**Lab3**

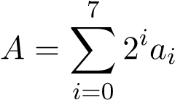
# Numeric Formats

## 2.1 Introduction

A collection of N binary digits (i.e. bits) has 2*N* possible states. To represent an unsigned binary number of 8 bits

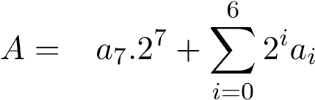
*a*7*a*6*a*5*a*4*a*3*a*2*a*1*a*0 (2.1)

as an integer A, we incorporate the following formula.

 (2.2)

The addition and subtraction of binary numbers is straight forward, however, by adding two N bits numbers results in N+1 bits generally. Similarly the multiplication of two N bit numbers result in 2N numbers. This result cannot be represented in N bits and the results are catastrophic in a DSP.

To represent negative numbers, we use the 2’s complement method. Using a signed number of N bits, we can represent integers from 2*N* 1 to 2*N* 1 1. An eight bit binary signed number can be converted into integer using the following formula.

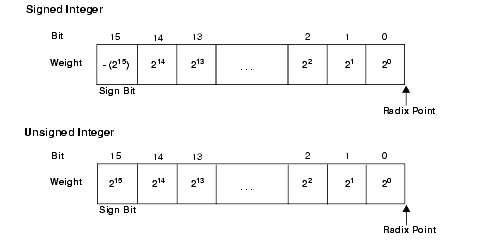
 (2.3)

ADSP-BF53x/BF56x Blackfin family processors support 8-, 16-, 32-, and 40-bit fixedpoint data in hardware. Special features in the computation units allow support of other formats in software. In this lab we will discuss various aspects of these data formats.

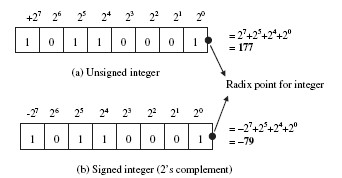
### 2.1.1 Fixed Point Format

The value of an unsigned integer is interpreted in the usual binary sense as they are positive, and no sign information is contained in the bits. The least significant words of multipleprecision numbers are treated as unsigned numbers. Signed numbers supported by the ADSPBF53x/BF56x Blackfin family are in twos-complement format.

The fixed point data is supported in both integer and fractional format. In an integer, the radix point is assumed to lie to the right of the least significant bit (LSB), so that all magnitude bits have a weight of 1 or greater. This format is shown in Fig. 2.1. Note in twos-complement format, the sign bit has a negative weight. An example representation is shown in Fig. 2.2.



**Figure 2.1:** Integer format



**Figure 2.2:** An example of integer format

### 2.1.2 Fractional Format

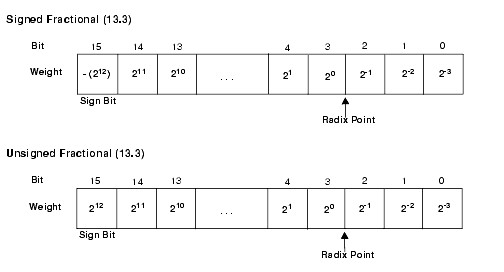
In a fractional format, the assumed radix point lies within the number, so that some or all of the magnitude bits have a weight of less than 1. In the format shown in Fig. 2.3, the assumed radix point lies to the left of the three LSBs, and the bits have the weights indicated.

The native formats for the Blackfin processor family are a signed fractional 1*.M* format and an unsigned fractional 0*.N* format, where *N* is the number of bits in the data word and

M = N - 1.

The notation used to describe a format consists of two numbers separated by a period (.); the first number is the number of bits to the left of the radix point, the second is the number of bits to the right of the radix point. For example, 16.0 format is an integer format; all bits lie to the left of the radix point. The format in Fig. 2.3 is 13.3.

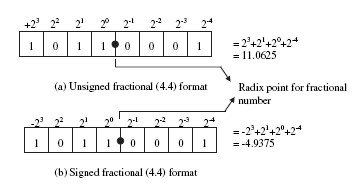
For example, a radix point is positioned to the left of the four LSBs with the weights indicated in Fig. 2.4(a) for the unsigned fractional number and in Fig. 2.4(b) for the signed fractional number. Note that the number to the right of the radix point assumes a fractional



**Figure 2.3:** Examples of 8-bit binary data format for integer numbers of 177 (a) and -79

(b)

binary bit, with a weighting of 2 *p* where p = 1, 2, 3, and 4. In this case, the lowest fractional increment is 2 4 (or 0.0625). For the number to the left of the radix point, the weighting increases from 2*q* where *q* = 0*,*1*,*2*,* and 3. The weighting of the MSB (or sign bit) depends on whether the number is signed or unsigned.

