17th Symposium IMEKO TC 4, 3rd Symposium IMEKO TC 19 and 15th IWADC Workshop
Instrumentation for the ICT Era
Sept. 8-10, 2010, Kosice, Slovakia

Simulation of errors in floating-point DSP calculations

Vilmos Pálfi[1], and István Kollár[1]

[1] *Department of Measurement and Information Systems, Budapest University of Technology and Economics, postal address: H-1521 Budapest, Pf. 91., phone: +36 1 463-1774, fax: +36 1 463-4112, e-mail: palfi.vilmos@upcmail.hu; kollar@mit.bme.hu*

Abstract-Quantization errors like ADC errors and roundoff errors are similar in the sense that they are difficult to analyze and predict, and therefore need careful tests/measurements. This paper discusses the possibilities of the simulation of single-precision floating-point roundoff errors. It illustrates the examination of the error and discusses it in one of the most common signal processing procedures, the fast Fourier Transform. It shows a case when spurious spectral peaks appear due to roundoff error.

I. Introduction

In digital signal processing, quantization errors have two main sources: A/D conversion, and roundoff in the computer, usually a digital signal processor. Therefore, to properly characterize the data processing chain, both need to be investigated. The usual problem is, however, that analytic results are rare. Most of these are of statistical nature, and they are based on the noise model of roundoff: for fixed-point, the roundoff is noise-like, with variance $LSB^2/12$, it is independent of the signal, and it is usually white. Floating-point error is less regular, but also has general approximation rules: its variance is about $\mathrm{var}\{v\} \approx 0.180 \cdot 2^{-p} \cdot \mathrm{var}\{x\}$, with p the precision and x the signal ([1], page 257, Eq. (12.24)), and it is uncorrelated with the signal, and it is white. These properties are usually all valid, but need careful experimentation because they are usually only approximations with no guarantee. However, experimentation is not easy: tools are often not available, and if available, they need to be cross-checked because even small deviations from the theory can cause noticeable errors.

II. Possibilities for examination

This subsection tries to sum up the common properties of the different examination methods. To simulate single-precision number representation, the following six methods are available:

- Calculations using variables of class “single” in Matlab,
- Using a roundoff simulation toolbox in Matlab,
- Using an object-based finite precision toolbox in Matlab,
- Using a DSP simulator,
- Using a DSP itself for calculations,
- Using variables of class “float” in C++ language.

We have tried all six, and achieved by proper settings that all of them yield exactly the same results. This will be discussed below. We have not tried to use an extended-precision accumulator, because one of the main aims was to achieve exactly the same simulation results. The six methods above can be divided into two groups: Matlab based methods and C/C++ based methods. First the Matlab based methods will be discussed.

A. Matlab based methods

Using quantization toolboxes in Matlab allows us to define and create floating-point quantizers with almost any bit numbers. When passing higher precision variables to these quantizers they quantize the given data to the defined precision, and return it with the quantized value. To keep compatibility with all the above methods, simple operations (addition, multiplication, e.g.) are also supported with strict rules: every partial result is quantized, no temporary growth of the precision is allowed. This is important because Matlab can handle complex numbers and multiplying complex numbers can be traced back to four real multiplications and two summations. Due to the strict rules after every multiplication the result is quantized, so this can be considered as

the worst case from the error's point of view. Using these quantizers and operations the FFT's algorithm can be implemented and performed with single precision.

Another possibility is to use the built-in Matlab class *single*, which represents numbers in single precision format. These variables can be passed to every function and then the function will be performed with single precision. But using the single class sometimes the precision is higher than we expect. For example, let us see an easy butterfly implementation:

```
function[c, d]=butterfly(a, b, w)
c=a+b*w;
d=a-b*w;
return;
```

Here, sometimes *b*w* multiplications result is not casted to single so its precision is higher than 24 when the summation is proceed. Therefore only one quantization occurs instead of two which is advantageous from the error's point of view, but the question is how well does this behave imitate a DSP's behavior. Replacing the multiplications with the following simple function the roundoff error on the output will equal to the error reached with the toolboxes:

```
function xy=mul(x, y)
re1 = single(real(x) * real(y));
re2 = single(imag(x) * imag(y));
im1 = single(real(x) * imag(y));
im2 = single(imag(x) * real(y));
xy = (re1 - re2) + j * (im1 + im2);
```

With explicit cast operators the growth of the precision is avoidable.

