# PERFORMANCE OF FIXED-POINT FFT'S: ROUNDING AND SCALING CONSIDERATIONS

P. Kabal[1,2] and B. Sayar[2]

[1]Electrical Engineering
McGill University
Montreal, Quebec H3A 2A7

[2]INRS-Télécommunications
Université du Québec
Verdun, Quebec H3E 1H6

## Abstract

The calculation of the discrete Fourier transform using a fast Fourier transform (FFT) algorithm with fixed-point arithmetic is considered. The input data is scaled to prevent overflow and to maintain accuracy. The implementation uses 16-bit fixed-point representation for the data and provides for double precision accumulation of sums and products. Algorithm variants as well as different rounding options are compared. Execution times for implementations based on a single chip signal processor are given. These show that a considerable increase in accuracy can be obtained with only a small penalty in execution time, by applying an alternating form of rounding rather than truncation.

## 1. Introduction

The need for the computation of the discrete Fourier transform (DFT) arises in a variety of applications in speech, radar and sonar signal processing. A DFT can be implemented either with special-purpose hardware, or as a program on a digital signal processing (DSP) chip.

A common feature both in DSP chips and standalone multipliers is a double precision accumulator. The focus will be on 16-bit representation of data with judicious use of 32-bit accumulation of products and sums. This paper considers more general rounding schemes and variants of the FFT algorithms that do not conveniently fit into the analysis framework used in [1].

## 2. Calculation with Fixed-Point Arithmetic

An example of a single chip implementation of a signal processor with features as described above is the Texas Instruments TMS320 DSP chip. Since this chip has 16-bit wide input/output paths, values are generally represented as 16-bit numbers. At the expense of multiple write or read cycles, 32-bit values can also be stored or retrieved from memory.

Data values will be represented as two's complement fractions. The DSP multiplier takes two 16-bit quantities (each represented by a sign bit with 15 data bits) to produce a 32-bit result, consisting of a sign extended 31-bit product (sign bit with 30 data bits). Because of the extra bit of head room in the product register, two products can be summed without overflow. This will be of benefit in complex multiplication since the components of a product of two complex numbers are formed as the sum of products of real quantities. The arithmetic unit also provides for shifting 16-bit data values before adding to the accumulator and for the extraction of specific 16-bit fields from the accumulator.

## 3. Scaling for the Discrete Fourier Transform

Consider a complex sequence $\{x_n\}$ of length $N$. The discrete Fourier transform of $\{x_n\}$ is the complex sequence $\{X_k\}$ of length $N$, defined as

$$X_k = \sum_{n=0}^{N-1} x_n W_N^{-nk} , \qquad k = 0, ..., N-1 , \tag{1}$$

where $W_N = \exp(j2\pi/N)$. It will be assumed that the calculation of the DFT, whether in direct form or using an FFT algorithm, will include a scaling by $1/N$. With this assumption, $|x_n| < 1$, $n = 0, \ldots, N-1$, will guarantee a representable output value. Dynamic verification of this bound is not simple since calculation of the magnitude (squared) requires the calculation of the sum of products. A test on the magnitudes of the real and imaginary components of the input sequence is easier to implement.

Let the real and imaginary components of the input sequence satisfy

$$|\mathrm{Re}[x_n]| < a \qquad \text{and} \qquad |\mathrm{Im}[x_n]| < a . \tag{2}$$

These bounds result in $|X_k| < \sqrt{2}Na$, $k = 0, 1, \ldots, N-1$. With scaling by $1/N$, and the requirement that the output magnitude be less than one, $a = 1/\sqrt{2}$. However, this is not the tightest bound on the real and imaginary components of the input data.

It can be shown that the magnitudes of the real and imaginary parts of the output of the complex DFT are bounded by $NaS_N$. For $N$ a multiple of 4,

$$S_N = \frac{4}{N} \frac{\cos(\pi/N)}{\sin(\pi/N)} . \tag{3}$$

For $N$ even but not a multiple of 4, $S_N = S_{2N}$ and for $N$ odd, $S_N = S_{4N}$. Also $1 \le S_N < 4/\pi$. Then the output components are representable in fractional notation if the real and imaginary parts of the input data satisfy

$$|\mathrm{Re}[x_n]| < \frac{\pi}{4} \qquad \text{and} \qquad |\mathrm{Im}[x_n]| < \frac{\pi}{4} . \tag{4}$$

Note that for nearly equal real and imaginary components, these bounds can result in values of $x_n$ and $X_k$ which are larger than unity in magnitude.