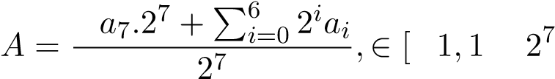


**Figure 2.4:** Example of 8-bit binary data formats for a fractional number

As stated earlier, the product and addition of tow N bit numbers may not result in an N bit number, and be sure for the multiplication that it will never! A DSP has to use complicated algorithms to tackle such situations. The problem can be easily handled by using numbers in the range of -1 and 1, because the product of two numbers in the range [-1,1] always result in a number in the range [-1,1].

In 2’s complement fractional representation, an N bit binary number can represent numbers from 

So an 8 bit binary word can be represented as a fractional number by the following formula.

] (2.4)

This representation is sometimes referred to Q-format. As seen, the fractional representation does not change the MSB, rather some other digits are applied after the MSB. So an N bit binary word with the MSB as signed bit can represent fraction using the other N-1 bits. The number is said to have Q-(N-1) format. Therefore, in (2.4), A is a Q-7 number.

### 2.1.3 Addition and Subtraction

Care must be taken in adding two binary numbers. Say we have two Q-15 numbers and the result of their addition may not be in the range and could not be represented correctly by Q-15 format. Simply to say the number is not in the range [ 1*,*1 215]. This is known as overflow. The overflows give erroneous results and extreme care must be taken to when implementing DSP algorithms.

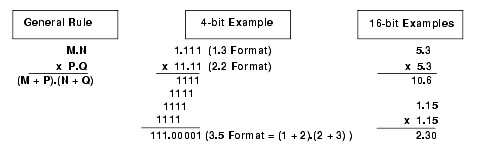
To tackle overflow, one way could be to scale the numbers and make them small enough so that their result is always in the range [-1,1). Another way to prevent overflow, and underflow, is to use saturation. In saturation, the number in the range closest to the result is used to represent the number.

### 2.1.4 Binary Multiplication

The result of multiplication of two Q-N format numbers result in a Q-2N format number, a predicament that the DSP has to solve via complex manipulation and functions. The conversion of data to decimal fractional format can handle this problem.

In addition and subtraction, both operands must be in the same format (signed or unsigned, radix point in the same location), and the result format is the same as the input format. Addition and subtraction are performed the same way whether the inputs are signed or unsigned.

In multiplication, however, the inputs can have di↵erent formats, and the result depends on their formats. The ADSP-BF53x/BF56x Blackfin family assembly language allows you to specify whether the inputs are both signed, both unsigned, or one of each (mixed-mode). The location of the radix point in the result can be derived from its location in each of the inputs. This is shown in Fig. 2.5. The product of two 16-bit numbers is a 32-bit number. If the inputs formats are *M.N* and *P.Q*, the product has the format (*M* + *P*)*.*(*N* + *Q*). For example, the product of two 13.3 numbers is a 26.6 number. The product of two 1.15 numbers is a 2.30 number.



**Figure 2.5:** Multiplication of two fixed point numbers

A product of 2 twos-complement numbers has two sign bits. Since one of these bits is redundant, you can shift the entire result left one bit. Additionally, if one of the inputs was a 1.15 number, the left shift causes the result to have the same format as the other input (with 16 bits of additional precision). For example, multiplying a 1.15 number by a 5.11 number yields a 6.26 number. When shifted left one bit, the result is a 5.27 number, or a 5.11 number plus 16 LSBs.

The ADSP-BF53x/BF56x Blackfin family provides a means (a signed fractional mode) by which the multiplier result is always shifted left one bit before being written to the result register. This left shift eliminates the extra sign bit when both operands are signed, yielding a result that is correctly formatted.

When both operands are in 1*.*15 format, the result is 2*.*30 (30 fractional bits). A left shift causes the multiplier result to be 1*.*31 which can be rounded to 1*.*15. Thus, if you use a signed fractional data format, it is most convenient to use the 1*.*15 format.