B. C/C++ based methods

These methods mean direct calculations which can only simulate IEEE single precision using the float class. Sensitivity to precision cannot be easily investigated, only the error of the given calculation can be evaluated.

The first step can be to switch off optimization from the C-compiler, but by itself, this is not enough to achieve full control and provide compatibility to other methods. C/C++ can not handle complex numbers by default, so a struct or a class has to be implemented. We used C++ so we implemented the *Complex* class to perform FFT-s. An important part of the class is overloaded multiplication operator:

```
Complex Complex::operator*(const Complex& complex)
{
    Complex(re * complex.re - im * complex.im, re * complex.im + im * complex.re);
}
```

At first sight it is easy to create this class but due to optimizations during compilation precision growth may appear in calculations which affects on the roundoff error. To keep the precision's value 24 in every operation the following modifications are needed:

```
Complex Complex::operator*(const Complex& complex)
{
    float re1, re2, im1, im2;
    re1 = re * complex.re;
    re2 = im * complex.im;
    im1 = re * complex.im;
    im2 = im * complex.re;
    return Complex(re1 - re2, im1 + im2);
}
```

This is very similar to the solution used in Matlab to avoid the increase of the precision. To verify the simulation results the same input was created in C++ and Matlab, the same FFT algorithms were performed, and the errors were compared. We found that despite all the thorough preparations, the error was not the same in the two cases. The difference is caused by a deviation in π's value, and by the trigonometric functions. It is surprising that even

exporting π from Matlab in 16-digit number, and importing to C++, the result is not the same. The reason is the difficulties of implementation import of digital numbers to binary. Furthermore, in Matlab the complex roots of the unity can be created easily thanks to handling complex numbers $\left(W_N^k = \exp(j2\pi \times k / N)\right.$. In C/C++, the *exp*() function is unable to handle a complex parameter, so we have to construct an object from class *Complex* with parameters $\cos(2\pi \times k / N)$, $\sin(2\pi \times k / N)$. About trigonometric functions the IEEE standard only says that the deviation from the exact value must not be larger than 1 LSB, the direction of the deviation is not defined. So there is no guarantee that *Complex* object constructed with $\cos(2\pi \times k / N)$, $\sin(2\pi \times k / N)$ parameters will equal bit by bit to *single* objects in Matlab constructed with the *exp*() function. To avoid the difference caused by the trigonometric functions both the sine input and the complex roots of unity were created in Matlab, then saved into a binary file. The C++ program opened this file and read the needed values, and then the FFT was performed. These preparations were enough to get the same roundoff errors on DSP, DSP simulator and in the C++ program. So the same result was reached in six different ways which we think is a good result because very different platforms were used during the examination.

III. Examination results

A. The Fast Fourier transform

The examination of the roundoff error will be illustrated in the case of Fast Fourier transform so let us shortly sum up the most important properties of this algorithm. It produces the discrete Fourier transform of a sampled digital signal much faster than the original DFT algorithm: instead of N^2 it needs only $N*\log_2(N)$ complex additions/multiplications (where N is the number of samples). In this paper the radix-2 FFT will be discussed which means that N must be the power of two. The FFT reaches its rapidity by rearranging the samples and producing an N point DFT from two, $N/2$ point DFT-s. The radix-2 type of the FFT can be divided into two groups: Decimation In Time (DIT) and Decimation In Frequency (DIF). The difference between them is the way how the samples are rearranged. The roundoff errors will be examined in the case of radix-2 FFT DIT.

Figure 1 shows the basis of the FFT, which is the so-called butterfly. A butterfly has two complex inputs, and produces two complex outputs after a multiplication and summation.

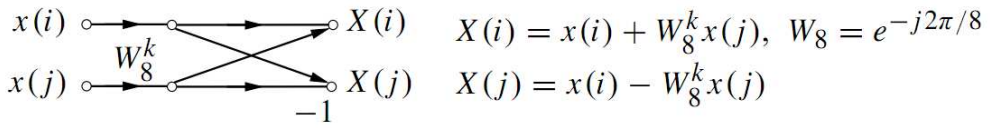

Figure 1. The butterfly and its functionality.