### 3.1 FFT Algorithms

Attention will be restricted to basic radix-2 FFT algorithms [2]. The FFT butterflies take a pair of complex data values and process them to produce a new pair of complex data values which can occupy the same storage locations as the input data.

A form of the basic decimation-in-time (DIT) butterfly computation is shown in Fig. 1. The magnitude of the complex values grows by at most a factor of two across a butterfly,

$$\max(|G_k|, |G_l|) \le \max(|G'_k|, |G'_l|) \le 2\max(|G_k|, |G_l|) . \tag{5}$$

At each butterfly, shifting values by one bit to accomplish a scaling by $1/2$ is appropriate. This gives the overall fixed scaling by $1/N$.

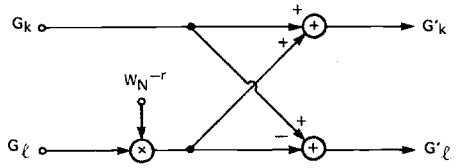


Fig. 1 Decimation-in-time butterfly

An important property of the DIT form of the FFT is that the intermediate results are DFT's of subsequences of the input data. As a result, if the magnitudes of the real and imaginary parts of the input sequence are bounded by $\pi/4$, all of the intermediate results will have real and imaginary parts which are less than one in magnitude.

With a double precision accumulator, there are several options for the implementation of the DIT butterfly. In the lower branch, the result of the complex multiplication can be kept in double precision in anticipation



ICASSP 86, TOKYO CH2243-4/86/0000-0221 $1.00 © 1986 IEEE

of the subsequent addition. Because of the extra bit of head room in the product register, no overflow can occur if the input data is scaled so as to guarantee representable output values. Scaling by the factor of $1/2$ is accomplished by shifting one bit position as results are stored from the accumulator. This last step is the point at which rounding can be applied.

An alternative strategy is to convert the result of the multiplication to a single precision quantity before addition. This reduces the number of interchanges between the accumulator and temporary storage. This version will be referred to as the single precision DIT algorithm as contrasted with the double precision DIT algorithm described in the previous paragraph. The multiplication by $W_N^{-r}$ in the butterfly implies that if the magnitude of the input complex value is greater than one, it is possible for the real and imaginary parts of the product to exceed one. However scaling at this point can be avoided since intermediate overflows can be ignored if the butterfly output is known to be representable.

The form of the butterfly for the decimation-in-frequency algorithm is shown in Fig. 2. Note that for the DIF algorithm, multiplication by the complex exponential term is performed at the output, rather than input stage of the butterfly. The magnitude of the complex values grows by at most a factor of two across a butterfly. Shifting values by one bit is used to scale by $1/2$ at each stage. The butterfly computation involves sums of single precision quantities followed by a multiplication by a complex exponential. The result of the sum must be converted to a single precision quantity for use as input to the multiplier. The resulting product in the lower branch of the butterfly is converted to single precision for storage. For the DIF algorithm, the intermediate results at the outputs of the butterflies are *not* DFT's of the original input data. The scaling requirement is that the magnitudes of the input real and imaginary parts each be less than $1/\sqrt{2}$, to guarantee that all butterfly outputs are less than one in magnitude.

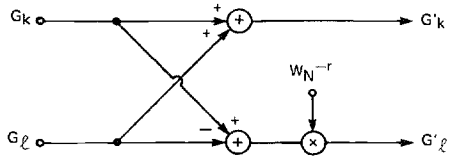


Fig. 2 Decimation-in-frequency butterfly

## 4. Rounding Options

At one or more places in the computation of the butterflies, a double precision value must be converted to a single precision value. Truncation simply discards the low-order part of the double precision value. On the other hand, rounding attempts to replace the double precision value with the nearest single precision value. However, an ambiguity arises whenever the value lies exactly halfway between two single precision values. This gives rise to a number of different methods of rounding, which differ only in the way they approximate the mid-way values.

