter3=PED_inter2.^2;          %The corresponding floating-point
                                            %operation

PED_inter3_Frac=PED_inter2_Frac*2;    %Update the fractional length of
                                      %the result of power calculation

PED_inter4_fi=PED_inter3_fi*2^(-PED_inter3_Frac); %Calculation of the
                                real-value of the PED_inter4_fi by (1)

PED_inter4_Frac=PED_inter3_Frac-8;    %Update the fractional length
                                      for the next step
                            %NOTE:
                                  %If the fractional length of the
                                  %the intermediate variables
                                  %increase significantly, we can
                                  %limit it with the following method.
                            %Important NOTE:
                                  %Calculation of the real value
                                  %is done with the old F, but the
                                  %step1 of Method2 is
                                  %done with the new F.

PED_inter5_fi=PED_inter4_fi*2^(PED_inter4_Frac); %Step1 in the Method2
PED_inter6_fi=round(PED_inter5_fi);              %Step2 in the Method2
PED_inter6_Frac=PED_inter4_Frac;        %Update the fractional length
                                        %for the next step

PED1 =sum(PED_inter6_fi);               %Perform the sum operation
```

```
PED1_Frac=PED_inter6_Frac;           %Update the fractional length for
                                     %the next step
PED=PED1*2^(-PED1_Frac);             %Calculation of the real-value of
                                     %the PED by (1)
PED0 = sum(PED_inter3);   %The corresponding floating-point  operation
```