Figure 2 shows the flowchart of the FFT, built up of butterfly elements for N=8. The butterflies are arranged into columns, and these columns are also called as the stages of the FFT. The number of the stages for an N point FFT is $\log_2(N)$.

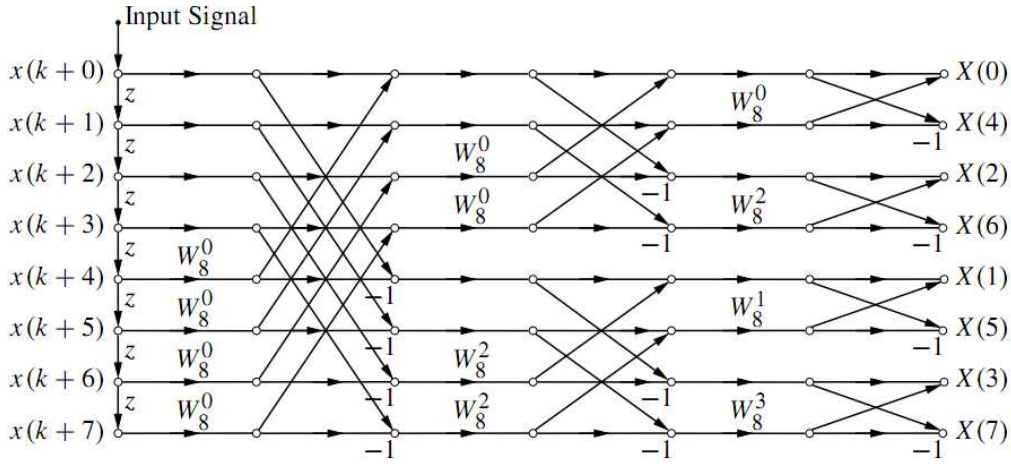

Figure 2. The flowchart of the FFT for N=8.

It is interesting that the output of the FFT is not in the correct order, so it has to be rearranged. The original order of the output is the so-called bit reversed order, this means that writing the indices in binary form and reading them from the right to the left will give the correct index of each output.

B. IEEE 754 single precision number format

In the examinations we analyze the roundoff error of the Fast Fourier transform with IEEE single precision numbers. The IEEE 754 standard specifies two types of single format floating point numbers: the original single precision and the extended single precision. An original single precision number is four bytes long, where 23 bits represent the mantissa, 8 bits represent the exponent and one bit is the sign of the number. Due to a hidden bit (the leading bit 1) the precision is 24. Extended single precision numbers are not specified as exactly as the original ones. The standard says that they contain at least 43 bits, where the mantissa is at least 32 bits long, and the exponent is larger then 10 bits. If two single precision numbers are multiplied in a PC, than the result can be precisely represented in the 80 bits long accumulator. But when the same operation is performed on a DSP there is no guarantee that an 80 bits long accumulator is available, so that's why the extended precision is important.

C. Examination properties

We will look *beyond* the noise model: investigate roundoff errors which deviate from noise. The examinations are built up of simulations. The roundoff error of the Fast Fourier transform is analyzed for a clear sine wave input, because it has a rather regular pattern, according to [3] in this case for fixed-point the roundoff error is expectable to deviate from noise, so we investigated its behavior also for floating-point arithmetic. The basic input signal is $x[k] = \sin(2\pi \times k / 8)$, and the sampling frequency is slightly shifted from coherence. The amount of the frequency shift is given in bins: $x[k] = \sin(2\pi \times (k + \Delta k) / 8)$. For example, if N=1024 point FFT is executed and the basic input signal's frequency is shifted by d=5 bin %, then the input will be $x[k] = \sin(2\pi \times k \times (128.05 / 1024))$ instead of the original $x[k] = \sin(2\pi \times k / 8) = \sin(2\pi \times k \times (128 / 1024))$ where d=0 bin %.

First the input is represented with single precision numbers, and the FFT is performed with single precision. Then the input is casted to double precision, and the result of a double precision FFT is considered as the correct (reference) result. The error is stored in the h vector, where h is:

$$h = \mathrm{abs}(X_{\mathrm{double}} - X_{\mathrm{single}}) \tag{1}$$

In the equation X_{double} and X_{single} are the complex outputs of double and single precision FFT-s, respectively.