1) *Up/Down Rounding.* The mid-way points are shifted up / down to the next allowable value (up-rounding / down-rounding).
2) *Magnitude Up/Down Rounding.* The mid-way points are moved away from / toward zero (magnitude up-rounding / magnitude down-rounding).
3) *Value-alternate Rounding.* Adjacent mid-way points are alternately moved up and down.
4) *Random Rounding.* The direction of rounding is pseudo-random.
7) *Stage-alternate Rounding.* In this new scheme, the direction of rounding alternates at each stage of the FFT.

Up or down rounding are simple to implement. Intuitively these methods tend to bias the output values by shifting the mean. Magnitude rounding is more complex to implement since it requires a sign test. Intuitively, magnitude rounding tends to alter the overall gain, but not the mean.

The last three rounding schemes could be grouped together under the rubric of unbiased techniques. Value-alternate and random rounding attempt to avoid both the change in gain and the shift in mean, but succeed only if the data exercises an equal number of up- and down-rounding steps. Stage-alternate rounding has the same goals, but attempts to cancel biases in one stage of the FFT by biases in the other stages.

Differences in the rounding techniques will manifest themselves only if the probability of hitting the mid-way points is significant. This occurs when single precision values are added. When the result (which has only one extra bit of precision) is scaled by $1/2$, the mid-way point can be expected to occur half of the time. It is for this case that the unbiased rounding schemes are useful.

## 5. Simulation Results

In the simulations, the input data were uniformly distributed, pseudo-random fixed-point fractional values between $-1/\sqrt{2}$ and $+1/\sqrt{2}$. This guarantees no overflow of the intermediate results of the FFT computations (butterfly outputs).

The root-mean square (rms) value of the random test input is approximately $1/\sqrt{6}$ or $-7.78$ dB relative to full scale. Note that this type of input sequence tends to make SNR figures look comparatively good. The input occupies nearly the full dynamic range and yet has a relatively large rms value. For more typical input signals the ratio of peak value to rms value is significantly larger.

A reference fixed-point result is also computed. If a fixed-point DFT is calculated with an arbitrarily large precision at the intermediate stages, errors will occur only when the final result is converted to single precision accuracy. The SNR for these fixed-point results is referred to in the plots as the ideal fixed-point DFT.

### 5.1 Assessing Bias

In attempting to assess and explain the behaviour of truncation and the various forms of rounding, a modified SNR definition may be used. Conventional SNR measures error as $E_k = \hat{X}_k - X_k$, where $\hat{X}_k$ is the output of the fixed-point DFT. In the experimental results, $\{X_k\}$ is the pseudo-random input sequence. This error expression penalizes changes in gain as well as shifts in the mean. Consider a modified error term, $E_k = a(\hat{X}_k - b) - X_k$. This expression includes the constants $a$ and $b$ which compensate for gain and mean respectively. Four cases can be identified. i) *Conventional SNR:* $a = 1$, $b = 0$; ii) *Gain Compensated SNR:* $a$ chosen to maximize the SNR, $b = 0$; iii) *Mean Compensated SNR:* $a = 1$, $b$ chosen to maximize the SNR; and iv) *Gain and Mean Compensated SNR:* both $a$ and $b$ chosen to maximize the SNR. The mean offset $b$ is restricted to have equal real and imaginary components. The main interest will be in looking at the gain and mean needed to get the best match between the true DFT and the fixed-point DFT. The increased SNR that results in choosing the gain and mean optimally will indicate the degree of match of the "shape" of the DFT.

### 5.2 Truncation Results

The resulting SNR's when truncation is used are shown in Fig. 3 for both one-way and two-way computations (DFT followed by inverse DFT). Using a model which assumes uncorrelated additive error components, it can be shown that the SNR for a two-way computation ($\mathrm{SNR}_2$) can be related to the SNR for a one-way computation ($\mathrm{SNR}_1$),

$$\mathrm{SNR}_2 = \frac{\mathrm{SNR}_1}{N+1} . \qquad (6)$$

The experimental results obey this relationship closely. Hence, for the subsequent results, reference is made only to the one-way SNR's.