### 2.1.5 Division in DSP Algorithms

Sometimes in DSP algorithms, division is involved extensively, for example, in a loop that runs 10,000 times, some statement like

*a* = *b/*78; (2.5)

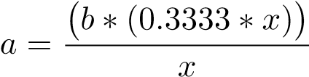
can seriously degrade the performance of the algorithm. This is because a division

* takes longer execution time than multiplication
* and it cannot be pipelined

Hence, some replacement of division is required. For division by a some power of 2, we just need to shift the number to the right by power number of bits. As an example, consider the statement *a* = *b/*128. The division can be easily replaced by a statement such as *a* = *b >>* 7 since 27 = 128.

However, when the denominator is not a power of 2, some complexity is involved. Let us discuss a case when *a* = *b/*3. Of course 3 cannot be represented by a power of 2. Therefore, shifting will not help. But note that. Multiplication is a less expensive operator than division, but 0*.*3333*...* has some problems of its own. Firstly, floating point is involved, which makes it di cult for a fixed point DSP in terms of e cient implementation. Also, floating point operations are slower than fixed point. Therefore, the above statement is of little use.

So there is a clear a need of transferring the floating point operation to the fixed point. And to do so, what we can do is that replace 0*.*3333 by its integer representation. Now how do we convert 0*.*3333 to an integer? The most easiest method of conversion is multiplication by a factor, say *x*. I.e. the statement is converted to

 (2.6)

Since at the end of the expression, there is a division by *x* involved, therefore, it is quite intuitive to make *x* some power of 2.

Let us take *x* = 28 = 256. Then the expression is replaced by

*a* = (*b* ⇤ 85) *>>* 8; (2.7)

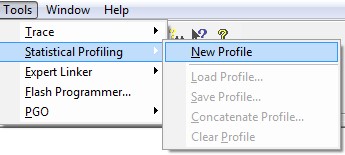
Shifting to the right always produces the result rounded towards 1. So 80 divided by 3 will always result in 26 rather than 27 if we use 2.7.

## 2.2 Practical

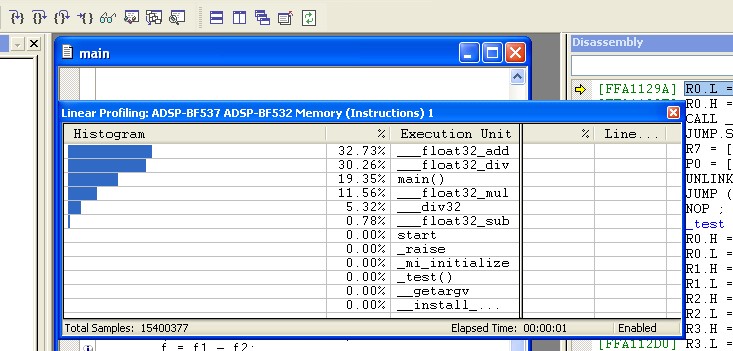
In VisualDSP++ 4.5, write a program that computes addition, subtraction, multiplication and division on two number of type float and fract16, 20000 times. Study the Linear Profile of the processes.

## 2.3 Hint

For viewing the linear profile, click the Tools dropdown menu and select New Profile in the Linear Profile submenu (see Fig. 2.6). The screen shot after the execution of the code should look something like Fig. 2.7.



**Figure 2.6:** Viewing the Statistical Profile



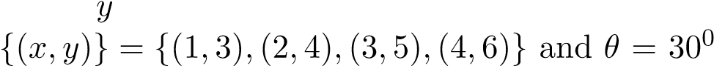
**Figure 2.7:** The statistical Profile of the practical

## 2.4 Questions

* What is the value 16-bit number 0x8888 in the following formats: Binary, Signed Inte-ger, Unsigned Integer, Signed Fractional, Unsigned Fractional?
* What is the 16-bit 2’s complement number representing the same value as the 8-bitnumbers 01001100 and 10001110?
* What is the decimal fractional number corresponding to the Q-7 format binary numbers10010001, 01101100, 11010101 and 00011111?
* What are the drawbacks of scaling?
* For Q-7, perform 11100101+11111000, 00101101+10011101 and 10110011-11101110.Does overflow occur in any of them?
* Consider the following equation



where x is a 5-bit number and the space allocated for y is only 16-bits. What is the most accurate fixed point representation of the equation in VisualDSP++?

* Using fixed point notations in VisualDSP++, perform the following function:*y* = *x/tan✓* where *✓*=1420 and *x* is a 3-bit variable. *y* is stored in 8-bits only.
* Implement the following equation in fixed point format in VisualDSP.*z* = *x*cos*✓* + sin*✓* where.
* Implement a function in VisualDSP that returns a variable y, given the signed 8-bitinput x, such that



Round the computation to +1.