As it was mentioned earlier we analyzed the error's pattern and size for a clean sine wave input when the sampling is almost coherent. The deviation from the coherent sampling (also called as frequency shift) is given in bin percent. The basic input signal where the frequency shift is 0 bin % is $x[k] = \sin(2\pi \times k / 8)$. In the simulations N=1024 point FFT-s were executed, so the basic input can be written as $x[k] = \sin(2\pi \times k \times 128 / 1024)$. Now, if the frequency is shifted with d bin %, then the input is $x[k] = \sin(2\pi \times k \times (128 + d) / 1024)$.

D. New results for a clear sine wave

First we studied the pattern of the error for the $x[k] = \sin(2\pi \times k / 8)$ input and found that the error is concentrated in four peaks so the pattern is totally different from our expectations based on the noise model assumption of quantization. Next we checked the pattern of the error for almost coherently sampled sine wave. Figure 3 shows the error on many different frequency shift values. Variable i is the index of simulations, the value of the frequency shift can be ascertained using the formula at the left bottom for each simulation. On small (almost 0) deviations the pattern is very regular. As the frequency shift increases the error appears on more outputs and sometimes the error peaks are unexpectedly high. Unfortunately these high peaks can be far away from the sine wave's frequency, so they may be considered as a small sine wave (See Figure 4 and Figure 5). If we compare Fig. 4 to Fig. 5 it is easy to see that in the single precision case due to roundoff errors around k=400 and k=600 peaks are above -80 dB although the double precision result shows that these peaks should be under -100 dB.

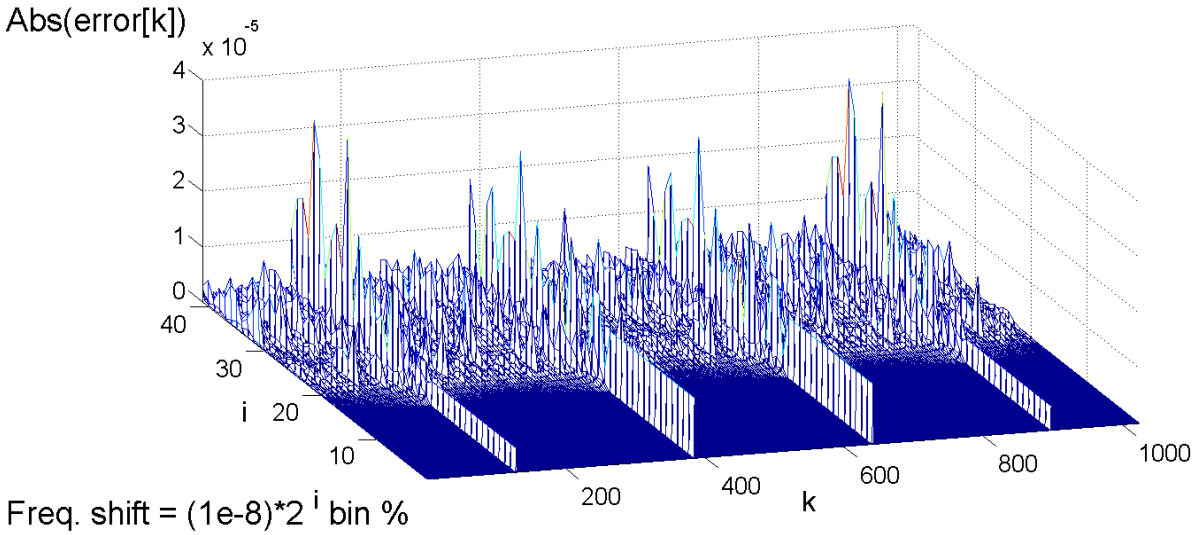

Figure 3. Error's pattern on different frequency shift values.