A perhaps initially surprising result is that the single precision DIT FFT algorithm outperforms the double precision DIT FFT algorithm. Further investigation into the details of the butterfly computation reveal that the biases due to truncation of the product and the subsequent truncation of the differences tend to cancel in the single precision version. Theoretically, the mean offset at the output of each single precision butterfly is half that of the double precision version. The mean offset computed using the modified SNR definition verifies this analysis. The single precision DIT algorithm has a mean offset of approximately $-Nv_{\mathrm{lsb}}/2$, while the double precision DIT algorithm has a mean offset of approximately $-Nv_{\mathrm{lsb}}$, where $v_{\mathrm{lsb}}$ is the weight of the least significant bit of a single precision value ($v_{\mathrm{lsb}} = 1/32768$). The increase in SNR with mean compensation is about 1.5 dB for the single precision version and about 4 dB for the double precision version.

The mean offset for the DIF algorithm is about the same as the double precision DIT algorithm. The conventional SNR for the DIF algorithm is

also about the same as that of the double precision DIT algorithm. With mean compensation, the SNR increases about 4.5 dB. This means that both forms of the DIT algorithm and the DIF algorithm have about the same mean compensated SNR. Gain compensation does not help either DIT algorithm but does increase the SNR for the DIF algorithm by a further 1.5–2 dB.

### 5.3 Conventional Rounding

First consider the DIT FFT algorithm with double precision intermediate values. Rounding occurs as the last step in a butterfly. The DIT FFT improves 8–10 dB with rounding, with the larger improvements occurring with the longer transform lengths. The form of rounding has almost no effect on the SNR. The optimal gain is essentially unity and the mean offset is small. This agrees with the notion that the mid-way values which are rounded differently for the different rounding options occur with very small probability.

For the DIT FFT with single precision intermediate values and the DIF FFT, rounding can be applied at two different places. The first is in the rounding of the double precision products to single precision. Here, the exact form of rounding is unimportant. The second place is the point at which the sum of two single precision values is stored in single precision. For this operation, the conventional rounding schemes (up or down rounding and magnitude rounding) perform similarly in terms of SNR ($\sim$ 1 dB spread). However, the biases for the different rounding schemes are not the same. As anticipated earlier, the magnitude rounding schemes tend to result in an optimal gain different from unity but a mean offset nearly equal to zero. The gain factor deviates from unity by about 0.08% for $N = 256$. The resulting gain compensated SNR is about 4 dB larger than the conventional SNR. For up- and down-rounding, the gain is essentially unity, but the magnitude of the mean offset is about $N v_{\text{lsb}}/2$. The resulting mean compensated SNR is about 4 dB larger than conventional SNR. Based on the different behaviour of rounding for sums and for products, a selective rounding scheme in which rounding (of any form) is applied only to the product and truncation used for the sum performs as well as any of the conventional rounding schemes. Compared to truncation, rounding applied at the product gives a 2 dB increase in SNR for the single precision DIT algorithm and a 5 dB increase for the DIF algorithm. Table 1 shows SNR values for the FFT algorithms for $N = 128$.

| $N = 128$ | $\times$ truncation $+$ truncation | $\times$ up-rounding $+$ truncation | $\times$ up-rounding $+$ up-rounding | $\times$ stage-alternate $+$ stage-alternate |
|---|---|---|---|---|
| dp DIT | 59.3 dB | 68.6 dB | 68.6 dB | 68.6 dB |
| sp DIT | 62.0 dB | 64.3 dB | 64.1 dB | 68.2 dB |
| DIF | 59.2 dB | 64.5 dB | 64.4 dB | 68.6 dB |

Table 1 SNR for the indicated forms of rounding at the products ($\times$) and sums ($+$)

In the direct computation of the DFT, rounding occurs as the last step in the process. For single precision results, the form of rounding does not affect the SNR. The rounded results are essentially the same as that for the ideal integer DFT and about 6 dB better than truncation.