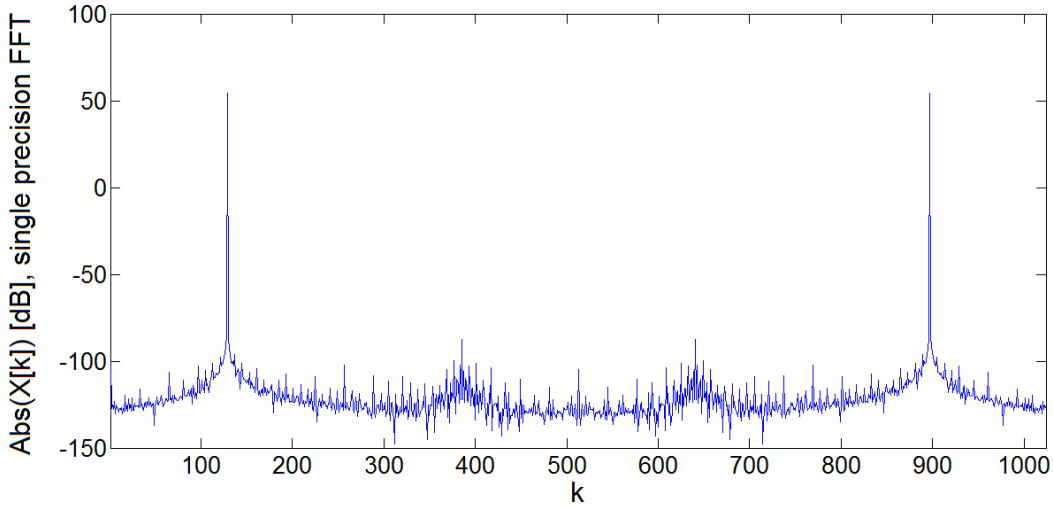

Figure 4. Single precision FFT output.

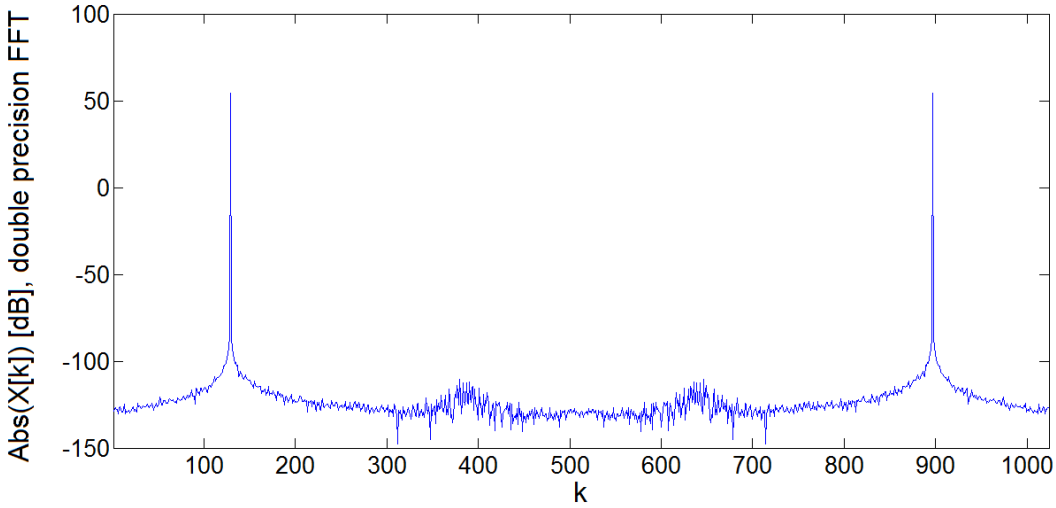

Figure 5. Double precision FFT output.

E. Comparing results with result assuming the noise model of quantization

The previous results show that in the case of almost coherent clean sine wave input the noise model assumption of roundoff is misleading. The elementary roundoff errors accumulate at some of the outputs so surprisingly high error peaks appear. Many simulations were run to measure the highest error peak. Then we compared these results with results assuming the noise model of quantization. Thanks to a very useful feature of Roundoff toolbox in Matlab every rounding operation (for example after a multiplication or addition) can be replaced with adding independent white noise to the exact value. This way our results can be compared with results assuming the noise model. A floating point number's variance is about $\mathrm{var}\{v\} \approx 0.180 \cdot 2^{-p} \cdot \mathrm{var}\{x\}$, with p the precision and x the signal [1]. The absolute value of the Fast Fourier Transform's roundoff error is Rayleigh distributed. Let us find the value X for which the probability is 99.9% that every peak of the roundoff error is smaller than X. In the case of N=1024 point FFT this means that only one error peak is expected to be larger than the X value. This X value can be seen in Fig. 6 as the horizontal line.