### 5.4 Unbiased Rounding

For the direct calculation of the DFT and the double precision DIT FFT algorithm, unbiased rounding performs the same as conventional rounding. However for the single precision DIT FFT and the DIF FFT, unbiased rounding offers a benefit at the sum step. The differences between the unbiased schemes are not large, but stage-alternate rounding stands out as being slightly better on the average ($\sim$ 1 dB) than value-alternate rounding, which is about the same as random rounding. The stage-alternate rounding scheme is about 4 dB better than conventional up-rounding (e.g. see Table 1).

The unbiased schemes give a useful increase in SNR for the DIF FFT and the single precision DIT FFT. The ease of implementation is an important consideration in the use of such a scheme. Of the unbiased schemes, stage-alternate rounding stands out as being the easiest to implement. The rounding operation consists of adding an appropriate offset before truncating. This offset is changed only outside the main computation loop in the FFT and hence is altered only $\log_2 N$ times. The best unbiased rounding scheme also comes with very little increase in computation or program size compared to conventional up-rounding.

As anticipated, these rounding schemes tend to result in optimal gain factors close to unity. The magnitude of the mean offset for stage-alternate rounding is about 1/3 of that for conventional up-rounding. The mean offsets for the other two unbiased schemes are essentially zero. In stage-alternate rounding, the main contribution to the mean offset is due to the last stage in which all output values are biased in the same direction.

The improvements in SNR due to the use of rounding are summarized in Fig. 4 (note the expanded vertical scale). The rounding results give the conventional SNR for up-rounding applied to the double precision DIT and the direct DFT algorithms, and stage-alternate rounding applied to the single precision DIT and the DIF algorithms.

The high accuracy of stage-alternate rounding applied to the FFT algorithms with single precision intermediate values has important ramifications in both hardware and software requirements.

## 6. Speed of Execution on the TMS32010

In this section, the trade-offs between speed and accuracy are examined for a specific processor, the TMS32010. Efficient programs written in TMS32010 assembly language [3] have been implemented for each of the computation procedures previously described.

### 6.1 FFT Algorithms

An important feature of the FFT algorithms considered here is that computations can be performed in-place — the output values can overlay the input values. This requires bit-reverse reordering at the input (DIT algorithm) or at the output (DIF algorithm). With in-place computations, a transform length of up to 64 complex points ($N = 64$) can be handled by the TMS32010 using on-chip random access memory (RAM) to store the data.

The bit reversal portion of the algorithms is implemented using straight-line code for maximum speed. The execution times for bit reversal are 20, 39 and 90 $\mu$s for $N = 16$, 32 and 64 respectively.

The execution times (excluding bit reversal time) for the double precision DIT are shown in Table 2. In this table rounding refers to conventional up-rounding. In Table 3, rounding combinations of interest for the DIF and single precision DIT algorithms are shown. The single precision DIT algorithm has the same execution time as the DIF algorithm. Selective rounding refers to the use of truncation at the sums and up-rounding (or down-rounding) at the products. The figures listed under rounding refer to up-rounding, down-rounding or stage-alternate rounding applied at both sums and products. Note that stage-alternate rounding can be implemented with essentially no speed penalty compared to conventional rounding. The figures for magnitude rounding are for magnitude up- or down-rounding applied at the sums and conventional up-rounding at the products. Note that rounding is relatively inexpensive in terms of execution time, yet increases the SNR by a considerable amount in all cases.

| | Execution time $\mu$s | |
|---|---|---|
| N | Truncation | Rounding |
| 16 | 429 | 453 |
| 32 | 1018 | 1082 |
| 64 | 2371 | 2523 |

Table 2 Execution time for double precision DIT

| | Execution time $\mu$s | | | |
|---|---|---|---|---|
| N | Truncation | Sel. round | Rounding | Mag. round |
| 16 | 378 | 391 | 404 | 442 |
| 32 | 890 | 922 | 954 | 1050 |
| 64 | 2067 | 2143 | 2219 | 2447 |

Table 3 Execution time for DIF and single precision DIT

Stage-alternate rounding applied either to the single precision DIT or the DIF algorithm gives a high SNR, about the same as that for the

double precision DIT algorithm with rounding. When comparing the speed figures, it can be seen that the schemes using stage-alternate rounding are faster ($\sim 12\%$) than the double precision DIT algorithm.

## 7. Summary and Conclusions

Several procedures for implementing the discrete Fourier transform of a set of samples on a 16-bit fixed-point processor have been examined. If speed is paramount, the DIF algorithm with truncation is a good choice. It is one of the two fastest schemes, yet its mean compensated SNR is high, indicating that it preserves the signal shape. Indeed, simply adding a constant value at the last stage of the computation can compensate for the mean offset.

If both accuracy and speed are important considerations, the single precision DIT algorithm with a new form of rounding, dubbed stage-alternate rounding, is a good candidate for implementation. It has a high SNR, the rounding involves only a small increase in computation time over truncation, and it offers an increased input dynamic range (input scaling to $\pi/4$ rather than $1/\sqrt{2}$ as in the DIF algorithm).

Stage-alternate rounding gives high accuracy without the need for full double precision accumulation. This translates to reduced execution time for a single chip DSP implementation or reduced complexity for hardware and custom chip implementations.

## References


1. T.-Thông and B. Liu, "Fixed-point fast Fourier transform error analysis", *IEEE Trans. Acoustics, Speech, Signal Processing*, vol. ASSP-24, pp. 563–573, Dec. 1976.
2. A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Prentice-Hall, 1975.
3. Texas Instruments, *TMS32010 User's Guide*, Texas Instruments, 1983.


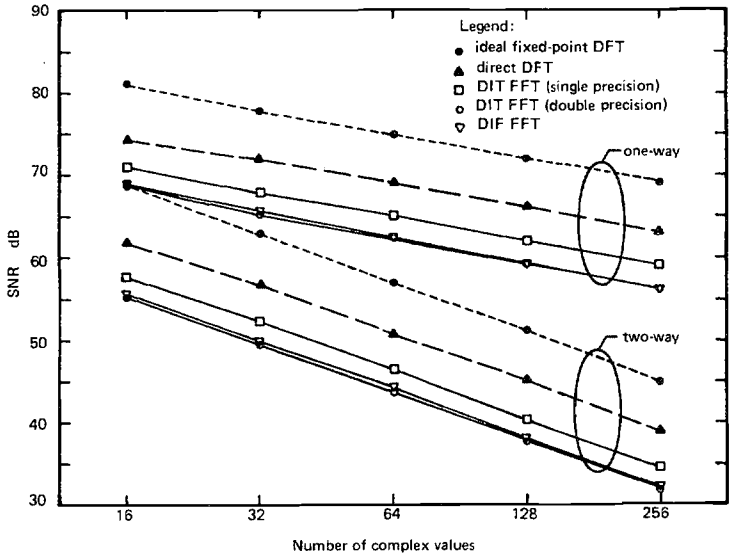


Fig. 3 SNR for truncation

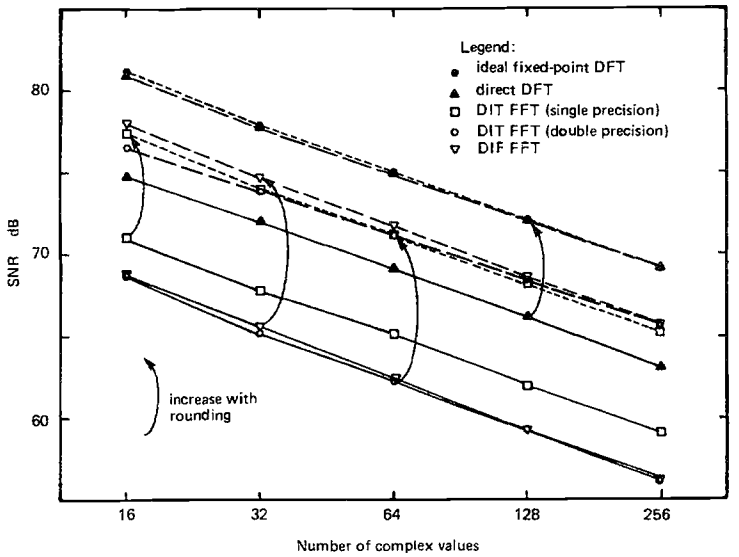


Fig. 4 SNR for truncation and rounding

6. 3. 4