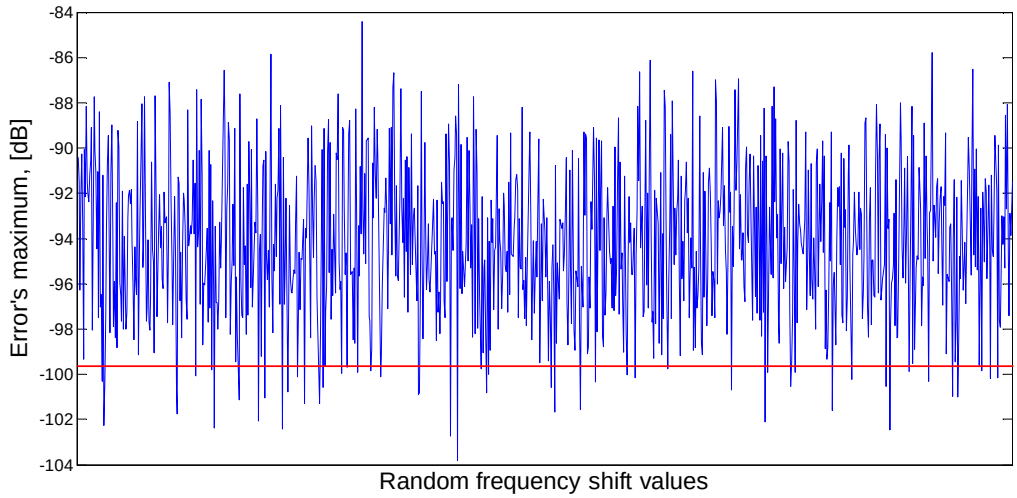

Figure 6. Maximum value of error peaks compared with the estimated value using the noise model of roundoff

In a very unfortunate situation 15 dB loss in the spurious-free dynamic range can be experienced when compared to the noise model. It is important to notice that the reason for the high peaks is "embedded" into the FFT algorithm; it is not the explicit cast operators in the code used to avoid precision growth which generate high error peaks. Simulations were also run without casting and the found reduction in the error's maximum was about 80% of the original value.

A last question is whether this case happens very rarely or not? Attempted but not perfect setting of harmonic sampling to good SNR signals is just the case we are simulating here. Thus, this is a practical case.

IV. Conclusions

In the paper we summarized and analyzed six different methods to examine the roundoff error of the FFT with single precision number representation. We managed to obtain exactly the same results for all 6 methods. The advantage of Matlab based methods is that complex numbers are handled. However, for us it was also a disadvantage because it was harder to examine what happens during the calculations. In C/C++ although we have to implement the handling of complex values, thanks to the compiler's optimizations we don't know either what really happens. Fortunately with forced castings this problem can be managed relatively easily, however we don't think that this coding style is popular in concrete applications. Anyway, it is a reassuring fact that the same errors can be produced with six different methods. Another conclusion is that for noiseless sine wave input the roundoff error can behave irregularly, and high peaks may appear far away from the sine's frequency. This means that the output of the FFT should be used with some extra care.

This work has been supported by the Hungarian Scientific Research Fund (OTKA), grant number TS-73496.

V. VI. References

[1] B. Widrow and I. Kollár (2008), "Quantization Noise: Roundoff Error in Digital Computation, Signal Processing, Control, and Communications," Cambridge University Press, Cambridge, UK. Home page: http://www.mit.bme.hu/books/quantization/

[2] Weinstein, C. J., "Roundoff Noise in Floating Point Fast Fourier Transform Computation," IEEE Transactions on Audio and Electroacoustics, Vol. 17, No. 3 ,pp. 209-15, 1969

[3] Pálfi Vilmos, Kollár István, Roundoff Errors in Fixed-Point FFT. In: WISP'2009, IEEE International symposium on Intelligent Signal Processing. Budapest, Hungary, 2009.08.26-2009.08.28. (IEEE) pp. 87-91. Paper W1B5.
http://mycite.omikk.bme.hu/doc/74336.pdf